万水计算机编程技术精品丛书

Visual C++ 6.0 编程技巧与实例分析

齐舒创作室　　编著

CD-ROM

中国水利水电出版社
www.waterpub.com.cn

万水计算机编程技术精品丛书

Visual C++ 6.0 编程技巧与实例分析

齐舒创作室　编著

毅　鸣　审校

中国水利水电出版社

内 容 提 要

Visual C++ 6.0 是支持 MFC 的强大而复杂的 Windows 32 位应用程序开发工具，相对其他开发工具有着先天的优势。然而，像任何一门技术一样，VC 也是易学难精，学到一定的程度后会出现一个难以突破的临界状态。本书的取向就是试图通过对一些精选的议题进行深入讲解和大量实例剖析，从而给处在临界状态及其左右的读者提供有效的帮助。

本书主要包括 Windows 平台编程剖析和 VC 编程技术两大部分。后者是本书核心内容，介绍了窗口、对话框、菜单、工具条和状态条、控件、动态链接库、自定义控件、ActiveX 控件、文件操作与注册表、数据库应用程序。另外，附录中还包括 Visual C++ 6.0 开发环境的操作技巧、Microsoft 基本类库速查等内容。

书　　名	**Visual C++ 6.0 编程技巧与实例分析**
作　　者	齐舒创作室　编著
审　　校	毅鸣
出版、发行	中国水利水电出版社(北京市三里河路 6 号 100044) 网址：www.waterpub.com.cn E-mail：sale@waterpub.com.cn 电话：(010)63202266(总机)、68331835(发行部)
经　　售	全国各地新华书店
排　　版	北京万水电子信息有限公司
印　　刷	北京天竺颖华印刷厂
规　　格	787×1092 毫米　16 开本　24.5 印张　550 千字
版　　次	1999 年 11 月第一版　1999 年 11 月北京第一次印刷
印　　数	0001—6000 册
定　　价	55.00 元(1CD，含配套书)

前　言

- Microsoft 的 Windows 在微机操作系统中毫无疑问地占据着霸主的地位
- C++是一门功能强大的面向对象编程语言
- MFC(Microsoft Foundation Class,微软基本类库)是微软为 C++程序员提供的一个面向对象的 Windows 编程接口
- 而 Visual C++是支持 MFC 的强大而复杂的 Windows 32 位应用程序开发工具,相对其他开发工具有着先天的优势

这些理由交织在一起,再加上人才招聘广告上大量出现 VC 之类的字眼,让人感到要成为一个真正的软件开发人员不能不去学习 Visual C++和 MFC。

于是,买了几本书开始学习 C++语言,或许已有的 C 语言基础帮你很快地有所领悟。往机器里装了一套 VC 后,利用几本入门书又熟悉了 Develope Studio 的开发环境,虽然还不知道 VC 在幕后搞了些什么名堂,你已用 AppWizard 生成了一个真正的但什么正事也不干的 Windows 程序。你觉得这还不够意思,于是开始向程序中添加自己的代码,程序"听话"时,能从中获得剂量不大的喜悦。但你觉得这还是不够意思,程序有时候也不"听话",别人编出来的那些精致程序的具体实现仍令人迷惑,自己编写的似乎都是些简单的玩具。总之,你发生了深入下去的兴趣,想要写出更"酷"的程序,但又感到了进阶的困难。如果你的情况是这样,那么你是本书的主要读者之一。

但是,像任何一门技术一样,VC 也是易学难精。所谓的可视化开发工具好象也并不那么可视,只要稍微想加入一点特性就必须亲手添加代码,要命的是 MFC 的机制令人觉得如此陌生,Application FrameWork(应用程序框架)的内部运作如此神秘,让人无从下手。不错,浅尝辄止的比较得出的结论当然是这样,只有真正掌握了它以后才能感觉到自如地驾驭程序的快乐,才能感觉到它的复杂是源自于它的强大。

