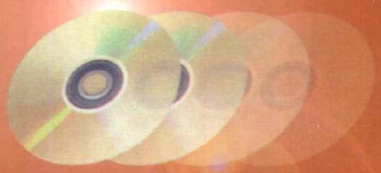

本书含光盘一张

不必艰难求索 本书最佳选择

胡峪 刘静 编著

Visual C++

VC++ 高级编程技巧与示例

西安电子科技大学出版社
http://www.xduph.com

内 容 简 介

本书是《VC++应用技巧与示例》的姊妹篇，其内容与《VC++应用技巧与示例》互补。本书从实用的角度出发，全面、系统地讲解了高级数据库编程技术、各种类型的 Winsock 网络通讯编程技术、Internet 编程技术、RPC 与串口通讯技术、多媒体播放技术和 DirectX 动画技术。在讲解这些技术时，我们力求以最简单的语言阐述技术的背景和实现方法；然后，讲解相关的函数；最后，对每一种编程技术我们都精心设计了一个示例。此外，本书还提供了若干实用的 C++类，以这些类为基础，读者可以非常容易地编制各种复杂的 VC++应用程序，简化编程，节省时间。书中的所有示例和类均由笔者精心设计并调试通过，希望本书能够在《VC++应用技巧与示例》的基础上，引导读者进一步学习更多、更加实用的编程技术。最后，希望读者喜欢我们为大家提供的 C++类并根据自己的需要对这些类进行扩展。

本书可供计算机程序员和计算机爱好者使用。

图书在版编目（CIP）数据

VC++高级编程技巧与示例/胡峪，刘静编著. —西安：西安电子科技大学出版社，2001.5

ISBN 7-5606-1010-2

Ⅰ.V… Ⅱ.① 胡… ② 刘… Ⅲ.C 语言－程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2001)第 12228 号

责任编辑 毛红兵 李纪澄

出版发行 西安电子科技大学出版社(西安市太白南路 2 号)

电 话 (029)8227828 邮 编 710071

http://www.xduph.com E-mail: xdupfxb@pub.xaonline.com

经 销 新华书店

印 刷 西安长青印刷厂

版 次 2001 年 5 月第 1 版 2001 年 5 月第 1 次印刷

开 本 787 毫米×1092 毫米 1/16 印张 26

字 数 613 千字

印 数 1～4 000 册

定 价 40.00 元 (含光盘)

ISBN 7-5606-1010-2/TP・0492

目　录

第一篇　数　据　库

第二篇 网络与计算机间通讯

第三篇　多　媒　体

第一篇 数 据 库

第1章

基于ODBC的数据库应用程序

1.1 概 述

ODBC[①]是可以用于访问不同数据源的SDK[②]，应用程序通过ODBC SDK调用ODBC驱动程序管理器，ODBC 驱动程序管理器将调用传递给数据库驱动程序，数据库驱动程序利用SQL[③]语言同DBMS[④]进行通讯。

通过ODBC和MFC[⑤]，你可以利用CDatabase类连接到数据库和进行事务处理，然后利用CRecordset类获取数据库中的记录，利用CRecordView 类显示数据，调用ODBC SDK和执行SQL查询。

利用CRecordView 进行数据库应用程序设计时，虽然利用CRecordView类编制很少的代码就能够实现显示、更改数据的功能，但是，使用CRecordView 的局限性也是十分明显的：一次只能对一个记录进行操作，因此不是非常方便。此外，利用CRecordView 时，当数据库中添加、删除或更改了某些字段后，同时需要更改视图类和记录集，需要重新编译应用程序，灵活性很差，因此本章所有示例均脱离了常规的采用CRecordView 类进行数据库显示和操作的方法，采用列表框等控件进行数据库应用程序设计，提高了应用程序的灵活性。

