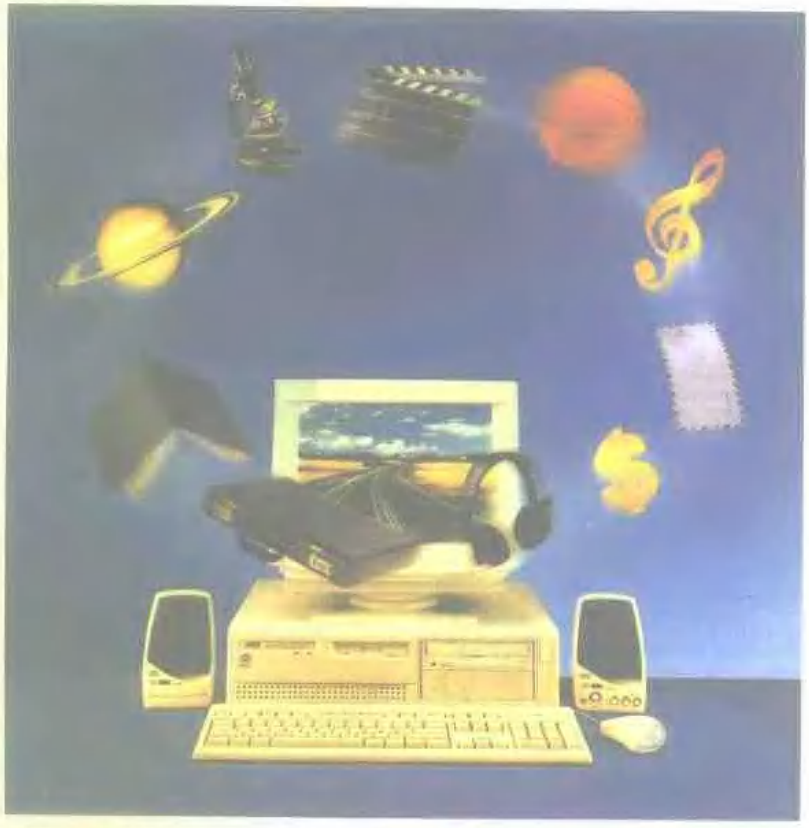
软件开发文集

第十一辑　第十二辑

《软件开发文集》编委会 编



- 用 MFC 编写 Windows 95 程序（Ⅵ～Ⅷ）
  - 对话框、特性页、控件
  - 文档 / 视图的结构
  - 打印及打印预览
- OLE 和 COM 如何解决部件软件设计问题（下）
- 略谈一些使用 OLE DB 接口的数据库的 COM 方法
- Microsoft ActiveX ——为 Internet 和 Intranet 开发高效的新内容和应用

科学出版社

软件开发文集

第十一　十二辑

《软件开发文集》编委会　编

科学出版社
1997

内 容 简 介

本书是为软件开发人员和计算机用户编写的《软件开发文集》系列书的第十一、十二辑，围绕着专题内容，向读者提供了开发技术管理规范、软件开发管理、开发平台、应用设计策略等方面的技术资料。

本书的重点文章是：用 MFC 编写 Windows 95 程序（Ⅵ～Ⅷ）；略谈使用 OLE DB 接口的数据库的 COM 方法；Microsoft ActiveX——为 Internet 和 Intranet 开发高效的新内容和应用等。

本书适用于软件开发人员和计算机用户学习参考。

图书在版编目(CIP)数据

软件开发文集　第十一、十二辑/《软件开发文集》编委会编.
北京：科学出版社，1997.5
ISBN 7-03-005801-1

Ⅰ.软…　Ⅱ.软…　Ⅲ.软件开发-文集　Ⅳ.TP311.52-53

中国版本图书馆 CIP 数据核字(96)第 25670 号

科学出版社出版
北京东黄城根北街 16 号
邮政编码：100717
新世纪印刷厂印刷
新华书店北京发行所发行　各地新华书店经售
*
1997 年 5 月第　一　版　　开本：787×1092　1/16
1997 年 5 月第一次印刷　　印张：11 1/2
印数：1—3000　　字数：260000