VC 这种较难掌握的工具,学到一定的程度后会出现一个临界状态,大量的问题需要弄明白,大量的技巧需要学习,只有迎难而上,多做练习实践,并获取有效的帮助资料,才能闯过这一关。当解决的问题、学到的技巧、练过的实例、读过的资料积累到一定的数量时,质变就要发生,对这种工具的总体把握感油然而生,从此踏上走向 VC 大师的道路。否则,跨不过这一临界点,学习曲线将就此终止,半途而废在所难免。而本书的取向就是试图通过对一些精选的议题进行深入讲解和大量实例剖析,从而给处在临界状态及其左右的读者提供有效的帮助。

本书的内容

本书主要包括以下三方面内容:

开发平台

这方面内容主要在第一章中涉及。读者在学习时，一个较容易被遗忘的前提是施展手脚的舞台——Windows 平台，而通常只专注于 VC 这个工具的使用。如果不能对消息处理、图形设备接口(GDI)、基于资源的编程、DLL(动态链接库)等 Win32 编程模式心中有数，那么前进将是困难的。

编程技术

工欲善其事，必先利其器。本书从以下一些具体议题入手，通过对实例的详细讲解，帮助读者掌握 MFC 编程，包括：窗口问题、对话框、菜单、工具条和状态条、控件、动态链接库、自定义控件、ActiveX 控件、文件操作与注册表、数据库应用程序。这是本书的核心内容，占全书主要篇幅，即从第二章到第十三章。

开发手册

这方面的内容主要在本书的附录中介绍。主要包括 Visual C++6.0 开发环境的操作技巧、Microsoft 基本类库速查以及配套光盘介绍等，以尽可能为读者学习和使用 VC 提供帮助。

实际上，本书中的实例只是掌握 MFC 的手段，要直接指导编程工作恐怕多少例子也是不够的。最终的掌握应当是能通过这些例子举一反三做别人没做过的事情，做任何一本书都没有写过的事情。如果读者达到这一步，那么这本书对于你的价值就远远超过它本身的价格。

本书未涉及的话题

读者可能会发现本书对某些高级话题并未涉及，例如线程同步、异常处理，这是考虑到内容涵盖量与篇幅这对矛盾的平衡。虽然希望尽可能多地向读者提供实用的知识，但本书并不打算面面俱到(实际上那样只能对每个话题泛泛而谈)，所以尽量选取那些必不可少的基础知识和在通常编程工作中遭遇频率比较高的应用问题进行深入讨论，以满足大多数读者的需求。

第一章　Windows 基础

本书要讲述的是如何用 Visual C++ 6.0 及其所带的类库 MFC(Microsoft Foundation Class)进行编程。但一个容易被忽略的前提是,所编写的程序是 32 位 Windows 应用程序,这意味着开发平台和操作平台都是 Windows。"勿在浮沙筑高台",对于开发人员来说,如果对其程序开发运行的基础——Win32 不甚了解,是不可能编写出好的程序的。

任何事物都有其区别于其他事物的特征,使其成为它自己。Windows 是一种操作系统,这句话所表达的信息使 Windows 与所有非操作系统区别开来,说明它具有操作系统的共性,虽然这也是 Windows 区别于其他事物的重要特征,但却不是本书的论题。Windows 在操作系统中也有其自身的个性使其与其他操作系统相区别,32 位的 Windows(包括 Windows95/98/NT,对于程序员来说统称 Win32,当然也包括 Win32s)在这方面的个性才是本章要讨论的,并且我们只挑选对程序员有意义的特性。对最终用户来说,开机出现蓝天白云画面、点击"开始"菜单中的项启动应用程序、然后在窗口中进行自己的工作,这就是 Windows。而对于程序员来说,消息处理、资源、图形设备接口、虚拟内存管理、动态链接库、API 等才是 Windows,是这些决定了 Windows 编程模式。

1.1　Windows 编程模式

1.1.1　事件驱动与消息处理

Windows 是事件驱动的操作系统,这意味着操作系统的每一部分和其他部分——以及与应用程序之间是通过 Windows 消息进行通信的。产生事件以响应窗口之间被传递的消息,响应用户与操作系统以及应用程序之间的交互,这是 Windows 程序员的主要工作。

