***AdjustWindowRect函数***

函数功能：该函数依据所需客户矩形的大小，计算需要的窗口矩形的大小。计算出的窗口矩形随后可以传递给CreateWindow函数，用于创建一个客户区所需大小的窗口。

BOOL AdjustWindowRect {

LPRECT lpRect;

[DWORD](http://baike.baidu.com/view/999769.htm)  dwStyle;

[BOOL](http://baike.baidu.com/view/1557195.htm)  bMENU; };

lpRect：指向RECT结构的指针，该结构包含所需客户区域的左上角和右下角的坐标。函数返回时，该结构容纳所需客户区域的窗口的左上角和右下角的坐标。

dwStyle：指定将被计算尺寸的窗口的窗口风格。(WS\_OVERLAPPEDWINDOW。WS\_OVERLAPPEDWINDOW可以创建一个拥有各种窗口风格的窗体，包括标题，系统菜单，边框，最小化和最大化按钮等。)

bMenu：指示窗口是否有菜单。如窗口有菜单，则设为0x1/\*\_TRUE\*/（非零）

返回值：如果函数成功，返回值为非零；如果函数失败，返回值为零。获取错误信息，参看GetLastError。

备注：客户矩形是指完全包含一个客户区域的最小矩形；窗口矩形是指完全包含一个窗口的最小矩形，该窗口包含客户区与非客户区。

当一个菜单条下拉出两行或更多行时，AdjustWindowRect函数不增加额外的空间。

速查：Windows NT：3.1以上版本：Windows：95以上版本：Windows CE：不支持；头文件：winuser.h；库文件：user32.lib。

***ARRAYSIZE 宏***

得到数组的长度

***ATOM数据类型***

ATOM类型为Windows中定义的新数据类型，其即unsigned short类型，在<WinDef.h>中的定义如下：

Typedef WORD ATOM;

typedef unsigned short WORD;

***BeginPaint 函数***

BeginPaint函数为指定窗口进行绘图工作的准备，并用将和绘图有关的信息填充到一个PAINTSTRUCT结构中。

HDC BeginPaint (

HWND hwnd,

LPPAINTSTRUCT lpPaint

);

hwnd:[输入]被重绘的窗口句柄

lpPaint:[输出]指向一个用来接收绘画信息的PAINTSTRUCT结构

返回值 编辑如果函数成功，返回值是指定窗口的“显示设备描述表”句柄。

如果函数失败，返回值是NULL，表明没有得到显示设备的内容。

Windows NT/2000/XP: 使用GetLastError得到更多的错误信息。

备注：编辑BeginPaint函数自动设置显示设备内容的剪切区域，而排除任何更新区域外的区域。该更新区域可以通过InvalidateRect或InvalidateRgn函数设置，也可以是系统在改变大小、移动、创建、滚动后设置的，亦是其他的影响客户区的操作来设置的。如果更新区域被标记为可擦除的，BeginPaint发送一个WM\_ERASEBKGND消息给窗口。

一个应用程序除了响应WM\_PAINT消息外，不应该调用BeginPaint。每次调用BeginPaint都应该有相应的EndPaint函数。

如果被绘画的客户区中有一个caret（caret：插入符。是窗口客户区中的一个闪烁的线，块，或位图。插入符通常表示文本或图形将被插入的地方。即一闪一闪的光标），BeginPaint自动隐藏该符号，而保证它不被擦除。

如果窗口类有一个背景刷，BeginPaint使用这个刷子来擦除更新区域的背景。

***CComPtr 类***

<https://msdn.microsoft.com/en-us/library/ezzw7k98(v=vs.120).aspx>

一个来控制COM接口指针的智能指针类。

CComPtr类继承自CComPtrBase类。

template < class T >

class CComPtr

T 一个COM接口用来定义存储的指针类型。

***CComPtrBase 类***

<https://msdn.microsoft.com/en-us/library/td098k1d.aspx>

此类提供了使用COM基础存储程序的智能指针类的基类。

template < class T >

class CComPtrBase

T 在智能指针中使用的对象类型。

***CComPtrBase::CoCreateInstance 方法***

<https://msdn.microsoft.com/en-us/library/810b85cb.aspx>

调用这个方法来创建一个与指定的类标识符或程序标识符联系的类的对象。

HRESULT CoCreateInstance (

LPCOLESTR szProgID,

LPUNKNOWN pUnkOuter = NULL,

DWORD dwClsContext = CLSCTX\_ALL

) throw( );

HRESULT CoCreateInstance (

REFCLSID rclsid,

LPUNKNOWN pUnkOuter = NULL,

DWORD dwClsContext = CLSCTX\_ALL

) throw( );

szProgID 指向程序标识符，使用来覆盖类标识符。

pUnkOuter 如果为空，表示对象不是作为集合的一部分创建。如果非空，是集合对象的IUnknown接口的指针。

dwClsContext

控制新创建的对象的代码将要运行的环境。

rclsid

与用来创建对象的数据和代码联系的类标识符。

返回值 成功返回S\_OK, 失败返回REGDB\_E\_CLASSNOTREG, CLASS\_E\_NOAGGREGATION, CO\_E\_CLASSSTRING 或 E\_NOINTERFACE。查看CoCreateInstance和CLSIDFromProgID得到错误描述。

如果第一种形式的方法被调用，CLSIDFromeProgID被使用覆盖CLSID。两种形式都会调用。在debug组建中，一个声明错误将会出现如果CComPtrBase::没有同样为NULL。

***CheckDlgButton 函数***

该函数改变按钮控制的选中状态。

BOOL CheckDlgButton（

HWND hDlg，

int nlDButton,

UINT uCheck

）；

hDlg：指向含有该按钮的对话框的句柄。

nlDButton：标识要修改的按钮。

uCheck：给定该按钮的选中状态。该参数可取下列值，这些值的含义如下：

BST\_CHECKED：设置按钮状态为己选中（checked）。

BST\_INDETERMINATE：设置按钮状态变灰，表示不确定状态。只有在该按钮具有BS\_3STATE或BS\_AUTO3STATE样式时才能使用该值。

BST\_UNCHECKED：设置按钮为未选中状态（unchecked）。

返回值：如果函数执行成功，返回值非零；如果函数失败，则返回值为零。若想获取更多错误信息，请调用 GetLastError函数。

***COMDLG\_FILTERSPEC 结构***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb773221(v=vs.85).aspx>

typedef struct \_COMDLG\_FILTERSPEC {

LPCWSTR pszName;

