商店管理系统 V1.0

开发者手册

项目简述

我实现的项目是面向实体店的商店管理系统，其中售货员需要登记销售商品、收银、找零的功能，管理员需要管理商品、统计销售、统计库存、统计收支的功能。

结合上课老师所讲，我把这些每一个操作都做成了独立的模块，以实现充分的模组化，方便后续对功能的补充。

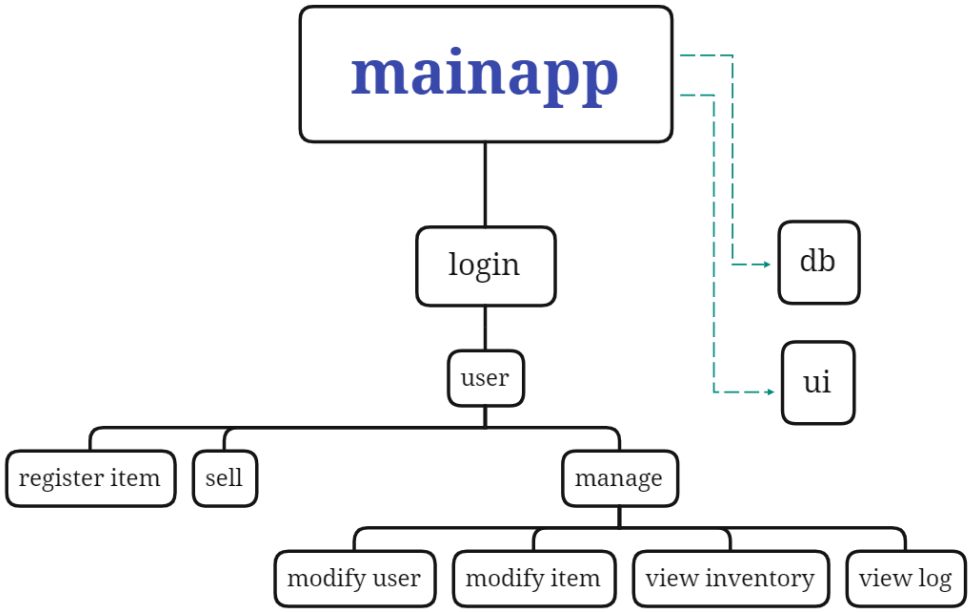
除此之外，我还实现了图形界面和应用了数据库，并按照程序需要，把写的函数封装了一下，以便各个模块调用。其中，图形界面使用了 EasyGraphicEngine 实现，数据库基于 SQLite 3.0。

Root 用户：

用户名:username

密码:password

模块结构图



过程性模块基本编写思路

综述

```
class logIn
{
    public:
        userInterface loginWindow;
        sys_edit userName,passWord;
        dataBase userDataBase;
        char userNameStr[100],passWordStr[100];
        void run();
        void processCmd(int);
        void goBack();
        void init();
        int checkPassWord(const char*,const char*);
        int checkAdmin(const char*);
    private:
};
```

以登录(login)模块为例。每个模块都含有四个主要的部分：初始化(init)、运行(run)、处理指令(processCmd)、返回(goBack)。

模块的调用

```
logIn runLogin;
runLogin.run();
```

在父模块中，声明一个子模块，然后运行子模块的运动(run)函数，即可完成调用。

初始化(init)

```
void logIn::init()
{
    userDataBase.openDatabase("users.db");
    strcpy(loginWindow.imageSrc,"./bg/login");
    loginWindow.init();
    loginWindow.buttonOffsetY=130;
    loginWindow.drawButtons(argumentCount,"Back","Go");
    loginWindow.drawEditBox(&userName,
                            loginWindow.windowWidth/2-100,
                            loginWindow.windowHeight/2-30-5,
                            200,
                            30,
                            "");
    loginWindow.drawEditBox(&passWord,
                            loginWindow.windowWidth/2-100,
                            loginWindow.windowHeight/2+5,
                            200,
                            30,
                            "");
}
```

