Windows API一日一练(1)第一个应用程序

要跟计算机进行交互，就需要计算机显示信息给人看到，或者发出声音给人听到，然后人看到或听到相应的信息后，再输入其它信息给计算机，这样就可以让计算机进行数据处理，把结果显示给我们。现在就来编写一个最简单的 Windows 应用程序，让它提示一行文字给我们看到，这就是简单的目标。

它实现的源程序和界面如下：

上面这个图，是从 VC++ 2005 里截出来的。这样可以看到源程序和显示的界面，很清楚地知道那些内容在那里显示，显示窗口里的标题是例子，就是 MessageBox 里的字符串“例子”的显示。“第一个应用程序”也是那样显示出来的。第一个应用程序是非常简单的，下面再来详细地解说每行程序的作用。

源程序如下：

```
#001  // First.cpp : 应用程序入口文件
#002  //
#003
#004  #include "stdafx.h"
#005  #include "First.h"
#006
#007  //
#008  //第一个例子。
#009  //蔡军生  2007/07/02
#010  //
#011  int APIENTRY _tWinMain(HINSTANCE hInstance,
#012                        HINSTANCE hPrevInstance,
#013                        LPTSTR    lpCmdLine,
#014                        int       nCmdShow)
#015  {
#016   UNREFERENCED_PARAMETER(hPrevInstance);
#017   UNREFERENCED_PARAMETER(lpCmdLine);
#018   UNREFERENCED_PARAMETER(hInstance);
#019   UNREFERENCED_PARAMETER(nCmdShow);
#020
#021   //获取桌面的句柄。
#022   HWND hWnd = GetDesktopWindow();
#023
#024   //显示一行消息。
#025   ::MessageBox(hWnd, _T("第一个应用程序"), _T("例子"), MB_OK);
#026
#027   //
#028   return 0;
#029  }
```

第 4 行是包含 Windows 的 API 头文件。在这个文件里包含一些系统的定义等。
第 5 行是包行 C++的头文件。
第 11 行是定义 WinMain 的入口。
第 16 行到第 19 行是指明不生产这些参数不使用的警告。
第 22 行是获取桌面的句柄。
第 25 行是显示一个窗口提示信息。
第 28 行是返回程序出错码。

Windows API一日一练(2)使用应用程序句柄

从上面这段程序就可以看到，_tWinMain 是应用程序的入口函数，这里是使用它的宏，定义在 tchar.h 头文件里，为什么要这样作宏定义的呢？由于 Windows 的应用程序要适应 UNICODE 和以前单字符的应用程序，由于 Windows 这两个 API 的定义是不一样的，如下：

UNICODE 的定义：

```
#define _tWinMain    wWinMain
```

单字符的定义：

```
#define _tWinMain    WinMain
```

只要经过这样的宏定义后，就可以适应不同字符宽度的函数接口了。由于我是采用 UNICODE 编译的，所以这里使用 wWinMain 函数定义，下面再把它的定义找出来，如下：


```
int
WINAPI
wWinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPWSTR lpCmdLine,
    int nShowCmd
    );
```

这里要详细地解释一下函数 wWinMain 的参数，它有四个参数。

hInstance 是当前应用程序的实例句柄，一般用来区分不同的资源使用。

hPrevInstance 是以前 Win98 使用的句柄，在 Win2000 以后的操作系统里都是空值 NULL。

lpCmdLine 是命令行参数，比如你在 Windows 开始菜单里运行一个程序，并添加参数在后面，就会传递给应用程序，后面再详细讨论。

nShowCmd 是窗口的显示方式，比如最大化显示，最小化显示，还是正常显示。

Windows 运行程序时，是通过运行库里的启动代码来调用 wWinMain 函数，它是在启动文件里如下调用：

```
#ifdef WPRFLAG
        mainret = wWinMain(
#else /* WPRFLAG */
        mainret = WinMain(
#endif /* WPRFLAG */
                    (HINSTANCE)&__ImageBase,
                    NULL,
                    lpszCommandLine,
                    StartupInfo.dwFlags & STARTF_USESHOWWINDOW
                     ? StartupInfo.wShowWindow
                     : SW_SHOWDEFAULT
                    );
```

这就是操作系统传递给应用程序的值，现在就来演示使用第一个参数 hInstance。

请看下面的例子：

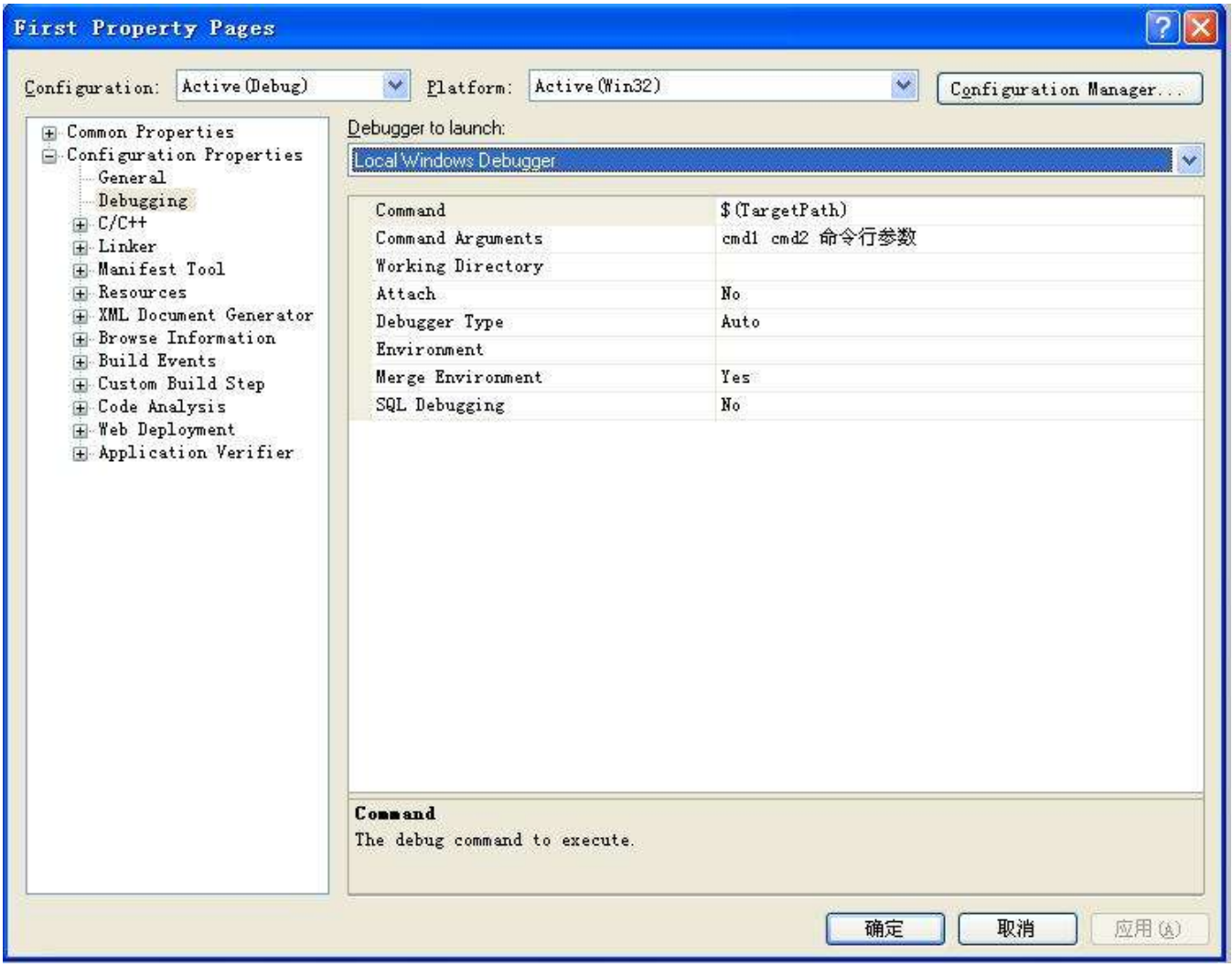
```
#001 #include "stdafx.h"
#002 #include "First.h"
#003
#004 //
#005 //第一个例子。
#006 //蔡军生 2007/07/03
```

```
#007 //
#008 int APIENTRY _tWinMain(HINSTANCE hInstance,
#009                     HINSTANCE hPrevInstance,
#010                     LPTSTR    lpCmdLine,
#011                     int       nCmdShow)
#012 {
#013     UNREFERENCED_PARAMETER(hPrevInstance);
#014     UNREFERENCED_PARAMETER(lpCmdLine);
#015     UNREFERENCED_PARAMETER(nCmdShow);
#016
#017     //使用应用程序句柄
#018     const int MAXSIZE_APPBUF = 256;
#019     TCHAR wAppTile[MAXSIZE_APPBUF];
#020     LoadString(hInstance,IDS_APP_TITLE,wAppTile,MAXSIZE_
APPBUF);
#021
#022     //获取桌面的句柄。
#023     HWND hWnd = GetDesktopWindow();
#024
#025     //显示一行消息。
#026     MessageBox(hWnd, _T("第一个应用程序"), wAppTile, MB_OK);
#027
#028     //
#029     return 0;
#030 }
```

这个例子是在前面的基础上修改的，主要添加了使用应用程序实例句柄。在第 19 行里定义了一个保存应用程序标题的缓冲区，然后在第 20 行里调用函数 **LoadString** 从应用程序的资源里加载字符串，它的第一个参数就使用到 hInstance 句柄。因此应用程序句柄是表示程序在资源上唯一的标识符。

Windows API一日一练(3)使用命令行参数

下面再接着练习使用命令行参数，先在 VC2005 调试设置里设置输入参数，如下图：



可以看到在 Command Arguments 里输入给程序传送的命令行参数(cmd1 cmd2 命令行参数)。

接着修改原来的程序如下:

```
#001 int APIENTRY _tWinMain(HINSTANCE hInstance,
#002                        HINSTANCE hPrevInstance,
#003                        LPTSTR    lpCmdLine,
#004                        int       nCmdShow)
#005 {
#006     UNREFERENCED_PARAMETER(hPrevInstance);
#007     UNREFERENCED_PARAMETER(nCmdShow);
#008
#009     //使用应用程序句柄
#010     const int MAXSIZE_APPBUF = 256;
#011     TCHAR wAppTile[MAXSIZE_APPBUF];
#012     ::LoadString(hInstance,IDS_APP_TITLE,wAppTile,MAXSIZE_APPB
UF);
#013
```

```
#014    //获取桌面的句柄。
#015    HWND hWnd = ::GetDesktopWindow();
#016
#017    //显示命令行参数。
#018    ::MessageBox(hWnd, lpCmdLine, wAppTile, MB_OK);
#019
#020
#021    //显示一行消息。
#022    ::MessageBox(hWnd, _T("第一个应用程序"), wAppTile, MB_OK);
#023
#024    //
#025    return 0;
#026 }
#027
```


在上面的程序里添加了第 18 行的代码，用来显示程序命令行的参数。它的显示结果如下：

这样就可以看到 WinMain 两个参数的使用了。现在就使用了第一个 API 函数 WinMain 了，就是这么简单地就学会了使用第一个 API 函数。

Windows API一日一练(4)MessageBox 函数

为了显示提示信息给用户，Windows 是提供了一个非常方便的 API 函数 MessageBox 给用户使用，使用这个 API 函数可以显示简单的文字信息出来，提醒或提示用户进行下一步操作。

函数声明如下：

```
WINUSERAPI
int
WINAPI
MessageBoxA(
    __in_opt HWND hWnd,
    __in_opt LPCSTR lpText,
    __in_opt LPCSTR lpCaption,
    __in UINT uType);
WINUSERAPI
int
WINAPI
MessageBoxW(
    __in_opt HWND hWnd,
    __in_opt LPCWSTR lpText,
    __in_opt LPCWSTR lpCaption,
    __in UINT uType);
#ifdef UNICODE
#define MessageBox MessageBoxW
#else
#define MessageBox MessageBoxA
#endif // !UNICODE
```

从上面可以看出，Windows 的 API 是两种声明，一种是使用到 ANSI 编码，一种是使用到 UNICODE 编码的 API 函数。通过宏定义把这两种 API 名称统一到 MessageBox 的声明。这是一种使用选择不同 API 的技术，在今后的编程里，大多数都需要使用 UNICODE 编码了，因为可以适应不同国家的语言显示，可以国际化编程，特别对于中文支持更加需要 UNICODE 编程。

下面来解释一下参数的定义：

hWnd 是指向父窗口的句柄，如果没有父窗口，可以把这个参数设置为 NULL。

lpText 是需要显示的文字。显示字符串的起始地址。

lpCaption 是在窗口上标题显示。

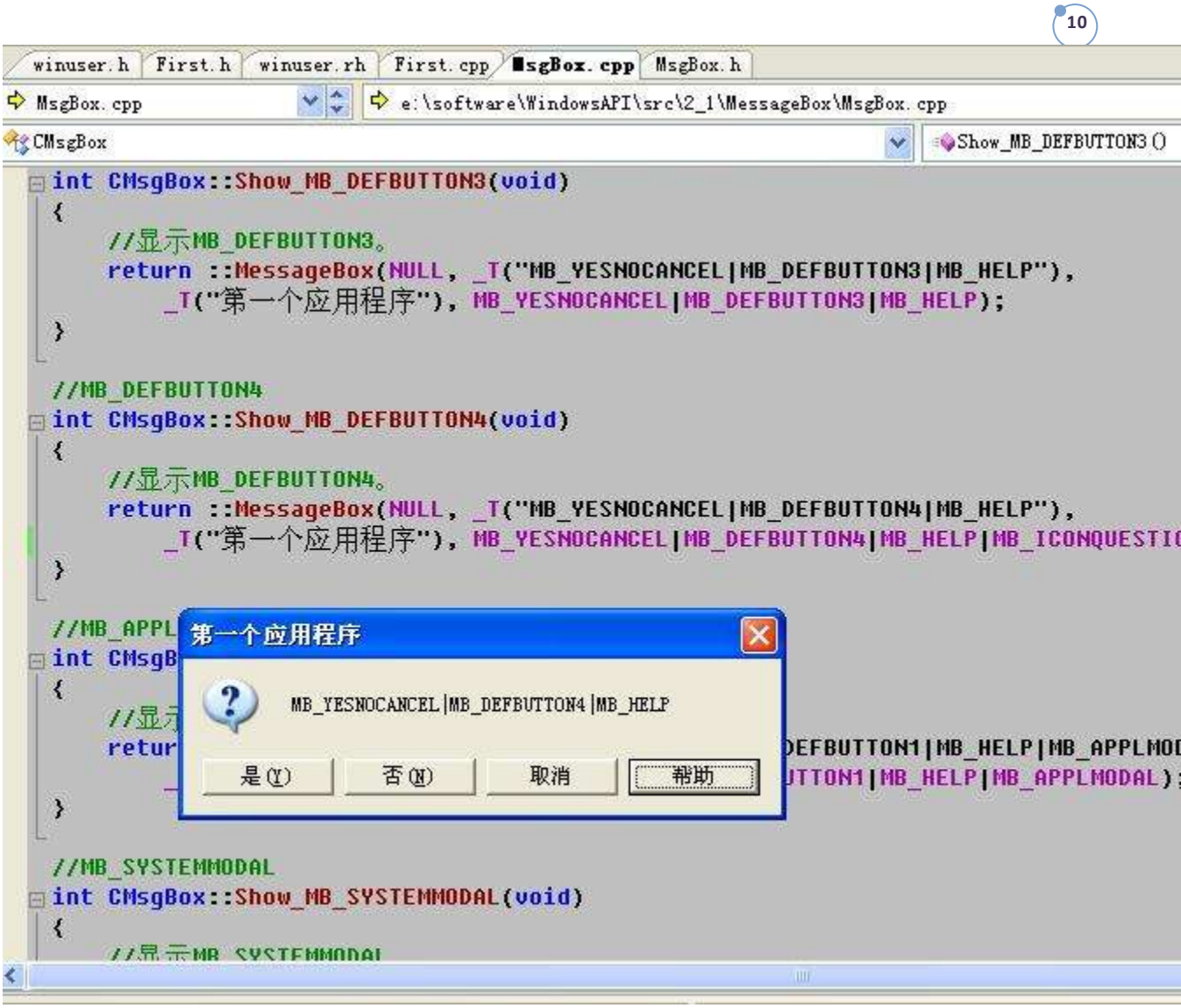
uType 是窗口组合按钮和显示图标的类型。后面再详细说明。

返回值是一个整数，如果有取消按钮，并且按下 ESC 键就返回 IDCANCEL。如果有其它按钮，并且按下，就返回相应的值。主要的值如下：

IDABORT 放弃按钮
IDCANCEL 取消按钮
IDCONTINUE 继续按钮
IDIGNORE 忽略按钮
IDNO 否按钮

IDOK 确定按钮
IDRETRY 重试按钮
IDTRYAGAIN 重试按钮
IDYES 是按钮

演示例子如下：



上面显示的代码是：

```
#001 //MB_DEFBUTTON4
#002 int CMsgBox::Show_MB_DEFBUTTON4(void)
#003 {
#004 //显示 MB_DEFBUTTON4。
#005 return ::MessageBox(NULL, _T("MB_YESNOCANCEL|MB_DEFBUTTON4|MB_HELP"),
#006         _T("第一个应用程序"), MB_YESNOCANCEL|MB_DEFBUTTON4|MB_HELP|MB_ICONQUESTION);
#007 }
```

uType 常用的选择值如下：

按钮类型：
MB_ABORTRETRYIGNORE
MB_CANCELTRYCONTINUE
MB_HELP
MB_OK
MB_OKCANCEL
MB_RETRYCANCEL
MB_YESNO
MB_YESNOCANCEL

图标类型：
MB_ICONEXCLAMATION
MB_ICONWARNING
MB_ICONINFORMATION
MB_ICONASTERISK
MB_ICONQUESTION
MB_ICONSTOP
MB_ICONERROR
MB_ICONHAND

设置缺省按钮值：
MB_DEFBUTTON1
MB_DEFBUTTON2
MB_DEFBUTTON3
MB_DEFBUTTON4

修改显示信息窗口的属性：
MB_APPLMODAL
MB_SYSTEMMODAL
MB_TASKMODAL
MB_RIGHT
MB_RTLREADING
MB_SETFOREGROUND
MB_TOPMOST
MB_SERVICE_NOTIFICATION

Windows API一日一练(5)RegisterClass 和 RegisterClassEx 函数

为了可以创建自己的窗口，就需要向 Windows 操作系统注册窗口类型，以便后面创建窗口时调用。当然，如果使用 Windows 预先注册的窗口是不需要注册的。

函数声明如下：
```
#if(WINVER >= 0x0400)
```

```
WINUSERAPI
ATOM
WINAPI
RegisterClassExA(
    __in CONST WNDCLASSEXA *);
WINUSERAPI
ATOM
WINAPI
RegisterClassExW(
    __in CONST WNDCLASSEXW *);
#ifdef UNICODE
#define RegisterClassEx  RegisterClassExW
#else
#define RegisterClassEx  RegisterClassExA
#endif // !UNICODE
```


函数的输入参数是一个 WNDCLASSEXA 或 WNDCLASSEXW 的指针。这里主要介绍 UNICODE 版本的函数定义，WNDCLASSEXW 的结构定义如下：

```
typedef struct tagWNDCLASSEXW {
    UINT        cbSize;
    /* Win 3.x */
    UINT        style;
    WNDPROC     lpfnWndProc;
    int         cbClsExtra;
    int         cbWndExtra;
    HINSTANCE   hInstance;
    HICON       hIcon;
    HCURSOR     hCursor;
    HBRUSH      hbrBackground;
    LPCWSTR     lpszMenuName;
    LPCWSTR     lpszClassName;
    /* Win 4.0 */
    HICON       hIconSm;
} WNDCLASSEXW, *PWNDCLASSEXW, NEAR *NPWNDCLASSEXW, FAR
*LPWNDCLASSEXW;
```

cbSize 是本结构的字节大小，一般设置为 sizeof(WNDCLASSEXW);
style 是窗口类型。
lpfnWndProc 是窗口处理消息的回调函数。
cbClsExtra 是窗口类型的扩展。
cbWndExtra 是窗口实例的扩展。
hInstance 是窗口实例句柄。
hIcon 是窗口图标。
hCursor 是窗口的光标。

hbrBackground 是窗口背景颜色。
lpszMenuName 是窗口菜单名称。
lpszClassName 是窗口类型的名称。
hIconSm 是窗口小图标。

调用这个函数的实例如下：


```
#001 //
#002 // 函数: MyRegisterClass()
#003 //
#004 // 目的: 注册一个窗口类型.
#005 //
#006 // 蔡军生 2007/07/12
#007 //
#008 ATOM MyRegisterClass(HINSTANCE hInstance)
#009 {
#010  WNDCLASSEX wcex;
#011
#012  wcex.cbSize = sizeof(WNDCLASSEX);
#013
#014  wcex.style          = CS_HREDRAW | CS_VREDRAW;
#015  wcex.lpfnWndProc = WndProc;
#016  wcex.cbClsExtra          = 0;
#017  wcex.cbWndExtra        = 0;
#018  wcex.hInstance      = hInstance;
#019  wcex.hIcon           = LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_TESTWIN));
#020  wcex.hCursor         = LoadCursor(NULL, IDC_ARROW);
#021  wcex.hbrBackground     = (HBRUSH)(COLOR_WINDOW+1);
#022  wcex.lpszMenuName      = MAKEINTRESOURCE(IDC_TESTWIN);
#023  wcex.lpszClassName     = szWindowClass;
#024  wcex.hIconSm        = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));
#025
#026  return RegisterClassEx(&wcex);
#027 }
```

第 10 行定义一个窗口结构的对象 wcex。
第 12 行设置窗口结构的大小。
第 14 行设置窗口类型。
第 15 行设置窗口消息处理函数 WndProc。
第 16 行设置窗口类型的扩展为空。
第 17 行设置窗口实例的扩展为空。
第 18 行设置窗口当前实例句柄 hInstance。
第 19 行设置窗口图标。

第 20 行设置光标为箭头。
第 21 行设置窗口背景颜色为白色。
第 22 行设置窗口菜单。
第 23 行设置窗口类型名称。
第 24 行设置窗口小图标。
第 26 行是调用函数 RegisterClassEx 注册这个窗口类型。

如果注册成功，返回这个窗口类型的标识号，可以用标识号进行创建窗口，查找窗口和注销窗口类型等等。如果失败返回的值是空，因此可以通过检查返回值为判断是否调用成功。

Windows API一日一练(6)CreateWindow 函数

一个窗口要显示，先要把它创建出来。那就需要调用 API 函数 CreateWindow 了，所以在注册窗口后的第二步，就需要调用创建窗口函数。

函数声明如下：

```
WINUSERAPI
HWND
WINAPI
CreateWindowExA(
    __in DWORD dwExStyle,
    __in_opt LPCSTR lpClassName,
    __in_opt LPCSTR lpWindowName,
    __in DWORD dwStyle,
    __in int X,
    __in int Y,
    __in int nWidth,
    __in int nHeight,
    __in_opt HWND hWndParent,
    __in_opt HMENU hMenu,
    __in_opt HINSTANCE hInstance,
    __in_opt LPVOID lpParam);
WINUSERAPI
HWND
WINAPI
CreateWindowExW(
    __in DWORD dwExStyle,
    __in_opt LPCWSTR lpClassName,
    __in_opt LPCWSTR lpWindowName,
    __in DWORD dwStyle,
    __in int X,
    __in int Y,
    __in int nWidth,
```

```
    __in int nHeight,
    __in_opt HWND hWndParent,
    __in_opt HMENU hMenu,
    __in_opt HINSTANCE hInstance,
    __in_opt LPVOID lpParam);
#ifdef UNICODE
#define CreateWindowEx CreateWindowExW
#else
#define CreateWindowEx CreateWindowExA
#endif // !UNICODE

#define CreateWindowA(lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)\
CreateWindowExA(0L, lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)
#define CreateWindowW(lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)\
CreateWindowExW(0L, lpClassName, lpWindowName, dwStyle, x, y,\
nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam)
#ifdef UNICODE
#define CreateWindow CreateWindowW
#else
#define CreateWindow CreateWindowA
#endif // !UNICODE
```

dwExStyle 是扩展的窗口类型。
lpClassName 是注册的窗口类型名称。
lpWindowName 是窗口名称。
dwStyle 是窗口类型。
X 是窗口左上角位置。
Y 是窗口左上角位置。
nWidth 是窗口的宽度。
nHeight 是窗口的高度。
hWndParent 是父窗口。
hMenu 是主菜单。
hInstance 是应用程序实例句柄。
lpParam 是传送给窗口的自定义参数。

调用这个函数的实例如下：

```
#001 //
#002 //   函数: InitInstance(HINSTANCE, int)
#003 //
#004 //   目的: 保存程序实例句柄，并创建窗口显示。
#005 //
```

```
#006 // 蔡军生 2007/07/12
#007 //
#008 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
#009 {
#010  HWND hWnd;
#011
#012  hInst = hInstance; // 保存实例句柄到全局变量里。
#013
#014  hWnd = CreateWindow(szWindowClass,
#015        szTitle,
#016        WS_OVERLAPPEDWINDOW,
#017        CW_USEDEFAULT,
#018        0,
#019        CW_USEDEFAULT,
#020        0,
#021        NULL,
#022        NULL,
#023        hInstance,
#024        NULL);
#025
#026  if (!hWnd)
#027  {
#028        return FALSE;
#029  }
#030
#031  ShowWindow(hWnd, nCmdShow);
#032  UpdateWindow(hWnd);
#033
#034  return TRUE;
#035 }
```


第 14 行里就是调用创建窗口函数。szWindowClass 是窗口注册名称，前面已经介绍过注册的。
第 15 行的 szTitle 是窗口显示的标题。
第 16 行是窗口显示类型。
第 17 行是缺省的左上角坐标。
第 18 行是窗口左上角坐标，由于 X 坐标设置为缺省的坐标了，所以会忽略这里的所有设置的值。
第 19 行是设置窗口的宽度。使用缺省值。
第 20 行同样忽略窗口的高度。
第 21 行是没有父窗口。
第 22 行是没有主菜单。
第 23 行是窗口当前实例句柄。
第 24 行是传递给窗口的自定义参数为空。

如果窗口创建成功就会返回这个窗口的句柄，否则返回空值。
通过这样设置，就可以创建各种各样的窗口，只要你自己喜欢的，就可以改变它。

Windows API一日一练(7)ShowWindow 函数

ShowWindow 的 API 函数是显示窗口，但它在第一次调用和以后的调用是有区别的。第一次调用时，它的输入参数 nCmdShow 是需要输入 WinMain 函数里传入来的 nCmdShow 参数，而不能是其它参数。

函数声明如下：

```
WINUSERAPI
BOOL
WINAPI
ShowWindow(
    __in HWND hWnd,
    __in int nCmdShow);
```

hWnd 是窗口的句柄。
nCmdShow 是窗口显示的状态。可能设置的值如下：
SW_FORCEMINIMIZE 是强制窗口最小化，主要使用在非窗口主线程的其它线程来操作。
SW_HIDE 是显示窗口为隐藏状态。
SW_MAXIMIZE 是显示窗口为最大化。
SW_MINIMIZE 是显示窗口为最小化。
SW_RESTORE 是从任务里恢复窗口显示。
SW_SHOW 是激活窗口为当前窗口，并且显示为当前的大小和位置。
SW_SHOWDEFAULT 是创建进程时显示窗口的值。
SW_SHOWMAXIMIZED 是激活窗口为当前窗口，并且显示最大化。
SW_SHOWMINIMIZED 是激活窗口为当前窗口，并且显示最小化。
SW_SHOWMINNOACTIVE 是显示窗口为最小化，但不激活它作为当前窗口。
SW_SHOWNA 是显示为当前的大小和位置，但不激活它作为当前窗口。
SW_SHOWNOACTIVATE 是显示当前窗口，但不激活它作为当前窗口。
SW_SHOWNORMAL 是显示当前窗口，但窗口是最小化或最大化时会恢复窗口为原来的大小和位置。

调用这个函数的实例如下：

```
#001 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
#002 {
#003  HWND hWnd;
#004
#005  hInst = hInstance; // 保存实例句柄到全局变量里。
#006
#007  hWnd = CreateWindow(szWindowClass,
#008         szTitle,
#009         WS_OVERLAPPEDWINDOW,
#010         CW_USEDEFAULT,
```

```
#011        0,
#012        CW_USEDEFAULT,
#013        0,
#014        NULL,
#015        NULL,
#016        hInstance,
#017        NULL);
#018
#019  if (!hWnd)
#020  {
#021        return FALSE;
#022  }
#023
#024  ShowWindow(hWnd, nCmdShow);
#025  UpdateWindow(hWnd);
#026
#027  //
#028  //蔡军生 2007/07/14
#029  //显示窗口测试。
#030  MessageBox(NULL,_T("最大化"),_T("测试"),MB_OK);
#031
#032  //显示窗口为最大化。
#033  ShowWindow(hWnd, SW_SHOWMAXIMIZED);
#034
#035
#036  MessageBox(NULL,_T("原来位置"),_T("测试"),MB_OK);
#037
#038  //显示窗口为原来位置。
#039  ShowWindow(hWnd, SW_SHOWNORMAL);
#040
#041
#042  return TRUE;
#043 }
```