此外，通过MFC编程时，在使用数据源之前用户必须注册，本章将讲解如何调用ODBC SDK在应用程序中自动注册数据源，免去用户手工注册的麻烦。

当需要同时更新多个记录时，可以使用事务处理，所谓事务处理就是指对数据库的一

① ODBC (Open DataBase Connectivity)：开放式数据库互连。

② SDK (Software Developing Kit)：软件开发包。

③ SQL (Structured Query Language)：结构化查询语言。

④ DBMS (DataBase Management System)：数据库管理系统。

⑤ MFC (Microsoft Fundation Class)：微软基础类。

系列数据的更新。当使用事务进行数据处理时，可以保证所有对数据库的操作都具有原子性，即：对数据库的一系列操作要么全部都执行，要么一个都不执行。在对规范化数据库进行操作时，我们经常涉及多个表，例如：某客户在商店购买了商品，需要从银行账户中支出一部分钱款到商店的账户；在进行数据库操作时，希望用户账户和商店账户的更新保持同步，如果出现用户的账户已经更新，而商店的账户更新失败时，应该取消所有操作。利用事务处理可以实现上述功能，因此，事务处理可以提高数据库操作的效率和安全性，通常将取消事务中所有操作的过程称做事务的回滚 (Roll Back)。

1.2 利用列表框浏览记录

1.2.1 概述

使用列表框控件显示数据库可以使用户同时浏览数据库中的所有记录。此外，还可以创建一个类，在类中使用列表控件显示指定记录集中的数据，从而使应用程序与数据库相对独立，当数据库更改后，应用程序不需要进行任何改动，因此应用程序的实用性很好。

1.2.2 技术核心

1. 基本思路

在这里，我们将创建一个可以显示指定记录集中数据的列表视图类，通过该类，可以实现记录集中所有数据的列表显示，使用该类将使数据库在更改之后不必重新编译程序。

创建该类需要完成的工作有：

(1) 创建一个 CListView 类的派生类。

(2) 在类中创建一个记录集对象，用于获取记录集中的数据。

(3) 修改该类的 OnInitialUpdate ( ) 函数：

- 调用 CRecordset::GetODBCFieldInfo ( ) 函数，获取记录集的字段信息，为列表视图添加各个列。
- 调用 CRecordset:: MoveFirst ( ) 函数将指针移动到第一个记录。
- 调用 CRecordset:: GetFieldValue ( ) 函数读取记录集指针所指向的记录的信息，填写列表视图类的各行、各列。

2. 将要用到的变量类型和函数：

下面我们来看一下将要用到的变量类型和函数：

(1) BOOL CRecordset::Open ( ) 函数：

该函数用于打开记录集，它的原型定义如下：

```
virtual BOOL Open( UINT nOpenType = AFX_DB_USE_DEFAULT_TYPE,
                   LPCTSTR lpszSQL = NULL, DWORD dwOptions = none );
throw (CDBException, CMemoryException );
```

该函数的各个形参定义如下：

● nOpenType：该成员变量指定记录集的打开方式，其默认值为 AFX_DB_USE_DEFAULT_TYPE，此时记录集将按照快照的方式打开。

你也可以指定打开方式。记录集的打开方式有：

CRecordset::dynaset：记录集将按照动态集的方式打开。按照这种方式打开的记录集可以向前后两个方向滚动，记录集的成员和顺序在记录集打开时就已经确定，其他用户对数据库的更改在进行获取（fetch）操作之后可以被反映出来。

CRecordset::snapshot：记录集将按照快照的方式打开。按照这种方式打开的记录集可以向前后两个方向滚动，记录集的成员和顺序在记录集打开时就已经确定，其他用户对数据库的更改无法被看到，除非关闭记录集然后再重新打开记录集。