该函数的目的是初始化整个模块，其中包括初始化模块的各个部分，如初始化 ui、打开数据库等等。这里包含对该图形界面上按钮、输入框等元素的绘制。

运行(run)

```
void logIn::run()
{
    init();
    int cmd=loginWindow.getCmd(argumentCount);
    processCmd(cmd);
}
```

该函数对于所有的过程性模块都是一样的，是供上级模块调用的。首先调用自身的初始化模块，然后获取一个指令（等待按下一个按钮），并进行相应的操作。

处理指令(processCmd)

```
void logIn::processCmd(int cmd)
{
    switch(cmd)
    {
        case 0:
        {
            goBack();
            break;
        }
```

```
  case 1:
  {
      userName.gettext(100,userNameStr);
      passWord.gettext(100,passWordStr);
      userName.destroy();
      passWord.destroy();

      int loginSuccess=checkPassWord(userNameStr,passWordStr);
      int isAdmin=checkAdmin(userNameStr);

      if(LOGIN_SUCCESS==loginSuccess)
      {
          User runUser;
          strcpy(runUser.userName,userNameStr);
          if(LOGIN_ISADMIN==isAdmin)
              runUser.admin=1;
          loginWindow.createPopup(("Hello, "+(std::string)userNameStr).c_str(),"Welcome");
          runUser.run();
      }
      else
      {
          loginWindow.createPopup("Login Failed.","OK");
      }
      run();
      break;
  }
```

此处分别对应两种指令；第一种是返回指令，按下后调用返回(goBack)函数，跳出这个模块，返回上一个模块。第二种是一般指令，按下后会执行相应的操作，然后重新调用自身的运行(run)函数，从而实现循环的效果，继续等待指令。

返回(goBack)

```
void logIn::goBack()
{
    userName.destroy();
    passWord.destroy();
    userDataBase.closeDatabase();
    loginWindow.close();
}
```

该函数中的内容主要是进行一些收尾工作，如对对象进行销毁，以及关闭图形界面、关闭数据库等。

各个模块的具体实现

各模块大体的框架如上面所述，而每个模块的具体功能的具体实现，烦请参考各模块源码中的注释。

功能性模块详解

由于功能性模块需要被过程性模块调用，因此对此部分进行详细解释，阐述清楚每个函数的使用方法。

图形界面(ui)

图形用户界面(userInterface)类

```
class userInterface//图形界面
{
    public:

        int windowWidth=1280;//窗口宽度
        int windowHeight=720;//窗口高度

        int buttonOffsetX=0;//按钮X偏移
        int buttonOffsetY=0;//按钮Y偏移

        PIMAGE backGround=newimage();//背景

        char imageSrc[100]="./bg/main";//背景路径

        void init();//初始化

        void drawButtons(int,...);//绘制按钮

        void drawEditBox(sys_edit*,int,int,int,int,const char*);//绘制输入框

        void close();//关闭

        int getCmd(int);//获取指令

        void createPopup(const char*,const char*);//绘制弹窗

    private:

};
```

该类是主要的图形界面类，内含对图形界面所有的控制。

弹窗(popUp)类

```
class popUp//弹出窗口
{
    public:
    int popupWidth=320;//弹窗宽度
    int popupHeight=240;//弹窗高度

    int popupOffsetX=0;//弹窗位置X偏移
    int popupOffsetY=0;//弹窗位置Y偏移

    void drawPopup(userInterface*);//绘制弹窗

    void getClick(userInterface*);//获取鼠标点击动作

    const char* content;//内容信息

    const char* buttonText;//按钮信息

};
```

该类是图形界面的一个子模块，目的是在现有窗口上再绘制一层小窗口，来达到提示效果。

初始化(init)函数

```
void userInterface::init()//初始化
{
    printf("initializing ui\n");
    initgraph(windowWidth,windowHeight,0);
    setcaption("Sale Manager V1.0 by w43322@NEU");
    getimage(backGround,imageSrc);
    putimage(0,0,backGround);
}
```

在声明一个新 ui 变量时必须先运行，用于初始化。

参数：无

返回值：无

关闭图形界面(close)函数

```
void userInterface::close()//关闭图形界面
{
    delimage(backGround);
}
```

