第13章　文件、串行化和数据库

本章的例子涉及从简单二进制文件到访问第三方的数据库管理系统等方面的内容。文件串行化的概念是MFC特有的，串行化允许你组织如何保存类对象到磁盘上，以便以较小的代价检索甚至升级它们。串行化从 archive文件中保存或调出， archive文件其实是一个二进制文件。

例63　访问二进制文件　讨论用MFC类访问和操作一个简单的二进制文件。

例64　访问标准I/O文件　讨论用MFC类访问文本文件。

例65　访问内存文件　讨论创建和操作内存中的二进制文件。尤其是，内存文件可以使你利用文件系统的方法来访问堆内存的单元。

例66　在数据类中实现串行化　在数据类中加入串行化功能。在本例中不实际串行化任何东西 —— 这些留到下五个例子来做。

例67　串行化SDI或MDI文档　利用MFC的内置功能自动串行化数据类。

例68　按要求串行化　按要求串行化一个或多个数据类，而不仅仅是保存和装入一个文档。

例69　透明地更新串行化的文档　讨论怎样如何通过简单地修改新数据类，以转化旧数据，使新数据类仍然能使用旧的串行化文档。

例70　串行化多态类　串行化一组数据类，这些类的唯一共同点是它们派生于同一个基类。

例71　串行化数据集　讨论MFC集类内置的串行化功能。

例72　访问ODBC数据库　讨论怎样使用MFC类访问一个满足ODBC协议的数据库。

例73　访问DAO数据库　讨论怎样使用MFC类访问一个满足DAO协议的数据库。

13.1　例63　访问二进制文件

目标

在一个平面磁盘文件中维护二进制数据。

策略

利用MFC的CFile类，该类封装了Windows API中处理二进制文件的函数。

步骤

1. 检查文件是否存在

要判断某个文件是否已在磁盘上存在，可以使用静态 CFile函数GetStatus( )。

```
CFileStatus status;
if (!CFile::GetStatus(sFile,status))
{
```

```
    msg.Format("%s does not exist",sFile);
    AfxMessageBox(msg);
}
```

2. 创建一个二进制文件

为了创建一个可写的二进制文件，首先创建一个 CFile类对象，然后利用Open( )成员函数打开并创建一个文件对象。

```
CFile file;
CString msg;
CString sFile("Wzd.tmp");
if (!file.Open(sFile, CFile::modeCreate|CFile::modeWrite))
{
    msg.Format("Failed to create %s.",sFile);
    AfxMessageBox(msg);
}
```

3. 使用二进制文件

1) 写二进制数据到文件中。

```
file.Write (buffer, sizeof (buffer));
```

2) 关闭文件，销毁文件对象。

```
file. Close( );
```

3) 打开一个二进制文件进行读操作。

```
if (!file.Open(sFile, CFile::modeRead))
{
    msg.Format("Failed to open %s.",sFile);
    AfxMessageBox(msg);
}
```

4) 获得二进制或其他类型文件的长度。

```
UINT nBytes = file.GetLength( );
```

5) 使用如下函数之一以改变文件读写的当前位置。

```
file.SeekToEnd();
file.Seek(20,          // file offset in bytes
    CFile::begin);     // also CFile::end and CFile::current
file.SeekToBegin();
```

6) 利用如下代码从二进制文件中读取数据，其中 nBytes是实际读取的字节数。若 nBytes比想要读取的字节数小或者为0，则已到文件结束处。

```
nBytes = file.Read (buffer, sizeof (buffer)) ;
```

7) 设置文件为只读，使用CFile的静态成员函数SetStatus( )。本例中，我们获取文件当前的任何状态，并加上只读状态标记。

```
if (CFile::GetStatus(sFile, status))
{
    status.m_attribute|=0x01;
    CFile::SetStatus(sFile, status);
}
```

8) 删除一个文件，使用CFile的另外一个静态函数Remove( )。

```
CFile::Remove (sFile);
```

说明

- C++类中，CFile和大多数其他MFC类一样包含了许多一个C++类中的Windows API调用。CWnd类对于窗口是这样，CMenu类对于菜单也是这样。
- 关于操作文本文件参见下一个例子。

CD说明

在CD上执行该工程时，在WzdView.cpp的OnTestWzd( )函数处设置断点。选取菜单上的Test/Wzd项，一步步地观察二进制文件的访问。

13.2 例64 访问标准I/O文件

目标

操作文本文件。

策略

利用MFC的CStdioFile类操作文本文件。CStdioFile从CFile类派生，并封装Windows API中用来处理文本文件的函数。

步骤

1. 检查一个文件是否存在

使用CFile类的GetStatus( )静态函数，判断一个文本文件是否存在。

```
CFileStatus status;
CString sFile("Wzd.txt");
if (!CFile::GetStatus(sFile,status))
{
   msg.Format("%s does not exist",sFile);
   AfxMessageBox(msg);
}
```

2. 创建一个文本文件

打开一个文本文件以写文本串。本例中，若所找文件无法打开，则创建一个新文件。

```
CStdioFile file;
if(!file.Open(sFile, CFile::modeWrite | CFile::typeText))
{
   if(!file.Open(sFile, CFile::modeCreate |
      CFile::modeWrite | CFile::typeText))
   {
      CString msg;
      msg.Format("Failed to create %s.",sFile);
   }
}
```

3. 使用文本文件

1) 写一个文本串到文件中。

```
file.WriteString (sRecord);
```

2) 关闭文件并销毁创建的文件对象。

```
file.Close( );
```

3) 打开一个文本文件进行读操作。

```
if (!file.Open(sFile, CFile::modeRead | CFile::typeText))
{
   msg.Format("Failed to open %s.",sFile);
   AfxMessageBox(msg);
}
```

4) 定位到文件的头和尾。

```
file.SeekToEnd( );
file.SeekToBegin( );
```

5) 从一个文本文件中读一个文本串。

```
file.ReadString (sRecord);
```

6) 删除一个文本文件。

```
CFile:: Remove (sFile);
```

说明

- 与二进制文件以同样方式对待每个字节所不同的是，标准 I/O文件使用fgets( )和fputs( )函数获取和输入文本串到文件中。
- 既然CStdioFile是从CFile派生的，在CFile中可用的函数在此都可用。

CD说明

- 在CD上执行该工程时，在 WzdView.cpp中的函数OnTestWzd( )处设置断点。选取菜单上的Test/Wzd项，一步步观看标准I/O文件的访问。

13.3 例65 访问内存文件

目标

在内存中生成一个文件，而不是在磁盘上生成一个临时文件。

策略

使用MFC的CMemFile类，该类是从CFile类派生的，封装分配内存的Windows API。

步骤

1. 创建一个内存文件

要创始一个内存文件，只要创建一个 CMemFile类对象。

```
CMemFile file;
```

创建一个CMemFile类对象自动打开一个内存文件。

2. 使用内存文件

1) 写内存文件。

```
file.Write (buffer, sizeof (buffer));
```

2) 判断内存文件的当前长度。

```
UINT nBytes = file.GetLength( );
```

3) 使用以下方法定位下次读写的位置。

```
file.SeekToEnd();
file.Seek(20,         // file offset in bytes
   CFile::begin);     // also CFile::end and CFile::current
file.SeekToBegin();
```

