華中科技大學

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 1501

学　　号： U201514888

姓　　名： 陈艺欣

指导教师： 甘早斌

报告日期： 2018.6.26

计算机科学与技术学院

目 录

1 实验一　低频读写器实验

1.1　实验目的

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）。

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。

1.2　实验内容及结果

1、完成低频读写器的标签读取试验；

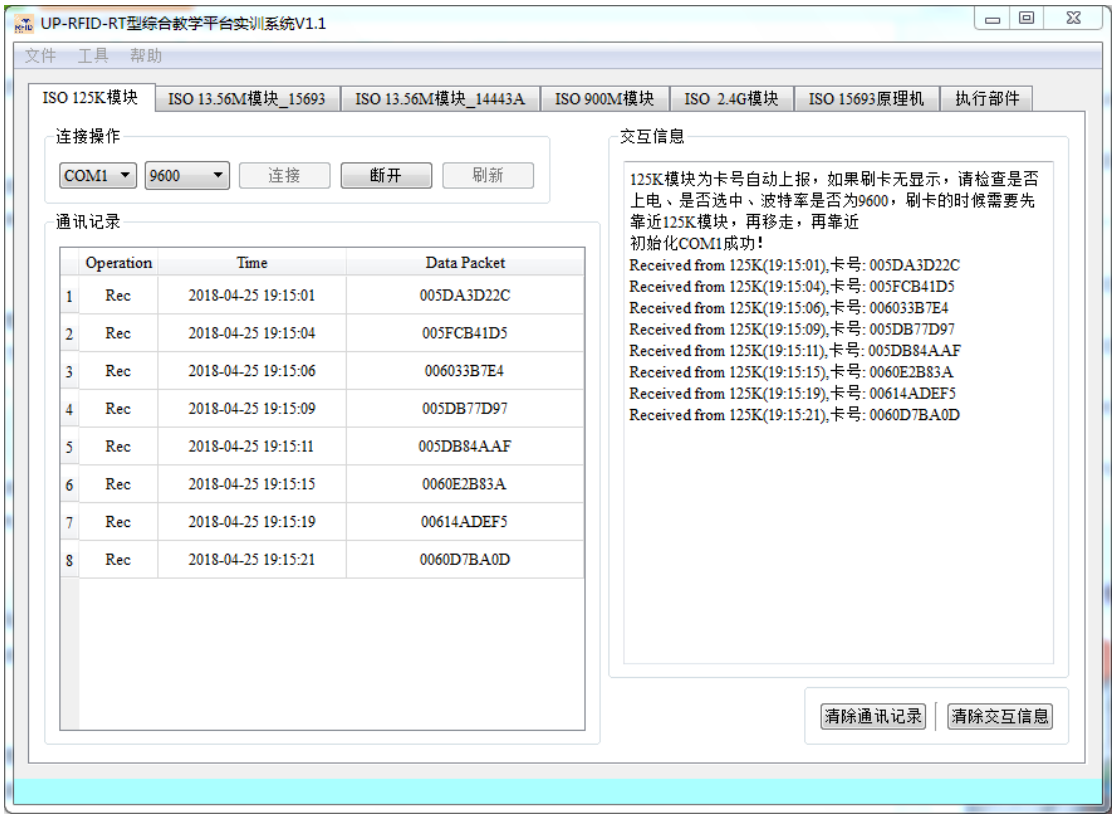

图 1 反复循环读取十张低频电子标签

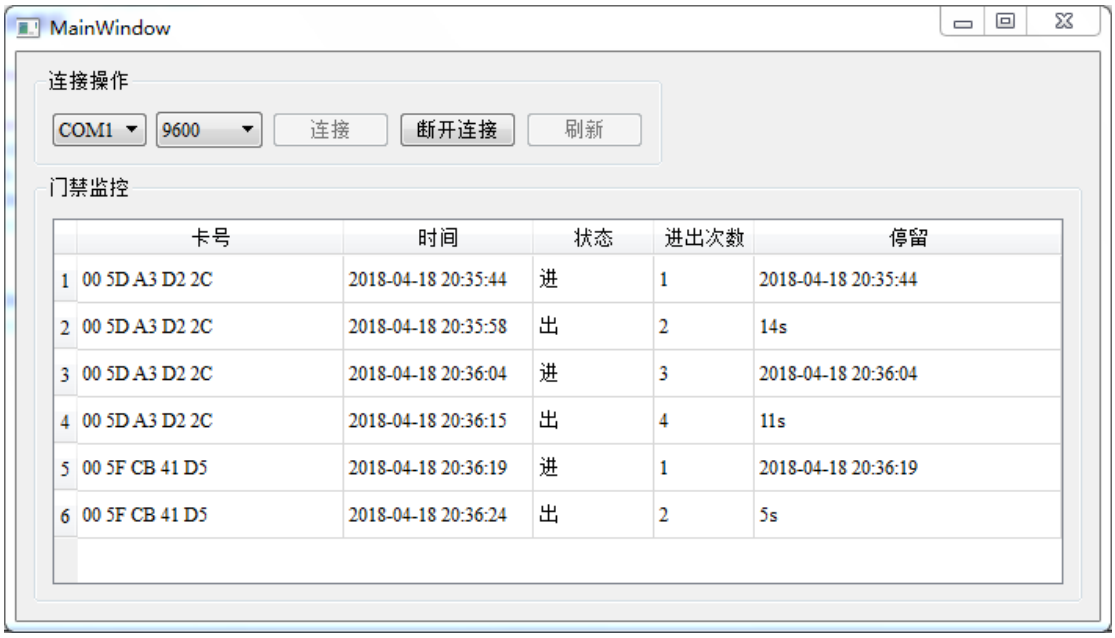

图 2 记录保存进出的历史记录、停留时间

1.3 实验体会与总结

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；

2、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？

```
/**
 * @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());
      }
   }
QByteArray data = serialPort->readAll();//读取串口发来的数据
//Qt有自带的封装好的函数，可以直接调用，但是读取到的数据QByteArray型，要想转化成QString，还要经过进一步调用CharStringtoHexString的函数，能使卡号显示出来。
```

总结：

通过这次实验，我大概了解了低频读写器 API 函数的调用方法。

1.4 核心源码说明

```
#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();
    }
}
```

2 实验二　高频读写器实验 ISO14443A

2.1　实验目的

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。

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。

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。

2.2　实验内容及结果

1、完成 ISO14443A 协议下标签寻卡、唤醒、休眠实验；

图 3 寻卡、唤醒、休眠

2、完成 ISO14443A 协议下标签内存读写实验；

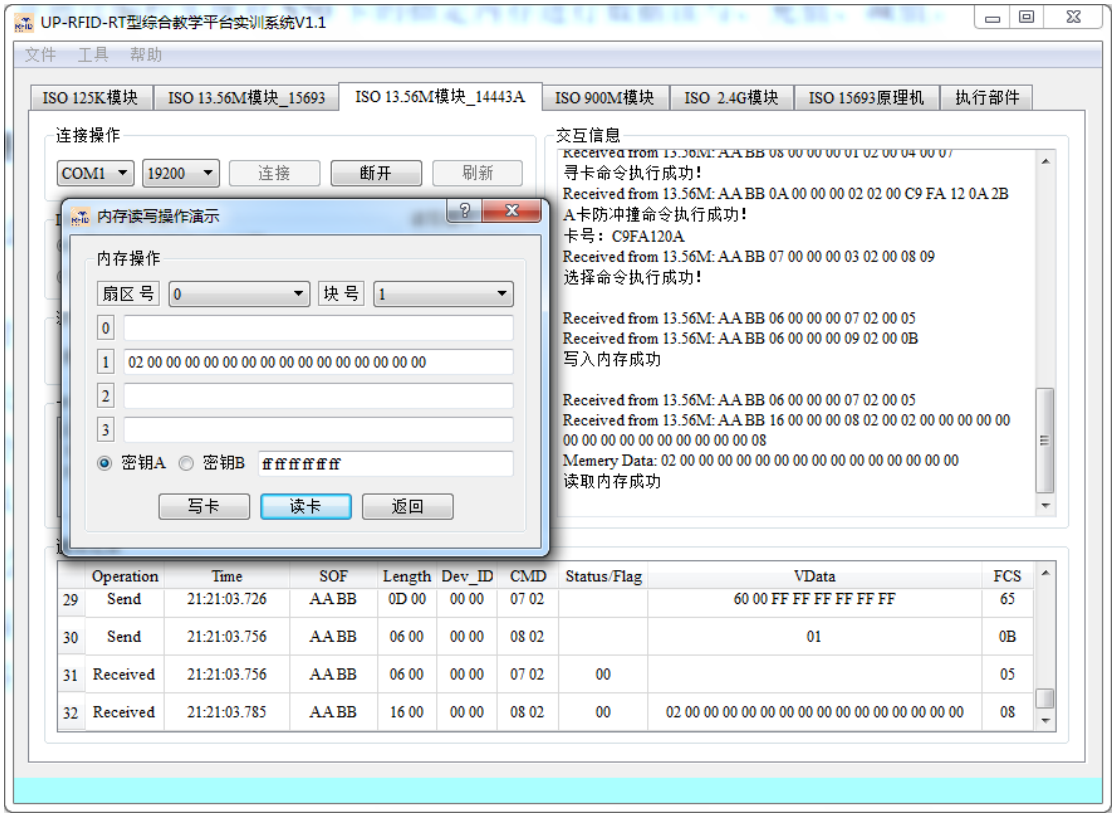

图 4 读卡实验

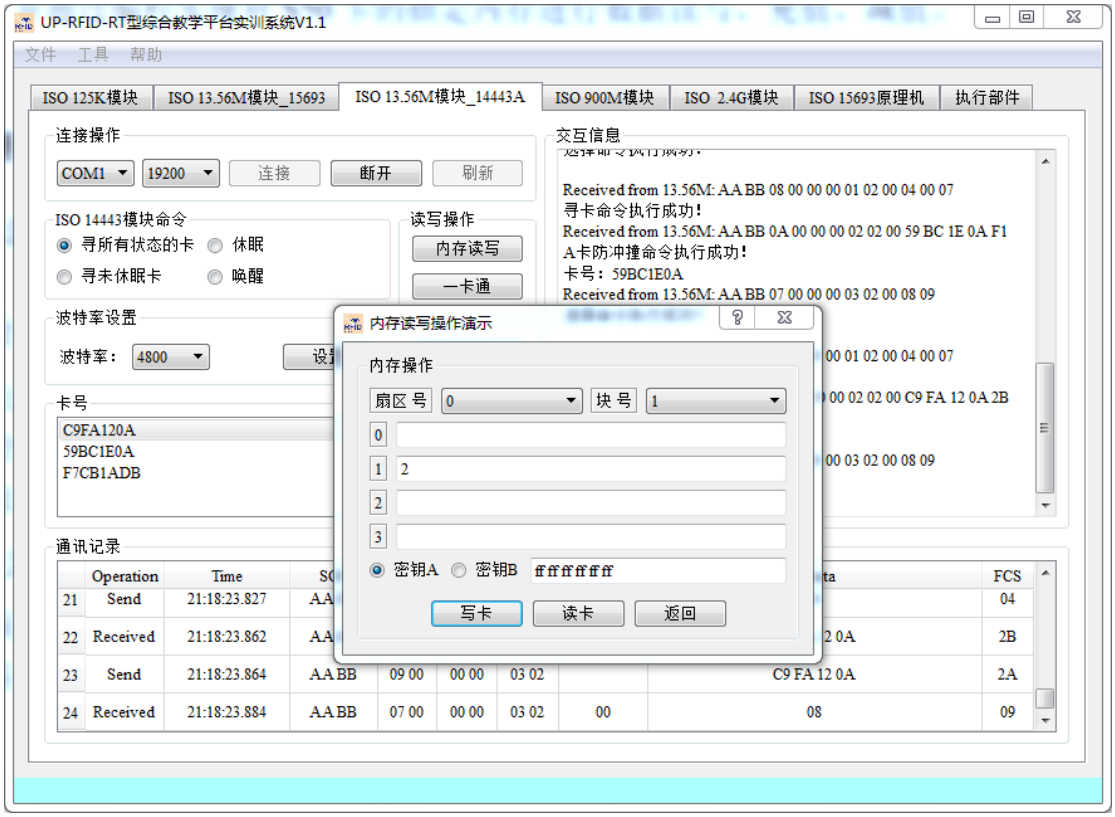

图 5 写卡实验

3、完成 ISO14443A 协议下标签一卡通实验；

图 6 一卡通充值

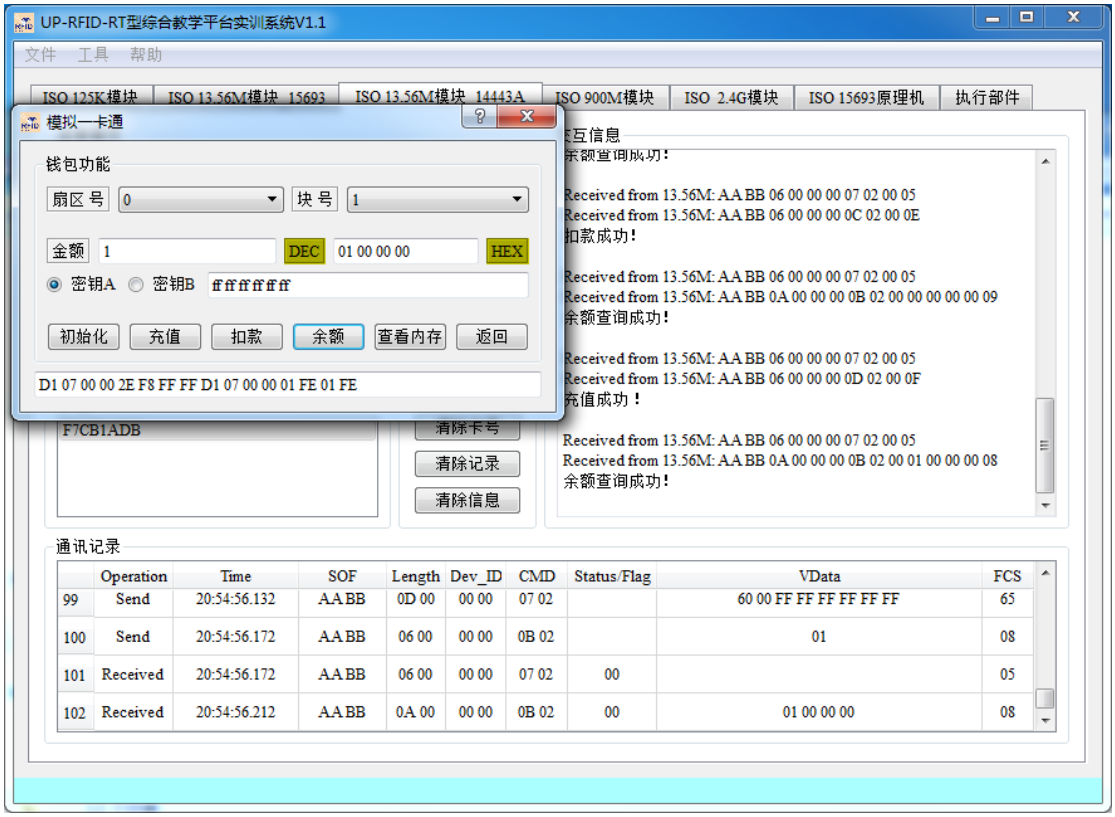

图 7 一卡通查询余额

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

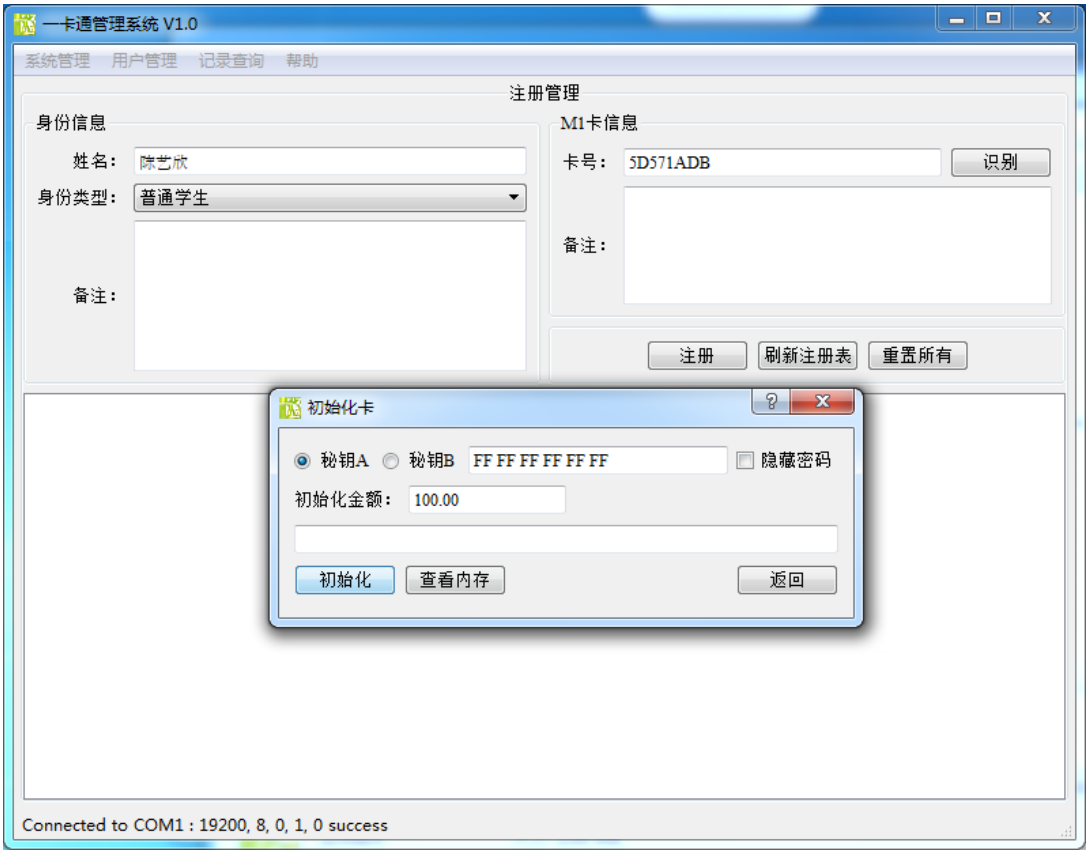
一卡通管理系统 V1.0
系统管理
用户管理
记录查询
帮助
注册管理
身份信息
姓名：
陈艺欣
身份类型：
普通学生
备注：
M1卡信息
卡号：
5D571ADB
识别
备注：
注册
刷新注册表
重置所有
初始化卡
秘钥A
秘钥B
FF FF FF FF FF FF
隐藏密码
初始化金额：
100.00
初始化
查看内存
返回
Connected to COM1 : 19200, 8, 0, 1, 0 success