定价：18.00 元

编委会名单

顾　问

杜家滨　冯玉琳　秦人华　林资山

主　编

郑茂松

副主编

（按姓氏笔画为序）

刘晓融　李　浩　廖恒毅

编　委

（按姓氏笔画为序）

王淑兰　巴建芬　华京梅　查良钿

编者的话

近年来，我国的计算机产业发展迅速，PC 机销量已居世界第六位。同时，我国有一批人数众多、技术水平相当不错的软件开发人员，他们分布在全国各地、各个行业、各个领域。然而，由于计算机产业，特别是软件产业充满了变化，极富动态性，加之，我国幅员辽阔，通信技术又落后于较发达的国家，从而，软件开发人员面临着这样的问题：了解最新技术动态不及时、不方便，获取最新技术信息比较困难，所得到的信息往往不够全面、完整和深入，相互之间缺乏交流，因此，重复开发、重复别人走过的弯路的情况屡见不鲜。另一方面，全世界积累的资料浩如烟海，价格亦非常昂贵，鉴于国内经济承受能力有限，许多技术信息不能及时获得。有鉴于此，我们组织有关人员翻译、编写了以技术专集为特色的《软件开发文集》系列书，该套书将陆续出版、发行，及时向用户提供最新技术资料，力图为解决上述问题做一点尝试。

《软件开发文集》系列书的宗旨是为软件开发人员开辟一个相互交流经验和体会的园地，提供一条了解新技术、新工具、新产品的渠道，设立一个探讨软件开发理论和开发技术的论坛，帮助软件开发人员更快、更直接地获得有关软件开发的信息，提高软件开发人员的技术水平和工作效率，充分发挥软件的作用，提高软件的使用价值，便进我国软件产业与世界同步发展。

《软件开发文集》是面向软件开发人员的中、高档次的技术专集，所收集的文章向软件开发人员提供了最新科技动态，以及开发技术规范、开发经验和技巧、软件开发管理、应用设计策略、开发平台方面的内容等，同时，根据用户在软件使用和开发过程中遇到的难点、疑点给予一定的解答和帮助。《软件开发文集》资料主要来源于微软公司的“Microsoft System Journal”，“Developer News”，“Microsoft Development Library”，“TechNet”等，并收入国内软件开发人员撰写的有关文章。欢迎国内广大软件开发人员为《软件开发文集》撰稿，介绍自己的经验、心得、体全。特别要注重文章内容道合我国国情。

目前，在国内面向计算机最终用户的图书、杂志丰富多彩，但面向软件开发人员的图书、杂志却寥寥无几。我们期望《软件开发文集》能够弥补这方面的空白，并且不辜负广大用户的期望，把撰好学术方向，确定好信息服务和技术支持的层次和深度，把《软件开发文集》办成深受广大用户喜发的丛书。

由于时间仓促，《软件开发文集》难免存在这样或那样的问题，敬请广大的用户及读者提出宝贵意见和建议，以利改进，为我国软件产业的发展作为应有的贡献。

《软件开发文集》编委会

1996.3.25

目　录

1

用 MFC 编写 Windows 95 程序(VI)

——对话框、特性页、控件

如果你用 Win32 编过特性页，你就知道要使特性页正确地工作是多么困难！有了 MFC，编写特性页将是小菜一碟，因为 MFC 可为你处理大多数特性页通知。一旦你用 MFC 实现了特性页，你就再也不会用 SDK 去实现它了。

许多程序员都喜欢 MFC 的一个优点：其对话框和控件使用面向对象的结构。用户要想在屏幕上弹出一个对话框，不用使用::DialogBox 成员函数。用户可以直接从 CDialog 类中派生出来，只需创建一个 CDialog 类的对象，其余的乱七八糟的工作就交给 MFC 去做。CDialog 类继承 CWnd 类的基本功能并添加一些自己的成分用来实现对话框的行为。对话框中的控件都是对象，由成员函数来操纵它们。对话数据交换和有效性检查技术使用户方便地从控件中存取数据。

如果一个对话框中控件太多而使对话框显得拥挤，用户可以使用标签对话页(tabbed dialog page)方便地把对话框转换成特性页。MFC 中的 CPropertySheet 类和 CPropertyPage 类支持特性页。在 MFC4.0 中，MFC 特性页奇迹般地实现了 Windows 95 和 Windows NT 3.51 中的许多功能。如果以前你曾经用 Win32 API 编写过特性页程序，你就知道使特性页正确工作是很费劲的。但在 MFC 中，这只是小菜一碟，因为 MFC 框架替你完成了大部分的特性页说明工作。用 MFC 方式使用特性页就不必像 SDK 方式那样进行细节性的说明。