4) 从内存文件中读数据。

```
nBytes = file.Read (buffer, sizeof (buffer));
```

5) 删除内存文件有两种方法：一是销毁CMemFile类对象，二是设置文件长度为0。

```
file.SetLength (0);
```

说明

- 内存文件为用户在内存中操作数据提供了一种方便的途径，而不必考虑释放和分配堆空间的内存段。另外，内存文件可用作临时 (.tmp)文件，这样可以加快应用程序的运行速度，同时没有命名和删除临时文件的麻烦。
- CMemFile创建的内存文件是在应用程序的堆空间中生成的。若想在全局堆上创建内存文件，相应地要使用 CSharedFile，特别是剪贴板要求其获取的数据必须在全局堆中。关于此问题的例子参见例76的剪切和粘贴。
- 既然CMemFile是从CFile而不是从CStdioFile派生来的，则必须实现 ReadString( )和WriteString( )函数，以方便地操作内存文件中的文本数据。

CD说明

在CD上执行该工程时，在 WzdView.cpp中的OnTestWzd( )函数上设置断点。单击菜单上的Test/Wzd项，一步一步地观察内存文件的访问。

13.4 例66 在数据类中实现串行化

目标

串行化数据类。

策略

实现从CObjct派生过来的数据类的串行化。MFC的CObject类添加一个虚重载函数到类中，该函数为Serialize( )。我们将用我们自己的 Serialize( )重载这个Serialize( )，使之装入和保存成员变量到磁盘上。

注意 本例中，数据类具有串行化自己的功能。实际如何串行化类到磁盘上参见下例。

步骤

1. 在数据类中加入Serialize( )

从MFC的CObject派生得到用户的数据类。不是从 CObject派生的类不能进行串行化。因为ClassWizard没有提供把CObject作为类的基类的功能，则需要加进这个基类。

```
class CMyData : public CObject
{
};
```

利用以下格式，在每个数据类中加入 Serialize( )成员函数。

```
void CWzdInfo2::Serialize(CArchive& ar)
{
   CObject::Serialize(ar);
   int version=1;
   if(ar.IsStoring())
   {
      // store data
   }
   else
   {
      // load data
   }
}
```

现在可以利用Serialize( )函数调出和保存每一个类的成员变量到存档设备中。如何串行化一个成员变量依赖于变量类型。有三种变量类型：不是从 CObject派生过来的简单变量，如integers, floats等；从CObject派生过来的变量；数据集，包括数组和列表。

2. 串行化简单变量

1) 要保存数据类中不是从CObject派生的成员变量，在Serialize( )函数的“保存数据”区使用<<重载操作符。

```
CObject::Serialize(ar);
   if(ar.IsStoring())
   {
   // store member variables not derived from CObject
      ar << m_sName;
      ar << m_sComment;
```

2) 要调出简单成员变量到数据表中，在 Serialize( )函数的“调出数据”区使用 >>重载操作符。

```
CObject::Serialize(ar);
if(ar.IsStoring())
{
// store data
else
{
// load member variables not derived from CObject
   ar >> m_sName;
   ar >> m_sComment;
}
```

3. 串行化派生于CObject的变量

调用成员变量的Serialize( )函数以保存派生于CObject的成员变量。

```
// load and store member variables derived from CObject
```

```
m_wzdInfo.Serialize(ar);
```

可以在类的Serialize( )函数的调出和保存区中，调用该Serialize( )函数，或者在if { }else { }表达式之后调用该 Serialize( )一次即可。若在变量类中没有实现 Serialize( )，则应该加上一个。

4. 串行化数据集

1) 保存数据集，使用如下语句。

```
nCount = m_WzdInfoList.GetCount();     // get number of
                                       // items in
  ar << nCount;                        // and store
  for (POSITION pos = m_WzdInfoList.GetHeadPosition(); pos;)
  {
     CWzdInfo2 *pInfo = m_WzdInfoList.GetNext(pos);
      pInfo->Serialize(ar);
  }
}
```

注意在这里我们首先保存数据集中工程的数目，然后一步一步地按顺序保存每个工程。

2) 使用如下语句，装入数据集。

```
// load a list of data
  ar >> nCount;
  while (nCount-- > 0)
  {
     CWzdInfo2* pInfo = new CWzdInfo2;
     pInfo->Serialize(ar);
     m_WzdInfoList.AddTail(pInfo);
  }
}
```

关于数据类串行化的详细清单，参见本节的“ 清单 —— 第一个数据类 ”和“ 清单 —— 第二个数据类 ”。

说明

- 串行化把数据类的每个成员变量按序列地存贮到磁盘上。调出或保存一个文档只不过是串行化上千个数据类而已。只要文档按它保存的顺序调出，就没有必要对文件进行其他控制。正如我们在后面例子看到一样，这种方法可以很快地把旧的文档格式“ 转化 ”为新的格式。关于串行化和CObject类的其他信息参见第2章。
- 串行化的存档设备可以是磁盘文件、内存文件或甚至是别的系统上的文件。在例 76中可以看到，串行化到内存文件可以为用户提供一种快速易用方式，来剪切和粘贴自己的数据类。
- 一些MFC类可以使用>>和<<重载操作符来串行化其数据，但是这些类要自己实现这种操作。如CString就使用这个操作符。可以通过测试查看还有哪些 MFC类有这样的功能。
- 把保存一个类对象并能在以后恢复它的能力叫做对象“ 存留 ”。

CD说明

本例子的工程称为“ Serialization ”，开始时在WzdDoc.cpp中Serialize( )上设置断点。然后

单击Test/Wzd菜单命令填充文档，之后单击File/Save开始串行化功能。

清单——第一个数据类

```
#ifndef WZDINFO1_H
#define WZDINFO1_H

#include "afxtempl.h"
#include "WzdInfo2.h"

class CWzdInfo1 : public CObject
{
public:
    CWzdInfo1();
    ~CWzdInfo1();

    //misc info
    CString             m_sGroupName;
    CString             m_sComment;
    CList<CWzdInfo2*,CWzdInfo2*> m_WzdInfo2List;
    void Serialize(CArchive& archive);

};
#endif

// WzdInfo1.cpp : implementation of the CWzdInfo class
//

#include "stdafx.h"
#include "WzdInfo1.h"

//////////////////////////////////////////////////////////////////////
// CWzdInfo

CWzdInfo1::CWzdInfo1()
{
    m_sGroupName=_T("");
    m_sComment=_T("");
}

CWzdInfo1::~CWzdInfo1()
{
    while (!m_WzdInfo2List.IsEmpty())
    {
        delete m_WzdInfo2List.RemoveHead();
    }
}


void CWzdInfo1::Serialize(CArchive& ar)
```

```
{
    CObject::Serialize(ar);
    int nCount;
    if (ar.IsStoring())
    {
        // name and comment
        ar << m_sGroupName;
        ar << m_sComment;

        // other list
        nCount = m_WzdInfo2List.GetCount();
        ar << nCount;
        for (POSITION pos = m_WzdInfo2List.GetHeadPosition(); pos;)
        {
            CWzdInfo2 *pInfo = m_WzdInfo2List.GetNext(pos);
            ar << pInfo;
        }
    }

    else
    {
        // name and comment
        ar >> m_sGroupName;
        ar >> m_sComment;

        // other list
        ar >> nCount;
        CObject* pInfo;
        while (nCount-- > 0)
        {
            ar >> pInfo;
            m_WzdInfo2List.AddTail((CWzdInfo2*)pInfo);
        }
    }
}
```