图 8 写卡

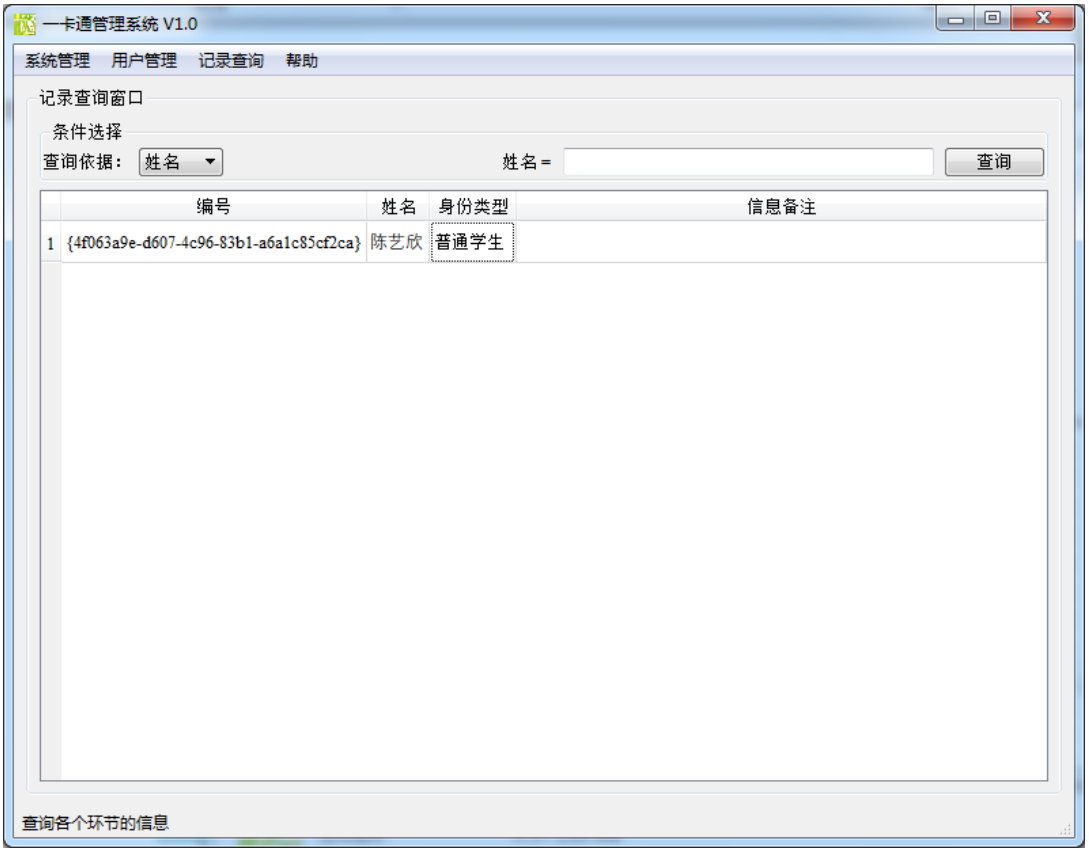
一卡通管理系统 V1.0
系统管理
用户管理
记录查询
帮助
记录查询窗口
条件选择
查询依据：
姓名
姓名=
查询
编号
姓名
身份类型
信息备注
1
{4f063a9e-d607-4c96-83b1-a6a1c85cf2ca}
陈艺欣
普通学生
查询各个环节的信息

图 9 读卡

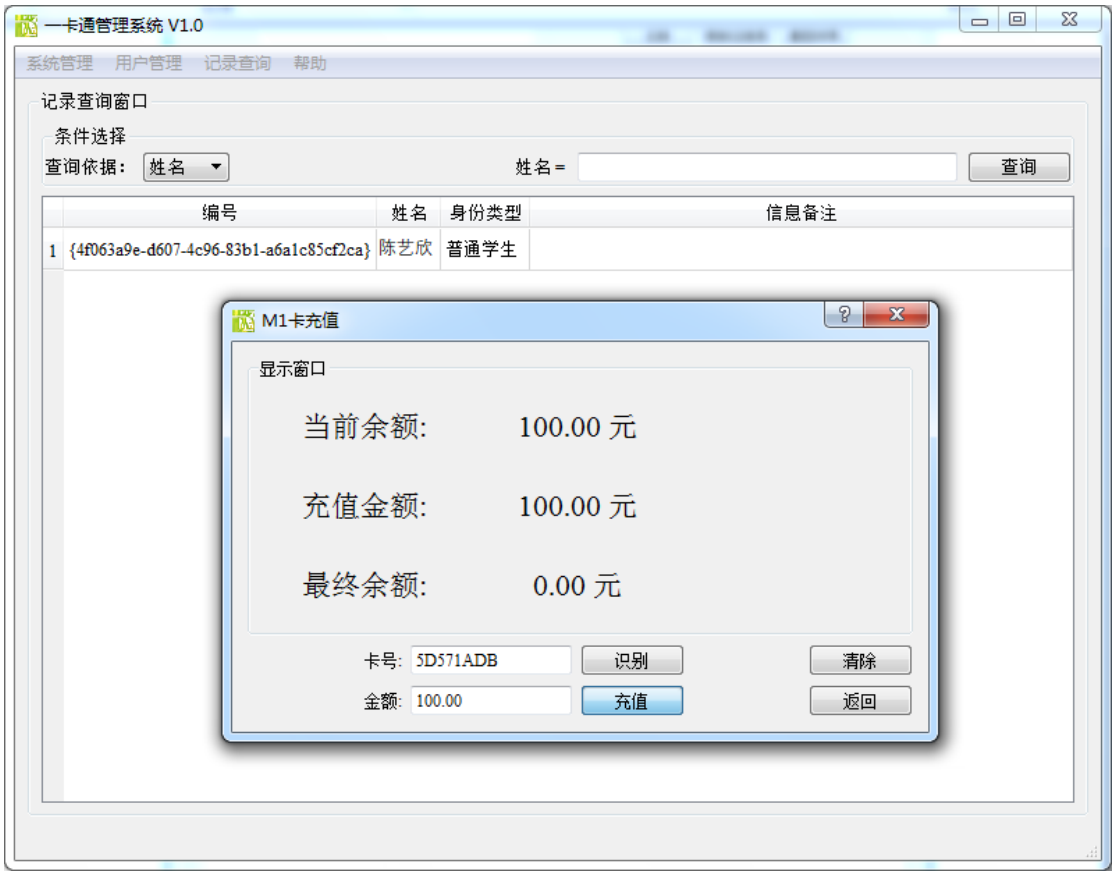
一卡通管理系统 V1.0
系统管理
用户管理
记录查询
帮助
记录查询窗口
条件选择
查询依据：
姓名
姓名=
查询
编号
姓名
身份类型
信息备注
1
{4f063a9e-d607-4c96-83b1-a6a1c85cf2ca}
陈艺欣
普通学生
M1卡充值
显示窗口
当前余额:
100.00 元
充值金额:
100.00 元
最终余额:
0.00 元
卡号:
5D571ADB
识别
清除
金额:
100.00
充值
返回

图 10 充值

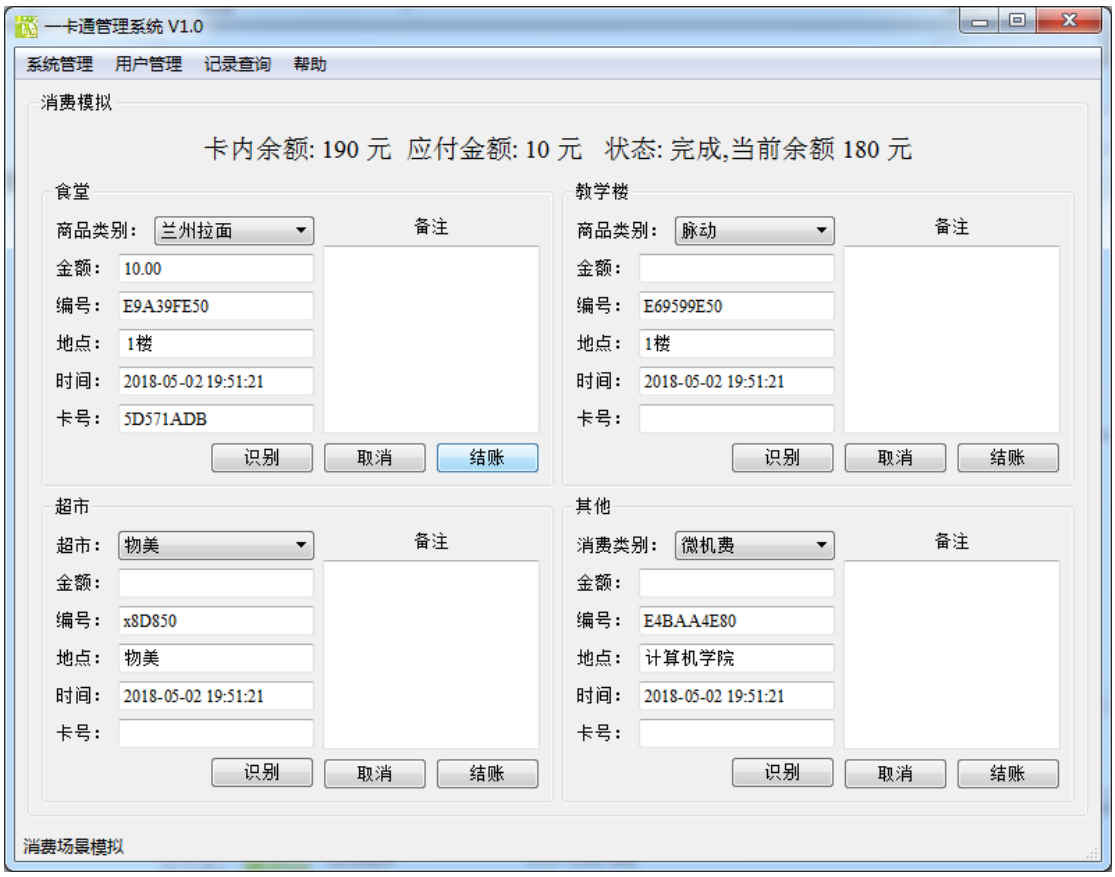

图 11 减值

2.3 实验体会与总结

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。

2、S50 卡第 1 扇区第 0 块是否可读写？为什么？

可以读写，因为只有第 0 扇区，第 0 块号为卡序列号，不能为只读的，其他的可以读写。

3、“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。

4、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。

5、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，就能将卡初始化为电子钱包。

总结：通过这次的实验，加深了我们对于标签读写与控制的理解，也对老师在课堂上讲的只是有了更加深入的理解，只有实际动手去操作了才能感受到 RFID 的种种知识以及理念。理论与实践相结合，使得我们对于这门课有了更好的了解，对知识的把握也更加深刻了。

2.4 核心源码说明

写卡 dialogccardconfig.cpp

```
#include "dialogcardconfig.h"
#include "ui_dialogcardconfig.h"
#include <QRegExp>
#include <QValidator>
#include <QDebug>
#include <QMessageBox>
/*****************************************
 *作者: jianghj@up-tech.com
 *日期: 2016-09-20
 *描述: 注册之后卡的配置对话框,主要用于初始化卡的存储格式,初始化金额
*****************************************/
DialogCardConfig::DialogCardConfig(QWidget *parent, SerialPortThread *serialPortThread) :
    QDialog(parent),
    ui(new Ui::DialogCardConfig)
{
    currentOps = -1 ;
    ui->setupUi(this);
    m1356dll = new M1356Dll();
    this->serialPortThread = serialPortThread;

connect(serialPortThread,SIGNAL(receivedMsg(QByteArray)),this,SLOT(onDecodeFrame(QByteArray)));//收到信息
    QRegExp rx("^[0-9a-fA-F]{17}");
    QValidator* validator = new QRegExpValidator(rx, this);
    ui->lineEdit_Pwd->setValidator(validator);
    ui->lineEdit_Pwd->setInputMask(tr("HH HH HH HH HH HH"));
    rx.setPattern("^[1-9]{1,3}\\.[0-9]{1,2}");
    validator = new QRegExpValidator(rx, this);
    ui->initValue->setValidator(validator);
    ui->initValue->installEventFilter(this);
    ui->radioButtonA->setChecked(true);
    ui->lineEdit_Pwd->setText(tr("FFFFFFFFFFFF"));
```

```
}

DialogCardConfig::~DialogCardConfig()
{
    this->disconnect(serialPortThread);
    delete ui;
}

/**
 * @brief DialogCardConfig::on_lineEdit_Pwd_cursorPositionChanged
 * @param arg1  原位置
 * @param arg2  新位置
 *  密码输入框光标位置发生改变时调用
 */
void DialogCardConfig::on_lineEdit_Pwd_cursorPositionChanged(int arg1, int arg2)
{
    if(arg1 > arg2)
        return;
    int len = ui->lineEdit_Pwd->text().length();
    int tem = 5 - (len - 5)/2;
    int cursorPositon = len - tem;
    if(arg2 > cursorPositon)
        ui->lineEdit_Pwd->setCursorPosition(cursorPositon);
}
/**
 * @brief DialogCardConfig::on_checkBox_clicked
 * @param checked  如果选中为true,否则为false
 *  显示密钥复选框点击事件
 */
void DialogCardConfig::on_checkBox_clicked(bool checked)
{
    if(checked)
        ui->lineEdit_Pwd->setEchoMode(QLineEdit::Password);
    else
        ui->lineEdit_Pwd->setEchoMode(QLineEdit::Normal);
}
/**
 * @brief DialogCardConfig::eventFilter
 * @param obj  产生事件的对象
 * @param event Qt事件
 * @return bool型值,表明此事件是否处理了
 *  事件过滤器
 */
bool DialogCardConfig::eventFilter(QObject *obj, QEvent *event)
{
    if(obj == ui->initValue)
    {
        if(event->type() == QEvent::FocusOut)
        {
            QString value = ui->initValue->text();
            if(value.length() == 0)
                ui->initValue->setText(tr("1.00"));
            else if(value.endsWith('.'))
                ui->initValue->setText(value + tr("00"));
            else if(!value.contains('.'))
                ui->initValue->setText(value + tr(".00"));
```

```
                else if(value.right(value.length() - value.indexOf('.')).length() == 2)
                    ui->initValue->setText(value + tr("0"));
            }
        }
        return QDialog::eventFilter(obj,event);
}

/**
 * @brief DialogCardConfig::onDecodeFrame
 * @param frame 14443的响应帧
 *  串口接收槽函数
 */
void DialogCardConfig::onDecodeFrame(QByteArray bytes)
{
        M1356_RspFrame_t frame = m1356dll->M1356_RspFrameConstructor(bytes);
        qDebug() <<"data: " << frame.vdata << frame.cmd << frame.sof;
        if(frame.cmd.remove(" ") == "0702" && frame.status == "00")//授权成功
        {
            switch (currentOps) {
            case 0: //init
            {
                uint16 frameLen;
                quint8 buffer[5];
                uint8 *p;
                QString str = ui->initValue->text();
                buffer[0] = 0x9;//绝对块号,这样的最好定义成宏,方便修改
                memset(buffer+1, 0, 5);
                float test = str.toFloat();
                memcpy(buffer + 1,&test,4);
                p = m1356dll->RC632_SendCmdReq(RC632_CMD_M1INITVAL,buffer,5);
                frameLen = BUILD_UINT16(p[0], p[1]);
                serialPortThread->writeData((char *)(p + 2 ),frameLen);
                currentOps = -1;
            }
                break;
            case 1: //mem
            {
                uint16 frameLen;
                quint8 buffer[1];
                uint8 *p;
                buffer[0] = 0x9;
                p = m1356dll->RC632_SendCmdReq(RC632_CMD_M1READ,buffer,1);
                frameLen = BUILD_UINT16(p[0], p[1]);
                serialPortThread->writeData((char *)(p + 2 ),frameLen);
                  currentOps = -1;
            }
                break;
            default:
                break;
            }
        }
        else if(frame.cmd.remove(" ") == "0802")
        {
            ui->lineEditMemData->setText(frame.vdata);
        }
}
```

```
/**
 * @brief DialogCardConfig::on_btn_Init_clicked
 *  初始化按钮点击事件
 */
void DialogCardConfig::on_btn_Init_clicked()
{
    if(!serialPortThread->serialPortIsOpen())
    {
        QMessageBox::warning(this,tr("温馨提示"),tr("请先连接读卡器后再试！
"),QMessageBox::Yes);
        return;
    }
    if(ui->lineEdit_Pwd->text().length() != 17)
    {
        QMessageBox::warning(this, "Error", tr("请在密钥区输入6个字节密钥！"));
        return;
    }
    this->authentication();
    currentOps = 0 ;
}
/**
 * @brief DialogCardConfig::on_btn_MemData_clicked
 *  查看内存数据
 */
void DialogCardConfig::on_btn_MemData_clicked()
{
    if(!serialPortThread->serialPortIsOpen())
    {
        QMessageBox::warning(this,tr("温馨提示"),tr("请先连接读卡器后再试！
"),QMessageBox::Yes);
        return;
    }
    if(ui->lineEdit_Pwd->text().length() != 17)
    {
        QMessageBox::warning(this, "Error", tr("请在密钥区输入6个字节密钥！"));
        return;
    }
    this->authentication();
    currentOps = 1;
}
/**
 * @brief DialogCardConfig::authentication
 *  授权
 */
void DialogCardConfig::authentication()
{
    uint16 frameLen;
    quint8 buffer[8];
    uint8 *p;
    if(ui->radioButtonA->isChecked())
        buffer[0] = 0x60;     // A密钥
    else
        buffer[0] = 0x61;   // B  密钥
    buffer[1] = 0x09;     //  绝对块号
    QString str = ui->lineEdit_Pwd->text().remove(" "); //六字节
    QSTRING_TO_HEX(str, (uint8*)(buffer+2),6);
```

```
        p = m1356dll->RC632_SendCmdReq(RC632_CMD_AUTHENTICATION,buffer,8);//获取卡密码
        frameLen = BUILD_UINT16(p[0], p[1]);
        serialPortThread->writeData((char *)(p + 2 ),frameLen);//写卡
}
/**
 * @brief DialogCardConfig::on_btn_Return_clicked
 *  返回按钮点击事件
 */
void DialogCardConfig::on_btn_Return_clicked()
{
        this->close();
}
```