在传统的过程式编程方式中,开发者对程序流程有完全的支配权,他让程序以 B 函数→A 函数→C 函数的顺序执行,就决不可能出现 A 函数→B 函数→C 函数的执行顺序。而在 Windows 中,开发者根本无法确定这种顺序。程序必须响应用户的鼠标和键盘输入,鼠标和键盘可指向许多用户界面对象(如菜单和按钮),这些对象又可调用许多函数中的一个。用户是随心所欲的,其执行动作毫无次序可言,因此程序执行顺序无法确定。编写这些被调用来响应用户界面对象动作的函数就成了构造 Windows 应用程序的主要内容。

程序如何知道哪个用户动作(事件)发生了呢?通过消息。用户动作一产生,戒备森严的 Windows 就发现了,于是向应用程序发送一条消息(经常还带着参数),告诉它用户干了件什么事情。应用程序得到这条消息后到自己的消息处理器——窗口过程中寻找对应的处理函数。如果找到了,就执行该处理函数,这也许又要调用其他函数,也可能不要,这就是它自己的事了。如果没有找到,也就是说应用程序根本没有打算处理这条消息,那么将交给

Windows来做缺省处理。如果Windows也不处理,那这条消息就不会产生任何响应。

Windows应用程序与操作系统和其他应用程序之间能进行消息的发送和接收,并且能处理几百条消息,尽管一个典型的应用程序仅响应其中的一小部分。Windows还为应用程序消息提供称为消息队列的保留区。

每个当前执行着的进程都有自己的消息队列。由Windows或其他应用程序向你的应用程序发送的消息都要进入此队列等待处理。

SDK程序以特定的循环调用Windows消息,该循环称为消息循环,通常如下:

```
MSG msg;
while(GetMessage(&msg,NULL,0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

程序进行完初始化工作后,这个简单的循环将一直运行,直到接收到终止消息才停止。每次循环调用Windows的API函数GetMessage()取得在队列中等候的最后一条消息,该函数的原形在Windows.h文件中说明如下:

```
BOOL GetMessage(MSG FAR * lpmsg,HWND hwnd,UINT uMsgFilterMin,UINT uMsgFilterMax)
```

它将检索到的消息放到一个MSG结构中以备应用程序使用其中的信息。从MSG结构的定义可以看出它所包含的信息:

```
typedef struct tagMSG {
// msg
    HWND   hwnd;
    UINT   message;
    WPARAM   wParam;
    LPARAM   lParam;
    DWORD   time;
    POINT   pt;
} MSG;
```

其中:

hwnd——接收此消息的窗口句柄。

message——指定消息号。

wParam——指定消息所带的附加信息,其具体含义依赖于成员message的值。

lParam——指定消息所带的附加信息,其具体含义依赖于成员message的值。

time——指定消息发送的时间。

pt——指定消息发送时光标在屏幕坐标中的位置。

TranslateMessage()函数将虚键(virtual-key)消息转化为字符消息。如果要接受字符输入,消息循环就必须包含该函数。DispatchMessage()函数将消息送到hwnd指定的窗口的窗

口过程。这两个函数都只有唯一的参数,即 GetMessage()检索到的消息。除了用户动作产生消息,也可以用 SendMessage()发送消息。

1.1.2 图形设备接口

Windows 是图形化的操作系统,它是如何处理有关图形的操作的呢? 基本上是用 GDI(实际上,GDI 只是系统最基本最原始的一个图形输出库,除了它以外还有更高层的 OpenGL、DirectDraw 等)。

许多 MS - DOS 程序直接将显示内容写到视频缓冲区和打印机端口,这样做的缺点是必须为每一种视频卡和打印机提供驱动程序。能不能有某种部件充当显示内容和显示硬件的中介,使程序将显示内容以统一的方法传给它,然后它能将这些内容写到多种硬件上? 这样,应用程序就不必在乎显示硬件的种类,只要用这种统一的方法对显示内容进行输出。Windows 提供了它——图形设备接口(GDI),它充当了应用程序和显示硬件之间的抽象层,接管了与显示硬件打交道的工作。您的程序不再直接操作硬件,而是调用 GDI 函数,它们引用一种称为设备上下文(Device Context)的数据结构,由 Windows 将设备上下文映射到物理硬件并做相应的 I/O 处理。

在程序员看来,GDI 是由三百多个函数调用和一些相关的数据类型、宏和结构组成的。其函数调用大致可分为几大类:

1.获取(或创建)和释放(或清除)设备上下文的函数。如 GetDC()和 ReleaseDC()。

2.获取有关设备上下文信息的函数。如用 GetTextMetrics()来获取有关设备上下文中当前选择的字体的尺寸信息。

3.绘图函数。在所有前提条件都满足后,这些函数是真正重要的部分。如用 TextOut()在设备上下文中绘制文本;用 Line()和 Arc()函数绘制线条和弧形。

4.设置和获取设备上下文的函数。设备上下文的"属性"决定有关绘图函数如何工作的细节。如用 SetTextColor()来指定 TextOut()或其他文本输出函数绘制文本时的颜色。

5.使用 GDI 对象的函数。GDI 在这有点混乱。举一个例子,缺省使用 GDI 函数绘制的直线都是实线并具有一个标准宽度。您可能需要更粗的直线或点划线。这种线宽和线风格不是设备上下文的属性,而是一个"逻辑画笔"的属性。可以通过在 CreatPen()中指定这些属性来创建一个逻辑画笔,然后将其选入设备上下文(用 SelectObject()函数)。在此期间所画的线条都使用该画笔,用完后再取消设备上下文中的画笔选择,并清除画笔对象。除画笔对象外,还有刷子、字体、位图等对象。

1.1.3 基于资源的编程

在 MS - DOS 下进行数据驱动的编程时,必须用常量初始化数据或提供独立的数据文件供程序读取。但进行 Windows 编程时,数据保存在一定格式的资源文件中。链接器用编译器输出的目标文件与此二进制的资源文件生成可执行程序。资源文件可包括位图、图标、菜单定义、对话框布局以及字符串,甚至可以包括自定义格式的资源。

例如,图 1 - 1 所示的是对话框资源。

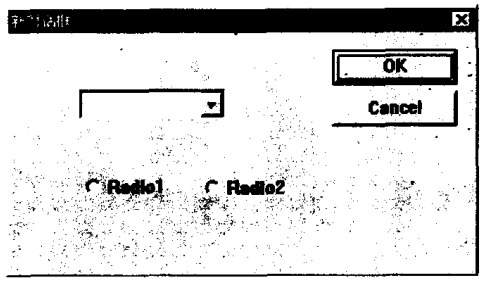

图 1-1 对话框资源

它在资源文件中的定义如下：