LPCWSTR pszSpec;

} COMDLG\_FILTERSPEC;

pszName 类型：LPCWSTR 一个包含过滤器友好名字的缓冲区指针。

pszSpec 类型：LPCWSTR 一个包含过滤模式的缓冲区指针。

***CreateDialogParam 函数***

该函数根据对话框模板资源创建一个无模式的对话框。在显示对话框之前，函数把一个应用程序定义的值作为WM\_INITDIALOG消息IParam参数传到对话框过程应用程序可用此值来初始化对话框控制。

HWND CreateDialogParam (

HINSTANCE hlnstancem,

LPCTSTR IpTemplateName，

HWND hWndParent，

DLGPROC IpDialogFunc,

LPARAM dwlniParam）；

hlnstance：标识一个模块的实例,该模块的可执行文件含有对话框模板。

IpTemplateName：标识对话框模板。此参数可以指向一个以NULL结尾的字符串的指针,该字符串指定对话框模板名,或是指定对话框模板的资源标识符的一个整型值。如果此参数指定了一个资源标识符，则它的高位字一定为零且低位字一定含有标识符。一定用 MAKEINTRESOURCE宏指令创建此值。

HwndParent：指定拥有对话框的窗口。

IpDialogFunc：指向对话框过程的指针。有关对话框过程的更详细的信息，请参见DialogProc。

dwlnitParam：指定传递到WM\_INITDIALOG消息的IParam参数中的对话框过程的值。

返回值：如果函数调用成功则返回值为指向对话框的窗口句柄。如果函数调用失败则返回值为NULL。

若想获得更多错误信息,请调用GetlastError函数。

备注：CreateDialogParam函数用 CreateWindowEx函数创建对话框。CreateDialogParam函数然后把一个WM\_INITDIALOG消息和一个WM\_SETFONT消息（如果模板指定DS SETFONT类型）传递到对话框过程。如果模板指定WS\_VISIBLE类型，则函数显示对话框，最后CreateDialogParam返回对话框的窗口句柄。

CreateDialogParam返回之后应用程序用ShowWindow显示对话框（如果还没有显示）。应用程序用DestroyWindoW函数来清除对话框。

Windows 95和以后版本：系统可支持每个对话框模板中最多255个控制。为把大于255个的控制放入对话框,需要在WM\_INITDIALOG消息处理器中创建控制,而不是把他们放入模板中。

Windows CE：IPTemplateName参数指向的对话框模板中DLGTEMPLATE结构并不支持所有的类型。

速查：Windows NT：3.1 及以上版本；Windows：95及以上版本；Windows CE：1.0及以上版本；头文件；winuser.h;库文件：use32.lib;Unicode:在Windows NT上实现为Unicode和ANSI两种版本。

***CreateEvent 函数***

<http://msdn.microsoft.com/en-us/library/windows/desktop/ms682396(v=vs.85).aspx>

用来创建或打开一个命名的或未命名的时间对象。如果想为对象指定一个访问掩码，应当使用CreateEventEx函数。

HANDLE WINAPI CreateEvent(

\_In\_opt\_ LPSECURITY\_ATTRIBUTES lpEventAttributes, 安全属性

\_In\_ BOOL bManualReset, 复位方式

\_In\_ BOOL bInitialState, 初始状态

\_In\_opt\_ LPCTSTR lpName 对象名称

);

lpEventAttributes[输入]

一个指向SECURITY\_ATTRIBUTES结构的指针，确定返回的句柄是否可被子进程继承。如果lpEventAttributes是NULL，此句柄不能被继承。

Windows NT/2000：lpEventAttributes的结构中的成员为新的事件指定了一个安全符。如果lpEventAttributes是NULL，事件将获得一个默认的安全符。

bManualReset[输入]

指定将事件对象创建成手动复原还是自动复原。如果是TRUE，那么必须用ResetEvent函数来手工将事件的状态复原到无信号状态。如果设置为FALSE，当一个等待线程被释放以后，系统将会自动将事件状态复原, 为无信号状态。

bInitialState[输入]

指定事件对象的初始状态。如果为TRUE，初始状态为有信号状态；否则为无信号状态。

lpName[输入]

指定事件的对象的名称，是一个以0结束的字符串指针。名称的字符格式限定在MAX\_PATH之内。名字是对大小写敏感的。

如果lpName指定的名字，与一个存在的命名的事件对象的名称相同，函数将请求EVENT\_ALL\_ACCESS来访问存在的对象。这时候，由于bManualReset和bInitialState参数已经在创建事件的进程中设置，这两个参数将被忽略。如果lpEventAttributes是参数不是NULL，它将确定此句柄是否可以被继承，但是其安全描述符成员将被忽略。

如果lpName为NULL，将创建一个无名的事件对象。

如果lpName的和一个存在的信号、互斥、等待计时器、作业或者是文件映射对象名称相同，函数将会失败，在GetLastError函数中将返回ERROR\_INVALID\_HANDLE。造成这种现象的原因是这些对象共享同一个命名空间。

终端服务(Terminal Services）：名称中可以加入"Global\"或是"Local\"的前缀，这样可以明确的将对象创建在全局的或事务的命名空间。名称的其它部分除了反斜杠(\），可以使用任意字符。详细内容可参考Kernel Object Name Spaces。

Windows 2000：在Windows 2000系统中，没有终端服务运行，"Global\"和"Local\"前缀将被忽略。名称的其它部分除了反斜杠（\），可以使用任意字符。