这一部分，我将介绍 MFC 的对话类和特性页类，以及将子窗口控件(如按钮和列表框等)自动打包的类。还将教你如何创建一个对话框和特性页，如何创建带有控件的对话框或者特性页的界面，如何创建能实现其自身行为的可移动和自包含的控件类，如何修改内置的控件类以增加功能，以及如何不费吹灰之力地创建一个能属主画(owner-draw)按钮。如果这是你第一次接触面向对象的 Windows 对话框和控件，那么就请坐好，慢慢体会，你将受益匪浅。

(一)对话框和 CDialog 类

对话框有两种类型：模态(modal)对话框和非模态(modeless)对话框。模态对话框在父程序做续前需要一个响应，即接态对话框在它未收到第一次响应之前不允许进入父应用程序的其他部分。非模态对话框不能阻止父程序的运行。也就是在进入父程序的其他

部分之前不需要一个响应，即非模态对话框在其父程序在前台运行时自己可以浮在后台。模态和非模态对话框的功能都封装在 MFC 的 CDialog 类中，CDialog 类的构造者接受两中变量：字符串的或者整型的资源标识符(resourced ID)和一个标识对话框的父 CWnd 指针(可为空)。资源标识符标识对话框的模板正如 SDK 应用中，一个对话框的设计是由一个包含在应用的资源文件中的对话框模板定义的。当对话框对象在内存中被实例化后，应用就通过调用对象的 DoModal 函数或 Create 函数在屏幕上显示一个对话框。DoMadal 函数创建一个模态对话框，Create 函数创建一个非模态对话框。

用户可以通过在对话框模板中说明对话框风格为 DS_SYSMODAL 把一个模态对话框转换成系统模态对话框。在 16 位的 Windows 系统中，这种方法创建一个使系统中其他窗口的输入都无效的模态对话框。在 Win32 环境下，系统模态对话框不影响其他进程的窗口，并且与标准的模态对话框稍有差异。Windows 自动地置系统模态对话框的 WS_EX_TOPMOST 位，使得此对话框总是显示在其他窗口的顶端。

一个普通的对话框可以直接通过实例化 CDialog 类来实现。图 1 给出了一个简单的 About 对话框的描述，该对话框只包含三个控件：两个用以显示程序信息的静态文本控件和一个 OK 按钮。显示这个对话框需要两行代码：

```
Cdialog dlg ("About",this);
dlg.DoModal ();
```

```
About DIALOG 32,32,186,98
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "About Object Builder"
FONT 8,"MS Sans Serif"
BEGIN
    CTEXT           "Object Builder Version 1.0",-1,12,28,162,10
    CTEXT           "Copyright ©1995 by me",-1,12,40,162,10
    DEFPUSHBUTTON   "OK",IDOK,68,72,50,14
END
```

图1 About 对话框的描述模板

"About"参数是模板的名字，指明此对话框的拥有者窗口，如果说明为 NULL(或者省略这个参数，系统默认为 NULL)，系统将自动将应用的最顶端的窗口作为此对话框的父窗口。因为 DoModal 函数创建一个模态对话框，所以只有对话框被释放后函数才返回。其返回值被传送到撤销对话框的 EndDialog 函数作为参数。除非用户使用自己的对话类并调用自己的 EndDialog 函数，否则，返回值总是 IDOK 或 IDCANCEL。到底是什么要看对话框是用 OK 按钮还是用 CANCEL 按钮释放的。即使对话框不包含 Cancel 按钮，DoModal 函数也可以返回 IDCANCEL 值，因为按 ESC 键和单击 CANCEL 按钮是等价的。

注意：用户不必要自己写一个处理对话框消息的过程，CDialog 类为用户提供了一个默认的对话过程，它知道当单击某个按钮是该如何动作。如果利用 DDX/DDV(后面将解释)，该过程甚至知道如何响应 WM_INITDIALOG 消息来初始化对话框的控件，并且知

道单击 OK 按钮时如何检索数据。

用户要想改变对话框响应 OK 按钮和 CANCLE 按钮的动作，或者给其他控件加些代码，就必须派生出自己的对话类。CDialog 类提供了三个关键函数：OnInitDialog，OnOk，OnCancle，可以被重载以便初始化对话框，响应 OK 按钮和 CANCLE 按钮。虽然每个函数都对应于不同的消息，用户不需要一个处理它们的消息映射，因为 CDialog 类把这三个函数定义为虚的。下面，我们将介绍使用这些函数的一个例子。但我们首先讨论一下出现在对话框中的各种控件和它们怎样与对话类接口。

