

# 浙江大学



## 图书管理系统设计报告

姓名 袁帅

学号 3080100590

班级 计算机 0803

# 一、需求分析

## 1.1 任务概述

设计并实现一个精简的图书管理系统，要求具有图书入库、查询、借书、还书、借书证管理等功能。

## 1.2 基本功能模块

图表 1 基本功能模块

模块名称	功能描述
管理员登陆	输入管理员 ID, 密码; 登入系统 或 返回 ID/密码 错误.
图书入库	<div>1. 单本入库</div> <div>2. 批量入库 (方便最后测试)</div> <p>图书信息存放在文件中, 每条图书信息为一行. 一行中的内容如下 (书号, 类别, 书名, 出版社, 年份, 作者, 价格, 数量)</p> <p>Note: 其中 年份、数量是整数类型; 价格是两位小数类型; 其余为字符串类型</p> <p>Sample:</p> <p>( book_no_1, Computer Science, Computer Architecture, xxx, 2004, xxx, 90.00, 2 )</p>
图书查询	<p>要求可以对书的 类别, 书名, 出版社, 年份(年份区间), 作者, 价格(区间) 进行查询. 每条图书信息包括以下内容:</p> <p>( 书号, 类别, 书名, 出版社, 年份, 作者, 价格, 总藏书量, 库存 )</p> <p>可选要求: 可以按用户指定属性对图书信息进行排序. (默认是书名)</p>
借书	<div>1.输入借书证卡号</div> <p>显示该借书证所有已借书籍 (返回, 格式同查询模块)</p> <div>2.输入书号</div> <p>如果该书还有库存, 则借书成功, 同时库存数减一.</p> <p>否则输出该书无库存, 且输出最近归还的时间。</p>
还书	<div>1.输入借书证卡号</div> <p>显示该借书证所有已借书籍 (返回, 格式同查询模块)</p> <div>2.输入书号</div> <p>如果该书在已借书籍列表内, 则还书成功, 同时库存加一.</p> <p>否则输出出错信息.</p>
借书证管理	增加或删除一个借书证.

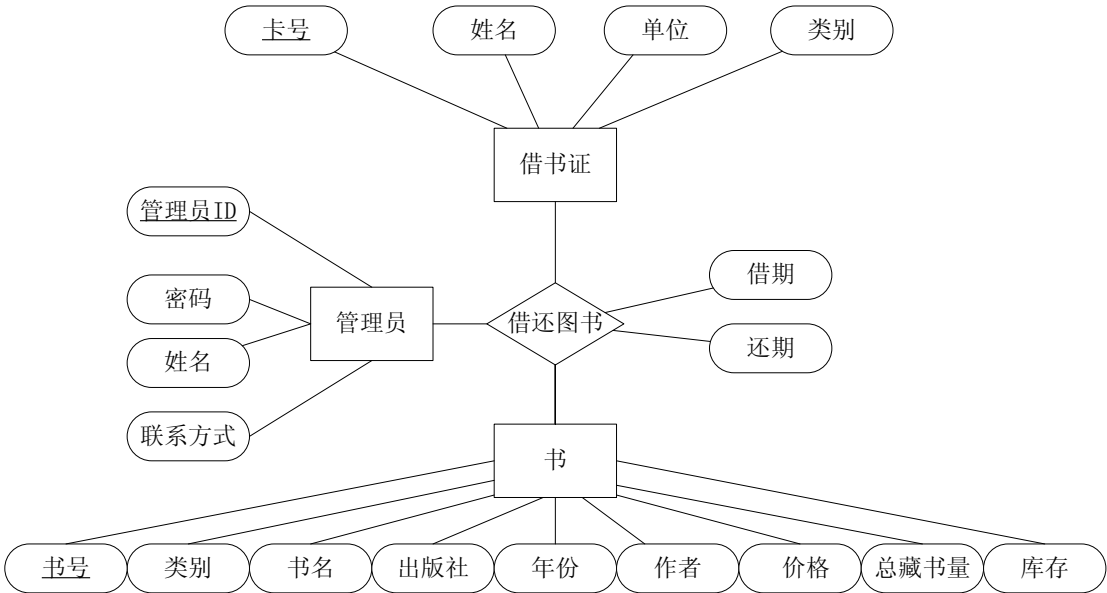
除图书查询功能外，其余功能模块都应该由图书管理员操作。

## 二、运用技术概述

本图书管理系统的设计以 SQL Server 2000 作为数据库平台、ODBC 和 ADO（在“高级查询”中使用了 ADO Data 控件）作为数据库的接口并运用 VC6.0 提供的 MFC 进行图形界面的设计。

## 三、数据库设计

### 3.1E-R 图



图表 2 借还图书的 E-R 图

### 3.2 基本数据对象

图表 3 基本数据对象

对象名称	包含属性
书	书号, 类别, 书名, 出版社, 年份, 作者, 价格, 总藏书量, 库存
借书证	卡号, 姓名, 单位, 类别 (教师 学生等)
管理员	管理员 ID, 密码, 姓名, 联系方式
借书记录	卡号, 借书证号, 借期, 还期, 经手人 (管理员 ID)

### 3.2 对应的数据库对象

图表 4 对应的数据库对象

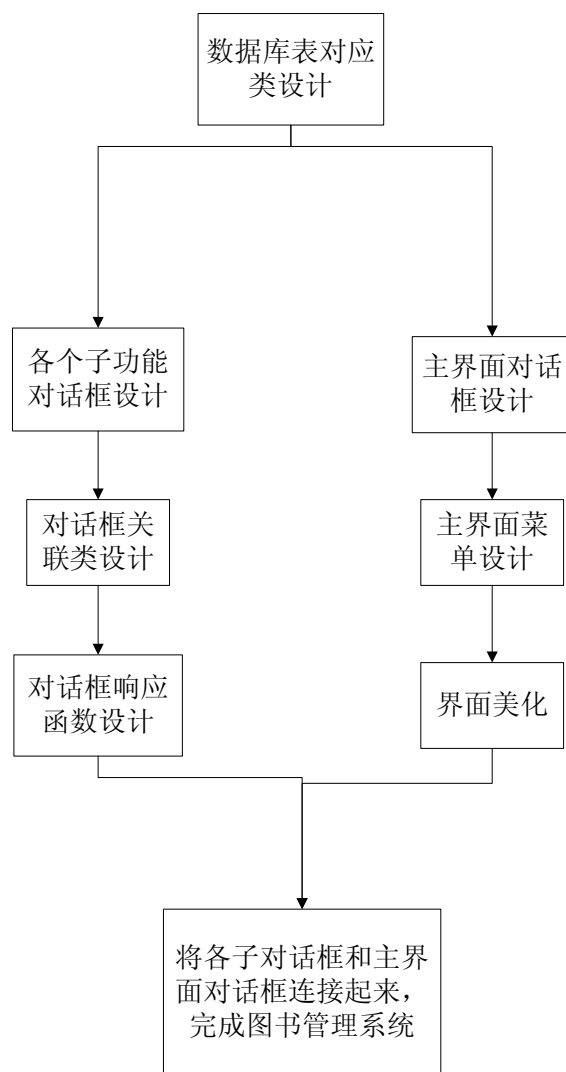
Name of the schema	attribute
book	bno, category, title, press, year, author, price, total, stock
borrow	cno, name, department, type
admini	admini_ID, code, admin_name, contact
borrow	cno, bno , borrow_date, return_date, admini_ID