在第 24 行里先调用这个函数显示创建进程的窗口状态。
第 33 行里显示窗口为最大化。
第 39 行里显示窗口为原来的状态。

这样就可以掌握了 ShowWindow 函数的使用。

Windows API一日一练(8)UpdateWindow 函数

UpdateWindow 函数是更新窗口的客户区，主要通过发送 WM_PAINT 消息来实现的。

函数声明如下：

```
WINUSERAPI
BOOL
WINAPI
UpdateWindow(
    __in HWND hWnd);
```

hWnd 是要更新窗口客户区的窗口句柄。

调用这个函数的实例如下：

```
#001 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
#002 {
#003  HWND hWnd;
#004
#005  hInst = hInstance; // 保存实例句柄到全局变量里。
#006
#007  hWnd = CreateWindow(szWindowClass,
#008        szTitle,
#009        WS_OVERLAPPEDWINDOW,
#010        CW_USEDEFAULT,
#011        0,
#012        CW_USEDEFAULT,
#013        0,
#014        NULL,
#015        NULL,
#016        hInstance,
#017        NULL);
#018
#019  if (!hWnd)
#020  {
#021        return FALSE;
#022  }
#023
#024  ShowWindow(hWnd, nCmdShow);
#025  UpdateWindow(hWnd);
#026
#027  //
#028  //蔡军生 2007/07/14
#029  //显示窗口测试。
#030  MessageBox(NULL,_T("最大化"),_T("测试"),MB_OK);
#031
#032  //显示窗口为最大化。
#033  ShowWindow(hWnd, SW_SHOWMAXIMIZED);
#034
#035
```

```
#036  MessageBox(NULL,_T("原来位置"),_T("测试"),MB_OK);
#037
#038  //显示窗口为原来位置。
#039  ShowWindow(hWnd, SW_SHOWNORMAL);
#040
#041
#042  return TRUE;
#043 }
```

第 25 行就是调用 UpdateWindow 函数来更新窗口的客户区。

一般创建窗口之后都需要调用 UpdateWindow 函数来更新窗口客户区的显示，否则是乱糟糟。

Windows API一日一练(9)WindowProc 和 DefWindowProc 函数

在 Windows 操作系统里，当窗口显示之后，它就可以接收到系统源源不断地发过来的消息，然后窗口就需要处理这些消息，因此就需要一个函数来处理这些消息。在 API 里定义了一个函数为回调函数，当系统需要向窗口发送消息时，就会调用窗口给出的回调函数 WindowProc，如果 WindowProc 函数不处理这个消息，就可以把它转向 DefWindowProc 函数来处理，这是系统的默认消息处理函数。当你按下菜单，或者点击窗口时，窗口需要运行这个消息处理函数。

函数 WindowProc 声明如下：

```
LRESULT CALLBACK WindowProc(        HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam
);
```

hwnd 是当前窗口的句柄。

uMsg 是系统发过来的消息。

wParam 是消息参数。

lParam 是消息参数。

这个函数一定是静态函数，也就是全局函数，在编译时已经确定了地址。由于它需要设置在注册的窗口类型里，如下：

```
#008 ATOM MyRegisterClass(HINSTANCE hInstance)
#009 {
#010  WNDCLASSEX wcex;
#011
#012  wcex.cbSize = sizeof(WNDCLASSEX);
#013
#014  wcex.style          = CS_HREDRAW | CS_VREDRAW;
#015  wcex.lpfnWndProc = WndProc;
```

第 15 行就是设置窗口的消息处理函数。

函数 DefWindowProc 声明如下：

```
LRESULT DefWindowProc(          HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam
);
```

这个函数参数跟上面那个函数是一样的。

只不过，它是处理所有默认的消息。

调用这两个函数的实例如下：

```
#001 //
#002 // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)
#003 //
#004 // 目的: 处理主窗口的消息.
#005 //
#006 // 蔡军生 2007/07/12
#007 //
#008 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
#009 {
#010  int wmId, wmEvent;
#011  PAINTSTRUCT ps;
#012  HDC hdc;
#013
#014  switch (message)
#015  {
#016  case WM_COMMAND:
#017        wmId    = LOWORD(wParam);
#018        wmEvent = HIWORD(wParam);
#019        // 菜单选项命令响应:
#020        switch (wmId)
#021        {
#022        case IDM_ABOUT:
#023             DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX),
hWnd, About);
#024             break;
#025        case IDM_EXIT:
#026             DestroyWindow(hWnd);
#027             break;
#028        default:
#029             return DefWindowProc(hWnd, message, wParam, lParam);
#030        }
#031        break;
#032  case WM_PAINT:
```

```
#033        hdc = BeginPaint(hWnd, &ps);
#034        //
#035        EndPaint(hWnd, &ps);
#036        break;
#037  case WM_DESTROY:
#038        PostQuitMessage(0);
#039        break;
#040  default:
#041        return DefWindowProc(hWnd, message, wParam, lParam);
#042  }
#043  return 0;
#044 }
```

第 8 行定义消息处理函数
第 14 行开始根据不同的消息作处理。
第 29 行和第 41 行都是调用 DefWindowProc 函数来处理未处理的消息。

有了窗口消息处理函数，就可以响应不同的消息，实现各种各样的功能。

Windows API一日一练(10)LoadAccelerators 函数

当用户使用软件时，往往有些功能是最常用的功能。作为开发人员，就需要让用户感觉到这个软件好用,这样就需要把他们最常用的功能用起来最方便,最快捷,能提高生产效率。在这方面，微软是做到家了，比如在键盘上有一个 Windows 键，Windows 很多功能都可以通过这个键与其它键来组合成快捷键，提高使用的方便性。现在就来介绍一下怎么样让你的程序也方便使用，就是使用快捷键。
函数 LoadAccelerators 声明如下：

```
WINUSERAPI
HACCEL
WINAPI
LoadAcceleratorsA(
    __in_opt HINSTANCE hInstance,
    __in LPCSTR lpTableName);
WINUSERAPI
HACCEL
WINAPI
LoadAcceleratorsW(
    __in_opt HINSTANCE hInstance,
    __in LPCWSTR lpTableName);
#ifdef UNICODE
#define LoadAccelerators LoadAcceleratorsW
#else
#define LoadAccelerators LoadAcceleratorsA
#endif // !UNICODE
```

hInstance 是应用程序实例句柄，用来从程序的资源文件里查找到快捷键定义。
lpTableName 是快捷键的定义表格。

调用这个函数的实例如下：

```
#001 //主程序入口
#002 //
#003 // 蔡军生 2007/07/12
#004 //
#005 int APIENTRY _tWinMain(HINSTANCE hInstance,
#006                       HINSTANCE hPrevInstance,
#007                       LPTSTR    lpCmdLine,
#008                       int       nCmdShow)
#009 {
#010  UNREFERENCED_PARAMETER(hPrevInstance);
#011  UNREFERENCED_PARAMETER(lpCmdLine);
#012
#013   //
#014  MSG msg;
#015  HACCEL hAccelTable;
#016
#017  // 加载全局字符串。
#018  LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
#019  LoadString(hInstance, IDC_TESTWIN, szWindowClass,
MAX_LOADSTRING);
#020  MyRegisterClass(hInstance);
#021
#022  // 应用程序初始化:
#023  if (!InitInstance (hInstance, nCmdShow))
#024  {
#025      return FALSE;
#026  }
#027
#028  hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_TESTWIN));
#029
#030  // 消息循环:
#031  while (GetMessage(&msg, NULL, 0, 0))
#032  {
#033      if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
#034      {
#035          TranslateMessage(&msg);
#036          DispatchMessage(&msg);
#037      }
#038  }
```

```
#039
#040  return (int) msg.wParam;
#041 }
```

第 28 行就是调用函数 LoadAccelerators 从资源里加载快捷键。

到这里，就已经了解和使用 LoadAccelerators 函数了。但还有一个问题，就是快捷键的资源是怎么样定义的呢？问得好，下面就带你看一下它的定义：


```
#001 //////////////////////////////////////////////////////////////////////////////
#002 //
#003 // Accelerator
#004 //
#005
#006 IDC_TESTWIN ACCELERATORS
#007 BEGIN
#008     "?",            IDM_ABOUT,              ASCII, ALT
#009     "/",            IDM_ABOUT,              ASCII, ALT
#010 END
#011
```

这是从资源文件 TestWin.rc 里拷贝出来的。IDC_TESTWIN 是快捷键表的名称，ACCELERATORS 是快捷键定义的关键字，BEGIN 是表示快捷键的开始，END 是表示快捷键的结束。

下面快捷键定义语法：

acctablename ACCELERATORS [optional-statements] {event, idvalue, [type] [options]... }

[optional-statements]是可选的选项。

event 是必须有的内容，它是定义的键码，或者键的 ASCII 码。

idvalue 是快捷键响应的 ID 命令。

type 是类型选择。

上面第 8 行和第 9 行里就是设置 ALT+‘?’或者 ALT + ‘/’作为快捷键，当用户按下 ALT+‘?’或者 ALT + ‘/’时就会弹出关于对话框。

Windows API一日一练(11)GetMessage 函数

应用程序为了获取源源不断的消息，就需要调用函数 GetMessage 来实现，因为所有在窗口上的输入消息，都会放到应用程序的消息队列里，然后再发送给窗口回调函数处理。

函数 GetMessage 声明如下：

```
WINUSERAPI
BOOL
WINAPI
GetMessageA(
    __out LPMSG lpMsg,
```

```
    __in_opt HWND hWnd,
    __in UINT wMsgFilterMin,
    __in UINT wMsgFilterMax);
WINUSERAPI
BOOL
WINAPI
GetMessageW(
    __out LPMSG lpMsg,
    __in_opt HWND hWnd,
    __in UINT wMsgFilterMin,
    __in UINT wMsgFilterMax);
#ifdef UNICODE
#define GetMessage GetMessageW
#else
#define GetMessage GetMessageA
#endif // !UNICODE
```


lpMsg 是从线程消息队列里获取到的消息指针。
hWnd 是想获取那个窗口的消息，当设置为 NULL 时是获取所有窗口的消息。
wMsgFilterMin 是获取消息的 ID 编号最小值，如果小于这个值就不获取回来。
wMsgFilterMax 是获取消息的 ID 编号最大值，如果大于这个值就不获取回来。
函数返回值可能是 0，大于 0，或者等于-1。如果成功获取一条非 WM_QUIT 消息时，就返回大于 0 的值；如果获取 WM_QUIT 消息时，就返回值 0 值。如果出错就返回-1 的值。

调用这个函数的例子如下：

```
#001 //主程序入口
#002 //
#003 // 蔡军生 2007/07/19
#004 // QQ: 9073204
#005 //
#006 int APIENTRY _tWinMain(HINSTANCE hInstance,
#007                       HINSTANCE hPrevInstance,
#008                       LPTSTR    lpCmdLine,
#009                       int       nCmdShow)
#010 {
#011  UNREFERENCED_PARAMETER(hPrevInstance);
#012  UNREFERENCED_PARAMETER(lpCmdLine);
#013
#014   //
#015  MSG msg;
#016  HACCEL hAccelTable;
#017
#018  // 加载全局字符串。
#019  LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

```
#020  LoadString(hInstance, IDC_TESTWIN, szWindowClass,
MAX_LOADSTRING);
#021  MyRegisterClass(hInstance);
#022
#023  // 应用程序初始化:
#024  if (!InitInstance (hInstance, nCmdShow))
#025  {
#026        return FALSE;
#027  }
#028
#029  hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_TESTWIN));
#030
#031  // 消息循环:
#032  BOOL bRet;
#033  while ( (bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
#034  {
#035        if (bRet == -1)
#036        {
#037            //处理出错。
#038
#039        }
#040        else if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
#041        {
#042            TranslateMessage(&msg);
#043            DispatchMessage(&msg);
#044        }
#045  }
#046
#047  return (int) msg.wParam;
#048 }
#049
```

第 33 行就是获取所有窗口的消息回来。

Windows API一日一练(12)TranslateAccelerator 函数

当应用程序运行时，用户按下快捷键，这样就产生了一个按键消息，那么 Windows 是怎么样把它转化为快捷键响应的消息呢？这就需要使用 TranslateAccelerator 函数。
TranslateAccelerator 函数主要的作用就是把消息跟快捷键表里定义的按键进行比较，如果发现有快捷键，就会把这个按键消息转换为 WM_COMMAND 或者 WM_SYSCOMMAND 消息给窗口的消息处理函数发送过去。
函数 TranslateAccelerator 声明如下：
WINUSERAPI

```
int
WINAPI
TranslateAcceleratorA(
    __in HWND hWnd,
    __in HACCEL hAccTable,
    __in LPMSG lpMsg);
WINUSERAPI
int
WINAPI
TranslateAcceleratorW(
    __in HWND hWnd,
    __in HACCEL hAccTable,
    __in LPMSG lpMsg);
#ifdef UNICODE
#define TranslateAccelerator TranslateAcceleratorW
#else
#define TranslateAccelerator TranslateAcceleratorA
#endif // !UNICODE
```

hWnd 是窗口句柄。
hAccTable 是快捷键定义表。
lpMsg 是当前消息。
如果函数调用成功就返回非 0 值。如果失败就返回 0 值。

调用这个函数的例子如下：

```
#001 //主程序入口
#002 //
#003 // 蔡军生 2007/07/19
#004 // QQ: 9073204
#005 //
#006 int APIENTRY _tWinMain(HINSTANCE hInstance,
#007                       HINSTANCE hPrevInstance,
#008                       LPTSTR    lpCmdLine,
#009                       int       nCmdShow)
#010 {
#011  UNREFERENCED_PARAMETER(hPrevInstance);
#012  UNREFERENCED_PARAMETER(lpCmdLine);
#013
#014   //
#015  MSG msg;
#016  HACCEL hAccelTable;
#017
#018  // 加载全局字符串。
#019  LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

```
#020  LoadString(hInstance, IDC_TESTWIN, szWindowClass,
MAX_LOADSTRING);
#021  MyRegisterClass(hInstance);
#022
#023  // 应用程序初始化:
#024  if (!InitInstance (hInstance, nCmdShow))
#025  {
#026        return FALSE;
#027  }
#028
#029  hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_TESTWIN));
#030
#031  // 消息循环:
#032  BOOL bRet;
#033  while ( (bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
#034  {
#035        if (bRet == -1)
#036        {
#037             //处理出错。
#038
#039        }
#040        else if (!TranslateAccelerator(msg.hwnd, hAccelTable,
&msg))
#041       {
#042             TranslateMessage(&msg);
#043             DispatchMessage(&msg);
#044        }
#045  }
#046
#047  return (int) msg.wParam;
#048 }
#049
```


第 40 行就是调用函数 TranslateAccelerator 来处理快捷键，并转换为命令消息发送出去。

Windows API一日一练(13)TranslateMessage 函数

TranslateMessage 是用来把虚拟键消息转换为字符消息。由于 Windows 对所有键盘编码都是采用虚拟键的定义，这样当按键按下时，并不得字符消息，需要键盘映射转换为字符的消息。

TranslateMessage 函数用于将虚拟键消息转换为字符消息。字符消息被投递到调用线程的消息队列中，当下一次调用 GetMessage 函数时被取出。当我们敲击键盘上的某个字符

键时，系统将产生 WM_KEYDOWN 和 WM_KEYUP 消息。这两个消息的附加参数（wParam 和 lParam）包含的是虚拟键代码和扫描码等信息，而我们在程序中往往需要得到某个字符的 ASCII 码，TranslateMessage 这个函数就可以将 WM_KEYDOWN 和 WM_ KEYUP 消息的组合转换为一条 WM_CHAR 消息（该消息的 wParam 附加参数包含了字符的 ASCII 码），并将转换后的新消息投递到调用线程的消息队列中。注意，TranslateMessage 函数并不会修改原有的消息，它只是产生新的消息并投递到消息队列中。
也就是说 TranslateMessage 会发现消息里是否有字符键的消息，如果有字符键的消息，就会产生 WM_CHAR 消息，如果没有就会产生什么消息。

函数 TranslateMessage 声明如下：

```
WINUSERAPI
BOOL
WINAPI
TranslateMessage(
    __in CONST MSG *lpMsg);
```

lpMsg 是检查需要转换的消息。

调用这个函数的例子如下：

```
#001 //主程序入口
#002 //
#003 // 蔡军生 2007/07/19
#004 // QQ: 9073204
#005 //
#006 int APIENTRY _tWinMain(HINSTANCE hInstance,
#007                       HINSTANCE hPrevInstance,
#008                       LPTSTR    lpCmdLine,
#009                       int       nCmdShow)
#010 {
#011  UNREFERENCED_PARAMETER(hPrevInstance);
#012  UNREFERENCED_PARAMETER(lpCmdLine);
#013
#014   //
#015  MSG msg;
#016  HACCEL hAccelTable;
#017
#018  // 加载全局字符串。
#019  LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
#020  LoadString(hInstance, IDC_TESTWIN, szWindowClass,
MAX_LOADSTRING);
#021  MyRegisterClass(hInstance);
#022
#023  // 应用程序初始化:
#024  if (!InitInstance (hInstance, nCmdShow))
#025  {
```

```
#026        return FALSE;
#027  }
#028
#029  hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_TESTWIN));
#030
#031  // 消息循环:
#032  BOOL bRet;
#033  while ( (bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
#034  {
#035         if (bRet == -1)
#036        {
#037            //处理出错。
#038
#039        }
#040        else if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
#041        {
#042            TranslateMessage(&msg);
#043            DispatchMessage(&msg);
#044        }
#045  }
#046
#047  return (int) msg.wParam;
#048 }
#049
```

第 42 行是调用函数 TranslateMessage 作消息转换工作。

Windows API一日一练(14)DispatchMessage 函数

前面已经介绍从系统队列里获取一条消息，然后经过快捷键的函数检查，又通过字符消息函数的转换，最后要做的事情就是调用 DispatchMessage 函数，它的意思就是说要把这条消息发送到窗口里的消息处理函数 WindowProc。

函数 DispatchMessage 声明如下：

```
WINUSERAPI
LRESULT
WINAPI
DispatchMessageA(
    __in CONST MSG *lpMsg);
WINUSERAPI
LRESULT
WINAPI
```

```
DispatchMessageW(
    __in CONST MSG *lpMsg);
#ifdef UNICODE
#define DispatchMessage DispatchMessageW
#else
#define DispatchMessage DispatchMessageA
#endif // !UNICODE
```


lpMsg 是指向想向消息处理函数 WindowProc 发送的消息。

调用这个函数的例子如下：

```
#001 //主程序入口
#002 //
#003 // 蔡军生 2007/07/19
#004 // QQ: 9073204
#005 //
#006 int APIENTRY _tWinMain(HINSTANCE hInstance,
#007                       HINSTANCE hPrevInstance,
#008                       LPTSTR    lpCmdLine,
#009                       int       nCmdShow)
#010 {
#011  UNREFERENCED_PARAMETER(hPrevInstance);
#012  UNREFERENCED_PARAMETER(lpCmdLine);
#013
#014   //
#015  MSG msg;
#016  HACCEL hAccelTable;
#017
#018  // 加载全局字符串。
#019  LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
#020  LoadString(hInstance, IDC_TESTWIN, szWindowClass,
MAX_LOADSTRING);
#021  MyRegisterClass(hInstance);
#022
#023  // 应用程序初始化:
#024  if (!InitInstance (hInstance, nCmdShow))
#025  {
#026       return FALSE;
#027  }
#028
#029  hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_TESTWIN));
#030
#031  // 消息循环:
```

```
#032  BOOL bRet;
#033  while ( (bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
#034  {
#035        if (bRet == -1)
#036        {
#037            //处理出错。
#038
#039        }
#040        else if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
#041        {
#042            TranslateMessage(&msg);
#043            DispatchMessage(&msg);
#044        }
#045  }
#046
#047  return (int) msg.wParam;
#048 }
#049
```

第 43 行就是调用函数 DispatchMessage 发送消息。

Windows API一日一练(15)PostQuitMessage 函数

自然界面里，各种生物都是有其生命周期的。程序也是有其生命周期的，创建时就是它出生了，当它运行工作中就是成年期，最后少不了要死亡的，那么程序的死亡是怎么样出现的呢？像以前介绍函数 GetMessage 里是使用一个循环不断地检测消息，周止复始的，是不可能出现死亡的，但它会检测到消息 WM_QUIT 就退出来。那现在问题是谁发送 WM_QUIT 消息出来呢？这就是 PostQuitMessage 函数所做的工作。当你点击窗口右上角的关闭时，Windows 就会把窗口从系统里删除，这时就会发出消息 WM_DESTROY 给窗口消息处理函数 WindowProc，WindowProc 收到这条消息后，最需要做的一件事情就是调用 PostQuitMessage 发出退出消息，让消息循环结束。

函数 PostQuitMessage 声明如下：

```
WINUSERAPI
VOID
WINAPI
PostQuitMessage(
    __in int nExitCode);
```

nExitCode 是退出标识码，它被放到 WM_QUIT 消息的参数 wParam 里。

调用这个函数的例子如下：

```
#001 //
#002 // 目的: 处理主窗口的消息.
```

```
#003 //
#004 // 蔡军生 2007/07/12    QQ:9073204
#005 //
#006 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
#007 {
#008  int wmId, wmEvent;
#009  PAINTSTRUCT ps;
#010  HDC hdc;
#011
#012  switch (message)
#013  {
#014  case WM_COMMAND:
#015        wmId    = LOWORD(wParam);
#016        wmEvent = HIWORD(wParam);
#017        // 菜单选项命令响应:
#018        switch (wmId)
#019        {
#020        case IDM_ABOUT:
#021            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX),
hWnd, About);
#022            break;
#023        case IDM_EXIT:
#024            DestroyWindow(hWnd);
#025            break;
#026        default:
#027            return DefWindowProc(hWnd, message, wParam, lParam);
#028        }
#029        break;
#030  case WM_PAINT:
#031        hdc = BeginPaint(hWnd, &ps);
#032        //
#033        EndPaint(hWnd, &ps);
#034        break;
#035  case WM_DESTROY:
#036        PostQuitMessage(0);
#037        break;
#038  default:
#039        return DefWindowProc(hWnd, message, wParam, lParam);
#040  }
#041  return 0;
#042 }
```

第 36 行就是调用函数 PostQuitMessage 来处理退出应用程序。

Windows API一日一练(16)BeginPaint和EndPaint函数

当人们使用软件时，大多数是想看到自己所需要的结果，比如玩 RPG 游戏，就是想看到自己所操作的主角做各种各样的事情。在 2D 的 RPG 游戏里，其实做的事情，就是不断地更新画面，也就是不断地显示 BMP 的图片。在普通的程序里，大多也是显示各种文本和图片的，但是有一种类型的应用程序是不怎么显示结果的，那就是服务程序。不管怎么样，只要我们想看到程序所执行后的结果，就需要在程序里显示出来。也就是需要调用 BeginPaint 和 EndPaint 函数。BeginPaint 函数的作用是告诉 Windows 系统，要开始向显示卡输出内容了，把这次显示的操作请求放到系统显示队列里。由于系统上的显示卡往往只有一个，那么这种资源是独占的，所以操作系统会让显示操作线性化，保证每个窗口的显示是独立进行的，而不是 A 窗口显示一部份，或者 B 窗口显示一部份，而是 A 窗口显示完成后再让 B 窗口显示。因此，BeginPaint 函数就是跟操作系统说，我需要显示了，你安排好吧。当 BeginPaint 返回时，就获取到系统的显示资源句柄，这样就可以调 GDI 一大堆函数来操作了。显示完成后，一定要记得调用函数 EndPaint，因为使用 BeginPaint 函数请求了独占的显示资源后，如果不释放回去，就会让其它程序永远获取不到显示资源了，这样系统就死锁了。如果你有空仔细地查看一下 Windows 源程序，就会发现 BeginPaint 函数和 EndPaint 函数怎样构成的。比如在调用 BeginPaint 函数时先把光标隐藏起来，接着再显示用户显示的东西，最后调用 EndPaint 函数后，又把隐藏的光标显示出来。

函数 BeginPaint 函数和 EndPaint 函数声明如下：

```
WINUSERAPI
HDC
WINAPI
BeginPaint(
    __in HWND hWnd,
    __out LPPAINTSTRUCT lpPaint);

WINUSERAPI
BOOL
WINAPI
EndPaint(
    __in HWND hWnd,
    __in CONST PAINTSTRUCT *lpPaint);
```

hWnd 是窗口句柄。

lpPaint 是获取显示参数。它的结构定义如下：

```
typedef struct tagPAINTSTRUCT {
 HDC hdc;
 BOOL fErase;
 RECT rcPaint;
 BOOL fRestore;
 BOOL fIncUpdate;
 BYTE rgbReserved[32];
```

} PAINTSTRUCT, *PPAINTSTRUCT;
hdc 是获取设备句柄。
fErase 是否擦新背景。
rcPaint 是显示的窗口大小。
fRestore、***fIncUpdate***、***rgbReserved*** 是保留使用的参数。

BeginPaint 函数的返回值也是显示设备的句柄。

调用这个函数的例子如下：

```
#001 //
#002 // 目的: 处理主窗口的消息.
#003 //
#004 // 蔡军生 2007/07/12    QQ:9073204
#005 //
#006 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
#007 {
#008  int wmId, wmEvent;
#009  PAINTSTRUCT ps;
#010  HDC hdc;
#011
#012  switch (message)
#013  {
#014  case WM_COMMAND:
#015        wmId    = LOWORD(wParam);
#016        wmEvent = HIWORD(wParam);
#017        // 菜单选项命令响应:
#018        switch (wmId)
#019        {
#020        case IDM_ABOUT:
#021            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX),
hWnd, About);
#022            break;
#023        case IDM_EXIT:
#024            DestroyWindow(hWnd);
#025            break;
#026        default:
#027            return DefWindowProc(hWnd, message, wParam, lParam);
#028        }
#029        break;
#030  case WM_PAINT:
#031        hdc = BeginPaint(hWnd, &ps);
#032        //
#033         EndPaint(hWnd, &ps);
```

```
#034         break;
#035  case WM_DESTROY:
#036         PostQuitMessage(0);
#037         break;
#038  default:
#039         return DefWindowProc(hWnd, message, wParam, lParam);
#040  }
#041  return 0;
#042 }
```

第 31 行调用函数 BeginPaint。
第 33 行调用函数 EndPaint。

Windows API一日一练(17)DialogBox 和 DialogBoxParam 函数

对话框是比较常用的窗口，比如当你想让用户输入一些参数时就可以使用对话框。或者提示一些警告的信息，都是可以使用对话框的。比如当你拷贝文件时，Windows 就会提示一个拷贝文件的进度对话框。对话框的使用范围比较广，并且它在设计时就可以看到运行的结果模样，这样方便设计。但对话框又分为两类，一种对话框运行后，一定要用户关闭那个对话框后才能返回到父窗口；一种对话框是不需要关闭后就可以直接返回父窗口。因此，软件开发人员就要考虑这个对话框的结果是否会影响后面的操作，如果这个对话框的结果跟后面的操作没有因果关系的，可以设置为第二种对话框。像拷贝文件的对话框就是第二种的对话框，称作无模式的对话框。如果设置为第一类，非要等那里拷贝文件才可以去操作其它东西，那么 Windows 就不方便使用了，这样会浪费大量的时间。让人等待，就是一个不好用的软件，所以软件开发人员设计软件时，要站在用户的立场思考问题，在保持软件正确的情况下，不要让人等待，任何让人等待超过 20 秒以上的软件，会让用户烦躁不安。如果非要等待的话，也要加入进度条对话框提示，这样可以有效地缓解用户烦躁不安的心情。这就跟你去银行排队时，可以坐在那里看着电视，感觉不到时间长的道理一样的。

函数 DialogBox 函数和 DialogBoxParam 函数声明如下：

```
#define DialogBoxA(hInstance, lpTemplate, hWndParent, lpDialogFunc) \
DialogBoxParamA(hInstance, lpTemplate, hWndParent, lpDialogFunc, 0L)

#define DialogBoxW(hInstance, lpTemplate, hWndParent, lpDialogFunc) \
DialogBoxParamW(hInstance, lpTemplate, hWndParent, lpDialogFunc, 0L)

#ifdef UNICODE
#define DialogBox DialogBoxW
#else
#define DialogBox DialogBoxA
#endif // !UNICODE

WINUSERAPI
INT_PTR
```

```
WINAPI
DialogBoxParamA(
   __in_opt HINSTANCE hInstance,
    __in LPCSTR lpTemplateName,
    __in_opt HWND hWndParent,
    __in_opt DLGPROC lpDialogFunc,
    __in LPARAM dwInitParam);

WINUSERAPI
INT_PTR
WINAPI
DialogBoxParamW(
    __in_opt HINSTANCE hInstance,
    __in LPCWSTR lpTemplateName,
    __in_opt HWND hWndParent,
    __in_opt DLGPROC lpDialogFunc,
    __in LPARAM dwInitParam);

#ifdef UNICODE
#define DialogBoxParam DialogBoxParamW
#else
#define DialogBoxParam DialogBoxParamA
#endif // !UNICODE
```


hInstance 是当前应用程序的实例句柄。
lpTemplateName 是对话框的资源模板。
hWndParent 是父窗口的句柄。
lpDialogFunc 是对话框的消息处理函数。
dwInitParam 是初始化参数，这里缺省设置为 0。

调用这个函数的例子如下：

```
#001 //
#002 // 目的: 处理主窗口的消息.
#003 //
#004 // 蔡军生 2007/07/12    QQ:9073204
#005 //
#006 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
#007 {
#008  int wmId, wmEvent;
#009  PAINTSTRUCT ps;
#010  HDC hdc;
#011
#012  switch (message)
#013  {
```

```
#014  case WM_COMMAND:
#015        wmId    = LOWORD(wParam);
#016        wmEvent = HIWORD(wParam);
#017        // 菜单选项命令响应:
#018        switch (wmId)
#019        {
#020        case IDM_ABOUT:
#021            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX),
hWnd, About);
#022            break;
#023        case IDM_EXIT:
#024            DestroyWindow(hWnd);
#025            break;
#026        default:
#027            return DefWindowProc(hWnd, message, wParam, lParam);
#028        }
#029        break;
#030  case WM_PAINT:
#031        hdc = BeginPaint(hWnd, &ps);
#032        //
#033        EndPaint(hWnd, &ps);
#034        break;
#035  case WM_DESTROY:
#036        PostQuitMessage(0);
#037        break;
#038  default:
#039        return DefWindowProc(hWnd, message, wParam, lParam);
#040  }
#041  return 0;
#042 }
```

第 21 行就是调用函数 DialogBox 来显示对话框窗口。

对话框的模板如下：

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

IDD_ABOUTBOX DIALOGEX 22, 17, 230, 75
STYLE DS_SETFONT | DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "关于"
FONT 9, "新宋体", 400, 0, 0x86
BEGIN
    ICON            IDI_TESTWIN,IDC_MYICON,14,9,21,21
```

