程序员加油站系列图书

Visual C++

高级编程

Effective programming

张力 编著

程序员加油站系列图书

Visual C++ 高级编程

张力 编著

人民邮电出版社

图书在版编目（CIP）数据

Visual C++高级编程 / 张力编著. —北京：人民邮电出版社，2002.3
ISBN 7-115-10188-4

Ⅰ. V… Ⅱ. 张… Ⅲ. C语言—程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2002）第013085号

内容提要

本书深入讲解了Visual C++6.0的高级编程方法，用大量的实例说明了MFC类和API函数的使用方法和技巧。全书共分6章，分别介绍了界面编程、图像和多媒体编程、系统编程、网络编程和串口编程等多方面的内容。

本书适用于具有Visual C++中高级编程水平的读者阅读。

程序员加油站系列图书

Visual C++高级编程

◆ 编　　著　张　力
责任编辑　张立科

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
读者热线　010-67180876
北京汉魂图文设计有限公司制作
北京密云春雷印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：28
字数：683千字　　2002年3月第1版
印数：1-5 000册　　2002年3月北京第1次印刷

ISBN 7-115-10188-4/TP・2820

定价：44.00元（附光盘）

前　言

微软推出 Visual C++已经有若干个年头了，作为程序开发工具，从最初的 1.0、1.5、2.0 发展到今天的 6.0 版本，每次新版本的推出，都会在原有基础上增色不少。

因为有越来越多的人加入到 VC 程序员的行列中，很多读者对 Visual C++开发环境和基本应用已经非常熟悉。现在他们需要一本无论在广度和深度上都有所突破的书籍来帮助他们百尺竿头，更进一步。

本书正是在这样的背景和目的下问世的。虽然它也是通过一个个的实例讲解 VC 编程的各种技巧和方法，但是在选用实例的过程中，笔者避开那些读者已经耳熟能详的例子，并结合高校 BBS 中一些热门的话题，如远程控制、屏幕取词、USB 端口通信等，给出笔者精心制作却又短小精悍的实例，以便读者能尽快掌握这些方法。这些实例都在 Windows 2000 下调试通过，读者可以直接运行光盘中的实例。

全书共分 6 章，分别介绍界面、多媒体及图像、系统及 Shell、网络和串口等方面的内容，试图全面而深入地讲解 Visual C++的高级特性，许多细节都是其他书籍中不曾有过的内容。

第 1 章介绍一些高级界面编程的技巧。对于使用软件的用户，界面常常比性能更重要。

第 2 章介绍常用的图像和多媒体编程技巧。因为图像和多媒体编程的重点在于数据格式，所以这一章挑选了有代表性的实例进行讲解。

第 3 章介绍有关操作系统和 API 编程的方法，同时也解释了如回调函数、钩子函数等重要概念，另外也给出了查看星号密码等热门实例。

第 4 章介绍了网络的基本应用编程。由于受制于网络特性，这方面的编程较为复杂，因而分为两章分别介绍，这一章给出了基本的套接字函数。

第 5 章在前一章的基础上，介绍了几个网络的实用方面的例子，让读者学以致用。

第 6 章给出了串口编程的两个实例。虽然 USB 接口并不是通常所称的“串口”，但由于它也采用串行通信的方式，因此也归入本章。

虽然笔者在成书时谨慎仔细，但囿于学识经验，难免有疏忽甚至错误之处。在此恳请读者不吝赐教和指正，以共同提高。

编　者

目 录

第 1 章　高级界面设计

一个好的程序往往有着不错的界面。作为程序员，也非常在意自己的成果是否精美。本章就通过若干个实例，来启发读者如何设计出漂亮的界面。

1.1　实现变形窗体

在图形操作系统中，最重要的界面元素就是窗体。MFC 工程有三类，分别基于单文档界面、多文档界面和对话框，这三类都是以窗体为载体的，基于对话框的工程是其中最简单的一类，一般来说，它不与文档发生联系，通过 DDX（对话框数据交换）机制完成与用户的交互。因此，它也没有与文档的一切操作，如保存、打开等。但是，麻雀虽小，五脏俱全，对话框类（CDialog）是窗体类（CWnd）的派生类，它也同样能实现窗体的一切特殊效果。窗体是由名为 CREATESTRUCT 的结构描述的，这个结构定义为

```
typedef struct tagCREATESTRUCT {
    LPVOID      lpCreateParams;
    HINSTANCE   hInstance;
    HMENU       hMenu;
    HWND        hwndParent;
    int         cy;
    int         cx;
    int         y;
    int         x;
    LONG        style;
    LPCSTR      lpszName;
    LPCSTR      lpszClass;
    DWORD       dwExStyle;
} CREATESTRUCT;
```

