高级Windows编程技术

和克智
王　平　著
白天琪

西安交通大学出版社

前　言

32 位 Windows（Windows NT，95，98 等）与 16 位 Windows（Windows 3.x）相比，不仅仅是数据的宽度提高了一倍，而且在功能上有了较大的改进。32 位 Windows 的抢先式多任务比起 16 位 Windows 的协作式多任务而言，使得多个应用程序可以更加协调地工作，更有利于实时任务的执行；32 位 Windows 对多媒体的支持更为广泛，编程更加简便；32 位 Windows 提供了更多的控件，使得应用程序更美观，操作更方便，用户界面更友好；32 位 Windows 的内存管理为应用程序提供了更大的存储空间，一个应用程序可使用的内存多达 4GB；32 位 Windows 的扩展文件管理允许使用长文件名，内存映像文件为开发数据库应用程序、文字处理程序和图像处理程序提供了更多的方便；……，等等。32 位 Windows 优越的性能吸引了越来越多的用户。由于 Windows 95 OSR2 版（俗称 Windows 97）修正了原 Windows 95 中的许多隐含错误，系统的稳定性得到了很大的提高，现在大多数个人用户都在自己的机器上安装了该操作系统，也有许多用户安装了 Windows 98（该系统的稳定性尚不能令人满意），甚至有的个人用户安装的是商用的操作系统——Windows NT。

32 位 Windows 优越的性能不仅使用户得益，同时也为 Windows 应用程序开发者提供了一个发挥自己想象力的舞台，充分利用 32 位 Windows 的各种控件和多媒体支持，程序员可以开发出美观宜人，图、文、声并茂的应用程序。然而，欲充分利用 32 位 Windows 的优越性能，程序员必须对该系统所提供的各种控件、接口以及编程技术有一个较深刻的理解。本书旨在从这些方面对读者提供一些帮助。

32 位 Windows 是一个庞大的系统，它涉及多个平台——Windows NT，Windows 95 以及新近出台的 Windows 98，而每一个平台又有着其各自特有的特性和编程要求，用一本书是无法将所有的内容都涉及到的。本书主要介绍对 32 位 Windows 新增控件和多媒体的编程技术，并对多线程编程作一简单的介绍。

全书共分 16 章和 10 个附录。第 1 章通过一个简单的 32 位 Windows 应用程序向读者介绍编写 32 位 Windows 应用程序的一般方法与步骤，在这一章中还重点介绍了消息分流器，书中所有的范例程序均使用了消息分流器。由于书中所有范例程序均是利用 Borland C++ 5.02（该版的 C++编译器就是 Borland 公司著名的可视化开发平台 C++ Builder 的编译器）软件包来开发的，所以在本章还简单地介绍了该软件包的一些用法。第 2 章介绍了图像列表控件，书中有多个范例程序都要用到该控件。在本章中，作者还特意设计了一个拼图游戏，该应用程序演示了图像列表的拖放功能。第 3 章介绍工具栏控件，该控件就像窗口菜单一样，大多数应用程序都需要它。第 4 章介绍进度条控件，几乎所有的软件安装程序都用到了该控件。第 5 章介绍标签控件，第 6 章介绍属性表控件，这两种控件被广泛地用在 32 位 Windows 的

各种对话框中。第 7 章介绍跟踪条控件，该控件与滚动条控件有异曲同工之妙，但在外观上则比滚动条更生动。第 8 章介绍标题控件，该控件很适合用于数据库应用程序中。第 9 章介绍列表视图控件，Windows 95 中的文件夹就是一个该控件的应用实例。第 10 章介绍树形视图控件，用该控件来显示磁盘上的目录则是再合适不过的了，Windows 95 中的资源管理器就是一个使用该控件的实例。第 11 章介绍微调按钮控件，与编辑控件组合使用是该控件的典型用法，本章还介绍了该控件的另一些用法。第 12 章介绍了多信息文本编辑控件，该控件是对编辑控件的扩展，它不仅支持对文字的多种修饰，还支持 OLE（Object Linking and Embedding——对象链接与嵌入）。第 13 章介绍动画控件，它是 Windows 对多媒体支持的一个接口。第 14 章利用一个 CD 播放器应用程序介绍了 Windows 对多媒体支持的一个主要接口——MCI 接口。第 15 章简单介绍了 Windows 对多媒体支持的另一个接口——MCIWnd 窗口类。第 16 章简单介绍多线程编程的方法与步骤。10 个附录分别列出了本书中介绍过的 API 函数、消息、通知消息、消息分流器、结构体、宏、消息宏以及各章节中表格和范例程序的索引。