## 四、应用系统设计

### 4.1. 系统总体设计

#### 4.1.1. 系统设计的总体过程

本图书管理系统是一个 Dialog Based 的应用程序。其设计的总体过程如下：



图表 5 系统设计的总体过程

#### 4.1.2. 数据库表对应类设计

使用 MFC ODBC 访问数据源的记录集需要派生 CRecordSet 类, 在本图书管理系统的设计中数据库各表和对应的 CRecordSet 派生类如下:

图表 6 CRecordSet 派生类

表名	对应的派生类名称
book	CBookRS
borrow	CBorrowRS
card	CCardRS
admini	CAdminiRS

本图书管理系统是一个 Dialog Based 的应用程序。各个对话框的信息如下:

图表 7 对话框信息

对话框名称及功能描述	对应类
登录	CLoginDlg
library (管理员登录后的主界面)	CMainMenu

单本入库	CSingleAddDlg
批量入库	CMultiplyAdd
图书查询	CBasicQuery
高级查询	CAdvancedQuery
借书	CBorrowBook
还书	CReturnBook
增加借书证	CCardNewDlg
删除借书证	CCardLostDlg

## 4.2 登录



图表 8 登录对话框界面

登录对话框的对应类是 CLoginDlg，其对应的成员变量和成员函数如下：

图表 9 登录对话框成员变量

变量名称	变量类型	访问特性	用途
m_adminiSet	CAdminRS	private	检查管理员登录信息是否正确
m_strAdminiID	CString	public	关联并记录用户在编辑框中输入的管理员 ID
m_strCode	CString	public	关联并记录用户在编辑框中输入的密码

图表 10 登录对话框成员函数

函数原型	功能
void CLoginDlg::OnOK()	点击“管理员登录”按钮的响应函数，判断管理员登录信息是否正确，错误则弹出相应出错提示，正确则打开主界面对话框，并把用户在编辑框中输入的管理员 ID 赋给当前登录的管理员 ID——全局变量 currentID。
void CLoginDlg::OnReader()	点击“读者登录”按钮的响应函数，打开“图书查询”对话框

1、点击“管理员登录”按钮：

输入管理员 ID 和密码、按下管理员登录时，先检查 ID 是否存在，否则弹出出错消息：



图表 11

若管理员 ID 存在，检查密码是否正确，若密码不匹配，弹出出错消息



图表 12

若管理员 ID 和密码均正确，则打开主菜单对话框。注意到借书记录的表需要管理员 ID 来作为其“经手人”属性，故应定义一个全局变量来记录当前登录的 ID。

以下是包含该全局变量的头文件“currentID.h”

```
#ifndef CURRENTID_H
#define CURRENTID_H
#include "stdafx.h"
CString currentID;
#endif
```