(二)MFC 的控件类

从1.0开始，Windows 就给普通用户界面对象(如按钮，圆钮，列表框，卷滚条等)提供了预定义窗口类。由子这些控件与其他窗口的关系是父子关系，所以被称为子窗口控件，这些特殊的窗口控件使用户每次需要一个按钮或一个转滚条时不必重新生成。MFC 把内置控件类型重新封装并把它们放在 C++的控件类中，如表1所示。WNDCLASS 列的项是该控件所基于的预定义窗口类。MFC4.0把控件类扩充了，使之包含 Windows 95的基本控件。

表1 MFC 的控件类

控件类型	MFC 类	WNDCLASS
按钮、圆钮、复选框	Cbutton	"button"
分组框		
静态文本控件	Cstatic	"static"
编辑控件	Cedit	"edit"
列表框	CListBox	"listbox"
组合框	ComboBox	"combobox"
卷滚条	CScroll	"Scrollbar"

使用 MFC 控件类的一个好处是，每个类都提供操纵那种类型各种控件的成员函数。使用编辑控件时，为了限制最多只能输入32个字符，C 程序员可以通过发送一个 EM_LIMITTEXT 消息来实现，如下：

```
SendMessage (hwndEditCtrl,EM_LIMITTEXT,30,0);
```

假设 ctlEdit 是一个 CEdit 类型的对象，MFC 程序员得像下面这样写：

```
ctlEdit. Limit Text(32);
```

使用 MFC 控件类的另一个好处是可以使用这些控件类作为基类来定制自己的控件类。若想要一个只接收数字的编辑控件吗？只要从 CEdit 类派生一个 CNumberEdit 类，把 WM_CHAR 消息映射到 CNubmer::OnChar，再写一个 OnChar 函数用来传送数字关键字到基类，面忽略其他关键字。用这种方式来定制一个控件的行为和创建一个自包含的控件是很容易的。属主画(owner_draw)按钮就是一个自包含控件的很好的例子，属主画按钮是一个能自己画自己而不需要它的父窗口来画的按钮。MFC 甚至能使控件响应自己的

说明，当要建立一个行为独立于父类的可重用控件类时，这是一个很好的特点。

有两种方式从 MFC 控件类创建一个控件：

方法1　是先创建一个控件对象，然后调用对象的 Create 或 CreateEx 函数，这种方法当控件在顶层窗口创建时常常采用。下面的描述用来从 MFC 的 CButton 类创建一个标识为“Reset”的按钮。

```
CButton ctlPushButton;
ctlPushButton.Create("Reset",
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        CRect (x,y,cx,cy),this,IDC_RESET);
```

方法2　在对话框中用得比较普遍。如果控件是从对话框模板创建的，就不必使用 Create 或 GreateEx 函数重新创建，如果控件是要用 MFC 的成员函数访问的，用户就得从相应的类创建对象，并把它和该控件联系起来。实现的一种简便方法是使用 CWnd::GetDlgItem 函数，该函数接收一个控件标识符，返回一个 CWnd 指针。为了保证访问安全，可以把 CWnd 指针强制转换成相应的控件类型。下面的描述是用来获得对话框中的一个按钮的 CButton 指针。

```
Cbutton *pPushButton=(CButton *) GetDlgItem(IDC_RESET);
```

现在 pPushButton 可以使用 CButton 类的成员函数来操纵该控件了。

GetDlgItem 函数返回的指针可能是临时的，不能存下来为以后使用。例如，不要指望在 OnInitDialog 函数中使用 GetDlgItem 函数获得的指针在 OnOk 中也能用，因为这个指针可能不再有效，原因是 MFC 利用消息间的空余时间来删除由 GetDlgItem 和其他函数创建的临时对象。一种解决办法是在自己的对话类中包含私有成员函数，返回对单个控件的引用。假设一个对话框包含两个编辑控件，标识符分别为 IDC_NAME 和 IDC_PHONE。如果对话类包含下面的成员函数：

```
CEdit& ctlName () {return (CEdit *) GetDlgItem (IDC_NAME);}
CEdit& ctlPhone () {return *(CEdit *) GetDlgItem (IDC_PHONE);}
```