清单——第二个数据类

```
#ifndef WZDINFO2_H
#define WZDINFO2_H


class CWzdInfo2 : public CObject
{
public:

enum STATES {
    OLD,
    NEW,
    MODIFIED,
```

```
    DELETED
    };

    DECLARE_SERIAL(CWzdInfo2)

    CWzdInfo2();
    CWzdInfo2(CString sName,int nVersion);

    void Set(CString sName,CString sComment,int nVersion, int nState);

    // misc info
    CString    m_sName;
    CString    m_sComment;
    int        m_nState;

    void Serialize(CArchive& archive);

    CWzdInfo2& operator=(CWzdInfo2& src);

};
#endif

// WzdInfo2.cpp : implementation of the CWzdInfo class
//

#include "stdafx.h"
#include "WzdInfo2.h"

/////////////////////////////////////////////////////////////////////////////
// CWzdInfo

IMPLEMENT_SERIAL(CWzdInfo2, CObject, 0)


CWzdInfo2::CWzdInfo2()
{
    m_sName=_T("");
    m_sComment=_T("");
    m_nState=CWzdInfo2::NEW;
}

CWzdInfo2::CWzdInfo2(CString sName,int nVersion) :
    m_sName(sName),m_nVersion(nVersion)
{
    m_sComment=_T("");
    m_nState=CWzdInfo2::OLD;
}

void CWzdInfo2::Set(CString sName,CString sComment,int nVersion, int nState)
```

```
{
    m_sName=sName;
    m_sComment=sComment;
    m_nState=nState;
}


void CWzdInfo2::Serialize(CArchive& ar)
{
    CObject::Serialize(ar);
    if(ar.IsStoring())
    {
        // data
        ar << m_sName;
        ar << m_sComment;
        ar << m_nState;

    }
    else
    {
        // data
        ar >> m_sName;
        ar >> m_sComment;
        ar >> m_nState;

    }
}


CWzdInfo2& CWzdInfo2::operator=(CWzdInfo2& src)
{
    if(this != &src)
    {
        m_sName = src.m_sName;
        m_sComment = src.m_sComment;
        m_nState = src.m_nState;
    }
    return *this;
}
```

13.5 例67 串行化SDI或MDI文档

目标

利用MFC框架的内建特征串行化SDI和MDI文档到磁盘。

策略

利用文档类中的两个函数实现从磁盘中调出文档或串行化文档到磁盘上。第一个是文档

类的Serialize( )函数，通过对它编程，系统地调用文档中每个数据类实例的 Serialize( )成员函数(在前例中，我们已经添加了该成员函数)。第二个是DeleteContents( )，用来初始化文档。我们还将编辑一段代码来定制打开文件对话框的文档扩展名。

步骤

1. 使用文档类的Serialize( )

定位Document类的Serialize( )成员函数，如前例一样，在Serialize( )函数中填入代码。在此只是简单地串行化文档类的每个成员变量到作为参数所传递过来的存档类参考中。

2. 使用文档类的DeleteContents( )

使用ClassWizard，在文档类中加入 DeleteContents( )重载函数。当选择 File/New或者File/Open菜单命令时，该重载函数被自动调用，提供一个重新初始化文档的机会。

```
void CWzdDoc::DeleteContents()
{
   // called by new and open document
   // opportunity to initialize the data collections
   // that make up your document
   while (!m_WzdInfo1List.IsEmpty())
   {
      delete m_WzdInfoList.RemoveHead();
   }
   CDocument::DeleteContents();
}
```

在Document类中不必添加其他重载函数或消息处理函数。当单击 File/New或者File/Open时，应用程序框架将自动地打开一个Open File对话框，提示用户输入一个文件名。一旦选定，应用程序将打开一个文件和一个存档，然后调用文档类的 DeleteContents( )。一旦文档被初始化，应用程序会调用Serialize( )函数。用户选择File/save或File/Save As时也是如此。

如果用户单击的是File/New菜单命令，只有DeleteContents( )被调用。

自动打开的File Dialog使用你给出的文件扩展名，在文件列表中过滤掉其他非此类型的文档。也可在使用AppWizard创建应用程序时指定这个文件扩展名。如果没有这么做，也可以在应用程序的字符串表中编辑一个字符串，指定文件扩展名。

3. 指定文件扩展名

为了定制在应用程序中打开供用户选择文件名的文件对话框，可以在字符串资源表中编辑一个字符串。定位定义文档类型的字符串。在本例中， ID为IDC_WZDTYPE。对这个字符串所需的修改如下面灰底和下划线的部分。

```
\nWzd\nWzd\nWzd Files (*.wzd)\n.wzd\nWzd.Document\nWzd Document
```

说明

- 严格地说，即使是DeleteContents( )，也只是在SDI应用程序才被用到。当一个MDI应用程序打开一个新的或已经存在的文档时，一个新的文档类被创建。因此，任意文档的初始化工作都可以在文档类的构造函数中进行。但是，习惯上应使用 DeleteContents( )。

CD说明

本例子的工程叫“ Serialization ”。首先在WzdDoc.cpp中的Serialize( )上设置断点。然后单击Test/Wzd菜单命令来填充文档。最后，单击 File/save开始串行化工作。

清单 —— 文档类

```
// WzdDoc.h : interface of the CWzdDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_WZDDOC_H__CA9038EE_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_)
#define AFX_WZDDOC_H__CA9038EE_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "afxtempl.h"
#include "WzdInfo1.h"
#include "WzdInfo2.h"

class CWzdDoc : public CDocument
{
protected:                      // create from serialization only
    CWzdDoc();
    DECLARE_DYNCREATE(CWzdDoc)

// Attributes
public:
    CList<CWzdInfo1*,CWzdInfo1*> *GetInfo1List(){return &m_WzdInfo1List;};

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWzdDoc)
    public:
    virtual void Serialize(CArchive& ar);
    virtual void DeleteContents();
    virtual BOOL OnNewDocument();
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CWzdDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
```

```
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CWzdDoc)
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CList<CWzdInfo1*,CWzdInfo1*> m_WzdInfo1List;
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif // !defined(
    AFX_WZDDOC_H__CA9038EE_B0DF_11D1_A18C_DCB3C85EBD34__INCLUDED_)

// WzdDoc.cpp : implementation of the CWzdDoc class
//

#include "stdafx.h"
#include "Wzd.h"

#include "WzdDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CWzdDoc

IMPLEMENT_DYNCREATE(CWzdDoc, CDocument)

BEGIN_MESSAGE_MAP(CWzdDoc, CDocument)
    //{{AFX_MSG_MAP(CWzdDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
//////////////////////////////////////////////////////////////////////////
// CWzdDoc construction/destruction

CWzdDoc::CWzdDoc()
{
    // TODO: add one-time construction code here
}

CWzdDoc::~CWzdDoc()
{
}

BOOL CWzdDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    return TRUE;
}


//////////////////////////////////////////////////////////////////////////
// CWzdDoc serialization

void CWzdDoc::Serialize(CArchive& ar)
{
    int nCount;
    if (ar.IsStoring())
    {
        nCount = m_WzdInfo1List.GetCount();
        ar << nCount;
        for (POSITION pos = m_WzdInfo1List.GetHeadPosition(); pos;)
        {
            CWzdInfo1 *pInfo = m_WzdInfo1List.GetNext(pos);
            pInfo->Serialize(ar);
        }
    }
    else
    {
        ar >> nCount;
        while (nCount-- > 0)
        {
            CWzdInfo1* pInfo = new CWzdInfo1;
            pInfo->Serialize(ar);
            m_WzdInfo1List.AddTail(pInfo);
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CWzdDoc diagnostics

#ifdef _DEBUG
void CWzdDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CWzdDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CWzdDoc commands

void CWzdDoc::DeleteContents()
{
    // called with new and open document
    // opportunity to initialize the data collections that make up our document
    while (!m_WzdInfo1List.IsEmpty())
    {
        delete m_WzdInfo1List.RemoveHead();
    }


    CDocument::DeleteContents();
}
```