使用该全局变量时需加上：

```
#include "currentID.h"
extern CString currentID;
```

在管理员登录成功后更新 currentID：

```
currentID=m_strAdminiID;
```

然后打开主界面对话框。

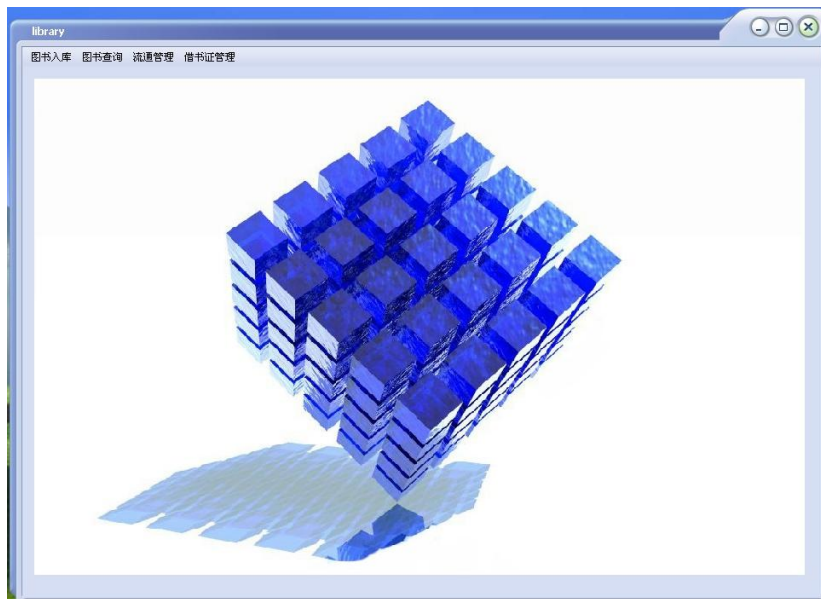
2、点击“管理员登录”按钮：

打开“图书查询”对话框，此时读者可以对书名进行模糊查询：

3、点击“取消”按钮：退出系统。

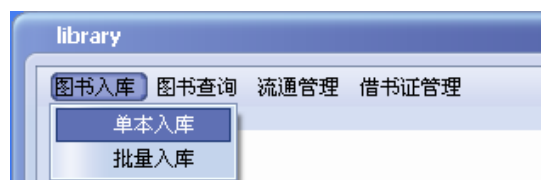
## 4.3 主界面设计

出于对友好的用户界面和交互的考虑，我在主界面添加了位图和菜单：



图表 13

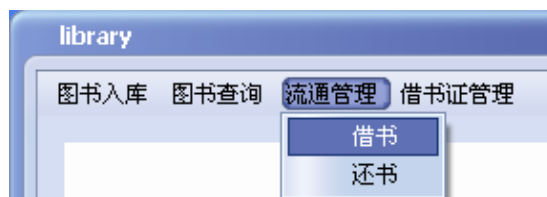
各菜单项如下：



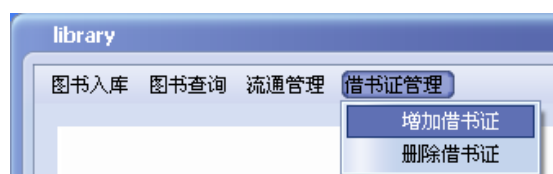
图表 14



图表 15



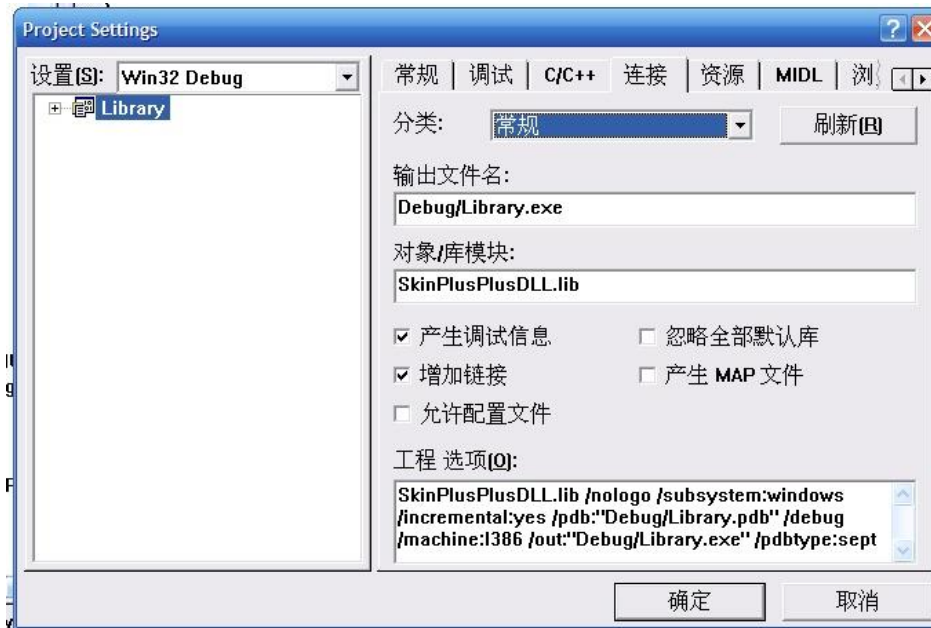
图表 16



图表 17

此外，我还使用了 SkinPlusPlus 来美化界面，先是在当前目录中加入 SkinPlusPlusDLL.lib, SkinPlusPlus.h, XPCorona.ssk, SkinPlusPlusDLL.dll 文件，再在工程中添加 SkinPlusPlus.h，然后在 Project Settings 的连接->对象/库模块中添加 SkinPlusPlusDLL.lib





图表 18

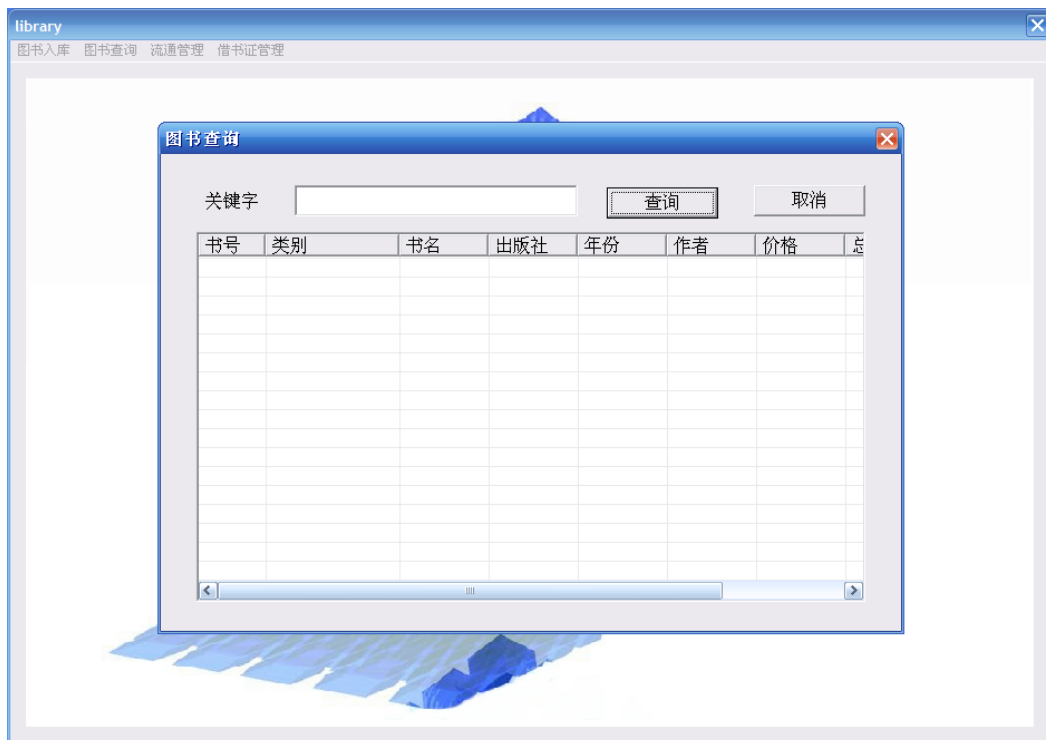
最后在 `BOOL CLibraryDlg::OnInitDialog()` 函数中添加

```
InitializeSkin(_T("XPCorona.ssk"));
```

并添加头文件:

```
#include "SkinPlusPlus.h"
```

这样当编译链接运行时就是如上效果，而不是以下的普通界面：



图表 19 没有用 SkinPlusPlus 时的普通界面

4.4 单本入库



图表 20 单本入库对话框界面

单本入库对话框的对应类是 CSingleAddDlg，其对应的成员变量和成员函数如下：

图表 21 单本入库对话框的成员变量

变量名称	变量类型	访问特性	用途
m_bookSet	CBookRS	private	检查入库的书籍是否已存在
m_author	CString	public	关联并记录用户在编辑框中输入的作者
m_bookNum	CString	public	关联并记录用户在编辑框中输入的书号
m_category	CString	public	关联并记录用户在编辑框中输入的类别
m_press	CString	public	关联并记录用户在编辑框中输入的出版社
m_title	CString	public	关联并记录用户在编辑框中输入的书名
m_price	CString	public	关联并记录用户在编辑框中输入的价格
m_year	long	public	关联并记录用户在编辑框中输入的年份
m_number	long	public	关联并记录用户在编辑框中输入的数量

图表 22 单本入库对话框的成员函数

函数原型	功能
void CSingleAddDlg::OnOK()	点击“确定”按钮的响应函数，判断入库的书籍是否存在，并进行相应的入库操作。

先对入库的书籍进行查询，如果入库的是新书，则增加一条新的记录：

```
UpdateData(); //更新数据
if (!m_bookSet.IsOpen()) {
    m_bookSet.Open();
}

m_bookSet.m_strFilter.Format("bno='%s'", m_bookNum);
m_bookSet.Requery();
//若该书不存在，则创建新项目
if(m_bookSet.IsEOF())
{
    m_bookSet.AddNew();
}
```

```

m_bookSet.m_bno=m_bookNum;
m_bookSet.m_category=m_category;
m_bookSet.m_title=m_title;
m_bookSet.m_press=m_press;
m_bookSet.m_year=m_year;
m_bookSet.m_author=m_author;
m_bookSet.m_price=m_price;
m_bookSet.m_total=m_number;
m_bookSet.m_stock=m_number;
m_bookSet.Update();
}

```

例如插入如下新书:

图表 23

点击“确定”:

图表 24

在 SQL Server 2000 中进行检验:

<pre>select * from book where bno='234'</pre>									
	bno	category	title	press	year	author	price	total	stock
1	234	计算机	数据库	浙江大学出版社	2010	db	25.00	2	2

图表 25

如果入库的是旧书，则更新其总藏书量和库存：

```
else
{
    m_bookSet.Edit();
    m_bookSet.m_total+=m_number;
    m_bookSet.m_stock+=m_number;
    m_bookSet.Update();
}
```

例如再添加一本刚才的书：

图表 26

点击“确定”

图表 27

在 SQL Server 2000 中进行检验：

<pre>select * from book where bno='234'</pre>									
	bno	category	title	press	year	author	price	total	stock
1	234	计算机	数据库	浙江大学出版社	2010	db	25.00	3	3