可以这样访问控件：

```
ctlName ().LimitText (32);
ctlPhone ().LimitText (32);
```

另一种方法是在对话类中包含两个 CEdit 对象，并在 OnInitDialog 中用 CWnd::Attach 函数把它们连结到那两个编辑控件：

```
m_ctlName.Attach (::GetDlgItem (IDC_NAME));
m_ctlPhone.Attach (::GetDlgItem (IDC_PHONE));
```

现在 Cedit 访问编辑控件是无缝的：

```
m_ctlName.LimitText (32);
m_ctlPhone.LimitText (32);
```

因为 CEdit 对象是对话类的成员，当对话框被释放时使用者不必显示地删除它

们。

如果在对话框中使用一个派生控件类，简单地把控件连结到派生类的对象是不够的。如果把一个 CNumberEdit 控制连结到一个从对话模板创建的编辑控件对象，如下：

```
m_ctlPhone.Attach(::GetDlgItem (IDC_PHONE));
```

该控件将还是表现为一个标准的 CEdit 控件，因为 Attach 并不通过派生类的消息映射重发送该控件的消息。怎么办呢？不使用 Attach 函数，改用 MFC 的 SubclassDlgItem 函数把控件连结到一个 CNumEdit 对象：

```
m_ctlPhone.SubclassDlgItem (IDC_PHONE,this);
```

SubclassDlgItem 函数将控件子类化(subclass)来截断消息，现在该编辑控件就表现为 CNumEdit 风格了，因为 CNumEdit 类得到消息后将首先执行自己的处理过程。

子窗口通过发送 WM_COMMAND 消息与父窗口通信。在 Win32 中，WM_COMMAND 消息的 wParam 参数高位字包含一个通知码(notificationcode)，低位字包含控制的标识符，lParam 是控件的窗口句柄。通知码告诉父窗口产生该消息的事件：单击某按钮，或者是一个编辑控件中文本的改变等。按钮通知的标识符以 BN_前缀，其他通知类型还包括编辑控件的 EN_通知、列表框控件的 LBN_通知、组合框控件的 CBN_通知，静态控件不发送通知消息，卷滚条发送 WM_HSCROLL 和 WM_VSCROLL 消息，而不是 WM_COMMAND 消息。

MFC 为各种通知类型提供了消息映射宏，可以方便地把与控件有关的事件与对话类的成员函数联系起来。例如，下面的语句使用 ON_BN_CLICKED 宏把标识符为 IDC_RESET 的按钮与 CMyDialog 类中名为 OnReset 的成员函数联系起来。

```
BEGIN_MESSAGE_MAP (CMyDialog,CDialog)
    ON_BN_CLICKED (IDC_RESET,OnReset)
END_MESSAGE_MAP ()
CMyDialog::OnReset
{
    // Do something when Reset is clicked
```

MFC 文档中包含了控件消息的全部清单和相应的消息映射宏。

有时要求有一个控件来处理自己的通知消息。假设要写一个漫游硬盘目录结构的可重用的列表框，在 C 中，用来指明被单击的目录名的 LBN_DBLCLK 消息是由列表框的父窗口处理的，父窗口将对改变目录作出反应，修改列表框。但在 MFC 中，因为它用虚函数 OnChildNotify 把通知返回到控件，列表框自己能处理双击。下面是列表框处理 LBN_DBLCLK 通知的代码：

```
BOOL CMyListBox::OnChildNotify (UINT message,
                                WPARAM wParam,
                                LPARAM lParam,
                                LRESULT* pLResult)
{
    if (message == WM_COMMAND) {
      if (HIWORD (wParam) == LBN_DBLCLK) {
```

```
            .
            .// Respond to double clicks
            .
          return TRUE;
        }
      }
      return FALSE;
  }
```

MFC 也把 WM_MEASUREITEM，WM_DRAMITEM，WM_COMPAREITEM 和 WM_DELETEITEM 消息返回给产生这些消息的控件，使得属主画控制能自己画自己。这个功能很有用，因为它允许使用者完全按自包含方式写控制类。而自包含控制很容易转接到其他应用中去。这就是为什么要用面向对象编码的一个原因。