书中第 1 章和 10 个附录由王平撰写，第 15 和第 16 章由白天琪撰写，其它部分均为和克智撰写，全书由和克智主编。

本书适合已初步了解 Windows 编程，并打算进一步提高自己编程技术的读者阅读，对于尚无 Windows 编程经验的读者，作者推荐先行阅读西安交通大学出版社 1997 年 10 月出版的《实用 Windows 编程技术》（和克智著）一书。这是因为本书是以前者的续篇形式撰写的，书中许多内容与前者有关。

本书注重实用性和指导性，尽量不涉及过多的理论探讨。书中所用的范例程序基本都是作者多年所编写的 Windows 应用程序的部分内容，或它们的简写版，有的范例本身就是一个完整的实用程序。虽然所有范例程序都是利用 Borland C++ 5.02 软件包开发的，实际上它们均可以在 BC 4.5 及以后版本的编译器中编译通过。也可以利用 VC 4.0 及以后的版本来编译本书的范例程序，但个别程序需要做适当的改动，这里不再赘述。

本书配有 1 片软盘，需要者可与西安交通大学出版社发行科联系。

和克智

1998 年 11 月

目　录

第 1 章

32 位 WINDOWS 编程简述

在 32 位 Windows 中，不仅仅是基本数据的长度从 16 位扩展到了 32 位。更重要的是它提供了十多个新的公用控件（Common Control）。随之而来也就有了更多的消息，其中最明显的是伴随新公用控件的通知消息（Notification Messages）。另外，由于窗口的外观发生了变化，因而在窗口中所显示的一些要素也与 16 位 Windows 有所不同。所有一切变化，当然地影响着应用程序的设计。

随着 32 位 Windows 的广泛应用以及软件技术的发展，Windows 应用程序开发平台制造商也开发出一些新的软件包，以满足人们开发 32 位 Windows 应用程序的需要。美国 Borland International Research 公司的 Borland C++ 5.02 版（以下简称 BC502）就是一个比较优秀的软件包。它不仅可以用来开发 Windows 应用程序(包括 16 位和 32 位)，也可以用来开发 DOS 应用程序；利用该软件包，程序员既可以使用 C 语言编写面向过程的应用程序，也可以用 C++ 语言编写面向对象的应用程序，甚至还可以用汇编语言开发软件。BC502 还提供了一个 OWL 库和一个 Microsoft 公司的 MFC 库，为开发面向对象的 Windows 应用程序提供了强有力的支持。除此之外，该软件包还包含一个高效的数据库引擎（Database Engine——DBE）。利用 DBE，使得应用程序可以访问几乎所有当前国际上流行的数据库。

本章通过一个简单的 Windows 95 应用程序，对利用 BC502 软件包开发 32 位 Windows 应用程序作一简单地介绍。

1.1 一个简单的演示程序

SIMPLE 是一个简单的 32 位 Windows 应用程序，该程序演示了利用 C 语言开发 32 位应用程序的一般方法与步骤。

1.1.1 头文件 SIMPLE.H

```
/*      SIMPLE.H
        By He Kezhi, 1998.1          */

//      Macros
```

```
#define     IDM_SHOW        101
#define     IDM_REMOVE      102
#define     IDM_EXIT        103

#define     IDM_ABOUT       201

#define     IDS_MSG     301

//      Function prototypes
LRESULT CALLBACK     WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK     AboutProc(HWND, UINT, WPARAM, LPARAM);

void    Simple_OnCommand(HWND, int, HWND, UINT);
BOOL Simple_OnCreate(HWND, LPCREATESTRUCT);
void    Simple_OnDestroy(HWND);
void    Simple_OnPaint(HWND);

//      End of SIMPLE.H
```

1.1.2 头文件 ABOUT.H

```
/*      ABOUT.H
        By He Kezhi, 1998.1         */

void    About_OnCommand(HWND, int, HWND, UINT);

//      End of ABOUT.H
```

