

目录

第一章 项目介绍	1
第二章 项目设计	1
2.1. 界面设计	1
2.1.1. 主界面设计	1
2.1.2. 子界面设计	2
2.2. 功能设计	3
2.2.1. 主界面功能实现	3
2.2.2. 子界面功能实现	5
1) 读写功能	5
2) 钱包功能	6
3) 云端交互*	7
4) 消费计时*	8
第三章 功能测试	9
3.1. 基础功能测试	9
3.2. 创新功能测试	9

**部分为创新功能*

第一章 项目介绍

1.1. 项目背景

当前，RFID 的使用场景日益广泛，在 RFID 广泛应用的同时对于 RFID 读卡器的需求日益明显。RFID 芯片通常只是作为数据存储的载体，需要通过 RFID 读卡器进行数据的交互，否则 RFID 芯片中的数据只是一堆无用的 0,1，所以对于 RFID 读卡器的设计对于 RFID 的技术来说十分重要。RFID 读卡器的设计直接决定了这个 RFID 的芯片具有的功能，是 RFID 项目开发中的重要一环，通过不同的读卡器程序编写，可以使得同一款 RFID 芯片适用于不同的场景。为了做到不同的场景适应，需要编写的上位机程序针对 RFID 读卡器所开放的不通接口进行组合设计，以实现设计要求的功能。

1.2. 项目主题

基于 MFC 设计 RFID 读卡器的上位机软件。

1.3. 项目简介

本项目通过 RFID 读卡器提供的接口，设计 MFC 的上位机软件，主要实现的是对 RFID 芯片卡的信息读取和芯片中存储的数据进行读写，以及与读写工作相关的信息记录与数据云端同步。

项目主要通过 Visual Studio 进行实现，其中使用到了读卡器提供的接口，同时使用到了云端服务器用作数据同步和额外的信息读取。

项目优势：①简单易用，界面功能清晰，功能分布合理。②安全性，针对芯片中数据容易被人为修改的问题，提供云端同步功能，实现对于对于数据被人为修改的校验。③通用性，针对于 RFID 卡中可以保存的一些无需修改的信息，将其存储于云端中，减少对于卡片存储空间的使用，如需要进行功能定制可以直接通过定制数据库的方式实现 RFID 的信息存储功能。如发生卡片丢失，只需要将云端数据中对于的 RFID 卡的 id 进行修改，且其中的信息数据不会丢失。

第二章 项目设计

2.1. 界面设计

2.1.1. 主界面设计

首先新建一个基于对话框的 MFC 工程，删除默认按钮，添加固定功能部分，主要是使用 Edit Control 和 Button，实现卡片 Uid 的读取以及读写部分必要的密钥和区块选择。因为子界面实现的是多页的效果，使用工具箱中的 Tab Control 控件，主界面最终效果如下图-1

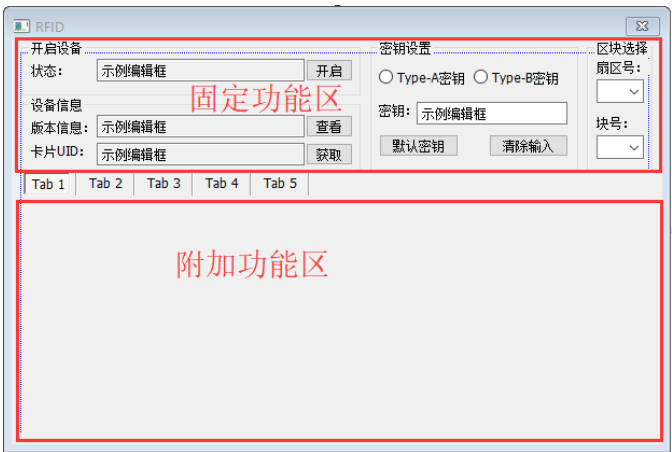


图-1 主界面效果图

2.1.2. 子界面设计

子界面部分主要是功能实现，分为 4 个主要的功能区：读写操作，钱包操作，云端信息获取与同步，消费计时，每个子界面功能区都是一个独立的 dialog。

1) 读写操作区

实现功能：读写 RFID 芯片卡中的数据。设计依据芯片中每个扇区有 4 个块通过选择扇区和块的方式进行读写，所以只需要 4 个输入框和对应的按钮即可，效果如下图 2-1。