(三)简单小结

我们已经了解了对话框控件是怎样处理的，现在来看看怎样用上面说的编辑控件在对话框中输入名称和电话号码。假设对话模板已经创建，下一步是从 CDialog 类中派生出对话类。图2给出了派生一个 CMyDialog 类的代码。当创建了对话框并调用了虚函数 OnInitDialog 后，初始的名称和电话号码(如果有)从成员变量 m_strName 和 m_strPhone 中复制到编辑控件中。单击 OK 按钮后，对话框的 OnOk 函数又从控件中获得文本，并把它们复制到成员变量中去。CMyDialog 类没有重载 OnCancle 成员函数，因为 CDialog 类缺省的 OnCancle 函数(仅仅释放对话框并返回 IDCANCLE 给 DoMadal 函数)就已工作得非常好了。

注意：到 CString 类型变量 m_strName 和 m_strPhone 被声明为公共类型而非私有类型，这样对话框的父窗口也能从对话框中存取数据。假设主窗口类包含 CString 类型数组 m_strName 和 m_strPhone 用来存放名称和电话号码，还有一个 m_nCurrentEntry 数据成员说明当前的名称和电话号码。下面是激活对话框的代码：

```
CMyDialog dlg (this);
dlg.m_strName = m_strName[m_nCurrentEntry];
dlg.m_strPhone = m_strPhone[m_nCurrentEntry];
if (dlg.DoMadal () = = IDOKY;
  m_strName[m_nCurrentEntry] = dlg.m_strName;
  m_strPhone[m_nCurrentEntry] = dlg.m_strPhone;
}
```

名称和电话号码是通过把当前名称和电话号码复制到对话的 m_strName 和 m_strPhone 数据成员中来传递到对话框的。相反，通过把这些数据成员的值复制到当前名称和电话号码中，对话框检索名称和电话号码，但这要在单击 OK 按钮释放对话框时才执行。这种技术不必要像 SDK 应用中那样通常使用一个全局变量来传出或传入对话框的数据。当然，对话类的 m_strName 和 m_strPhone 数据成员必须声明为公共类型(或者在主窗口类中声明该对话类是友类)。

注意：上面的代码中没有说明对话框构造函数(constructor)中的模板名称，因为一个

以模板名称建立的 CMyDialog 构造函数是被包含在类的声明里的。MFC 程序员常用这种技术建立对话类和相应的对话模板间的联系。

```
Class CMyDialog : public CDialog
{
private:
    CEdit &ctlName() { return *(CEdit*) GetDlgItem (IDC_NAME);}
    CEdit &ctlPhone () {return *(CEdit*) GetDlgItem (IDC_PHONE);}
public:
    CString m_strName;
    CString m_strPhone;
    CMyDialog (CWnd* pParentWnd=NULL);
        CDialog ("NameAndPhoneDialog", pParentWnd) {}
    virtual BOOL OnInitDialog ();
protected:
    virtual void OnOk ();
};
BOOL CMyDialog::OnInitDialog
{
        // Call the base class first
        CDialog::OnInitDialog ();
        // Initialize the edit controls
        ctlName ().LimitText (32);
        ctlPhone ().LimitText (32);
        ctlName ().SetWindowText (m_strName);
        ctlPhone ().SetWindowText (m_strPhone);
        return TRUE;
}
void CMyDialog::OnOK ()
{
      // Retrieve the data from the edit controls
      ctlName ().GetWindowText (m_strName);
      ctlPhone ().GetWindowText (m_strPhone);

      // Call the base class to dismiss the dialog
      CDialog::OnOK ();
}
```