读卡

```
#include "settingsdialog.h"
#include "ui_settingsdialog.h"

#include <QtSerialPort/QSerialPortInfo>
#include <QIntValidator>
#include <QLineEdit>

QT_USE_NAMESPACE
/**
 * @brief blankString
 *  默认填充,也就是说串口属性信息不存在时显示"N/A"
 */
static const char blankString[] = QT_TRANSLATE_NOOP("SettingsDialog", "N/A");
/*****************************************
 *作者: jianghj@up-tech.com
 *日期: 2016-09-20
 *描述:  串口连接对话框,通过此页面可以对串口进行详细的配置
*****************************************/
SettingsDialog::SettingsDialog(QWidget *parent) :
        QDialog(parent),
        ui(new Ui::SettingsDialog)
{
        ui->setupUi(this);
        intValidator = new QIntValidator(0, 4000000, this);
        ui->baudRateBox->setInsertPolicy(QComboBox::NoInsert);
        connect(ui->btn_Apply, SIGNAL(clicked()), this, SLOT(apply()));
        connect(ui->serialPortInfoListBox, SIGNAL(currentIndexChanged(int)), this,
SLOT(showPortInfo(int)));
        connect(ui->baudRateBox, SIGNAL(currentIndexChanged(int)), this,
SLOT(checkCustomBaudRatePolicy(int)));
        connect(ui->serialPortInfoListBox, SIGNAL(currentIndexChanged(int)), this,
SLOT(checkCustomDevicePathPolicy(int)));

        fillPortsParameters();
        fillPortsInfo();
        updateSettings();
}

SettingsDialog::~SettingsDialog()
{
        delete ui;
```

```
}
/**
 * @brief SettingsDialog::settings
 * @return    串口配置信息
 *  作为一个常量返回当前串口配置
 */
SettingsDialog::Settings SettingsDialog::settings() const
{
    return currentSettings;
}
/**
 * @brief SettingsDialog::showPortInfo
 * @param idx  选中的串口名的索引值
 *  显示串口相关的详细信息
 */
void SettingsDialog::showPortInfo(int idx)
{
    if (idx == -1)
        return;

    QStringList list = ui->serialPortInfoListBox->itemData(idx).toStringList();//获取信息列
表
    ui->descriptionLabel->setText(tr("描述: %1").arg(list.count() > 1 ? list.at(1):
tr(blankString)));
    ui->manufacturerLabel->setText(tr("制造商: %1").arg(list.count() > 2 ? list.at(2) :
tr(blankString)));
    ui->serialNumberLabel->setText(tr("序列号: %1").arg(list.count() > 3 ? list.at(3) :
tr(blankString)));
    ui->locationLabel->setText(tr("位置: %1").arg(list.count() > 4 ? list.at(4) :
tr(blankString)));
    ui->vidLabel->setText(tr("厂商标识: %1").arg(list.count() > 5 ? list.at(5) :
tr(blankString)));
    ui->pidLabel->setText(tr("产品ID: %1").arg(list.count() > 6 ? list.at(6) : tr(blankString)));
}
/**
 * @brief SettingsDialog::apply
 *  应用按钮点击事件
 */
void SettingsDialog::apply()
{
    updateSettings();
    hide();
    emit applySettings();
}
/**
 * @brief SettingsDialog::checkCustomBaudRatePolicy
 * @param idx  索引值
 *  监控是否选择了custom项(波特率)
 */
void SettingsDialog::checkCustomBaudRatePolicy(int idx)
{
    bool isCustomBaudRate = !ui->baudRateBox->itemData(idx).isValid();
    ui->baudRateBox->setEditable(isCustomBaudRate);
    if (isCustomBaudRate) {
        ui->baudRateBox->clearEditText();
```

```
            QLineEdit *edit = ui->baudRateBox->lineEdit();
            edit->setValidator(intValidator);
        }
}
/**
 * @brief SettingsDialog::checkCustomDevicePathPolicy
 * @param idx  索引值
 *  监控是否选择了custom项(串口名)
 */
void SettingsDialog::checkCustomDevicePathPolicy(int idx)
{
    bool isCustomPath = !ui->serialPortInfoListBox->itemData(idx).isValid();
    ui->serialPortInfoListBox->setEditable(isCustomPath);
    if (isCustomPath)
        ui->serialPortInfoListBox->clearEditText();
}
/**
 * @brief SettingsDialog::fillPortsParameters
 *  串口参数信息
 */
void SettingsDialog::fillPortsParameters()
{
    ui->baudRateBox->addItem(QStringLiteral("9600"), QSerialPort::Baud9600);
    ui->baudRateBox->addItem(QStringLiteral("19200"), QSerialPort::Baud19200);
    ui->baudRateBox->addItem(QStringLiteral("38400"), QSerialPort::Baud38400);
    ui->baudRateBox->addItem(QStringLiteral("115200"), QSerialPort::Baud115200);
    ui->baudRateBox->addItem(tr("Custom"));
    ui->baudRateBox->setCurrentIndex(1);

    ui->dataBitsBox->addItem(QStringLiteral("5"), QSerialPort::Data5);
    ui->dataBitsBox->addItem(QStringLiteral("6"), QSerialPort::Data6);
    ui->dataBitsBox->addItem(QStringLiteral("7"), QSerialPort::Data7);
    ui->dataBitsBox->addItem(QStringLiteral("8"), QSerialPort::Data8);
    ui->dataBitsBox->setCurrentIndex(3);

    ui->parityBox->addItem(tr("None"), QSerialPort::NoParity);
    ui->parityBox->addItem(tr("Even"), QSerialPort::EvenParity);
    ui->parityBox->addItem(tr("Odd"), QSerialPort::OddParity);
    ui->parityBox->addItem(tr("Mark"), QSerialPort::MarkParity);
    ui->parityBox->addItem(tr("Space"), QSerialPort::SpaceParity);

    ui->stopBitsBox->addItem(QStringLiteral("1"), QSerialPort::OneStop);
#ifdef Q_OS_WIN
    ui->stopBitsBox->addItem(tr("1.5"), QSerialPort::OneAndHalfStop);
#endif
    ui->stopBitsBox->addItem(QStringLiteral("2"), QSerialPort::TwoStop);

    ui->flowControlBox->addItem(tr("None"), QSerialPort::NoFlowControl);
    ui->flowControlBox->addItem(tr("RTS/CTS"), QSerialPort::HardwareControl);
    ui->flowControlBox->addItem(tr("XON/XOFF"), QSerialPort::SoftwareControl);
}
/**
 * @brief SettingsDialog::fillPortsInfo
 *  填充串口描述信息
 */
void SettingsDialog::fillPortsInfo()
{
```

```
    ui->serialPortInfoListBox->clear();
    QString description;
    QString manufacturer;
    QString serialNumber;
    foreach (const QSerialPortInfo &info, QSerialPortInfo::availablePorts()) {
        QStringList list;
        description = info.description();
        manufacturer = info.manufacturer();
        serialNumber = info.serialNumber();
        list << info.portName()
             << (!description.isEmpty() ? description : blankString)
             << (!manufacturer.isEmpty() ? manufacturer : blankString)
             << (!serialNumber.isEmpty() ? serialNumber : blankString)
             << info.systemLocation()
             << (info.vendorIdentifier() ? QString::number(info.vendorIdentifier(), 16) :
blankString)
             << (info.productIdentifier() ? QString::number(info.productIdentifier(), 16) :
blankString);

        ui->serialPortInfoListBox->addItem(list.first(), list);
    }

    ui->serialPortInfoListBox->addItem(tr("Custom"));
}
/**
 * @brief SettingsDialog::updateSettings
 *  更新串口配置
 */
void SettingsDialog::updateSettings()
{
    currentSettings.name = ui->serialPortInfoListBox->currentText();

    if (ui->baudRateBox->currentIndex() == 4) {
        currentSettings.baudRate = ui->baudRateBox->currentText().toInt();
    } else {
        currentSettings.baudRate = static_cast<QSerialPort::BaudRate>(

ui->baudRateBox->itemData(ui->baudRateBox->currentIndex()).toInt());
    }
    currentSettings.stringBaudRate = QString::number(currentSettings.baudRate);

    currentSettings.dataBits = static_cast<QSerialPort::DataBits>(
                ui->dataBitsBox->itemData(ui->dataBitsBox->currentIndex()).toInt());
    currentSettings.stringDataBits = ui->dataBitsBox->currentText();

    currentSettings.parity = static_cast<QSerialPort::Parity>(
                ui->parityBox->itemData(ui->parityBox->currentIndex()).toInt());
    currentSettings.stringParity = ui->parityBox->currentText();

    currentSettings.stopBits = static_cast<QSerialPort::StopBits>(
                ui->stopBitsBox->itemData(ui->stopBitsBox->currentIndex()).toInt());
    currentSettings.stringStopBits = ui->stopBitsBox->currentText();

    currentSettings.flowControl = static_cast<QSerialPort::FlowControl>(

ui->flowControlBox->itemData(ui->flowControlBox->currentIndex()).toInt());
    currentSettings.stringFlowControl = ui->flowControlBox->currentText();
```

```
}
/**
 * @brief SettingsDialog::on_btn_Refresh_clicked
 *  刷新按钮点击事件
 */
void SettingsDialog::on_btn_Refresh_clicked()
{
    this->fillPortsInfo();
}
```