```
//////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_DIALOG1 DIALOG DISCARDABLE  0, 0, 187, 98
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "新对话框"
FONT 10, "System"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,130,7,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,130,24,50,14
    COMBOBOX        IDC_COMBO1,28,23,59,23,CBS_DROPDOWN | CBS_SORT |
                    WS_VSCROLL | WS_TABSTOP
    CONTROL         "Radio1",IDC_RADIO1,"Button",BS_AUTORADIOBUTTON,30,56,41,14
    CONTROL         "Radio2",IDC_RADIO2,"Button",BS_AUTORADIOBUTTON,78,58,46,11
END
```

这种方式将程序逻辑和它处理的数据分离，大大降低了二者的耦合。对于软件国际化，也是十分有用的，可以将程序要用到的文本字符串放在资源文件中，不同语言的版本只需用不同语言的资源文件进行链接即可。

1.1.4 动态链接库 DLL

在 MS-DOS 环境下，所有模块都是在程序建立期间静态链接的。Windows 允许动态链接，也就是说特制的库文件可以在运行期间装载连接。所谓动态链接库(DLL)是由函数、对象及资源组成的二进制库文件。多个应用程序可以共享同一个动态链接库，从而节省内存和磁盘空间。动态链接库不能直接执行，也不接收消息。它是独立的文件，其中包含能被其他程序或 DLL 调用来完成某项工作的函数，以及能被利用的资源。只有在其他模块调用动态链接库中的函数或使用其中的资源时，它才发挥作用。

所谓“动态链接”，是指 Windows 把一个模块中的函数调用链接到库模块中的实际函数的过程。在程序开发过程中，通常要将各种目标模块(.obj)、运行库(.lib)文件链接到一起，以及经常还有可能包括一个编译好的资源文件，以便创建 Windows 的.exe 文件，这种链接是

“静态链接”。动态链接却发生在运行时。Windows 本身的很多模块就是用动态链接库实现的，例如三个基本模块 Kernel32.dll、User32.dll、GDI32.dll，以及各种驱动程序如 Mouse.drv、System.drv，这些库都能被所有 Windows 应用程序在运行时使用。

尽管一个动态链接库模块可能有其他扩展名（如.exe 或.fon），但标准的扩展名是 dll。只有带.dll 扩展名的动态链接库能被 Windows 自动加载。如果文件为其他扩展名，则程序必须显式地使用 LoadLibrary 函数加载该模块。

1.1.5 Windows NT 的特点

Windows NT 是一个新的 32 位操作系统，包含了一个更高级的文件系统。它具有保密性，多线性，真正的先入为主的多任务性，更高级的网络性能以及可以在 RISC 体系结构的机器上运行等特点。在 Windows NT 环境下，可以运行任何基于 Windows 的 32 位应用程序。

1.2 Windows 的窗口类

Windows 的窗口类并不是面向对象意义上的类，它确实对一些东西进行了封装，但并不能继承，也没有私有与公有的概念。一个窗口类是一套属性集，系统用它作为模板创建窗口对象。每个窗口都是窗口类的一个成员。所有窗口类都与特定的进程相关。

一个进程必须在创建窗口之前注册一个窗口类，从而将相关的窗口过程、类样式和其他类属性与一个类名关联起来。当使用 CreateWindow()或 CreateWindowEx()创建窗口时，指定一个类名，系统就会用与该类名相关的窗口过程、类样式和其他类属性创建一个该类的窗口对象。

窗口类所包含的元素及用途如表 1-1 所示。

表 1-1 窗口类元素及其用途

元 素	用 途
类名	标识该类，与其他已注册的类相区别
窗口过程地址	处理送入窗口的所有消息的函数的指针
实例句柄	标识注册该类的应用程序或 DLL
类的光标	定义系统为该类的窗口显示的鼠标光标
类的图标	定义该类大图标和小图标
类的背景刷子	定义填充客户区的颜色和模式
类的菜单	为没有明确定义菜单的窗口指定缺省菜单
类的额外字节数	指定系统为该类保留的额外字节数，该类的所有窗口均可将它用于应用程序定义的用途，系统将其初始化为零
窗口的额外字节数	指定系统为该类的每个窗口保留的字节数，窗口可将它用于应用程序定义的任何用途，系统将其初始化为零

1.2.1 窗口类的结构

Windows.h 中定义了窗口类的结构(WNDCLASS 及 WNDCLASSEX)如下:

```
typedef struct _WNDCLASS {
        UINT      style;
        WNDPROC lpfnWndProc;
        int       cbClsExtra;
        int       cbWndExtra;
        HANDLE    hInstance;
        HICON     hIcon;
        HCURSOR hCursor;
        HBRUSH    hbrBackground;
        LPCTSTR lpszMenuName;
        LPCTSTR lpszClassName;
} WNDCLASS;
typedef struct _WNDCLASSEX {
        UINT      cbSize;
        UINT      style;
        WNDPROC lpfnWndProc;
        int       cbClsExtra;
        int       cbWndExtra;
        HANDLE    hInstance;
        HICON     hIcon;
        HCURSOR hCursor;
        HBRUSH    hbrBackground;
        LPCTSTR lpszMenuName;
        LPCTSTR lpszClassName;
        HICON     hIconSm;
} WNDCLASSEX;
```

其中:

style——指定窗口样式。可以用位操作符“|”结合各种样式。例如 CS_HREDRAW|CS_VREDRAW 指定窗口在客户区的宽度以及高度变化时均重画整个窗口。

lpfnWndProc——是指向窗口过程的指针。必须用 CallWindowProc()函数调用窗口过程。

cbClsExtra——指定了分配给窗口类的额外字节数。

cbWndExtra——指定了分配给每个窗口实例的字节数。

hInstance——包含类的窗口过程的实例句柄。

hIcon——类图标的句柄。它必须是一个图标资源的句柄。

hCursor——类光标的句柄。它必须是一个光标资源的句柄。

hbrBackground——背景刷子的句柄。

lpszMenuName——指向类菜单的一个连续字符串的指针。如果用整型值标识菜单,就要用 MAKEINTRESOURCE 宏将其转化为与 Win32 资源管理函数兼容的资源类型。

lpszClassName——指向类名第一个字符串的指针或原子值。

cbSize —— WNDCLASSEX 多于 WNDCLASS 的成员，它指定该结构的字节数。将它设置为 sizeof(WNDCLASSEX)。

hIconSm —— WNDCLASSEX 多于 WNDCLASS 的成员，它是与该窗口类关联的小图标的句柄。

1.2.2 窗口类的类型

窗口类有三种类型：

1.系统类。

2.应用程序全局类。

3.应用程序局部类。

它们的区别在于作用范围、注册和注销的时机和方式。

• 系统类

系统注册的系统类供所有进程使用。任何进程可以在任何时候使用一个系统类，因为它由系统注册，进程无法注销它。

每个 Win32 应用程序都可获得自己的一份系统类的拷贝。表 1－2 描述了各系统类。

表 1－2 各种系统类及其描述

类	描述
Button	按钮类
ComboBox	组合框类
ComboLBox	包含一个组合框的列表框
Edit	编辑框控件类
ListBox	列表框类
MDIClient	MDI 客户区窗口类
ScrollBox	滚动框类
Static	静态控件类
# 32768	菜单类
# 32769	桌面窗口类
# 32770	对话框窗口类
# 32771	任务切换窗口类

• 应用程序全局类

应用程序全局类由可执行程序或动态链接库注册，并能被进程中其他模块访问。例如，如果一个 DLL 调用 RegisterClassEx()函数注册一个用于定义某自定义控件的应用程序全局类，则装载该 DLL 的进程就可以创建该自定义控件的实例。

要注销一个应用程序全局类并释放其存储空间，请使用 UnregisterClass()函数。

• 应用程序局部类

应用程序局部类是由可执行程序或 DLL 注册以便于自己使用的。虽然可以注册任意

数量的局部类,但通常只注册一个。该窗口类支持应用程序主窗口的窗口过程。

系统在注册该类的应用程序关闭时注销局部类,应用程序也可以调用 UnregisterClass()函数去除一个局部类并释放其存储空间。

1.2.3 注册窗口类

一个窗口类定义了窗口的属性,如样式、图标、光标、菜单及窗口过程。注册窗口类的第一步是填充一个WNDCLASS或WNDCLASSEX结构,然后将此结构传给RegisterClassEx()函数。

要注册一个应用程序全局类,需在 style 成员中指定 CS_GLOBALCLASS 风格,而注册一个应用程序局部类则不要指定该风格。

如果用 ANSI 版本的 RegisterClassEx()(即 RegisterClassExA())进行注册,则应用程序要求系统使用 ANSI 字符集向该类的窗口传递消息的文本参数;如果用的是 Unicode 版本的(即 RegisterClassExW()),则要求系统使用 Unicode 字符集传递文本参数。

注册窗口类的可执行程序或 DLL 就是该类的拥有者。系统通过 WNDCLASSEX 结构的 hInstance 成员判断拥有关系。对于 DLL 来说,hInstance 成员必须是 DLL 的实例句柄。

1.3 窗口过程

每个窗口类都有一个与之相关联的窗口过程,它是一个负责处理传送到该类的所有窗口的消息的函数,窗口的外观和行为都取决于响应这些消息的窗口过程。每个窗口都是特定窗口类的一个实例,窗口类决定了窗口用于处理消息的缺省窗口过程,所有属于相同窗口类的窗口共享同一个窗口过程。

应用程序通常至少注册一个新的窗口类和相关的窗口过程,然后就可以创建该类的窗口实例了。因为这些窗口共享同一窗口过程,也就意味着相同的代码段可以同时被多个不同的调用者调用,因此从窗口过程修改共享的资源时必须小心。

1.3.1 窗口过程的结构

一个窗口过程实际上是一个函数,它带四个参数,返回一个 32 位值。四个参数分别为窗口句柄,消息标识符,以及两个数据类型分别为 WPARAM、LPARAM 的消息参数。一个典型的窗口过程的声明如下:

```
LRESULT CALLBACK WindowProc(    HWND hwnd,          // handle to window
  UINT uMsg,          // message identifier
  WPARAM wParam,    // first message parameter
  LPARAM lParam     // second message parameter);
```

1.3.2 缺省窗口过程

缺省的窗口过程函数 DefWindowProc 定义了所有窗口共享的基本操作,它给窗口提供了

最少的基本功能。应用程序定义的窗口过程应将所有它不处理的消息传给 DefWindowProc 函数,以获得缺省的处理。

1.3.3 窗口过程子类化

当应用程序创建一个窗口时,系统将为它分配一块内存以存储该窗口的特定信息,包括为该窗口处理消息的窗口过程的地址。当系统将一个消息传递给窗口时,它将搜索其窗口过程的地址然后将消息传给它。

子类化是这样一种技术,它允许应用程序在消息被发送到窗口之前截取并处理它。通过子类化一个窗口过程,应用程序可以增强、修改或监视窗口的行为。应用程序可以子类化属于系统全局类的窗口,例如子类化编辑控件,使其只能接收数字,不接收其他字符。但不能子类化其他应用程序的窗口。

应用程序子类化窗口方法是用一个新的窗口过程地址代替原始的窗口过程地址,这样子类过程将接收所有发送给该窗口的消息。对于接收的消息,它可作出三种处理:将其传给原始窗口过程;修改该消息然后再传给原始窗口过程;自己处理该消息而不传给原始窗口过程。如果子类过程处理消息,它的处理可以在将其传给原始窗口过程之前、之后,或之前之后都处理。

系统提供了两种类型的子类化:实例类型和全局类型。对于实例类型的子类化,应用程序只替换一个窗口实例的窗口过程地址,子类化一个已经存在的窗口必须使用此类型。而对于全局子类化,应用程序将替换窗口类的 WNDCLASS 结构中的窗口过程地址。随后创建的该类的所有窗口都将使用替换后的窗口过程地址,但此前已有的窗口不受影响。

要子类化一个窗口,应调用 SetWindowLong 函数,并指定窗口的句柄、GWL_WNDPROC 标志位以及指向子类过程的指针。该函数返回原始窗口过程的指针,并使用此指针将消息传给原始窗口过程。子类过程必须使用 CallWindowProc 函数来调用原始窗口过程。下面的例子显示了如何子类化一个对话框中的编辑控件,子类化后的编辑控件可接收所有键盘输入,包括回车和制表键。

```
WNDPROC wpOrigEditProc;

LRESULT APIENTRY EditBoxProc(
        HWND hwndDlg,
        UINT uMsg,
        WPARAM wParam,
        LPARAM lParam)
{
        HWND hwndEdit;
        switch(uMsg)
        {
        case WM_INITDIALOG:
                // Retrieve the handle to the edit control.
                hwndEdit = GetDlgItem(hwndDlg, ID_EDIT);
                // Subclass the edit control.
```