CRecordset::forwardOnly：按照这种方式打开的记录集只能以只读的方式打开，并且只能向前滚动。

● lpszSQL：该参数为一个字符串，它可以存储的数据如下：

NULL，空字符串；

表的名字；

一个 SQL 的 SELECT 语句，可以带 WHERE 或 ORDER BY 条件；

一个 SQL 的 CALL 语句，通过 CALL 语句可以调用 SQL 存储过程。

● dwOptions：打开记录集的选项，默认值为 none。该参数还可以取以下值，也可以取以下值进行或运算后的组合值：

CRecordset::none：没有选项，该选项同其他选项互斥，不能进行组合，此时，记录集可以通过 Edit 或 Delete 操作进行更新，也可以通过 Addnew 添加新的记录，对记录的多行存储不进行优化，对多行读取不支持。在记录集中滚动时，删除操作将不被取消。此外，书签也不被支持，而对已经更改过的记录的检测将被执行。

CRecordset::appendOnly：此时不允许编辑和删除记录，只允许添加新记录。该选项同 CRecordset::readOnly 互斥。

CRecordset::readOnly：此时记录集为只读的。该选项同 CRecordset::appendOnly 互斥。

CRecordset::optimizeBulkAdd：使用该选项时，将利用 SQL 语句进行大量数据的一次性存取，未调用 ODBC API 中 SQLSetPos( ) 函数时才可以使用该函数。该选项同 CRecordset::useMultiRowFetch 互斥。

CRecordset::useMultiRowFetch：此时将执行数据的批量读取，通过一次读取操作就可以获取多行记录，从而避免一行一行地读取记录，提高了效率。该选项同 CRecordset::optimizeBulkAdd 互斥。当该选项有效时，CRecordset::noDirtyFieldCheck 自动被设成有效，此时，双缓冲区功能无效；对于只能向前滚动的记录集，选项 CRecordset::useExtendedFetch 将自动有效。

CRecordset::skipDeletedRecords：该选项有效时，在记录集中滚动的、所有已经删除的记录都将被忽略。该选项在只向前滚动的记录集中无效。

CRecordset::useBookmarks：该选项有效时，在记录集中允许使用书签。书签使数据的读取变慢，但提高了查找数据的效率。该选项对只向前滚动的记录集无效。

CRecordset::noDirtyFieldCheck：该选项关闭对修改过的记录的自动检查功能，这将提

高效率，但是你必须手工地调用 SetFieldDirty ( ) 函数和 SetFieldNull ( ) 函数来标记已经修改过的记录。当 CRecordset::useMultiRowFetch 选项有效时，CRecordset::noDirtyFieldCheck 选项将自动有效。但是，SetFieldDirty ( ) 函数和 SetFieldNull ( ) 函数在数据批量读取（bulk row fetching）时使用。

CRecordset::executeDirect：不使用预先准备的 SQL 语句，当不打算调用 CRecordset::Requery ( )函数时，置该选项有效。

CRecordset::useExtendedFetch：此时将调用 SQLExtendedFetch ( ) 函数，而非 SQLFetch ( )函数。该选项用于只能向前滚动的记录集的批量读取。在使用只能向前滚动的记录集时，如果 CRecordset::useMultiRowFetch 选项有效，则该选项自动有效。

CRecordset::userAllocMultiRowBuffers：用户将为数据分配缓冲区。如果你想自己分配缓冲区，则同时置 CRecordset::useMultiRowFetch 选项有效。当该选项有效而没有设 CRecordset::useMultiRowFetch 选项有效时将导致错误。

● CRecordset::Open ( ) 函数的调用模式如下：

```
// 使用默认 SQL 语句，使用书签，不进行对修改过的记录的检测：
    m_set.Open( CRecordset::snapshot, NULL,
                CRecordset::useBookmarks |
                CRecordset::noDirtyFieldCheck );

// 以动态集的方式打开记录集，采用 SQL 查询：
    m_set.Open( CRecordset::dynaset,
                _T( "Select L_Name from Customer" ) );

// 使用默认值：
    m_set.Open( );
```

(2) CRecordset::IsOpen ( )函数

使用该函数判断记录集是否已经打开。当记录集打开时，该函数返回 1；否则，返回 0。

(3) CODBCFieldInfo 结构

该结构专门用于存储由 CRecordset::GetODBCFieldInfo ( ) 函数读取的记录集中各个字段的信息，该结构的定义如下：

```
struct CODBCFieldInfo
{
    CString m_strName;
    SWORD m_nSQLType;
    UDWORD m_nPrecision;
    SWORD m_nScale;
    SWORD m_nNullability;
};
```