充值 rechargedialog.cpp

```
#include "rechargedialog.h"
#include "ui_rechargedialog.h"
#include "database/dbmanager.h"
#include <QRegExp>
#include <QRegExpValidator>
/****************************************
 *作者: jianghj@up-tech.com
 *日期: 2016-09-20
 *描述:  充值对话框,本对话框需要检测管理员是否登陆
****************************************/
RechargeDialog::RechargeDialog(QWidget *parent,SerialPortThread *serialPortThread) :
    QDialog(parent),
    ui(new Ui::RechargeDialog)
{
    ui->setupUi(this);
    currentOps = -1;
    last_value = 0.0;
    this->serialPortThread = serialPortThread;

connect( this->serialPortThread,SIGNAL(receivedMsg(QByteArray)),this,SLOT(onDecodeFra
me(QByteArray)));//连接收到信息槽与信号
    m1356dll = new M1356Dll();
    messageBox = new QMessageBox(this);
    messageBox->setStandardButtons(QMessageBox::Yes);
    messageBox->setWindowTitle(tr("温馨提示"));
    messageBox->setIcon(QMessageBox::Warning);
    connect(this,SIGNAL(calcOps(float)),this,SLOT(on_readValue(float)));//连接槽与信号
    QRegExp rx("^[1-9]{1,3}\\.[0-9]{1,2}");
    QRegExpValidator *validator = new QRegExpValidator(rx, this);
    ui->lineEdit_money->setValidator(validator);
    ui->lineEdit_money->installEventFilter(this);
}

RechargeDialog::~RechargeDialog()
{
    delete m1356dll;
    delete messageBox;
    delete ui;
}
/**
 * @brief RechargeDialog::on_btn_recharge_clicked
 *  充值按钮点击事件
 */
void RechargeDialog::on_btn_recharge_clicked()
```

```
{
    QString cardId = ui->lineEdit_cardId->text();
    if(cardId.count() <= 2)
    {
        ui->labelMessage->setText(tr("好像没看见卡号,请先读取卡号!"));
        return;
    }
    QString recharge = ui->lineEdit_money->text();
    float money = recharge.toFloat();
    QString currentValue = ui->label_currentValue->text().split(tr(" ")).at(0);
    if(money == 0.0)
    {
        ui->labelMessage->setText(tr("请填写充值金额!"));
        return;
    }
    ui->label_chargeValue->setText(recharge + tr("  元"));
    last_value = money + currentValue.toFloat();
    if(last_value >= 999.99)
    {
        ui->labelMessage->setText(tr("如果充值,您卡内余额超限,为了您的财产安全,请先
消费后再充"));
        return;
    }
    this->authentication();
    currentOps = 20;
}
/**
 * @brief RechargeDialog::eventFilter
 * @param obj  触发的对象
 * @param event  当前的事件
 * @return  此处只在RechargeDialog前面过滤,返回值含义未变
 *  事件过滤器
 */
bool RechargeDialog::eventFilter(QObject *obj, QEvent *event)
{
    if(obj == ui->lineEdit_money)
    {
        if(event->type() == QEvent::FocusOut)
        {
            QString value =    ((QLineEdit *)obj)->text();
            if(value.length() == 0)
                ((QLineEdit *)obj)->setText(tr("0.00"));
            else if(value.endsWith('.'))
                ((QLineEdit *)obj)->setText(value + tr("00"));
            else if(!value.contains('.'))
                ((QLineEdit *)obj)->setText(value + tr(".00"));
            else if(value.right(value.length() - value.indexOf('.')).length() == 2)
                ((QLineEdit *)obj)->setText(value + tr("0"));
        }
    }
    return QDialog::eventFilter(obj,event);
}
/**
 * @brief RechargeDialog::on_btn_return_clicked
 *  返回按钮点击事件
 */
```

```
void RechargeDialog::on_btn_return_clicked()
{
    this->close();
}
/**
 * @brief RechargeDialog::authentication
 *  授权操作
 */
void RechargeDialog::authentication()
{
    uint16 frameLen;
    quint8 buffer[8];
    uint8 *p;
    buffer[0] = 0x60;     //A密钥
    buffer[1] = 0x09;     // 绝对块号
    for(int i = 2 ; i < 8 ; i ++)
        buffer[i] = 0xFF;
    p = m1356dll->RC632_SendCmdReq(RC632_CMD_AUTHENTICATION,buffer,8);//连接串口
    frameLen = BUILD_UINT16(p[0], p[1]);
    serialPortThread->writeData((char *)(p + 2 ),frameLen);//写数据
}
/**
 * @brief RechargeDialog::on_btn_clear_clicked
 *  清除按钮点击事件
 */
void RechargeDialog::on_btn_clear_clicked()
{
    ui->label_chargeValue->setText(tr("0.00  元"));
    ui->label_currentValue->setText(tr("0.00  元"));
    ui->label_lastValue->setText(tr("0.00  元"));
    ui->lineEdit_cardId->clear();
    ui->lineEdit_money->clear();
    ui->labelMessage->clear();
}
/**
 * @brief ConsumePage::on_readValue
 * @param value  卡内余额
 *  读取卡内余额后调用
 */
void RechargeDialog::on_readValue(float value)
{
    switch (currentOps) {
    case 21:
        currentOps = -1;
        ui->label_currentValue->setText(QString::number(value,'f',2) + tr("  元"));
        break;
    case 20:
        {
            currentOps = -1;
            ui->label_lastValue->setText(QString::number(value,'f',2) + tr("  元"));
            RechargeTableModel *rechargemodel = new RechargeTableModel(this);
            rechargemodel->bindTable();
            rechargemodel->addRecord(ui->lineEdit_cardId->text(),CurrentDateTime(),
                                     ui->label_currentValue->text().split(tr(" ")).at(0),
```

```
                                                ui->label_chargeValue->text().split(tr(" ")).at(0),
                                                ui->label_lastValue->text().split(tr(" ")).at(0),tr("
校园充值中心"));
                ui->labelMessage->setText(tr("充值成功!"));
                ui->lineEdit_cardId->clear();
                ui->lineEdit_money->setText(tr("0.00"));
            }
            break;
        default:
            break;
        }
}
/**
 * @brief RechargeDialog::on_btn_inventory_clicked
 *  识别按钮点击事件
 */
void RechargeDialog::on_btn_inventory_clicked()
{
    if(!serialPortThread->serialPortIsOpen())
    {
        messageBox->setText(tr("请先连接读卡器后再试！"));
        messageBox->exec();
        return;
    }
    uint16 frameLen;
    quint8 buffer[1];
    uint8 *p;
    memset(buffer, 0, 1);
    buffer[0] = RC632_14443_ALL;
    p = m1356dll->RC632_SendCmdReq(RC632_CMD_REQUEST_A,buffer,1);
    frameLen = BUILD_UINT16(p[0], p[1]);
    serialPortThread->writeData((char *)(p + 2 ),frameLen);
    this->on_btn_clear_clicked();
}
/**
 * @brief RechargeDialog::on_cardIdReceived
 * @param tagId  卡号
 *  接收到卡号时调用
 */
void RechargeDialog::on_cardIdReceived(QString tagId)
{
    bool flag = false;
    RegisterTableModel *model = new RegisterTableModel(this);
    model->bindTable();
    model->findRecord(tagId);
    if(model->findRecord(tagId) == -1)
        flag = true;
    ui->lineEdit_cardId->setText(tagId);
    if(flag)
    {
        messageBox->setText("该卡未注册,不能使用,谢谢!");
        messageBox->exec();
        ui->btn_recharge->setEnabled(false);
    }
    else
    {
```

```
            ui->btn_recharge->setEnabled(true);
            currentOps = 21;
            this->authentication();
        }
}
/**
 * @brief RechargeDialog::onDecodeFrame
 * @param bytes  接收到的数据
 *  串口接收槽函数
 */
void RechargeDialog::onDecodeFrame(QByteArray bytes)
{
        M1356_RspFrame_t frame = m1356dll->M1356_RspFrameConstructor(bytes);
        if(frame.status == "00")
        {
            if(frame.cmd.remove(" ") == "0702")//授权成功
            {
                switch (currentOps) {
                case 20: //init
                {
                    uint16 frameLen;
                    quint8 buffer[5];
                    uint8 *p;
                    buffer[0] = 0x9;
                    memset(buffer+1, 0, 5);
                    float value = last_value;
                    memcpy(buffer + 1,&value,4);
                    p =
m1356dll->RC632_SendCmdReq(RC632_CMD_M1INITVAL,buffer,5);
                    frameLen = BUILD_UINT16(p[0], p[1]);
                    serialPortThread->writeData((char *)(p + 2 ),frameLen);
                }
                    break;
                case 21: //value
                {
                    uint16 frameLen;
                    quint8 buffer[1];
                    uint8 *p;
                    buffer[0] = 0x9;
                    p = m1356dll->RC632_SendCmdReq(RC632_CMD_M1READVAL,
(const uint8*)buffer, 1);
                    frameLen = BUILD_UINT16(p[0], p[1]);
                    serialPortThread->writeData((char *)(p + 2 ),frameLen);
                }
                    break;
                default:
                    break;
                }
            }
            else if(frame.cmd.remove(" ") == "0B02" && currentOps == 21)
            {
                Float2Bytes temp;
                QSTRING_TO_HEX(frame.vdata.remove(" "),(quint8 *)temp.value_b,4);
                QString modify_value = QString::number(temp.value_f,'f',2);
                emit calcOps(modify_value.toFloat());
            }
            else if(frame.cmd.remove(" ") == "0A02" && currentOps == 20)
```

```
            {
                emit calcOps(last_value);
            }
        }
        else {
            if(frame.cmd.remove(" ") == "0702")//授权成功
            {
                messageBox->setText(tr("授权失败,请将卡放到可识别区域!"));
                messageBox->exec();
                currentOps = -1;
            }
            else if(frame.cmd.remove(" ") == "0B02")
            {
                messageBox->setText(tr("读卡失败,请注意卡和读卡器之间的距离"));
                messageBox->exec();
                currentOps = -1;
            }
            else if(frame.cmd.remove(" ") == "0A02")
            {
                messageBox->setText(tr("写卡失败,请将卡放到可识别区域!"));
                messageBox->exec();
                currentOps = -1;
            }
        }
}
```

3 实验三　高频读写器实验 ISO15693

3.1　实验目的

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。

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。

3.2　实验内容及结果

1、完成 ISO15693 协议下的单标签和多标签手工寻卡和自动寻卡；

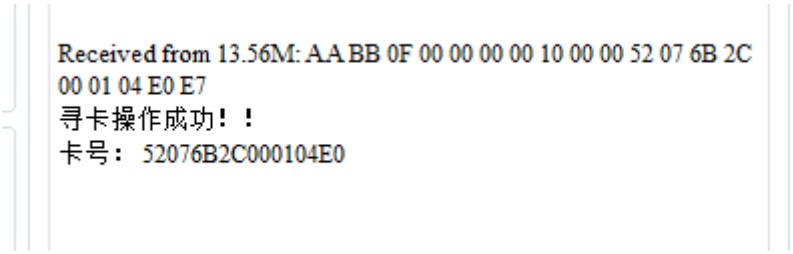

图 12 单标签寻卡

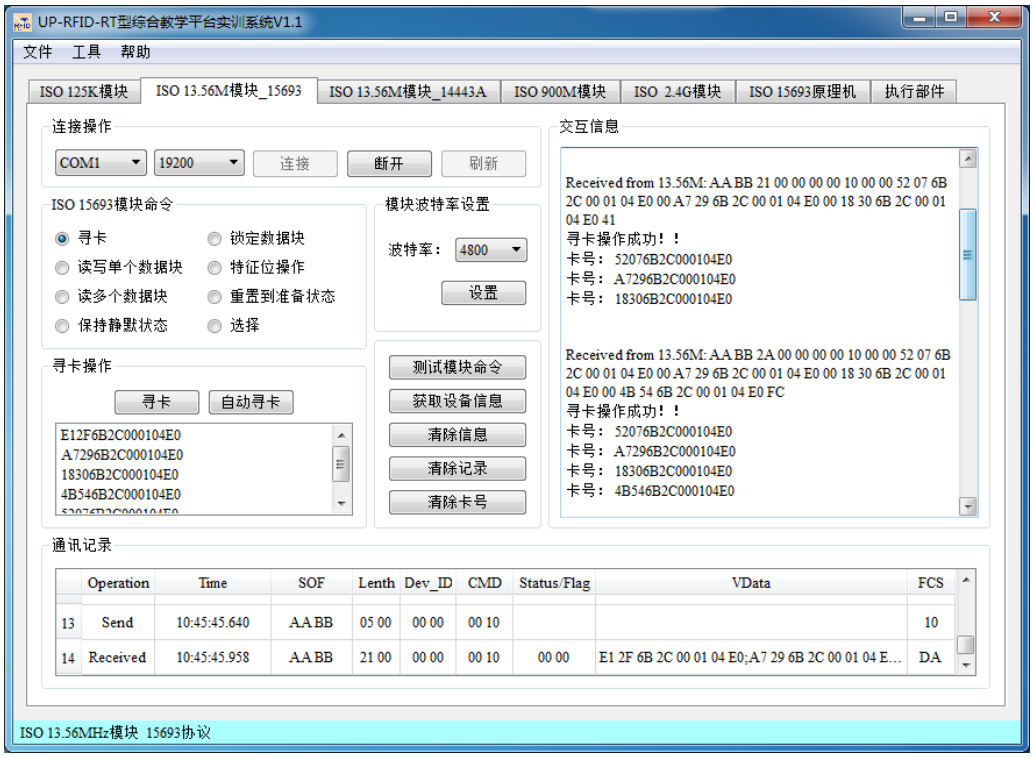

图 13 多标签寻卡

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；

图 14 指定地址范围读卡实验

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；

图 15 静默实验

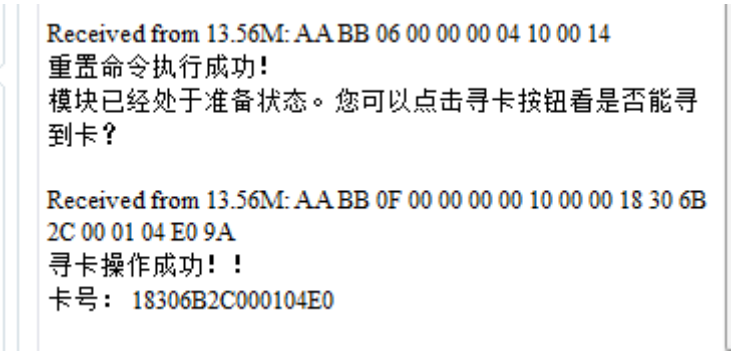

图 16 恢复实验

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

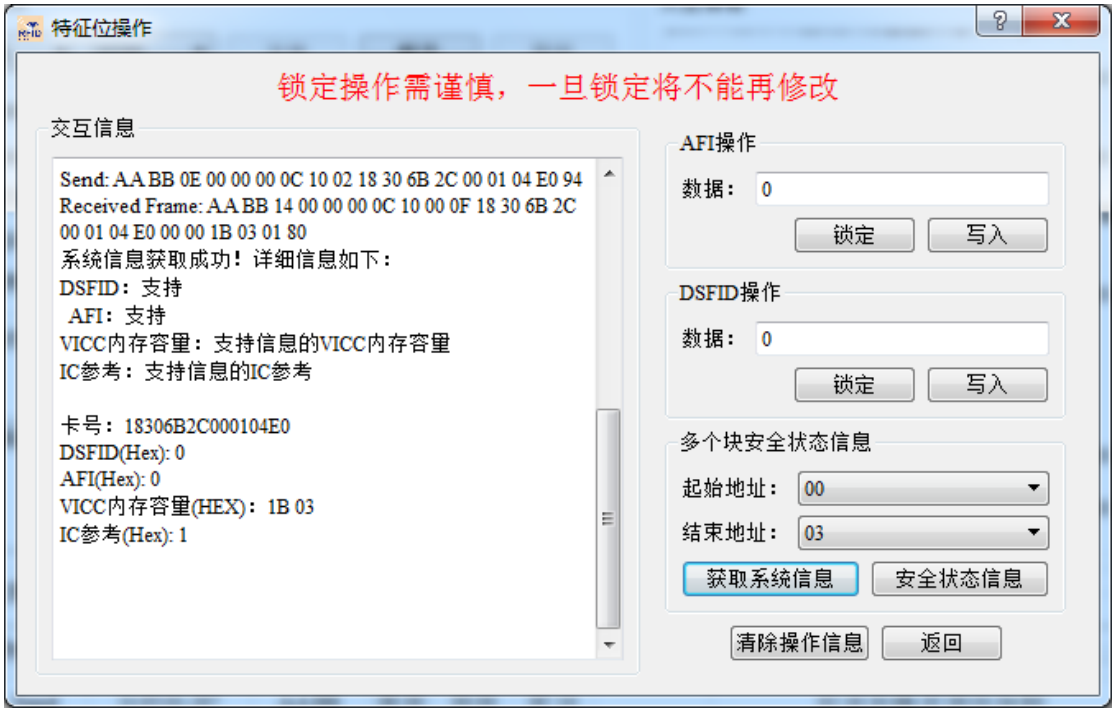

图 17 DSFID、AFI 的读写和块安全位的读取实验

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

图 18 开发实例

3.3 实验体会与总结

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。

2、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。

3、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。

总结：通过这次实验，我学会了通过试验箱对 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，了解了握高频读写器 API 函数的调用方法。