图2　一个简单的派生对话类

如果电话号码编辑控制是一个 CNumEdit 控制而不是标准的 CEdit 控制，CMyDialog 类有什么不同呢?图3给出了使用 CNumEdit 控制的 CMyDialog 类的原代码和 CNumEdit 类的原代码。CMyDialog 类声明了一个 CEdit 数据成员和一个 CNumEdit 数据成员，分别映射到 OnInitDialog 成员函数中的两个编辑控制。CEdit 控制用 Attach，CNumEdit 用 Subc lassDlgItem 以使 CNumEdit 消息映射。现在电话号码编辑控制只接收数字，因为只有数字(包括退格键，退格键也通过 OnChar 消息获得)被传送到基类作处理。

```
class CNumEdit : public CEdit
{
protected:
    afx_msg void OnChar (UINT,UINT,UINT);
    DECLARE_MESSAGE_MAP ()
};
BEGIN_MESSAGE_MAP (CNumEdit,CEdit)
     ON_WM_CHAR ()
END_MESSAGE_MAP ()
void CNumEdit::OnChar (UINT nChar,UINT nRepCnt,UINT nFlags)
{
        if (((nChar >='0') && (nChar <= '9')) || (nChar = = VK_BACK))
           CEdit::OnChar (nChar,nRepCnt,nFlags);
}
Class CMyDialog : public CDialog
{
private:
    CEdit m_ctlName;
    CNumEdit m_ctlPhone;
public:
    CString m_strName;
    CString m_strPhone;
    CMyDialog (CWnd* pParentWnd = NULL):
         CDialog ("NameAndPhoneDialog", pParentWnd) {}
    virtual BOOL OnInitDialog ();
protected:
    virtual void OnOK ();
};
BOOL CMyDialog::OnInitDialog ()
{
        CDialog::OnInitDialog ();
        m_ctlName. Attach (::GetDlgItem (IDC_NAME));
        m_ctlPhone. SubclassDlgItem (IDC_PHONE,this);
        m_ctlName. LimitText (32);
        m_ctlPhone. LimitText (32);
        m_ctlName. SetWindowText (m_strName);
        m_ctlPhone. SetWindowText (m_strPhone);
        return TRUE;
}
void CMyDialog::OnOK ()
{
        m_ctlName. GetWindowText (m_strName);
        m_cltPhone. GetWindowText (m_strPhone);
        CDialog::OnOK ();
}
```

图3 使用 CNumEdit 控件的派生对话类

(四)对话数据交换和数据有效性检查

这些例子充分描绘了 MFC 程序员如何处理模态对话框和控制。也稍微谈了一下怎样使用一个常规的控制类(如 CEdit)作为基类来派生出自己的控制类。但还有一个重要的疑惑没有解答:对话数据交换和数据有效性检查,通常称为 DDX 和 DDV。

当使用 Visual C++的 ClassWizard 构建一个对话类时,你可以使用 ClassWizard 创建对话数据成员并把它们与对话框中的控制联系起来。这很简单:从控制标识符列表框中选择一个标识符,单击 Add Variable 按钮,填上变量名和变量的特性。对话框创建后,各个控制便被初始化成相应成员变量的值。当单击 OK 按钮释放对话框时,各控制的值又被送回到各成员变量中去,并做简单的数据有效性检查。例如,在一个编辑控制中输入整数时超过规定的范围,将弹出一个消息框提示用户有错。消息框被释放后,对话框还在屏幕上并且输入焦点定在刚才出错的控制上,一个闪动的光标提示用户输入有效的数据。正是 MFC 框架的 DDX/DDV 机制使得 ClassWizard 能把对话框中的控制和对话类中的数据成员连结在一起。内幕是这样的:ClassWizard 在你的对话类中增加了一个 DoDataExchange 成员函数,它包含一个或多个与控制和数据成员有关的 DDX 或 DDV 函数调用。下面是只包含一个 DDX/DDV 连结的对话框,用户在编辑控制中输入一个整数,DoDataExchange 函数如下:

```
void CMyDialog::DoDataExchange (CDataExchange* pDX)
{
        DDX_Text (pDX,IDC_EDIT,m_nIntVal);
        DDV_MinMaxInt (pDX,m_nIntVal,0,100);
}
```

IDC_EDIT 是编辑控制的标识符,m_nIntVal 是与之相关的成员变量。传递给 DoDataExchange 的 pDX 参数是一个由 MFC 框架提供的指向 CDataExchange 对象的指针。CDataExchange 对象告诉 DDX_Text 和 DDV_MinMaxInt 数据传送的方向:是从数据成员传送到控制,还是从控制传送到数据成员。DoDataExchange 通常称为数据映射,因为它把对话框的控制和对话数据成员联系起来,就像消息映射把消息和成员函数联系起来一样。