1.1.3 源程序文件 SIMPLE.C

```
/*      SIMPLE.C
        By He Kezhi, 1998.1         */

#include     <windows.h>
#include     <windowsx.h>
#include     "simple.h"

char    _szAppName[] = "Simple", _szMsg[80];

#pragma argsused
int APIENTRY     WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,,
                         LPSTR lpCmdLine, int nCmdShow)
{
        WNDCLASSEX      wndClass;
        HWND        hWnd;
        MSG   msg;

        wndClass.cbSize             = sizeof(WNDCLASSEX);
        wndClass.style              = CS_HREDRAW | CS_VREDRAW;
```

```
    wndClass.lpfnWndProc      = (WNDPROC)WndProc;
    wndClass.cbClsExtra       = 0;
    wndClass.cbWndExtra       = sizeof(WORD);
    wndClass.hInstance        = hInstance;
    wndClass.hIcon            = LoadIcon(hInstance, _szAppName);
    wndClass.hCursor          = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground    = GetStockObject(WHITE_BRUSH);
    wndClass.lpszMenuName     = _szAppName;
    wndClass.lpszClassName    = _szAppName;
    wndClass.hIconSm          = LoadIcon(hInstance, "Simple16");

    if(!RegisterClassEx(&wndClass))
        return FALSE;

    hWnd = CreateWindow(_szAppName,
                        "简单的 Windows 95 演示程序",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        NULL,
                        NULL,
                        hInstance,
                        NULL);
    if(!hWnd)
        return FALSE;

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

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

    return msg.wParam;
}

LRESULT CALLBACK    WndProc(HWND hWnd, UINT uMessage, WPARAM wParam, LPARAM lParam)
{
    switch(uMessage)    {
        HANDLE_MSG(hWnd, WM_COMMAND, Simple_OnCommand);
        HANDLE_MSG(hWnd, WM_CREATE, Simple_OnCreate);
        HANDLE_MSG(hWnd, WM_DESTROY, Simple_OnDestroy);
        HANDLE_MSG(hWnd, WM_PAINT, Simple_OnPaint);

        default:
            return DefWindowProc(hWnd, uMessage, wParam, lParam);
    }
}
```

```
#pragma argsused
void    Simple_OnCommand(HWND hWnd, int idCommand, HWND hWndCtl, UINT uCodeNotify)
{
        HMENU       hMenu;

        switch(idCommand)  {
                case    IDM_SHOW:
                        hMenu = GetMenu(hWnd);
                        EnableMenuItem(hMenu, IDM_SHOW, MF_BYCOMMAND | MF_GRAYED);
                        EnableMenuItem(hMenu, IDM_REMOVE, MF_BYCOMMAND | MF_ENABLED);

                        SetWindowWord(hWnd, 0, 1);

                        InvalidateRect(hWnd, NULL, TRUE);

                        break;

                case    IDM_REMOVE:
                        hMenu = GetMenu(hWnd);
                        EnableMenuItem(hMenu, IDM_REMOVE, MF_BYCOMMAND | MF_GRAYED);
                        EnableMenuItem(hMenu, IDM_SHOW, MF_BYCOMMAND | MF_ENABLED);

                        SetWindowWord(hWnd, 0, 0);

                        InvalidateRect(hWnd, NULL, TRUE);

                        break;

                case    IDM_EXIT:
                        DestroyWindow(hWnd);

                        break;

                case    IDM_ABOUT:
                        DialogBox(GetWindowInstance(hWnd), "About", hWnd, (DLGPROC)AboutProc);

                        break;
        }
}

BOOL Simple_OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct)
{
        if(LoadString(lpCreateStruct->hInstance, IDS_MSG, _szMsg, sizeof(_szMsg)) == 0)
                MessageBox(hWnd, "加载字符串资源失败！", "错误", MB_OK);

        SetWindowWord(hWnd, 0, 0);

        return TRUE;
}

#pragma argsused
```

```
void    Simple_OnDestroy(HWND hWnd)
{
        PostQuitMessage(0);
}

void    Simple_OnPaint(HWND hWnd)
{
        if(GetWindowWord(hWnd, 0) == 1) {
                PAINTSTRUCT     ps;
                RECT rc;

                BeginPaint(hWnd, &ps);

                GetClientRect(hWnd, &rc);
                DrawText(ps.hdc, _szMsg, -1, &rc, DT_CENTER | DT_VCENTER | DT_SINGLELINE);

                EndPaint(hWnd, &ps);
        }
}

//      End of SIMPLE.C
```