```
                wpOrigEditProc = (WNDPROC) SetWindowLong(hwndEdit,
                GWL_WNDPROC, (LONG) EditSubclassProc);
                //
                // Continue the initialization procedure.
                //
                return TRUE;
        case WM_DESTROY:
                // Remove the subclass from the edit control.
                SetWindowLong(hwndEdit, GWL_WNDPROC,
                (LONG) wpOrigEditProc);
                //
                // Continue the cleanup procedure.
                //
                break;
        }
    return FALSE;
    UNREFERENCED_PARAMETER(lParam);
}
// Subclass procedure
LRESULT APIENTRY EditSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
    if (uMsg == WM_GETDLGCODE)
        return DLGC_WANTALLKEYS;
    return CallWindowProc(wpOrigEditProc, hwnd, uMsg,
        wParam, lParam);
}
```

1.4 应用程序框架

MFC 提供了一种应用程序框架,其核心是 Document/View 结构。绝大多数程序都要处理一种或多种文挡,Document 类用于创建、打开和保存文档,View 类用于文档的显示和打印等,支持用户与文档的交互。应用程序中的一个或多个视窗是主框架窗口 CFrameWnd 类的子窗口,框架窗口中通常还包含工具栏、状态栏等子窗口。

1.4.1 MFC 应用程序框架的功能

MFC 应用程序框架提供了以下的通用功能:

1.全面支持 File、Open、Save、Save As 功能,且采用了文件列表形式。
2.可支持滚动窗口和切分窗口形式。
3.可支持状态条和工具条形式。
4.采用上下文相关帮助形式。

5.自动处理进入对话框的数据。

6.支持动态链接(DLL)。

7.支持 Internet 编程。

1.4.2 应用程序框架的结构

MFC 应用程序的框架结构主要由应用类、视图、框架类和文档类四个基本部分组成。MFC 编程实际上就是对这四个类进行定制,以满足实际应用程序的需要。为了能对这些类进行有目的,符合要求的修改,就要求程序员对这些类的结构、用途以及它们之间的联系有一个整体的了解。下面就分别来介绍它们。

• CwinApp 应用类

MFC 提供了应用级的类 CwinApp ,它用来控制整个应用程序的运行过程,它的几个虚拟函数提供了应用程序的接口。可以通过重载 CwinApp::InitApplication 和 InitInstance 来初始化程序,函数 Run 用于启动应用程序的主消息循环, 重载函数 ExitInstance 可以完成应用程序退出前的清理工作。CwinApp 类是 MFC 应用框架的重要一环,负责文档模板的管理和部分菜单命令的处理等。CwinApp 类还有很多附加的特性,如分析命令行参数,处理初始化文件,维护最近打开的文件列表,提供帮助,系统支持等。CwinApp 类封装了 Windows 应用程序的事例句柄 m_hInstance,大多数使用该句柄的 Win32 函数被定义为其成员函数。

现在我们以 firex01 应用程序为例来说明 CwinApp 类结构。该实例的源程序可在本书配套光盘的 Chapter1 \ firex01 \ 目录下找到。

CwinApp 头文件结构如下:

```
// firex01.h: main header file for the FIREX01 application
#if !defined(AFX_FIREX01_H__E0384409_0E51_11D3_8046_D657E035461D__INCLUDED_)
#define AFX_FIREX01_H__E0384409_0E51_11D3_8046_D657E035461D__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"       // main symbols
/////////////////////////////////////////////////////////////////////////////
// CFirex01App:
// See firex01.cpp for the implementation of this class
//

class CFirex01App : public CWinApp
{
public:
    CFirex01App();

// Overrides        // 构造函数
```

```
        重载初始化虚函数
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFirex01App)
        public:
        virtual BOOL InitInstance();  // 初始化虚函数
        //}}AFX_VIRTUAL

// Implementation
        // 实现消息映射
        //{{AFX_MSG(CFirex01App)
        afx_msg void OnAppAbout();  // 定义 OnAppAbout()消息函数
                // 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()
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}  // 插入消息声明
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FIREX01_H__E0384409_0E51_11D3_8046_D657E035461D__INCLUDED_)
```

这段代码在开头令人困惑，#if!defined 后面跟着一个很长的字符串，这是包含保护(incluing guarding)的明智形式，文件的实际内容是类 CFire01App 的定义，该类继承了 CWinApp，它提供读者所需的大多数 MFC 类。

CWinApp 应用文件结构如下：

```
// firex01.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "firex01.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "firex01Doc.h"
#include "firex01View.h"

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

/////////////////////////////////////////////////////////////////////////////
// CFirex01App
// 声明消息映射
```