```
    LTEXT           "TestWin Version
1.0",IDC_STATIC,49,10,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 2007",IDC_STATIC,49,20,119,8
    DEFPUSHBUTTON   "确定",IDOK,185,51,38,16,WS_GROUP
END
```

IDD_ABOUTBOX 是对话框模板的名称。
DIALOGEX 是对话框定义的关键字。
22, 17, 230, 75 是对话框的坐标和大小。
STYLE 是设置对话框的显示类型。
CAPTION 是定义标题名称。这里是"关于"。
FONT 是定义对话的字体。
ICON 是定义一个图标显示。
LTEXT 是定义显示静态文本。
DEFPUSHBUTTON 是定义一个按钮。

Windows API一日一练(18)EndDialog 函数

上一次介绍了怎么样显示对话框的函数，那么怎么样关闭对话框呢？这就需要使用到函数 EndDialog。这个函数只能在对话框的消息处理函数里使用，并且这个函数调用之后，没有立即就删除对话框的，而是设置了操作系统里的结束标志。当操作系统查检到有这个标志时，就去删除对话框的消息循环，同时也去释放对话框占用的资源。其实对话框的生命周期是这样的，先由函数 DialogBox 创建对话框，这样函数 DialogBox 完成创建对话框但还没有显示前会发出消息 WM_INITDIALOG，让对话框有机会初始化上面所有窗口或控件的显示，比如设置文本框的字符串等。最后当用户点出确定或者取消的按钮，就收到两个命令 IDOK 或 IDCANCEL，这时就可以调用函数 EndDialog 来结束对话框的生命。

函数 EndDialog 声明如下：

```
WINUSERAPI
BOOL
WINAPI
EndDialog(
    __in HWND hDlg,
    __in INT_PTR nResult);
```

hDlg 是对话框窗口的句柄。
nResult 是设置给函数 DialogBox 的返回值。

调用这个函数的例子如下：

```
#001 // 显示关于对话框。
#002 //
#003 // 蔡军生 2007/07/12
#004 //
```

```
#005 INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
#006 {
#007  UNREFERENCED_PARAMETER(lParam);
#008  switch (message)
#009  {
#010  case WM_INITDIALOG:
#011        return (INT_PTR)TRUE;
#012
#013  case WM_COMMAND:
#014        if (LOWORD(wParam) == IDOK || LOWORD(wParam) ==
IDCANCEL)
#015        {
#016            EndDialog(hDlg, LOWORD(wParam));
#017            return (INT_PTR)TRUE;
#018        }
#019        break;
#020  }
#021  return (INT_PTR)FALSE;
#022 }
```

第 16 行就是调用函数 EndDialog 来关闭对话框。

Windows API一日一练(19)DestroyWindow 函数

以前已经介绍过直接点击关闭按钮来关闭应用程序，但想删除一个窗口对象，需要用到什么函数的呢？比如创建了 30 个窗口，想把第 12 个窗口关闭删除掉，那就需要使用函数 DestroyWindow。当调用 DestroyWindow 函数后，操作系统就会进行一系列的删除动作，先发送 WM_DESTROY 消息，接着发送 WM_NCDESTROY 消息。如果这个窗口还有子窗口或者是其它窗口的所有者，就需要给所有子窗口发送删除消息。

函数 DestroyWindow 声明如下：

```
WINUSERAPI
BOOL
WINAPI
DestroyWindow(
    __in HWND hWnd);
```

hWnd 是要删除的窗口句柄。

调用这个函数的例子如下：

```
#001 //
#002 // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)
#003 //
#004 // 目的: 处理主窗口的消息.
```

```
#005 //
#006 // 蔡军生 2007/07/12    QQ:9073204
#007 //
#008 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
#009  {
#010  int wmId, wmEvent;
#011  PAINTSTRUCT ps;
#012  HDC hdc;
#013
#014  switch (message)
#015  {
#016  case WM_COMMAND:
#017        wmId    = LOWORD(wParam);
#018        wmEvent = HIWORD(wParam);
#019        // 菜单选项命令响应:
#020        switch (wmId)
#021        {
#022        case IDM_ABOUT:
#023            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX),
hWnd, About);
#024            break;
#025        case IDM_EXIT:
#026            DestroyWindow(hWnd);
#027            break;
#028        default:
#029            return DefWindowProc(hWnd, message, wParam, lParam);
#030        }
#031        break;
#032  case WM_PAINT:
#033        hdc = BeginPaint(hWnd, &ps);
#034        //
#035        EndPaint(hWnd, &ps);
#036        break;
#037  case WM_DESTROY:
#038        PostQuitMessage(0);
#039        break;
#040  default:
#041        return DefWindowProc(hWnd, message, wParam, lParam);
#042  }
#043  return 0;
#044 }
```

第 26 行是当收到菜单按钮退出命令的消息，就调用函数 DestroyWindow，然后它发出消息 WM_DESTROY 给第 37 行那里进行处理。

Windows API一日一练(20)LoadString、LoadIcon 和 LoadCursor 函数

在编写国际化的应用程序里，经常要使用不同语言的字符串。比如中文菜单里叫做“文件”，而在英语里叫做“File”。开发这种软件的功能是一样的，只是界面上显示的文字不一样而已。为了方便这种软件的开发，在 Windows 里经常使用的方法就是替换掉显示的字符串，比如指定在中文里就显示“文件”，在英语里就显示“File”，是通过函数 LoadString 从不同的资源里加载不同的字符串显示来实现的。其实所有可变的字符串，都可以使用函数 LoadString 从资源里加载字符串显示。

图像的信息是非常方便人们记忆的，像交通信号一样，就是使用各种各样的图标，每个人看了就会明白是什么东西，因此在程序里使用图标来标识程序。只要看到这个图标，就知道是这个软件做什么用的。比如在程序的左上角显示的图标，就需要调用函数 LoadIcon 从资源里加载到内存里，然后再显示出来。

光标更是最常用的图标了，时时刻刻都可以看到它。当你在编辑文件时，光标显示为一个“I”字形图标，当你在桌面操作文件时，显示为一个箭头。这样变化的光标，就是因为设置了不同的图标。光标的使用，就是根据不同的区域来作不同的指示。更换光标的函数是 LoadCursor。

函数 LoadString 声明如下：

```
WINUSERAPI
int
WINAPI
LoadStringA(
    __in_opt HINSTANCE hInstance,
    __in UINT uID,
    __out_ecount(cchBufferMax) LPSTR lpBuffer,
    __in int cchBufferMax);
WINUSERAPI
int
WINAPI
LoadStringW(
    __in_opt HINSTANCE hInstance,
    __in UINT uID,
    __out_ecount(cchBufferMax) LPWSTR lpBuffer,
    __in int cchBufferMax);
#ifdef UNICODE
#define LoadString  LoadStringW
#else
#define LoadString  LoadStringA
#endif // !UNICODE
```

hInstance 是应用程序实例句柄。
uID 是资源中的字符串编号。
lpBuffer 是接收从资源里拷贝字符串出来的缓冲区。
cchBufferMax 是指明缓冲的大小。

函数 LoadIcon 声明如下：

```
WINUSERAPI
HICON
WINAPI
LoadIconA(
    __in_opt HINSTANCE hInstance,
    __in LPCSTR lpIconName);
WINUSERAPI
HICON
WINAPI
LoadIconW(
    __in_opt HINSTANCE hInstance,
    __in LPCWSTR lpIconName);
#ifdef UNICODE
#define LoadIcon  LoadIconW
#else
#define LoadIcon  LoadIconA
#endif // !UNICODE
```

hInstance 是应用程序实例句柄。
lpIconName 是指向图标的名称。

函数 LoadCursor 声明如下：

```
WINUSERAPI
HCURSOR
WINAPI
LoadCursorA(
    __in_opt HINSTANCE hInstance,
    __in LPCSTR lpCursorName);
WINUSERAPI
HCURSOR
WINAPI
LoadCursorW(
    __in_opt HINSTANCE hInstance,
    __in LPCWSTR lpCursorName);
#ifdef UNICODE
#define LoadCursor  LoadCursorW
#else
#define LoadCursor  LoadCursorA
#endif // !UNICODE
```

hInstance 是应用程序实例句柄。
lpCursorName 是光标的名称。

调用这三个函数的例子如下：

```
#001  //
#002  //  函数: MyRegisterClass()
#003  //
#004  //  目的: 注册一个窗口类型.
#005  //
#006  //  蔡军生  2007/07/12
#007  //
#008  ATOM MyRegisterClass(HINSTANCE hInstance)
#009  {
#010   WNDCLASSEX wcex;
#011
#012   wcex.cbSize = sizeof(WNDCLASSEX);
#013
#014   wcex.style          = CS_HREDRAW | CS_VREDRAW;
#015   wcex.lpfnWndProc  = WndProc;
#016   wcex.cbClsExtra        = 0;
#017   wcex.cbWndExtra       = 0;
#018   wcex.hInstance     = hInstance;
#019       wcex.hIcon             = LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_TESTWIN));
#020       wcex.hCursor      = LoadCursor(NULL, IDC_ARROW);
#021   wcex.hbrBackground    = (HBRUSH)(COLOR_WINDOW+1);
#022   wcex.lpszMenuName     = MAKEINTRESOURCE(IDC_TESTWIN);
#023   wcex.lpszClassName     = szWindowClass;
#024   wcex.hIconSm       = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));
#025
#026   return RegisterClassEx(&wcex);
#027  }
```

第 19 行就是加载显示的图标，使用宏 MAKEINTRESOURCE 来转换为合适的类型，它的定义如下：

```
#define IS_INTRESOURCE(_r) ((((ULONG_PTR)(_r)) >> 16) == 0)
#define MAKEINTRESOURCEA(i) ((LPSTR)((ULONG_PTR)((WORD)(i))))
#define MAKEINTRESOURCEW(i) ((LPWSTR)((ULONG_PTR)((WORD)(i))))
#ifdef UNICODE
#define MAKEINTRESOURCE   MAKEINTRESOURCEW
#else
#define MAKEINTRESOURCE   MAKEINTRESOURCEA
#endif // !UNICODE
```

第 20 行是加载箭头的光标来显示。

```
// 加载全局字符串。
LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadString(hInstance, IDC_TESTWIN, szWindowClass, MAX_LOADSTRING);
```

上面两行就是调用函数 LoadString 从资源里获取显示的字符串。

Windows API一日一练(21)SetWindowLongPtr 和 GetWindowLongPtr 函数

在软件开发里，大家一直对着这个问题是执着的，那是"复用"。总想自己写出来的代码，可以适应性很强，到那里都可以落地生根。因此，面向对象的语言就层出不穷，每个都坚称可以最大地复用代码。在面向对象里，C++是非常强大的。下面就来用 C++把上面介绍的程序封装起来，这样可以复用，或者说条理更加清晰。

```
#001
#002 int APIENTRY _tWinMain(HINSTANCE hInstance,
#003                        HINSTANCE hPrevInstance,
#004                        LPTSTR    lpCmdLine,
#005                        int       nCmdShow)
#006 {
#007  UNREFERENCED_PARAMETER(hPrevInstance);
#008  UNREFERENCED_PARAMETER(lpCmdLine);
#009
#010  CCaiWin caiWin;
#011
#012  caiWin.MyRegisterClass(hInstance);
#013  if (!caiWin.InitInstance(hInstance,nCmdShow))
#014  {
#015        return 0;
#016  }
#017
#018  return caiWin.RunMessage();
#019 }
```

这段代码跟前面介绍的调用，就是不一样了。
第 10 行创建了一个 CCaiWin 的对象 caiWin。
第 12 行调用对象 CCaiWin 里的注册函数 MyRegisterClass 来注册一个窗口。
第 13 行就是初始化一个窗口的创建。
第 18 行就是调用对象 caiWin 的消息处理函数 RunMessage。

这样就制定了一个基本应用的框架，可以任意修改对象里的内容，都不会影响这个函数里的调用，也就是说，只要不改那几个函数就可以永远不用修改 WinMain 函数里的内容了。

接着下来，再来看看类 CCaiWin 是怎么样编写的。它的类定义如下：

```
#001 #include <string>
#002
#003 //
#004 //封装一个窗口类。
#005 //蔡军生 2007/07/27
#006 //
#007 class CCaiWin
#008 {
#009 public:
#010  CCaiWin(void);
#011  virtual ~CCaiWin(void);
#012
#013  ATOM MyRegisterClass(HINSTANCE hInstance);
#014  bool InitInstance(HINSTANCE hInstance, int nCmdShow);
#015  int RunMessage(void);
#016  HINSTANCE GetAppInstance(void)
#017   {
#018        return m_hInstance;
#019  }
#020 protected:
#021  static LRESULT CALLBACK WndProc(HWND hWnd,
#022        UINT message, WPARAM wParam, LPARAM lParam);
#023  static INT_PTR CALLBACK About(HWND hDlg,
#024        UINT message, WPARAM wParam, LPARAM lParam);
#025 protected:
#026  HINSTANCE m_hInstance;
#027  HWND m_hWnd;
#028
#029  std::wstring m_strWindowClass;
#030  std::wstring m_strTitle;
#031 };
```

第 7 行定义类 CCaiWin。

第 13 行是声明 MyRegisterClass 注册函数。

第 14 行是声明 InitInstance 初始化窗口函数。

第 15 行是声明 RunMessage 消息处理函数。

第 16 行是定义 GetAppInstance 函数获取应用程序句柄。

第 21 行是声明窗口的消息处理函数。它是静态成员函数，所以它有全局的地址，因此它是没有 this 指针的，不能直接地访问这个类里的成员变量。需要使用其它方法给它传递。

第 23 行是关于对话框的消息处理函数。

第 26 行是保存应用程序句柄。

第 27 行是保存主窗口的句柄。

第 29 行是保存注册窗口名称。

第 30 行是保存窗口显示的标题。

下面再来仔细地查看类的实现文件。

```
#001 //
#002 //   函数: InitInstance(HINSTANCE, int)
#003 //
#004 //   目的: 保存程序实例句柄，并创建窗口显示。
#005 //
#006 //   蔡军生 2007/07/27 QQ:9073204
#007 //
#008 bool CCaiWin::InitInstance(HINSTANCE hInstance, int nCmdShow)
#009 {
#010  //
#011    m_hInstance = hInstance;
#012
#013    m_hWnd = CreateWindow(m_strWindowClass.c_str(),
m_strTitle.c_str(),
#014         WS_OVERLAPPEDWINDOW,
#015         CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance,
NULL);
#016
#017  if (!m_hWnd)
#018  {
#019         return false;
#020  }
#021
#022  //保存类的指针到窗口 GWL_USERDATA 字段，
#023  //以便消息函数里可以获取类指针。
#024  SetWindowLongPtr(m_hWnd,
GWL_USERDATA,(LONG)(LONG_PTR)this);
#025
#026  ShowWindow(m_hWnd, nCmdShow);
#027  UpdateWindow(m_hWnd);
#028
#029  return true;
#030 }
```

这里创建窗口，跟以前创建窗口，只有一个地方不一样，那就是在第 24 行里调用 SetWindowLongPtr 函数保存对象指针到窗口用户自定义数据里，这样做就是让后面的静态成员函数 WndProc 可以访问类成员。如下：

```
#001 //
#002 // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)
#003 //
#004 // 目的: 处理主窗口的消息.
#005 //
#006 // 蔡军生 2007/07/27    QQ:9073204
```

```
#007 //
#008 LRESULT CALLBACK CCaiWin::WndProc(HWND hWnd, UINT message,
#009
#010 WPARAM wParam, LPARAM lParam)
#011 {
#012  //获取窗口对应的类指针。
#013  LONG_PTR plptrWin =
GetWindowLongPtr(hWnd,GWLP_USERDATA);
#014  if (plptrWin == NULL)
#015  {
#016      return DefWindowProc(hWnd, message, wParam, lParam);
#017  }
#018
#019  //
#020  CCaiWin* pWin = reinterpret_cast<CCaiWin*>(plptrWin);
#021
#022  int wmId, wmEvent;
#023  PAINTSTRUCT ps;
#024  HDC hdc;
#025
#026  switch (message)
#027  {
#028  case WM_COMMAND:
#029      wmId    = LOWORD(wParam);
#030      wmEvent = HIWORD(wParam);
#031      // 菜单选项命令响应:
#032      switch (wmId)
#033      {
#034      case IDM_ABOUT:
#035          DialogBox(pWin->GetAppInstance(),
MAKEINTRESOURCE(IDD_ABOUTBOX),
#036              hWnd, CCaiWin::About);
#037          break;
#038      case IDM_EXIT:
#039          DestroyWindow(hWnd);
#040          break;
#041      default:
#042          return DefWindowProc(hWnd, message, wParam, lParam);
#043      }
#044      break;
#045  case WM_PAINT:
#046      {
#047          hdc = BeginPaint(hWnd, &ps);
#048          //
```

```
#049            std::wstring strShow(_T("C++窗口类的实现,2007-07-27"));
#050            TextOut(hdc,10,10,strShow.c_str(),(int)strShow.length());
#051
#052            //
#053            EndPaint(hWnd, &ps);
#054        }
#055        break;
#056    case WM_DESTROY:
#057        //设置窗口类指针为空。
#058        SetWindowLongPtr(hWnd, GWL_USERDATA,NULL);
#059
#060        PostQuitMessage(0);
#061        break;
#062    default:
#063        return DefWindowProc(hWnd, message, wParam, lParam);
#064    }
#065    return 0;
#066 }
```


上面第 13 行就是获取窗口里保存的类对象指针，然后再作类型转换为窗口 CCaiWin 的指针，这样就可以使用类的成员了，比如在第 35 行里的调用 pWin->GetAppInstance()。

其实在封装静态成员函数这里，就有三种方法传递类指针，上面介绍这种是最简单的。一种是 MFC 里使用的，它是采用一个窗口和类指针映射数组来实现的。一种是 WTL 里使用叫做 THUNK 代码实现窗口与静态函数的关联。像上面这种方法，在游戏 Second Life 的源程序就使用它，如果是一般的应用程序，而不是大框架，使用这种简单的方法，就是最好的。

函数 GetWindowLongPtr 和 SetWindowLongPtr 声明如下：

```
WINUSERAPI
LONG
WINAPI
GetWindowLongA(
    __in HWND hWnd,
    __in int nIndex);
WINUSERAPI
LONG
WINAPI
GetWindowLongW(
    __in HWND hWnd,
    __in int nIndex);
#ifdef UNICODE
#define GetWindowLong GetWindowLongW
#else
#define GetWindowLong GetWindowLongA
#endif // !UNICODE
```

```
WINUSERAPI
LONG
WINAPI
SetWindowLongA(
    __in HWND hWnd,
    __in int nIndex,
    __in LONG dwNewLong);
WINUSERAPI
LONG
WINAPI
SetWindowLongW(
    __in HWND hWnd,
    __in int nIndex,
    __in LONG dwNewLong);

#ifdef UNICODE
#define SetWindowLong SetWindowLongW
#else
#define SetWindowLong SetWindowLongA
#endif // !UNICODE
#define GetWindowLongPtrA    GetWindowLongA
#define GetWindowLongPtrW    GetWindowLongW
#ifdef UNICODE
#define GetWindowLongPtr GetWindowLongPtrW
#else
#define GetWindowLongPtr GetWindowLongPtrA
#endif // !UNICODE

#define SetWindowLongPtrA    SetWindowLongA
#define SetWindowLongPtrW    SetWindowLongW
#ifdef UNICODE
#define SetWindowLongPtr SetWindowLongPtrW
#else
#define SetWindowLongPtr SetWindowLongPtrA
#endif // !UNICODE
```

hWnd 是窗口句柄。
nIndex 是访问窗口对象数据的索引值。比如像 GWLP_USERDATA、GWLP_WNDPROC。
dwNewLong 是设置的新值。

Windows API一日一练(23)SetTextColor 函数

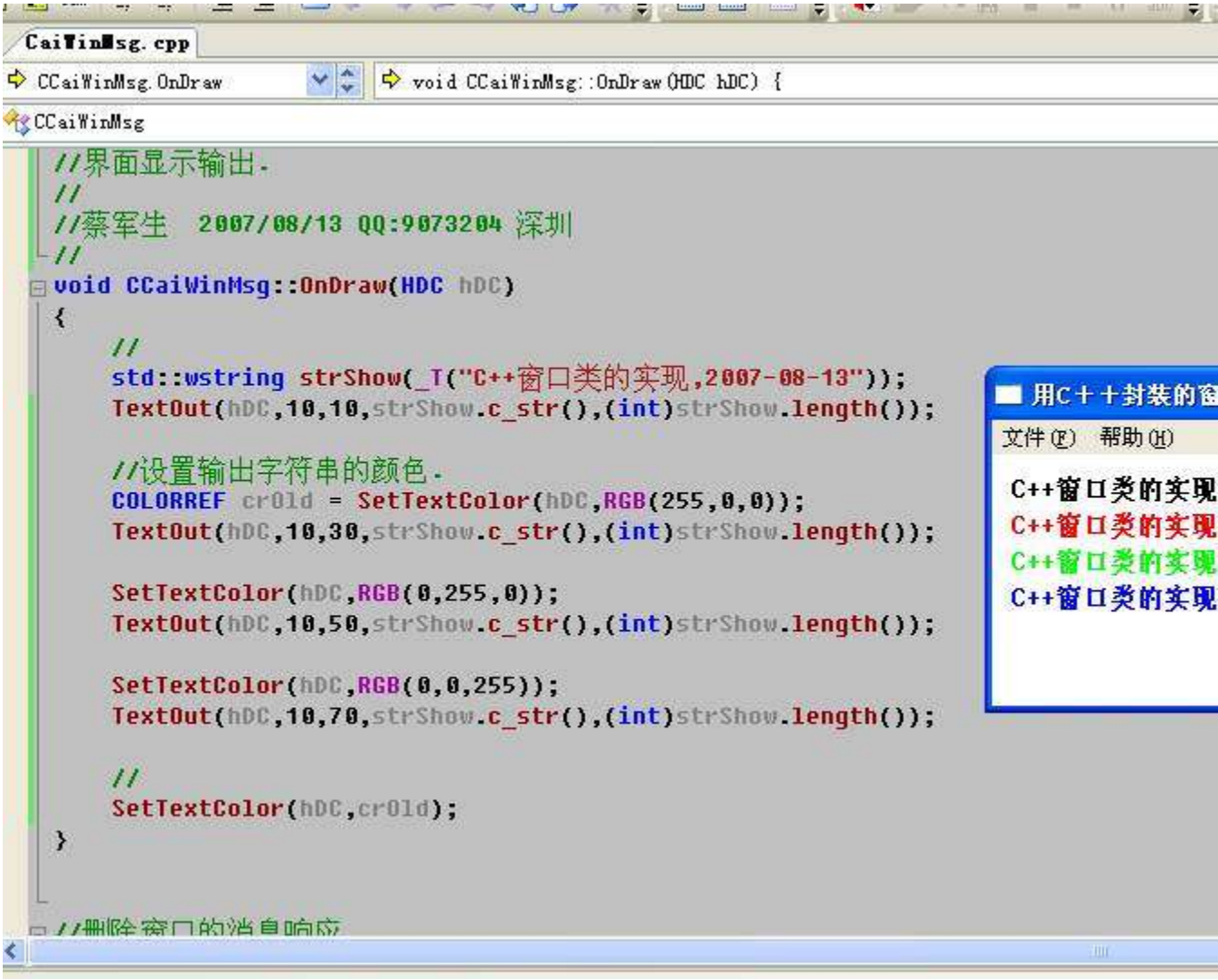
世界是多姿多彩的，色彩是不可以缺少的。在软件开发里，不同的字符颜色往往用来区分不同的数据类型，比如严重的警告，就是使用红色，当然这也是跟交通里红绿灯是一样的道理。最根本的原因，还是人类对自然的选择。毕竟人眼对不同的颜色作出了不同的选择。由于红色光波穿透性比较好，所以就形成以红色来警告的信号。说来也怪，人类的身体不管是什么颜色，流着的血一定是红色的。在软件开发里，要把输出字符的颜色符合现在人位的习惯，不要背离它，否则就不是"以人为本"的科学理念，如果开发出不是"以人为本"的软件是没有客户使用的。在 Windows 里，微软是选择以蓝色为基调的配色方案，深蓝色的 XP 界面比 WIN2000 界面，让人感觉到耳目一新的感觉，深深地把年轻人吸引住了。看到国内杀毒软件，很多是选择绿色为基调，其实就是绿色也就是代表了安全的选择。可见，在软件领域里颜色是非常重要的，面对不同的客户要选择不同的颜色。比如 SAP 软件，选择是黄色为基调，比较有特色。

函数 SetTextColor 声明如下：
WINGDIAPI COLORREF WINAPI SetTextColor(__in HDC hdc, __in COLORREF color);
hdc 是当前设备的句柄。
color 是设置当前设备字符输出颜色。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/08/13 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //
#009  std::wstring strShow(_T("C++窗口类的实现,2007-08-13"));
#010  TextOut(hDC,10,10,strShow.c_str(),(int)strShow.length());
#011
#012  //设置输出字符串的颜色.
#013  COLORREF crOld = SetTextColor(hDC,RGB(255,0,0));
#014  TextOut(hDC,10,30,strShow.c_str(),(int)strShow.length());
#015
#016  SetTextColor(hDC,RGB(0,255,0));
#017  TextOut(hDC,10,50,strShow.c_str(),(int)strShow.length());
#018
#019  SetTextColor(hDC,RGB(0,0,255));
#020  TextOut(hDC,10,70,strShow.c_str(),(int)strShow.length());
#021
#022  //
#023  SetTextColor(hDC,crOld);
#024 }
```

第 13 行设置字符的颜色为红色。
第 16 行设置字符的颜色为绿色。
第 19 行设置字符的颜色为蓝色。
第 23 行恢复原来的颜色，这个一定要记得做，否则后面显示会出错。



Windows API一日一练(24)DrawText 函数

DrawText 函数与前面介绍的 TextOut 函数都是文本输出函数，但它们是有区别的。DrawText 函数是格式化输出函数，而 TextOut 函数不具备这样的功能。因而 DrawText 函数比 TextOut 函数功能强大，可以让文本输出时左对齐，或者右对齐，或者中间对齐，还可以让文本适应输出矩形内，如果超出时可以截断，或者显示为省略号的方式。DrawText 函数在表格方式显示时肯定要使用到的函数。

函数 DrawText 声明如下：
WINUSERAPI

```
int
WINAPI
DrawTextA(
    __in HDC hdc,
    __inout_ecount(cchText) LPCSTR lpchText,
    __in int cchText,
    __inout LPRECT lprc,
    __in UINT format);
WINUSERAPI
int
WINAPI
DrawTextW(
    __in HDC hdc,
    __inout_ecount(cchText) LPCWSTR lpchText,
    __in int cchText,
    __inout LPRECT lprc,
    __in UINT format);
#ifdef UNICODE
#define DrawText DrawTextW
#else
#define DrawText DrawTextA
#endif // !UNICODE
```

hdc 是当前设备的句柄。
lpchText 是输出文本的缓冲区首地址。
cchText 是输出文本的字符个数。
lprc 是输出的显示区域。
format 是用什么格式输出。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/08/27 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //
#009  std::wstring strShow(_T("C++窗口类的实现,2007-08-27"));
#010  TextOut(hDC,10,10,strShow.c_str(),(int)strShow.length());
#011
#012  //设置输出字符串的颜色.
#013  COLORREF crOld = SetTextColor(hDC,RGB(255,0,0));
#014
```