其中：

m_strName 为字段名称；

m_nSQLType 为记录集的 SQL 数据类型。可以是 ODBC SQL 数据类型，也可以是由驱动程序指定的 SQL 数据类型。ODBC SQL 数据类型的列表可以参考 OBDC SDK 附录 D 中“SQL Data Types”专题，而驱动程序指定的 SQL 数据类型则可以查询数据库驱动程序的文档。

(4) CRecordset::GetODBCFieldInfo ( )函数

通过该函数可以获取记录集中各个字段的信息。

该函数的原型为：

```
void GetODBCFieldInfo( LPCTSTR lpszName, CODBCFieldInfo& fieldinfo );
throw( CDBException );
void GetODBCFieldInfo( short nIndex, CODBCFieldInfo& fieldinfo );
throw( CDBException );
```

其中：

lpszName 为字段的名称；

fieldinfo 为指向 CODBCFieldInfo 结构的指针；

nIndex 为从 0 开始的字段序号。

(5) CDBVariant 类

CDBVariant 类没有基类，该类是专门用于存储 ODBC 中各种类型数据的通用类，其作用与 OLE 自动化中的 COleVariant 类相似。其中，CDBVariant::m_dwType 成员变量为 DWORD 型变量，用于标识存储于 CDBVariant 类中数据的类型。在读取 CDBVariant 类中的数据之前，必须先根据该成员的取值判断 CDBVariant 类中存储的是什么类型的数据，然后再从 CDBVariant 类中对应的成员变量中获取数据。m_dwType 的取值和含义及对应的 CDBVariant 类成员变量如表 1 - 1 所示。

表 1 - 1　m_dwType 的含义

m_dwType 的取值	对应的数据类型	对应的 CDBVariant 类成员变量
DBVT_NULL	NULL	类中所有成员变量无效
DBVT_BOOL	BOOL	m_boolVal
DBVT_UCHAR	unsigned char	m_chVal
DBVT_SHORT	short	m_iVal
DBVT_LONG	long	m_lVal
DBVT_SINGLE	float	m_fltVal
DBVT_DOUBLE	double	m_dblVal
DBVT_DATE	TIMESTAMP_STRUCT	m_pdate
DBVT_STRING	CString	m_pstring
DBVT_BINARY	CLongBinary	m_pbinary

表 1 - 1 中，TIMESTAMP_STRUCT 是一个结构，定义在 SQLEXT.H 中，用于存储日期。TIMESTAMP_STRUCT 结构的定义如下：

```
typedef struct tagTIMESTAMP_STRUCT
    {
        SWORD year;
        UWORD month;
        UWORD day;
        UWORD hour;
        UWORD minute;
        UWORD second;
        UDWORD fraction;
    } TIMESTAMP_STRUCT;
```

而 CLongBinary 类用于存储和处理数据库中的大二进制数据 (binary large objects，或 BLOBs)，其成员变量如下：

DWORD CLongBinary::m_dwDataLength：该成员变量以字节的形式存储数据的长度，该数据的 HGLOBAL 句柄存储在 m_hData 成员变量中。m_dwDataLength 所指定的数据长度可能会比实际分配的内存小。

HGLOBAL CLongBinary::m_hData：该成员变量存储的是 BLOB 的 HGLOBAL 句柄。

(6) CRecordset::GetFieldValue ( )函数

该函数的原型定义如下：

```
void GetFieldValue(LPCTSTR lpszName, CDBVariant& varValue, short nFieldType=DEFAULT_FIELD_
TYPE );
throw( CDBException, CMemoryException );

void GetFieldValue( short nIndex, CDBVariant& varValue, short nFieldType = DEFAULT_FIELD_TYPE );
throw( CDBException, CMemoryException );

void GetFieldValue( LPCTSTR lpszName, CString& strValue );
throw( CDBException, CMemoryException );

void GetFieldValue( short nIndex, CString& strValue );
throw( CDBException, CMemoryException );
```