在停止使用一个用户界面是必须调用，以释放内存。

参数：无

返回值：无

绘制按钮(drawButtons)函数

```
void userInterface::drawButtons(int argc,...)//绘制按钮
{
    char *buttonName[100];
    int midHeight=windowHeight/2;
    int midWidth=windowWidth/2;
    int buttonHeight=60;
    int buttonWidth=200;
    int buttonInterval=20;
    int buttonTop;
    int buttonLeft=midWidth-buttonWidth/2;
    va_list argv;
    va_start(argv,argc);
    for(int i=0;i<argc;i++)
        buttonName[i]=va_arg(argv,char*);
    va_end(argv);
    if(argc%2==0)//if n==2,4,6,8...
    {
```

用于绘制可交互的按钮。

参数：按钮数量

返回值：无

获取指令(getCmd)函数

```
int userInterface::getCmd(int argc)//获取指令
{
    int midHeight=windowHeight/2;
    int midWidth=windowWidth/2;
    int buttonHeight=60;
    int buttonWidth=200;
    int buttonInterval=20;
    int buttonTop;
    if(argc%2==0)//if n==2,4,6,8...
    {
        buttonTop=midHeight-buttonInterval/2
                  -buttonHeight*(argc/2)-buttonInterval*(argc/2-1);
    }
    else//if n==1,3,5,7,9...
    {
        buttonTop=midHeight-buttonHeight/2
                  -buttonHeight*(argc/2)-buttonInterval*(argc/2);
    }
    for(;is_run();delay_fps(60))
```

用于得知鼠标点击了哪个按钮。

参数：按钮数量

输出值：无

绘制输入框(drawEditBox)函数

```
void userInterface::drawEditBox(sys_edit* editBox,int x,int y,
                    int width,int height,const char* setText)
//绘制输入框
{
    editBox->create(false);
    editBox->size(width,height+8);
    editBox->setbgcolor(YELLOW);
    editBox->setfont(24,0,"Consolas");
    editBox->move(x,y);
    editBox->visible(true);
    editBox->settext(setText);
}
```

用于绘制输入框。

参数：输入框指针，X 坐标，Y 坐标，宽度，高度，初始显示内容

返回值：无

绘制弹窗(createPopup)函数

```
void userInterface::createPopup(const char* content,const char* buttonText)
//绘制弹窗
{
    popUp newpopup;
    newpopup.content=content;
    newpopup.buttonText=buttonText;
    newpopup.drawPopup(this);
    newpopup.getClick(this);
}
```

用于绘制弹窗。

参数：弹窗内容，按钮内容

返回值：无

数据库(db)

数据库(database)类

```
class dataBase
{
    public:
        sqlite3 *dbPointer;//数据库指针
        char* errorMsg=0;//错误信息
        void openDatabase(const char*);//打开数据库
        void closeDatabase();//关闭数据库
        void insertRecord(const char*,int,...);//插入记录
        dbLines selectRecords(const char*,const char*);//查找记录，返回行号
        dbRecord selectRecordByRowid(const char*,int);//根据行号选择单条记录，返回记录内容
        void updateRecord(const char*,const char*,const char*,const char*,const char*);
        //更新记录内容
    private:
        static int selectRecordsCallback(void *lineList,int argc,char **argv,char **azColName)
        //查找记录的回调
        {
            printf("---select records callback func called---\n");
            ((dbLines*)lineList)->vec.push_back(atoi(argv[0]));
            printf("line id:%s\n",argv[0]);
            return 0;
        }
        static int selectRecordByRowidCallback(void *RecordMap,int argc,char **argv,char **azColName)
        //选择记录的回调
        {
            printf("--select record by rowid callback func called--\n");
            for(int i=0;i<argc;i++)
            {
                printf("colname:%s argv:%s\n",azColName[i],argv[i]);
                ((dbRecord*)RecordMap)->mp.insert(std::pair<std::string,std::string>(azColName[i],argv[i]));
            }
            return 0;
        }
};
```