```
#015  RECT rcText;
#016
#017  //显示不全.
#018  rcText.left = 10;
#019  rcText.top = 30;
#020  rcText.right = 100;
#021  rcText.bottom = 50;
#022
#023  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#024        DT_LEFT|DT_SINGLELINE|DT_END_ELLIPSIS);
#025
#026  //完全显示,左对齐.
#027   rcText.left = 10;
#028  rcText.top = 50;
#029  rcText.right = 300;
#030  rcText.bottom = 80;
#031
#032  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#033        DT_LEFT|DT_SINGLELINE|DT_END_ELLIPSIS);
#034
#035
#036  SetTextColor(hDC,RGB(0,0,255));
#037  //完全显示,右对齐.
#038  rcText.left = 10;
#039  rcText.top = 80;
#040  rcText.right = 300;
#041  rcText.bottom = 110;
#042
#043  strShow = _T("A&bcd");
#044  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#045        DT_RIGHT|DT_SINGLELINE|DT_END_ELLIPSIS);
#046
#047
#048  //
#049  SetTextColor(hDC,crOld);
#050 }
```


Windows API一日一练(25)CreateSolidBrush 函数

当你看到 Windows 显示的按钮时，背景颜色是灰色的。当你看到缺省的窗口背景时，它是白色的。当你的老板需要你创建一个黑色背景的按钮时，你会怎么样做呢？其实在 Windows 里先用 API 函数 CreateSolidBrush 创建画刷，然后调用 FillRect 函数来填充

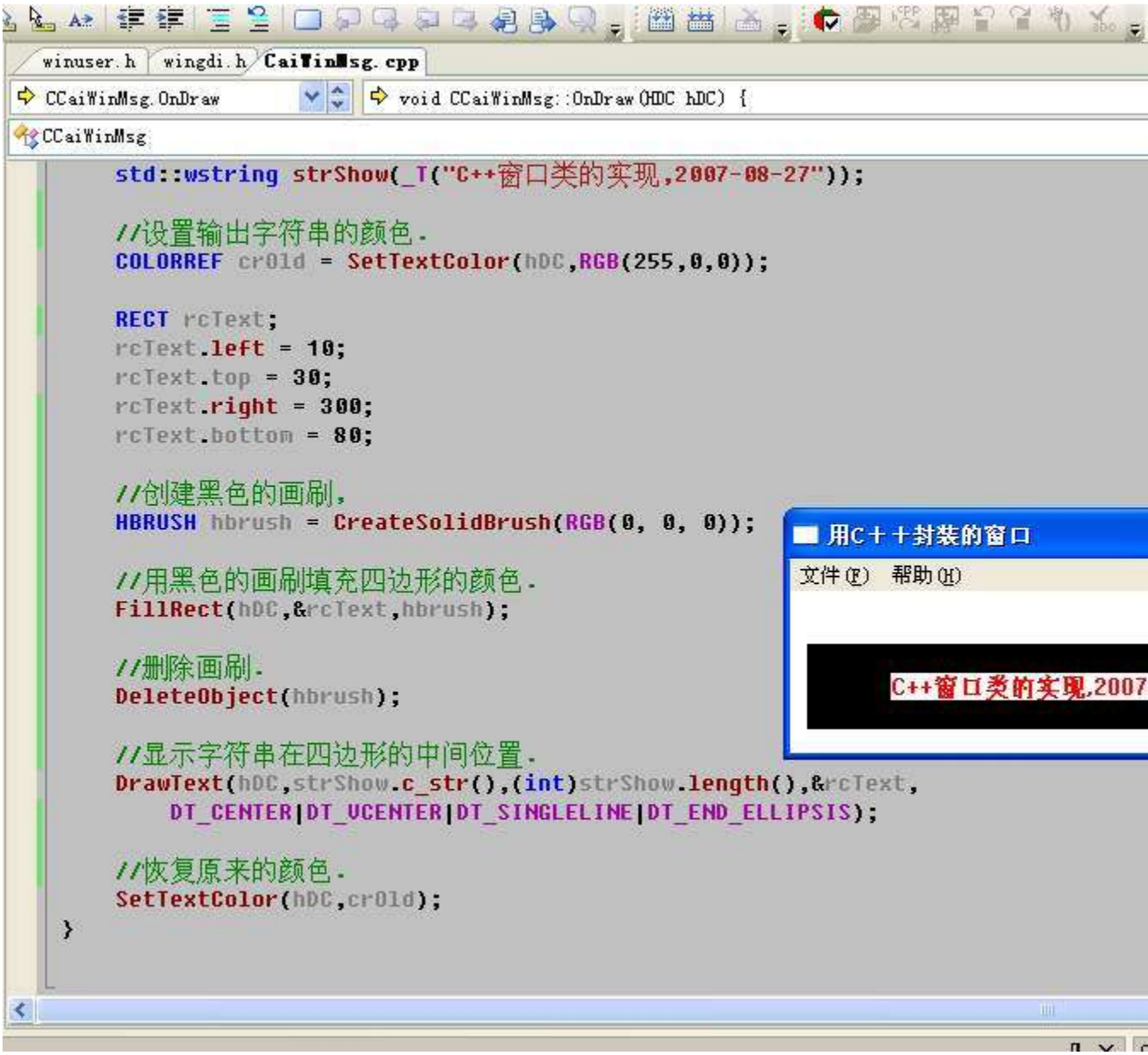
背景。这样来，不管你需要什么样的背景，都随心所欲了吧。现在先来搞懂 CreateSolidBrush 函数，下次再来练习 FillRect。

函数 CreateSolidBrush 声明如下：
WINGDIAPI HBRUSH WINAPI CreateSolidBrush(__in COLORREF color);
color 是画刷颜色。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/08/29 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //
#009  std::wstring strShow(_T("C++窗口类的实现,2007-08-27"));
#010
#011  //设置输出字符串的颜色.
#012  COLORREF crOld = SetTextColor(hDC,RGB(255,0,0));
#013
#014  RECT rcText;
#015  rcText.left = 10;
#016  rcText.top = 30;
#017  rcText.right = 300;
#018  rcText.bottom = 80;
#019
#020  //创建黑色的画刷,
#021  HBRUSH hbrush = CreateSolidBrush(RGB(0, 0, 0));
#022
#023  //用黑色的画刷填充四边形的颜色.
#024  FillRect(hDC,&rcText,hbrush);
#025
#026  //删除画刷.
#027  DeleteObject(hbrush);
#028
#029  //显示字符串在四边形的中间位置.
#030  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#031        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);
#032
#033  //恢复原来的颜色.
#034  SetTextColor(hDC,crOld);
#035 }
```

第 21 行是创建黑色的画刷。它的效果图如下：



Windows API一日一练(27)SetBkMode 函数

上面已经介绍输出红色的字符串时，发现背景的黑色也变成白色了，这样的输出是破坏背景的。那需要使用什么方法来保持背景不变，而又能输出红色的字符串呢？比如按钮的文字颜色是黑色的，而背景是灰色的。这就需要使用 SetBkMode 函数来设置 DrawText 函数的输出方式，显示设备共有两种输出方式：OPAQUE 和 TRANSPARENT。OPAQUE 的方式是用当前背景的画刷的颜色输出显示文字的背景，而 TRANSPARENT 是使用透明的输出，也就是文字的背景是不改变的。

函数 SetBkMode 声明如下：

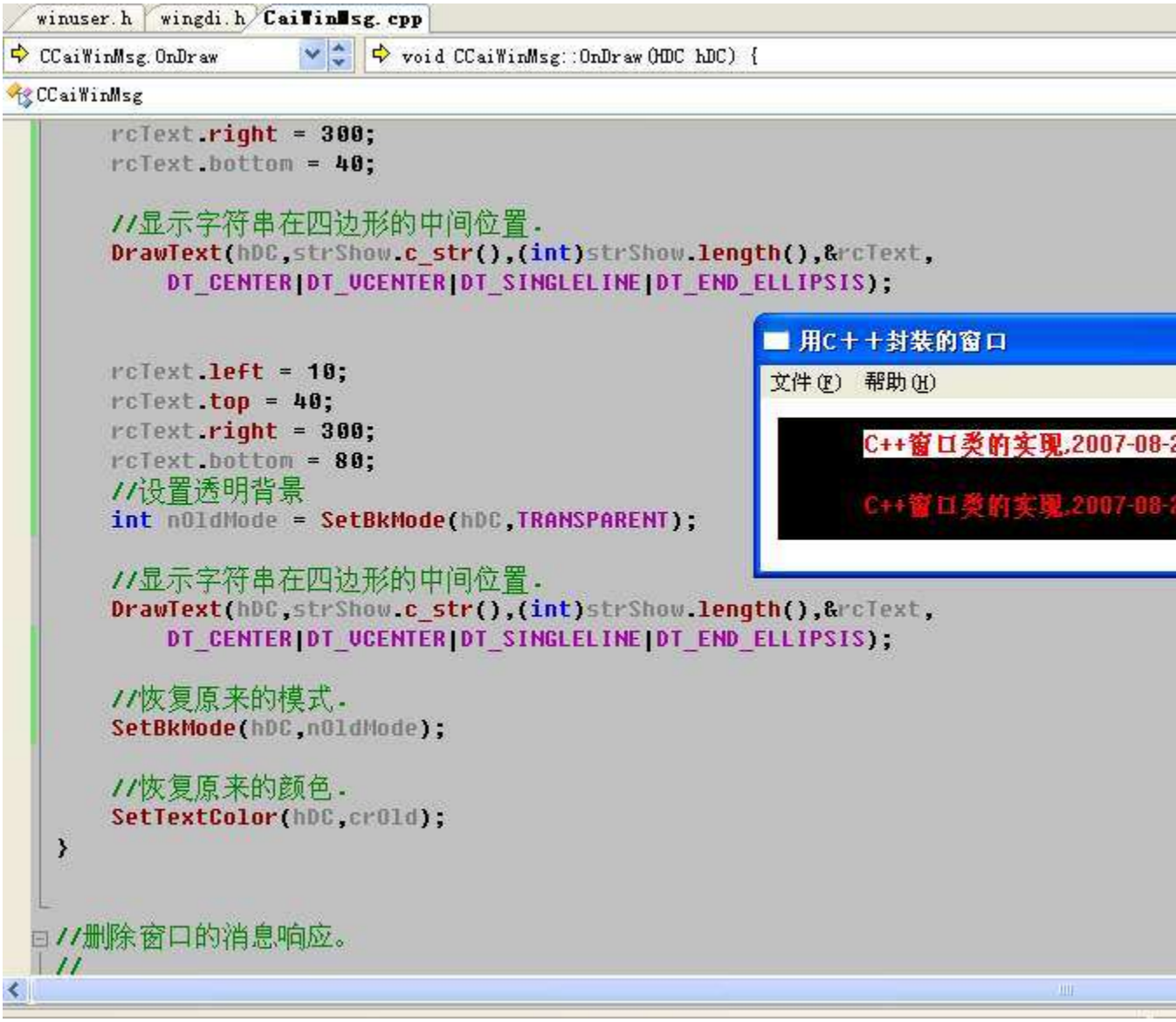
WINGDIAPI int　WINAPI SetBkMode(__in HDC hdc, __in int mode);
hDC 是当前设备的句柄。
mode 是要设置的模式。
函数返回值是前一次设置的模式。

调用这个函数的例子如下：


```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/01 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //
#009  std::wstring strShow(_T("C++窗口类的实现,2007-08-27"));
#010
#011  //设置输出字符串的颜色.
#012  COLORREF crOld = SetTextColor(hDC,RGB(255,0,0));
#013
#014  RECT rcText;
#015  rcText.left = 10;
#016  rcText.top = 10;
#017  rcText.right = 300;
#018  rcText.bottom = 80;
#019
#020  //创建黑色的画刷,
#021  HBRUSH hbrush = CreateSolidBrush(RGB(0, 0, 0));
#022
#023  //用黑色的画刷填充四边形的颜色.
#024  FillRect(hDC,&rcText,hbrush);
#025
#026  //删除画刷.
#027  DeleteObject(hbrush);
#028
#029
#030  rcText.left = 10;
#031  rcText.top = 10;
#032  rcText.right = 300;
#033  rcText.bottom = 40;
#034
#035  //显示字符串在四边形的中间位置.
#036  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#037        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);
#038
```

```
#039
#040  rcText.left = 10;
#041  rcText.top = 40;
#042  rcText.right = 300;
#043  rcText.bottom = 80;
#044  //设置透明背景
#045  int nOldMode = SetBkMode(hDC,TRANSPARENT);
#046
#047  //显示字符串在四边形的中间位置.
#048  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#049        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);
#050
#051  //恢复原来的模式.
#052  SetBkMode(hDC,nOldMode);
#053
#054  //恢复原来的颜色.
#055  SetTextColor(hDC,crOld);
#056 }
```


本程序运行的效果图如下：



第一次显示是使用 OPAQUE 的方式显示。
第二次显示是使用 TRANSPARENT 的方式显示。

Windows API一日一练(28)CreateFont 函数

文字的出现历史可以追溯到甲骨文的使用，直到今天使用的宋体文字。在软件开发里，经常遇到是跨国语言的使用，由于世界在变平，全球在变小，交通运输非常发达，由我所在的深圳，向东坐飞机 12 个小时就可以到达伦敦，向西坐飞机 12 小时就可以到达美国，可算得上朝发夕至。比如像炒外汇的人，一天可以不用睡觉都在工作着，比如早上炒东京的汇市，下午就可以炒伦敦的，晚上就可以炒纽约的了。软件的开发，也在全球化，比如昨晚在美国开发，早上就可以变成中国开发同样的软件了，一天 24 小时开发，这样加速软件的开发。全球的市场已经变得同步化了，开发的软件可以适应任何有人类的地方。经常开发的软件，就需要有中英双语化。这样就需要使用到不同的字体，才能适应国际化的需要，下面就来学习怎么样创建字体，并且使用它。

函数 CreateFont 声明如下：

```
WINGDIAPI HFONT    WINAPI CreateFontA( __in int cHeight, __in int cWidth,
__in int cEscapement, __in int cOrientation, __in int cWeight, __in DWORD
bItalic,
                    __in DWORD bUnderline, __in DWORD bStrikeOut, __in
DWORD iCharSet, __in DWORD iOutPrecision, __in DWORD iClipPrecision,
                    __in DWORD iQuality, __in DWORD iPitchAndFamily,
__in_opt LPCSTR pszFaceName);
WINGDIAPI HFONT    WINAPI CreateFontW( __in int cHeight, __in int cWidth,
__in int cEscapement, __in int cOrientation, __in int cWeight, __in DWORD
bItalic,
                    __in DWORD bUnderline, __in DWORD bStrikeOut, __in
DWORD iCharSet, __in DWORD iOutPrecision, __in DWORD iClipPrecision,
                    __in DWORD iQuality, __in DWORD iPitchAndFamily,
__in_opt LPCWSTR pszFaceName);
#ifdef UNICODE
#define CreateFont CreateFontW
#else
#define CreateFont CreateFontA
#endif // !UNICODE
```

cHeight 是字体的高度。
cWidth 是字体的宽度。
cEscapement 是字体的倾斜角。
cOrientation 是字体的倾斜角。
cWeight 是字体的粗细。
bItalic 是字体是否斜体。
bUnderline 是字体是否有下划线。
bStrikeOut 是字体是否有删除线。
iCharSet 是字体使用的字符集。
iOutPrecision 是指定如何选择合适的字体。
iClipPrecision 是用来确定裁剪的精度。
iQuality 是怎么样跟选择的字体相符合。
iPitchAndFamily 是间距标志和属性标志。
pszFaceName 是字体的名称。

调用这个函数的例子如下：

```
#001 //创建字体.
#002 //
#003 //蔡军生 2007/09/03 QQ:9073204 深圳
#004 //
#005 HFONT CCaiWinMsg::GetFont(void)
```

```
#006 {
#007  LOGFONT lf; //字符的结构
#008
#009  //获取当前系统的字体.
#010  GetObject(GetStockObject(SYSTEM_FONT), sizeof(LOGFONT),
#011        &lf);
#012
#013  //设置字体的属性.
#014  lf.lfWeight = FW_BOLD;
#015  lf.lfItalic = true;
#016  lf.lfHeight = 26;
#017
#018  //设置为宋体.
#019  wsprintf(lf.lfFaceName,_T("%s"),_T("宋体"));
#020
#021  //创建字体并返回
#022  return CreateFont(lf.lfHeight, lf.lfWidth,
#023        lf.lfEscapement, lf.lfOrientation, lf.lfWeight,
#024        lf.lfItalic, lf.lfUnderline, lf.lfStrikeOut, lf.lfCharSet,
#025        lf.lfOutPrecision, lf.lfClipPrecision, lf.lfQuality,
#026        lf.lfPitchAndFamily, lf.lfFaceName);
#027
#028 }
```


Windows API一日一练(29)SelectObject 和 DeleteObject 函数

Windows 显示设备的属性，共有下面几种：位图、画刷、字体、画笔、区域。如果要设置它们到当前设备里，就需要使用 SelectObject 函数，比如上面介绍的字体设置，就会用到这个函数。当你创建一个位图时，这时 Windows 就会在内存里分配一块内存空间，用来保存位图的数据。当你创建字体时，也会分配一块内存空间保存字体。如果程序只是分配，而不去删除，就会造成内存使用越来越多，最后导到 Windows 这幢大楼倒下来。如果你忘记删除它，就造成了内存泄漏。因此，当你创建显示设备资源时，一定要记得删除它们啊，否则运行你的程序越长，就导致系统不稳定。记得使用 DeleteObject 函数去删除它们，把占用的内存释放回去给系统。

函数 SelectObject 和 DeleteObject 声明如下：

```
WINGDIAPI HGDIOBJ WINAPI SelectObject(__in HDC hdc, __in HGDIOBJ h);
WINGDIAPI BOOL WINAPI DeleteObject( __in HGDIOBJ ho);
```

hDC 是当前设备的句柄。
h，ho 是设备对象，其实它就是内存的地址。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/01 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //
#009  std::wstring strShow(_T("C++窗口类的实现,2007-09-04"));
#010
#011  //设置输出字符串的颜色.
#012  COLORREF crOld = SetTextColor(hDC,RGB(255,0,0));
#013
#014  RECT rcText;
#015  rcText.left = 10;
#016  rcText.top = 10;
#017  rcText.right = 300;
#018  rcText.bottom = 80;
#019
#020  //创建黑色的画刷,
#021  HBRUSH hbrush = CreateSolidBrush(RGB(0, 0, 0));
#022
#023  //用黑色的画刷填充四边形的颜色.
#024  FillRect(hDC,&rcText,hbrush);
#025
#026  //删除画刷.
#027  DeleteObject(hbrush);
#028
#029
#030  rcText.left = 10;
#031  rcText.top = 10;
#032  rcText.right = 300;
#033  rcText.bottom = 40;
#034
#035  //显示字符串在四边形的中间位置.
#036  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#037        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);
#038
#039
#040  rcText.left = 10;
#041  rcText.top = 40;
#042  rcText.right = 300;
#043  rcText.bottom = 80;
#044  //设置透明背景
```

```
#045  int nOldMode = SetBkMode(hDC,TRANSPARENT);
#046
#047  //设置新字体.
#048  HGDIOBJ hOldFont = SelectObject(hDC,GetFont());
#049
#050  //显示字符串在四边形的中间位置.
#051  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#052        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);
#053
#054  //恢复原来的字体.
#055  HGDIOBJ hFont = SelectObject(hDC,hOldFont);
#056  DeleteObject(hFont);
#057
#058  //恢复原来的模式.
#059  SetBkMode(hDC,nOldMode);
#060
#061  //恢复原来的颜色.
#062  SetTextColor(hDC,crOld);
#063 }
```


Windows API一日一练(30)GetTextMetrics 函数

在做报表里，经常要把输出的内容进行错落有致的排列，让用户看起来更加舒服。比如使用标题的字体输出后，再使用其小号的字体进行输出。这样就需要知道每种字体的高度，才让两行文字输出不重叠在一起，就需要计算每种字体的高度。这时就需使用 GetTextMetrics 函数来获取字体的高度。

函数 GetTextMetrics 声明如下：

```
#ifndef NOTEXTMETRIC

WINGDIAPI BOOL WINAPI GetTextMetricsA( __in HDC hdc, __out
LPTEXTMETRICA lptm);
WINGDIAPI BOOL WINAPI GetTextMetricsW( __in HDC hdc, __out
LPTEXTMETRICW lptm);
#ifdef UNICODE
#define GetTextMetrics GetTextMetricsW
#else
#define GetTextMetrics GetTextMetricsA
#endif // !UNICODE
```

hdc 是当前设备的句柄。
lptm 是获取当前字体属性的保存结构。它的结构定义如下：

```
typedef struct tagTEXTMETRICW
    {
    LONG tmHeight;
    LONG tmAscent;
    LONG tmDescent;
    LONG tmInternalLeading;
    LONG tmExternalLeading;
    LONG tmAveCharWidth;
    LONG tmMaxCharWidth;
    LONG tmWeight;
    LONG tmOverhang;
    LONG tmDigitizedAspectX;
    LONG tmDigitizedAspectY;
    WCHAR tmFirstChar;
    WCHAR tmLastChar;
    WCHAR tmDefaultChar;
    WCHAR tmBreakChar;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    }      TEXTMETRICW;
```


调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/06 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //
#009  std::wstring strShow(_T("C++窗口类的实现,2007-09-04"));
#010
#011  //设置输出字符串的颜色.
#012  COLORREF crOld = SetTextColor(hDC,RGB(255,0,0));
#013
#014  RECT rcText;
#015
#016  //设置新字体.
#017  HGDIOBJ hOldFont = SelectObject(hDC,GetFont());
#018
#019  //获取当前字体的高度.
```

```
#020  TEXTMETRIC tmFont;
#021  if (GetTextMetrics(hDC,&tmFont))
#022  {
#023        rcText.left = 10;
#024        rcText.top = 40;
#025        rcText.right = 300;
#026        rcText.bottom = rcText.top + tmFont.tmHeight;
#027  }
#028  else
#029  {
#030        rcText.left = 10;
#031        rcText.top = 40;
#032        rcText.right = 300;
#033        rcText.bottom = 80;
#034  }
#035
#036  //创建黑色的画刷,
#037  HBRUSH hbrush = CreateSolidBrush(RGB(0, 0, 0));
#038
#039  //用黑色的画刷填充四边形的颜色.
#040  FillRect(hDC,&rcText,hbrush);
#041
#042  //删除画刷.
#043  DeleteObject(hbrush);
#044
#045  //设置透明背景
#046  int nOldMode = SetBkMode(hDC,TRANSPARENT);
#047
#048  //显示字符串在四边形的中间位置.
#049  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#050        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);
#051
#052  //恢复原来的字体.
#053  HGDIOBJ hFont = SelectObject(hDC,hOldFont);
#054  DeleteObject(hFont);
#055
#056  //恢复原来的模式.
#057  SetBkMode(hDC,nOldMode);
#058
#059  //恢复原来的颜色.
#060  SetTextColor(hDC,crOld);
#061 }
#062
```

Windows API一日一练(31)MoveToEx 和 LineTo 函数

现在的世界流行图形界面，而不是文字，因此在软件开发里，肯定需要画图的，比如简单地画线，画一些比较特别的图形。比如让你画一个走动的时钟，就需要不断地画秒针、分针等等。MoveToEx 是用来移动当前画笔的位置，LineTo 是用来画直线的函数，其实在计算机图形里的直线显示是使用光栅图形学里的原理。

函数 MoveToEx 和 LineTo 声明如下：

```
WINGDIAPI BOOL WINAPI MoveToEx( __in HDC hdc, __in int x, __in int y,
__out_opt LPPOINT lppt);
```

hdc 是当前设备的句柄。
x 是 X 轴的位置，水平方向，一般原点是在屏幕左上角的位置。
y 是 Y 轴的位置，垂直方向。
lppt 是移动前的坐标位置。

```
WINGDIAPI BOOL WINAPI LineTo( __in HDC hdc, __in int x, __in int y);
```

hdc 是当前设备的句柄。
x 是 X 轴的位置，水平方向，一般原点是在屏幕左上角的位置。
y 是 Y 轴的位置，垂直方向。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/08 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //移到指定位置.
#009  POINT ptLeftTop;
#010  ptLeftTop.x = 10;
#011  ptLeftTop.y = 10;
#012  MoveToEx(hDC,ptLeftTop.x,ptLeftTop.y,NULL);
#013
#014  //从(10，10)到(100，100)画一条直线.
#015  ptLeftTop.x = 100;
#016  ptLeftTop.y = 100;
#017  LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#018
#019 }
```

Windows API一日一练(32)CreatePen 函数

画画是讲究色彩与线条，不同的地方是采用不同的画笔。上面显示直线，都是采用设备缺省的画笔来画直线。现在就来介绍怎么样创建自己的画笔，比如设置画笔的颜色，画笔的大小。像 Windows 按钮显示为 3D 的形状，其实就是用两种颜色画笔分别画相应的线，就生成按钮。而创建画笔就需要使用到 CreatePen 函数。

函数 CreatePen 声明如下：

WINGDIAPI HPEN WINAPI CreatePen(__in int iStyle, __in int cWidth, __in COLORREF color);

iStyle 是画笔的类型，比如是实线，还是虚线等等。

cWidth 是线的宽度。

color 是线的颜色。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生  2007/09/10 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //移到指定位置.
#009  POINT ptLeftTop;
#010  ptLeftTop.x = 10;
#011  ptLeftTop.y = 10;
#012  MoveToEx(hDC,ptLeftTop.x,ptLeftTop.y,NULL);
#013
#014  //修改直线的颜色,粗细.
#015  HPEN hPen = CreatePen(PS_SOLID, 10, RGB(0, 255, 0));
#016
#017  //设置当前设备的画笔.
#018  HGDIOBJ hOldPen = SelectObject(hDC,hPen);
#019
#020  //从(10,10)到(100,100)画一条直线.
#021  ptLeftTop.x = 100;
#022  ptLeftTop.y = 100;
#023  LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#024
#025  //恢复原来的画笔.
#026  SelectObject(hDC,hOldPen);
#027
#028  //删除创建的画笔.
#029  DeleteObject(hPen);
#030
```

```
#031 }
```

Windows API一日一练(33)ExtCreatePen 函数

使用前面介绍的 CreatePen 函数来画大于 1 的直线时，会发现直线两端全是圆角的，有时候需要画得有角，那么这样的函数就不满足需求了，这时就需要使用另一个 API 函数 ExtCreatePen 来创建合适的画笔。ExtCreatePen 函数可以创建几何画笔，还可以创建装饰用的画笔，装饰的画笔是用来画一些图案使用的，这样就需要快速的算法来实现，显示比几何的画笔在速度上快很多。比如你在画 GPS 地图时，当需要装饰使用的图案，就可以这种画笔，会明显地提高显示的速度。

函数 ExtCreatePen 声明如下：

```
WINGDIAPI HPEN WINAPI ExtCreatePen( __in DWORD iPenStyle,
                                    __in DWORD cWidth,
                                    __in CONST LOGBRUSH *plbrush,
                                    __in DWORD cStyle,
                                    __in_ecount_opt(cStyle) CONST DWORD *pstyle);
```

iPenStyle 是画笔的类型。
cWidth 是画笔的宽度，当创建装饰画笔时宽度一定要设置为 1。
plbrush 是画笔的属性。
cStyle 是后面自定义样式数组的个数。
pstyle 是自定义样式数组。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/10 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //移到指定位置.
#009  POINT ptLeftTop;
#010  ptLeftTop.x = 10;
#011  ptLeftTop.y = 10;
#012  MoveToEx(hDC,ptLeftTop.x,ptLeftTop.y,NULL);
#013
#014  //修改直线的颜色,粗细.
#015  LOGBRUSH lb;
#016  lb.lbStyle = BS_SOLID;
#017  lb.lbColor = RGB(0,0,255);
#018  lb.lbHatch = 0;
#019
```

```
#020 //创建装饰笔.
#021 HPEN hPen = ExtCreatePen(PS_COSMETIC | PS_DASH,
#022       1, &lb, 0, NULL);
#023
#024 //设置当前设备的画笔.
#025 HGDIOBJ hOldPen = SelectObject(hDC,hPen);
#026
#027 //从(10,10)到(100,100)画一条直线.
#028 ptLeftTop.x = 100;
#029 ptLeftTop.y = 100;
#030 LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#031
#032 //创建一个端点是平的画笔.
#033 HPEN hPenGeom = ExtCreatePen(PS_GEOMETRIC | PS_SOLID|
#034       PS_ENDCAP_FLAT,16, &lb, 0, NULL);
#035
#036 SelectObject(hDC,hPenGeom);
#037 //从(100,100)到(10,100)画一条直线.
#038 ptLeftTop.x = 10;
#039 ptLeftTop.y = 100;
#040 LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#041
#042 //创建一个端点是圆角的画笔.
#043 HPEN hPenGeomRound = ExtCreatePen(PS_GEOMETRIC | PS_SOLID|
#044       PS_ENDCAP_ROUND,16, &lb, 0, NULL);
#045
#046 SelectObject(hDC,hPenGeomRound);
#047 //从(10,100)到(10,10)画一条直线.
#048 ptLeftTop.x = 10;
#049 ptLeftTop.y = 10;
#050 LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#051
#052
#053 //恢复原来的画笔.
#054 SelectObject(hDC,hOldPen);
#055
#056 //删除创建的画笔.
#057 DeleteObject(hPen);
#058 DeleteObject(hPenGeom);
#059 DeleteObject(hPenGeomRound);
#060
#061 }
```

Windows API一日一练(34)GetSysColor 函数

当你需要自己显示一个与众不同的按钮时，就需要使用下面的方法来创建。当然这里也是综合地使用前面学习过的知识进行一次综合的练习。演示怎么样使用众多的 API 函数，这里还可以学会使用 GetSysColor 函数来获取系统的颜色。

```
WINUSERAPI
DWORD
WINAPI
GetSysColor(
    __in int nIndex);
```

nIndex 是系统定义的颜色索引值。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/12 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //显示一个按钮。
#009  //设置按钮背景颜色。
#010  RECT rcText;
#011  rcText.left = 10;
#012  rcText.top = 10;
#013  rcText.right = 100;
#014  rcText.bottom = 100;
#015
#016  HBRUSH hbrush = CreateSolidBrush(GetSysColor(COLOR_3DFACE));
#017  //用黑色的画刷填充四边形的颜色.
#018  FillRect(hDC,&rcText,hbrush);
#019  DeleteObject(hbrush);
#020
#021  //
#022  HPEN hLight = CreatePen(PS_SOLID,1,
#023        GetSysColor(COLOR_3DHIGHLIGHT));
#024  HPEN hShdaow = CreatePen(PS_SOLID,1,
#025        GetSysColor(COLOR_3DDKSHADOW));
#026
#027  //
#028  //移到指定位置.
#029  POINT ptLeftTop;
#030  ptLeftTop.x = 20;
#031  ptLeftTop.y = 20;
```

```
#032  MoveToEx(hDC,ptLeftTop.x,ptLeftTop.y,NULL);
#033
#034  //显示白线。
#035  HGDIOBJ hOldPen = SelectObject(hDC,hLight);
#036  ptLeftTop.x = 20;
#037  ptLeftTop.y = 90;
#038  LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#039
#040  ptLeftTop.x = 20;
#041  ptLeftTop.y = 20;
#042  MoveToEx(hDC,ptLeftTop.x,ptLeftTop.y,NULL);
#043  ptLeftTop.x = 90;
#044  ptLeftTop.y = 20;
#045  LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#046
#047  //显示黑色线。
#048  SelectObject(hDC,hShdaow);
#049  ptLeftTop.x = 90;
#050  ptLeftTop.y = 90;
#051  LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#052
#053  ptLeftTop.x = 20;
#054  ptLeftTop.y = 90;
#055  LineTo(hDC,ptLeftTop.x,ptLeftTop.y);
#056
#057  //
#058  SelectObject(hDC,hOldPen);
#059  DeleteObject(hLight);
#060  DeleteObject(hShdaow);
#061
#062  int nOldMode = SetBkMode(hDC,TRANSPARENT);
#063  //输出文字。
#064  std::wstring strShow(_T("按钮"));
#065  DrawText(hDC,strShow.c_str(),(int)strShow.length(),&rcText,
#066        DT_CENTER|DT_VCENTER|DT_SINGLELINE|DT_END_ELLIPSIS);

#067
#068  //
#069  SetBkMode(hDC,nOldMode);
#070 }
```

这是综合使用前面学习过的 API 函数来画一个按钮，如果有什么不了解的，请看看以前学习过的 API 函数。

Windows API一日一练(36)SetWindowText 函数

在开发软件里，有时候需要改变窗口上标题的文字。比如按钮上的文字，为了适应多国语言的显示，就需要改变它显示的内容。这时就需要使用 SetWindowText 函数来改它的内容。下面的例子就实现改变按钮的标题。