Windows NT 4.0以及早期版本，Windows 95/98：名称中除了反斜杠(\），可以使用任意字符。

如果函数调用成功，函数返回事件对象的句柄。如果对于命名的对象，在函数调用前已经被创建，函数将返回存在的事件对象的句柄，而且在GetLastError函数中返回ERROR\_ALREADY\_EⅪSTS。

如果函数失败，函数返回值为NULL，如果需要获得详细的错误信息，需要调用GetLastError。

***CreateFile 函数***

这是一个多功能的函数，可打开或创建以下对象，并返回可访问的句柄：控制台，通信资源，目录（只读打开），磁盘驱动器，文件，邮槽，管道。

HANDLE WINAPI CreateFile (

\_In\_ LPCTSTR lpFileName,

\_In\_ DWORD dwDesiredAccess,

\_In\_ DWORD dwShareMode,

\_In\_opt\_ LPSECURITY\_ATTRIBUTES lpSecurityAttributes,

\_In\_ DWORD dwCreationDisposition,

\_In\_ DWORD dwFlagsAndAttributes,

\_In\_opt\_ HANDLE hTemplateFile

);

lpFileName String 要打开的文件的名或设备名。这个字符串的最大长度在ANSI版本中为MAX\_PATH，在unicode版本中为32767。

dwDesiredAccess指定类型的访问对象。如果为 GENERIC\_READ 表示允许对设备进行读访问；如果为 GENERIC\_WRITE 表示允许对设备进行写访问（可组合使用）；如果为零，表示只允许获取与一个设备有关的信息 。

另外，还可以指定下面的控制标志：

标准控制权限(16-23位掩码)：

DELETE 删除对象的权限。

READ\_CONTROL 从对象的安全描述符中读取信息的权限，但不包括SACL(系统访问控制列表)中的信息。

WRITE\_DAC 修改对象安全描述符中的DACL(随机访问控制列表)的权限

WRITE\_OWNER 修改对象安全描述符中的属主的权限

SYNCHRONIZE 同步化使用对象的权限，即可以创建一个线程等待信号量释放(但有些对象不支持这个权限)。

STANDARD\_RIGHTS\_REQUIRED 等价于前面四种权限的总合(通常这四种是必须具有的权限)。

STANDARD\_RIGHTS\_READ 一般等价于READ\_CONTROL

STANDARD\_RIGHTS\_WRITE 一般等价于READ\_CONTROL

STANDARD\_RIGHTS\_EXECUTE 一般等价于READ\_CONTROL

STANDARD\_RIGHTS\_ALL 等价于前面五种权限的总合。

特殊控制权限(0-15位掩码)：

SPECIFIC\_RIGHTS\_ALL

ACCESS\_SYSTEM\_SECURITY

MAXIMUM\_ALLOWED

GENERIC\_READ

GENERIC\_WRITE

GENERIC\_EXECUTE

GENERIC\_ALL

注：实质上是通过ACCESS\_MASK结构体的一个双字值来设置标准权限、特殊权限和一般权限的。

dwShareMode Long， 如果是零表示不共享； 如果是FILE\_SHARE\_DELETE表示随后打开操作对象会成功只要删除访问请求；如果是FILE\_SHARE\_READ随后打开操作对象会成功只有请求读访问；如果是FILE\_SHARE\_WRITE 随后打开操作对象会成功只有请求写访问。

lpSecurityAttributes SECURITY\_ATTRIBUTES， 指向一个SECURITY\_ATTRIBUTES结构的指针，定义了文件的安全特性（如果操作系统支持的话）

dwCreationDisposition Long，下述常数之一：

CREATE\_NEW 创建文件；如文件存在则会出错

CREATE\_ALWAYS 创建文件，会改写前一个文件

OPEN\_EXISTING 文件必须已经存在。由设备提出要求

OPEN\_ALWAYS 如文件不存在则创建它

TRUNCATE\_EXISTING 将现有文件缩短为零长度

dwFlagsAndAttributes Long， 一个或多个下述常数

FILE\_ATTRIBUTE\_ARCHIVE 标记归档属性

FILE\_ATTRIBUTE\_COMPRESSED 将文件标记为已压缩，或者标记为文件在目录中的默认压缩方式

FILE\_ATTRIBUTE\_NORMAL 默认属性

FILE\_ATTRIBUTE\_HIDDEN 隐藏文件或目录

FILE\_ATTRIBUTE\_READONLY 文件为只读

FILE\_ATTRIBUTE\_SYSTEM 文件为系统文件

FILE\_FLAG\_WRITE\_THROUGH 操作系统不得推迟对文件的写操作

FILE\_FLAG\_OVERLAPPED 允许对文件进行重叠操作

FILE\_FLAG\_NO\_BUFFERING 禁止对文件进行缓冲处理。文件只能写入磁盘卷的扇区块

FILE\_FLAG\_RANDOM\_ACCESS 针对随机访问对文件缓冲进行优化

FILE\_FLAG\_SEQUENTIAL\_SCAN 针对连续访问对文件缓冲进行优化

FILE\_FLAG\_DELETE\_ON\_CLOSE 关闭了上一次打开的句柄后，将文件删除。特别适合临时文件

也可在Windows NT下组合使用下述常数标记：

SECURITY\_ANONYMOUS， SECURITY\_IDENTIFICATION， SECURITY\_IMPERSONATION， SECURITY\_DELEGATION， SECURITY\_CONTEXT\_TRACKING， SECURITY\_EFFECTIVE\_ONLY

hTemplate File，hTemplateFile为一个文件或设备句柄，表示按这个参数给出的句柄为模板创建文件（就是将该句柄文件拷贝到lpFileName指定的路径，然后再打开）。它将指定该文件的属性扩展到新创建的文件上面，这个参数可用于将某个新文件的属性设置成与现有文件一样，并且这样会忽略dwAttrsAndFlags。通常这个参数设置为NULL，为空表示不使用模板，一般为空。