3.4 核心源码说明

```
#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"

#define Button_Count_BOOKS 4//按钮个数
#define Edit_Count_BOOKS 6//文本框个数
#define Label_Count_BOOKS 6//标签个数
#define Table_Column_BOOKS 6//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
enum Edit_Index_Books{ID_Books = 0, Name_Books, Author_Books, PublishingHouse_Books, Count_Books, Resid

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

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
};

#endif // BOOKSMANAGE_H
#include "borrow_return.h"

//借书界面
Borrow_Return::Borrow_Return(QWidget *parent) : QWidget(parent)
{
    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);
}
//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）"<<"乘
    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());
}

//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
    {
        for(int i=0; i < Edit_Count_BORROW_RETURN; i++)
        {
            Edit_User[i]->setText(query.value(i).toString());
        }
        //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())
        {
            if(sql->SelectRecord(Edit_User[CardId_User_Borrow]->text(), query.value(0).toString()).next())
            {
                return;
            }
            if(query.value(5).toInt() <= 0)
```

```
                {
                    return;
                }
                if(sql->InsertRecord(Edit_User[CardId_User_Borrow]->text(), query.value(0).toString()))//将用户ID和
                {
                    //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
                }
            }
            else
            {
                if(!sql->SelectRecord(Edit_User[CardId_User_Borrow]->text(), query.value(0).toString()).next())
                {
                    return;
                }
                if(sql->DeleteRecord(Edit_User[CardId_User_Borrow]->text(), query.value(0).toString()))//将用户ID和
                {
                    //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
                }
            }
            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
} #include "record.h"

//还书界面
Record::Record(QWidget *parent) : QWidget(parent)
{
    QVBoxLayout *MainLayout = new QVBoxLayout();//主布局
    QHBoxLayout *TableLayout = new QHBoxLayout();//表格布局
    QHBoxLayout *ButtonLayout = new QHBoxLayout();//按钮布局
    QHBoxLayout *EditLayout = new QHBoxLayout();//按钮布局
    QVBoxLayout *TopLayout = new QVBoxLayout();//上部布局
    QStringList LabelText,ButtonText;
    ButtonText<<"搜索"<<"删除";
    for(int i=0; i<Button_Count_Record; i++)
    {
        Button[i] = new QPushButton();
        Button[i]->setText(ButtonText.at(i));
        ButtonLayout->addWidget(Button[i]);
    }
    ButtonLayout->addStretch();
```

```
    LabelText<<"用户卡号"<<"书籍卡号";
    QString pattern("[A-Fa-f9-0]*");
    QRegExp regExp(pattern);
    for(int i=0; i<Edit_Count_Record; i++)
    {
        Label[i] = new QLabel();
        Label[i]->setText(LabelText.at(i));
        EditLayout->addWidget(Label[i]);
        Edit[i] = new QLineEdit();
        EditLayout->addWidget(Edit[i]);
        Edit[i]->setValidator(new QRegExpValidator(regExp, this));
    }
    //组合框
    QGroupBox *TabGroupBox = new QGroupBox();
    QGroupBox *GroupBox = new QGroupBox();
    sql = new Sqlite();

    Table = new QTableWidget();//表格
    Table->setColumnCount(Table_Column_Record);//设置列数
    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);
    TopLayout->addLayout(EditLayout);
    GroupBox->setLayout(TopLayout);
    TabGroupBox->setTitle("借书列表");//设置组合框标题
    TabGroupBox->setLayout(TableLayout);//这是组合框布局

    //设置布局
    MainLayout->addWidget(GroupBox);
    MainLayout->addLayout(ButtonLayout);
    MainLayout->addWidget(TabGroupBox);
    this->setLayout(MainLayout);
    SetSlot();
}

void Record::SetSlot()
{
    connect(Button[Delete_Record],SIGNAL(clicked()),this,SLOT(delete_record()));//删除按钮连接槽函数delete_
    connect(Button[Select_Record],SIGNAL(clicked()),this,SLOT(select_record()));//查找按钮连接槽函数select_R
    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_
}
//搜索按钮单击事件
void Record::select_record()
{
    QSqlQuery query;
    query = sql->SelectRecord(Edit[UserID_Record]->text(),Edit[BookID_Record]->text());
    ShowTable(query);//更新表格
    ClearEdit();//清空文本框
}

//删除按钮槽函数
void Record::delete_record()
{
```

```
    //删除书籍
    bool ret = sql->DeleteRecord(Edit[UserID_Record]->text(),Edit[BookID_Record]->text());
    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->SelectRecord());//更新表格
}

//清空文本框
void Record::ClearEdit()
{
    for(int i = 0; i < Edit_Count_Record; 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 Record::get_table_line(int row, int col)
{
    for(int i = 0; i < Edit_Count_Record; i++)
    {
        Edit[i]->setText(Table->item(row,i)->text());
    }
}
//显示表格
void Record::ShowTable(QSqlQuery query)
{
    //表头
    Table->setHorizontalHeaderLabels(QStringList()<<"用户卡号"<<"书籍卡号");
    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++)
        {
            Table->setItem(row, col, new QTableWidgetItem(query.value(col).toString()));//显示信息
        }
        row++;
    }while(query.next());
}
```

```
//清空文本框和表格
void Record::Clear()
{
	ShowTable(sql->SelectRecord());
}

//设置卡号
void Record::SetCard(QString cardID)
{
	QSqlQuery query = sql->SelectUser(cardID);
	if(query.next())//如果是用户
	{
		Edit[UserID_Record]->setText(cardID);//显示用户卡号
		return;
	}
	query = sql->SelectBooks(cardID);
	if(query.next())//如果是书
	{
		Edit[BookID_Record]->setText(cardID);//显示用户卡号
	}
} #include "sqlite.h"

Sqlite::Sqlite()
{
}
/*连接数据库*/
bool Sqlite::Connect()
{
	db = QSqlDatabase::addDatabase("QSQLITE");
	db.setDatabaseName(DATABASE);
	if(!db.open()) return false;
	QSqlQuery query;
	query.exec("create table user_15693 (cardID vchar, name vchar, gender vchar, age int, primary key (cardID))");
	query.exec("create table books_15693 (booksID vchar, name vchar, author vchar, publishing_house vchar, count i
	query.exec("create table record_15693 (cardID vchar, booksID vchar, FOREIGN KEY (cardID ) REFERENCES
user(booksID))");
	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 + ");";
	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)
{
    return Insert("user_15693", """+cardID+"', "'+name+"', "'+gender+"', "+QString::number(age));
}
//向books表中添加
bool Sqlite::InsertBooks(QString booksID, QString name, QString author, QString publishing_house, int count, int resi
{
    return Insert("books_15693", """+booksID+"', "'+name+"', "'+author+"', "'+publishing_house+"', "+QString::num
}
//向record表中添加
bool Sqlite::InsertRecord(QString cardID, QString booksID)
{
    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 res
{
        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)
{
    return Updata("user_15693","cardID = '"+cardID+"', name = '"+name+"', gender = '"+gender+"', age = "+QString
}
//修改books表中数据
bool Sqlite::UpdataBooks(QString booksID, QString name, QString author, QString publishing_house, int count, int re
{
    return Updata("books_15693","booksID = '"+booksID+"', name = '"+name+"', author = '"+author+"', publishing_
"+QString::number(count)+", residue = "+QString::number(residue), "booksID = '"+booksID+"'");
}
//查询user表中数据
QSqlQuery Sqlite::SelectUser(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 Select("user_15693", "*", where);
}
//查询books表中数据
QSqlQuery Sqlite::SelectBooks(QString booksID, QString name, QString author, QString publishing_house, int count)
```

```
{
    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));
    }

    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+"'
} #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();
    }
} #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）"};//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]);
    }
    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));

    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
    }
    ButtonLayout->addStretch(0);
    ButtonLayout->setSpacing(20);
```

```
    Table = new QTableWidget();
    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_bo
    connect(Button[Updata_Books],SIGNAL(clicked()),this,SLOT(updata_books()));//修改按钮连接槽函数updata_
    connect(Button[Select_Books],SIGNAL(clicked()),this,SLOT(select_books()));//查找按钮连接槽函数select_bo
    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_
}

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, QMess
            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
        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, QMessage
        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[PublishingHou
        if(!ret)
        {
            QMessageBox::warning(NULL, "warning", "添加失败，卡号已存在！", QMessageBox::Yes, QMessageB
            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
            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
        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
        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, QMessage
        return;
    }
    //修改书籍信息
    bool ret =
sql->UpdataBooks(Edit[ID_Books]->text(),Edit[Name_Books]->text(),Edit[Author_Books]->text(),Edit[PublishingHo
Edit[Residue_Books]->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(),E
    else
        query =
sql->SelectBooks(Edit[ID_Books]->text(),Edit[Name_Books]->text(),Edit[Author_Books]->text(),Edit[PublishingHou
    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
    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());
}

//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());
}
```