图表 28

## 4.5 批量入库



图表 29 批量入库对话框界面

批量入库对话框的对应类是 `CMultiplyAdd`，其对应的成员变量和成员函数如下：

图表 30 批量入库对话框的成员变量

变量名称	变量类型	访问特性	用途
<code>m_bookSet</code>	<code>CBookRS</code>	<code>private</code>	检查入库的书籍是否已存在

图表 31 批量入库对话框的成员函数

函数原型	功能
<code>void CSingleAddDlg::OnOK()</code>	点击“确定”按钮的响应函数，打开“打开文件”对话框所选的文件，读入数据，判断要入库的书籍是否存在，并进行相应的入库操作。

点击“确定”时，创建 `CFileDialog` 类，弹出打开文件的对话框，通过 `GetFileName()` 取得文件名。由于得到的文件名是 `CString` 类型，要用 `fopen` 打开文件还需要将 `CString` 类型转换为 `char *` 类型：

```
CString filePath;//用户所选择的文件的路径
CFileDialog filedlg(TRUE);//设置为 TRUE 读文件模式
if(IDOK==filedlg.DoModal())//显示打开文件对话框
    filePath=filedlg.GetFileName();//得到 CFileDialog 的文件名
UpdateData();//更新数据
int length;
char * FileName;//文件名
//将 CString 类型的 filePath 转换为 char * 类型的 FileName
length=filePath.GetLength();//得到 filePath 长度
FileName=new char[length];//分配内存空间
//复制 filePath 内容到 FileName
for(int j=0;j<length;j++)
    *(FileName+j)=filePath.GetAt(j);
*(FileName+length)='\0';

FILE *file;
file=fopen(FileName,"r");
if(!file) //判断是否打开成功
{
    cout<<"文件打开失败"<<endl;
    exit(0);
}
```

}



图表 32

按照格式要求，编辑“data.txt”文件的内容如下：

```
(ys01,农业,Uuvi,浙江大学出版社,1993,Wljahtif,44.13,4)
(ys02,经济,Xon,上海交通大学出版社,2008,Ay,19.52,4)
(ys03,计算机,Bqylrbsv,人民邮电出版社,2008,Xolwajy,7.62,1)
```

点击“确定”：



图表 33

接下来是处理从文件读到的字符串，并从中提出要入库书籍的各个属性的信息：

```
char *attribute[10]; //相应属性的字符串值
char *line; //读到的一行字符串
//分配内存空间
line=(char *)malloc(1000);
for(int i=0;i<10;i++){
    attribute[i]=(char *)malloc(1000);
}

while(fgets(line,1000,file)!=NULL)
{
    line[strlen(line)-1]='\0';
    for(int i=0;i<strlen(line);i++)
    {
        //忽略")"字符
        if(line[i]==')')
        {
```

```

        line[i]='\0';
    }
}
//第一个是" ("字符, 忽略, 并以", "为分隔符将读到的字符串写入 attribute 数组
元素中
attribute[0]=strtok(line+1, ",");
for(int k=1;k<=7;k++){
    attribute[k]=strtok(NULL, ",");
}
//根据 attribute 的字符串值插入数据
CString m_bno,m_category,m_title,m_press,m_author,m_price;
int m_year,m_number;
m_bno.Format("%s",attribute[0]);
m_category.Format("%s",attribute[1]);
m_title.Format("%s",attribute[2]);
m_press.Format("%s",attribute[3]);
m_year=atoi(attribute[4]);
m_author.Format("%s",attribute[5]);
m_price.Format("%s",attribute[6]);
m_number=atoi(attribute[7]);

```

剩下的操作与“单本入库”一样，先对入库的书籍进行查询，如果入库的是新书，则增加一条新的记录；如果入库的是旧书，则更新其总藏书量和库存。

## 4.6 基本查询



图表 34 图书查询对话框界面

图书基本查询对话框的对应类是 CBasicQuery，其对应的成员变量和成员函数如下：

图表 35 基本查询对话框的成员变量

变量名称	变量类型	访问特性	用途
m_bookSet	CBookRS	private	检查入库的书籍是否已存在
m_ctrList	CListCtrl	public	显示查找结果并按用户指定属性对图书信息进

			行排序
m_strKeyWord	CString	public	关联并记录用户在编辑框中输入的书名关键字

图表 36 基本查询对话框的成员函数

函数原型	功能
BOOL CBasicQuery::OnInitDialog()	在初始化“图书查询”对话框时增加 ListControl 的表头设置列宽和拓展风格。
void CBasicQuery::OnQuery()	点击“查询”按钮的响应函数，对图书进行模糊查询。
void CBasicQuery::OnColumnclickListctrl(NMHDR* pNMHDR, LRESULT* pResult)	按用户点击表头的某一列时，按其指定属性对图书信息进行排序。

基本查询采用 MFC 对 ODBC 封装的 CRecordSet 的操作，其中 m\_strFilter 为记录集的 select 语句中的 where 子句，故只需把 m\_bookSet 的 m\_strFilter 赋为“(属性)like ‘%(关键字)%’”即可根据书名的关键字对图书进行模糊查询。相应代码如下：

```
UpdateData(); //将编辑框内容更新到与此绑定的成员变量 m_strKeyWord 中
if (!m_bookSet.IsOpen()) {
    m_bookSet.Open();
}
//设置查询条件，根据书名关键字进行模糊查询
m_bookSet.m_strFilter="title like '%" + m_strKeyWord + "%'";
m_bookSet.Requery();
//若找不到该书，则弹出提示并返回
if (m_bookSet.IsEOF()) {
    AfxMessageBox("查无此书");
    return;
}
```

在数据的显示上，我用的是 ListControl 控件。ListControl 一般有大图标（LVS\_ICON）、小图标（LVS\_SMALLICON）、列表（LVS\_LIST）和报告（LVS\_REPORT）显示方式，这里选择“报告”以显示详细资料。



图表 37

ListControl 的显示需要在对话框初始化时进行设置：

```
BOOL CBasicQuery::OnInitDialog()
{
    CDialog::OnInitDialog();
```



```

//添加表头
m_ctrList.InsertColumn(0,"书号");
m_ctrList.InsertColumn(1,"类别");
m_ctrList.InsertColumn(2,"书名");
m_ctrList.InsertColumn(3,"出版社");
m_ctrList.InsertColumn(4,"年份");
m_ctrList.InsertColumn(5,"作者");
m_ctrList.InsertColumn(6,"价格");
m_ctrList.InsertColumn(7,"总藏书量");
m_ctrList.InsertColumn(8,"库存");
//设置列宽
m_ctrList.SetColumnWidth(0,60);
m_ctrList.SetColumnWidth(1,120);
m_ctrList.SetColumnWidth(2,80);
m_ctrList.SetColumnWidth(3,80);
m_ctrList.SetColumnWidth(4,80);
m_ctrList.SetColumnWidth(5,80);
m_ctrList.SetColumnWidth(6,80);
m_ctrList.SetColumnWidth(7,80);
m_ctrList.SetColumnWidth(8,80);
//设置 ListControl 的拓展风格
m_ctrList.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);
//LVS_EX_FULLROWSELECT 为选中某行使整行高亮功能，为“报告”显示方式所特有
// LVS_EX_GRIDLINES 为网格线功能，为“报告”显示方式所特有
return TRUE; // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}