***CreateWindow宏***

创建新窗口。

Windows API宏， 在WinUser.h中根据是否已定义Unicode被分别定义为CreateWindowW和CreateWindowA，后两者又被分别定义为对CreateWindowExW和CreateWindowExA函数的调用。

HWND CreateWindow(

LPCTSTR lpClassName,

LPCTSTR lpWindowName,

DWORD dwStyle,

int x,

int y,

int nWidth，

int nHeight，

HWND hWndParent,

HMENU hMenu，

HANDLE hlnstance，

LPVOID lpParam）；

lpClassName 指向一个空结束的字符串或整型数atom。如果该参数是一个整型量，它是由此前调用theGlobalAddAtom函数产生的全局量。这个小于0xC000的16位数必须是lpClassName参数字的低16位，该参数的高位必须是0。

如果lpClassName是一个字符串，它指定了窗口的类名。这个类名可以是任何用函数RegisterClass注册的类名，或是任何预定义的控制类名。请看说明部分的列表。

LPWindowName 指向一个指定窗口名的空结束的字符串指针。

如果窗口风格指定了标题条，由lpWindowName指向的窗口标题将显示在标题条上。当使用Createwindow函数来创建控制例如按钮，选择框和静态控制时，可使用lpWindowName来指定控制文本。

dwStyle 指定创建窗口的风格。该参数可以是下列窗口风格的组合再加上说明部分的控制风格。风格意义：

WS\_BORDER：创建一个带边框的窗口。

WS\_CAPTION：创建一个有标题框的窗口（包括WS\_BORDER风格）。

WS\_CHILD：创建一个子窗口。这个风格不能与WS\_POPUP风格合用。

WS\_CHILDWINDOW：与WS\_CHILD相同。

WS\_CLIPCHILDREN：当在父窗口内绘图时，排除子窗口区域。在创建父窗口时使用这个风格。

WS\_CLIPSIBLINGS：排除子窗口之间的相对区域，也就是，当一个特定的窗口接收到WM\_PAINT消息时，WS\_CLIPSIBLINGS 风格将所有层叠窗口排除在绘图之外，只重绘指定的子窗口。如果未指定WS\_CLIPSIBLINGS风格，并且子窗口是层叠的，则在重绘子窗口的客户区时，就会重绘邻近的子窗口。

WS\_DISABLED：创建一个初始状态为禁止的子窗口。一个禁止状态的窗口不能接受来自用户的输入信息。

WS\_DLGFRAME：创建一个带对话框边框风格的窗口。这种风格的窗口不能带标题条。

WS\_GROUP：指定一组控制的第一个控制。这个控制组由第一个控制和随后定义的控制组成，自第二个控制开始每个控制，具有WS\_GROUP风格，每个组的第一个控制带有WS\_TABSTOP风格，从而使用户可以在组间移动。用户随后可以使用光标在组内的控制间改变键盘焦点。

WS\_HSCROLL：创建一个有水平滚动条的窗口。

WS\_ICONIC：创建一个初始状态为最小化状态的窗口。与WS\_MINIMIZE风格相同。

WS\_MAXIMIZE：创建一个初始状态为最大化状态的窗口。

WS\_MAXIMIZEBOX：创建一个具有最大化按钮的窗口。该风格不能与WS\_EX\_CONTEXTHELP风格同时出现，同时必须指定WS\_SYSMENU风格。

WS\_OVERLAPPED：产生一个层叠的窗口。一个层叠的窗口有一个标题条和一个边框。与WS\_TILED风格相同。

WS\_OVERLAPPEDWINDOW：创建一个具有WS\_OVERLAPPED，WS\_CAPTION，WS\_SYSMENU WS\_THICKFRAME，WS\_MINIMIZEBOX，WS\_MAXIMIZEBOX风格的层叠窗口，与WS\_TILEDWINDOW风格相同。

WS\_POPUP：创建一个弹出式窗口。该风格不能与WS\_CHILD风格同时使用。

WS\_POPUPWINDOW：创建一个具有WS\_BORDER，WS\_POPUP,WS\_SYSMENU风格的窗口，WS\_CAPTION和WS\_POPUPWINDOW必须同时设定才能使窗口某单可见。

WS\_SIZEBOX：创建一个可调边框的窗口，与WS\_THICKFRAME风格相同。

WS\_SYSMENU：创建一个在标题条上带有窗口菜单的窗口，必须同时设定WS\_CAPTION风格。

WS\_TABSTOP：创建一个控制，这个控制在用户按下Tab键时可以获得键盘焦点。按下Tab键后使键盘焦点转移到下一具有WS\_TABSTOP风格的控制。

WS\_THICKFRAME：创建一个具有可调边框的窗口，与WS\_SIZEBOX风格相同。

WS\_TILED：产生一个层叠的窗口。一个层叠的窗口有一个标题和一个边框。与WS\_OVERLAPPED风格相同。

WS\_TILEDWINDOW:创建一个具有WS\_OVERLAPPED，WS\_CAPTION，WS\_SYSMENU， WS\_THICKFRAME，WS\_MINIMIZEBOX，WS\_MAXIMIZEBOX风格的层叠窗口。与WS\_OVERLAPPEDWINDOW风格相同。

WS\_VISIBLE：创建一个初始状态为可见的窗口。

WS\_VSCROLL：创建一个有垂直滚动条的窗口。

X 指定窗口的初始水平位置。对一个层叠或弹出式窗口，X参数是屏幕坐标系的窗口的左上角的初始X坐标。对于子窗口，x是子窗口左上角相对父窗口客户区左上角的初始X坐标。如果该参数被设为CW\_USEDEFAULT则系统为窗口选择缺省的左上角坐标并忽略Y参数。CW\_USEDEFAULT只对层叠窗口有效，如果为弹出式窗口或子窗口设定，则X和y参数被设为零。