4 实验四　超高频读写器实验

4.1　实验目的

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。

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。

4.2　实验内容及结果

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；

图 19 单标签识别实验

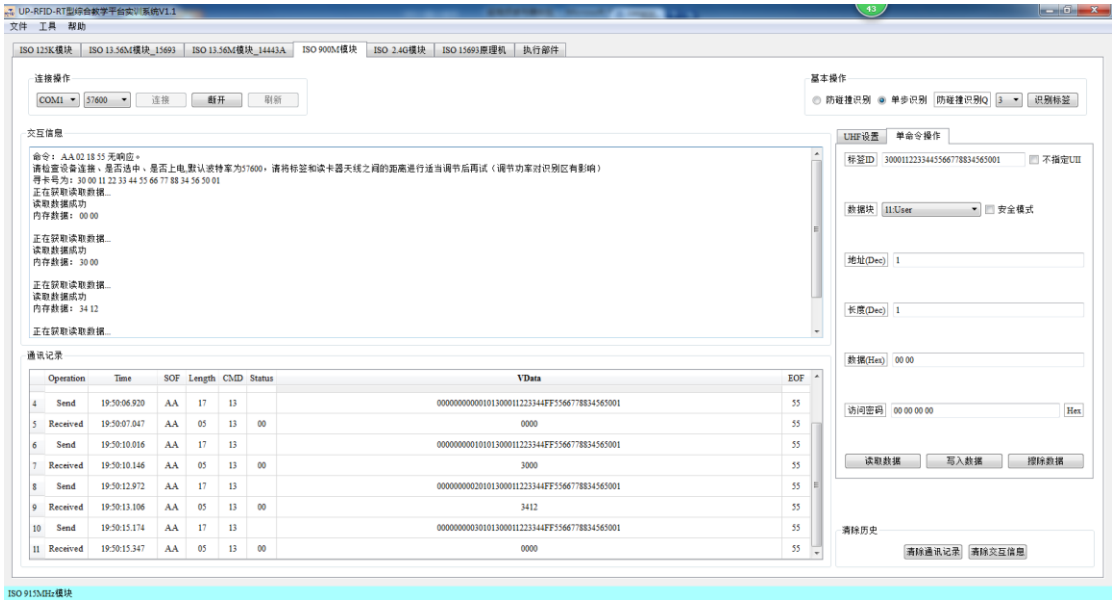

图 20 多标签实验

两张卡重叠地读取（即一张在上一张在下），只能读取到下面的卡。

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；

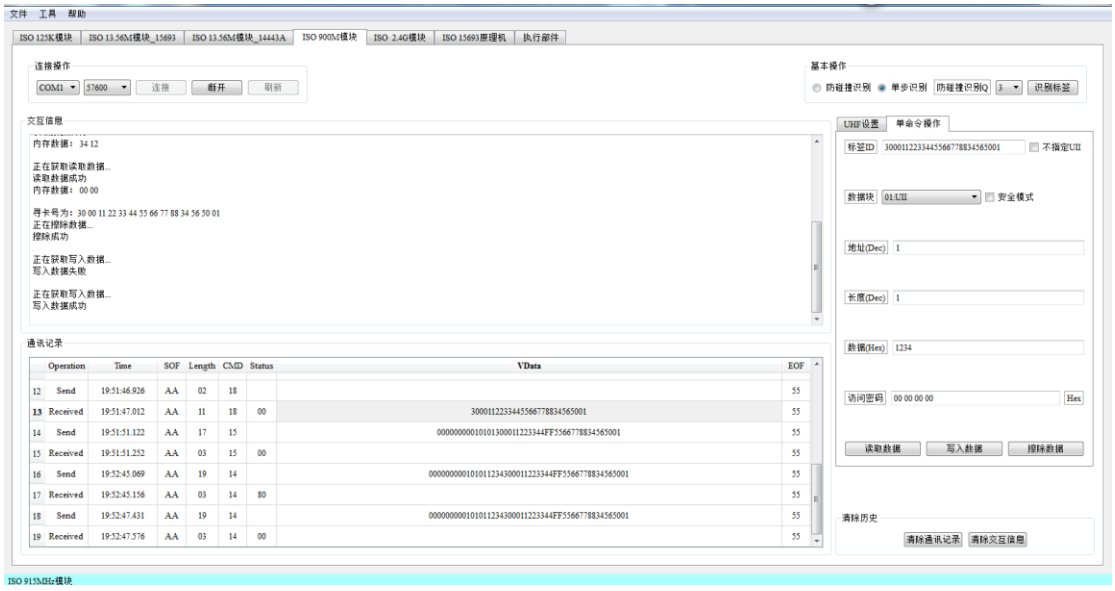

图 21 读写擦除

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。

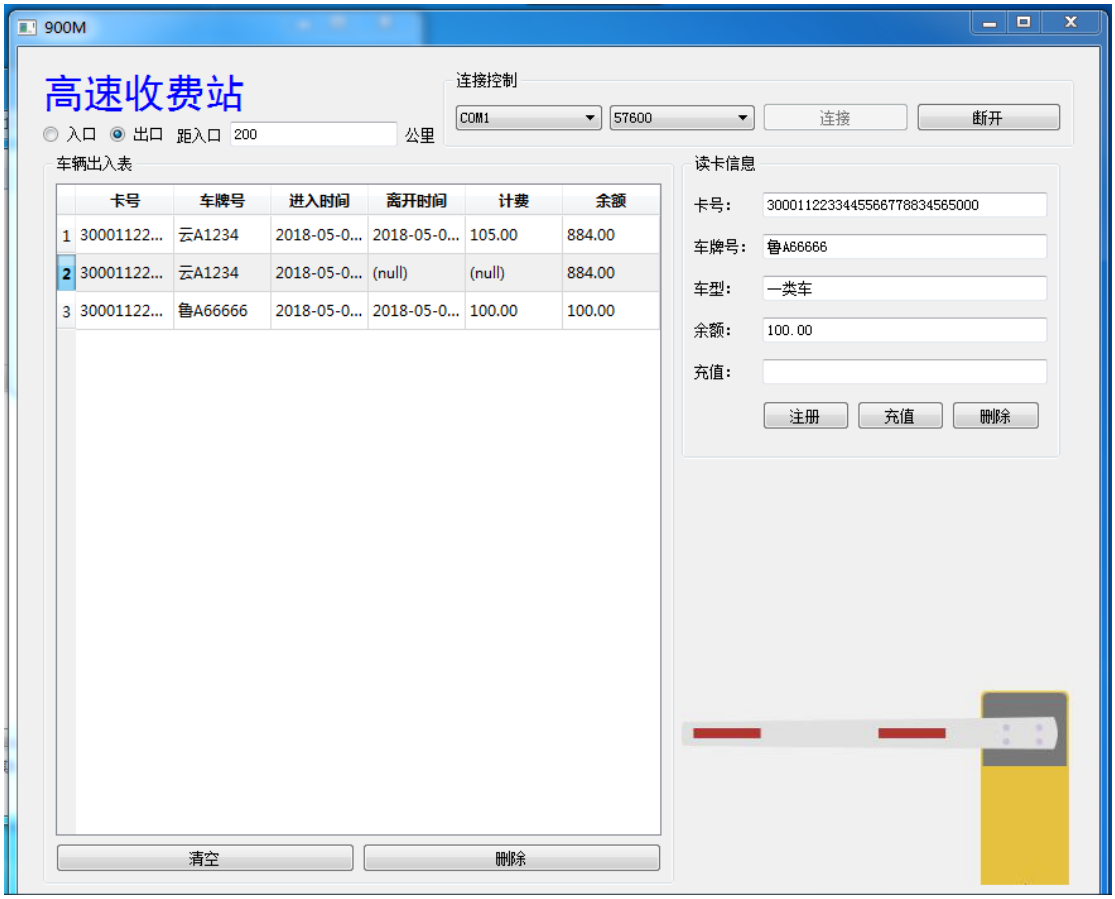

图 22 开发实例

4.3 实验体会与总结

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；

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、EPC 标签存储器分为哪几个区？各区有何功能？

Reserved 区：存储 Kill Password（灭活口令）和 Access Password（访问口令）；

3、EPC 标签可以通过哪些措施来保证各个存储区的信息安全？

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。

4、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。

总结：通过这次实验，我学会了通过试验箱对 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。

4.4 开发实例源码

```
#include "regist_widget.h"
#include <QMessageBox>
#include <QDebug>

Regist_Widget::Regist_Widget(QWidget *parent) : QWidget(parent)
{
    /*注册账号界面 按钮和标签的文本*/
    char Button_Name[][50] = {"提交","重置","取消"};
    char Label_Name[][50] = {"卡号：","车牌号：", "金额：", "车型："};

    QVBoxLayout *label_layout = new QVBoxLayout();//标签的布局
    /*实例化标签，将标签添加到布局中*/
    for(int i = 0; i < LABEL_COUNT_REGIST; i++)
    {
        Label[i] = new QLabel();
        Label[i]->setText(Label_Name[i]);
        label_layout->addWidget(Label[i]);
    }
    label_layout->setSpacing(30);

    QVBoxLayout *edit_layout = new QVBoxLayout();//文本框的布局
    /*实例化文本框， 将文本框添加到布局中*/
    for(int i = 0; i < EDIT_COUNT_REGIST; i++)
    {
        Edit[i] = new QLineEdit();
        edit_layout->addWidget(Edit[i]);
    }
    QString pattern("[9-0]{3}");
    QRegExp regExp(pattern);
    Edit[Balance_Regist]->setValidator(new QRegExpValidator(regExp, this));
    pattern = "[\u4e00-\u9fa5]{1}[A-Fa-f]{1}[9-0]{5}";
    regExp.setPattern(pattern);
    Edit[Plate_number_Regist]->setValidator(new QRegExpValidator(regExp, this));

    /*实例化车类型下拉列表*/
    Types = new QComboBox();
    /*将类型添加到列表中*/
    QStringList types_text;
    types_text<<"一类车"<<"二类车"<<"三类车"<<"四类车"<<"五类车";
    Types->addItems(types_text);
    /*将列表添加到布局中*/
    edit_layout->addWidget(Types);

    /*说明标签*/
    instruction = new QLabel("一类车:0.5元/公里  二类车:1元/公里  \n三类车:1.5/公里     四类车:1.8/公里  \n五类车:2/公里");

    edit_layout->setSpacing(30);//设置间距

    /*按钮布局*/
    QHBoxLayout *button_layout = new QHBoxLayout();
```

```
    /*实例化按钮，将按钮添加到布局中*/
    for(int i = 0; i< BUTTON_COUNT_REGIST; i++)
    {
        PushButton[i] = new QPushButton();
        PushButton[i]->setText(Button_Name[i]);
        button_layout->addWidget(PushButton[i]);
    }
    button_layout->setSpacing(30);

    /*设置总体布局*/
    QGridLayout *mainlayout = new QGridLayout();
    mainlayout->addLayout(label_layout,0,0,1,1);
    mainlayout->addLayout(edit_layout,0,1,1,1);
    mainlayout->addWidget(instruction, 1,1,1,2);
    mainlayout->addLayout(button_layout,2,0,1,2);

    this->setLayout(mainlayout);

    /*按钮的单机事件连接槽*/
    connect(PushButton[Regist_Regist], SIGNAL(clicked()), this,
SLOT(Uhf_Regist_Button_Click()));   //连接按钮单击事件连接Uhf_Connect_Button_Click()函数
    connect(PushButton[Rese_Registt], SIGNAL(clicked()), this,
SLOT(Uhf_Rese_Button_Click()));//断开按钮单击事件连接Uhf_Disconnect_Button_Click()函数
    connect(PushButton[Cancel_Regist], SIGNAL(clicked()), this,
SLOT(Uhf_Cancel_Button_Click()));//注册按钮单击事件连接Uhf_Update_Button_Click()函数
}

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();
        }
}
void Regist_Widget::Uhf_Rese_Button_Click()
{
        /*清空文本框*/
        for(int i = 0; i < EDIT_COUNT_REGIST; i++)
        {
            Edit[i]->clear();
        }
}
void Regist_Widget::Uhf_Cancel_Button_Click()
{
        /*关闭窗口*/
        this->close();
}
void Regist_Widget::Set_CardID(QString Card_id)
{
        Edit[0]->setText(Card_id);
}
#include "sqlite.h"

Sqlite::Sqlite()
{
}
/*连接数据库*/
bool Sqlite::connect()
{
        db = QSqlDatabase::addDatabase("QSQLITE");
        db.setDatabaseName(DATABASE);
        if(!db.open()) return false;
        QSqlQuery query;
        /*创建user表  cardID和plate_number作为联合主键*/
        //卡号、车牌号、车类型、余额
        query.exec("create table user_900M (cardID vchar, plate_number vchar, type vchar, balance
float, primary key (cardID,plate_number))");
        /*创建record表  cardID和plate_number作为外键*/
        //卡号、车牌号、进入时间、出去时间、计费、余额
        query.exec("create table record_900M (cardID vchar, plate_number vchar, inTime vchar,
outTime vchar, consumption float, balance float,FOREIGN KEY (cardID ) REFERENCES
user(cardID), FOREIGN KEY (plate_number ) REFERENCES user(plate_number))");
```

```
        return true;
}

/*添加user数据*/
bool Sqlite::add_user(char *cardID, char *plate_number, char *type, float balance)
{
        char command[256];
        sprintf(command, "insert into user_900M values('%s', '%s', '%s', %2f);", cardID,
plate_number, type, balance);
        qDebug("ADD USER:%s\n",command);
        QSqlQuery query;
        return query.exec(command);
}

/*添加record数据*/
bool Sqlite::add_record(char *cardID, char *plate_number, char *inTime, char *outTime, float
consumption,float balance)
{
        char command[256];
        sprintf(command, "insert into record_900M values('%s', '%s', '%s', '%s', %2f, %2f);", cardID,
plate_number, inTime, outTime, consumption, balance);
        qDebug("ADD RECORD:%s\n",command);
        QSqlQuery query;
        return query.exec(command);
}
/*更改user中的数据*/
bool Sqlite::update_user(char *cardID, char *plate_number, float balance)
{
        char command[256];
        sprintf(command, "update user_900M set balance = %2f where cardID = '%s' and
plate_number = '%s'", balance, cardID, plate_number);
        QSqlQuery query;
        qDebug("UPDATE USER:%s\n",command);
        return query.exec(command);
}
//更改record中的数据
bool Sqlite::update_record(char *cardID, char *plate_number, char *inTime, char *outTime, float
consumption,float balance)
{
        char command[256];
        sprintf(command, "update record_900M set outTime = '%s', consumption = %2f, balance
= %2f where cardID = '%s' and plate_number = '%s' and inTime = '%s'", outTime, consumption,
balance, cardID, plate_number, inTime);
        QSqlQuery query;
        qDebug("UPDATE RECORD:%s\n",command);
        return query.exec(command);
}

//查找（表名称，条件）
QSqlQuery Sqlite::select(const char *table, char *where)
{
        char command[256];
        sprintf(command, "select * from %s", table);
        if(where!=NULL)
        {
                char tmp[256];
                strcpy(tmp,command);
```

```
            sprintf(command,"%s where %s",tmp,where);
        }
        QSqlQuery query;
        qDebug("SELECT:%s\n",command);
        query.exec(command);
        return query;
}

//删除（表名称，条件）
bool Sqlite::del(const char *table, char *where)
{
        char command[256];
        sprintf(command, "delete from %s", table);
        if(where!=NULL)
        {
            char tmp[256];
            strcpy(tmp,command);
            sprintf(command,"%s where %s",tmp,where);
        }
        QSqlQuery query;
        qDebug("DEL:%s\n",command);
        return query.exec(command);
}

Sqlite::~Sqlite()
{
        db.close();
} #include "uhf_thread.h"
#include <QMessageBox>
#include <QDebug>
#include <QObject>

UHF_Thread::UHF_Thread(QObject *parent) : QThread(parent)
{
        serialport = new QSerialPort();
        Dll = new M900Dll();//dll链接库
}
UHF_Thread::~UHF_Thread()
{

}
void UHF_Thread::run()
{
        char data[1024];
        //帧的标志，数据的长度，长度，指令，状态，数据数组下标
        int flag = UHF_RPC_SOF, length=0, len=0, cmd=0, status=0, index = 0;
        while(nRunFlag)
        {
            char ch;
            if(serialport->bytesAvailable())//如果可以读取
            {
                if(!serialport->read(&ch,1) || (ch&0xff) == 0xff)//如果没有读到数据 或者 读到
的是0xff 退出本次循环
                    continue;
                switch(flag)
                {
```

```
                case UHF_RPC_SOF://头
                    /*初始化长度、数组和数组下标*/
                    len = 0;
                    index = 0;
                    memset(data,0,sizeof(data));
                    if((ch&0xff) == UHF_SOF)
                        flag = UHF_RPC_LEN;
                    break;
                case UHF_RPC_LEN://长度
                    len = ch;
                    flag = UHF_RPC_CMD;
                    break;
                case UHF_RPC_CMD://指令
                    cmd = ch;
                    flag = UHF_RPC_STA;
                    len--;
                    break;
                case UHF_RPC_STA://状态
                    status = ch;
                    flag = UHF_RPC_DAT;
                    len--;
                    length = len-1;//length  是数据的长度
                    break;
                case UHF_RPC_DAT://数据
                    len--;
                    if(len > 1)
                    {
                        data[index++] = ch;
                    }else if(len == 1)
                    {
                        flag = UHF_RPC_EOF;
                    }else if(len <= 0)
                    {
                        flag = UHF_RPC_SOF;
                    }
                    break;
                case UHF_RPC_EOF://尾
                    flag = UHF_RPC_SOF;
                    if((ch&0xff) == UHF_EOF)
                    {
                        Process_data(cmd,data,length,status);//数据处理
                        serialport->clear();
                    }
                    break;
                default:
                    flag = UHF_RPC_SOF;
                    break;
                }
            }
            QThread::msleep(10);
        }
}

void UHF_Thread::Process_data(int cmd, char data[], int length, int status)
{
    Tools tools;
```

```
    switch(cmd)
    {
    case UHFCMD_GET_STATUS:// 询问状态
        break;
    case UHFCMD_GET_POWER:// 读取功率
        break;
    case UHFCMD_SET_POWER:// 设置功率
        break;
    case UHFCMD_GET_FRE:// 读取频率
        break;
    case UHFCMD_SET_FRE:// 设置频率
        break;
    case UHFCMD_GET_VERSION:// 读取版本信息
        break;
    case UHFCMD_INVENTORY:// 识别标签（单标签识别）
        emit this->cardID(tools.CharStringtoHexString(NULL,data,length));//发送信号cardID
（）
        break;
    case UHFCMD_INVENTORY_ANTI:// 识别标签（防碰撞识别）
        break;
    case UHFCMD_STOP_GET:// 停止操作
        break;
    case UHFCMD_READ_DATA:// 读取标签数据
        break;
    case UHFCMD_WRITE_DATA:// 写入标签数据
        break;
    case UHFCMD_ERASE_DATA:// 擦除标签数据
        break;
    case UHFCMD_LOCK_MEM:// 锁定标签
        break;
    case UHFCMD_KILL_TAG:// 销毁标签
        break;
    case UHFCMD_INVENTORY_SINGLE: // 识别标签（单步识别）
        break;
    case UHFCMD_WIEGAND_INVENTORY:// 韦根识别
        break;
    case UHFCMD_SINGLE_READ_DATA:// 读取标签数据（不指定UII）
        break;
    case UHFCMD_SINGLE_WRITE_DATA:// 写入标签数据（不指定UII）
        break;
    default:
        break;
    }
}

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;
    }
}

bool UHF_Thread::UHF_INIT()//初始化UHF
{
    char *data = (char *)Dll->UHF_Connect();//获取连接发送指令
    int len = serialport->write(data+1,data[0]);//发送数据，data,len
    if(len)
    {
        qDebug("WRITE SUCCESS!");
    }
    else
    {
        qDebug("WRITE FAILED!");
    }
    return true;
}

bool UHF_Thread::Read_CardID()
{
    char *data = (char *)Dll->UHF_Inventory();//获取读卡指令
    if(serialport->write(data+1,data[0]))//data,len
    {
        qDebug("WRITE SUCCESS!");
    }
    else
    {
        qDebug("WRITE FAILED!");
    }
    return true;
}
```

5 实验五　RFID 综合应用实验

5.1　需求分析

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。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。

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，保存个人信息（姓名，性别，年龄）至数据库中，并对卡进行初始化，即写入个人信息，写至块 0。将卡与个人姓名关联起来（采用实名制）。另外四张卡初始化与四本图书关联，在数据库中录入卡号、书名、作者、出版社、总本数、剩余本数、可借时间，标识图书的唯一性。将最近的 5 次借阅信息同时存储在数据库中。

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。当有借阅书籍超期未还时，将不能借阅新的书籍。

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。

5.2　系统详细设计

5.2.1 系统结构设计

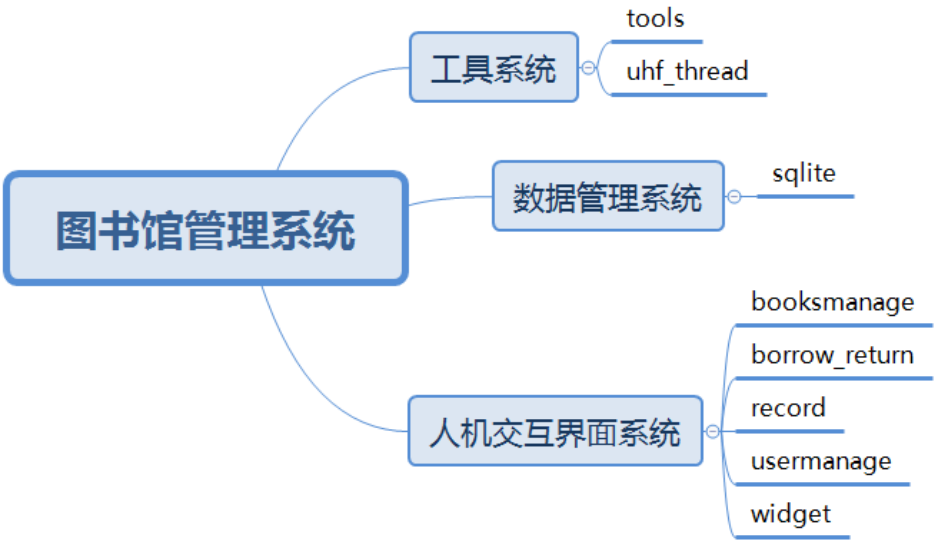

图 23 系统结构图

5.2.2 改动

1) tools:
   添加函数：
   ChineseToHex(QString Str);
   作用：输入中文字符串，输出转成用 16 进制 ASCii 码的字符串。
2) uhf_thread:
   添加函数：
   WriteCard(QString UID, int ADDR, uint8 *DATA);
   作用：将输入的 DATA 写入卡号为 UID，块号为 ADDR 的数据块。
3) sqlite:
   修改建表语句为：

```
query.exec("create table books_15693 (booksID vchar, name vchar, author vchar, publishing_house vchar, count int, residue int, availableTime int, primary key (booksID))");
query.exec("create table record_15693 (cardID vchar, booksID vchar, brrowDate vdate, FOREIGN KEY (cardID ) REFERENCES user(cardID), FOREIGN KEY (booksID ) REFERENCES user(booksID))");
```

即新增表项 availableTime 和 borrowDate。

4) booksmanagement:
   添加“可借时长”文本框及标签。
5) borrow_return:
   表格新增“借阅日期”项，当查询到有过期未还书籍时，将借书使能标志置为 0。

```
QDate today=QDate::currentDate();
QDate BorrowDate=qry.value(2).toDate();
BorrowDate=BorrowDate.addDays(days);
qDebug()<<BorrowDate;
```

```
        if(BorrowDate<today) Borrowable=0;
```

此时若要借阅新书，则弹出提示框“请先归还过期书籍”。

```
if(Borrowable==0){
    QMessageBox::warning(NULL,"warning","请先归还过期书籍！",QMessageBox::Yes,QMessageBox::Yes);
    return;
}
```

6) usermanage:

在函数 add_user();中，加入：

```
emit WriteUsrInfo(Edit[ID_User]->text(),Edit[Name_User]->text(),Edit[Gender_User]->text(),Edit[Age_User]->text(),0);
```

即当添加用户时给 widget 发送信号，控制 uhf 线程进行写卡。

在函数 delete_user();中，加入：

```
emit DeleteUserInfo(Edit[ID_User]->text());
```

即当删除用户时给 widget 发送信号，控制 uhf 线程将卡内数据清零。

7) widget:

添加函数：

Add_User_Button_Clicked(QString uid, QString name, QString gen, QString age, int rcds);

作用：响应添加用户信号，对数据进行编码，调用 WriteCard 函数写卡。

Delete_User_Button_Clicked(QString uid);

作用：响应删除用户信号，调用 WriteCard 函数清零。

新增连接语句如下：

```
connect(user_manage,SIGNAL(WriteUsrInfo(QString,QString,QString,QString,int)),this,SLOT(Add_User_Button_Clicked(QString,QString,QString,QString,int)));
connect(user_manage,SIGNAL(DeleteUserInfo(QString)),this,SLOT(Delete_User_Button_Clicked(QString)));
```

5.2.3 系统数据设计

1)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）；

2）数据字典

表 1 用户信息表数据字典

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 2 图书基本信息表

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
availableTime	int			书的可借时间

表 3 借书记录信息表

字段名	字段类型	主/外键	字段值约束	说明
cardID	vchar	外键		借书者的 ID
booksID	vchar	外键		被借阅书籍的 ID
borrowDate	vdate			此条借阅记录的日期