图 2-1 读写操作区

2) 钱包操作区

实现功能：RFID 卡中存储金额的最基础功能，根据选择的扇区和块调用读写账户的函数进行实现，同时使用文件操作记录所有的充值与消费记录。此时需要用于输入金额的操作区和历史记录区用于展示历史记录，效果如下图 2-2。



图 2-2 钱包操作区

3) 云端交互区

实现功能：从云端读写卡片对应的用户信息，同时实现数据的同步以及检查。通过编辑框实现数据的读取和展示，界面设计以编辑框和对应的功能按键为主。效果如下图 2-3

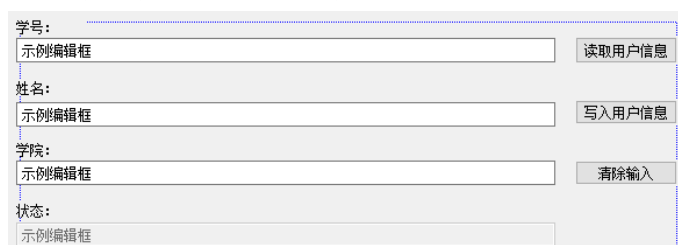


图 2-3 云端交互区

4) 消费计时区

实现功能：实现 RFID 的计时收费功能可用于需要计时收费的场景。主要也是通过输入控件和按钮实现，同时带钱包功能相同的历史记录区。效果如下图 2-4




图 2-4 消费计时区

2.2. 功能设计

2.2.1. 主界面功能实现

主界面主要实现的为固定功能，主要为读取读卡器版本信息，读取卡片 UID，以及读写操作相关的选项选择区。同时主界面通过 tab control 控件实现子界面间的切换。

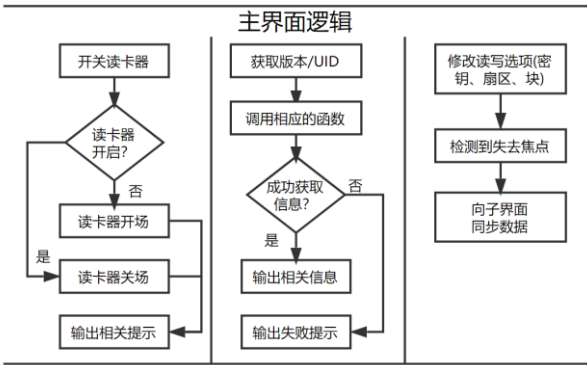


图-3 主界面逻辑

1) 读取读卡器版本信息

此功能直接使用读卡器提供的接口函数 Reader_Version 实现，直接调用即可。

同时引入报错提示功能，将所有被报错情况封装，根据报错码获取相应的错误信息，然后用过 Edit Control 进行展示，错误信息类如下图 4

```
class Hits
{
public:
    Hits() {
        error_hit[0] = "IFD_OK 执行成功";
        error_hit[1] = "IFD_ICC_TypeError 卡片类型不对";
        error_hit[2] = "IFD_ICC_NoExist 无卡";
        error_hit[3] = "IFD_ICC_NoPower 有卡未上电";
        error_hit[4] = "IFD_ICC_NoResponse 卡片无应答";
        error_hit[5] = "IFD_ICC_BCCError BCC校验错误";
        error_hit[6] = "IFD_ICC_TimeOut 接收超时";
        error_hit[7] = "IFD_ICC_RunFail 执行失败";
        error_hit[8] = "IFD_ICC_SiteFail 卡片位置错误";
        error_hit[9] = "IFD_ICC_SetFail 设置失败";
        error_hit[10] = "IFD_NoSlot 无卡";
        error_hit[11] = "IFD_ConnectError 读卡器连接错";
        error_hit[12] = "IFD_UnConnected 未建立连接(没有执行打开设备函数)";
        error_hit[13] = "IFD_BadCommand (动态库)不支持该命令";
        error_hit[14] = "IFD_CheckSumError 信息校验和出错";
        error_hit[15] = "IFD_ICC_PowerFail 卡片上电失败";
        error_hit[16] = "IFD_ICC_ResetFail 卡片复位失败";
        error_hit[17] = "IFD_ICC_PowerOffFail 卡片下电失败";
    };
    string error_hit[18];
};
```