函数 SetWindowText 声明如下：

```
WINUSERAPI
BOOL
WINAPI
SetWindowTextA(
    __in HWND hWnd,
    __in_opt LPCSTR lpString);
WINUSERAPI
BOOL
WINAPI
SetWindowTextW(
    __in HWND hWnd,
    __in_opt LPCWSTR lpString);
#ifdef UNICODE
#define SetWindowText SetWindowTextW
#else
#define SetWindowText SetWindowTextA
#endif // !UNICODE
```

hWnd 是窗口的句柄。

lpString 是要需要显示的文字。

调用这个函数的例子如下：

```
#001 //
#002 // 响应命令.
#003 // 蔡军生 2007/09/14 QQ:9073204
#004 //
#005 LRESULT CCaiWinMsg::OnCommand(int nID,int nEvent)
#006 {
#007  // 菜单选项命令响应:
#008  switch (nID)
#009  {
#010  case IDC_CREATEBTN:
#011        //显示一个按钮。
#012        if (!m_hBtn)
#013        {
#014              m_hBtn = CreateWindow(_T("BUTTON"),_T("按钮"),
#015                 WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
```

```
#016                50,50,100,32,
#017                 m_hWnd,(HMENU)IDC_BTN,m_hInstance,NULL);
#018        }
#019        break;
#020  case IDC_BTN:
#021        OutputDebugString(_T("按钮按下\r\n"));
#022        {
#023            static bool bChangeText = true;
#024            if (bChangeText)
#025            {
#026                //设置按钮的文字。
#027                SetWindowText(m_hBtn,_T("改变它"));
#028            }
#029            else
#030            {
#031                //设置按钮的文字。
#032                SetWindowText(m_hBtn,_T("按钮"));
#033            }
#034
#035            //每一次都改变。
#036            bChangeText = !bChangeText;
#037        }
#038        break;
#039  default:
#040        return CCaiWin::OnCommand(nID,nEvent);
#041  }
#042
#043  return 1;
#044 }
```


Windows API一日一练(37)MoveWindow 函数

当你设计一个对话框的窗口时，就需要布局好所有按钮、文本显示框等等，由于每个按钮都是一个窗口，那么就需要移动这些窗口到合适的位置，这时就需要使用到 MoveWindow 函数。或者当你的界面需要动态地修改按钮位置，比如窗口放大了，按钮就需要跟着移动，否则按钮还在原来的位置，放大也不会移动按钮的位置，这时也需要使用 MoveWindow 函数重新设置按钮的位置。只要你想移动窗口，就可以考虑使用这个函数来实现。

函数 MoveWindow 声明如下：

```
WINUSERAPI
BOOL
WINAPI
MoveWindow(
```

```
    __in HWND hWnd,
    __in int X,
    __in int Y,
    __in int nWidth,
    __in int nHeight,
    __in BOOL bRepaint);
```


hWnd 是窗口的句柄。
X 是窗口在 X 轴的位置。
Y 是窗口在 Y 辆的位置。
nWidth 是窗口的宽度。
nHeight 是窗口的高度。
bRepaint 是设置是否重画窗口和父窗口。当设置为 TRUE 时，就进行重画。当设置为 FALSE 时，就不进行重画，需要手功进行更新指定的区域。

调用这个函数的例子如下：

```
#001 //
#002 // 响应命令.
#003 // 蔡军生 2007/09/16 QQ:9073204
#004 //
#005 LRESULT CCaiWinMsg::OnCommand(int nID,int nEvent)
#006 {
#007  // 菜单选项命令响应:
#008  switch (nID)
#009  {
#010  case IDC_CREATEBTN:
#011        //显示一个按钮。
#012        if (!m_hBtn)
#013        {
#014              m_hBtn = CreateWindow(_T("BUTTON"),_T("按钮"),
#015                 WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
#016                 50,50,100,32,
#017                  m_hWnd,(HMENU)IDC_BTN,m_hInstance,NULL);
#018        }
#019        break;
#020  case IDC_BTN:
#021        OutputDebugString(_T("按钮按下\r\n"));
#022        {
#023             static bool bChangeText = true;
#024             if (bChangeText)
#025             {
#026                 //设置按钮的文字。
#027                 SetWindowText(m_hBtn,_T("改变它"));
#028
#029                 //改变按钮窗口的位置和大小。
```

```
#030             MoveWindow(m_hBtn,10,10,100,32,TRUE);
#031         }
#032         else
#033         {
#034             //设置按钮的文字。
#035             SetWindowText(m_hBtn,_T("按钮"));
#036
#037             //改变按钮窗口的位置和大小。
#038             MoveWindow(m_hBtn,50,50,100,32,TRUE);
#039         }
#040
#041         //每一次都改变。
#042         bChangeText = !bChangeText;
#043       }
#044       break;
#045  default:
#046       return CCaiWin::OnCommand(nID,nEvent);
#047  }
#048
#049  return 1;
#050 }
```


Windows API一日一练(38)SetWindowPos 函数

有一天，用户突然对我说，你这个窗口能不能放到最顶端，这样操作和打开文件就很不方便了。这个功能就需要改变窗口的属性了。比如大家使用 QQ 时，就有一个功能，设置 QQ 的窗口在最顶端，不管你选择了什么窗口，QQ 的界面永远都在最前面。又像 FlashGet 的状态查看窗口，一直保持在窗口的最前端，让你看到当前下载的流量程况。现在股票那么火爆，很多人一边工作，一边查看股票，如果错失了机会，又少了很多钱的啊！面对这样的需求，就需要把一些窗口永远摆在最前面，这样起到提示用户的作用。因此，学会使用 SetWindowPos 函数，就成为能否让软件满足客户需求的关键了。与 MoveWindow 函数相比，SetWindowPos 函数的功能比较强大一点。

函数 SetWindowPos 声明如下：

```
WINUSERAPI
BOOL
WINAPI
SetWindowPos(
    __in HWND hWnd,
    __in_opt HWND hWndInsertAfter,
    __in int X,
    __in int Y,
    __in int cx,
```

```
    __in int cy,
    __in UINT uFlags);
```

hWnd 是窗口的句柄。
hWndInsertAfter 是窗口 Z 顺序属性。
X 是窗口在 X 轴的位置。
Y 是窗口在 Y 辆的位置。
cx 是窗口的宽度。
cy 是窗口的高度。
uFlags 是选择设置的标志。

调用这个函数的例子如下：

```
#001 //
#002 // 响应命令.
#003 // 蔡军生 2007/09/16 QQ:9073204
#004 //
#005 LRESULT CCaiWinMsg::OnCommand(int nID,int nEvent)
#006 {
#007  // 菜单选项命令响应:
#008  switch (nID)
#009  {
#010  case IDC_CREATEBTN:
#011        //显示一个按钮。
#012        if (!m_hBtn)
#013        {
#014              m_hBtn = CreateWindow(_T("BUTTON"),_T("按钮"),
#015                 WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
#016                 50,50,100,32,
#017                  m_hWnd,(HMENU)IDC_BTN,m_hInstance,NULL);
#018        }
#019        break;
#020  case IDC_BTN:
#021        OutputDebugString(_T("按钮按下\r\n"));
#022        {
#023             static bool bChangeText = true;
#024             if (bChangeText)
#025             {
#026                 //设置按钮的文字。
#027                 SetWindowText(m_hBtn,_T("改变它"));
#028
#029                 //改变按钮窗口的位置和大小。
#030                 MoveWindow(m_hBtn,10,10,100,32,TRUE);
#031
#032                 //改变主窗口为最顶端窗口。
```

```
#033          SetWindowPos(m_hWnd,HWND_TOPMOST,0,0,0,0,SWP_NOMOVE|SWP_NOSIZE);
#034          }
#035          else
#036          {
#037              //设置按钮的文字。
#038              SetWindowText(m_hBtn,_T("按钮"));
#039
#040              //改变按钮窗口的位置和大小。
#041              MoveWindow(m_hBtn,50,50,100,32,TRUE);
#042
#043              //改变主窗口为普通窗口。
#044          SetWindowPos(m_hWnd,HWND_NOTOPMOST,0,0,0,0,SWP_NOMOVE|SWP_NOSIZE);
#045          }
#046
#047          //每一次都改变。
#048          bChangeText = !bChangeText;
#049      }
#050      break;
#051  default:
#052      return CCaiWin::OnCommand(nID,nEvent);
#053  }
#054
#055  return 1;
#056 }
```


Windows API一日一练(39)AnimateWindow 函数

当你开发一款年轻人使用的软件时，肯定想用一点动感的特性来吸引他们。比如 QQ 软件就有这个特性，当你把它放到桌面边上时，就会自动隐藏起来，当你的鼠标放到那里时，就会自动慢慢移动出来，副有动感的特性，肯定让年轻人喜欢上它的。还有当你开发软件时，想先动态地显示公司的商标，或者公司的宣传材料时，就会使用到闪屏的效果。其实这两个特性都可以使用 API 函数 AnimateWindow 来实现的，下面就来介绍怎么使用它。

函数 AnimateWindow 声明如下：

```
#if(WINVER >= 0x0500)
WINUSERAPI
BOOL
WINAPI
AnimateWindow(
    __in HWND hWnd,
    __in DWORD dwTime,
```

```
    __in DWORD dwFlags);
#endif /* WINVER >= 0x0500 */
```

hWnd 是窗口的句柄。
dwTime 是动态出现的时间。
dwFlags 是显示效果的标志设置。

调用这个函数的例子如下：


```
#001 //消息处理函数。
#002 //
#003 // 蔡军生 2007/08/13 QQ:9073204
#004 // 蔡军生 2007/09/17 QQ:9073204 添加动画窗口显示和隐藏。
#005 //
#006 LRESULT CCaiWinMsg::OnMessage(UINT nMessage,
#007                               WPARAM wParam, LPARAM lParam)
#008 {
#009  //
#010  switch(nMessage)
#011  {
#012  case WM_PAINT:
#013       return OnPaint(wParam,lParam);
#014       break;
#015  case WM_CREATE:
#016
#017       break;
#018  case WM_DESTROY:
#019       return OnDestroy(wParam,lParam);
#020       break;
#021  case WM_ACTIVATEAPP:
#022       if (wParam == TRUE)
#023       {
#024           //窗口从上到下显示出来。
#025           AnimateWindow(m_hWnd,1000,AW_SLIDE|AW_VER_POSITIVE);
#026       }
#027       else
#028       {
#029           //窗口从下到上隐藏起来。
#030           AnimateWindow(m_hWnd,1000,AW_HIDE|AW_VER_NEGATIVE);
#031       }
#032       break;
#033  }
#034
#035  return CCaiWin::OnMessage(nMessage,wParam,lParam);
```

```
#036 }
#037
```

Windows API一日一练(40)CreateRectRgn 和 CombineRgn 函数

创新是永恒的追求。当大家习惯 Windows 的界面时，又想自己开发的软件耳目一新的感觉，那么就得要改变窗口的形状，比如心形的窗口，总之是不规则的窗口。这时就需要使用到叫做区域的技术。区域就是把不同的形状的图形进行组合，然后可以填充它，或者在它那里显示。比如 MSN 左边的 TAB 按钮，应就是区域的运用就可以实现它了。

函数 CreateRectRgn 声明如下：

```
WINGDIAPI HRGN     WINAPI CreateRectRgn( __in int x1, __in int y1, __in int x2, __in int y2);
```

x1 和 ***y1*** 是区域左上角的 X 轴和 Y 轴坐标。
x2 和 ***y2*** 是区域右下角的 X 轴和 Y 轴坐标。
返回值是创建的区域。

函数 CombineRgn 声明如下：

```
WINGDIAPI int      WINAPI CombineRgn( __in_opt HRGN hrgnDst, __in_opt HRGN hrgnSrc1, __in_opt HRGN hrgnSrc2, __in int iMode);
```

hrgnDst 是组合的区域。
hrgnSrc1 是想组合的第一个区域。
hrgnSrc2 是想组合的第二个区域。
iMode 是区域的组合方式，比如相与，相或，异或等等。

调用这个函数的例子如下：

```
#001 //
#002 //界面显示输出.
#003 //
#004 //蔡军生 2007/09/19 QQ:9073204 深圳
#005 //
#006 void CCaiWinMsg::OnDraw(HDC hDC)
#007 {
#008  //创建两个方形区域。
#009  HRGN rgnRect1 = CreateRectRgn(10,10,100,100);
#010  HRGN rgnRect2 = CreateRectRgn(50,50,200,200);
#011
#012  //合并两个区域。
#013  CombineRgn(rgnRect1,rgnRect1,rgnRect2,RGN_XOR);
#014
#015  //创建画刷。
#016  HBRUSH hbrush = CreateSolidBrush(RGB(0, 0, 0));
#017
```

```
#018  //填充区域。
#019  FillRgn(hDC,rgnRect1,hbrush);
#020
#021  //删除画刷.
#022  DeleteObject(hbrush);
#023
#024  //删除创建的两个区域.
#025  DeleteObject(rgnRect1);
#026  DeleteObject(rgnRect2);
#027
#028 }
```


Windows API一日一练(41)FindWindowEx 函数

当你想控制一个现有的窗口程序时，就需要获取那个程序的窗口句柄。比如有一些黑客软件需要查找到窗口，然后修改窗口的标题。在外挂流行的今天，惊奇地发现它们也可以修改输入窗口的文字。这其中，就需要使用到 FindWindowEx 函数来定位窗口。下面就来使用这个函数来实现控制 Windows 里带的计算器程序。打开计算器程序，最小化在状态下面，运行本例子，点击创建按钮后，就可以点按钮，就会把计算器显示在最前面。

函数 FindWindowEx 声明如下：

```
#if(WINVER >= 0x0400)
WINUSERAPI
HWND
WINAPI
FindWindowExA(
    __in_opt HWND hWndParent,
    __in_opt HWND hWndChildAfter,
    __in_opt LPCSTR lpszClass,
    __in_opt LPCSTR lpszWindow);
WINUSERAPI
HWND
WINAPI
FindWindowExW(
    __in_opt HWND hWndParent,
    __in_opt HWND hWndChildAfter,
    __in_opt LPCWSTR lpszClass,
    __in_opt LPCWSTR lpszWindow);
#ifdef UNICODE
#define FindWindowEx FindWindowExW
#else
#define FindWindowEx FindWindowExA
```

#endif // !UNICODE
hWndParent 是找查窗口的父窗口句柄，如果父窗口是桌面，就可以设置为 NULL。
hWndChildAfter 是子窗口开始位置。
lpszClass 是窗口注册的类型。
lpszWindow 是窗口的标题。

调用这个函数的例子如下：

```
#001 //
#002 // 响应命令.
#003 // 蔡军生 2007/09/20 QQ:9073204
#004 //
#005 LRESULT CCaiWinMsg::OnCommand(int nID,int nEvent)
#006 {
#007  // 菜单选项命令响应:
#008  switch (nID)
#009  {
#010  case IDC_CREATEBTN:
#011        //显示一个按钮。
#012        if (!m_hBtn)
#013        {
#014              m_hBtn = CreateWindow(_T("BUTTON"),_T("按钮"),
#015                 WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
#016                 50,50,100,32,
#017                  m_hWnd,(HMENU)IDC_BTN,m_hInstance,NULL);
#018        }
#019        break;
#020  case IDC_BTN:
#021        {
#022            //查找计算器的窗口。
#023            HWND hWnd = FindWindowEx(NULL, NULL,NULL,_T("计
算器"));
#024            if (hWnd != NULL)
#025            {
#026                //窗口是否最小化。
#027                if (IsIconic(hWnd))
#028                {
#029                    //恢复窗口。
#030                    ShowWindow(hWnd,SW_RESTORE);
#031                }
#032                else
#033                {
#034                    //显示窗口。
```

```
#035                    ShowWindow(hWnd,SW_SHOWNORMAL);

#036                }
#037
#038                //把窗口显示到最前面。
#039                BringWindowToTop(hWnd);
#040
#041                OutputDebugString(_T("按钮按下\r\n"));
#042            }
#043        }
#044        break;
#045  default:
#046        return CCaiWin::OnCommand(nID,nEvent);
#047  }
#048
#049  return 1;
#050 }
```


Windows API一日一练(42)CreateThread 函数

随着时代的发展，计算机技术发展得很快。CPU 已经从单核心到多核心的转变，也就是一个 CPU 里具备了同时做多件事情的能力，而不是过去的分时复用了，而是实实在在地做多件事情。因此，开发软件也进入了一个新时代，就是多线程软件的开发时代。如何合理地分配多个线程同时运行，是提高软件效率的关键因素了。比如像网络游戏的客户端里，就可以使用一个线程不断地更新游戏的界面，分配另外一个线程不断地发送和接收网络的数据，这样 CPU 的两个核心都在不断地工作。如果还像以前那样使用一个线程的话，就会发现只有一个内核在做事情。当然使用线程的编程模型，也会大大地简化软件的复杂性。下面就来使用线程的 API 函数。

函数 CreateThread 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
CreateThread(
    __in_opt LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in     SIZE_T dwStackSize,
    __in     LPTHREAD_START_ROUTINE lpStartAddress,
    __in_opt LPVOID lpParameter,
    __in     DWORD dwCreationFlags,
    __out_opt LPDWORD lpThreadId
    );
```

lpThreadAttributes 是线程的属性。
dwStackSize 是线程的栈大小。
lpStartAddress 是线程函数的开始地址。
lpParameter 是传送给线程函数的参数。
dwCreationFlags 是创建线程标志，比如挂起线程。
lpThreadId 是标识这个线程的 ID。

调用这个函数的例子如下：

```
#001 //
#002 //线程运行函数。
#003 //蔡军生 2007/09/21
#004 //
#005 static DWORD WINAPI ThreadProc(
#006                         LPVOID lpParameter
#007                         )
#008 {
#009  //输出到调试窗口。
#010  OutputDebugString(_T("ThreadProc 线程函数运行\r\n"));
#011
#012  //线程返回码。
#013  return 0;
#014 }
#015
#016
#017
#018 //
#019 // 响应命令.
#020 // 蔡军生 2007/09/21 QQ:9073204
#021 //
#022 LRESULT CCaiWinMsg::OnCommand(int nID,int nEvent)
#023 {
#024  // 菜单选项命令响应:
#025  switch (nID)
#026  {
#027  case IDC_CREATEBTN:
#028        //显示一个按钮。
#029        if (!m_hBtn)
#030        {
#031              m_hBtn = CreateWindow(_T("BUTTON"),_T("按钮"),
#032                 WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
#033                 50,50,100,32,
#034                  m_hWnd,(HMENU)IDC_BTN,m_hInstance,NULL);
#035        }
```

```
#036        break;
#037  case IDC_BTN:
#038        {
#039            //传给线程的简单参数。
#040            int nParam = 110;
#041
#042            //保存线程的 ID。
#043            DWORD dwThreadID = 0;
#044
#045            //创建线程。
#046            HANDLE hThread = CreateThread(
#047                NULL,                   //安全属性使用缺省。
#048                0,                      //线程的堆栈大小。
#049                ThreadProc,                 //线程运行函数地址。
#050                &nParam,                //传给线程函数的参数。
#051                0,                      //创建标志。
#052                &dwThreadID);       //成功创建后的线程标识码。
#053
#054            //等待线程结束。
#055            WaitForSingleObject(hThread,INFINITE);
#056
#057            //删除的线程资源。
#058            CloseHandle(hThread);
#059
#060            //
#061            OutputDebugString(_T("按钮按下\r\n"));
#062
#063        }
#064        break;
#065  default:
#066        return CCaiWin::OnCommand(nID,nEvent);
#067  }
#068
#069  return 1;
#070 }
```

Windows API一日一练(43)WaitForSingleObject 函数

上面已经介绍怎么样创建一个简单的线程，在那里就需要使用函数 WaitForSingleObject，它是用来做什么的呢？其实它是用来实现等待线程结束的，它的机理是这样的，通知 Windows 操作系统，现在我进入睡眼状态，当我关注的对象条件是否满足，如果满足了就吵醒我。在那里关注的对象是线程是否退出，这是一个条件测试。如果不等待线程关闭，就去删除线程的资源，就会出错的。使用前面的线程是简单一些，但它不合适复用，现在就使

用 C＋＋的类来封装一个比较好用的类，这样就可以很方便地创建很多线程来使用，当然还可以继承它，实现更复杂的功能，下面就来学习这个例子。

函数 WaitForSingleObject 声明如下：

```
WINBASEAPI
DWORD
WINAPI
WaitForSingleObject(
    __in HANDLE hHandle,
    __in DWORD dwMilliseconds
    );
```

hHandle 是等待对象的句柄。
dwMilliseconds 是等待的时间条件，可以永远等待下去。

调用这个函数的例子如下：

```
#001 #pragma once
#002
#003 //线程类。
#004 //蔡军生 2007/09/23
#005 class CThread
#006 {
#007 public:
#008
#009  CThread(void)
#010  {
#011         m_hThread = NULL;
#012  }
#013
#014  virtual ~CThread(void)
#015  {
#016        if (m_hThread)
#017        {
#018             //删除的线程资源。
#019             ::CloseHandle(m_hThread);
#020        }
#021
#022  }
#023
#024  //创建线程
#025  HANDLE CreateThread(void)
#026  {
#027        //创建线程。
```

```
#028         m_hThread = ::CreateThread(
#029             NULL,                  //安全属性使用缺省。
#030             0,                     //线程的堆栈大小。
#031             ThreadProc,                 //线程运行函数地址。
#032             this,                  //传给线程函数的参数。
#033             0,                     //创建标志。
#034             &m_dwThreadID);        //成功创建后的线程标识码。
#035
#036        return m_hThread;
#037  }
#038
#039  //等待线程结束。
#040  void WaitFor(DWORD dwMilliseconds = INFINITE)
#041  {
#042        //等待线程结束。
#043        ::WaitForSingleObject(m_hThread,dwMilliseconds);
#044  }
#045
#046 protected:
#047  //
#048  //线程运行函数。
#049  //蔡军生 2007/09/21
#050  //
#051  static DWORD WINAPI ThreadProc(LPVOID lpParameter)
#052  {
#053        //输出到调试窗口。
#054        ::OutputDebugString(_T("ThreadProc 线程函数运行\r\n"));
#055
#056        //线程返回码。
#057        return 0;
#058  }
#059
#060 protected:
#061   HANDLE m_hThread;         //线程句柄。
#062   DWORD m_dwThreadID;            //线程 ID。
#063
#064 };
#065
```

使用这个线程：

```
#001 //
#002 // 响应命令.
#003 // 蔡军生 2007/09/21 QQ:9073204
#004 //
```

```
#005 LRESULT CCaiWinMsg::OnCommand(int nID,int nEvent)
#006 {
#007  // 菜单选项命令响应:
#008  switch (nID)
#009  {
#010  case IDC_CREATEBTN:
#011        //显示一个按钮。
#012        if (!m_hBtn)
#013        {
#014              m_hBtn = CreateWindow(_T("BUTTON"),_T("按钮"),
#015                 WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
#016                 50,50,100,32,
#017                  m_hWnd,(HMENU)IDC_BTN,m_hInstance,NULL);
#018        }
#019        break;
#020  case IDC_BTN:
#021        {
#022            CThread threadDemo;
#023            threadDemo.CreateThread();
#024            threadDemo.WaitFor();
#025
#026             //
#027             OutputDebugString(_T("按钮按下\r\n"));
#028        }
#029        break;
#030  default:
#031        return CCaiWin::OnCommand(nID,nEvent);
#032  }
#033
#034  return 1;
#035 }
```


Windows API一日一练(44)wsprintf函数

接着上面，再继续实现更加强大的线程类。从上面的 C++类里可以看到，要在静态函数里使用类的成员就需要获取 this 指针，也就是通过 CreateThread 函数里把类的 this 指针传送进来的，这样在函数 ThreadProc 里的参数 lpParameter 就是 this 指针了。因此把参数 lpParameter 转换为 CThread 类指针，这样就可以使用类的成员。在这个例子里使用 wsprintf 函数来格式化线程 ID 输出来，下面就来详细地分析例子吧。

函数 wsprintf 声明如下：

WINUSERAPI

```
int
WINAPIV
wsprintfA(
    __out LPSTR,
    __in __format_string LPCSTR,
    ...);
WINUSERAPI
int
WINAPIV
wsprintfW(
    __out LPWSTR,
    __in __format_string LPCWSTR,
    ...);
#ifdef UNICODE
#define wsprintf wsprintfW
#else
#define wsprintf wsprintfA
#endif // !UNICODE
```


LPWSTR 是格化后输出的缓冲区。
LPCWSTR 是输入格式化字符串。
...是可变参数。

调用这个函数的例子如下：

```
#001 #pragma once
#002
#003 //线程类。
#004 //蔡军生 2007/09/23
#005 class CThread
#006 {
#007 public:
#008
#009  CThread(void)
#010  {
#011         m_hThread = NULL;
#012  }
#013
#014  virtual ~CThread(void)
#015  {
#016        if (m_hThread)
#017        {
#018             //删除的线程资源。
#019             ::CloseHandle(m_hThread);
#020        }
```

```
#021
#022 }
#023
#024 //创建线程
#025 HANDLE CreateThread(void)
#026 {
#027       //创建线程。
#028        m_hThread = ::CreateThread(
#029           NULL,                 //安全属性使用缺省。
#030           0,                    //线程的堆栈大小。
#031           ThreadProc,               //线程运行函数地址。
#032           this,                 //传给线程函数的参数。
#033           0,                    //创建标志。
#034           &m_dwThreadID);       //成功创建后的线程标识码。
#035
#036       return m_hThread;
#037 }
#038
#039 //等待线程结束。
#040 void WaitFor(DWORD dwMilliseconds = INFINITE)
#041 {
#042       //等待线程结束。
#043       ::WaitForSingleObject(m_hThread,dwMilliseconds);
#044 }
#045
#046 protected:
#047 //
#048 //线程运行函数。
#049 //蔡军生 2007/09/21
#050 //
#051 static DWORD WINAPI ThreadProc(LPVOID lpParameter)
#052 {
#053       //转换传送入来的参数。
#054       CThread* pThread = reinterpret_cast<CThread *>(lpParameter);
#055       if (pThread)
#056       {
#057           //线程返回码。
#058           //调用类的线程处理函数。
#059            return pThread->Run();
#060       }
#061
#062       //
#063       return -1;
#064 }
```

```
#065
#066 //线程运行函数。
#067 //在这里可以使用类里的成员，也可以让派生类实现更强大的功能。
#068 //蔡军生 2007/09/24
#069 virtual DWORD Run(void)
#070 {
#071         //输出到调试窗口。
#072         ::OutputDebugString(_T("Run()线程函数运行\r\n"));
#073
#074         TCHAR chTemp[128];
#075         wsprintf(chTemp,_T("ThreadID=%d\r\n"),m_dwThreadID
);
#076         ::OutputDebugString(chTemp);
#077
#078         return 0;
#079 }
#080
#081 protected:
#082  HANDLE m_hThread;          //线程句柄。
#083  DWORD m_dwThreadID;          //线程 ID。
#084
#085 };
#086
```

在这个例子里主要在第 54 行获取类的指针，然后在 59 行调用运行类的成员函数 Run()。通过这样的调用后，所有线程的代码都可以写在类 CThread 里面了，这样就达到编写更简单的代码和复用的目的，因此 Run 函数设计为虚函数，让派生类可以改写这个类的功能。

Windows API一日一练(45)CreateEvent 和 SetEvent 函数

当你创建一个线程时，其实那个线程是一个循环，不像上面那样只运行一次的。这样就带来了一个问题，在那个死循环里要找到合适的条件退出那个死循环，那么是怎么样实现它的呢？在 Windows 里往往是采用事件的方式，当然还可以采用其它的方式。在这里先介绍采用事件的方式来通知从线程运行函数退出来，它的实现原理是这样，在那个死循环里不断地使用 WaitForSingleObject 函数来检查事件是否满足，如果满足就退出线程，不满足就继续运行。当在线程里运行阻塞的函数时，就需要在退出线程时，先要把阻塞状态变成非阻塞状态，比如使用一个线程去接收网络数据，同时使用阻塞的 SOCKET 时，那么要先关闭 SOCKET，再发送事件信号，才可以退出线程的。下面就来演示怎么样使用事件来通知线程退出来。

函数 CreateEvent 声明如下：

```
WINBASEAPI
__out
```

```
HANDLE
WINAPI
CreateEventA(
    __in_opt LPSECURITY_ATTRIBUTES lpEventAttributes,
    __in     BOOL bManualReset,
    __in     BOOL bInitialState,
    __in_opt LPCSTR lpName
    );
WINBASEAPI
__out
HANDLE
WINAPI
CreateEventW(
    __in_opt LPSECURITY_ATTRIBUTES lpEventAttributes,
    __in     BOOL bManualReset,
    __in     BOOL bInitialState,
    __in_opt LPCWSTR lpName
    );
#ifdef UNICODE
#define CreateEvent CreateEventW
#else
#define CreateEvent CreateEventA
#endif // !UNICODE
```


lpEventAttributes 是事件的属性。
bManualReset 是指事件手动复位，还是自动复位状态。
bInitialState 是初始化的状态是否处于有信号的状态。
lpName 是事件的名称，如果有名称，可以跨进程共享事件状态。

调用这个函数的例子如下：

```
#001 #pragma once
#002
#003 //线程类。
#004 //蔡军生 2007/09/23 QQ:9073204
#005 class CThread
#006 {
#007 public:
#008
#009  CThread(void)
#010  {
#011          m_hThread = NULL;
#012          m_hEventExit = NULL;
#013  }
#014
#015  virtual ~CThread(void)
```

```
#016 {
#017     if (m_hThread)
#018     {
#019         //删除的线程资源。
#020         ::CloseHandle(m_hThread);
#021     }
#022
#023     if (m_hEventExit)
#024     {
#025         //删除事件。
#026         ::CloseHandle(m_hEventExit);
#027     }
#028
#029 }
#030
#031 //创建线程
#032 HANDLE CreateThread(void)
#033 {
#034     //创建退出事件。
#035      m_hEventExit = ::CreateEvent(NULL,TRUE,FALSE,NULL);
#036     if (!m_hEventExit)
#037      {
#038         //创建事件失败。
#039         return NULL;
#040     }
#041
#042     //创建线程。
#043      m_hThread = ::CreateThread(
#044         NULL,                //安全属性使用缺省。
#045         0,                   //线程的堆栈大小。
#046         ThreadProc,              //线程运行函数地址。
#047         this,                //传给线程函数的参数。
#048         0,                   //创建标志。
#049         &m_dwThreadID);      //成功创建后的线程标识码。
#050
#051     return m_hThread;
#052 }
#053
#054 //等待线程结束。
#055 void WaitFor(DWORD dwMilliseconds = INFINITE)
#056 {
#057     //发送退出线程信号。
#058     ::SetEvent(m_hEventExit);
#059
```