5.3 系统实现与系统测试

(1) 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。

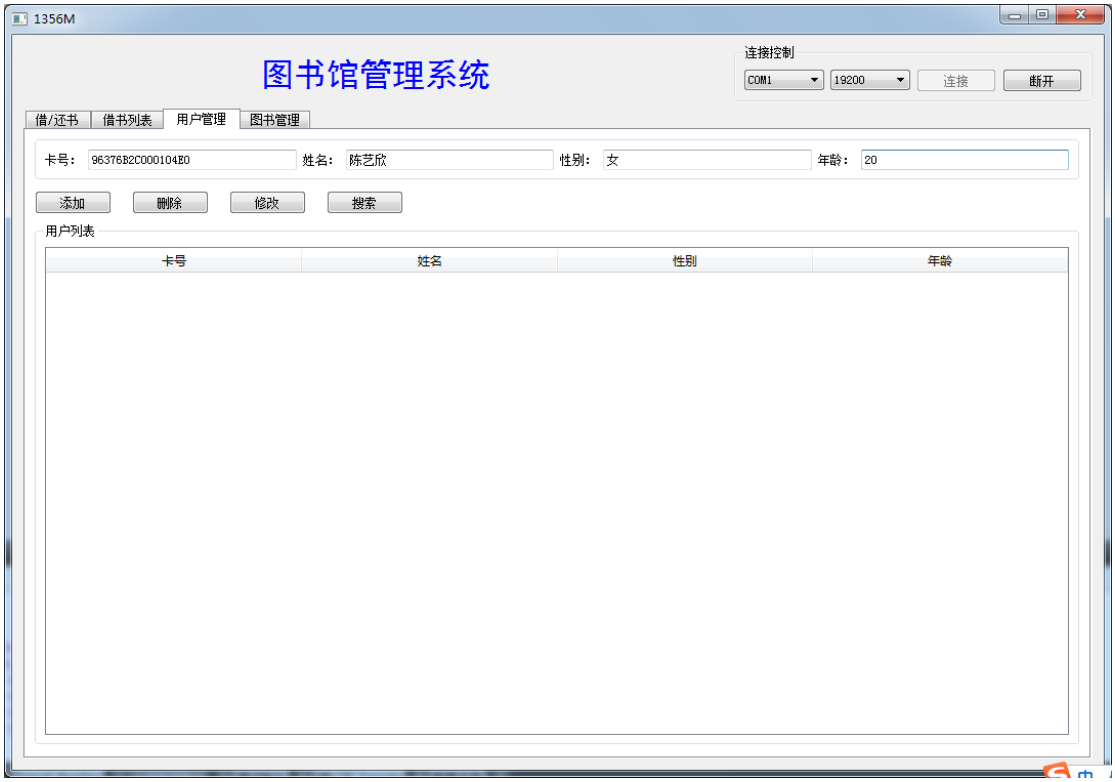

图 24 输入用户信息

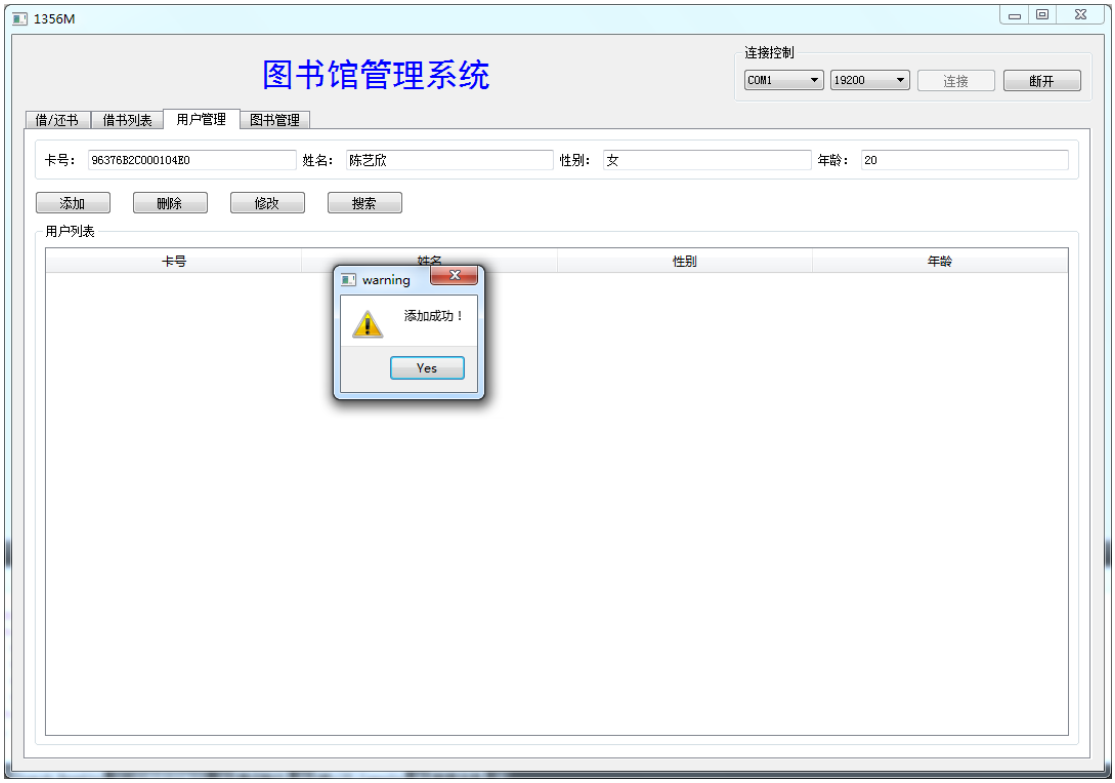

图 25 添加成功

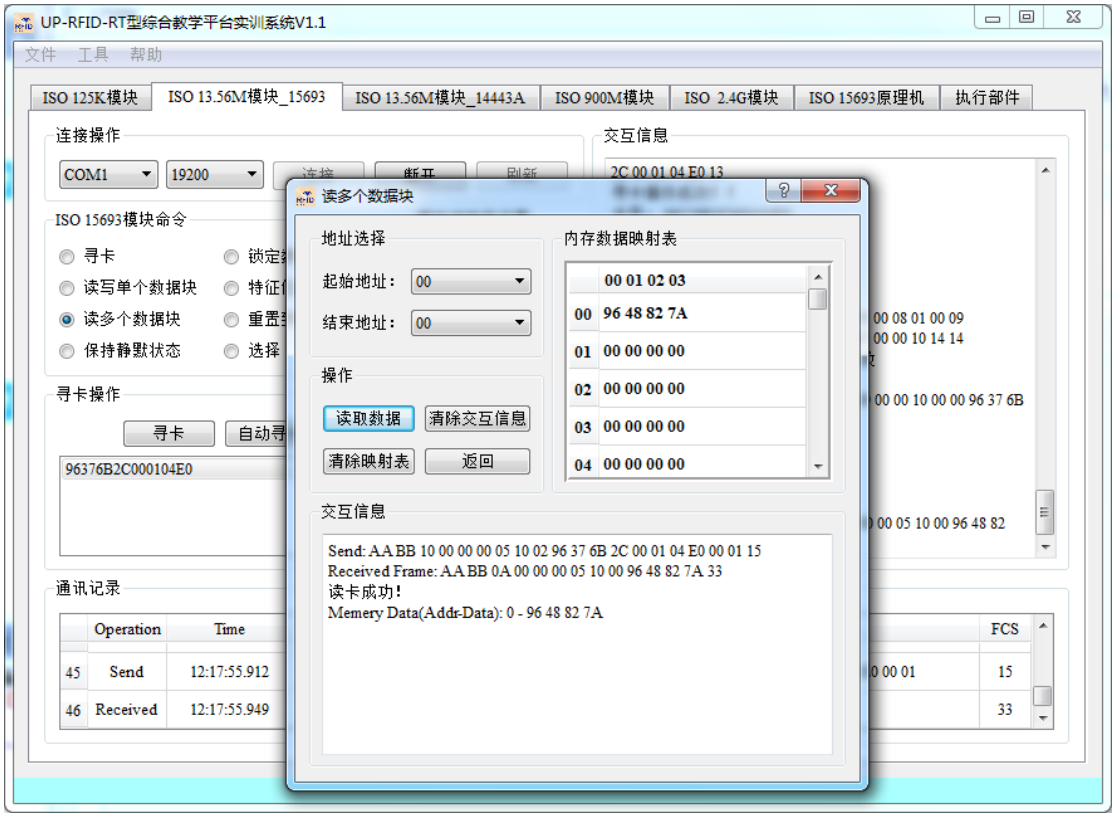

图 26 成功写入数据

(2) 图书与电子标签关联管理；

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 27 将信息与卡相关联

图 28 信息写入数据库

(3) 用户毕业时的销卡管理，清除卡内借/还数据以及个人数据；

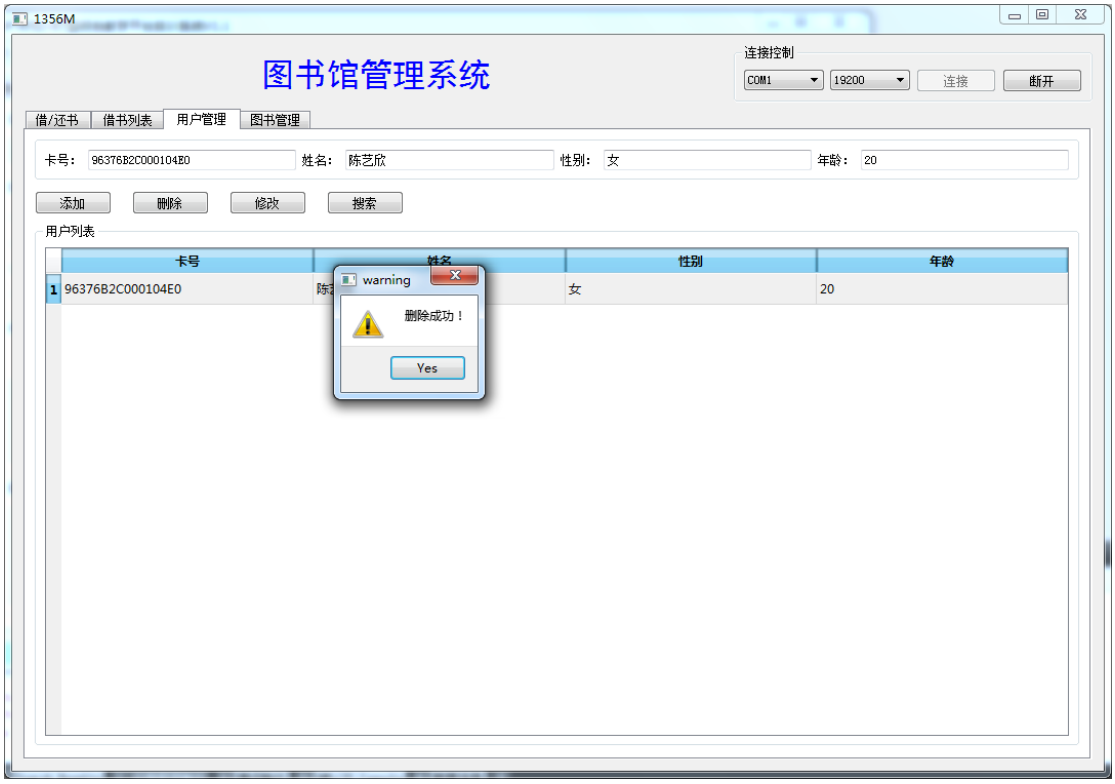

图 29 用户销卡

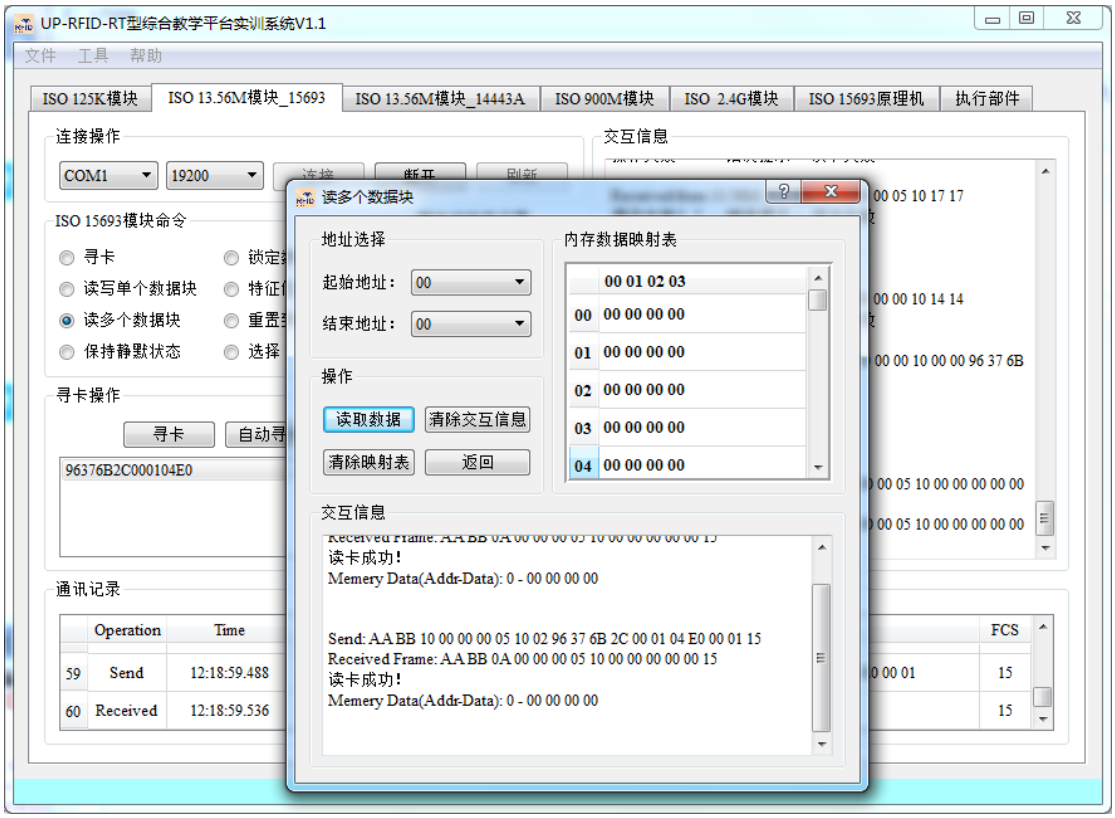

图 30 数据清零

(4) 用户借或还图书时，在数据库中保存借阅记录信息，假定记录信息不超过 5 条；

图 31 借书并记录

图 32 借书记录

（5）若借书时有过期借阅书籍，则需先归还书籍再借阅。

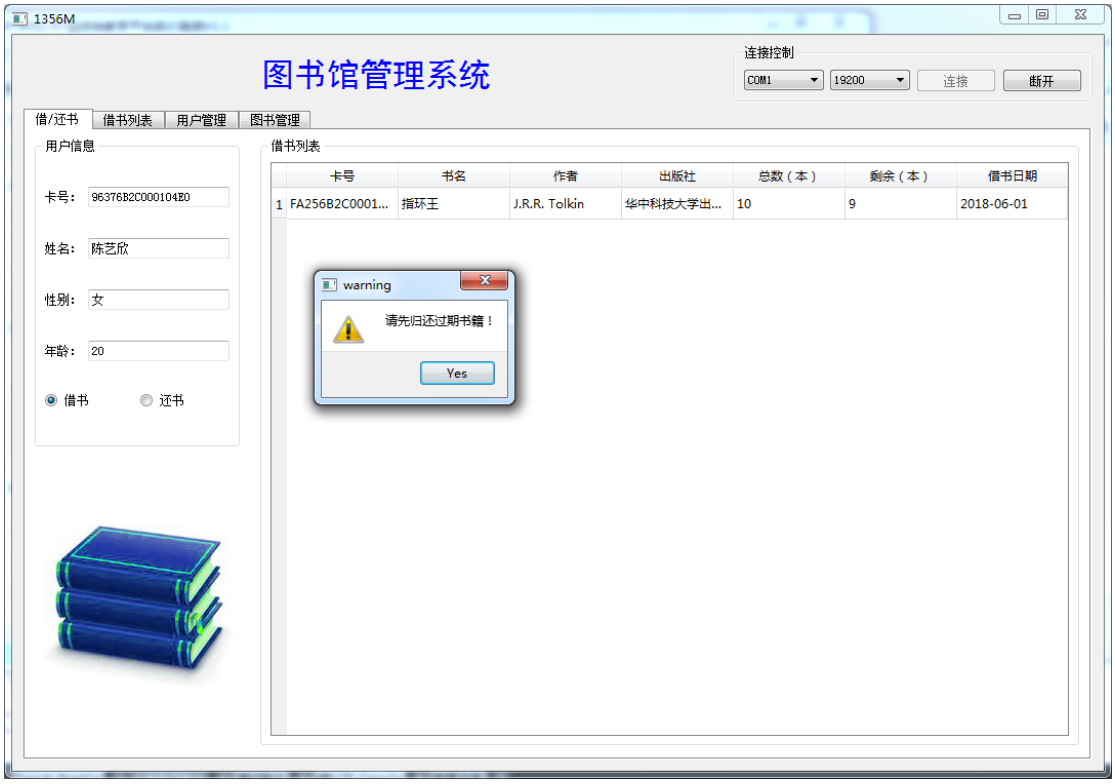

图 33 请先归还过期书籍

图 34 归还书籍后成功借阅

5.4 总结

通过本次综合实验，我了解了 RFID 通信原理以及 ISO15693 通信协议。这次综合实验一开始，我对如何做实验毫无头绪，有两三次实验课都是去了之后对着样例代码做一些小小的改动，进行尝试，但是没什么进展，既不知道自己要做什么，也不知道自己要怎么进行 RFID 通信操作。之后我开始仔细看样例代码，不急着进行改动，而是尽量先把每一行代码都看懂。基本看完之后，我对整个样例程序和自己要做的改进大概有了个想法。

之后就没再遇到什么难题了。在/硬件资料/模块资料目录下，15693 协议的数据传输格式给的很清楚，在卡里写入中文也仅仅是自己添加了一个汉字转 16 进制 ASCII 码的工具函数。由于时间有限和图书馆刷卡借书的操作限制，我并没能把所有工作都写完，比如说如果要在用户卡里写记录，那需要借完书之后再刷一遍用户卡，实际应用起来就会很奇怪，于是我抛弃了这个功能，只添加了写入个人信息的功能。