图-4 错误信息类

通过调用接口函数后返回值在状态 Edit Control 中显示提示。

2) 读取卡片 UID

此功能同样使用接口函数进行实现，此处就是设计读取信息后 unsigned char 的转化，此处的转换限制要求只能每次转换一个 unsigned char，否则会造成被认为是 2 个 unsigned char 组成的中文的情况。

```
void CRFIDDlg::OnBnClickedButtonUid()
{
    // TODO: 在此添加控件通知处理程序代码
    Hits hits;
    unsigned char uid[50];
    int uid_len;
    int status = 0 - find_14443(uid, &uid_len);
    if (status) { //判断是否读取成功
        CString hits(hits.error_hit[status].c_str());
        meUid.SetWindowText(hits);
    }
    else {
        CString Uid, tmp;
        for (int i = 0; i < uid_len; i++) { //转换
            tmp.Format(_T("%02X"), uid[i]);
            Uid += tmp;
        }
        meUid.SetWindowText(Uid);
        synchronous(); //同步功能, 向子界面同步UID
    }
}
```

图-5 读取 UID

3) 读写操作相关选项区

此区域主要用于读写相关选项,主要为密钥和扇区及块的选择,默认密钥类型为B值为FFFFFFFFFFFF,当使用默认密钥按钮时将对应选项设置为上述值。当修改了相关内容之后需要将密钥选择同步到各个子界面中,选择扇区和块的操作也需要同步,同步触发条件使用事件失去焦点(KILLFOCUS),即当改控件失去输入焦点时。具体操作流程如下图-6 代码部分不再展示。

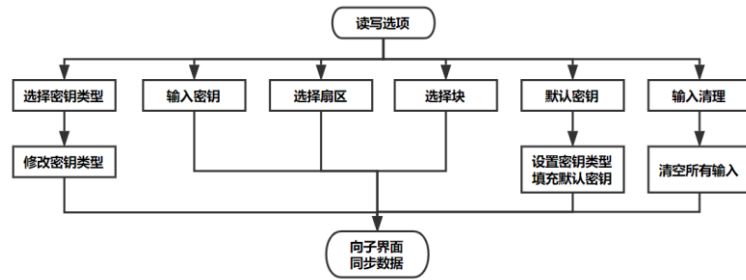


图-6 读写选项功能流程图

4) Tab Control 实现子界面功能切换

- i. 首先在主界面的头文件中添加所有子界面的类的头文件,并在主界面类中声明。

```
#include "CRnw.h"
#include "CMoney.h"
#include "CHit.h"
#include "CUserInfo.h"
#include "CTimeMeter.h"

class CRFIDDlg : public CDialog{
public:
    CTabCtrl m_tabs;
    CRnw m_rnw;
    CMoney m_money;
    CUserInfo m_user;
    CTimeMeter m_time;
};
```

图-7 添加头文件和定义变量

- ii. 给主页面的 Tab Control 控件添加选项,并创建子窗口

```
//添加并创建子窗口
m_tabs.InsertItem(0, _T("读写操作"));
m_tabs.InsertItem(1, _T("电子钱包"));
m_tabs.InsertItem(2, _T("用户信息"));
m_tabs.InsertItem(3, _T("计时系统"));
m_rnw.Create(IDD_CRnw, GetDlgItem(IDD_RFID_DIALOG));
m_money.Create(IDD_CMoney, GetDlgItem(IDD_RFID_DIALOG));
m_user.Create(IDD_CUserInfo, GetDlgItem(IDD_RFID_DIALOG));
m_time.Create(IDD_CTimeMeter, GetDlgItem(IDD_RFID_DIALOG));
```