Y 指定窗口的初始垂直位置。对一个层叠或弹出式窗口，y参数是屏幕坐标系的窗口的左上角的初始y坐标。对于子窗口，y是子窗口左上角相对父窗口客户区左上角的初始y坐标。对于列表框，y是列表框客户区左上角相对父窗口客户区左上角的初始y坐标。如果层叠窗口是使用WS\_VISIBLE风格位创建的并且X参数被设为CW\_USEDEFAULT，则系统将忽略y参数。

nWidth 以设备单元指明窗口的宽度。对于层叠窗口，nWidth或是屏幕坐标的窗口宽度或是CW\_USEDEFAULT。若nWidth是CW\_USEDEFAULT，则系统为窗口选择一个缺省的高度和宽度：缺省宽度为从初始X坐标开始到屏幕的右边界，缺省高度为从初始Y坐标开始到目标区域的顶部。CW\_USEDEFAULT只对层叠窗口有效；如果为弹出式窗口和子窗口设定CW\_USEDEFAULT标志则nWidth和nHeight被设为零。

nHeight 以设备单元指明窗口的高度。对于层叠窗口，nHeight是屏幕坐标的窗口宽度。若nWidth被设为CW\_USEDEFAULT，则系统忽略nHeight参数。

hWndParent 指向被创建窗口的父窗口或所有者窗口的句柄。若要创建一个子窗口或一个被属窗口，需提供一个有效的窗口句柄。这个参数对弹出式窗口是可选的。Windows NT 5.0；创建一个消息窗口，可以提供HWND\_MESSAGE或提供一个己存在的消息窗口的句柄。

hMenu 菜单句柄，或依据窗口风格指明一个子窗口标识。对于层叠或弹出式窗口，hMenu指定窗口使用的菜单：如果使用了菜单类，则hMenu可以为NULL。对于子窗口，hMenu指定了该子窗口标识（一个整型量），一个对话框使用这个整型值将事件通知父类。应用程序确定子窗口标识，这个值对于相同父窗口的所有子窗口必须是唯一的。

hlnstance 与窗口相关联的模块实例的句柄。

lpParam 指向一个值的指针，该值传递给窗口WM\_CREATE消息。该值通过在IParam参数中的CREATESTRUCT结构传递。如果应用程序调用CreateWindow创建一个MDI客户窗口，则lpParam必须指向一个CLIENTCREATESTRUCT结构。

返回值：如果函数成功，返回值为新窗口的句柄：如果函数失败，返回值为NULL。若想获得更多错误信息，请调用GetLastError函数。

备注

在返回前，CreateWindow给窗口过程发送一个WM\_CREATE消息。对于层叠，弹出式和子窗口，CreateWindow给窗口发送WM\_CREATE，WM\_GETMINMAXINFO和WM\_NCCREATE消息。消息WM\_CREATE的IParam参数包含一个指向CREATESTRUCT结构的指针。如果指定了WS\_VISIBLE风格，CreateWindow向窗口发送所有需要激活和显示窗口的消息。

***DefDlgProc 回调函数***

该函数为属于应用程序定义的对话框类的窗口过程提供缺省的消息处理。

LRESULT DefDlgProc（

HWND hDlg，

UINT Msg，

WPARAM wParam,

LPARAM Param

）；

hDlg：指定对话框。

Msg：指定消息数目。

wParam：指定消息中特定的其他信息。

Iparam：指定消息中特定的其他信息。

返回值：返回值指定消息处理的结果且依赖于发送的消息。

备注：DefDlgProc函数为对话框预定义类的窗口应用程序。此应用程序通过把消息传送到对框应用程序和为对话框应用程序返回的任何一个FALSE消息，提供缺省处理而为对话框提供内部的处理程序。为自定义对话框创建自定义应用程序的应用程序，常用DefDlgProc而不是DefWindowProc函数来执行缺省的消息处理。应用程序通过用合适的信息来填充一个WNCLASS结构和通过用RegisterClass函数登记的类创建自定义对话框类。一些应用程序用GetClasslnfo函数指定与定义对话框的名来填充此机构。在这种情况下，应用程序在登记之前至少改变IpszClassNamede数目。在所有的情况下，对于自定义对话框的WNDCLASS的cbWndExtra成员一定至少设置为DLGWINDOWEXTRA。DefDlgProc函数一定不要通过一个对话框应用程序来调用，这样做会导致循环执行。

速查：Windows NT：3.1及以上版本；Windows：95及以上版本；Windows CE：1.0及以上版本；头文件：winuser.h;库文件：user32.lib；Unicode：在Windows NT上实现为Unicode和ANSI两种版本。

***DefWindowProc 函数***

函数功能：该函数调用缺省的窗口过程来为应用程序没有处理的任何窗口消息提供缺省的处理。该函数确保每一个消息得到处理。调用DefWindowProc函数时使用窗口过程接收的相同参数。

DefWindowProc这个函数是默认的窗口处理函数，我们可以把不关心的消息都丢给它来处理。这个函数在处理关闭窗口消息WM\_CLOSE时，是调用DestroyWindow函数关闭窗口并且发WM\_DESTROY消息给应用程序；而它对WM\_DESTROY这个消息是不处理的（考虑为什么？）；我们在应用程序中对这个消息的处理是发出WM\_QUIT消息。因此WM\_CLOSE、WM\_DESTROY、WM\_QUIT这三个消息是先后产生的。

hWnd：指向接收消息的窗口过程的句柄。

Msg：指定消息类型。

wParam：指定其余的、消息特定的信息。该参数的内容与Msg参数值有关。

IParam：指定其余的、消息特定的信息。该参数的内容与Msg参数值有关。

返回值：返回值就是消息处理结果，它与发送的消息有关。

备注：对于Windows CE；如果Msg为WM\_SETTEXT那么返回0。

当DefWindowProc处理WM\_DESTROY消息时，它不自动调用PostQuitMessage。

速查：Windows NT 3.1以上版本；Windows：95以上版本：Windows CE以上版本；头文件；winuser.h；库文件：user32.lib；Unicode：在Windows NT环境中以Unicode和ANSI版本实现。

***DestroyWindow 函数***

销毁指定的窗口。这个函数通过发送WM\_DESTROY 消息和 WM\_NCDESTROY 消息使窗口无效并移除其键盘焦点。这个函数还销毁窗口的菜单，清空线程的消息队列，销毁与窗口过程相关的定时器，解除窗口对剪贴板的拥有权，打断剪贴板器的查看链。

BOOL DestroyWindow(

HWND hWnd

);

hWnd ：将被销毁的窗口的句柄。

返回值：如果函数成功，返回值为非零：如果函数失败，返回值为零。若想获得更多错误信息，请调用GetLastError函数。

备注：一个线程不能使用本函数销毁别的线程创建的窗口。如果这个窗口是一个不具有WS\_EX\_NOPARENTNOTIFY 样式的子窗口，则销毁窗口时将发WM\_PARENTNOTIFY 消息给其父窗口。

Windows CE: 本函数将不发送 WM\_NCDESTROY 消息.