数据库类，包含程序中用到的对数据库的所有操作。

查询结果(dbLines 和 dbRecord)类

```
class dbLines//返回查询结果所在的行号
{
    public:
        std::vector<int>vec;
};

class dbRecord//一条记录中的信息，标题和内容
{
    public:
        std::map<std::string,std::string>mp;
};
```

用于储存数据库查询返回的结果。

打开数据库(openDatabase)函数

```
void dataBase::openDatabase(const char* fileName)//打开数据库
{
    sqlite3_open(fileName,&dbPointer);
}
```

参数：文件名

返回值：无

关闭数据库(CloseDatabase)函数

```
void dataBase::closeDatabase()//关闭数据库
{
    sqlite3_close(dbPointer);
}
```

参数：无

返回值：无

插入记录(insertRecord)函数

```
void dataBase::insertRecord(const char* tableName,int argc,...)//插入记录
{
    char* collumName[100];
    char* collumValue[100];
    va_list argv;
    va_start(argv,argc);
    for(int i=0;i<argc;i++)
        collumName[i]=va_arg(argv,char*);
    for(int i=0;i<argc;i++)
        collumValue[i]=va_arg(argv,char*);
    va_end(argv);
    std::string sqlCmd;
    sqlCmd+="INSERT INTO ";sqlCmd+=(std::string)tableName;sqlCmd+=" (";
    for(int i=0;i<argc;i++)
    {
        sqlCmd+=(std::string)collumName[i];if(i+1!=argc)sqlCmd+=",";
    }
    sqlCmd+=")\n";
    sqlCmd+="VALUES (";
```

参数：表名，列数，第 1 列到第 n 列的名称，第 1 列到第 n 列的内容。

返回值：无

特别说明：限于 sql 语法规定，字符串类型在传参时两边需要加上单引号。

查找记录(selectRecords)函数

```
dbLines dataBase::selectRecords(const char* tableName,const char* whereStr="")//选择记录
{
    std::string sqlCmd;
    sqlCmd+="SELECT ";
    sqlCmd+="ROWID";
    sqlCmd+=" FROM ";
    sqlCmd+=(std::string)tableName;
    sqlCmd+=" ";
    sqlCmd+=(std::string)whereStr;
    sqlCmd+=" ";

    printf("---select cmd:---\n%s\n",sqlCmd.c_str());

    dbLines lineResult;

    sqlite3_exec(dbPointer,sqlCmd.c_str(),selectRecordsCallback,(void*)&lineResult,&errorMsg);

    return lineResult;
}
```

参数：表名，sql 的 where 语句（查找条件）

返回值：一个 dbLines 类，里边包含结果的行号。

根据行号返回记录(selectRecordByRowid)函数

```
dbRecord dataBase::selectRecordByRowid(const char* tableName,int rowid)
//根据行号选择记录
{
    dbRecord resultRecord;
    std::string sqlCmd;
    char tmp[25];

    sqlCmd+="SELECT * FROM ";
    sqlCmd+=(std::string)tableName;
    sqlCmd+=" WHERE ROWID = ";
    itoa(rowid,tmp,10);
    sqlCmd+=(std::string)tmp;
    sqlCmd+=" \n";

    printf("---select record by rowid cmd---\n%s\n",sqlCmd.c_str());

    sqlite3_exec(dbPointer,sqlCmd.c_str(),selectRecordByRowidCallback,(void*)&resultRecord,&errorMsg);

    return resultRecord;
}
```

参数：表名，行号

返回值：一个 dbRecord 类，里边包含一行记录。

更新记录(updateRecord)函数

```
void dataBase::updateRecord(const char* tableName,const char* whereWhat,
            const char* isWhat,const char* changeWhat,const char* toWhat)
//更新记录
{

    std::string sqlCmd;

    sqlCmd+="UPDATE ";
    sqlCmd+=(std::string)tableName;
    sqlCmd+=" SET ";
    sqlCmd+=(std::string)changeWhat;
    sqlCmd+=" = ";
    sqlCmd+=(std::string)toWhat;
    sqlCmd+=" where ";
    sqlCmd+=(std::string)whereWhat;
    sqlCmd+=" = ";
    sqlCmd+=(std::string)isWhat;
    sqlCmd+="\n";

    printf("----\n%s-----\n",sqlCmd.c_str());
```