这次实验我最大的收获是拆解问题的方法。在一开始的摸索阶段，我发现最难的不知学会“怎么做”，而是“要做什么”和“需要哪些资料”的问题。只要一开始不着急动手，先把要做的事和其中不了解的地方想清楚，在写代码的过程中会顺畅很多。只要知道自己要写什么，就算遇到有些不会写的地方，查一下就好了。

5.5 系统源代码

main.cpp

```cpp
#include "widget.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();

    return a.exec();
}
```

borrow_return.cpp

```cpp
#include "borrow_return.h"

//借书界面
Borrow_Return::Borrow_Return(QWidget *parent) : QWidget(parent)
{
    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);
}
//表格显示
void Borrow_Return::ShowTable(QSqlQuery query)
{
    Borrowable=1;
    //设置表格表头
    Table->setHorizontalHeaderLabels(QStringList()<<"卡号"<<"书名"<<"作者"<<"出版社
```

```
"<<"总数（本）"<<"剩余（本）"<<"借书日期");
    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<6; 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()));
        }
        int days=query.value(6).toInt();
        QSqlQuery qry=sql->SelectRecord(Edit_User[0]->text(),query.value(0).toString());
        qry.next();
        Table->setItem(row,6,new QTableWidgetItem(qry.value(2).toString()));
        QDate today=QDate::currentDate();
        QDate BorrowDate=qry.value(2).toDate();
        BorrowDate=BorrowDate.addDays(days);
        qDebug()<<BorrowDate;
        if(BorrowDate<today) Borrowable=0;
        row++;//行数增加
    }while(query.next());
}

//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
    {
        for(int i=0; i < Edit_Count_BORROW_RETURN; i++)
        {
            Edit_User[i]->setText(query.value(i).toString());
        }
        Borrowable=1;
```

```
            //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())
            {
                if(Borrowable==0){
                    QMessageBox::warning(NULL,"warning","请先归还过期书籍！
",QMessageBox::Yes,QMessageBox::Yes);
                    return;
                }
                if(sql->SelectRecord(Edit_User[CardId_User_Borrow]->text(),
query.value(0).toString()).next())
                {
                    return;
                }
                if(query.value(5).toInt() <= 0)
                {
                    return;
                }
                if(sql->InsertRecord(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);


                }
            }
            else
            {
                if(!sql->SelectRecord(Edit_User[CardId_User_Borrow]->text(),
query.value(0).toString()).next())
                {
                    return;
                }
```

```
                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);
                }
            }
            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
}
```

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();
    tool = new Tools();
    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
}

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(uhf, SIGNAL(cycle()), this, SLOT(Get_User_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()


connect(user_manage,SIGNAL(WriteUsrInfo(QString,QString,QString,QString,int
)),this,SLOT(Add_User_Button_Clicked(QString,QString,QString,QString,int)));
```

```
connect(user_manage,SIGNAL(DeleteUserInfo(QString)),this,SLOT(Delete_User
_Button_Clicked(QString)));
}

//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_INV
ENTORY16) != 0xff)//判断是否是读卡命令
    {

if(Dll.RC632_UartCalcFCS(((uint8*)(Info.data()+4)),BUILD_UINT8(Info.at(3),Inf
o.at(2))-1) == Info.at(Info.length() -1))//判断检验和
        {
            QString cardID = data.vdata.replace(" ","");//去掉空格
            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;
            }
        }
    }

}

//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]->curre
ntText().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;
        }
}

void Widget::Add_User_Button_Clicked(QString uid, QString name, QString gen,
QString age, int rcds){
        qDebug()<<"添加用户写卡";

        //写入姓名（0,1）
        QString HexName=tool->ChineseToHex(name);
        uint8 uint8_name[8];
        uint8 name_len=tool->StringToHex(HexName,uint8_name);
        uint8 name1[4], name2[4];
        for(int i=0;i<4;i++){
            name1[i]=uint8_name[i];
            name2[i]=uint8_name[i+4];
        }
        uhf->WriteCard(uid,0,name1);
        uhf->WriteCard(uid,1,name2);
        //写入性别，年龄，记录数（2）
        QString HexGen=tool->ChineseToHex(gen);
        uint8 uint8_gen[2];
        uint8 gen_len=tool->StringToHex(HexGen,uint8_gen);
        uint8 info[4];
        for(int i=0;i<2;i++) info[i]=uint8_gen[i];
        info[2]=age.toInt();
        info[3]=rcds;
        uhf->WriteCard(uid,2,info);

}

void Widget::Delete_User_Button_Clicked(QString uid){
        qDebug()<<"删除用户写卡";
        uint8 data[4];
        for(int i=0;i<4;i++) data[i]=0;
        for(int i=0;i<0x1C;i++) uhf->WriteCard(uid,i,data);
}

Widget::~Widget()
{

}
```

usermanage.cpp

```
#include "usermanage.h"

UserManage::UserManage(QWidget *parent) : QWidget(parent)
{
    QString LabelName[] = {"卡号：", "姓名：", "性别：", "年龄："}; //标签文本
    QString ButtonName[] = {"添加", "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
    uhf = new UHF_Thread();
    tool = new Tools();

    //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));

    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->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[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："};
     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(),Edi
t[Age_User]->text().toInt());
    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);


    //写卡
//      qDebug()<<"写卡";
    emit
WriteUsrInfo(Edit[ID_User]->text(),Edit[Name_User]->text(),Edit[Gender_User]->text(),Edit[
Age_User]->text(),0);

    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(),A
ge);
    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);
    emit DeleteUserInfo(Edit[ID_User]->text());
    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(),E
dit[Age_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[Gender_User]->text(),Ed
it[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");
        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());
}
```

utf_thread.cpp

```
#include "uhf_thread.h"


UHF_Thread::UHF_Thread(QObject *parent) : QThread(parent)
{
    serialport = new QSerialPort();
    Dll = new M1356Dll();
    tool = new Tools();
}
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);
            }
            QThread::msleep(100);
            emit cycle();//循环读取数据
        }
}


bool UHF_Thread::ReadCardID()
{
    uint16 cmd = RC632_CMD_ISO15693_INVENTORY16;
    char *data = (char *)Dll->RC632_SendCmdReq(cmd,NULL,0);//获取读卡指
令
    if(serialport->write(data+2,data[0]))//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;
    }
}

///_写卡
bool UHF_Thread::WriteCard(QString UID, int ADDR, uint8 *DATA){
    uint16 cmd = RC632_CMD_ISO15693_WRITE_SM;//写单个块或多个块命令字，多个块可能不支持
    uint8 Flag,Uid[8],Addr;
```

```
        Flag=02;
        uint8 UID_len = tool->StringToHex(UID,Uid);
        Addr = ADDR;

        uint8 vdata[14];
        uint8 len=0;
        vdata[len]=Flag;
        len++;
        for(uint8 i=0;i<UID_len;i++,len++){
            vdata[len]=Uid[i];
        }
        vdata[len]=Addr;
        len++;
        for(uint8 i=0;i<4;i++,len++){
            vdata[len]=DATA[i];
        }
        qDebug()<<"vdata is";
        for(int i=0;i<14;i++){
            qDebug("%x",vdata[i]);
        }
        char *senddata = (char *)Dll->RC632_SendCmdReq(cmd,vdata,14);//设置为
15693类型
        qDebug()<<"senddata is";
        for(int i=0;i<senddata[0];i++){
            qDebug("%x",senddata[i]);
        }
        if(serialport->write(senddata+2,senddata[0])>0)//data,len
        {
            qDebug("WRITE SUCCESS!");
        }
        else
        {
            qDebug("WRITE FAILED!");
        }
        return true;
}
```

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)
{
//      qDebug()<<space;
    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'));
            qDebug()<<hex;
        }
        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);
//          qDebug()<<temp;
        hex[i/2] = (quint8)temp.toInt(0,16);
//          qDebug()<<hex[i/2];
    }

    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();
        }
}


///汉字转16进制输出
QString Tools::ChineseToHex(QString Str){
    QTextCodec *codec = QTextCodec::codecForName("utf8");
    QString strout;
    QString unidata = codec->toUnicode(Str.toUtf8().data());
    for (int i=0; i<unidata.length(); ++i)
    {
        ushort num = unidata[i].unicode();
        if (num < 255)
            strout += "00";
        strout += QString::number(num,16);
    }
    qDebug()<<"unidata="<<unidata;
    qDebug()<<"strout="<<strout;
    return strout;
}
```

sqlite.cpp

```
#include "sqlite.h"

Sqlite::Sqlite()
{
}
/*连接数据库*/
bool Sqlite::Connect()
{
    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(DATABASE);
    if(!db.open()) return false;
    QSqlQuery query;
    query.exec("create table user_15693 (cardID vchar, name vchar, gender vchar, age int,
primary key (cardID))");
    query.exec("create table books_15693 (booksID vchar, name vchar, author vchar,
publishing_house vchar, count int, residue int, availableTime int, primary key (booksID))");
    query.exec("create table record_15693 (cardID vchar, booksID vchar, brrowDate vdate,
FOREIGN KEY (cardID ) REFERENCES user(cardID), FOREIGN KEY (booksID )
REFERENCES user(booksID))");
```

```
    return true;
}
//打印SQL语句
bool Sqlite::ExecSQL(QString cmd)
{
    QSqlQuery query;
    qDebug()<<cmd.toUtf8().data();
    bool rst=query.exec(cmd);
    if(!rst) qDebug()<<db.lastError();
    return rst;
}
//添加语句
bool Sqlite::Insert(QString table, QString value)
{
    QString cmd = "insert into " + table + " values(" + value + ");";
    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)
{
    return Insert("user_15693", """+cardID+"', '"+name+"', '"+gender+"',
"+QString::number(age));
}
//向books表中添加
bool Sqlite::InsertBooks(QString booksID, QString name, QString author, QString
publishing_house, int count, int residue, int available)
{
    return Insert("books_15693(booksID, name, author, publishing_house, count, residue,
availableTime)", """+booksID+"', '"+name+"', '"+author+"', '"+publishing_house+"',
"+QString::number(count)+", "+QString::number(residue)+", "+QString::number(available));
}
//向record表中添加
bool Sqlite::InsertRecord(QString cardID, QString booksID)
{
    QDate date= QDate::currentDate();
    QString StrDate=date.toString("yyyy-MM-dd");
    return Insert("record_15693(cardID, booksID, brrowDate)", """+cardID+"',
'"+booksID+"','"+StrDate+"""");
}

//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)
{
    return Updata("user_15693","cardID = '"+cardID+"', name = '"+name+"', gender =
'"+gender+"', age = '"+QString::number(age), "cardID = '"+cardID+"'");
}
//修改books表中数据
bool Sqlite::UpdataBooks(QString booksID, QString name, QString author, QString
publishing_house, int count, int residue)
{
    return Updata("books_15693","booksID = '"+booksID+"', name = '"+name+"', author =
'"+author+"', publishing_house = '"+publishing_house+"', count =
'"+QString::number(count)+", residue = '"+QString::number(residue), "booksID =
'"+booksID+"'");
}
//查询user表中数据
QSqlQuery Sqlite::SelectUser(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 Select("user_15693", "*", where);
}
//查询books表中数据
QSqlQuery Sqlite::SelectBooks(QString booksID, QString name, QString author, QString
publishing_house, int count)
{
    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));
        }

        return Select("books_15693", "*", where);
}

//查询record表中的数据
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+"')");
}
```

record.cpp

```
#include "record.h"

//还书界面
Record::Record(QWidget *parent) : QWidget(parent)
{
    QVBoxLayout *MainLayout = new QVBoxLayout();//主布局
    QHBoxLayout *TableLayout = new QHBoxLayout();//表格布局
    QHBoxLayout *ButtonLayout = new QHBoxLayout();//按钮布局
    QHBoxLayout *EditLayout = new QHBoxLayout();//按钮布局
    QVBoxLayout *TopLayout = new QVBoxLayout();//上部布局
    QStringList LabelText,ButtonText;
    ButtonText<<"搜索"<<"删除";
    for(int i=0; i<Button_Count_Record; i++)
    {
        Button[i] = new QPushButton();
        Button[i]->setText(ButtonText.at(i));
        ButtonLayout->addWidget(Button[i]);
    }
    ButtonLayout->addStretch();
    LabelText<<"用户卡号"<<"书籍卡号";
    QString pattern("[A-Fa-f9-0]*");
    QRegExp regExp(pattern);
    for(int i=0; i<Edit_Count_Record; i++)
    {
        Label[i] = new QLabel();
        Label[i]->setText(LabelText.at(i));
        EditLayout->addWidget(Label[i]);
        Edit[i] = new QLineEdit();
        EditLayout->addWidget(Edit[i]);
        Edit[i]->setValidator(new QRegExpValidator(regExp, this));
    }
    //组合框
    QGroupBox *TabGroupBox = new QGroupBox();
    QGroupBox *GroupBox = new QGroupBox();
    sql = new Sqlite();

    Table = new QTableWidget();//表格
    Table->setColumnCount(Table_Column_Record);//设置列数
    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);
```

```
        TopLayout->addLayout(EditLayout);
        GroupBox->setLayout(TopLayout);
        TabGroupBox->setTitle("借书列表");//设置组合框标题
        TabGroupBox->setLayout(TableLayout);//这是组合框布局

        //设置布局
        MainLayout->addWidget(GroupBox);
        MainLayout->addLayout(ButtonLayout);
        MainLayout->addWidget(TabGroupBox);
        this->setLayout(MainLayout);
        SetSlot();
}

void Record::SetSlot()
{

connect(Button[Delete_Record],SIGNAL(clicked()),this,SLOT(delete_record()));//
删除按钮连接槽函数delete_Record()

connect(Button[Select_Record],SIGNAL(clicked()),this,SLOT(select_record()));//
查找按钮连接槽函数select_Record()
        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)
}
//搜索按钮单击事件
void Record::select_record()
{
        QSqlQuery query;
        query =
sql->SelectRecord(Edit[UserID_Record]->text(),Edit[BookID_Record]->text());
        ShowTable(query);//更新表格
        ClearEdit();//清空文本框
}

//删除按钮槽函数
void Record::delete_record()
{
        //删除书籍
        bool ret =
sql->DeleteRecord(Edit[UserID_Record]->text(),Edit[BookID_Record]->text());
        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->SelectRecord());//更新表格
}

//清空文本框
void Record::ClearEdit()
{
        for(int i = 0; i < Edit_Count_Record; 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 Record::get_table_line(int row, int col)
{
        for(int i = 0; i < Edit_Count_Record; i++)
        {
            Edit[i]->setText(Table->item(row,i)->text());
        }
}
//显示表格
void Record::ShowTable(QSqlQuery query)
{
        //表头
        Table->setHorizontalHeaderLabels(QStringList()<<"用户卡号"<<"书籍卡号
");
        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++)
            {
                Table->setItem(row, col, new
QTableWidgetItem(query.value(col).toString()));//显示信息
            }
            row++;
        }while(query.next());
}

//清空文本框和表格
void Record::Clear()
{
        ShowTable(sql->SelectRecord());
}

//设置卡号
void Record::SetCard(QString cardID)
{
        QSqlQuery query = sql->SelectUser(cardID);
        if(query.next())//如果是用户
        {
            Edit[UserID_Record]->setText(cardID);//显示用户卡号
            return;
        }
        query = sql->SelectBooks(cardID);
        if(query.next())//如果是书
        {
            Edit[BookID_Record]->setText(cardID);//显示用户卡号
        }
}
```

booksmanage.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）", "可借时长（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]);
    }
    //设置卡号格式
    QString pattern("[A-Fa-f9-0]*");
    QRegExp regExp(pattern);
    Edit[ID_Books]->setValidator(new QRegExpValidator(regExp, this));

    //设置总数/剩余/可借时长格式
    pattern="[9-0]{3}";
    regExp.setPattern(pattern);
    Edit[Count_Books]->setValidator(new QRegExpValidator(regExp, this));
    Edit[Residue_Books]->setValidator(new QRegExpValidator(regExp, this));
    Edit[Available_Books]->setValidator(new QRegExpValidator(regExp, this));

    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
    }
    ButtonLayout->addStretch(0);
    ButtonLayout->setSpacing(20);

    Table = new QTableWidget();
    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(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）", "可借时长（天）"};
    for(int i = 0; i < Edit_Count_BOOKS; i++)
    {
        if(i==Residue_Books) continue;//剩余数量可以不填
        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().to
Int(),residue,Edit[Available_Books]->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().t
oInt(), Edit[Residue_Books]->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().to
Int());
        ShowTable(query);//更新表格
        ClearEdit();//清空文本框
}

//显示表格
void BooksManage::ShowTable(QSqlQuery query)
{
        //设置表头
        Table->setHorizontalHeaderLabels(QStringList()<<"卡号"<<"书名"<<"作者
"<<"出版社"<<"总计（本）"<<"剩余（本）"<<"可借时长（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());
}

//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());
}
```