速查：Windows NT：3.1以上版本；Windows：95以上版本；Windows CE：不支持：头文件：winuser.h；库文件：user32.lib。

***DialogBox 函数***

DialogBox是一个Windows API函数。它的作用是从一个对话框资源中创建一个模态对话框。该函数直到指定的回调函数通过调用EndDialog函数中止模态的对话框才能返回控制。该函数通过调用DialogBoxParam函数来实现。

int DialogBox(

HINSTANCE hInstance,

LPCTSTR lpTemplate,

HWND hWndParent,

DLGPROC lpDialogFunc

);

hlnstance 标识一个模块的实例，该模块的可执行文件含有对话框模板。

IpTemplate 标识对话框模板。此参数可以是指向一个以NULL结尾的字符串的指针，该字符串指定对话框模板名，或是指定对话框模板的资源标识符中的一个整型值。如果此参数指定了一个资源标识符则它的高位字一定为零，且低位字一定含有标识符。一定用 MAKEINTRESOURCE宏指令创建此值。

hWndParent 指定拥有对话框的窗口。

IpDialogFunc 指向对话框过程的指针。有关更详细的关于对话框过程的信息,请参见DialogProc。

返回值 如果函数调用成功，则返回值为在对函数EndDialog的调用中的nResult参数．该函数用于中止对话框。如果函数调用失败，则返回值为-1。若想获得更多的错误信息，请调用GetLastError函数

备注：

DialogBox宏用CreateWindowEx函数创建对话框。DialogBox函数然后把一个WM\_INITDIALOG消息（和一个WM-SETFONT消息，如果模板指定DS\_SETFONT类型）传递到对话框过程。不管模板是否指定WS\_VISIBLE类型，函数显示对话框，并且使拥有该对话框的窗口（也称属主窗口）失效，且为对话框启动它本身的消息循环来检索和传递消息。

当对话框应用程序调用EndDialog函数时，DialogBox函数清除对话框户止消息循环，使属主窗口生效（如果以前有效），且返回函数EndDialog调用中的nReSUlt参数。

***DispatchMessage 函数***

该函数分发一个消息给窗口程序。通常消息从GetMessage函数获得。消息被分发到回调函数（过程函数)，作用是消息传递给操作系统，然后操作系统去调用我们的回调函数，也就是说我们在窗体的过程函数中处理消息。

LONG DispatchMessage（CONST MSG\*lpmsg）；

lpmsg：指向含有消息的MSG结构的指针。

返回值：返回值是窗口程序返回的值。尽管返回值的含义依赖于被调度的消息，但返回值通常被忽略。

备注：MSG结构必须包含有效的消息值。如果参数lpmsg指向一个WM\_TIMER消息，并且WM\_TIMER消息的参数IParam不为NULL，则调用IParam指向的函数，而不是调用窗口程序。

***EnableWindow 函数***

Windows API函数。该函数允许/禁止指定的窗口或控件接受鼠标和键盘的输入，当输入被禁止时，窗口不响应鼠标和按键的输入，输入允许时，窗口接受所有的输入。

WINUSERAPI BOOL WINAPI EnableWindow(

\_In\_ HWND hWnd,

\_In\_ BOOL bEnable

);

hWnd:被允许/禁止的窗口句柄

bEnable:定义窗口是被允许，还是被禁止。若该参数为TRUE，则窗口被允许。若该参数为FALSE，则窗口被禁止。

返回值：编辑在 EnableWindow 成员函数调用之前，指示状态。 如果窗口此前已禁用，则返回值是非零。 返回值是0，则窗口先前启用或错误。若想获得更多的错误信息，可调用GetLastError函数。

编辑若窗口的允许状态将发生变化，WM\_ENABLE消息将在Enblewindow函数返回前发送出去，若窗口已被禁止，它所有的子窗口也被禁止，尽管并未向子窗口发送WM\_ENABLE消息。

窗口被激活前必须处于允许状态。比如，一个应用程序将显示一个无模式对话框并且已使该对话框的主窗口处于禁止状态，则在撤消该对话框之前须使其主窗口处于允许状态。否则，其他窗口将接受并被激活。若子窗口被禁止，则系统决定由哪个窗口接受鼠标消息时将忽略该窗口

缺省情况下，窗口被创建时被置为允许。若创建一个初始化为禁止状态的窗口，应用程序需要在CeateWindow或CeateWindowEX函数中定义WS\_DISABLED样式。窗口创建后，应用程序可用EnbleWindow来允许禁止窗口。

应用程序可利用此函数允许/禁止对话框中的某个控件，被禁止的控件既不能接受键盘输入，也不能被用户访问，但不能禁用一个拥有输入焦点的控制项。

***EndDialog 函数***

该函数清除一个模态对话框,并使系统中止对对话框的任何处理。

BOOL EndDialog(

HWND hDlg,

int nResult

);

hDlg：表示要被清除的对话框窗口。

NResult：指定从创建对话框函数返回到应用程序的值。

返回值：如果函数调用成功，则返回值为非零值；如果函数调用失败则返回值为零。若想获得错误信息请调用GetLastError函数。

备注：由DialogBox，DialogBoxParam、DialogBoxlndirect和DialogBoxlndirectParam函数创建的对话框一定要用EndDialog函数来清除。应用程序从对话框应用程序内部调用EndDialog函数，该函数不能为其他目的而供使用。对话框应用程序可以在任何时间调用EndDialog函数；甚至在WM\_INITDIALOG消息处理过程中。如果应用程序在WM\_INTDIALOG消息处理过程中调用该函数，则对话框在显示和输入焦点被设置之前对话框被清除。EndDialog函数并不立即清除对话框。而是设置一个标志，并且允许对话框应用程序把控制权返回系统。系统在试图从应用程序队列检索下一个消息之前检测标志。如果已经设置了标志则系统中止消息循环,清除对话框,且用nResUlt中的值作为从创建对话框的函数中返回的值。