13.6 例68 按要求串行化

目标

随时手工串行化数据类。

策略

假定已经为每一个数据类添加了一个 Serialize( )函数，如例66所示。在这里，我们将打开一个文件和存档，然后串行化数据类到存档中。

步骤

1. 保存串行化的数据类

1) 保存任何有Serialize( )函数的数据类到磁盘上。首先，创建一个文件；然后，打开一个

CArchive类的实例。

```
CFile file;
if (file.Open("filename.ext", CFile::modeCreate|
    CFile::modeWrite))
{
    CArchive ar(&file, CArchive::store);
```

注意，在这里创建存档实例时使用的是CArchive::store标志。

2) 使用该存档调用Serialize( )函数在相应的数据类中循环。本例子中，我们串行化 CList 变量。

```
CList<CWzdInfo1*,CWzdInfo1*> *pList =
    GetDocument()->GetInfo1List();
nCount = pList->GetCount();
ar << nCount;
for (POSITION pos = pList->GetHeadPosition(); pos;)
{
    CWzdInfo1 *pInfo = pList->GetNext(pos);
    pInfo->Serialize(ar);
}
```

3) 串行完之后，首先关闭存档，然后关闭存档和文件。

```
  ar.Close();
    file.Close();
}
```

2. 调出串行化的数据类

为了调出串行化数据类，首先打开一个文件，然后再打开存档调出数据。

```
CFile file;
if (file.Open("filename.ext", CFile::modeRead))
{
    CArchive ar(&file, CArchive::load);
    CList<CWzdInfo1*,CWzdInfo1*> *pList =
        GetDocument()->GetInfo1List();
    ar >> nCount;
    while (nCount-- > 0)
    {
        CWzdInfo1* pInfo = new CWzdInfo1;
        pInfo->Serialize(ar);
        pList->AddTail(pInfo);
    }
    ar.Close();
    file.Close();
}
```

注意，在这里创建存档实例时使用的是CArchive::load标志。

说明

若不想使用应用框架自动文档调出和保存功能，这种方法是非常有用的。例如，应用程序想保存自己的数据库和日志。后面，将使用这种方法来实现剪切和粘贴，将串行化一个共享的内存文件，而不是一个CFile类对象。

CD说明

本例的工程称为“ Serialization ”。首先在WzdDoc.cpp中的Serialize( )函数上设置断点。然后单击Test/Wzd菜单命令填充文档。最后单击File/Save开始串行化工作。

13.7 例69 透明地更新串行化的文档

目标

在不创建转换应用程序的情况下，让用户访问和转换应用程序文档的旧版本。

策略

在每个数据类的Serialize( )函数中增加版本控制，以不可见的方式把旧版本文档转换为当前的格式。本例只所以与例 66分开只是为了明白易懂，事实上，不可能在自己的数据类中加入Serialize( )函数却不增加版本控制功能。

步骤

1. 增加数据类的版本号

如例66一样准备好数据类。然而，在此又另外串行化一个变量，跟踪数据类版本的变量，并初始化该变量为1。

```
void CWzdInfo1::Serialize(CArchive& ar)
{
   int version=1;
   CObject::Serialize(ar);
   int nCount;
   if (ar.IsStoring())
   {
      // version
      ar << version;

      :  :  :
   }
   else
   {
      // version
      ar >> version;

      :  :  :
   }
}
```

2. 更新旧数据类

1) 过了几个星期后，决定在其以前的数据类中添加一个新变量。添加之后，必须修改该数据类的Serialize( )函数。首先是加大版本号。

```
void CWzdInfo::Serialize(CArchive& ar)
{
```

```
int version=2; //<<<<<<< incremented from 1
CObject::Serialize(ar);
if(ar.IsStoring())
{
   :  :  :
```

2) 在Serialize( )函数的存储部分，添加新变量到后面。不要把它和别的变量混淆起来。

```
// version
ar << version;

// data
:  :  :

// new with version 2
ar << m_nModNum;
```

3) 在Serialize( )函数的调出部分，若文档的版本与当前该类的版本匹配，则调出该变量。同保存一样，在不与其他变量混淆的情况下，在调出部分的后面作这种测试。

本例要求每个数据项在每次调出和保存时严格地以同样的顺序进行。

```
 // version
   ar >> version;

   // data
   :  :  :

   // new with version 2
   if (version>=2)
   {
      ar >> m_nModNum;
   }
}
```

4) 若文档是旧的版本，该变量没有从文档中获得它的值，它将相应地保持用户在构造函数中初始化给它的值。若文档是当前版本，该变量将从存档中得到它的值。关于使用版本控制的数据类的详细情况，参阅本节的“清单 —— 数据类”。

说明

- 每个数据类负责一个属于自己的版本号。串行化的文档应有多个版本号而不仅仅是一个。
- 本例所提供的方法只允许增加新成员变量到数据类串行化的后面。既不能删除一个变量，也不能与已有的变量混淆。若想删去一个旧的变量，可以使用如下的 case表达式。

```
switch (version)
case 1:
   // old way
case 2:
   // new way, with the sky being the limit
case 3:
```

使用这种方法，格式之间几乎没有什么共同点。这使得其他程序员很难读懂和使用这段代码。

CD说明

本例的工程是“ Serialization ”。首先在WzdDoc.cpp中的Serialize( )函数上设置断点，然后单击Test/Wzd菜单命令填充文档，最后单击File/Save开始串行化工作。