lpCreateParams 是指向未定数据类型的指针，在应用时需要进行强制类型转换，变为用来创建窗体的数据类型指针，Windows 根据它确定的地址访问数据，从而创建窗体；

hInstance 是拥有该窗体的程序实例句柄。Windows 根据这个句柄将窗体赋给对应的程序实例，因此，有关这个窗体的消息都由它处理；

hMenu 是新窗体将使用的菜单句柄。但如果这个窗体是子窗体（Child Window）时，这个参数只记录子窗体的 ID，因为子窗体是没有菜单项的；

hwndParent 是该窗体的父窗体句柄。当这个窗体是最高层时，该参数为 NULL；

cy 是窗体的高度；

cx 是窗体的宽度；

X 是窗体左上角的 *X* 轴坐标，当窗体是最高层时，坐标是相对于屏幕的；当它是子窗体时，坐标则是相对于父窗体的；

Y 是窗体左上角的 *Y* 轴坐标，坐标的概念与 *X* 相同；

style 则定义了窗体的基本属性；

lpszName 是字符型指针，指向窗体的名字，也就是窗体的标题；

lpszClass 也是字符型指针，指向窗体类型的名字，如“Static”、“Edit”等；

dwExStyle 定义了窗体的扩展属性。

Windows 就是根据这些参数来管理并显示窗体的。读者可能对有些参数的含义还不甚了解，比如坐标是怎样定义的，什么是窗体类型，窗体为什么要分为基本属性和扩展属性，具体都指些什么。在本章的后面几节中，将结合实例介绍，这里只要有个总体概念就可以了。下面就分析第一个实例：变形的不规则窗体。

1.1.1 建立工程

生成基于对话框的 MFC 程序，将其命名为 distort，整个工程的设置结果如图 1-1 所示。

最后生成的类有三个，CAboutDlg、CDistortApp、CDistortDlg，其中，我们要作大变动的是 CDistortDlg，因此，在后面各节中就给出该类的头文件和具体的程序，其他类则不作变动，此处就不列出了。在给出的源代码中，为方便读者阅读，凡是改动或添加的部分都用黑体字给出，并且也有相应的中文注释。