ClassWizard 只创建一个数据映射,数据的采集是由框架完成的。当创建一个对话框后,CDialog::OnInitDialog 调用 UpdateData 函数初始化对话框的控制,该 UpdateData 函数是从一个对话对象从 CWnd 类中一 FALSE 参数继承过来的。反过来,UpdateData 函数创建一个 CDialogExchange 对象并调用对话类的 DoDataExchange 函数,给它传递一个指向 CDialogExchange 对象的指针。DoDataExchange 调用的每个 DDX 函数都用一个成员变量初始化一个控制。在上面的例子中,m_nIntVal 从一个整数转换成文本格式并通过 DDX_Text 函数把它复制到编辑控制中,以后,当用户单击 OK 按钮时,CDialog::OnOK 以 TRUE 参数调用 UpdateData 函数,此时 UpdateData 函数所做的工作与前面的刚好相反:从控制中读出一个文本串,转换成整数,存储在成员变量 m_nIntVal 中。MFC 的 DDV_MinMaxInt 函数检查 m_nIntVal 是否落在0～100范围内,包括0和100,如果是这样,那么,DoDataExchange 函数返回并且释放对话框,否则,CDataeExchange 对象调用

Fail 函数，弹出一个警告消息，对话框还保留在屏幕上，输入焦点在编辑控制上。

DDX_Text 和 DDV_MinMaxInt 函数只是 MFC 提供的多个 DDX/DDV 函数中的两个函数；表2和表3包含了所有的 DDX/DDV 函数。它们的句法包含在 AFXDD_.H 中，DDX_Text 函数把编辑控制中的文本数据和数字数据互相转换，它接收多种类型的数据。其他的 DDX 函数只接收一种类型的数据。例如，与复选框控制相关联的成员变量只接收整型数据：0，1，2，分别对应于复选框的三种状态：去选、复选、不定（不定状态只适用于三状态复选框）。DDX_Radio 把一组圆钮和某个成员变量相关。成员变量在进入该函数时保存一个从0开始计数的索引指明将要选择的圆钮，返回时保存当前选中的圆钮的索引。传递给 DDX_Radio 的控制标识符用来标识该组圆钮中的第一个圆钮。MFC 在运行时动态定位该组圆钮的最后一个圆钮，方法是一直扫描直到发现一个标识下一组圆钮开始的 WS_GROUP 属性。DDX_LBIndex，DDX_LBString 和 DDX_LBStringExact 函数使用整型值或 CString 类型的值从列表框中得到或设置选择。DDX_CBIndex，DDX_CBString，DDX_CBStringExact 函数使用整型值或 CString 类型的值从组合框中得到或设置选择。最后，DDX_Scroll 在一个卷滚条和一个整数型对话数据成员之间建立联系，数据成员的整数值对应于卷滚块（scroll thumb）的位置。

表2 DDX 函数

函 数	描 述
DDX_Text	在一个 BYTE，int，UNIT，long，DWORD，CString，float，或 double 型变量与一个编辑控制之间建立联系
DDX_Check	在一个 int 型变量与一个复选控制之间建立联系。变量值0，1，2，分别对应于复选框的三种状态：去选，复选，不定
DDX_Radio	在一个 int 型变量与一组圆钮控制之间建立联系。变量值从0开始计数，标识复选上的圆钮
DDX_LBIndex	在一个 int 型变量与一个列表框之间建立联系。变量值是当前选定的项的索引号
DDX_LBSting	在一个 CSting 型变量与一个列表框之间建立联系，当设置选项时不建立联系，变量值是当前选定的项的文本
DDX_LBStringExact	在一个 CSting 型变量与一个列表框之间建立联系，变量值是当前选定的项的文本
DDX_CBIndex	在一个 int 型变量与一个组合框之间建立联系。变量值是当前选定的项的索引号
DDX_CBSting	在一个 CSting 型变量与一个组合框之间建立联系，设置选项时不区分大小写，变量值是当前选定的项的文本
DDX_CBStringExact	在一个 CSting 型变量与一个组合框之间建立联系，变量值是当前选定的项的文本
DDX_Scroll	在一个卷滚条和一个对话数据成员之间建立联系，数据成员的整数值对应于卷滚块（scroll thumb）的位置

表3 DDV 函数

函 数	描 述
DDV_MinMaxByte	检查一个 BYTE 类型变量是否在规定范围内
DDV_MinMaxInt	检查一个 int 类型变量是否在规定范围内
DDV_MinMaxLong	检查一个 long 类型变量是否在规定范围内
DDV_MinMaxUInt	检查一个 UINT 类型变量是否在规定范围内
DDV_MinMaxDWord	检查一个 DWORD 类型变量是否在规定范围内
DDV_MinMaxFloat	检查一个 float 类型变量是否在规定范围内
DDV_MinMaxDouble	检查一个 double 类型变量是否在规定范围内
DDV_MaxChars	检查一个 CString 类型变量包含的字符是否超过规定的数目