```

在查询的响应函数中，也要添加 ListControl 的显示：

```

m_ctrList.DeleteAllItems(); //进行查询时，清空 ListControl 原有的内容
m_ctrList.SetRedraw(FALSE); // ListControl 内容进行大数据量更新时，避免闪烁
//将全部的结果显示在 ListControl 中
int i=0;
CString str; //将非 CString 类型的成员变量转换为 CString 类型以便显示
while (!m_bookSet.IsEOF()){
    m_ctrList.InsertItem(i,m_bookSet.m_bno); //插入 bno 属性的值作为索引
    //添加其他内容
    m_ctrList.SetItemText(i,1,m_bookSet.m_category);
    m_ctrList.SetItemText(i,2,m_bookSet.m_title);
    m_ctrList.SetItemText(i,3,m_bookSet.m_press);
    str.Format("%d",m_bookSet.m_year);
    m_ctrList.SetItemText(i,4,str);
    m_ctrList.SetItemText(i,5,m_bookSet.m_author);
    str.Format("%f",m_bookSet.m_price);
    m_ctrList.SetItemText(i,6,str);
    str.Format("%d",m_bookSet.m_total);
    m_ctrList.SetItemText(i,7,str);
    str.Format("%d",m_bookSet.m_stock);
    m_ctrList.SetItemText(i,8,str);
    i++;
    m_bookSet.MoveNext();
}
m_bookSet.Close();
m_ctrList.SetRedraw(TRUE);

```

最后是按用户指定属性对图书信息进行排序的功能。ListControl 控件以“报告”形式显示时，列表的表头可以当做按钮来使用，当点击表头时，ListControl 就会向父窗口发送 LVN\_COLUMNCLICK 消息。故可以在 LVN\_COLUMNCLICK 消息加入相应函数，使得当

用户点击表头的某一列时，对该列进行排序。

一般来说，点击表头实现排序需要由第三方的排序函数来实现，例如通过排序的回调函数。

我这里是直接在 OnColumnclick 函数中进行冒泡排序的：

```
void CBasicQuery::OnColumnclickListctrl(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    static int sortstyl=1;//排序方式
    int selectcol = pNMListView->iSubItem;//获得当前所选列
    int listcount;//总行数
    listcount = m_ctrList.GetItemCount();
    CString temp0,temp1,temp2;
    if(sortstyl==1)
    {
        //冒泡排序
        for(int j = 1;j<=listcount;j++)
        {
            for(int i=0;i<listcount-j;i++)
            {
                temp1 = m_ctrList.GetItemText(i,selectcol);
                temp2 = m_ctrList.GetItemText(i+1,selectcol);
                if(temp1 > temp2)
                {
                    for(int n=0;n<9;n++)//共有 9 列
                    {
                        temp1 = m_ctrList.GetItemText(i,n);
                        temp2 = m_ctrList.GetItemText(i+1,n);
                        m_ctrList.SetItemText(i,n,temp2);
                        m_ctrList.SetItemText(i+1,n,temp1);
                    }
                }
            }
        }
        //点击第二次时将换一种方式排
    }
    else if(sortstyl==-1)
    {
        for(int j = 1;j<=listcount;j++)
        {
            for(int i=0;i<listcount-j;i++)
            {
                temp1 = m_ctrList.GetItemText(i,selectcol);
                temp2 = m_ctrList.GetItemText(i+1,selectcol);
                if(temp2 > temp1)
                {
                    for(int n=0;n<9;n++)//共有 9 列
                    {
                        temp1 = m_ctrList.GetItemText(i,n);
                        temp2 = m_ctrList.GetItemText(i+1,n);
                        m_ctrList.SetItemText(i,n,temp2);
                        m_ctrList.SetItemText(i+1,n,temp1);
                    }
                }
            }
        }
        sortstyl = sortstyl*(-1);//每点击一次 sortstyl 乘以负 1，这样下次点击就换成另一种方式排序
    }
    *pResult = 0;
}
```

例如根据书号排序：(列旁边有个三角形符号表示选中了该列)，一开始是升序：

[illegible]

图表 39

4.7 高级查询



图表 40 高级查询对话框界面

高级查询对话框的对应类是 CAdvancedQuery，其对应的成员变量和成员函数如下：

图表 41 高级查询对话框的成员变量

变量名称	变量类型	访问特性	用途
m_adodc	CAdodc	public	连接数据库并对数据库中的数据进行操作
m_datagrid	CDataGrid	public	接收 ADO Data 控件结果集中的数据并显示查找结果
m_author	CString	public	关联并记录用户在编辑框中输入的作者
m_category	CString	public	关联并记录用户在编辑框中输入的类别
m_press	CString	public	关联并记录用户在编辑框中输入的出版社
m_title	CString	public	关联并记录用户在编辑框中输入的书名
m_fromYear	CString	public	关联并记录用户在编辑框中输入的年份下限
m_toYear	CString	public	关联并记录用户在编辑框中输入的年份上限
m_fromPrice	CString	public	关联并记录用户在编辑框中输入的价格下限
m_toPrice	CString	public	关联并记录用户在编辑框中输入的价格上限

图表 42 高级查询对话框的成员函数

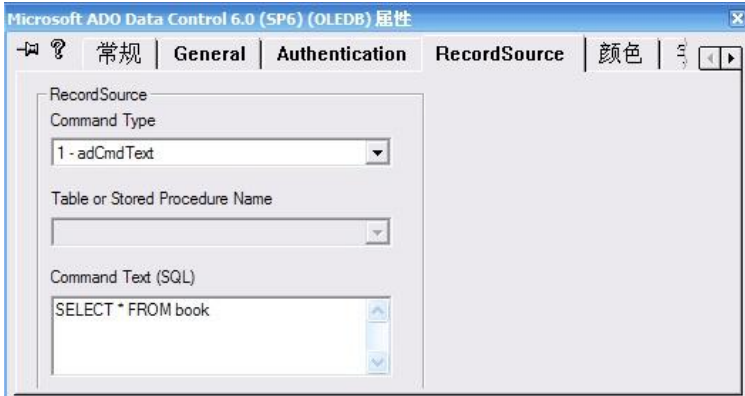
函数原型	功能
------	----

void CAdvancedQuery::OnOK()	点击“确定”按钮的响应函数，根据输入的条件对图书进行查询。
BOOL CAdvancedQuery::OnInitDialog()	在初始化“图书查询”对话框时显示所有图书。

高级查询我是用 ADO Data 控件来连接数据库并对数据库中的数据进行访问操作的，ADO Data 需要设置连接的 ODBC 数据源名称和记录源：



图表 43 设置 ODBC 数据源名称



图表 44 设置记录源

由于 ADO Data 控件不能够显示数据库中数据的内容，需要与其他控件结合使用。我添加的是 DataGrid 控件，只需在 DataSource 中选择 ADO Data 控件的 ID 就可以使 DataGrid 控件接收 ADO Data 控件结果集中的数据并显示。