图-8 添加并创建子窗口

- iii. 控制子界面范围,并设置显示(设置默认显示第0项),此处需要将子界面属性修改为 child

```
//设置子窗口位置
CRect rc;
m_tabs.GetClientRect(rc);
rc.top += 145;
rc.right += 9;
rc.bottom += 125;
rc.left += 9;
//设置显示
m_rnw.MoveWindow(&rc);
m_rnw.ShowWindow(true);
m_money.MoveWindow(&rc);
m_money.ShowWindow(false);
m_user.MoveWindow(&rc);
m_user.ShowWindow(false);
m_time.MoveWindow(&rc);
m_time.ShowWindow(false);
m_tabs.SetCurSel(0);
```

图-9 控制子界面范围,并设置显示

5) 同步功能

此功能目的为为子界面同步数据,子界面中需要数据读写的部分都需要获取主界面提供的读写参数,在每次参数修改后触发失去焦点(KILLFOCUS)时间,然后调用同步功能进行同步。

```
void CRFIDDlg::synchronous()
{
    CString key, sen, blk, uid;
    meKey.GetWindowText(key);
    mcbSen.GetWindowText(sen);
    mcbBlock.GetWindowText(blk);
    meUid.GetWindowText(uid);
    m_rnw.blk = blk;
    m_rnw.sen = sen;
    m_rnw.key_type = key_type;
    m_rnw.key = key;
}
```

图-10 数据同步（部分）

2.2.2. 子界面功能实现

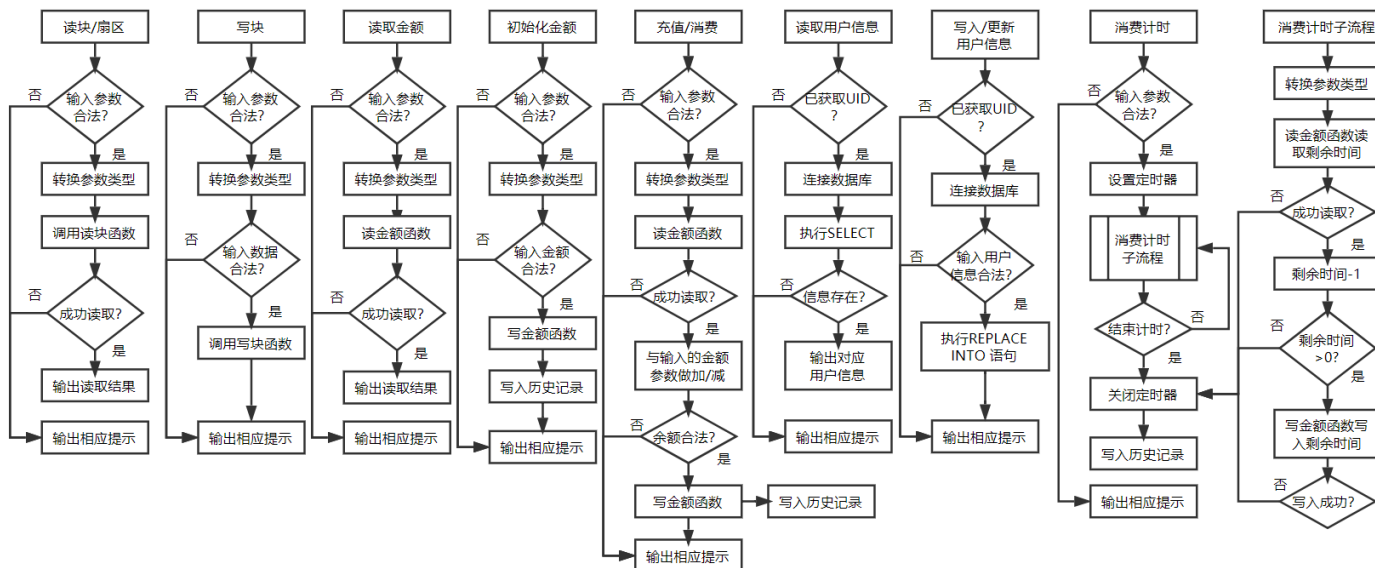


图-11 子界面功能流程图

子界面中的消费计时中的时间初始化与充值与金额的类似故不在展示

1) 读写功能

读写界面的类中包含 4 个参数 blk,sen,key_type,key，分别对应块，扇区，密钥类型，密钥的数据，此数据从主界面获取。

i. 读块功能（读扇区功能省略，度扇区功能实际上就是 4 次读块功能）

读块功能调用 `int read_block`，我们只需要提供扇区、块、密钥、密钥类型参数，所以在调用该函数前我们需要先检测这 4 个参数是否合法。

```
if (key_type == 0) {
    meStatus.SetWindowText(_T("请选择密钥类型"));
    return;
}
```