1.1.4 源程序文件 ABOUT.C

```
/*      ABOUT.C
        By He Kezhi, 1998.1         */

#include        <windows.h>
#include        <windowsx.h>
#include        "about.h"
#include        "simple.h"

LRESULT CALLBACK        AboutProc(HWND hDlg, UINT uMessage,    WPARAM wParam, LPARAM lParam)
{
        switch(uMessage)    {
                case    WM_COMMAND:
                        (void)HANDLE_WM_COMMAND(hDlg, wParam, lParam, About_OnCommand);

                        return TRUE;
        }

        return FALSE;
}

#pragma argsused
void    About_OnCommand(HWND hDlg, int idCommand, HWND hWndCtl, UINT uCodeNotify)
{
        switch(idCommand)  {
                case    IDOK:
                        EndDialog(hDlg, TRUE);
```

```
    }
}

//      End of ABOUT.C
```

1.1.5 资源描述文件 SIMPLE.RC

```
/*      SIMPLE.RC
        By He Kezhi, 1998.1          */

#include      "simple.h"

SimpleMENU
{
      POPUP "演示"
      {
            MENUITEM "显示文字", IDM_SHOW
            MENUITEM "取消文字", IDM_REMOVE, GRAYED
            MENUITEM SEPARATOR
            MENUITEM "退出", IDM_EXIT
      }

      POPUP "帮助"
      {
            MENUITEM "关于", IDM_ABOUT
      }

}

Simple        ICON  "simple.ico"
Simple16      ICON  "simple16.ico"

STRINGTABLE
{
      IDS_MSG, "这是一个简单的 Windows 95 演示程序"
}

About         DIALOGEX  22, 18, 167, 74
EXSTYLE       WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
STYLE         DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP |
                        WS_VISIBLE |        WS_CAPTION
CAPTION       "关于 Simple"
FONT          12, "System", 400, 0
{
      CONTROL "确认", IDOK, "BUTTON", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE |
                  WS_GROUP | WS_TABSTOP, 132, 29, 32, 14
      CONTROL "Simple", -1, "STATIC", SS_ICON | WS_CHILD | WS_VISIBLE, 2, 2, 16, 16
      CONTROL "一个简单的 Windows 95 应用程序", -1, "STATIC", SS_CENTER | WS_CHILD |
                  WS_VISIBLE | WS_GROUP, 30, 6, 126, 8
```

```
        CONTROL "1.00 版", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP,
                64, 20, 32, 8
        CONTROL "和克智", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP,
                56, 36, 48, 8
        CONTROL "版权所有  违者必究", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE |
                WS_GROUP, 40, 60, 100, 9
        CONTROL "", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 28, 52, 124, 1
}

//      End of SIMPLE.RC
```

1.2 做工程

BC502 软件包提供了两种将源程序转换成可执行文件的手段——使用命令行和使用集成环境。这里只对使用集成环境编译、连接应用程序的方法与步骤做一简单的介绍。

5.02 版 BC 集成环境的外观与当前国内流行的 3.1 版 BC 大同小异，但在许多用法上却大相径庭。其主要差异表现在做工程方面。该版本的工程文件不仅包含了更多的信息，而且工程文件的扩展名也从.PRJ 变成了.IDE。

1.2.1 创建工程文件

在集成环境窗口的主菜单中选择 File | New | Project...菜单项，系统将显示一个如图 1-1 所示的新工程对话框，该工程对话框主要由 4 个区域组成：

Project Path and Name 工程路径及工程名区。该区仅含一个编辑框，用户可以在编辑框中输入存放工程文件的路径和工程文件名。比如，输入：

```
c:\bc5\User\Simple\Simple
```

就指出工程文件的文件名是 Simple.ide，并要求将该文件存放在 c:\bc5\User\Simple 子目录中。若指定的子目录不存在，系统会自动创建该目录。

Target Name 目标文件名。这里所说目标文件指的是做工程后所生成的可执行文件。该区也只有一个编辑框，用户可以在该编辑框中输入目标文件名。注意：当用户在输入工程路径及工程名时，该编辑框中的信息将随用户的输入而变：它总是将用户输入的最后一个独立字符串作为工程文件名。设用户在工程路径及工程名区中输入的仍为上述字符串，则目标文件名编辑框中的内容就会变成 Simple。所以，通常不必在该编辑框中进行输入，而是在工程路径及工程名区中将路径和文件名一并输入即可。当然，用户也可以输入一个与工程文件名不同的可执行文件名，但这样作似乎没有什么理由。

Target Type 目标类型。即最终所生成的可执行文件的类型。该区由一个列表框和两个下拉式组合框组成。列表框中列出了所有可执行文件的类型，以供用户选择。本例选择的是表中的第一项——应用程序（Application）。在 Platform（程序运行平台）组合框中列有 3 种平台的 4 种形式：Win32，Win16，DOS（Standard）和 DOS（Overlay），它们分别对应 32