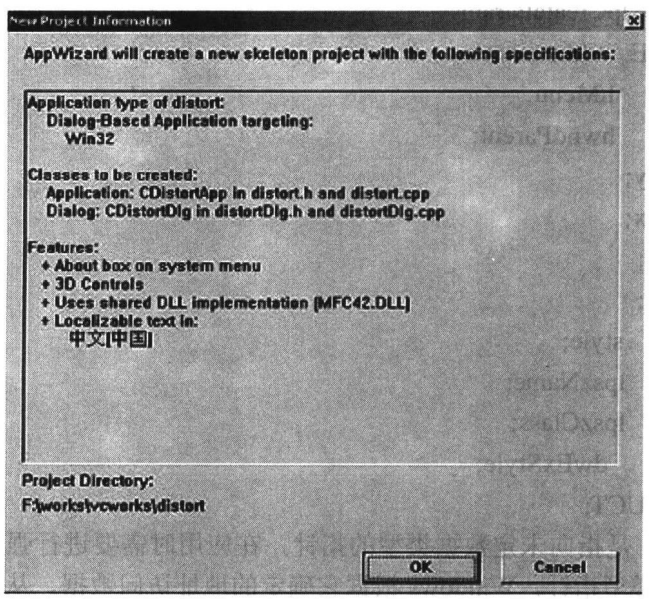

图 1-1 distort 工程总览

1.1.2 CDistortDlg 头文件

```
// distortDlg.h : header file
```

```
#if !defined(AFX_DISTORTDLG_H__C239447A_1C20_4113_B682_8AB1632D4DD7__IN
CLUDED_)
#define
AFX_DISTORTDLG_H__C239447A_1C20_4113_B682_8AB1632D4DD7__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////
// CDistortDlg dialog

class CDistortDlg : public CDialog
{
// Construction
public:
    CDistortDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CDistortDlg)
    enum { IDD = IDD_DISTORT_DIALOG };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDistortDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CDistortDlg)

    // Visual C++开发环境已经重载了虚函数 OnIniDialog 和 OnPaint
    // 因此只需更改函数体就行了，而不用重新声明
```

```
        virtual BOOL OnInitDialog();            // 初始化对话框
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();                 // 绘制窗体
        afx_msg HCURSOR OnQueryDragIcon();

        // 重载 WM_NCHITTEST 消息，使得单击窗体任何区域也能拖动窗体
        afx_msg UINT OnNcHitTest(CPoint point);
        // 增加定时器，为窗体增加变形效果
        afx_msg void OnTimer(UINT nIDEvent);
        // 在撤销对话框对象时，关闭定时器
        afx_msg void OnDestroy();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
    private:
        // 记录窗体初始的矩形
        CRect m_rectWnd;
        // 用来设置窗体所在的区域
        CRgn m_rgn;
    };

    //{{AFX_INSERT_LOCATION}}
    // Microsoft Visual C++ will insert additional declarations immediately before the previous line.

    #endif
// !defined(AFX_DISTORTDLG_H__C239447A_1C20_4113_B682_8AB1632D4DD7__INCLUDED_)
```

在 CDistortDlg 类中，添加了三个消息处理函数 OnNcHitTest、OnTimer 和 OnDestroy，分别处理 WM_NCHITTEST、WM_TIMER 和 WM_DESTROY 消息。

消息是 Windows 编程中非常重要的一个概念，可以将它理解为“中断”。也许有的读者曾用 TC 编写过 DOS 环境下的“俄罗斯方块”游戏，Windows 中的事件驱动编程原理就与此类似。在俄罗斯方块游戏中，如果没有键盘输入时，即没有事件发生，方块就慢慢往下掉；当用户按下功能键，如“变形键”时，程序通过截获键盘产生的中断，响应这一消息，并在屏幕上显示出方块变形后的形状。Windows 也是如此，只是它封装了中断，然后转换成消息，送到消息对应的目的地。接收这一消息的对象一般是窗体及其派生对象，如按钮、文本框等。同样的事件，可能产生不同类型、不同数目的消息和不同的接收对象。如同样一个鼠标左键单击的事件，送到按钮时，Windows 会产生 BN_CLICKED 消息，而送到列表框的下拉箭头时，Windows 就会产生 CBN_DROPDOWN 消息了。没有消息产生时，Windows 就调用 OnIdle 函数来完成一些后台工作。

下面的代码解释了 Windows 工作的主要原理：

```
for (;;)
{
    // 第一部分：检查是否可以开始空闲时的工作，即检查有无消息需要处理
    while (bIdle &&
        !::PeekMessage(&m_msgCur, NULL, NULL, NULL, PM_NOREMOVE))
    {
        // call OnIdle while in bIdle state
        if (!OnIdle(lIdleCount++))
            bIdle = FALSE; // assume "no idle" state
    }

    // 第二部分：处理消息
    do
    {
        // pump message, but quit on WM_QUIT
        if (!PumpMessage())
            return ExitInstance();

        // reset "no idle" state after pumping "normal" message
        if (IsIdleMessage(&m_msgCur))
        {
            bIdle = TRUE;
            lIdleCount = 0;
        }

    } while (::PeekMessage(&m_msgCur, NULL, NULL, NULL, PM_NOREMOVE));
}
```

根据上面的代码，可以认为，Windows 程序就是没有固定结束时间的循环体。先看看循环体的第二部分，当有消息产生时，Windows 将消息放入消息队列，清除空闲状态，并从消息队列中取出消息进行处理。当处理的消息是 WM_QUIT，即关闭程序消息时，退出循环体，完成退出的一系列工作后关闭程序，结束进程，否则，继续看消息队列中是否有消息，没有了，就转入第一部分。循环体的第一部分则引导程序开始空闲时的工作。