图-12 参数合法性检验（部分）

之后我们需要将密钥转换成参数需要的 `unsigned char` 类型，同时扇区和块的参数也要转换成 `int` 类型。

```
//参数类型转换
string keys = CStringA(key); //密钥类型转换
transform(keys.begin(), keys.end(), keys.begin(), toupper);
unsigned char keyc[6];
for (int i = 0; i < 6; i++) //一次转换两个字符
    keyc[i] = (keys[i * 2] >= 'A' ? (keys[i * 2] - 'A' + 10) : (keys[i * 2] - '0')) * 16 + (keys[i * 2 + 1] >= 'A' ? (keys[i * 2 + 1] - 'A' + 10) : (keys[i * 2 + 1] - '0'));
int sens = strtol(CStringA(sen), NULL, 16); //扇区转换
int blks = strtol(CStringA(blk), NULL, 16); //块转换
```

图-13 密钥及参数类型转换

然后调用 `read_block` 实现读取，此处还是需要类似读取 UID 的时候的数据转换处理，同时需要对块 3 单独处理，块 3 需要分开填入三个 Edit Control，其他块直接填入即可

```

unsigned char data[100];
int len;
int status = 0 - read_block(sens, blks, key_type, keyc, data, &len);
if (status) { //判断是否读取成功
    CString hits(hit.error_hit[status].c_str());
    meStatus.SetWindowText(hits);
    return;
}
else {
    CString Data, tmp;
    if (blks == 3) { //块3单独处理
        CString Data0, Data1, Data2, tmp;
        for (int i = 0; i < 6; i++) {
            tmp.Format(_T("%02X"), data[i]);
            Data0 += tmp;
        }
        for (int i = 6; i < 10; i++) {
            tmp.Format(_T("%02X"), data[i]);
            Data1 += tmp;
        }
        for (int i = 10; i < 16; i++) {
            tmp.Format(_T("%02X"), data[i]);
            Data2 += tmp;
        }
        meBlock3_1.SetWindowText(Data0);
        meBlock3_2.SetWindowText(Data1);
        meBlock3_3.SetWindowText(Data2);
        meStatus.SetWindowText(_T("读取成功"));
    }
    else {
        for (int i = 0; i < len; i++) {
            tmp.Format(_T("%02X"), data[i]);
            Data += tmp;
        }
        if (blks == 0) meBlock0.SetWindowText(Data);
        if (blks == 1) meBlock1.SetWindowText(Data);
        if (blks == 2) meBlock2.SetWindowText(Data);
        meStatus.SetWindowText(_T("读取成功"));
    }
}
}

```

图-14 读块并展示内容

ii. 写块功能

写块功能首先也需要检验参数合法性，同时需要检测输入合法性，在块 3 中还要检测数据长度合法性。此部分检验跳过，做法与读块类似。第二步是对参数的处理，此步骤除对输入数据的处理与读块不同之外全部相同，对于输入数据的转换先准备一个数据容器用于存放转换后的数据，初始全部置 0，然后根据输入数据进行反向转换填充，也就是从最低位开始转换，同时注意奇数个时需要单独转换一次。转换完毕之后调用 `wirte_block`，之后的提示部分略

```

transform(datas.begin(), datas.end(), datas.begin(), toupper);
unsigned char Data[16] = { 0 }; //设置数据长度默认全部为0
for (int i = 0; i < 16 && i * 2 < len; i++) { //反向转换 2个一组进行转换 单独处理最后奇数个数据
    if (i * 2 + 1 == len) Data[15 - i] = (datas[len - i * 2 - 1] >= 'A' ? (datas[len - i * 2 - 1] - 'A' + 10) :
(datas[len - i * 2 - 1] - '0'));
    else Data[15 - i] = (datas[len - i * 2 - 2] >= 'A' ? (datas[len - i * 2 - 2] - 'A' + 10) : (datas[len - i * 2 - 2] -
'0')) * 16 + (datas[len - i * 2 - 1] >= 'A' ? (datas[len - i * 2 - 1] - 'A' + 10) : (datas[len - i * 2 - 1] - '0'));
}
int status = 0 - write_block(blks, sens, key_type, keyc, Data, 16);