其中：

lpszName 为字段名称；

varValue 为指向 CDBVariant 类对象的指针，该类对象存储的是当前记录、当前字段的值；

nFieldType 为字段的 ODBC C 语言数据类型，当该参数的取值为 DEFAULT_FIELD_TYPE 时，GetFieldValue ( )函数将自动把数据类型从 SQL 数据类型转换成 C 语言数据类型(见表 1 - 2)。当该参数为其他取值时，你可以人为指定数据类型或选择一个适当的数据类型，例如：你可以将任何类型的数据按照 SQL_C_CHAR 类型存储。

表 1 - 2　ODBC C 语言数据类型与 SQL 数据类型对照表

ODBC C 语言数据类型	SQL 数据类型
SQL_C_BIT	SQL_BIT
SQL_C_UTINYINT	SQL_TINYINT
SQL_C_SSHORT	SQL_SMALLINT
SQL_C_SLONG	SQL_INTEGER
SQL_C_FLOAT	SQL_REAL
SQL_C_DOUBLE	SQL_FLOAT 或 SQL_DOUBLE
SQL_C_TIMESTAMP	SQL_DATE SQL_TIME SQL_TIMESTAMP
SQL_C_CHAR	SQL_NUMERIC SQL_DECIMAL SQL_BIGINT SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR
SQL_C_BINARY	SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY

(7) CRecordset::Close ( )函数

调用该函数关闭记录集。

1.2.3　使用列表视图类显示记录集数据的示例

具备了上述知识后，编写一个用列表视图类显示数据的应用程序就非常简单了。下面我们来看一下具体步骤。

第一步：创建应用程序框架。

创建一个名为 ODBCList 的工程。

在 AppWizard 的第一步中选择单文档界面。

在 AppWizard 的第二步中选择无数据库支持，如图 1 - 1 所示。

在 AppWizard 的第六步，也就是 AppWizard 的最后一步中选择 CListView 作为视图类的基类，如图 1 - 2 所示。

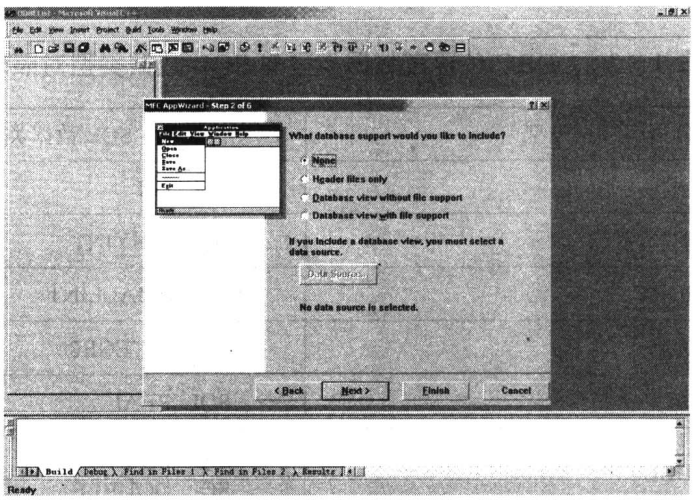

图 1-1 在 AppWizard 的第二步中选择无数据库支持

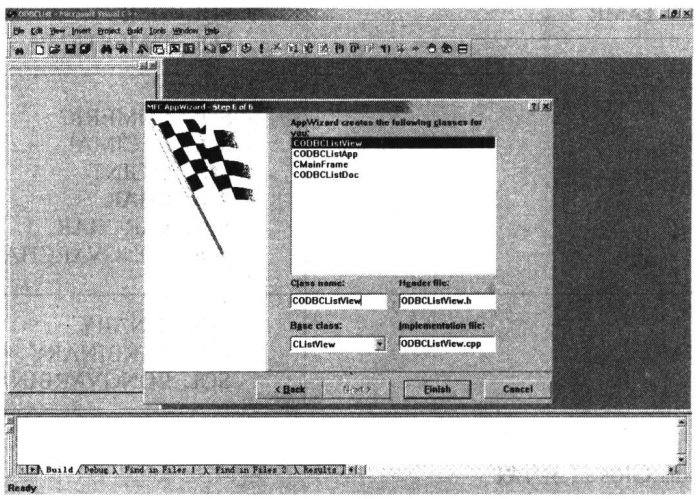

图 1-2 在 AppWizard 的第六步中选择 CListView 做视图类的基类

第二步：注册数据源。

本书提供一个名为 Company 的 Access 数据库。在使用数据库之前，必须进行注册，注册数据源的方法如下：

(1) 进入控制面板，选择管理工具。在管理工具中选择图标“数据源 (ODBC)”，激活 ODBC 数据源管理器。

注意：

本示例代码基于 Windows 2000 编制，因此在注册数据源时，与 Windows 98 略有不同，在 Windows 98 的控制面板中，直接选择“数据源 (ODBC)”即可激活 ODBC 数据源管理器。

(2) 在 ODBC 数据源管理器中，选择“User DSN”页面。

(3) 点击“添加”按钮，此时弹出一个名为“创建新数据源”的对话框，如图 1 - 3 所示。

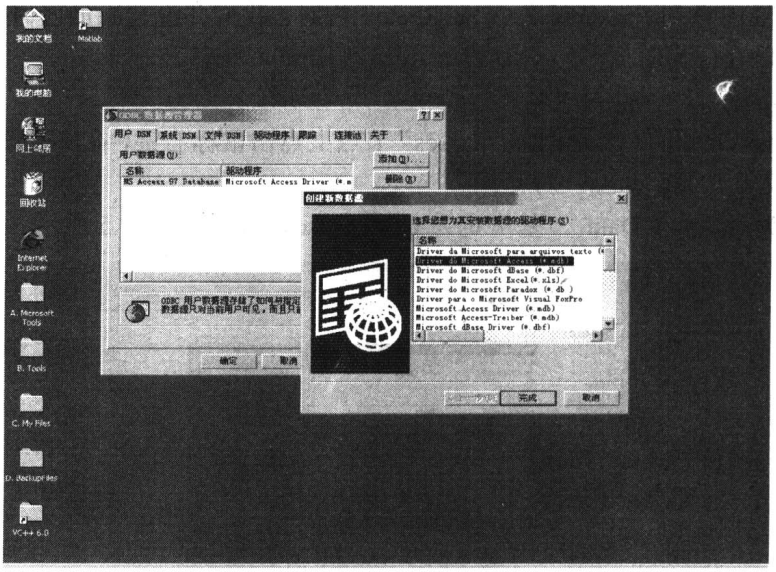

图 1 - 3　在“创建新数据源”对话框中选择驱动程序

(4) 在“创建新数据源”对话框中，选择“Driver do Microsoft Access”列表项，然后选择“完成”按钮。

(5) 在随即弹出的名为“ODBC Microsoft Access 安装”的对话框中，输入数据源名称和说明，点击“选择”按钮，利用“选择数据库对话框”找到并打开 Company.mdb，如图 1 - 4 所示。