清单——数据类

```
#ifndef WZDINFO2_H
#define WZDINFO2_H


class CWzdInfo2 : public CObject
{
public:

enum STATES {
   OLD,
   NEW,
   MODIFIED,
   DELETED
   };

   CWzdInfo2();
   CWzdInfo2(CString sName,int nVersion);

   void Set(CString sName,CString sComment,int nVersion, int nState);

   // misc info
   CString m_sName;
   CString m_sComment;
   int m_nVersion;
   int m_nState;

   // new with version 2
   int m_nModNum;

   void Serialize(CArchive& archive);

   CWzdInfo2& operator=(CWzdInfo2& src);

};
#endif

// WzdInfo2.cpp : implementation of the CWzdInfo class
//

#include "stdafx.h"
#include "WzdInfo2.h"
```

```
//////////////////////////////////////////////////////////////////////////
// CWzdInfo

CWzdInfo2::CWzdInfo2()
{
    m_sName=_T("");
    m_sComment=_T("");
    m_nVersion=1;
    m_nState=CWzdInfo2::NEW;

    // new with version 2
    m_nModNum = 0;
}

CWzdInfo2::CWzdInfo2(CString sName,int nVersion) :
    m_sName(sName),m_nVersion(nVersion)
{
    m_sComment=_T("");
    m_nState=CWzdInfo2::OLD;

    // new with version 2
    m_nModNum = 0;
}

void CWzdInfo2::Set(CString sName,CString sComment,int nVersion, int nState)
{
    m_sName=sName;
    m_sComment=sComment;
    m_nVersion=nVersion;
    m_nState=nState;
    m_nModNum = 0;
}

void CWzdInfo2::Serialize(CArchive& ar)
{
    int version=2;
    CObject::Serialize(ar);
    if(ar.IsStoring())
    {
        // version
        ar << version;

        // data
        ar << m_sName;
        ar << m_sComment;
        ar << m_nVersion;
        ar << m_nState;

        // new with version 2
```

```
        ar << m_nModNum;

    }
    else
    {
        // version
        ar >> version;

        // data
        ar >> m_sName;
        ar >> m_sComment;
        ar >> m_nVersion;
        ar >> m_nState;

        // new with version 2
        if (version>=2)
        {
            ar >> m_nModNum;
        }
    }
}

CWzdInfo2& CWzdInfo2::operator=(CWzdInfo2& src)
{
    if(this != &src)
    {
        m_sName = src.m_sName;
        m_sComment = src.m_sComment;
        m_nVersion = src.m_nVersion;
        m_nState = src.m_nState;
    }
    return *this;
}
```

13.8 例70 串行化多态类

目标

串行化一系列未知类型的数据类实例。

策略

在前一个例子中，串行化了一系列的类，但它们的类型是相同的。因为它们的类型相同，使用简单的new操作符可以重新创建每个数据实例。

```
while (nCount-- > 0)
{
    CWzdInfo2 pInfo=new CWzdInfo2; <<all instances of CWzdInfo2
    pInfo->Serialize(ar);
```

```
    m_WzdInfo2List.AddTail((CWzdInfo2*)pInfo);
}
```

但是如果数据包括一些基类的多态实例，前面的方法就无法构造这个数据类。

为了串行化多态类，和前面的例子一样，从 CObject派生每个数据类。然后，在每个数据类中加入两个宏：DECLARE_SERIAL( )和IMPLEMENT_SERIAL( )。这些宏在数据表中添加一个新的重载操作符，从而允许在不知道其派生类的情况下进行串行化。

注意 只有在数据类不包含派生于CObject的成员变量时，该方法才可行。例如，若数据类使用CList来维护一个列表，就不能使用这种方法。

步骤

串行化一个多态类

1) 从CObject派生一个数据类，并为它添加一个 Serialize( )函数(见例66)。

2) 在每个数据类的.h文件中包含DECLARE_SERIAL( )宏。

```
class CWzdInfo : public CObject
{
public:

    DECLARE_SERIAL(CWzdInfo)
        :   :   :
```

3) 在每个数据类的.cpp文件中包含IMPLEMENT_SERIAL( )宏。

```
//////////////////////////////////////////////////////////////////
// CWzdInfo
IMPLEMENT_SERIAL(CWzdInfo, CObject, 0)
CWzdInfo::CWzdInfo()
{
}
```

4) 确认这个数据类也有一个不带参数的构造函数。 MFC使用CreateObject( )自动地创建该类的实例， CreateObject( )是找该类的不带参数的构造函数来创建实例的。

5) 在保存这个数据类到磁盘时，要使用＜＜重载操作符，而不用该类的 Serialize( )函数。

```
for (POSITION pos = m_WzdInfoList.GetHeadPosition(); pos;)
{
    CWzdInfo2 *pInfo = m_WzdInfoList.GetNext(pos);
    ar << pInfo;
}
```

<<操作符不仅调用数据类的Serialize( )函数，而且同时串行化类的名称，以便在下次用来重新构造该类的实例。这个操作符在 DECLARE_SERIAL宏中加入。

6) 在从磁盘中调出数据类时，要使用 >>重载操作符。

```
CObject* pInfo;
while (nCount-- > 0)
{
    ar >> pInfo;
    m_WzdInfoList.AddTail(pInfo);
}
```

>>操作符使用<<操作符保存的信息创建对象。因此，不必知道哪个派生类被创建。

说明

- IMPLEMENT_SERIAL有一个称为模式数(schema number)的参数，可以用来作为数据类的当前版本号。使用下面查询以获得模式数：

```
UINT Version = ar.GetObjectSchema( );
```

然而，这个模式数只在用户串行化类时使用 >>和<<重载操作符时才有效。保存数据类的当前版本号的一般方法如例子69所示。若使用那种方法，则只须设置模式数为 0，以后再也不存取它即可。但是，若确实要用它，必须保证版本号与宏中的 VERSIONABLE_SCHEMA进行或操作。如下例所示，否则，改变版本号将引发一个存档异常错误。

```
IMPLEMENT_SERIAL (CWzdInfo, CObject, VERSIONABLE_SCHEMA|0);
```

- CObject类添加了每个数据类都要重载的Serialize( )函数。另一方面，DECLARE_SERIAL和IMPLEMENT_SERIAL宏只是添加了本例中需要的>>和<<重载成员函数。

CD说明

本例子的工程是“ Serialization ”。首先在WzdDoc.cpp中的Serialize( )函数上设置断点，然后，单击Test/Wzd菜单命令填充文档，最后单击File/Save开始串行化工作。

13.9 例71 串行化数据集

目标

利用MFC数据集类(如CList、CArray等)的内置串行化功能，使得更容易地串行化数据集。

策略

在前例中，当串行化一个数据集时，在集合中循环，每次单独地串行化一个元素。然而，每个MFC数据集有它自己的Serialize( )函数，可以自动地串行化集合元素。本例子中，我们将使用Serialize( )函数。

步骤

1. 串行化一个数据集

为了串行化数据集，只须调用它的 Serialize( )函数就可以了。

```
void CWzdInfo1::Serialize(CArchive& ar)
{
   CObject::Serialize(ar);
   if (ar.IsStoring())
   {
      m_WzdInfo2List.Serialize(ar);
   }
   else
   {
      m_WzdInfo2List.Serialize(ar);
```

```
    }
}
```

不需要做其他修改，就可以串行化数据集到磁盘上了。然而，数据集中的每个数据类只是简单地作为内存块被复制，精心为数据类设计的 Serialize( )函数并没有被调用。若希望Serialize( )被调用，则需要重载数据集类的SerializeElement( )。

2. 重载SerializeElement( )

1) 为了重载数据集合类的SerializeElement( )，需要在数据类的.h文件中加入如下声明：

```
void AFXAPI SerializeElements(CArchive& ar,
    CWzdInfo2** ppElements, int nCount);
```

2) 使用以下语句实现SerializeElement( )函数。

```
void AFXAPI SerializeElements(CArchive& ar,
    CWzdInfo2** ppElements, int nCount)
{
    CWzdInfo2 *pWzdInfo2;
    for (int i = 0; i < nCount; i++)
    {
        if (ar.IsStoring())
        {
            pWzdInfo2 = *(ppElements + i);
        }
        else
        {
            pWzdInfo2 = new CWzdInfo2;
            *(ppElements + i) = pWzdInfo2;
        }
        pWzdInfo2->Serialize(ar);
    }
}
```