当从队列中取出消息后，Windows 下一步要做的就是确定处理该消息的函数并调用之。Windows 管理着每个窗体的属性，包括窗体的大小、位置、状态（最大、最小化、层叠）和属性（按钮、文本框、下拉框等等），当鼠标事件（移动、左键单击、双击等等）发生时，Windows 就根据鼠标的位置、窗体的位置以及属性，将事件翻译成相应的消息发给鼠标当前所属的窗体。那么，窗体又如何知道该由哪个成员函数来处理这个消息呢，奥妙就在实现文件（*.CPP）的消息映射宏中。消息映射宏将在下面介绍 CDistortDlg 类的实现时讲解。

回到本节的实例中，此处重载的三个消息，WM_NCHITTEST 是当鼠标单击窗体非客户区时产生的，即鼠标单击窗体的标题栏、边框时，Windows 会向窗体发送该消息，根据消息映射宏的定义，Windows 调用 OnNcHitTest 函数。同样，处理另两个消息时，Windows 分别调用 OnTimer 和 OnDestroy 函数。WM_DESTROY 是窗体将被撤销时发送的消息，通俗的说，当用户按下窗体右上方的“关闭”按钮时，就向窗体发送了这一个消息。WM_TIMER 是定时器消息，产生这一消息的方法将在下面介绍。消息响应函数的参数 nIDEvent 是定时器事件号。窗体可能设置了多个定时器，定时器到时调用 OnTimer 函数时，为了确定是哪个定时器到时，就需要用无符号整型参数 nIDEvent 来标识。

1.1.3 实现变形窗体

在 Visual C++的资源管理器中，将对话框模板的属性（Styles）设为“popup”、无边界（Border）为 none。在所给出 CDistortDlg 的成员函数中，略去了未作改动的函数，如 DoDataExchange、OnSysCommand 等。

```
// distortDlg.cpp : implementation file
//

#include "stdafx.h"
#include "distort.h"
#include "distortDlg.h"

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

/////////////////////////////////////////////////////////////////////////////
// CDistortDlg dialog

    // ... 略去开发环境自动生成的函数体

BEGIN_MESSAGE_MAP(CDistortDlg, CDialog)
    //{{AFX_MSG_MAP(CDistortDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_NCHITTEST()
    ON_WM_TIMER()
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

此处的消息映射宏就是Windows确定消息响应函数的奥妙所在。Visual C++中多次通过巧妙的宏定义，使得代码简洁而又高效，同时也提高了可读性。以消息映射宏为例，Visual C++中的定义是：

```
#define BEGIN_MESSAGE_MAP(theClass, baseClass) \
AFX_MSGMAP* theClass::GetMessageMap() const \
{ return &theClass::messageMap; } \
AFX_MSGMAP theClass::messageMap = \
{ &(baseClass::messageMap), \
(AFX_MSGMAP_ENTRY*) &(theClass::_messageEntries) }; \
AFX_MSGMAP_ENTRY theClass::_messageEntries[] = \
{
#define ON_COMMAND(id, memberFxn) \
#define END_MESSAGE_MAP() \
{ 0, 0, 0, 0, AfxSig_end, (AFX_PMSG)0 } \
};
```

其中的AfxSig_end 是消息项(MESSAGE ENTRY)的结束标志。

上面的宏定义是不是让你赞叹不已却又眼花缭乱？一般说来，程序员并不需要会设计这样堪称艺术品的宏定义，只要能看懂其机理，会使用就可以了。因此这里笔者就简单介绍这个宏，而不具体展开了。AFX_MSGMAP_ENTRY是一个特殊的结构，定义如下：

```
struct AFX_MSGMAP_ENTRY
{
UINT nMessage;   // Windows消息
UINT nCode; // 控件号或WM_NOTIFY 号
UINT nID; // 控件ID (或 0 ，对于Windows消息)
UINT nLastID; // 用来确定控件ID的范围
UINT nSig; // 签名类型或消息的地址
AFX_PMSG pfn; // 要调用的程序 (或特殊值)
}
```

由这个结构的定义可以看出，这个结构详细描述了消息，所谓的传递消息，就是对这样的结构进行操作。BEGIN_MESSAGE_MAP()宏，就是定义了一个AFX_MESGMAP_ENTRY类型的外部数组，并以END_MESSAGE_MAP()宏中定义的结束标志结尾。这个数组的成员就是窗体要处理的消息，也是由宏定义的，对于特殊的消息，也有专门的宏给出定义，如本例中的消息就都是用这些专门的宏来定义的。对于一般的消息，用ON_COMMAND()宏定义，在该宏中，第一个参数id是消息号，第二个参数memberFxn就是消息响应函数的地址，也就是函数名。这样建立了一张消息映射表，Windows就能根据id调用相应的memberFxn了。

讲到这里，读者应该能给出在头文件中宏DECLARE_MESSAGE_MAP()的定义了吧，这个宏声明了AFX_MSGMAP_ENTRY类型的静态变量messageMap、静态数组_messageEntries[]和AFX_MSGMAP_ENTRY*类型的常值虚函数GetMessageMap()。如下所示：