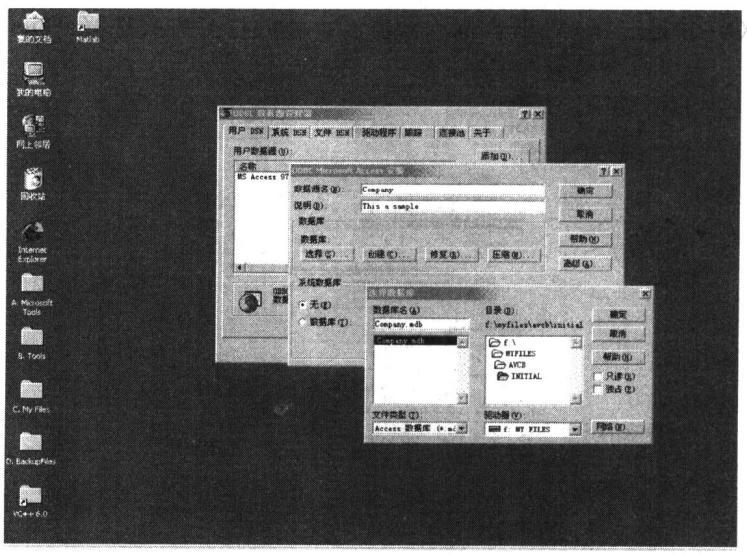

图 1 - 4　选择并打开数据库文件

(6) 点击“ODBC Microsoft Access 安装”对话框和“创建新数据源”对话框中的“确定”按钮，完成数据源的注册。

第三步：为工程添加记录集。

也许你已经注意到了，在创建 ODBCList 工程时，我们并没有添加任何数据库支持。实际上，我们可以在任何时候利用 ClassWizard 添加数据库支持和记录集。下面我们来添加数据库的记录集。

(1) 在 ClassView 的“ODBCList classes”节点上点击鼠标右键，在快捷菜单中选择“Add new class”，激活“New class”对话框。

(2) 将新类命名为 CData ，基类定为 CRecordset 类，如图 1 - 5 所示。

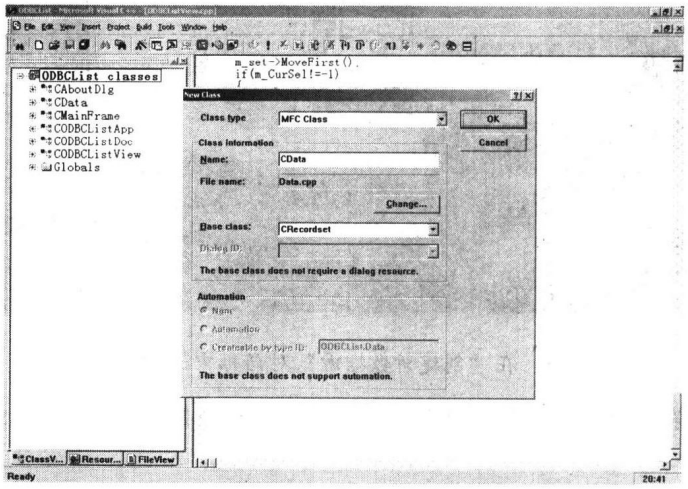

图 1 - 5 创建一个 CRecordset 的派生类

(3) 点击“OK”按钮，随即将弹出一个名为“Database Options”的对话框，在该对话框中选择 ODBC，并且选择“Company”数据源，表的打开方式选择“Dynaset”，如图 1 - 6 所示。

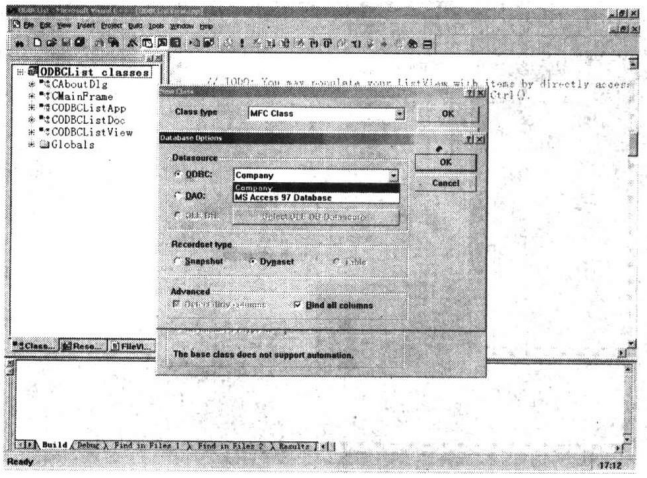

图 1 - 6 设置记录集参数