速查：Windows NT：3.1及以上版本；Windows：95及以上版本；Windows CE：1.0及以上版本；头文件(Header)：Declared in Winuser.h, include Windows.h；库文件(Import library)：user32.lib。

***EndPaint 函数***

EndPaint函数标记指定窗口的绘画过程结束；这个函数在每次调用BeginPaint函数之后被请求，但仅仅在绘画完成以后。

BOOL EndPaint(

HWND hWnd,

CONST PAINTSTRUCT \*lpPaint

);

hWnd：[输入]已经被重画的窗口的HANDLE

lpPaint：[输入]指向一个PAINTSTRUCT结构，该结构包含了绘画信息，是BeginPaint函数返回的返回值：

返回值：返回值始终是非0

备注：PAINTSTRUCT 结构体包含了用于绘制窗口客户区的信息。

PAINTSTRUCT定义：

typedef struct tagPAINTSTRUCT {

HDC hdc;

BOOL fErase;

RECT rcPaint;

BOOL fRestore;

BOOL fIncUpdate;

BYTE rgbReserved[32];

} PAINTSTRUCT, \*PPAINTSTRUCT;

hdc是用于绘制的句柄，fErase如果为非零值则擦除背景，否则不擦除背景，rcPaint 通过制定左上角和右下角的坐标确定一个要绘制的矩形范围，该矩形单位相对于客户区左上角，后面三个参数都是系统预留的，编程一般用不到。

***EnterCriticalSection函数、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。

WINBASEAPI VOID WINAPI EnterCriticalSection(

\_\_inout LPCRITICAL\_SECTION lpCriticalSection

);

是多线程中用来确保同一时刻只有一个线程操作被保护的数据的操作函数，相关的多线程数据操作函数还有：

InitializeCriticalSection(&cs);//初始化临界区

EnterCriticalSection(&cs);//进入临界区

//操作数据

MyMoney\*=10;//所有访问MyMoney变量的程序都需要这样写Enter.. Leave... LeaveCriticalSection(&cs);//离开临界区

DeleteCriticalSection(&cs);//删除临界区

***GetDlgItem 函数***

返回窗口中指定参数ID的子元素的句柄，可以通过返回的句柄对窗口内的子元素进行操作。

HWND GetDlgItem(

HWND hDlg,

intnID DlgItem

);

hDlg：标识含有控件的对话框。

nlDDlgltem：指定将被检索的控件标识符。

返回值：如果函数调用成功则返回值为给定控件的窗口句柄。如果函数调用失败，则返回值为NULL，表示为一个无效的对话框句柄或一个不存在的控件。若想获得更多错误信息，请调用GetLastError函数。

备注：可以通过使用任何父子窗口对来使用GetDlgltem函数，而不仅只是对话框。只要hDlg参数指定一个父窗口，且子窗口有一个独立的标识符（象CreateWindow中hMenu参数指定的或创建子窗口的CreateWindowEx指定的那样），GetDlgltem就会返回一个有效的句柄到子窗口。

***GetMessage 函数***

函数GetMessage 是 从调用线程的消息队列里取得一个消息并将其放于指定的结构。此函数可取得与指定窗口联系的消息和由PostThreadMesssge寄送的线程消息。此函数接收一定范围的消息值。GetMessage不接收属于其他线程或应用程序的消息。获取消息成功后，线程将从消息队列中删除该消息。函数会一直等待直到有消息到来才有返回值。

GetMessageW(

\_Out\_ LPMSG lpMsg,

\_In\_opt\_ HWND hWnd,

\_In\_ UINT wMsgFilterMin,

\_In\_ UINT wMsgFilterMax

);

lpMsg：指向MSG结构的指针，该结构从线程的消息队列里接收消息信息。

hWnd：取得其消息的窗口的句柄。当其值取NULL时，GetMessage为任何属于调用线程的窗口检索消息，线程消息通过PostThreadMessage寄送给调用线程。

wMsgFilterMin：指定被检索的最小消息值的整数。

wMsgFilterMax：指定被检索的最大消息值的整数。

返回值：如果函数取得WM\_QUIT之外的其他消息，返回非零值。如果函数取得WM\_QUIT消息，返回值是零。如果出现了错误，返回值是-1。例如，当hWnd是无效的窗口句柄或lpMsg是无效的指针时。若想获得更多的错误信息，请调用GetLastError函数。

***GetSystemDirectory 函数***

这个函数能取得Windows系统目录（System目录）的完整路径名。在这个目录中，包含了所有必要的系统文件。根据微软的标准，其他定制控件和一些共享组件也可放到这个目录。通常应避免在这个目录里创建文件。在网络环境中，往往需要管理员权限才可对这个目录进行写操作。

UINT WINAPI GetSystemDirectory (

\_\_out LPTSTR lpBuffer,

\_\_in UINT uSize

);

lpBuffer String，用于装载系统目录路径名的一个字串缓冲区。它应事先初始化成“路径”字符串的长度+1。通常至少要为这个缓冲区分配MAX\_PATH个字符的长度

nSize Long，lpBuffer字串的最大长度

***GetWindowLong 函数（老）***

GetWindowLong是一个Windows API函数。该函数获得指定窗口的有关信息，函数也获得在额外窗口内存中指定偏移位地址的32位度整型值。

LONG GetWindowLong (

HWND hWnd,

int nlndex

);

hWnd 目标窗口的句柄。它可以是窗口句柄及间接给出的窗口所属的窗口类。

nlndex 需要获得的相关信息的类型。

指定要获得值的大于等于0的值的偏移量。有效值的范围从0到额外窗口内存空间的字节数一4例如，若指定了12位或多于12位的额外类存储空间，则应设为第三个32位整数的索引位8。要获得任意其他值，指定下列值之一：