```
#060        //等待线程结束。
#061        ::WaitForSingleObject(m_hThread,dwMilliseconds);
#062    }
#063
#064    protected:
#065    //
#066    //线程运行函数。
#067    //蔡军生 2007/09/21
#068    //
#069    static DWORD WINAPI ThreadProc(LPVOID lpParameter)
#070    {
#071        //转换传送入来的参数。
#072        CThread* pThread = reinterpret_cast<CThread *>(lpParameter);
#073        if (pThread)
#074        {
#075            //线程返回码。
#076            //调用类的线程处理函数。
#077            return pThread->Run();
#078        }
#079
#080        //
#081        return -1;
#082    }
#083
#084    //线程运行函数。
#085    //在这里可以使用类里的成员，也可以让派生类实现更强大的功能。
#086    //蔡军生 2007/09/25
#087    virtual DWORD Run(void)
#088    {
#089        //输出到调试窗口。
#090        ::OutputDebugString(_T("Run()线程函数运行\r\n"));
#091
#092        //线程循环。
#093        for (;;)
#094        {
#095            DWORD dwRet = WaitForSingleObject(m_hEventExit,0);
#096            if (dwRet == WAIT_TIMEOUT)
#097            {
#098                //可以继续运行。
#099                TCHAR chTemp[128];
#100                wsprintf(chTemp,_T("ThreadID=%d\r\n"),m_dwThreadID)
;
#101                ::OutputDebugString(chTemp);
#102
```

```
#103                 //目前没有做什么事情，就让线程释放一下 CPU。
#104                 Sleep(10);
#105             }
#106             else if (dwRet == WAIT_OBJECT_0)
#107             {
#108                 //退出线程。
#109                 ::OutputDebugString(_T("Run() 退出线程\r\n"));
#110                 break;
#111             }
#112             else if (dwRet == WAIT_ABANDONED)
#113             {
#114                 //出错。
#115                 ::OutputDebugString(_T("Run() 线程出错\r\n"));
#116                 return -1;
#117             }
#118         }
#119
#120         return 0;
#121  }
#122
#123 protected:
#124  HANDLE m_hThread;          //线程句柄。
#125  DWORD m_dwThreadID;          //线程 ID。
#126
#127  HANDLE m_hEventExit;    //线程退出事件。
#128 };
#129
```


上面在第 35 行创建线程退出事件，第 95 行检查事件是否可退出线程运行，第 58 行设置退出线程的事件。

Windows API一日一练(46)EnterCriticalSection 和 LeaveCriticalSection 函数

多个线程操作相同的数据时，一般是需要按顺序访问的，否则会引导数据错乱，无法控制数据，变成随机变量。为解决这个问题，就需要引入互斥变量，让每个线程都按顺序地访问变量。这样就需要使用 EnterCriticalSection 和 LeaveCriticalSection 函数。

函数 EnterCriticalSection 和 LeaveCriticalSection 声明如下：

```
WINBASEAPI
VOID
WINAPI
EnterCriticalSection(
```

```
    __inout LPCRITICAL_SECTION lpCriticalSection
    );

WINBASEAPI
VOID
WINAPI
LeaveCriticalSection(
    __inout LPCRITICAL_SECTION lpCriticalSection
    );
```


lpCriticalSection 是创建临界区对象。

调用函数的例子如下：

```
#001 CCaiWinMsg::CCaiWinMsg(void)
#002 {
#003   m_hBtn = NULL;
#004   m_nCount = 0;
#005
#006   m_pThreadA = NULL;
#007   m_pThreadB = NULL;
#008
#009  //
#010  InitializeCriticalSection(&m_csCount);
#011
#012 }
#013
#014 CCaiWinMsg::~CCaiWinMsg(void)
#015 {
#016  DeleteCriticalSection(&m_csCount);
#017 }
#018
```

第 10 行是创建临界区对象。
第 16 行是删除临界区对象。

```
#001 //
#002 //窗口的消息处理类。
#003 //蔡军生 2007/08/13
#004 //
#005 class CCaiWinMsg :
#006  public CCaiWin
#007 {
#008 public:
#009  CCaiWinMsg(void);
```

```
#010  virtual ~CCaiWinMsg(void);
#011
#012  //线程操作函数。
#013  int AddCount(void)
#014  {
#015        //
#016        EnterCriticalSection(&m_csCount);
#017        int nRet = m_nCount++;
#018        LeaveCriticalSection(&m_csCount);
#019
#020        return nRet;
#021  }
```


在函数 AddCount 里调用 EnterCriticalSection 和 LeaveCriticalSection 来互斥访问变量 m_nCount。通过上面这种方法，就可以实现多线程按顺序地访问相同的变量。

Windows API一日一练(47)CreateSemaphore 和 ReleaseSemaphore 函数

在开发软件的过程中，多线程的程序往往需要实现相互通讯，比如几个线程添加一个消息到队列里，而另一个线程在睡眠时，就需要唤醒那个线程来处理事情。在这其中，就需要使用到信号量来进行同步。CreateSemaphore 是创建信号量，ReleaseSemaphore 是增加信号量。

函数 CreateSemaphore 和 ReleaseSemaphore 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
CreateSemaphoreA(
    __in_opt LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
    __in     LONG lInitialCount,
    __in     LONG lMaximumCount,
    __in_opt LPCSTR lpName
    );
WINBASEAPI
__out
HANDLE
WINAPI
CreateSemaphoreW(
    __in_opt LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
    __in     LONG lInitialCount,
    __in     LONG lMaximumCount,
    __in_opt LPCWSTR lpName
```

```
    );
#ifdef UNICODE
#define CreateSemaphore CreateSemaphoreW
#else
#define CreateSemaphore CreateSemaphoreA
#endif // !UNICODE
```


lpSemaphoreAttributes 是信号量的安全属性。
lInitialCount 是初始化的信号量。
lMaximumCount 是允许信号量增加到最大值。
lpName 是信号量的名称。

```
WINAPI
ReleaseSemaphore(
    __in      HANDLE hSemaphore,
    __in      LONG lReleaseCount,
    __out_opt LPLONG lpPreviousCount
    );
```

hSemaphore 是要增加的信号量句柄。
lReleaseCount 是增加的计数。
lpPreviousCount 是增加前的数值返回。

调用函数的例子如下：

```
#001 //线程运行函数。
#002 //在这里可以使用类里的成员，也可以让派生类实现更强大的功能。
#003 //蔡军生 2007/10/10 QQ:9073204 深圳
#004 DWORD CThreadSemaphore::Run(void)
#005 {
#006  //输出到调试窗口。
#007  ::OutputDebugString(_T("Run()线程函数运行\r\n"));
#008
#009  //
#010  const LONG cMax = 10;
#011   m_hSemaphore = CreateSemaphore(
#012       NULL,   // 缺省的安全属性。
#013       0,   // 初始化为 0 个信号量。
#014       cMax,   // 最大为 10 个信号量。
#015       NULL); // 不命名。
#016
#017  if (m_hSemaphore == NULL)
#018  {
#019       return -1;
#020  }
```

```
#021
#022  //
#023  const int nMaxObjs = 2;
#024  HANDLE hWaitObjects[nMaxObjs] = {m_hEventExit,m_hSemaphore};
#025
#026  //线程循环。
#027  for (;;)
#028  {
#029        DWORD dwRet =
WaitForMultipleObjects(nMaxObjs,hWaitObjects,FALSE,INFINITE);
#030        if (dwRet == WAIT_TIMEOUT)
#031        {
#032            //可以继续运行。
#033            TCHAR chTemp[128];
#034            wsprintf(chTemp,_T("CThreadSemaphore::Run()
ThreadID=%d\r\n"),m_dwThreadID);
#035            ::OutputDebugString(chTemp);
#036
#037            //目前没有做什么事情，就让线程释放一下 CPU。
#038            Sleep(10);
#039        }
#040        else if (dwRet == WAIT_OBJECT_0)
#041        {
#042            //退出线程。
#043            ::OutputDebugString(_T("Run() 退出线程\r\n"));
#044            break;
#045        }
#046        else if (dwRet == WAIT_OBJECT_0+1)
#047        {
#048            //可以继续运行。
#049            TCHAR chTemp[128];
#050            wsprintf(chTemp,_T("CThreadSemaphore::Run()
Semaphore,ThreadID=%d\r\n"),m_dwThreadID);
#051            ::OutputDebugString(chTemp);
#052
#053            //
#054
#055        }
#056        else if (dwRet == WAIT_ABANDONED)
#057        {
#058            //出错。
#059            ::OutputDebugString(_T("Run() 线程出错\r\n"));
#060            return -1;
#061        }
```

```
#062 }
#063
#064 //
#065 if (m_hSemaphore)
#066 {
#067       CloseHandle(m_hSemaphore);
#068        m_hSemaphore = NULL;
#069 }
#070
#071 return 0;
#072 }
#073
```

第 11 行就是创建信号量。
第 29 行等信号量事件和退出事件。

```
#001
#002 //
#003 //增加信号量
#004 //蔡军生 2007/10/10 QQ:9073204 深圳
#005 //
#006 void IncSemaphore(void)
#007 {
#008       if (m_hSemaphore)
#009       {
#010           if (!ReleaseSemaphore(
#011                m_hSemaphore, // 要增加的信号量。
#012               1,        // 增加 1.
#013               NULL) )     // 不想返回前一次信号量。
#014           {
#015
#016           }
#017       }
#018 }
#019
```

Windows API一日一练(48)PostThreadMessage 函数

在写服务器程序里,很多地方都需要使用到线程池。特别现在多处理器的 CPU 越来越普及，使用多个线程池是明显提高服务器程序的性能。在以消息为基础的 Windows 系统里，使用消息来处理是最简单的线程池办法，不但使用起来简单，而且理解起来也很简单的方法。创建多个线程后，就可以根据线程的 ID 来向不同的线程发送消息，每个线程都处理自己的消

息。而发送消息给线程的函数是 PostThreadMessage 函数。下面来演示怎么样使用这个函数。

函数 PostThreadMessage 声明如下：

```
WINUSERAPI
BOOL
WINAPI
PostThreadMessageA(
    __in DWORD idThread,
    __in UINT Msg,
    __in WPARAM wParam,
    __in LPARAM lParam);
WINUSERAPI
BOOL
WINAPI
PostThreadMessageW(
    __in DWORD idThread,
    __in UINT Msg,
    __in WPARAM wParam,
    __in LPARAM lParam);
#ifdef UNICODE
#define PostThreadMessage PostThreadMessageW
#else
#define PostThreadMessage PostThreadMessageA
#endif // !UNICODE
```

idThread 是线程 ID。
Msg 是发送的消息 ID。
wParam 是消息参数。
lParam 是消息参数。

调用函数的例子如下：

```
#001 //线程运行函数。
#002 //在这里可以使用类里的成员，也可以让派生类实现更强大的功能。
#003 //蔡军生 2007/10/11 QQ:9073204 深圳
#004 DWORD CThreadMsg::Run(void)
#005 {
#006  //创建线程消息队列。
#007  MSG msg;
#008  PeekMessage(&msg, NULL, WM_USER, WM_USER+1000,
PM_NOREMOVE);
#009
#010  //
```

```
#011  for (;;)
#012  {
#013        //查找是否有线程消息处理。
#014        BOOL bRes = PeekMessage(&msg, NULL, WM_USER,
WM_USER+1000, PM_REMOVE);
#015        if (bRes)
#016        {
#017            //在这里处理本线程的消息。
#018            ::OutputDebugString(_T("CThreadMsg::Run() 有消息处理
\r\n"));
#019        }
#020        else
#021        {
#022            //等线程退出事件。
#023            DWORD dwRet = WaitForSingleObject(m_hEventExit,0);
#024            if (dwRet == WAIT_TIMEOUT)
#025            {
#026                //目前没有做什么事情，就让线程释放一下 CPU。
#027                Sleep(10);
#028            }
#029            else
#030            {
#031                //退出线程。
#032                ::OutputDebugString(_T("CThreadMsg::Run() 退出线程
\r\n"));
#033                break;
#034            }
#035        }
#036  }
#037
#038  //
#039  return 0;
#040 }
```

上面实现线程的消息处理。

```
#001 class CThreadMsg :
#002  public CThread
#003 {
#004 public:
#005  CThreadMsg(void);
#006  virtual ~CThreadMsg(void);
#007
#008  //发送一条消息给线程处理。
```

```
#009 //蔡军生 2007/10/11 QQ:9073204 深圳
#010 void PostMessage(void)
#011 {
#012       //
#013       ::PostThreadMessage(m_dwThreadID,WM_USER+100,0,0);
#014 }
#015 protected:
#016 virtual DWORD Run(void);
#017 };
```


上面是调用函数 PostThreadMessage 发送消息给线程处理。

Windows API一日一练(49)SetThreadPriority 和 GetThreadPriority 函数

Windows 是抢先式执行任务的操作系统，无论进程还是线程都具有优先级的选择执行方式，这样就可以让用户更加方便处理多任务。比如当你一边听着音乐，一边上网时，这时就可以把音乐的任务执行级别高一点，这样不让音乐听起来断断续续。当你编写网络程序时，一个线程从网络接收数据，一个线程写数据到硬盘，这时也可以把网络接收线程的优先级设置高一点，因为可以把接收到的数据写到内存里去，然后缓存起来再写到硬盘里。还有一种任务，当你写的程序需要在空闲时才去执行，这时就需设置线程的优先级。这样就使用到 SetThreadPriority 和 GetThreadPriority 函数来设置线程的优先级和获取线程的优先级。

函数 SetThreadPriority 和 GetThreadPriority 声明如下：

```
BOOL
WINAPI
SetThreadPriority(
    __in HANDLE hThread,
    __in int nPriority
    );
```

hThread 是线程的句柄。
nPriority 是线程的优先级。

```
WINBASEAPI
int
WINAPI
GetThreadPriority(
    __in HANDLE hThread
    );
```

hThread 是线程的句柄。
返回值是线程的优先级。

调用函数的例子如下：

```
#001 //设置线程的优先级和获取线程的优先级。
#002  //蔡军生 2007/10/11 QQ:9073204 深圳
#003  void ThreadPriority(void)
#004  {
#005      //
#006      ::SetThreadPriority(m_hThread,THREAD_PRIORITY_HIGHEST);
#007
#008      //
#009      if (::GetThreadPriority(m_hThread) == THREAD_PRIORITY_HIGHEST)
#010      {
#011          //
#012          OutputDebugString(_T("THREAD_PRIORITY_HIGHEST\r\n"));
#013      }
#014
#015  }
```


Windows API一日一练(50)SuspendThread 和 ResumeThread 函数

操作系统对线程有几种状态的变化：执行，挂起和恢复执行。
当线程做完任务或者现在想暂停线程运行，就需要使用 SuspendThread 来暂停线程的执行，当然恢复线程的执行就是使用 ResumeThread 函数了。这两个函数使用很简单的，下面就来看看例子是怎么样使用的。

函数 SuspendThread 和 ResumeThread 声明如下：

```
WINBASEAPI
DWORD
WINAPI
SuspendThread(
    __in HANDLE hThread
    );

WINBASEAPI
DWORD
WINAPI
ResumeThread(
    __in HANDLE hThread
    );
```

hThread 是线程的句柄。

调用函数的例子如下：

```
#001  //线程的暂停和恢复。
#002  //蔡军生 2007/10/15 QQ:9073204 深圳
#003  void ThreadSuspendResume(void)
#004  {
#005        ::SuspendThread(m_hThread);
#006
#007        Sleep(10);
#008        ::ResumeThread(m_hThread);
#009  }
#010
```


第 5 行是暂停线程执行。
第 8 行是继续线程执行

Windows API一日一练(51)CreateDirectory 和 RemoveDirectory 函数

在信息爆炸的年代里，把各种信息分类已经是一种非常必要的功能，比如把股票行情数据保存到硬盘里，就需要分开几个目录保存。比如你写 LOG 到硬盘时，也需要分成几个目录来保存，这样让维护人员很好地找到出错的信息，或者有用的信息。像我在开发银行的信用卡系统时，就需要把所有通过网络传送的数据全部生成文本文件保存到当日的目录里，也就是说每天都需要创建一个目录，把所有交易的数据生成 LOG 保存进去。那么面对这样的需求，你是怎么样去创建目录的呢？这就需要使用到 Windows API 函数 CreateDirectory 创建目录，当然目录过多时也需要调用函数 RemoveDirectory 来删除不需要的目录，然而函数 RemoveDirectory 只能删除空的目录，也就是目录下没有文件和子目录才能删除。

函数 CreateDirectory 和 RemoveDirectory 声明如下：

```
WINBASEAPI
BOOL
WINAPI
CreateDirectoryA(
    __in      LPCSTR lpPathName,
    __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes
    );
WINBASEAPI
BOOL
WINAPI
CreateDirectoryW(
    __in      LPCWSTR lpPathName,
    __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes
    );
#ifdef UNICODE
#define CreateDirectory CreateDirectoryW
```

```
#else
#define CreateDirectory CreateDirectoryA
#endif // !UNICODE

WINBASEAPI
BOOL
WINAPI
RemoveDirectoryA(
    __in LPCSTR lpPathName
    );
WINBASEAPI
BOOL
WINAPI
RemoveDirectoryW(
    __in LPCWSTR lpPathName
    );
#ifdef UNICODE
#define RemoveDirectory RemoveDirectoryW
#else
#define RemoveDirectory RemoveDirectoryA
#endif // !UNICODE
```


lpPathName 是目录的路径。
lpSecurityAttributes 是目录的安全属性。
*返回值*是执行是否成功。

调用函数的例子如下：

```
#001 //创建目录。
#002            //蔡军生 2007/10/16 QQ:9073204 深圳
#003            std::wstring strDir(_T("c:\\log"));
#004            if (!CreateDirectory(strDir.c_str(),NULL))
#005            {
#006                OutputDebugString(_T("创建目录不成功
\r\n"));
#007
#008            }
#009            else
#010            {
#011                //删除空目录。
#012                RemoveDirectory(strDir.c_str());
#013            }
#014
```

Windows API一日一练(52)GetCurrentDirectory 和 SetCurrentDirectory 函数

在开发软件里，常常碰到要读取当前目录下的配置参数文件，或者打开当前目录下别的程序来运行，那么就需要获取当前进程的目录位置，这就需要使用函数 GetCurrentDirectory 获取当前进程所有在的目录。同时也可以使用 SetCurrentDirectory 函数来改变进程的当前目录。

函数 GetCurrentDirectory 和 SetCurrentDirectory 声明如下：

```
WINBASEAPI
DWORD
WINAPI
GetCurrentDirectoryA(
    __in DWORD nBufferLength,
    __out_ecount_part_opt(nBufferLength, return + 1) LPSTR lpBuffer
    );
WINBASEAPI
DWORD
WINAPI
GetCurrentDirectoryW(
    __in DWORD nBufferLength,
    __out_ecount_part_opt(nBufferLength, return + 1) LPWSTR lpBuffer
    );
#ifdef UNICODE
#define GetCurrentDirectory GetCurrentDirectoryW
#else
#define GetCurrentDirectory GetCurrentDirectoryA
#endif // !UNICODE

WINBASEAPI
BOOL
WINAPI
SetCurrentDirectoryA(
    __in LPCSTR lpPathName
    );
WINBASEAPI
BOOL
WINAPI
SetCurrentDirectoryW(
    __in LPCWSTR lpPathName
    );
#ifdef UNICODE
#define SetCurrentDirectory SetCurrentDirectoryW
#else
```

```
#define SetCurrentDirectory SetCurrentDirectoryA
#endif // !UNICODE
```

nBufferLength 是缓冲区的大小。
lpBuffer 是接收目录的缓冲区指针。
lpPathName 是设置的目录。

调用函数的例子如下：

```
#001  //获取或者改变当前目录路径。
#002  //蔡军生 2007/10/17 QQ:9073204 深圳
#003  void GetCurDir(void)
#004  {
#005      //
#006      TCHAR szBuf[MAX_PATH];
#007      ZeroMemory(szBuf,MAX_PATH);
#008      if (GetCurrentDirectory(MAX_PATH,szBuf) > 0)
#009      {
#010          //获取进程目录成功。
#011          OutputDebugString(szBuf);
#012      }
#013      else
#014      {
#015          //改变当前目录位置。
#016          SetCurrentDirectory(_T("C:\\"));
#017      }
#018
#019      OutputDebugString(_T("\r\n"));
#020  }
#021
```

Windows API一日一练(53)CreateFile 函数

在软件的需求里，把有用的数据保存起来是非常重要的功能。比如每天的股票行情数据需要保存起来，以便生成 K 线图。比如游戏客户端的 LOG 需要保存起，以便客户端出错时可以把 LOG 发送回来分析它出错的原因。比如银行每天进行交易时，也需要把所有交易的数据保存到文件备份起来，以便进行结算。还有在数据采集领域更是需要保存更多的数据，比如从 DV 里读取视频和语音数据出来，就会生成 12G 的巨型文件。比如读 DVD 光盘里，把光盘做成虚拟光驱也有 9G 大小。因此，创建文件是非常普通的功能，这个肯定是掌握，并且非常会使用的。当然这个 CreateFile 函数不但可以创建文件，还可以打串口、并口、网络、USB 设备等功能。

函数 CreateFile 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
CreateFileA(
    __in     LPCSTR lpFileName,
    __in     DWORD dwDesiredAccess,
    __in     DWORD dwShareMode,
    __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    __in     DWORD dwCreationDisposition,
    __in     DWORD dwFlagsAndAttributes,
    __in_opt HANDLE hTemplateFile
    );
WINBASEAPI
__out
HANDLE
WINAPI
CreateFileW(
    __in     LPCWSTR lpFileName,
    __in     DWORD dwDesiredAccess,
    __in     DWORD dwShareMode,
    __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    __in     DWORD dwCreationDisposition,
    __in     DWORD dwFlagsAndAttributes,
    __in_opt HANDLE hTemplateFile
    );
#ifdef UNICODE
#define CreateFile CreateFileW
#else
#define CreateFile CreateFileA
#endif // !UNICODE
```


lpFileName 是文件或设备的名称。
dwDesiredAccess 是访问属性。
dwShareMode 是共享属性。
lpSecurityAttributes 是安全属性。
dwCreationDisposition 是创建属性。
dwFlagsAndAttributes 是文件标志和属性。
hTemplateFile 是文件模板。

调用函数的例子如下：

```
#001  //创建文件。
#002  //蔡军生 2007/10/18 QQ:9073204  深圳
#003  void CreateFileDemo(void)
```

```
#004 {
#005       //
#006       HANDLE hFile
= ::CreateFile(_T("CreateFileDemo.txt"),     //创建文件的名称。
#007           GENERIC_WRITE,        // 写文件。
#008           0,                    // 不共享读写。
#009           NULL,                 // 缺省安全属性。
#010           CREATE_ALWAYS,        // 如果文件存在，也创建。
#011           FILE_ATTRIBUTE_NORMAL, // 一般的文件。
#012           NULL);                // 模板文件为空。
#013
#014       if (hFile == INVALID_HANDLE_VALUE)
#015       {
#016           //
#017           OutputDebugString(_T("CreateFile fail!\r\n"));
#018       }
#019 }
```


Windows API一日一练(54)WriteFile 和 ReadFile 函数

读写文件是每个 Windows 软件开发人员都需要做的工作。可见这项工作是非常重要的，毕竟各种各样的数据都需要保存起来，以便作各种各样的分析，或者通过网络传送给别人。像大家用 BT 下载的电影，在那个 BT 软件里，就需要不断从网络里接收到数据，然后再把这些数据保存到文件里合适的位置，就可以生成跟发行者那里一样的文件，这样才可以播放出来。又比如我在玩《征途》的游戏里，刚刚打开游戏时，它就不断从服务器上下载更新的文件下来，然后保存到硬盘。WriteFile 函数是用来写数据到文件，ReadFile 函数是从文件里读取数据出来。但这两个函数不但可以读取写磁盘的文件，也可以接收和发送网络的数据，还有读写串口、USB、并口等设备的数据。在读写文件里，首先就是先打开文件，然后判断打开是否成功。在写文件时，同时要注意磁盘的空间是否满等问题。在读取文件时，往往需要读取不同位置的文件，比如要读取一个 4G 的视频文件，就不可能完全把它读取到内存里，因此就需要对文件进行定位读取。

函数 WriteFile 和 ReadFile 声明如下：

```
WINBASEAPI
BOOL
WINAPI
WriteFile(
    __in        HANDLE hFile,
    __in_bcount(nNumberOfBytesToWrite) LPCVOID lpBuffer,
    __in        DWORD nNumberOfBytesToWrite,
    __out_opt   LPDWORD lpNumberOfBytesWritten,
    __inout_opt LPOVERLAPPED lpOverlapped
    );
```

```
WINBASEAPI
BOOL
WINAPI
ReadFile(
    __in        HANDLE hFile,
    __out_bcount_part(nNumberOfBytesToRead, *lpNumberOfBytesRead)
LPVOID lpBuffer,
    __in        DWORD nNumberOfBytesToRead,
    __out_opt   LPDWORD lpNumberOfBytesRead,
    __inout_opt LPOVERLAPPED lpOverlapped
    );
```


hFile 是文件句柄。
lpBuffer 是读写数据缓冲区。
nNumberOfBytesToWrite 是多少数据要写入。
lpNumberOfBytesWritten 是已经写入多少数据。
nNumberOfBytesToRead 是多少数据要读取。
nNumberOfBytesToRead 是已经读取多少数据。
lpOverlapped 是异步读写的结构。

调用函数的例子如下：

```
#001  //创建、写入、读取文件。
#002  //蔡军生 2007/10/21 QQ:9073204 深圳
#003  void CreateFileDemo(void)
#004  {
#005      //
#006      HANDLE hFile = ::CreateFile(_T("CreateFileDemo.txt"),    //创建文
件的名称。
#007          GENERIC_WRITE|GENERIC_READ,         // 写和读文件。
#008          0,                    // 不共享读写。
#009          NULL,                   // 缺省安全属性。
#010          CREATE_ALWAYS,          // 如果文件存在，也创建。
#011          FILE_ATTRIBUTE_NORMAL, // 一般的文件。
#012          NULL);                  // 模板文件为空。
#013
#014      if (hFile == INVALID_HANDLE_VALUE)
#015      {
#016          //
#017          OutputDebugString(_T("CreateFile fail!\r\n"));
#018      }
#019
#020      //往文件里写数据。
```

```
#021        const int BUFSIZE = 4096;
#022        char chBuffer[BUFSIZE];
#023        memcpy(chBuffer,"Test",4);
#024        DWORD dwWritenSize = 0;
#025        BOOL bRet
= ::WriteFile(hFile,chBuffer,4,&dwWritenSize,NULL);
#026        if (bRet)
#027        {
#028            //
#029            OutputDebugString(_T("WriteFile 写文件成功\r\n"));
#030        }
#031
#032        //先把写文件缓冲区的数据强制写入磁盘。
#033        FlushFileBuffers(hFile);
#034
#035        //
#036        //从文件里读取数据。
#037        LONG lDistance = 0;
#038        DWORD dwPtr = SetFilePointer(hFile, lDistance, NULL,
FILE_BEGIN);
#039        if (dwPtr == INVALID_SET_FILE_POINTER)
#040        {
#041            //获取出错码。
#042            DWORD dwError = GetLastError() ;
#043            //处理出错。
#044        }
#045
#046        DWORD dwReadSize = 0;
#047        bRet = ::ReadFile(hFile,chBuffer,4,&dwReadSize,NULL);
#048        if (bRet)
#049        {
#050            //
#051            OutputDebugString(_T("ReadFile 读文件成功\r\n"));
#052        }
#053        else
#054        {
#055            //获取出错码。
#056            DWORD dwError = GetLastError();
#057            //处理出错。
#058            TCHAR chErrorBuf[1024];
#059            wsprintf(chErrorBuf,_T("GetLastError()=%d\r\n"),dwError);
#060            OutputDebugString(chErrorBuf);
#061        }
#062
```

```
#063 }
```

Windows API一日一练(55)FlushFileBuffers 和 SetFilePointer 函数

在 PC 硬件体系结构里，速度最快的存储器是 CPU 里面的寄存器，接着到二级缓存，再到系统 RAM 内存，最后才到硬盘。由于这样的体系结构，就决定了操作系统对文件的操作方式，或者说是最优化的算法。比如操作系统接收到写文件的数据时，就会先把数据保存到 RAM 里，然后在合适的时间或者合适的数据量时再写到硬盘里。但有时候我们希望数据一定要保存到硬盘里，而不是保存在 RAM 里，这时就需要使用函数 FlushFileBuffers 来把 RAM 里的数据保存到硬盘里。文件的结构是一个有序的队列，有头有尾，当读写文件后，就会移动文件里的文件指针。有时候想移动到特定的位置读取数据。比如读取一个 BMP 的文件，它有文件头和数据块组成，就需要先读取文件头，然后根据文件头里指示数据块开始位置去读取图片显示数据，这时就需要使用到 SetFilePointer 函数。

函数 FlushFileBuffers 和 SetFilePointer 声明如下：

```
WINBASEAPI
BOOL
WINAPI
FlushFileBuffers(
    __in HANDLE hFile
    );

WINBASEAPI
DWORD
WINAPI
SetFilePointer(
  __in      HANDLE hFile,
    __in      LONG lDistanceToMove,
    __in_opt PLONG lpDistanceToMoveHigh,
    __in      DWORD dwMoveMethod
    );
```

hFile 是文件句柄。
lDistanceToMove 是文件指针距离头或尾的长度。
lpDistanceToMoveHigh 是文件指针距离头或尾的长度高位长度。
dwMoveMethod 是相对文件头、文件尾或者当前位置的方式。

调用函数的例子如下：

```
#001 //创建、写入、读取文件。
#002 //蔡军生 2007/10/21 QQ:9073204 深圳
#003 void CreateFileDemo(void)
```