图表 45 DataGrid 的设置

接下来是处理查询条件。

首先需要判断需不需要对对应属性进行查询操作，我的做法是把所有编辑框的变量都设为 CString 类型，再通过 CString 的 IsEmpty() 函数判断是否输入。（注意 m\_fromYear、m\_toYear、m\_fromPrice、m\_toPrice 这四个绑定的成员变量也需要设为 CString 类型，而不是 int 型和 float 型，因为 int 型和 float 型默认显示 0，若用户输入 0 是无法判断的。）如果对应属性的关键字非空，则插入相应的条件语句。

还有一个关键点在于各个条件语句之间的 " and " 连接问题，我的做法是若当前关键字非空，则求出前面非空的关键字的个数，如果个数大于 0 就应在相应的条件语句前加入 " and "，具体代码如下：

```
int i,j;
int count=0;
CString str;
CString strFilter;//查询条件
CString astrKeyWord[8];//指向 CString 的指针数组，记录用户输入的关键字
astrKeyWord[0]=m_category;
astrKeyWord[1]=m_title;
astrKeyWord[2]=m_press;
astrKeyWord[3]=m_author;
//m_fromYear、m_toYear、m_fromPrice、m_toPrice 这四个绑定的成员变量设为
CString 类型，以便检查输入是否为空
astrKeyWord[4]=m_fromYear;
astrKeyWord[5]=m_toYear;
astrKeyWord[6]=m_fromPrice;
astrKeyWord[7]=m_toPrice;
for(i=0;i<8;i++)
{
    if(!astrKeyWord[i].IsEmpty())
    {
        //若当前关键字非空，检查前面的关键字是否为空已决定是否要插入 " and "
        for(j=0;j<i;j++)
        {
            if(!astrKeyWord[j].IsEmpty())
                count++; //count 求出前面非空的关键字的个数
        }
        //若前面非空的关键字的个数，则需要在当前的查询语句中插入 " and "
        if(count>0)
        {
```

精确值

```
        strFilter+=" and ";
    }
    //以下是针对各个关键字，插入相应的查询语句
    //注意：年份和价格为不严格的大于小于关系，这样当上下限相等时就可以查询
    if(i==0)
    {
        str=" category like'%" +m_category+"%' ";
        strFilter+=str;
    }
    if(i==1)
    {
        str=" title like'%" +m_title+"%' ";
        strFilter+=str;
    }
    if(i==2)
    {
        str=" press like'%" +m_press+"%' ";
        strFilter+=str;
    }
    if(i==3)
    {
        str=" author like'%" +m_author+"%' ";
        strFilter+=str;
    }
    if(i==4)
    {
        str.Format(" year>=%s ",m_fromYear);
        strFilter+=str;
    }
    if(i==5)
    {
        str.Format(" year<=%s ",m_toYear);
        strFilter+=str;
    }
    if(i==6)
    {
        str.Format(" price>=%s ",m_fromPrice);
        strFilter+=str;
    }
    if(i==7)
    {
        str.Format(" price<=%s ",m_toPrice);
        strFilter+=str;
    }
}
}
```

通过 ADO Data 对象的 SetRecordSource()函数设置 ADO Data 控件的数据源，只需把要查询的 select 语句传入 SetRecordSource()的参数中，并用 Refresh()函数刷新结果集的内容。由于可能存在空查询，故需要进行判断并执行相应操作。

```
CString cSource;//ADO 查询语句
cSource="SELECT bno AS 书号,category AS 类别,title AS 书名,press AS 出版社,year AS 年份,author AS 作者,price AS 价格,total AS 总藏书量,stock AS 库存 FROM book";
//根据输入是否非空来决定显示的内容
count=0;
for(i=0;i<8;i++)
{
    if(!astrKeyWord[i].IsEmpty())
```

```

        count++;    //count 求出所有非空的关键字的个数
    }
    //如果输入非空，则显示查询结果；如果输入为空，则显示所有图书的信息
    if(count!=0)
    {
        cSource+=" WHERE ";
        cSource+=strFilter;
    }
    m_adodc.SetRecordSource(cSource);
    m_adodc.Refresh();

```

最后设置 DataGrid 的显示，先用 long(n)把整数转换为\_variant\_t 类型，使用 DataGrid 对象的 GetColumns()得到 CColumns 对象，再用 GetItem()所返回的 CColumn 对象确定表格中具体一列，最后用 SetWidth()设置列的宽度。

```

    _variant_t vIndex;
    vIndex=long(0);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(1);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(2);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(3);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(4);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(5);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(6);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);
    vIndex=long(7);
    m_datagrid.GetColumns().GetItem(vIndex).SetWidth(100);

```

再添加 CColumn、CColumns 和\_variant\_t 类所定义的头文件

```

#include "column.h"
#include "columns.h"
#include "COMDEF.H"

```



4.8 借书



图表 46 借书对话框界面

借书对话框的对应类是 C BorrowBook，其对应的成员变量和成员函数如下：

图表 47 借书对话框的成员变量

变量名称	变量类型	访问特性	用途
m_cardSet	C CardRS	private	检查卡号信息是否正确
m_bookSet	C BookRS	private	查询和修改图书信息
m_borrowSet	C BorrowRS	private	查询和修改借书信息
m_strBookNum	CString	public	关联并记录用户在编辑框中输入的书号
m_strCardNum	CString	public	关联并记录用户在编辑框中输入的借书证卡号
m_ctrList	C ListCtrl	public	显示查找结果

图表 48 借书对话框的成员函数

函数原型	功能
void CBorrowBook::OnQuery()	点击“已借书籍”按钮的响应函数，根据输入的条件对指定借书证的已借书籍进行查询。
void CBorrowBook::OnOK()	点击“确定借书”按钮的响应函数，通过调用下面的四个自定义函数判断能够借书，并完成相应的功能。
bool CBorrowBook::IsValidCard(const CString &cardNum)	判断卡号信息是否正确。
bool CBorrowBook::AnyBookExpired(const CString &cardNum)	判断该借书证所借的书是否超过借书期限或借书限额。
bool CBorrowBook::Nobook(const CString &bookNum)	检查所借图书是否不存在或没有库存。
bool CBorrowBook::BorrowBook(const CString &cardNum, const CString &bookNum)	所有的判断都通过后，调用此函数来完成借书功能。

BOOL CBorrowBook::OnInitDialog()	在初始化“借书”对话框时增加 ListControl 的表头设置列宽和拓展风格。
----------------------------------	--

点击“已借书籍”：  
 首先检查卡号信息是否正确，若卡号不存在则弹出出错信息。  
 然后通过 borrow 表检查是否有已借书籍，若没有已借书籍则弹出提示；如果存在已借书籍  
 则通过 borrow 表找到的 bno 再到 book 表找出详细信息，并把内容显示到 ListControl 中。  
 ListControl 的显示详见“基本查询”对话框的设计。  
 点击“确定借书”：  
 首先检查卡号信息是否正确，若卡号不存在则拒绝借书。  
 其次检查借书是否超过借书期限和借书限额，如果超过则拒绝借书。  
 再所借图书是否不存在或没有库存，如果是则拒绝借书。  
 如果以上条件都通过的话，则允许借书，并在 book 表中将这本书的库存减 1，在 borrow 表  
 中增加一条新纪录。  
 具体实现参见源代码。