对于应用程序来说，只须简单地用保存在数据集中的数据类的类名代替 CWzdInfo2。

3) 若用户处理的是一系列如前一个例子所讨论的异态的数据类，则要用如下的语法来实现SerializeElement( )函数。

```
void AFXAPI SerializeElements(CArchive& ar,
    CWzdInfo2** ppElements, int nCount)
{
    CWzdInfo2 *pWzdInfo2;
    for (int i = 0; i < nCount; i++)
    {
        if (ar.IsStoring())
        {
            pWzdInfo2 = *(ppElements + i);
            ar << pWzdInfo2;
        }
        else
        {
            ar >> pWzdInfo2;
            *(ppElements + i) = pWzdInfo2;
```

```
        }
    }
}
```

该函数与上一步所用函数的唯一区别在于 <<和>>操作符的使用。

有关实现SerializeElement( )函数的数据类的详细清单，参见本节的“ 清单 —— 数据类 ”。

说明

- SerializeElement( )是一个模板重载函数。若调用参数与数据集类要求的参数不一致，编译器不会显示语法错误，当然数据集类也不会调用该函数。
- 本例主要针对一个模板集类，它与一般集类(如CObList)的唯一区别是SerializeElement( )实现语法不一样，这一点留给读者去考虑。

CD说明

本例的工程称为“ Serialization ”。首先在WzdDoc.cpp中的Serialize( )函数上设置一个断点。然后单击Test/Wzd菜单命令填充文档。最后单击 File/Save菜单命令开始串行化工作。

清单 —— 数据类

```
#ifndef WZDINFO1_H
#define WZDINFO1_H

#include "afxtempl.h"
#include "WzdInfo2.h"

void AFXAPI SerializeElements(CArchive& ar, CWzdInfo2** pWzdInfo2, int nCount);

class CWzdInfo1 : public CObject
{
public:
    CWzdInfo1();
    ~CWzdInfo1();

    // misc info
    CString             m_sGroupName;
    CString             m_sComment;
    CList<CWzdInfo2*,CWzdInfo2*> m_WzdInfo2List;
    void Serialize(CArchive& archive);

};
#endif

// WzdInfo1.cpp : implementation of the CWzdInfo class
//

#include "stdafx.h"
#include "WzdInfo1.h"
```

```
//////////////////////////////////////////////////////////////////////////
// CWzdInfo

void AFXAPI SerializeElements(CArchive& ar, CWzdInfo2** ppElements, int nCount)
{
   CWzdInfo2 *pWzdInfo2;
   for (int i = 0; i < nCount; i++)
   {
      if (ar.IsStoring())
      {
         pWzdInfo2 = *(ppElements + i);
      }
      else
      {
         pWzdInfo2 = new CWzdInfo2;
         *(ppElements + i) = pWzdInfo2;
      }
      pWzdInfo2->Serialize(ar);
   }
}

CWzdInfo1::CWzdInfo1()
{
   m_sGroupName=_T("");
   m_sComment=_T("");
}

CWzdInfo1::~CWzdInfo1()
{
   while (!m_WzdInfo2List.IsEmpty())
   {
      delete m_WzdInfo2List.RemoveHead();
   }
}

void CWzdInfo1::Serialize(CArchive& ar)
{
   int version=1;
   CObject::Serialize(ar);
   if (ar.IsStoring())
   {
      // version
      ar << version;

      // name and comment
      ar << m_sGroupName;
      ar << m_sComment;

      // other list
```

```
        m_WzdInfo2List.Serialize(ar);
    }
    else
    {
        // version
        ar >> version;

        // name and comment
        ar >> m_sGroupName;
        ar >> m_sComment;

        // other list
        m_WzdInfo2List.Serialize(ar);
    }
}
```

13.10 例72 访问ODBC数据库

目标

在应用程序中访问ODBC兼容的数据库。

策略

利用MFC的CDatabase和CRecordSet类访问ODBC数据库。CDatabase类用来打开数据库，CRecordSet类用来打开数据库中的表并读取其中的记录。

步骤

1. 设置应用程序

首先，要保证以下的包含文件在 Stdafx.h文件中。若在创建应用程序时，为数据库支持选择了Header files Only, 则AppWizard把这些包含文件添加到工程中。

```
#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>          // MFC ODBC database classes
#endif                      // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>         // MFC DAO database classes
#endif                      // _AFX_NO_DAO_SUPPORT
```

2. 使用ODBC数据库

1) 可以在任何地方创建一个数据库类的实例。但习惯上，这个实例应放在文档类中。

```
CDocument   m_WzdDocument;
```

2) 然后，使用Open( )成员函数打开数据库，从用户响应 New或者Open File命令。这些命令可以在文档类中进行处理。

```
if(!m_WzdDatabase.Open(
```

```
    NULL,                    // Data source name,
                             // NULL if defined in
                             // DSN below
    FALSE,                   // exclusive access,
                             // NOT SUPPORTED,
                           // should always be FALSE
    FALSE,                 // TRUE = read only access
    "ODBC;DSN=MS Access 97 Database")
  // connect string where DSN= is the data source name
  // found in ODBC32 utility, UID= is user id, PSW= is password
    )
    {
      AfxMessageBox("Failed to open database.");
    }
```

这里用的连接字符串基于 ODBC32设置应用程序中的数据库条目的名称。可以在系统的控制面板中找到ODBC32设置应用程序。使用ODBC32，可以增加和删除其他数据库。甚至可以使用ODBC去访问一个DAO数据库。

3) 一般也习惯于在文档类中创建一个捆绑函数以返回指向数据库的一个指针。

```
// Attributes
public:
  CDatabase *GetDatabase(){return &m_WzdDatabase;};
```

4) 可以在文档类的DeleteContents( )中关闭数据库。

```
// close database
m_WzdDatabase.Close();
```

3. 打开一个ODBC记录集

1) 使用ClassWizard创建一个派生于CRecordSet的一个新的记录集类。选择 CRecordSet将会使ClassWizard进入一个以前未见到的状态，它使 ClassWizard进入Record Set Wizard。首先提示选择想要创建的记录类的类型： ODBC或DAO。默认选项是ODBC。然后，从当前ODBC数据库列表中选择一个记录集要访问的数据库。 ClassWizard试图打开所选中的数据库。若成功，则提示选择哪个(些)表被包含在记录集中。选择多表将允许同时在多个表上进行操作，如进行连接时。最后， ClassWizard创建派生的记录集类。注意在此类中所选择表的每一字段都对应一个成员变量。当记录集打开后，在其中游览时，可以通过这些成员变量获得字段值。

2) 用如下方式打开数据库记录集类。

```
CWzdRecordSet wzdSet(GetDocument()->GetDatabase());
wzdSet.Open();
```

4. 使用ODBC记录集

1) 使用以下方式，滚动记录集中的记录。

```
while (! wzdSet.IsEOF())
{
  // column values can be accessed from record
  // set member variables like these:
  //    wzdSet.m_CustomerID
  //    wzdSet.m_CompanyName
  //    wzdSet.m_ContactName
```

```
    //     etc.

    wzdSet.MoveNext();
}
```