参数：表名，在哪里（列名），是“什么”的时候，把什么，改成什么

例如，(“INVENTORY”,”BARCODE”,”2292006”,”STOCK”,”10000”)

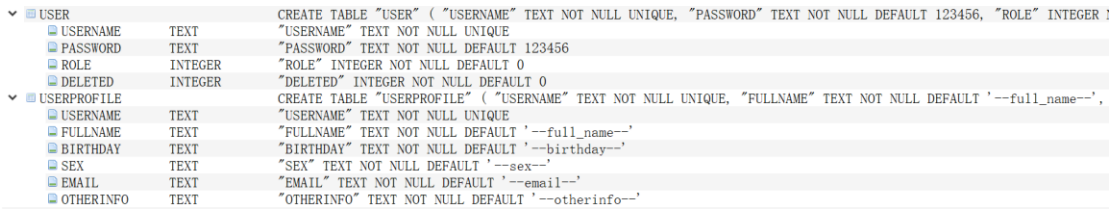
意思是在 INVENTORY 表中，在 BARCODE 为 2292006 的行，把 STOCK 改成 10000。

返回值：无

特别说明：限于 sql 语法规定，字符串类型在传参时两边需要加上单引号。

数据库结构

用户(users.db)



USER		CREATE TABLE "USER" ("USERNAME" TEXT NOT NULL UNIQUE, "PASSWORD" TEXT NOT NULL DEFAULT 123456, "ROLE" INTEGER
USERNAME	TEXT	"USERNAME" TEXT NOT NULL UNIQUE
PASSWORD	TEXT	"PASSWORD" TEXT NOT NULL DEFAULT 123456
ROLE	INTEGER	"ROLE" INTEGER NOT NULL DEFAULT 0
DELETED	INTEGER	"DELETED" INTEGER NOT NULL DEFAULT 0
USERPROFILE		CREATE TABLE "USERPROFILE" ("USERNAME" TEXT NOT NULL UNIQUE, "FULLNAME" TEXT NOT NULL DEFAULT '--full_name--',
USERNAME	TEXT	"USERNAME" TEXT NOT NULL UNIQUE
FULLNAME	TEXT	"FULLNAME" TEXT NOT NULL DEFAULT '--full_name--'
BIRTHDAY	TEXT	"BIRTHDAY" TEXT NOT NULL DEFAULT '--birthday--'
SEX	TEXT	"SEX" TEXT NOT NULL DEFAULT '--sex--'
EMAIL	TEXT	"EMAIL" TEXT NOT NULL DEFAULT '--email--'
OTHERINFO	TEXT	"OTHERINFO" TEXT NOT NULL DEFAULT '--otherinfo--'

包含两个表，第一个储存基本信息，第二个储存附加信息。

以用户名为链接键。

	USERNAME	PASSWORD	ROLE	DELETED
	过滤	过滤	过滤	过滤
1	w43322	11221bxx	1	0
2	raywang777	11221bxx	0	0
3	username	password	1	0
4	Username	Password	0	1

基础信息：用户名、密码、角色(1-管理员|0-普通用户)、删除标记(1-已删除|0-未删除)。

	USERNAME	FULLNAME	BIRTHDAY	SEX	EMAIL	OTHERINFO
	过滤	过滤	过滤	过滤	过滤	过滤
1	raywang777	Ray Wang	20020523	Male	raywang777@foxmail.com	Arthur.
2	w43322	Wang Yilin	20020523	Male	877961980@qq.com	NEU/School of …
3	username	Root User	20210508	Unknown	dalianwangyilin@gmail.…	123
4	Username	Fullname	Birthday	Sex	Email	Otherinfo

附加信息：全名、生日、性别、电子邮件、其他信息。

日志(log.db)