```
#004 {
#005     //
#006     HANDLE hFile = ::CreateFile(_T("CreateFileDemo.txt"),     //创建文
件的名称。
#007         GENERIC_WRITE|GENERIC_READ,         // 写和读文件。
#008         0,                  // 不共享读写。
#009         NULL,                 // 缺省安全属性。
#010         CREATE_ALWAYS,          // 如果文件存在，也创建。
#011         FILE_ATTRIBUTE_NORMAL, // 一般的文件。
#012         NULL);                // 模板文件为空。
#013
#014     if (hFile == INVALID_HANDLE_VALUE)
#015     {
#016         //
#017         OutputDebugString(_T("CreateFile fail!\r\n"));
#018     }
#019
#020     //往文件里写数据。
#021     const int BUFSIZE = 4096;
#022     char chBuffer[BUFSIZE];
#023     memcpy(chBuffer,"Test",4);
#024     DWORD dwWritenSize = 0;
#025     BOOL bRet = ::WriteFile(hFile,chBuffer,4,&dwWritenSize,NULL);
#026     if (bRet)
#027     {
#028         //
#029         OutputDebugString(_T("WriteFile 写文件成功\r\n"));
#030     }
#031
#032     //先把写文件缓冲区的数据强制写入磁盘。
#033     FlushFileBuffers(hFile);
#034
#035     //
#036     //从文件里读取数据。
#037     LONG lDistance = 0;
#038     DWORD dwPtr = SetFilePointer(hFile, lDistance, NULL,
FILE_BEGIN);
#039     if (dwPtr == INVALID_SET_FILE_POINTER)
#040     {
#041         //获取出错码。
#042         DWORD dwError = GetLastError() ;
#043         //处理出错。
#044     }
#045
```

```
#046        DWORD dwReadSize = 0;
#047        bRet = ::ReadFile(hFile,chBuffer,4,&dwReadSize,NULL);
#048        if (bRet)
#049        {
#050            //
#051            OutputDebugString(_T("ReadFile 读文件成功\r\n"));
#052        }
#053        else
#054        {
#055            //获取出错码。
#056            DWORD dwError = GetLastError();
#057            //处理出错。
#058            TCHAR chErrorBuf[1024];
#059            wsprintf(chErrorBuf,_T("GetLastError()=%d\r\n"),dwError);
#060            OutputDebugString(chErrorBuf);
#061        }
#062
#063  }
```


Windows API一日一练(56)SetEndOfFile 和 GetFileSizeEx 函数

有一天，我正在开发 BT 软件，它有这样的一个功能，就是先把文件的大小分配好，然后再慢慢地往里面对应的位置写入相应的数据。这样的好处，就是可以先把磁盘空间占用起来，以便后面的下载顺利进行。要实现这个功能，就需要创建一个空的文件，然后把文件指针设置到相应大小的位置，然后再调用函数 SetEndOfFile 来设置文件的结束位置，这样文件就有相应的大小了。在 BT 软件的开发里，也发现目录的处理时需要详细地记录目录里的文件大小，这就需要使用 GetFileSizeEx 函数来获取文件的大小。由于 BT 里的视频文件比较大，有可能几 G 的，一定要使用 GetFileSizeEx 函数来处理，这样就可以获取比较大的文件而不出错。

函数 FlushFileBuffers 和 SetFilePointer 声明如下：

```
WINBASEAPI
BOOL
WINAPI
SetEndOfFile(
    __in HANDLE hFile
    );

BOOL
WINAPI
GetFileSizeEx(
    __in HANDLE hFile,
```

```
    __out PLARGE_INTEGER lpFileSize
    );
```

hFile 是文件句柄。
lpFileSize 是获取文件返回的大小。

调用函数的例子如下：


```
#001  //创建、写入、读取文件。
#002  //蔡军生 2007/10/23 QQ:9073204 深圳
#003  void CreateFileDemo(void)
#004  {
#005      //
#006      HANDLE hFile = ::CreateFile(_T("CreateFileDemo.txt"),    //创建文
件的名称。
#007          GENERIC_WRITE|GENERIC_READ,         // 写和读文件。
#008          0,                    // 不共享读写。
#009          NULL,                   // 缺省安全属性。
#010          CREATE_ALWAYS,          // 如果文件存在，也创建。
#011          FILE_ATTRIBUTE_NORMAL, // 一般的文件。
#012          NULL);                  // 模板文件为空。
#013
#014      if (hFile == INVALID_HANDLE_VALUE)
#015      {
#016          //
#017          OutputDebugString(_T("CreateFile fail!\r\n"));
#018      }
#019
#020      //往文件里写数据。
#021      const int BUFSIZE = 4096;
#022      char chBuffer[BUFSIZE];
#023      memcpy(chBuffer,"Test",4);
#024      DWORD dwWritenSize = 0;
#025      BOOL bRet = ::WriteFile(hFile,chBuffer,4,&dwWritenSize,NULL);
#026      if (bRet)
#027      {
#028          //
#029          OutputDebugString(_T("WriteFile 写文件成功\r\n"));
#030      }
#031
#032      //先把写文件缓冲区的数据强制写入磁盘。
#033      FlushFileBuffers(hFile);
#034
#035      //
#036      //从文件里读取数据。
#037      LONG lDistance = 0;
```

```
#038        DWORD dwPtr = SetFilePointer(hFile, lDistance, NULL,
FILE_BEGIN);
#039        if (dwPtr == INVALID_SET_FILE_POINTER)
#040        {
#041            //获取出错码。
#042            DWORD dwError = GetLastError() ;
#043            //处理出错。
#044        }
#045
#046        DWORD dwReadSize = 0;
#047        bRet = ::ReadFile(hFile,chBuffer,4,&dwReadSize,NULL);
#048        if (bRet)
#049        {
#050            //
#051            OutputDebugString(_T("ReadFile 读文件成功\r\n"));
#052        }
#053        else
#054        {
#055            //获取出错码。
#056            DWORD dwError = GetLastError();
#057            //处理出错。
#058            TCHAR chErrorBuf[1024];
#059            wsprintf(chErrorBuf,_T("GetLastError()=%d\r\n"),dwError);
#060            OutputDebugString(chErrorBuf);
#061        }
#062
#063        //
#064        //
#065        //移动文件指针到新的位置。
#066        lDistance = 3;
#067        dwPtr = SetFilePointer(hFile, lDistance, NULL, FILE_BEGIN);
#068
#069        //设置文件新的结束位置。
#070        ::SetEndOfFile(hFile);
#071
#072        //获取文件的大小。
#073        LARGE_INTEGER liFileSize;
#074        ::GetFileSizeEx(hFile,&liFileSize);
#075
#076        TCHAR chTemp[128];
#077        wsprintf(chTemp,_T("GetFileSizeEx()=%d\r\n"),liFileSize);
#078        OutputDebugString(chTemp);
#079
#080
```

```
#081        //关闭文件。
#082        if (hFile != INVALID_HANDLE_VALUE)
#083        {
#084            //
#085            CloseHandle(hFile);
#086        }
#087
#088  }
```


Windows API一日一练(57)CopyFile 和 MoveFile 函数

在信息的社会里，共享信息是非常重要的。比如你有一个很好的相片，要拷给朋友去分享。又或者你在写一些比较重要的数据，要进行不定时备份时，也就需要拷贝文件到不同的目录里。这样就需要使用到函数 CopyFile 来拷贝文件，它能够把一份文件拷贝多一份出来。我在开发一个数据采集的软件里，由于这个软件是可以不同的用户共同使用，用户跟我说有这样的需求，就是不同的用户的数据移动到不同的目录里，这样方便他们管理数据，也方便他们在上千个文件里找到自己有用的文件，而不会与别人的文件混在一起，这样就需要把采集数据完成后把文件移到相应的目录，这样就需要使用函数 MoveFile 来移动文件。

函数 CopyFile 和 MoveFile 声明如下：

```
WINBASEAPI
BOOL
WINAPI
CopyFileA(
    __in LPCSTR lpExistingFileName,
    __in LPCSTR lpNewFileName,
    __in BOOL bFailIfExists
    );
WINBASEAPI
BOOL
WINAPI
CopyFileW(
    __in LPCWSTR lpExistingFileName,
    __in LPCWSTR lpNewFileName,
    __in BOOL bFailIfExists
    );
#ifdef UNICODE
#define CopyFile CopyFileW
#else
#define CopyFile CopyFileA
#endif // !UNICODE
```

```
WINBASEAPI
BOOL
WINAPI
MoveFileA(
    __in LPCSTR lpExistingFileName,
    __in LPCSTR lpNewFileName
    );
WINBASEAPI
BOOL
WINAPI
MoveFileW(
    __in LPCWSTR lpExistingFileName,
    __in LPCWSTR lpNewFileName
    );
#ifdef UNICODE
#define MoveFile MoveFileW
#else
#define MoveFile MoveFileA
#endif // !UNICODE
```


调用函数的例子如下：

```
#001  //拷贝和移动文件。
#002  //蔡军生 2007/10/24 QQ:9073204 深圳
#003  void CopyAndMoveFile(void)
#004  {
#005      //拷贝文件。
#006      BOOL bRes = ::CopyFile(_T("CreateFileDemo.txt"),
#007          _T("CreateFileDemo_New.txt"),FALSE);
#008      if (bRes)
#009      {
#010          //
#011          OutputDebugString(_T("拷贝文件成功!\r\n"));
#012      }
#013
#014      //移动文件。
#015      bRes = ::MoveFile(_T("CreateFileDemo.txt"),
#016          _T(".\\Debug\\CreateFileDemo.txt"));
#017      if (bRes)
#018      {
#019          //
#020          OutputDebugString(_T("移动文件成功!\r\n"));
#021      }
#022  }
```

#023

Windows API一日一练(58)FindFirstFile 和 FindNextFile 函数

在开发软件的过程里，经常需要维护目录里的数据。比如在开发银行的信用卡系统里，由于每天创建的 LOG 非常多，那么一个很大的硬盘，在 6 个月后，就占用了很多空间。这时就有这样的一个需求，把所有超过 6 个月的 LOG 数据定期删除掉。要实现这个功能就得遍历整个目录，把文件名称和文件创建的时间都超过 6 个月时间的文件删除掉。因此，就需使用到下面的 API 函数 FindFirstFile 和 FindNextFile 来实现这样的功能，FindFirstFile 函数是查找到目录下的第一个文件或目录，FindNextFile 函数是查找下一文件或目录。

函数 FindFirstFile、FindNextFile 和 FindClose 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
FindFirstFileA(
    __in LPCSTR lpFileName,
    __out LPWIN32_FIND_DATAA lpFindFileData
    );
WINBASEAPI
__out
HANDLE
WINAPI
FindFirstFileW(
    __in LPCWSTR lpFileName,
    __out LPWIN32_FIND_DATAW lpFindFileData
    );
#ifdef UNICODE
#define FindFirstFile FindFirstFileW
#else
#define FindFirstFile FindFirstFileA
#endif // !UNICODE

WINBASEAPI
BOOL
WINAPI
FindNextFileA(
    __in HANDLE hFindFile,
    __out LPWIN32_FIND_DATAA lpFindFileData
    );
```

```
WINBASEAPI
BOOL
WINAPI
FindNextFileW(
    __in HANDLE hFindFile,
    __out LPWIN32_FIND_DATAW lpFindFileData
    );
#ifdef UNICODE
#define FindNextFile FindNextFileW
#else
#define FindNextFile FindNextFileA
#endif // !UNICODE

WINBASEAPI
BOOL
WINAPI
FindClose(
    __inout HANDLE hFindFile
    );
```


lpFileName 是目录名称。一般使用通配符。
lpFindFileData 是找到的文件或目录属性。
hFindFile 是下一个文件或目录的句柄。

调用函数的例子如下：

```
#001 #pragma once
#002
#003 //
#004 //遍历一个目录的文件。
#005 //蔡军生 2007/10/25 QQ:9073204 深圳
#006 //
#007 class CFindFile
#008 {
#009 public:
#010
#011  CFindFile(void)
#012  {
#013         m_hFind = INVALID_HANDLE_VALUE;
#014         m_bFound = false;
#015        memset(&m_FindFileData,0,sizeof(m_FindFileData));
#016  }
#017
#018  ~CFindFile(void)
#019  {
```

```
#020        if (m_hFind != INVALID_HANDLE_VALUE)
#021        {
#022            ::FindClose(m_hFind);
#023        }
#024    }
#025
#026    //找到第一个文件。
#027    void First(LPCTSTR lpFileName)
#028    {
#029        m_hFind = ::FindFirstFile(lpFileName,&m_FindFileData);
#030        if (m_hFind != INVALID_HANDLE_VALUE)
#031        {
#032            m_bFound = true;
#033        }
#034        else
#035        {
#036            m_bFound = false;
#037        }
#038    }
#039
#040    //查找一下文件。
#041    void Next(void)
#042    {
#043        m_bFound = FindNextFile(m_hFind, &m_FindFileData) ?
true:false;
#044    }
#045
#046    //是否可以查找一下文件。
#047    bool IsOK(void) const
#048    {
#049        return m_bFound;
#050    }
#051
#052    //返回当前文件的属性。
#053    const WIN32_FIND_DATA& GetCurFile(void)
#054    {
#055        return m_FindFileData;
#056    }
#057
#058 protected:
#059    HANDLE m_hFind; //保存当查找的位置句柄。
#060    bool m_bFound; //当前查找是否成功。
#061    WIN32_FIND_DATA m_FindFileData;//保存当前文件的属性。
#062
```

```
#063 };
#064
```

使用如下：

```
#001 //查找文件。
#002            CFindFile findDemo;
#003            for (findDemo.First(_T(".\\*"));
#004                findDemo.IsOK();
#005                findDemo.Next())
#006            {
#007                //
#008                OutputDebugString(findDemo.GetCurFile().cFileName);
#009                OutputDebugString(_T("\r\n"));
#010            }
#011
```


Windows API一日一练(59)CreateFileMapping 和 MapViewOfFile 函数

在开发软件过程里，也经常碰到进程间共享数据的需求。比如 A 进程创建计算数据，B 进程进行显示数据的图形。这样的开发方式可以把一个大程序分开成独立的小程序，提高软件的成功率，也可以更加适合团队一起开发，加快软件的开发速度。下面就来使用文件映射的方式进行共享数据。先要使用函数CreateFileMapping 来创建一个想共享的文件数据句柄，然后使用 MapViewOfFile 来获取共享的内存地址，然后使用 OpenFileMapping 函数在另一个进程里打开共享文件的名称，这样就可以实现不同的进程共享数据。

函数 CreateFileMapping、MapViewOfFile 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
CreateFileMappingA(
    __in     HANDLE hFile,
    __in_opt LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
    __in     DWORD flProtect,
    __in     DWORD dwMaximumSizeHigh,
    __in     DWORD dwMaximumSizeLow,
    __in_opt LPCSTR lpName
    );
WINBASEAPI
__out
HANDLE
WINAPI
CreateFileMappingW(
```

```
    __in     HANDLE hFile,
    __in_opt LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
    __in     DWORD flProtect,
    __in     DWORD dwMaximumSizeHigh,
    __in     DWORD dwMaximumSizeLow,
    __in_opt LPCWSTR lpName
    );
#ifdef UNICODE
#define CreateFileMapping CreateFileMappingW
#else
#define CreateFileMapping CreateFileMappingA
#endif // !UNICODE

WINBASEAPI
__out
LPVOID
WINAPI
MapViewOfFile(
    __in HANDLE hFileMappingObject,
    __in DWORD dwDesiredAccess,
    __in DWORD dwFileOffsetHigh,
    __in DWORD dwFileOffsetLow,
    __in SIZE_T dwNumberOfBytesToMap
    );
```

hFile 是创建共享文件的句柄。
lpFileMappingAttributes 是文件共享的属性。
flProtect 是当文件映射时读写文件的属性。
dwMaximumSizeHigh 是文件共享的大小高位字节。
dwMaximumSizeLow 是文件共享的大小低位字节。
lpName 是共享文件对象名称。
hFileMappingObject 是共享文件对象。
dwDesiredAccess 是文件共享属性。
dwFileOffsetHigh 是文件共享区的偏移地址。
dwFileOffsetLow 是文件共享区的偏移地址。
dwNumberOfBytesToMap 是共享数据长度。

调用函数的例子如下：

```
#001  //文件共享。
#002  //蔡军生 2007/10/27 QQ:9073204 深圳
#003  void FileMapping(void)
#004  {
#005        //打开共享的文件对象。
#006        m_hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS,
#007            FALSE,_T("TestFileMap"));
```

```
#008        if (m_hMapFile)
#009        {
#010            //显示共享的文件数据。
#011            LPTSTR lpMapAddr =
(LPTSTR)MapViewOfFile(m_hMapFile,FILE_MAP_ALL_ACCESS,
#012                0,0,0);
#013            OutputDebugString(lpMapAddr);
#014        }
#015        else
#016        {
#017            //创建共享文件。
#018            m_hMapFile =
CreateFileMapping( (HANDLE)0xFFFFFFFF,NULL,
#019                PAGE_READWRITE,0,1024,_T("TestFileMap"));
#020
#021            //拷贝数据到共享文件里。
#022            LPTSTR lpMapAddr =
(LPTSTR)MapViewOfFile(m_hMapFile,FILE_MAP_ALL_ACCESS,
#023                0,0,0);
#024            std::wstring strTest(_T("TestFileMap"));
#025            wcscpy(lpMapAddr,strTest.c_str());
#026
#027            FlushViewOfFile(lpMapAddr,strTest.length()+1);
#028        }
#029  }
```


Windows API 一日一练 (60)CreateIoCompletionPort 和 GetQueuedCompletionStatus 函数

在 Windows 系统里，使用完成端口是高性能的方法之一，比如把完成端口使用到线程池和网络服务器里。现在就通过线程池的方法来介绍怎么样使用完成端口，高性能的服务器以后再仔细地介绍怎么样构造它。其实完成端口是一个队列，所有的线程都在等消息出现，如果队列里有消息，就每个线程去获取一个消息执行它。先用函数 CreateIoCompletionPort 来创建一个消息队列，然后使用 GetQueuedCompletionStatus 函数来从队列获取消息，使用函数 PostQueuedCompletionStatus 来向队列里发送消息。通过这三个函数就实现完成端口的消息循环处理。

函数 CreateIoCompletionPort、GetQueuedCompletionStatus、PostQueuedCompletionStatus 声明如下：

```
WINBASEAPI
__out
HANDLE
```

```
WINAPI
CreateIoCompletionPort(
    __in     HANDLE FileHandle,
    __in_opt HANDLE ExistingCompletionPort,
    __in     ULONG_PTR CompletionKey,
    __in     DWORD NumberOfConcurrentThreads
    );
WINBASEAPI
BOOL
WINAPI
GetQueuedCompletionStatus(
    __in HANDLE CompletionPort,
    __out LPDWORD lpNumberOfBytesTransferred,
    __out PULONG_PTR lpCompletionKey,
    __out LPOVERLAPPED *lpOverlapped,
    __in DWORD dwMilliseconds
    );

WINBASEAPI
BOOL
WINAPI
PostQueuedCompletionStatus(
    __in     HANDLE CompletionPort,
    __in     DWORD dwNumberOfBytesTransferred,
    __in     ULONG_PTR dwCompletionKey,
    __in_opt LPOVERLAPPED lpOverlapped
    );
```

FileHandle 是关联的文件句柄。
ExistingCompletionPort 是已经存在的完成端口。
CompletionKey 是传送给处理函数的参数。
NumberOfConcurrentThreads 是有多少个线程在访问这个消息队列。
CompletionPort 是已经存在的完成端口。
lpCompletionKey 是传送给处理函数的参数。
lpOverlapped 是传送给处理函数的参数。
dwMilliseconds 是等待时间。
dwNumberOfBytesTransferred 是传送了多少个字节。

调用函数的例子如下：

```
#001 #pragma once
#002
#003 #include "Thread.h"
#004
#005
#006 //使用 IOCP 实现线程池。
```

```
#007 //蔡军生 2007/10/29 QQ:9073204 深圳
#008 class CThreadPools
#009 {
#010 public:
#011
#012  CThreadPools(void)
#013  {
#014         m_nThreadCount = 2;
#015   }
#016
#017  ~CThreadPools(void)
#018  {
#019  }
#020
#021  bool Init(void)
#022  {
#023        //创建一个 IOCP。
#024        m_hQueue = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, m_nThreadCount);
#025        if (m_hQueue == NULL)
#026        {
#027             //创建 IOCP 失败。
#028             return false;
#029        }
#030  }
#031
#032  int GetThreadCount(void) const
#033  {
#034        return m_nThreadCount;
#035  }
#036
#037  //线程池处理的内容。
#038  DWORD Run(void)
#039  {
#040        DWORD dwBytesTransfered;
#041        ULONG_PTR dwCompletionKey;
#042
#043        OVERLAPPED* pOverlapped;
#044
#045        //等一个 IOCP 的消息。
#046        while (GetQueuedCompletionStatus(m_hQueue, &dwBytesTransfered, &dwCompletionKey, &pOverlapped, INFINITE))
#047        {
```

```
#048            if (pOverlapped == ((OVERLAPPED*) ((__int64) -1)) )
#049            {
#050                //退出。
#051                OutputDebugString(_T("退出 \r\n"));
#052                break;
#053            }
#054            else
#055
#056            {
#057                WPARAM request = (WPARAM) dwCompletionKey;
#058
#059                //处理消息。
#060                OutputDebugString(_T("GetQueuedCompletionStatus
\r\n"));
#061            }
#062        }
#063
#064        return 0;
#065    }
#066
#067    //发送处理的消息。
#068    bool QueueRequest(WPARAM wParam)
#069    {
#070        //往 IOCP 里发送一条消息。
#071        if (!PostQueuedCompletionStatus(m_hQueue, 0,
(ULONG_PTR) wParam, NULL))
#072        {
#073            return false;
#074        }
#075
#076        return true;
#077    }
#078
#079    //关闭所有线程。
#080    void Close(void)
#081    {
#082        for (int i = 0; i < m_nThreadCount; i++)
#083        {
#084            PostQueuedCompletionStatus(m_hQueue, 0, 0,
(OVERLAPPED*) ((__int64) -1) );
#085        }
#086    }
#087
#088 protected:
```

```
#089  //接收消息处理的队列。
#090  HANDLE m_hQueue;
#091
#092  //线程个数。
#093  int m_nThreadCount;
#094 };
#095
#096 ////////////////////////////////////////////////////////////////////////////
#097 class CThreads :
#098  public CThread
#099 {
#100 public:
#101  CThreads(CThreadPools* pPool)
#102  {
#103          m_pPool = pPool;
#104  }
#105  virtual ~CThreads(void)
#106  {
#107
#108  }
#109
#110
#111 protected:
#112  //
#113  //线程运行函数。
#114  //在这里可以使用类里的成员，也可以让派生类实现更强大的功能。
#115  //蔡军生 2007/10/29
#116  virtual DWORD Run(void)
#117  {
#118        //
#119        if (m_pPool)
#120        {
#121             return m_pPool->Run();
#122        }
#123
#124        return -1;
#125  }
#126
#127 protected:
#128  CThreadPools* m_pPool;
#129
#130 };
```

Windows API一日一练(61)GetDriveType 函数

经常碰到这样的需求，比如你需要保存一个文件到一个目录里去，这个目录或许是用户指定的目录，那么你就需要确保这个目录是否有效的，这样就需要去测试这个目录是否允许写文件？这个目录是否存在？这个目录是否可写的？或者这个目录是否网络上的目录？要完成这个任务，就得使用函数 GetDriveType 来完成。GetDriveType 函数可以获取目录和盘号的属性。

函数 GetDriveType 声明如下：

```
WINBASEAPI
UINT
WINAPI
GetDriveTypeA(
    __in_opt LPCSTR lpRootPathName
    );
WINBASEAPI
UINT
WINAPI
GetDriveTypeW(
    __in_opt LPCWSTR lpRootPathName
    );
#ifdef UNICODE
#define GetDriveType GetDriveTypeW
#else
#define GetDriveType GetDriveTypeA
#endif // !UNICODE
```

lpRootPathName 是目录或盘号的名称。
返回值是目录的属性，有如下值：

```
DRIVE_UNKNOWN
DRIVE_NO_ROOT_DIR
DRIVE_REMOVABLE
DRIVE_FIXED
DRIVE_REMOTE
DRIVE_CDROM
DRIVE_RAMDISK
```

调用函数的例子如下：

```
#001 //获取目录或磁盘的属性。
#002  //蔡军生 2007/10/30 QQ:9073204 深圳
#003  void Disk(void)
#004  {
#005        //获取 C:目录的属性。
```

```
#006    std::wstring strTest(_T("C:\\WINDOWS\\"));
#007    UINT nRes = ::GetDriveType(strTest.c_str());
#008    switch(nRes)
#009    {
#010    case DRIVE_UNKNOWN:
#011        OutputDebugString(_T("DRIVE_UNKNOWN\r\n"));
#012        break;
#013    case DRIVE_NO_ROOT_DIR:
#014        OutputDebugString(_T("DRIVE_NO_ROOT_DIR\r\n"));
#015        break;
#016    case DRIVE_REMOVABLE:
#017        OutputDebugString(_T("DRIVE_REMOVABLE\r\n"));
#018      break;
#019    case DRIVE_FIXED:
#020        OutputDebugString(_T("DRIVE_FIXED\r\n"));
#021      break;
#022    case DRIVE_REMOTE:
#023        OutputDebugString(_T("DRIVE_REMOTE\r\n"));
#024        break;
#025    case DRIVE_CDROM:
#026        OutputDebugString(_T("DRIVE_CDROM\r\n"));
#027        break;
#028    case DRIVE_RAMDISK:
#029        OutputDebugString(_T("DRIVE_RAMDISK\r\n"));
#030        break;
#031    default:
#032      break;
#033    }
#034
#035    //判断盘号是否光驱。
#036    strTest = _T("d:\\");
#037    nRes = ::GetDriveType(strTest.c_str());
#038    switch(nRes)
#039    {
#040    case DRIVE_UNKNOWN:
#041        OutputDebugString(_T("DRIVE_UNKNOWN\r\n"));
#042        break;
#043    case DRIVE_NO_ROOT_DIR:
#044        OutputDebugString(_T("DRIVE_NO_ROOT_DIR\r\n"));
#045        break;
#046    case DRIVE_REMOVABLE:
#047        OutputDebugString(_T("DRIVE_REMOVABLE\r\n"));
#048        break;
#049    case DRIVE_FIXED:
```

```
#050            OutputDebugString(_T("DRIVE_FIXED\r\n"));
#051            break;
#052        case DRIVE_REMOTE:
#053            OutputDebugString(_T("DRIVE_REMOTE\r\n"));
#054            break;
#055        case DRIVE_CDROM:
#056            OutputDebugString(_T("DRIVE_CDROM\r\n"));
#057            break;
#058        case DRIVE_RAMDISK:
#059            OutputDebugString(_T("DRIVE_RAMDISK\r\n"));
#060            break;
#061        default:
#062            break;
#063        }
#064  }
```


Windows API一日一练(62)GetDiskFreeSpaceEx 函数

有一次客户给我打来了投诉电话，说我的软件太不好用了，导致他们丢失了很多数据。后来我仔细地查看 LOG，分析出来的原因，其实是很简单的，就是磁盘的空间不够了。我给客户说他们的电脑磁盘空间不够了导致出错的问题，但客户反问我为什么不提示磁盘空间不足。是啊，为什么不提示磁盘的空间不足呢？为了解决这个需求，就需要使用到这个函数 GetDiskFreeSpaceEx。

函数 GetDiskFreeSpaceEx 声明如下：

```
WINBASEAPI
BOOL
WINAPI
GetDiskFreeSpaceExA(
    __in_opt LPCSTR lpDirectoryName,
    __out_opt PULARGE_INTEGER lpFreeBytesAvailableToCaller,
    __out_opt PULARGE_INTEGER lpTotalNumberOfBytes,
    __out_opt PULARGE_INTEGER lpTotalNumberOfFreeBytes
    );
WINBASEAPI
BOOL
WINAPI
GetDiskFreeSpaceExW(
    __in_opt LPCWSTR lpDirectoryName,
    __out_opt PULARGE_INTEGER lpFreeBytesAvailableToCaller,
    __out_opt PULARGE_INTEGER lpTotalNumberOfBytes,
    __out_opt PULARGE_INTEGER lpTotalNumberOfFreeBytes
```

```
    );
#ifdef UNICODE
#define GetDiskFreeSpaceEx GetDiskFreeSpaceExW
#else
#define GetDiskFreeSpaceEx GetDiskFreeSpaceExA
#endif // !UNICODE
```


lpDirectoryName 是驱动器的名称。
lpFreeBytesAvailableToCaller 是用户可用的磁盘空间。
lpTotalNumberOfBytes 是磁盘总共的空间。
lpTotalNumberOfFreeBytes 是磁盘空闲的空间。以上都是字节为单位。

调用函数的例子如下：

```
#001  //获取磁盘剩余空间。
#002  //蔡军生 2007/11/01 QQ:9073204 深圳
#003  void DiskFree(void)
#004  {
#005      //
#006      ULARGE_INTEGER nFreeBytesAvailable;
#007      ULARGE_INTEGER nTotalNumberOfBytes;
#008      ULARGE_INTEGER nTotalNumberOfFreeBytes;
#009      //
#010      if (GetDiskFreeSpaceEx(_T("C:"),
#011          &nFreeBytesAvailable,
#012          &nTotalNumberOfBytes,
#013          &nTotalNumberOfFreeBytes))
#014      {
#015          TCHAR chBuf[256];
#016          wsprintf(chBuf,_T("Av=%I64d,Total=%I64d,Free=%I64d\r\n
"),
#017              nFreeBytesAvailable,
#018              nTotalNumberOfBytes,
#019              nTotalNumberOfFreeBytes);
#020          OutputDebugString(chBuf);
#021      }
#022  }
```

Windows API一日一练(63)RegOpenKeyEx 和 RegCreateKeyEx 函数

由于电脑经常会关闭，或者应用程序也会经常关闭，但有一些参数是经常需要保存。比如当你打开程序，并设置了窗口的大小，想每次打开时都设置窗口为上次打开的大小。这样就需要保存窗口的大小，那么窗口大小的参数保存到那里呢？其实在 Windows 里最方便的做法，就是保存到注册表里。又比如游戏登录时，总是想保存最后一个登录的用户，那么也需要保

存这个用户到注册表里。其实注册表是 Windows 保存系统配置的数据库，比如不同的语言设置，不同的时区设置，不同的用户登录，不同的权限等等。下面就来学习怎么样使用函数 RegOpenKeyEx 来打开注册表里的键和用函数 RegCreateKeyEx 来创建新的键。

函数 RegOpenKeyEx 和 RegCreateKeyEx 声明如下：