2) 使用以下方式，返回第一个记录。

```
wzdSet.MoveFirst( );
```

3) 使用以下方式，判断记录集是否可以被更新或添加。

```
// if we can't update or append records, leave now
    if (!wzdSet.CanUpdate() || !wzdSet.CanAppend())
        return;
```

4) 使用以下方式，在记录集中添加一个记录。

```
try
{
    // add new record
    wzdSet.AddNew();

    // initialize each field
    wzdSet.m_CustomerID="ABCDE";
    wzdSet.m_CompanyName="ABC Inc.";
    //etc.

    // update database
    wzdSet.Update();
}
catch (CDBException *e)
{
    // AddNew failed
    AfxMessageBox(e->m_strError);
    e->Delete();
}
```

注意，一个记录只有在调用了 CRecordSet::Update( )后才实际上被添加。

5) 使用以下方式，编辑数据库中的一个记录。

```
try
{
    wzdSet.Edit();

    // update affected fields
    wzdSet.m_CompanyName="ABCEF Inc.";
    wzdSet.m_ContactName="Frank";
    //etc.

    // update database
    wzdSet.Update();
}
catch (CDBException *e)
{
    // Edit failed
```

```
    AfxMessageBox(e->m_strError);
    e->Delete();
}
```

6) 使用以下方式，从数据库中删除一个记录。

```
try
{
    // delete record to which we ' re currently opened
    wzdSet.Delete();
}
catch (CDBException *e)
{
    // Delete failed
    AfxMessageBox(e->m_strError);
    e->Delete();
}
```

7) 使用以下方式关闭记录集。

```
wzdSet.Close( );
```

5. 用WHERE打开一个ODBC记录集

可以使用以下方式，用简单的WHERE SQL过滤器打开记录集，该过滤器过滤掉所有不匹配的记录。 在本例中，只有在[Country]字段上取值为UK的记录才可被访问。

```
wzdSet.m_strFilter = "[Country]='UK'";
if(!wzdSet.Open() || wzdSet.IsEOF())
{
    AfxMessageBox("Cannot find records.");
}

// scroll through records as above...
while (! wzdSet.IsEOF())
{
    wzdSet.MoveNext();
}
wzdSet.Close();
```

6. 用SELECT打开一个ODBC记录集

可以使用如下代码用完全的 SQL SELECT语句打开记录集。该语句允许被指定为 join、sort等。请注意， m_strFilter中的内容会被自动加到SQL命令后面。

```
wzdSet.m_strFilter = ""; // appended to the following!
if (!wzdSet.Open(AFX_DB_USE_DEFAULT_TYPE,
    //>>>>>>>>>> SELECT STARTS
    "SELECT [CustomerID], [CompanyName], [ContactName], \
    [ContactTitle], [Address], [City], [Region], [PostalCode], \
    [Country], [Phone], [Fax] \FROM [Customers] \
    WHERE [Country] = 'Mexico'") ||
    //<<<<<<<<<<< SELECT ENDS
    wzdSet.IsEOF())
{
```

```
    AfxMessageBox("Cannot find records.");
}
// scroll through records as above...
while (! wzdSet.IsEOF())
{
    wzdSet.MoveNext();
}
wzdSet.Close();
```

7. 创建ODBC事务

一些数据库操作可能要求一次访问几个数据库表。这时，一个表变化要求别的表也作相应改变。在这种情况下，若在改变一个表时发生了错误，用户必须取消这种变化，还要相应地取消在其他表中所作的改变。用数据库术语，称为事务恢复。 CDatabase的BeginTrans( )、CommitTrans( )和RollbackTrans( )成员函数支持事务恢复。

```
// Create a transaction we can rollback
GetDocument()->GetDatabase()->BeginTrans();
try
{
    // perform transactions on several database tables/records
    wzdSet.Open();
    wzdSet.Update();
    :  :  :
    // No exception occurred so we can commit
    // the previous transactions
    GetDocument()->GetDatabase()->CommitTrans();
}
catch (CDBException *e)
{
    // An exception occurred, rollback the transaction
    GetDocument()->GetDatabase()->Rollback();
    AfxMessageBox(e->m_strError);
    e->Delete();
}
```

说明

- 用户也可用CDatabase::OpenEx( )打开一个数据库。然而，这样做，用户必须保证省略掉ODBC，否则，部分或全部连接字符串的打开将失败。若数据库要求有用户 ID和口令，则可以在连接字符串中指定这些值。若省略掉，则系统在运行时要求用户输入这些值，如果找不到指定的数据库，它也会提示用户。然而，若使用 OpenEx( )函数，也可以设置选项以决定是要显示还是不显示这种提示要求。若只是想简单地在不告知用户前提下，判断数据库是否存在的话，则后一种选项是有用的。
- 当使用CDatabase::OpenEx( )时，系统只通过数据库异常 (database exceptions)来报错。因此必须使用try/catch语句来捕获任何错误。
- ODBC驱动程序允许应用程序访问任何第三方供应商的数据库管理系统 (DBMS)。然而，这种灵活性带来一个性能隐患。因为 ODBC在处理之前，用户数据库请求首先编码为一个SQL文本命令，驱动程序必须把它转换成一种内部命令。若移植性对应用程序来

说非常重要，则可能别无选择。若在最近一段时间，使用的是同一供应商的产品，那么在可能的情况下，应直接使用供应商所提供的数据库 API，是以牺牲移植性换取速度。DAO的效率要比ODBC高些，只是因为它不是使用 SQL与管理系统交流，而是直接封装进微软Jet Engine数据库管理系统的API。

CD说明

在CD上执行该工程时，首先在 WzdView.cpp中的OnTestWzd( )上设置断点。然后单击Test/Wzd菜单项，一步步地观察ODBC数据库的访问。

13.11 例73 访问DAO数据库

目标

访问一个由Microsoft's Jet数据库管理系统生成的数据库，也即 Microsoft Access应用程序生成的数据库。

策略

使用MFC的CDaoDatabase和CDaoRecordSet类打开数据库并访问其记录。这两个类的许多功能与MFC访问ODBC数据库的类是一样。但是他们有较高的性能和灵活性。

步骤

1. 设置应用程序

首先必须保证以下的包含文件在 Stdafx.h中。若在创建应用程序时，为数据库支持选择了Header File Only，则AppWizard将把这些包含文件加到工程中。

```
#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>          // MFC ODBC database classes
#endif                      // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>         // MFC DAO database classes
#endif                      // _AFX_NO_DAO_SUPPORT
```

2. 使用DAO数据库

1) 可以在应用程序的任何地方创建 CDaoDocument类的实例。但习惯上，这实例应放到文档类中。

```
CDaoDocument m_WzdDocument;
```

2) 然后，用Open( )成员函数打开数据库，以响应 New或Open File命令，这些命令在文档类中进行处理。

```
try
{
   m_WzdDatabase.Open(
   "C:\\jeswanke\\examples\\Sampdata.mdb",    // Data source name,
                                              // NULL if defined
```

```
                                                    // in DSN = below
    FALSE,                                          // TRUE = exclusive
                                                    // access to database
    FALSE);                                         // TRUE = read only access
}
catch (CDaoException *e)
{
    AfxMessageBox(e->m_pErrorInfo->m_strDescription);
    e->Delete();
    return FALSE;
}
```