图表 49 按下“已借书籍”



图表 50 输入不正确的借书卡号：拒绝借书并弹出错误提示



图表 51 所借书籍不存在：拒绝借书并弹出错误提示



图表 52 所借书籍没有库存：拒绝借书并弹出错误提示



图表 53 所借书籍没有库存：出错误提示之后提示最近归还的时间



图表 54 借书总数超额：拒绝借书并弹出错误提示



图表 55 借书过期：拒绝借书并弹出错误提示



图表 58 还书对话框的成员变量

变量名称	变量类型	访问特性	用途
m_cardSet	CCardRS	private	检查卡号信息是否正确
m_bookSet	CBookRS	private	查询和修改图书信息
m_borrowSet	CBorrowRS	private	查询和修改借书信息
m_strBookNum	CString	public	关联并记录用户在编辑框中输入的书号
m_strCardNum	CString	public	关联并记录用户在编辑框中输入的借书证卡号
m_ctrList	CListCtrl	public	显示查找结果

图表 59 还书对话框的成员函数

函数原型	功能
void CReturnBook::OnQuery()	点击“已借书籍”按钮的响应函数，根据输入的条件对指定借书证的已借书籍进行查询。
void CReturnBook::OnOK()	点击“确定”按钮的响应函数，完成还书的功能。
BOOL CReturnBook::OnInitDialog()	在初始化“还书”对话框时增加 ListControl 的表头设置列宽和拓展风格。

点击“已借书籍”：

与“借书”对话框的“已借书籍”实现相同。

点击“确定还书”：

先查找该卡未还的书籍，如果找不到纪录，说明该书已换或不存在，此时拒绝操作并弹出提示。

若找到该书籍，则先检查该书所借的天数，如果超过借书期限则弹出过期的提示并继续进行还书操作。

最后将还书日期改成当前日期，并在 book 表中对该藏书库存加 1

4.10 增加借书证



图表 60 增加借书证对话框界面

增加借书证对话框的对应类是 CCardNewDlg，其对应的成员变量和成员函数如下：

图表 61 增加借书证对话框的成员变量

变量名称	变量类型	访问特性	用途
m_cardSet	CCardRS	private	查询和修改借书证信息

m_cardNum	CString	public	关联并记录用户在编辑框中输入的借书证卡号
m_department	CString	public	关联并记录用户在编辑框中输入的单位
m_name	CString	public	关联并记录用户在编辑框中输入的姓名
m_type	CString	public	关联并记录用户在编辑框中输入的类别

图表 62 增加借书证对话框的成员函数

函数原型	功能
void CCardNewDlg::OnOK()	点击“确定”按钮的响应函数，检查输入数据是否合理并执行相应操作。

检查该借书证是否已经存在，若不存在则增加新记录，若已存在则拒绝增加并弹出提示：

```
UpdateData(); //将编辑框中的数据更新到成员变量 m_cardNum 中
if (!m_cardSet.IsOpen()) {
    m_cardSet.Open();
}
//查找借书证，检查是否已经存在
m_cardSet.m_strFilter.Format("cno='%s'", m_cardNum);
m_cardSet.Requery();
//若该卡不存在，则在 card 表中增加一条新的记录
if (m_cardSet.IsEOF())
{
    m_cardSet.AddNew();
    m_cardSet.m_cno=m_cardNum;
    m_cardSet.m_name=m_name;
    m_cardSet.m_department=m_department;
    m_cardSet.m_type=m_type;
    m_cardSet.Update();
    AfxMessageBox("申请成功!");
}
//若该卡已存在，则弹出相应提示
else
{
    AfxMessageBox("该卡已存在");
}
CDialog::OnOK();
```

例如增加如下一张新的借书证：

图表 63

点击“确定”





图表 64

在 SQL Server 2000 中进行检验：

```
select * from card
```

	cno	name	department	type
1	1	a	a	T
2	123	s	d	U
3	2	b	b	U
4	3	a	b	T
5	abc	A	CS	U

图表 65

再增加同样一张卡，则弹出出错提示：



图表 66

## 4.11 删除借书证



图表 67 删除借书证对话框界面

删除借书证对话框的对应类是 CCardLostDlg，其对应的成员变量和成员函数如下：

图表 68 删除借书证对话框的成员变量

变量名称	变量类型	访问特性	用途
m_cardSet	CCardRS	private	查询和修改借书证信息
m_cardNum	CString	public	关联并记录用户在编辑框中输入的借书证卡号

图表 69 删除借书证对话框的成员函数

函数原型	功能
void CCardLostDlg::OnOK()	点击“确定”按钮的响应函数，检查输入数据是否合理并执行相应操作。

与“增加借书证”类似，查找借书证，检查是否已经存在。若已存在则删除相应记录，若不存在则拒绝操作并弹出提示。

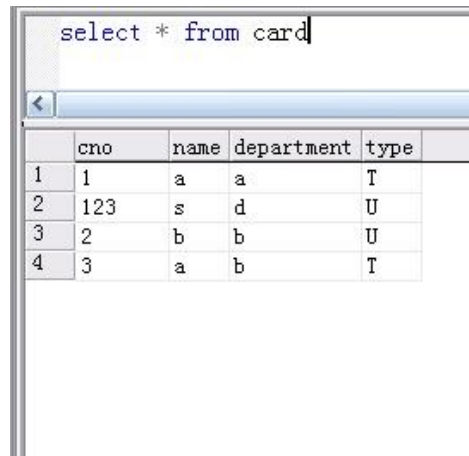
```
if (!m_cardSet.IsOpen()) {
    m_cardSet.Open();
}
UpdateData();
m_cardSet.m_strFilter.Format("cno= '%s'",m_cardNum);
m_cardSet.Requery();
//如果找到，则删除该记录，并提示注销成功
if(!m_cardSet.IsEOF())
{
    m_cardSet.Delete();
    AfxMessageBox("注销成功");
}
//若找不到，则提示该卡不存在
else
{
    AfxMessageBox("该卡不存在");
}
CDialog::OnOK();
```

例如删除刚才新增的借书证：



图表 70

在 SQL Server 2000 中进行检验，该记录已不存在：



The screenshot shows a window with a text area containing the SQL query `select * from card`. Below the text area is a table with the following data:

	cno	name	department	type
1	1	a	a	T
2	123	s	d	U
3	2	b	b	U
4	3	a	b	T

图表 71

## 五、已解决的 bug 及调试过程

1、一开始我的基本查询和借书、还书对话框用的不是 ListControl 控件，而是跟《数据库课程设计》中一样，用一个 List Box 显示查找结果清单，另一个 List Box 显示详细信息。如借书的对话框：



The screenshot shows a dialog box titled "借书" (Borrow Book). It contains two input fields: "借书卡号" (Borrow Card Number) and "藏书编号" (Collection Number), each with an "编辑" (Edit) button. Below these fields are two empty List Boxes. At the bottom, there are three buttons: "已借书籍" (Already Borrowed Books), "确定借书" (Confirm Borrow Book), and "取消" (Cancel).