```

图-15 输入数据转换

2) 钱包功能

i. 查询余额功能

此处实现方式调用 `read_account` 实现，其要求的输入参数与读块一致，所以此处的合法性判断也一致，同时输入参数的转换也是一致的，即不在展示，此处展示调用 `read_account` 读取金额信息

```

//读取余额
LONG money;
int status = 0 - read_account(sens, blks, key_type, keyc, &money);
if (status) {
    CString hits(hit.error_hit[status].c_str());
    meStatus.SetWindowText(hits);
}
else {
    CString Money;
    Money.Format(_T("%ld"), money);
    meMoney.SetWindowText(Money);
}

```

图-16 读取余额

ii. 初始化、充值、消费功能

初始化实际上就是直接写入，过程和写块类似，充值与消费需要先读出余额然后进行相应的操作在写入。此处的参数合法性判断和参数转换与之前的操作类似不在展示，此功能只需要读入输入的金额然后进行转换并调用 `wirte_account` 写入即可，之后的信息提示部分略。

```
CString tmp;
meMoney.GetWindowText(tmp); //读取输入金额
LONG money(_ttoi(tmp));
int status = 0 - write_account(sens, blks, key_type, keyc, money); //写入
```

图-17 余额初始化

充值与消费功能都需要先读出余额在进行操作（此步骤即为读取余额），然后将修改后的余额写回，消费的时候需要额外注意此时消费之后余额会不会小于 0，如小于 0 则消费失败，而充值的时候没有这个要求，所以此处以消费为例

```
LONG money; //读取余额
int status = 0 - read_account(sens, blks, key_type, keyc, &money);
if (status) {
    CString hits(hit_error_hit[status].c_str());
    meStatus.SetWindowText(hits);
    return;
}
else {
    CString Money;
    meSub.GetWindowText(Money);
    LONG tmp(_ttoi(Money));
    money -= tmp; //消费，充值时此处为+
    if (money < 0) {
        meStatus.SetWindowText(_T("余额不足"));
        return;
    }
    status = 0 - write_account(sens, blks, key_type, keyc, money); //写入
```

图-18 消费操作

iii. 文件读写操作

在进行初始或者充值消费的时候需要写入历史记录，在写入历史记录时加入时间戳，此处直接使用 CFILE 进行文件的读写操作（CFile.open 中的参数是为了打开并保留文件中内容），此处使用初始化时展示文件写入操作。

```
CFile file;
file.Open(_T("history.txt"), CFile::modeCreate | CFile::modeWrite | CFile::modeNoTruncate); //打开文件
file.SeekToEnd(); //移到末尾
CString str;
CTime t;
t = CTime::GetCurrentTime(); //获取时间
str = t.Format("%Y/%m/%d %X");
str += _T(" UID:") + uid; //设置输入信息
str += _T(" Init:") + tmp + _T("\r\n");
file.Write(CStringA(str), strlen(CStringA(str))); //写入
file.Close(); //关闭
mHistory.InsertString(mHistory.GetCount(), str); //添加到历史记录中
```

图-19 文件写入

文件在加载历史记录时读入，通过 CStdioFile 实现分行读取文件，此处不使用 CFile 主要是是 CFile 需要手动分行显示，使用 CStdioFile 可以直接按行读入。

```
void CMoney::OnBnClickedButtonHisLoad()
{
    // TODO: 在此添加控件通知处理程序代码
    CStdioFile file;
    file.Open(_T("history.txt"), CFile::modeRead | CFile::typeText);
    CString his;
    while (file.ReadString(his)) { //分行读取
        mHistory.AddString(his); //展示
    }
}
```