```
#define DECLARE_MESSAGE_MAP() \
static AFX_MSGMAP_ENTRY _messageEntries[]; \
static AFX_MSGMAP messageMap; \
virtual AFX_MSGMAP* GetMessageMap() const;
```

下面介绍对话框程序一般都要重载的虚函数OnInitDialog，它也是消息响应函数，处理WM_INITDIALOG消息。

```
//////////////////////////////////////////////////////////////////////////
// CDistortDlg message handlers

    修改OnInitDialog函数建立一个椭圆区域
    并调用SetWindowRgn将该区域分配给窗口

// 定义定时器事件号
#define EVENT_REDRAW 0          // 定义重绘事件号为0

//   初始化对话框窗体
BOOL CDistortDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING,          IDM_ABOUTBOX,
strAboutMenu);
        }
    }
```

```
    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon

    // TODO: Add extra initialization here

    // 获得对话框客户区的尺寸
    GetClientRect(m_rectWnd);

    // 创建窗体区域并与窗体建立联系
    m_rgn.CreateEllipticRgn(0,0,m_rectWnd.right,m_rectWnd.bottom);
    SetWindowRgn((HRGN)m_rgn,TRUE);

    // 设置重画窗口的定时器
    // 每2秒重绘一次窗体
    this->SetTimer(EVENT_REDRAW,2000,NULL);

    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

WM_INITDIALOG消息是当对话框已经建立将要在屏幕上显示出来时发送的。对于模态对话框，创建并显示都是由DoModal函数完成的，函数返回后，对话框也被撤销，并消失，细节都封装在Visual C++中，无法定制。如果在调用DoModal前更改对话框的属性，将出现断言(ASSERT)错，要想定制对话框，只有重载OnInitDialog函数，在对话框创建之后、显示之前来改变其属性。本例也是一个模态对话框，DoModal函数在CDistortApp的InitInstance中被调用。OninitDialog函数的前半部分是创建系统菜单和缺省图标，如何改变系统菜单和缺省图标，只需简单地调用几个函数就可以了，这里略去不提。在“// TODO: Add extra initialization here”后面就是定制对话框的程序。窗体重绘时是刷新客户区的内容，本例中，窗体也是按照客户区的尺寸进行设计。

椭圆窗体效果是通过SetWindowRgn函数实现的，它的定义为：

int SetWindowRgn(HRGN hRgn, BOOL bRedraw);

其中，参数 hRgn 是要设置的窗体区域句柄；

参数 bRedraw 则决定是否要重绘窗体的客户区。

当设置了窗体区域后，Windows 就不在区域外作图，就好像有一把无形的“剪刀”把区域外的部分给剪掉了，绘出的窗体的形状和区域所确定的一样，而不论这个区域是怎样的稀奇古怪。

在本实例中，bRedraw 取值 TRUE，表示要重新绘制客户区，Windows 向窗体发送 WM_PAINT 消息，从而调用消息响应函数 OnPaint，另一方面，本例中的窗体和客户区同样

大小，因而整个窗体都得到了重绘。

```
// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

//  通过建立区域和调用SetWindowRgn，
//  已经建立一个不规则形状的窗口，
//  下面的例子程序是修改OnPaint函数使窗口形状看起来像一个球形体
void CDistortDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    if (IsIconic())
    {
        SendMessage(WM_ICONERASEBKGND,(WPARAM)dc.GetSafeHdc(),0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        // 重新编辑else块里的代码
        // 绘制一个不带边界的椭圆
        dc.SelectStockObject(NULL_PEN);

        // 获得窗口客户区的尺寸
        CRect rect;
        GetClientRect(rect);

        CBrush *pBrushOld;
        CBrush brushNew;
```