注意与ODBC打开数据库不同的是，用户可以直接指定要打开的 Access *.mdb文件名。并且这里支持以独占方式打开。

3) 一般也习惯于在文档类中创建一个捆绑函数以获取指向数据库的指针。

```
// Attributes
public:
    CDaoDatabase *GetDatabase(){return &m_WzdDatabase;};
```

4) 使用以下代码关闭数据库。

```
if (m_WzdDatabase.IsOpen())
    m_WzdDatabase.Close();
```

若用户试图关闭一个未打开的DAO数据库，系统将产生一个异常错误。

3. 使用DAO记录集

1) 使用ClassWizard创建一个从CDaoRecordSet派生的新的记录集类。选择CDaoRecordSet将会导致ClassWizard进入一种以前可能未见过的状态。它使 ClassWizard进入Record Set Wizard。首先要求选择要创建的记录集的类型： ODBC或者DAO。选择DAO，然后，系统将引导在磁盘中查找 *.mdb数据库文件。 ClassWizard将试图打开所选择的数据库，若成功，则提示选择记录集类中所要包含的表。选择多表，将允许同时在多个表上进行操作，如 Join。最后，ClassWizard创建这个数据集类。注意此类中所选择的表的每一字段都对应一个成员变量。当记录集打开后，在其中游览时，可以通过这些成员变量获取记录字段值。

2) 使用以下方式，打开记录集。

```
CWzdRecordSet wzdSet(GetDocument()->GetDatabase());
wzdSet.Open();
```

3) 使用以下方式，在记录集中滚动记录。

```
while (! wzdSet.IsEOF())
{
    // values can be accessed from record set member variables:
    //    wzdSet.m_CustomerID
    //    wzdSet.m_CompanyName
    //    wzdSet.m_ContactName
    //    etc.

    wzdSet.MoveNext();
}
```

4) 使用以下方式，返回第一个记录。

```
wzdSet.MoveFirst( );
```

5) 使用CDaoRecordSet类的FindFirst( )和FindNext( )函数查找某一字段值与某一规则匹配的记录。在本例中，我们将查找字段 [country]取值为UK的所有记录。

```
if (wzdSet.FindFirst("[Country]='UK'"))
{
   while (wzdSet.FindNext("[Country]='UK'"))
   {
   }
}
```

6) 使用以下方式，判断记录集中的表是否可以修改或添加。

```
// if we can't update or append records, leave now
if (!wzdSet.CanUpdate() || !wzdSet.CanAppend())
   return;
```

7) 使用如下方式，在表中添加一个记录。

```
try
{
   // add new record
   wzdSet.AddNew();

   // initialize each field
   wzdSet.m_CustomerID="ABCDX";
   wzdSet.m_CompanyName="ABC Inc.";
   //etc.

   // update database
   wzdSet.Update();
}
catch (CDaoException *e)
{
   // AddNew failed
   AfxMessageBox(e->m_pErrorInfo->m_strDescription);
   e->Delete();
}
```

注意必须调用Update( )之后记录才实现被添加到数据库中。

8) 使用以下方式编辑记录集中的一个记录。

```
try
{
   wzdSet.Edit();

   // update affected fields
   wzdSet.m_CompanyName="ABCEF Inc.";
   wzdSet.m_ContactName="Frank";
   //etc.

   // update database
   wzdSet.Update();
}
catch (CDaoException *e)
{
```

```
    // Edit failed
    AfxMessageBox(e->m_pErrorInfo->m_strDescription);
    e->Delete();
}
```

9) 使用以下方式从记录集中删除一个记录。

```
try
{
    // delete record to which we're currently opened
    wzdSet.Delete();
}
catch (CDaoException *e)
{
    // Delete failed
    AfxMessageBox(e->m_pErrorInfo->m_strDescription);
    e->Delete();
}
```

10) 使用以下方式，关闭记录集。

```
wzdSet.Close( );
```

若记录集类是在栈上创建的，返回时，将析构类实例并关闭记录集。

4. 用WHERE打开一个DAO记录集

使用以下方式，用简单的 SQL WHERE 语句打开一个数据库表。本例中，只有在[Country]字段上取值为UK的记录才被访问。

```
wzdSet.m_strFilter = "[Country]='UK'";
wzdSet.Open();
if(wzdSet.IsEOF())
{
    AfxMessageBox("Cannot find records.");
}

// scroll through records as above...
while (! wzdSet.IsEOF())
{
    wzdSet.MoveNext();
}
wzdSet.Close();
```

5. 用SORT打开一个DAO记录集

1) 使用以下方式，用简单的 SQL SORT语句打开一个表。本例中，记录集中记录是按[ContactName]字段取值的字母大小排列。

```
wzdSet.m_strSort = "[ContactName]";
wzdSet.Open();
:  :  :
```

6. 用Select语句打开DAO记录集

1) 使用以下方式，用完全的SQL SELECT语句打开一个数据记录集。注意， m_strFilter或m_strSort中的内容将被增添到SQL语句的后面。

```
wzdSet.m_strFilter = ""; // appended to the following!
wzdSet.m_strSort = "";
```

```
wzdSet.Open(AFX_DB_USE_DEFAULT_TYPE,
   "SELECT [CustomerID], [CompanyName], [ContactName], \
   [ContactTitle], [Address], [City], [Region], [PostalCode], \
   [Country], [Phone], [Fax] \
   FROM [Customers] WHERE [Country] = 'Mexico'");
if (wzdSet.IsEOF())
{
   AfxMessageBox("Cannot find records.");
}

// scroll through records as above...
while (! wzdSet.IsEOF())
{
   wzdSet.MoveNext();
}
wzdSet.Close();
```

7. 创建DAO事务

一些数据库操作可能同时需要访问多个表，一个表的变化相应地要求另一个的表改变。在这种情况下，若在改变一个表时出现了错误，则要求取消这种变化，并要取消在其他表中的改变。用数据库术语，称为“事务恢复”。CDaoDatabase类的BeginTrans( ) CommitTrans( )和RollbackTrans( )成员函数支持这种事务恢复。

```
// Create a transaction we can rollback
GetDocument()->GetDatabase()->m_pWorkspace->BeginTrans();
try
{
   // perform transactions on several database tables/records
   wzdSet.Open();
   wzdSet.Update();
   // Success
   GetDocument()->GetDatabase()->m_pWorkspace->CommitTrans();
}
catch (CDaoException *e)
{
   // An exception occurred, rollback the transaction
   GetDocument()->GetDatabase()->m_pWorkspace->Rollback();
   AfxMessageBox(e->m_pErrorInfo->m_strDescription);
   e->Delete();
}
```

说明

- 对于经常使用用完整定义的 SQL语句打开数据库的情况下，可以考虑在记录集类中添加一个Open( )成员函数。这类函数可能叫 SelectByCountry (arg)，它总是使用WHERE '[Country]' = arg的方式打开记录集。

- MFC类库也包括CDaoQueryDef，它允许预定义一个SQL查询在以后使用。
- MFC类库也包括CDaoTableDef，允许创建一个新的DAO数据库表。

CD说明

在CD上运行该工程时，首先在 WzdView.cpp中的OnTestWzd( )上设置断点，然后单击Test/Wzd菜单命令，一步步观看DAO数据库的访问。