图-20 文件读入

清除历史记录直接使用 file.Open(_T("history.txt"), CFile::modeCreate | CFile::modeWrite); 即覆盖原有的历史记录文件。

3) 云端交互*

此部分主要是将记录写入云服务中的数据库中，同时可以根据卡片 uid 索引存储在云服务器中的用户信息等。数据库的部署，使用 LAMP 环境，使用 ODBC 进行连接，ODBC 驱动使用 Mysql 官方安装包(32 位)，之后在电脑端控制面板-系统和安全-管理工具-ODBC 数据源(32 位)中配置 Mysql。同时在 stdafx.h 添加 #include <odbcinst.h> #include "afxdb.h" 即可。具体配置方式参见 [ODBC 连接配置](#)。

使用数据库

a) 打开数据库，直接使用内置 CDatabase，CDatabase.Open 中对应的参数为之前本地配置的连接参数。

```
CDatabase db;
db.Open(NULL, FALSE, FALSE, _T("ODBC;DSN=RFID;UID=root;PWD=root"));
if (!db.IsOpen()) {
    MessageBox(_T("连接失败"));
    db.Close();
    return;
}
```


图-21 打开数据库

- b) 使用数据库索引信息，此处的信息索引主要是根据卡片 UID 去数据库中搜索对应的用户信息，所以需要判断 UID 是否已经获取

```
CRecordset rs(&db);
CString str;
if (!cid.GetLength()) { //判断是否获取到卡片UID
    mStatus.SetWindowText(_T("请获取卡片UID"));
    return;
}
//构建查询语句，此处为了区别userid 使用cid代替卡片UID
str.Format(_T("SELECT * FROM Users WHERE cid = '%s'"),cid);
rs.Open(CRecordset::ForwardOnly, str); //接收结果
if (rs.IsEOF()) { //没有查询到任何信息
    mStatus.SetWindowText(_T("用户信息不存在"));
}
else { //查询成功，展示信息，此处略
}
rs.Close();
db.Close();
return;
```

图-22 查询用户信息

- c) 增改数据库，此处主要是针对用户信息进行添加或者修改，不需要进行删除，所以直接使用 REPLACE INTO 语句进行覆盖插入，此处也需要判断输入是否合法，此部分判断不在展示。

```
str.Format(_T("REPLACE INTO Users(cid, uid, uname, college) VALUES('%s', '%s', '%s', '%s')"), cid,id,name,college);
try{
    db.ExecuteSQL(str); //执行语句
}
catch (CDBException *e){
    //异常处理省略
}
```

图-23 插入或修改用户信息

- d) 数据库会执行对于每一次修改金额等操作的同步，卡上每个扇区的每个块的金额部分初始化之后会自动插入将信息插入数据库，在修改之后也会同步修改数据库，历史记录同时也会保存在数据库中，当数据库中数据和实际卡中数据不匹配时，就可以确定此卡片被人为修改过。

4) 消费计时*

消费计时功能，用于按时计费的场景，如使用机房或其他学校设施。设计方式还是通过 read_account 和 write_account 实现，通过定时函数 SetTimer(1, 1000,NULL)每秒钟写入一次数据做到实时计时功能。此功能的时间初始化和增加使用时间，等效于将时间转为 account 中数据，然后进行初始化或者增加，此处不在展示。此处展示计时部分。

```
void CTimeMeter::OnBnClickedButtonUse()
{
    // TODO: 在此添加控件通知处理程序代码
    if (start) {
        start = false;
        mStatus.SetWindowText(_T("开启使用"));
        KillTimer(1);
        //写入历史记录部分省略
        cnt = 0;
    }
    else {
        //参数合法性判断省略
        start = true;
        mStatus.SetWindowText(_T("停止使用"));
        SetTimer(1, 1000,NULL);
    }
}
```