```
WINADVAPI
LONG
APIENTRY
RegOpenKeyExA (
    __in HKEY hKey,
    __in_opt LPCSTR lpSubKey,
    __reserved DWORD ulOptions,
    __in REGSAM samDesired,
    __out PHKEY phkResult
    );
WINADVAPI
LONG
APIENTRY
RegOpenKeyExW (
    __in HKEY hKey,
    __in_opt LPCWSTR lpSubKey,
    __reserved DWORD ulOptions,
    __in REGSAM samDesired,
    __out PHKEY phkResult
    );
#ifdef UNICODE
#define RegOpenKeyEx RegOpenKeyExW
#else
#define RegOpenKeyEx RegOpenKeyExA
#endif // !UNICODE


WINADVAPI
LONG
APIENTRY
RegCreateKeyExA (
    __in HKEY hKey,
    __in LPCSTR lpSubKey,
    __reserved DWORD Reserved,
    __in_opt LPSTR lpClass,
    __in DWORD dwOptions,
    __in REGSAM samDesired,
    __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes,
```

```
    __out PHKEY phkResult,
    __out_opt LPDWORD lpdwDisposition
    );
WINADVAPI
LONG
APIENTRY
RegCreateKeyExW (
    __in HKEY hKey,
    __in LPCWSTR lpSubKey,
    __reserved DWORD Reserved,
    __in_opt LPWSTR lpClass,
    __in DWORD dwOptions,
    __in REGSAM samDesired,
    __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    __out PHKEY phkResult,
    __out_opt LPDWORD lpdwDisposition
    );
#ifdef UNICODE
#define RegCreateKeyEx RegCreateKeyExW
#else
#define RegCreateKeyEx RegCreateKeyExA
#endif // !UNICODE
```


hKey 是主键。
lpSubKey 是子键。
ulOptions 是选项。
samDesired 是键的操作。
phkResult 是打开的键返回。
lpClass 是新键值。

调用函数的例子如下：

```
#001 //打注册表。HKEY_CURRENT_USER\"Software"\"Wincpp"\"testreg"
#002  //蔡军生 2007/11/02 QQ:9073204 深圳
#003  HKEY GetAppRegistryKey(void)
#004  {
#005        HKEY hAppKey = NULL;
#006        HKEY hSoftKey = NULL;
#007        HKEY hCompanyKey = NULL;
#008
#009        //打开 HKEY_CURRENT_USER\"Software"。
#010        if (RegOpenKeyEx(HKEY_CURRENT_USER, _T("software"),
0, KEY_WRITE|KEY_READ,
#011              &hSoftKey) == ERROR_SUCCESS)
#012        {
```

```
#013            DWORD dw;
#014            //创建并打开 HKEY_CURRENT_USER\"Software"\"Wincpp"
#015            if (RegCreateKeyEx(hSoftKey, _T("Wincpp"), 0,
REG_NONE,
#016                REG_OPTION_NON_VOLATILE,
KEY_WRITE|KEY_READ, NULL,
#017                &hCompanyKey, &dw) == ERROR_SUCCESS)
#018            {
#019                //创建并打开
HKEY_CURRENT_USER\"Software"\"Wincpp"\"testreg"
#020                RegCreateKeyEx(hCompanyKey, _T("testreg"), 0,
REG_NONE,
#021                    REG_OPTION_NON_VOLATILE,
KEY_WRITE|KEY_READ, NULL,
#022                    &hAppKey, &dw);
#023            }
#024        }
#025
#026        //关闭打开的键值。
#027        if (hSoftKey != NULL)
#028        {
#029            RegCloseKey(hSoftKey);
#030        }
#031
#032        if (hCompanyKey != NULL)
#033        {
#034            RegCloseKey(hCompanyKey);
#035        }
#036
#037        return hAppKey;
#038  }
```


Windows API一日一练(64) RegSetValueEx 和 RegDeleteValue 函数

上一次说到怎么创建注册表的键，但接着下来的问题就是怎么样保存数据到注册表里。注册表使用树形的方式管理数据，所以它的扩展和访问都是比较灵活的。不过注册表是系统重要信息库，每当 Windows 系统加载时，都首先从硬盘里读取它出来，才知道每台电脑所有硬件配置信息，然后再加载不同的驱动程序。因此，注册表作为系统重要的文件，不要往里面写超过 2K 的数据大小，这样可以提高系统的速度。下面就来介绍一下怎么样保存一个字符串的键值。它需要使用 RegSetValueEx 函数来设置键值和使用 RegDeleteValue 函数来删除原来的键值。

函数 RegSetValueEx 和 RegDeleteValue 声明如下：

```
WINADVAPI
LONG
APIENTRY
RegSetValueExA (
    __in HKEY hKey,
    __in_opt LPCSTR lpValueName,
    __reserved DWORD Reserved,
    __in DWORD dwType,
    __in_bcount_opt(cbData) CONST BYTE* lpData,
    __in DWORD cbData
    );
WINADVAPI
LONG
APIENTRY
RegSetValueExW (
    __in HKEY hKey,
    __in_opt LPCWSTR lpValueName,
    __reserved DWORD Reserved,
    __in DWORD dwType,
    __in_bcount_opt(cbData) CONST BYTE* lpData,
    __in DWORD cbData
    );
#ifdef UNICODE
#define RegSetValueEx RegSetValueExW
#else
#define RegSetValueEx RegSetValueExA
#endif // !UNICODE

WINADVAPI
LONG
APIENTRY
RegDeleteValueA (
    __in HKEY hKey,
    __in_opt LPCSTR lpValueName
    );
WINADVAPI
LONG
APIENTRY
RegDeleteValueW (
    __in HKEY hKey,
    __in_opt LPCWSTR lpValueName
    );
#ifdef UNICODE
```

```
#define RegDeleteValue RegDeleteValueW
#else
#define RegDeleteValue RegDeleteValueA
#endif // !UNICODE
```

hKey 是主键。
lpValueName 是键名称。
dwType 是键值类型。
lpData 是键的数据。
cbData 是键值的大小。

调用函数的例子如下：

```
#001  //打注册表。HKEY_CURRENT_USER\"Software"\"Wincpp"\"testreg"
#002  // \"Windows"\\"winsize" = "800*600"
#003  //蔡军生 2007/11/04 QQ:9073204 深圳
#004  BOOL WriteProfileString(LPCTSTR lpszSection, LPCTSTR lpszEntry,
#005        LPCTSTR lpszValue)
#006  {
#007        //
#008        LONG lResult;
#009        if (lpszEntry == NULL) //删除整键。
#010        {
#011            HKEY hAppKey = GetAppRegistryKey();
#012            if (hAppKey == NULL)
#013            {
#014                return FALSE;
#015            }
#016
#017            lResult = ::RegDeleteKey(hAppKey, lpszSection);
#018            RegCloseKey(hAppKey);
#019        }
#020        else if (lpszValue == NULL)
#021        {
#022            //删除键值。
#023            HKEY hAppKey = GetAppRegistryKey();
#024            if (hAppKey == NULL)
#025            {
#026                return FALSE;
#027            }
#028
#029            HKEY hSecKey = NULL;
#030            DWORD dw;
#031            RegCreateKeyEx(hAppKey, lpszSection, 0, REG_NONE,
```

```
#032                REG_OPTION_NON_VOLATILE, KEY_WRITE|KEY_READ,
NULL,
#033                &hSecKey, &dw);
#034            RegCloseKey(hAppKey);
#035
#036            if (hSecKey == NULL)
#037            {
#038                return FALSE;
#039            }
#040
#041            //
#042            lResult = ::RegDeleteValue(hSecKey,
(LPTSTR)lpszEntry);
#043            RegCloseKey(hSecKey);
#044        }
#045        else
#046        {
#047            //设置键值。
#048            HKEY hAppKey = GetAppRegistryKey();
#049            if (hAppKey == NULL)
#050            {
#051                return FALSE;
#052            }
#053
#054            HKEY hSecKey = NULL;
#055            DWORD dw;
#056            //创建子键。
#057            RegCreateKeyEx(hAppKey, lpszSection, 0, REG_NONE,
#058                REG_OPTION_NON_VOLATILE, KEY_WRITE|KEY_READ,
NULL,
#059                &hSecKey, &dw);
#060            RegCloseKey(hAppKey);
#061
#062            if (hSecKey == NULL)
#063            {
#064                return FALSE;
#065            }
#066
#067            //设置子键中的项值。
#068            lResult = RegSetValueEx(hSecKey, lpszEntry, NULL,
REG_SZ,
#069                (LPBYTE)lpszValue,
(lstrlen(lpszValue)+1)*sizeof(TCHAR));
#070            RegCloseKey(hSecKey);
```

```
#071        }
#072        return lResult == ERROR_SUCCESS;
#073
#074  }
```

Windows API一日一练(65)RegQueryValueEx 函数

上一次介绍怎么样保存数据到注册表里，这次就需要从注册表里读取数据出来了。在这个例子里是读取字符串数据出来，主要调用函数 RegQueryValueEx 来实现。下面的例子里就是先查询键值的长度，然后再读取内容出来。

函数 RegQueryValueEx 声明如下：

```
WINADVAPI
LONG
APIENTRY
RegQueryValueExA (
    __in HKEY hKey,
    __in_opt LPCSTR lpValueName,
    __reserved LPDWORD lpReserved,
    __out_opt LPDWORD lpType,
    __out_bcount_opt(*lpcbData) LPBYTE lpData,
    __inout_opt LPDWORD lpcbData
    );
WINADVAPI
LONG
APIENTRY
RegQueryValueExW (
    __in HKEY hKey,
    __in_opt LPCWSTR lpValueName,
    __reserved LPDWORD lpReserved,
    __out_opt LPDWORD lpType,
    __out_bcount_opt(*lpcbData) LPBYTE lpData,
    __inout_opt LPDWORD lpcbData
    );
#ifdef UNICODE
#define RegQueryValueEx RegQueryValueExW
#else
#define RegQueryValueEx RegQueryValueExA
#endif // !UNICODE
```

hKey 是主键。
lpValueName 是键值名称。

lpType 是类型。
lpData 是读出来数据保存地方。
lpcbData 是读取数据多少。

调用函数的例子如下：

```
#001 //打注册表返回值。HKEY_CURRENT_USER\"Software"\"Wincpp"\"testreg"
#002  // \"Windows"\\"winsize" = "800*600"
#003  //蔡军生 2007/11/05 QQ:9073204 深圳
#004  std::wstring GetProfileString(LPCTSTR lpszSection, LPCTSTR lpszEntry,
#005        LPCTSTR lpszDefault)
#006  {
#007        //打开应用程序键。
#008        HKEY hAppKey = GetAppRegistryKey();
#009        if (hAppKey == NULL)
#010        {
#011            return lpszDefault;
#012        }
#013
#014        HKEY hSecKey = NULL;
#015        DWORD dw;
#016
#017        //打开子键。
#018        RegCreateKeyEx(hAppKey, lpszSection, 0, REG_NONE,
#019            REG_OPTION_NON_VOLATILE, KEY_WRITE|KEY_READ, NULL,
#020            &hSecKey, &dw);
#021        RegCloseKey(hAppKey);
#022
#023        if (hSecKey == NULL)
#024        {
#025            return lpszDefault;
#026        }
#027
#028        //查询键值。
#029        std::wstring strValue;
#030        DWORD dwType=REG_NONE;
#031        DWORD dwCount=0;
#032
#033        //先查询键值的长度。
#034        LONG lResult = RegQueryValueEx(hSecKey, (LPTSTR)lpszEntry,
NULL, &dwType,
#035            NULL, &dwCount);
#036        if (lResult == ERROR_SUCCESS)
#037        {
#038            strValue.resize(dwCount);
```

```
#039
#040            //查询键值。
#041            lResult = RegQueryValueEx(hSecKey, (LPTSTR)lpszEntry,
NULL, &dwType,
#042                (LPBYTE)strValue.data(), &dwCount);
#043
#044        }
#045
#046        RegCloseKey(hSecKey);
#047        if (lResult == ERROR_SUCCESS)
#048        {
#049            return strValue;
#050        }
#051
#052        return lpszDefault;
#053  }
```


Windows API一日一练(66)CreateWaitableTimer和SetWaitableTimer函数

用户感觉到软件的好用，就是可以定时地做一些工作，而不需要人参与进去。比如每天定时地升级病毒库，定时地下载电影，定时地更新游戏里的人物。要想实现这些功能，就可以使用定时器的 API 函数 CreateWaitableTimer 和 SetWaitableTimer 来实现了，这对 API 函数创建的时钟是比较精确的，可以达到 100 倍的 10 亿分之一秒。

函数 CreateWaitableTimer 和 SetWaitableTimer 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
CreateWaitableTimerA(
    __in_opt LPSECURITY_ATTRIBUTES lpTimerAttributes,
    __in     BOOL bManualReset,
    __in_opt LPCSTR lpTimerName
    );
WINBASEAPI
__out
HANDLE
WINAPI
CreateWaitableTimerW(
    __in_opt LPSECURITY_ATTRIBUTES lpTimerAttributes,
    __in     BOOL bManualReset,
    __in_opt LPCWSTR lpTimerName
```

```
    );
#ifdef UNICODE
#define CreateWaitableTimer CreateWaitableTimerW
#else
#define CreateWaitableTimer CreateWaitableTimerA
#endif // !UNICODE
```



```
WINBASEAPI
BOOL
WINAPI
SetWaitableTimer(
    __in      HANDLE hTimer,
    __in      const LARGE_INTEGER *lpDueTime,
    __in      LONG lPeriod,
    __in_opt PTIMERAPCROUTINE pfnCompletionRoutine,
    __in_opt LPVOID lpArgToCompletionRoutine,
    __in      BOOL fResume
    );
```

lpTimerAttributes 是设置定时器的属性。
bManualReset 是是否手动复位。
lpTimerName 是定时器的名称。
hTimer 是定时器的句柄。
lpDueTime 是设置定时器时间间隔，当设置为正值是绝对时间；当设置为负数是相对时间。
lPeriod 是周期。
pfnCompletionRoutine 是设置回调函数。
lpArgToCompletionRoutine 是传送给回调函数的参数。
fResume 是设置系统是否自动恢复。

调用函数的例子如下：

```
#001 //创建定时器
#002  //蔡军生 2007/11/06 QQ:9073204  深圳
#003  int CreateTestTimer(void)
#004  {
#005        HANDLE hTimer = NULL;
#006        LARGE_INTEGER liDueTime;
#007
#008        //设置相对时间为 10 秒。
#009        liDueTime.QuadPart = -100000000;
#010
#011        //创建定时器。
```

```
#012        hTimer = CreateWaitableTimer(NULL, TRUE,
_T("TestWaitableTimer"));
#013        if (!hTimer)
#014        {
#015            return 1;
#016        }
#017
#018        OutputDebugString(_T("10 秒定时器\r\n"));
#019
#020        // 设置 10 秒钟。
#021        if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL,
0))
#022        {
#023            //
#024            CloseHandle(hTimer);
#025            return 2;
#026        }
#027
#028        //等定时器有信号。
#029        if (WaitForSingleObject(hTimer, INFINITE) != WAIT_OBJECT_0)
#030        {
#031            OutputDebugString(_T("10 秒定时器出错了\r\n"));
#032            //
#033            CloseHandle(hTimer);
#034            return 3;
#035        }
#036        else
#037        {
#038            //10 秒钟到达。
#039            OutputDebugString(_T("10 秒定时器到了\r\n"));
#040        }
#041
#042        //
#043        CloseHandle(hTimer);
#044        return 0;
#045  }
```


Windows API一日一练(67)SetTimer和KillTimer函数

在前面介绍了一对定时器的 API 函数使用，现在又介绍另外一对 API 函数的使用。它使用起来比前的函数要简单一些，但它一般是使用到有窗口的程序里，并且它的精度也没有前面的 API 函数高，对于一些要求不高的场合还是非常合适的。它是采用消息通知的方式，每当定时到了就会收到一条消息。

函数 SetTimer 和 KillTimer 声明如下：

```
WINAPI
SetTimer(
    __in_opt HWND hWnd,
    __in UINT_PTR nIDEvent,
    __in UINT uElapse,
    __in_opt TIMERPROC lpTimerFunc);

WINUSERAPI
BOOL
WINAPI
KillTimer(
    __in_opt HWND hWnd,
    __in UINT_PTR uIDEvent);
```


hWnd 是窗口接收定时器的句柄。
nIDEvent 是定时器事件标识号。
uElapse 是定时器的毫秒值。
lpTimerFunc 是定时到达回调函数。

调用函数的例子如下：

```
#001 //设置定时器。
#002          ::SetTimer(m_hWnd,          //指向窗口的句柄。
#003              IDT_TIMER1,         // 定时器标识。
#004              1000,               // 10 秒
#005              (TIMERPROC) NULL);    // 不使用回调函数。
```

接收 **WM_TIMER** 消息并关闭定时器：

```
#001 case WM_TIMER:
#002        {
#003            if (IDT_TIMER1 == wParam)
#004            {
#005                OutputDebugString(_T("定时器测试消息关闭\r\n"));
#006                ::KillTimer(m_hWnd,IDT_TIMER1);
#007            }
#008
#009        }
#010        break;
```

Windows API一日一练(68)QueryPerformanceCounter 函数

精确的时间计时，有时候是非常必要的。比如播放多媒体时视频与音频的时间同步，还有在测试代码的性能时，也需要使用到非常精确的时间计时。还有测试硬件的性能时，也需要精确的时间计时。这时就需要使用 QueryPerformanceCounter 来查询定时器的计数值，如果硬件里有定时器，它就会启动这个定时器，并且不断获取定时器的值，这样的定时器精度，就跟硬件时钟的晶振一样精确的。

函数 QueryPerformanceCounter 和 QueryPerformanceFrequency 声明如下：

```
WINBASEAPI
BOOL
WINAPI
QueryPerformanceCounter(
    __out LARGE_INTEGER *lpPerformanceCount
    );

WINBASEAPI
BOOL
WINAPI
QueryPerformanceFrequency(
    __out LARGE_INTEGER *lpFrequency
    );
```

lpPerformanceCount 是返回定时器当前计数值。
QueryPerformanceFrequency 是返回定时器的频率。

调用函数的例子如下：

```
#001 //精确时钟查询。
#002  //蔡军生 2007/11/08 QQ:9073204 深圳
#003  void TestHighTimer(void)
#004  {
#005        //
#006        LARGE_INTEGER nFreq;
#007        LARGE_INTEGER nLastTime1;
#008        LARGE_INTEGER nLastTime2;
#009
#010        //获取是否支持精确定时器。
#011        if (QueryPerformanceFrequency(&nFreq))
#012        {
#013            //
#014            const int nBufSize = 256;
#015            TCHAR chBuf[nBufSize];
#016
#017            //显示定时器的频率。
#018            wsprintf(chBuf,_T("LastTime=%I64d\r\n"),nFreq);
```

```
#019             OutputDebugString(chBuf);
#020
#021             //获取定时器的值。
#022             QueryPerformanceCounter(&nLastTime1);
#023             wsprintf(chBuf,_T("LastTime=%I64d\r\n"),nLastTime1);
#024             OutputDebugString(chBuf);
#025
#026             Sleep(0);
#027
#028             //获取定时器的值。
#029             QueryPerformanceCounter(&nLastTime2);
#030             wsprintf(chBuf,_T("LastTime=%I64d\r\n"),nLastTime2);
#031             OutputDebugString(chBuf);
#032
#033
#034             //计算时间是花费多少秒。
#035             float fInterval = nLastTime2.QuadPart - nLastTime1.QuadPart;
#036             swprintf(chBuf,nBufSize,_T("花
费:%f\r\n"),fInterval/(float)nFreq.QuadPart);
#037             OutputDebugString(chBuf);
#038         }
#039
#040  }
```


Windows API一日一练(69)GetTickCount 函数

时间计时，也不是越精确越好，有时只需要有一个计时就行了。这样就可以使用毫秒级别的计时函数 GetTickCount。这个函数是记录了系统启动以来的时间毫秒，当超过 49.7 天，这个值变为从 0 开始，也就是说 49.7 天是一个周期。当不同的两次函数调时，就返回两次时间差值。

函数 GetTickCount 声明如下：

```
WINBASEAPI
DWORD
WINAPI
GetTickCount(
    VOID
    );
```

调用函数的例子如下：

```
#001 //一般的时钟计时。
#002  //蔡军生 2007/11/09 QQ:9073204  深圳
#003  void TestTickCount(void)
```

```
#004 {
#005       //获取第一次计时值。
#006       DWORD dwStart = ::GetTickCount();
#007       for (int i = 0; i < 10; i++)
#008       {
#009           //计算时间间隔。
#010           DWORD dwInterval = ::GetTickCount() - dwStart;
#011
#012           Sleep(100);
#013
#014           //显示时间的间隔。
#015           const int nBufSize = 256;
#016           TCHAR chBuf[nBufSize];
#017           wsprintf(chBuf,_T("dwInterval=%d\r\n"),dwInterval);
#018           OutputDebugString(chBuf);
#019       }
#020
#021 }
```


Windows API一日一练(70)GetSystemTime 和 GetLocalTime 函数

时间是一个非常重要的信息，比如写 LOG 时，就需要把时间输出来，跟踪程序是什么时候出错的。或者当你开发一个银行交易系统时，就要记录当前交易的时间，以便后面可以输出报表，打印给信用卡用户。根据不同的需求，可能需要使用不同的时间，目前有 UTC 和本地时间。UTC 是格林威治时间，也就是全球标准时间。本地时间就是相对于 UTC 而言的，比如中国北京是在东 8 区，相对于 UTC 就多了 8 个小时。一般使用到的时间都是使用本地时间，也就是调用函数 GetLocalTime。

函数 GetSystemTime 和 GetLocalTime 声明如下：

```
WINBASEAPI
VOID
WINAPI
GetSystemTime(
    __out LPSYSTEMTIME lpSystemTime
    );

WINBASEAPI
VOID
WINAPI
GetLocalTime(
    __out LPSYSTEMTIME lpSystemTime
    );
```

lpSystemTime 是获取系统时间的结构。

调用函数的例子如下：

```
#001
#002  //获取系统时间。
#003  //蔡军生 2007/11/11 QQ:9073204 深圳
#004  void TestSystem(void)
#005  {
#006      //获取系统的 UTC 时间。
#007      SYSTEMTIME stUTC;
#008      ::GetSystemTime(&stUTC);
#009
#010      //显示时间的间隔。
#011      const int nBufSize = 256;
#012      TCHAR chBuf[nBufSize];
#013      wsprintf(chBuf,_T("UTC: %u/%u/%u %u:%u:%u:%u %d\r\n"),

#014          stUTC.wYear, stUTC.wMonth, stUTC.wDay,
#015          stUTC.wHour, stUTC.wMinute, stUTC.wSecond,
#016          stUTC.wMilliseconds,stUTC.wDayOfWeek);
#017      OutputDebugString(chBuf);
#018
#019
#020      //获取当地的时间。
#021      SYSTEMTIME stLocal;
#022      ::GetLocalTime(&stLocal);
#023
#024      //显示时间的间隔。
#025      wsprintf(chBuf,_T("Local: %u/%u/%u %u:%u:%u:%u %d\r\n"),

#026          stLocal.wYear, stLocal.wMonth, stLocal.wDay,
#027          stLocal.wHour, stLocal.wMinute, stLocal.wSecond,
#028          stLocal.wMilliseconds,stLocal.wDayOfWeek);
#029      OutputDebugString(chBuf);
#030
#031  }
#032
```

上面两个函数在我测试时输出的结果，如下：

```
UTC: 2007/11/11 1:53:1:46 0
Local: 2007/11/11 9:53:1:46 0
```

Windows API一日一练(71)GetComputerName 函数

当你在一个大公司里面，当一个网络管理员时，发现成千上万个电脑需要你去管理时，怎么去区分这些电脑呢？那肯定是通过计算机的名称。而这个网络管理员又需要你开发一套软件，它可以把所有电脑的名称自动地上报给他。面对这样的需求，就可以使用下面的函数 GetComputerName 来获取计算机的名称，并通过网络传送给管理员。

函数 GetComputerName 声明如下：

```
WINBASEAPI
BOOL
WINAPI
GetComputerNameA (
    __out_ecount_part(*nSize, *nSize + 1) LPSTR lpBuffer,
    __inout LPDWORD nSize
    );
WINBASEAPI
BOOL
WINAPI
GetComputerNameW (
    __out_ecount_part(*nSize, *nSize + 1) LPWSTR lpBuffer,
    __inout LPDWORD nSize
    );
#ifdef UNICODE
#define GetComputerName GetComputerNameW
#else
#define GetComputerName GetComputerNameA
#endif // !UNICODE
```

lpBuffer 是获取电脑名称的缓冲区。
nSize 是输入缓冲区的大小和输出电脑名称的大小。

调用函数的例子如下：

```
#001 //
#002  //获取当前计算机的名称。
#003  //蔡军生 2007/11/12 QQ:9073204 深圳
#004  void GetPCName(void)
#005  {
#006          //
#007          const int nBufSize = MAX_COMPUTERNAME_LENGTH + 1;
#008          TCHAR chBuf[nBufSize];
#009          ZeroMemory(chBuf,nBufSize);
#010
#011          //获取当前计算机的名称
#012          DWORD dwRet = nBufSize;
```

```
#013        if (GetComputerName(chBuf,&dwRet))
#014         {
#015               //
#016               OutputDebugString(chBuf);
#017         }
#018         else
#019         {
#020               OutputDebugString(_T("获取计算名称出错!"));
#021         }
#022
#023         OutputDebugString(_T("\r\n"));
#024
#025  }
#026
#027
```


Windows API一日一练(72)GetUserName 函数

随着系统安全性的加强，每个系统里越来越多不同的帐号登录。假如你正在开发一个上网安全软件，让不同的用户有不同的上网权限，这样就需要识别当前的用户是什么帐号登录了，然后再作出权限分配。还有出错时，也需要对当前帐号进行记录下来，因为不同的帐号有不同的权限，有些磁盘是不允许操作的。面对这些需求，就需要使用函数 GetUserName。

函数 GetUserName 声明如下：

```
WINADVAPI
BOOL
WINAPI
GetUserNameA (
    __out_ecount_part(*pcbBuffer, *pcbBuffer) LPSTR lpBuffer,
    __inout LPDWORD pcbBuffer
    );
WINADVAPI
BOOL
WINAPI
GetUserNameW (
    __out_ecount_part(*pcbBuffer, *pcbBuffer) LPWSTR lpBuffer,
    __inout LPDWORD pcbBuffer
    );
#ifdef UNICODE
#define GetUserName GetUserNameW
#else
#define GetUserName GetUserNameA
```

```
#endif // !UNICODE
```

lpBuffer 是获取名称缓冲区。
pcbBuffer 是缓冲区的大小和返回帐号的大小。

调用函数的例子如下：


```
#001 //
#002  //获取当前登录用户的名称。
#003  //蔡军生 2007/11/13 QQ:9073204 深圳
#004  void GetUserName(void)
#005  {
#006        //
#007        const int nBufSize = UNLEN + 1;
#008        TCHAR chBuf[nBufSize];
#009        ZeroMemory(chBuf,nBufSize);
#010
#011        //获取当前登录用户的名称
#012        DWORD dwRet = nBufSize;
#013       if (::GetUserName(chBuf,&dwRet))
#014        {
#015              //
#016              OutputDebugString(chBuf);
#017        }
#018        else
#019        {
#020              OutputDebugString(_T("获取登录用户名称出错!"));
#021        }
#022
#023        OutputDebugString(_T("\r\n"));
#024
#025  }
```

Windows API一日一练(73)GetVersionEx 函数

Windows 发展还是非常快速的，从 Win95，到 Win98，再到 Win2000 和 XP 系统。每个系统的功能也有所不同，要想在程序里区别不同的系统，就需要获取系统的版本信息。比如 XP 里有防火墙，而其它以前的系统里没有带有的。但有时编写了一个服务器程序，或者编写 BT 程序，又需要设置一个端口对外面接收连接，因此在 XP 系统里就需要设置防火墙的端口，而在 XP 以前的系统里就没有必要设置端口了。这样的需求，就可以使用函数 GetVersionEx 来区分不同的系统。

函数 GetVersionEx 声明如下：

```
WINBASEAPI
BOOL
WINAPI
GetVersionExA(
    __inout LPOSVERSIONINFOA lpVersionInformation
    );
WINBASEAPI
BOOL
WINAPI
GetVersionExW(
    __inout LPOSVERSIONINFOW lpVersionInformation
    );
#ifdef UNICODE
#define GetVersionEx GetVersionExW
#else
#define GetVersionEx GetVersionExA
#endif // !UNICODE
```


lpVersionInformation 是返回系统版本的信息。

调用函数的例子如下：

```
#001 //
#002  //获取当前登录用户的名称。
#003  //蔡军生 2007/11/14 QQ:9073204 深圳
#004  void GetWinVersion(void)
#005  {
#006        //
#007        OSVERSIONINFO osvi;
#008        osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
#009
#010        //获取系统的版本信息。
#011        ::GetVersionEx(&osvi);
#012        bool bIsWindowsXPorLater = (osvi.dwMajorVersion > 5) ||
#013                                   (  (osvi.dwMajorVersion  ==  5)  &&
(osvi.dwMinorVersion >= 1) );
#014
#015        //显示当前的版本。
#016        if (bIsWindowsXPorLater)
#017        {
#018              OutputDebugString(_T("Windows XP 或更新版本!\r\n"));
#019        }
#020        else
#021        {
```

```
#022                OutputDebugString(_T("Windows XP 以前版本!\r\n"));
#023        }
#024    }
```