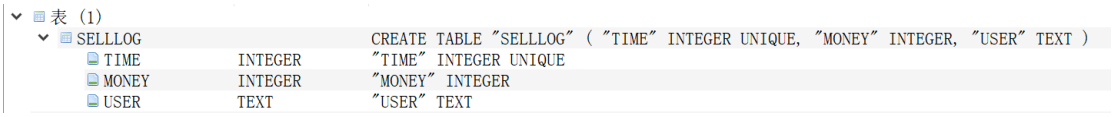
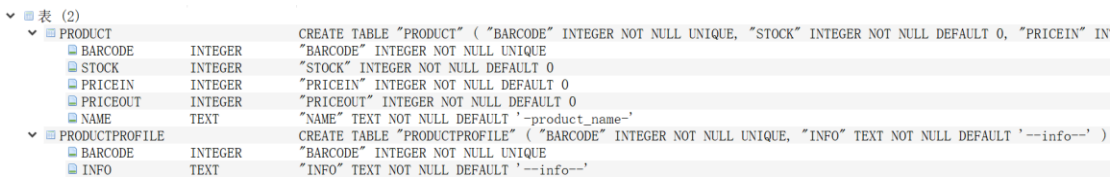


表 (1)		
SELLLOG		CREATE TABLE "SELLLOG" ("TIME" INTEGER UNIQUE, "MONEY" INTEGER, "USER" TEXT)
TIME	INTEGER	"TIME" INTEGER UNIQUE
MONEY	INTEGER	"MONEY" INTEGER
USER	TEXT	"USER" TEXT

	TIME	MONEY	USER
	过滤	过滤	过滤
1	1622169711	1300	username
2	1622169988	3380	username
3	1622170085	1820	username
4	1622265722	966	username
5	1622266873	260	w43322
6	1622270267	138	raywang777
7	1622270271	0	raywang777
8	1622270274	0	raywang777
9	1622270277	0	raywang777
10	1622270280	0	raywang777
11	1622270283	0	raywang777
12	1622270286	0	raywang777
13	1622270307	0	raywang777
14	1622270310	138	raywang777
15	1622270313	0	raywang777
16	1622270314	0	raywang777
17	1622270316	0	raywang777
18	1622270317	0	raywang777
19	1622270319	0	raywang777
20	1622270321	0	raywang777
21	1622270330	0	raywang777
22	1622270332	0	raywang777
23	1622270333	0	raywang777
24	1622270335	0	raywang777
25	1622270336	0	raywang777
26	1622270337	0	raywang777
27	1622270339	0	raywang777

包含交易时间（从 1970 年 1 月 1 日开始计算的秒数）、交易额（单位为分）、交易员三个信息。

仓库(inventory.db)



分两个表，第一个储存基本信息，第二个储存附加信息。

以条形码为链接键。

	BARCODE	STOCK	PRICEIN	PRICEOUT	NAME
	过滤	过滤	过滤	过滤	过滤
1	2292006	187	200	260	NEW Paper Book
2	6924743919228	191	150	450	Lay's Chips (Original)
3	6940426382677	82	50	138	MiaoQianQian Tissue

基础信息包含条形码、库存、进价（单位为分）、出货价（单位为分）、商品名。

	BARCODE	INFO
	过滤	过滤
1	2292006	|PAGE:100|COVER:BROWN|
2	6924743919228	|PAGE:100|
3	6940426382677	|LAYER:3|

附加信息：其他信息。

未来更新计划

除了在《用户手册》中提到的对功能性的改进，在开发者的角度，我还计划加入以下的改进。

目前的密码储存方式是以明文方式储存，而这是直接储存在数据库文件的，而数据库文件又没有加密，因此这无疑很不安全。在下一个版本中，我计划储存密码的 MD5，然后登陆的时候计算所输入的密码的 MD5，再进行比对。这样的话，由于没有储存明文，而 MD5 这类哈希算法的本质又保证了即使数据库被黑，也几乎无法反推出密码，数据的安全性就大大提高。

目前软件的功能还比较初级，性能算不上太好，可能无法胜任特别大的数据库体量。后续我计划对程序做出一些优化，让程序可以适应较大的数据库。

目前的软件鲁棒性欠佳，面对一些未定义行为，容易崩溃或产生未知结果，因此对用户的要求较高。后续我会加强程序的鲁棒性，并充分利用 try-catch 机制，让程序在面对不正确的操作的时候减少崩溃和 bug。