图表 72 借书对话框的初始版本

我对查询“已借书籍”功能的实现见如下代码：（其中 CBookBorrowDlg 是原先借书对话框对应的类，OnQuery()是“已借书籍”按钮的响应函数，OnSelchangeListResult()是选择 List Box 中一项的响应函数，m\_listResult 为上面显示查找结果清单的 List Box，m\_listDetail 为下面显示详细信息的 List Box）

```
void CBookBorrowDlg::OnQuery()
{
    UpdateData(); //将编辑框内容更新到 m_strCardNum 中
    m_listResult.ResetContent(); //清空结果列表的内容（如果有）
    if (!m_borrowSet.IsOpen()) {
        m_borrowSet.Open();
    }
    if (!m_bookSet.IsOpen()) {
```

```

        m_bookSet.Open();
    }
    m_borrowSet.m_strFilter.Format("cno='%s'",m_strCardNum);
    m_borrowSet.Requery();
    while(!m_borrowSet.IsEOF())
    {
        m_bookSet.m_strFilter.Format("bno='%s'",m_borrowSet.m_bno);
        m_bookSet.Requery();
        if (m_bookSet.IsEOF()){
            AfxMessageBox("没有已借书籍");
            return;
        }
        m_listResult.AddString(m_bookSet.m_title); //把书名添加到结果列表中
        m_borrowSet.MoveNext();
    }
}

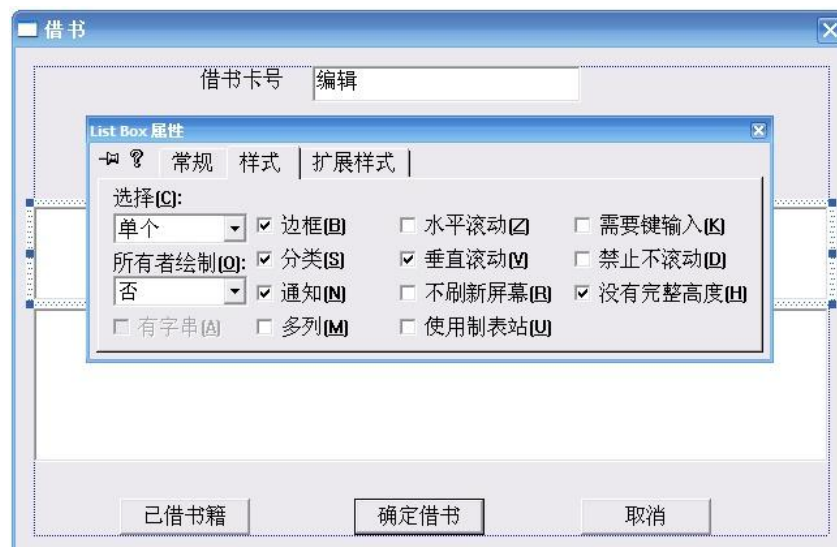
void CBookBorrowDlg::OnSelchangeListResult()
{
    CString aColCaption[18]; //aColCaption 存放列标题信息
    aColCaption[0]="书号: ";
    aColCaption[1]="类别: ";
    aColCaption[2]="书名: ";
    aColCaption[3]="出版社: ";
    aColCaption[4]="年份: ";
    aColCaption[5]="作者: ";
    aColCaption[6]="价格: ";
    aColCaption[7]="总藏书量: ";
    aColCaption[8]="库存: ";
    m_listDetail.ResetContent(); //清空书目详细信息列表中的原有内容
    int curSel; //记录集中的当前位置
    CString strDetail; //内容
    CString strLine; //列表中的一行, 行=标题: +内容
    curSel=m_listResult.GetCurSel(); //取得结果列表中当前的选中位置
    m_borrowSet.SetAbsolutePosition(1); //将记录集指针移到最开始的位置
    //根据游标 curSel 来把记录集指针移到相应位置
    while(curSel!=0)
    {
        m_borrowSet.MoveNext();
        curSel--;
    }
    //到 book 表中查询图书的详细信息
    m_bookSet.m_strFilter.Format("bno='%s'",m_borrowSet.m_bno);
    m_bookSet.Requery();
    for (int i=0;i<m_bookSet.m_nFields;i++){
        m_bookSet.GetFieldValue(i,strDetail); //从记录集中取出第 i 列的值, 存在 strDetail 中
        strLine.Empty(); // 清空原有的内容
        strLine+=aColCaption[i]; //增加标题
        strLine+=strDetail; //增加内容
        m_listDetail.AddString(strLine); //把该行添加到书目详细信息的列表中
    }
}

```

算法本身是没有问题的, 运行时的内容也是正确的, 但却出现了详细列表的内容与结果列表所点击的项目不对应的问题。一开始以为问题出在 OnSelchangeListResult() 中的游标上,

后来发现原来不是代码问题，而是 List Box 自动将内容做了排序。

于是我放弃使用 List Box，学习用 ListControl 来显示结果。后来听说只需把 List Box 属性里的“分类”默认的钩去掉就可以解决 List Box 显示的问题：



图表 73

但我最后仍然采用的是 ListControl，因为 ListControl 实现根据用户所选定的列进行排序还是比较方便的。

2、一开始的“单本入库”对话框执行入库操作时，总是会弹出“对于造型说明无效的字符值”这样的错误提示。后来发现是变量类型不匹配的问题：由于 SQL 定义 book 时把 year、total、stock 定义成 int 类型，把 price 定义为 decimal(7,2) 类型，于是我在类向导里把输入的年份和数量定为 int 型，price 定为 float 型，但 book 表对应的 CRecordSet 派生类 CBookRS 类中的成员变量却是这么声明的：

```
CString    m_bno;  
CString    m_category;  
CString    m_title;  
CString    m_press;  
long       m_year;  
CString    m_author;  
CString    m_price;  
long       m_total;  
long       m_stock;
```

于是我相应地把输入的年份和数量定为 long 型，price 定为 CString 型，就解决了这一问题。

3、一开始没有考虑查询时空输入的情况，后来经同学提醒，发现一开始的“高级查询”版本空输入会导致崩溃，于是加入了输入是否为空的判断，解决了这个问题。

4、一开始的“高级查询”不支持模糊查询功能，这主要是使用 ADO Data 连接数据库的缘故。我之前试过在“基本查询”中用这种方式进行模糊查询，但效果跟精确匹配一样，在网上也看到了同样的说法，于是就没有在使用 ADO Data 的“高级查询”中做模糊查询。验收后觉得“高级查询”不支持模糊查询功能很亏，于是又试了一下，发现 ADO Data 还是可以实现模糊查询的。

## 六、设计总结

在做这次图书管理系统之前，我只接触过一点 windows 编程的皮毛，从未用过 MFC。一开始我只是按照《数据库课程设计》按部就班地做，后来通过不断查找资料学习，我渐渐能够深入了解和独立使用 MFC。设计的过程中我也尝试了不少新东西，例如 ADO、SkinPlusPlus 和 ListControl 等控件，发现这些貌似神秘的东西其实并没有想象中那么困难。在这个过程中，我也遇到了不少问题和麻烦，为了实现一些功能也尝试了不少版本。总的来说，虽然艰辛，但收获良多。

当然，由于时间比较仓促，本图书管理系统还不是很完善，有待进一步改进。