图-24 计时开始与结束

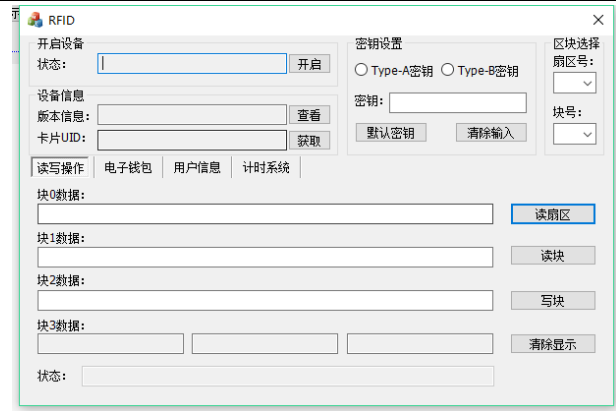

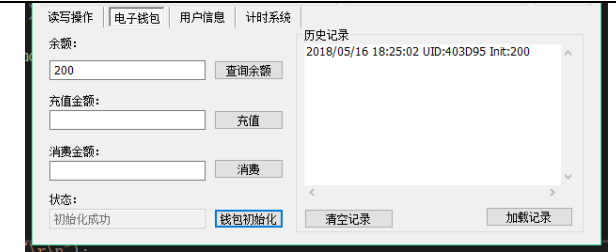
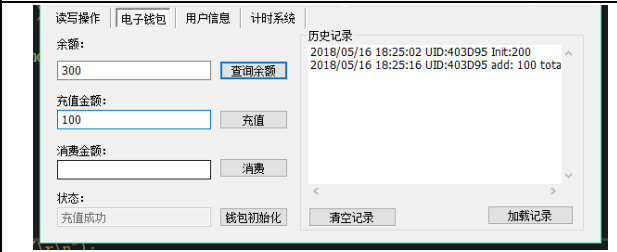
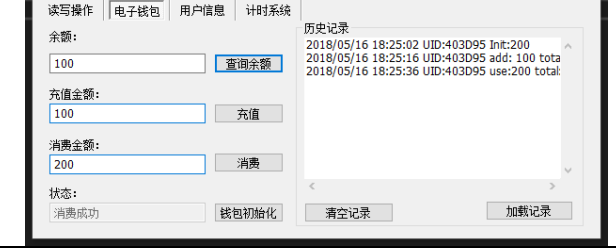
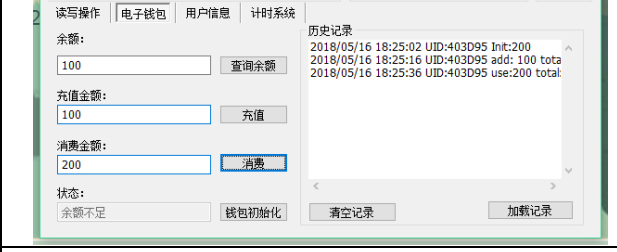
在消息响应函数中找到 WM_TIMER,然后添加响应函数 OnTimer(), 实现定时器定时执行部分。

```
void CTimeMeter::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    CDialog::OnTimer(nIDEvent);
    if (nIDEvent == 1) {
        //参数合法性 略
        //参数转换 略
        LONG times;
        int status = 0 - read_account(sens, blks, key_type, keyc, &times); //读取时间
        if (status) { //读取失败提示略
            OnBnClickedButtonUse(); //再次开关函数调以关闭
            return;
        }
        if (times-1) { //余时不足
            OnBnClickedButtonUse(); //再次开关函数调以关闭
            return;
        }
        status = 0 - write_account(sens, blks, key_type, keyc, times-1);
        //写入成功与更新界面 略
        cnt++;
    }
}
```





图-25 计时执行部分

第三章 功能测试

3.1. 基础功能测试

	
开启时界面	获取卡片 UID 同时读出卡中一个扇区数据
	
钱包初始化	钱包充值
	
钱包消费	余额不足情况

3.2. 创新功能测试

	
在当前卡不在数据库中时读取卡信息	写入用户信息后数据库查询结果
	

	
写入用户信息与对应数据库查询结果	更新用户信息与对应数据库查询结果
	
计时初始化	时间充值
	
使用计时	金额初始化和使用后同步到数据库