GWL\_EXSTYLE (-20) 获得扩展窗口风格。

GWL\_HINSTANCE (-6) 获得应用实例的句柄。

GWL\_HWNDPARENT (-8) 如果父窗口存在，获得父窗口句柄。

GWL\_ID (-12) 获得窗口标识。

GWL\_STYLE (-16) 获得窗口风格。

GWL\_USERDATA (-21) 获得与窗口有关的32位值。每一个窗口均有一个由创建该窗口的应用程序使用的32位值。

GWL\_WNDPROC (-4) 获得窗口过程的地址，或代表窗口过程的地址的句柄。必须使用CallWindowProc函数调用窗口过程。

hWnd参数为对话框句柄时，还可用下列值：

DWL\_DLGPROC (4) 获得对话框过程的地址，或一个代表对话框过程的地址的句柄。必须使用函数CallWindowProc来调用对话框过程。

DWL\_MSGRESULT (0) 获得在对话框过程中一个消息处理的返回值。

DWL\_USER (8) 获得应用程序私有的额外信息，例如一个句柄或指针。

返回值 编辑如果函数成功，返回值是所需的32位值；如果函数失败，返回值是0。若想获得更多错误信息请调用 GetLastError函数。

***HDC 设备上下文句柄***

HDC设备上下文是一种包含有关某个设备（如显示器或打印机）的绘制属性信息的 Windows 数据结构。所有绘制调用都通过设备上下文对象进行，这些对象封装了用于绘制线条、形状和文本的 Windows API。

***HIWORD 宏***

#define HIWORD(I) ( ( WORD ) ( ( ( DWORD )( I ) >> 16) & 0xFFFF ) )。

这个宏传回一个WORD值（16位的无符号整数，范围从0到0xxFFFF）。

一般用来取得LPARAM的高位(高16位)，用在滚动条相关的SB\_THUMBTRACK和SB\_THUMBPOSITION消息中取得信息。

***IFileDialog 接口***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb775966>

包含普通文件对话框初始化、显示、得到结果的方法。

IFileDialog接口继承自IModalWindow接口。

***IFileDialog::SetDefaultExtension 方法***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb775970(v=vs.85).aspx>

设置添加到文件名中的默认扩展名。

HRESULT SetDefaultExtension(

[in] LPCWSTR pszDefaultExtension

);

pszDefaultExtension 类型：LPCWSTR 一个包含扩展名文本的缓冲区指针。此字符串不应包含前面的点。例如，”jpg”是正确的，而”.jpg”是不正确的。

返回值 类型：HRESULT 如果方法成功返回S\_OK，否则它返回HRESULT错误代码。

如果这个方法在显示对话框前调用，当用户选择新的文件类型（查看SetFileTypes）时，对话框将自动更新默认扩展名。

***IFileDialog::SetFileName 方法***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb775974(v=vs.85).aspx>

设置当对话框打开时显示在File name编辑框中的文件名。

HRESULT SetFileName(

[in] LPCWSTR pszName

);

pszName 类型：LPCWSTR 一个指向文件名的指针。

返回值 类型：HRESULT 如果方法成功返回S\_OK，否则它返回HRESULT错误代码。

***IFileDialog::SetFileTypes 方法***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb775980(v=vs.85).aspx>

设置对话框能够打开或关闭的文件类型。

HRESULT SetFileTypes (

[in] UINT cFileTypes,

[in] const COMDLG\_FILTERSPEC \*rgFilterSpec

);

cFileTypes [in] 类型: UINT 由rgFilterSpec指定的数组中元素的数目。

rgFilterSpec [in] 类型: const COMDLG\_FILTERSPEC\* 一个COMDLG\_FILTERSPEC结构类型的数组，每个代表一个文件类型。

返回值 类型: HRESULT 如果方法成功，返回S\_OK,否则返回HRESULT错误代码如下：

E\_UNEXPECTED SetFileTypes已经被调用。

E\_UNEXPECTED 在IFileDialog::SetOptions方法中设置了FOS\_PICKFOLDERS标记。.

E\_INVALIDARG rgFilterSpec参数为NULL.

当使用了Open对话框，这儿的文件类型声明用于过滤视图。当使用Save对话框，这些值决定哪个文件扩展名添加到文件名。

这个方法必须在对话框显示前调用，并且一个对话框实例只能调用一次。一旦普通项目对话框被显示，文件类型将不能被更改。

***IFileDialog::SetOkButtonLabel 方法***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb761830(v=vs.85).aspx>

设置Open或Save按钮的文本。

HRESULT SetOkButtonLabel (

[in] LPCWSTR pszText

);

pszText 类型：LPCWSTR 一个包含按钮文本的缓冲区指针。

返回值 类型：HRESULT 如果方法成功返回S\_OK，否则它返回HRESULT错误代码。

***IFileDialog::SetTitle方法***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb761834>

设置对话框的标题

HRESULT SetTitle(

[in] LPCWSTR pszTitle

);

pszTitle 类型：LPCWSTR 一个包含标题文本的缓冲区指针。

返回值 类型：HRESULT 如果方法成功返回S\_OK，否则它返回HRESULT错误代码。

***IFileSaveDialog 接口***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb775688>

通过增加保存对话框的特别方法扩展IFileDialg接口，包括提供支持收集元数据来在文件里持续保存。

此接口继承自IFileDialog接口。

***IModalWindow 接口***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb761686(v=vs.85).aspx>

包含代表一个模型窗口的方法。

IModalWindow接口继承自IUnknown接口。

***IModalWindow::Show 方法***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb761688(v=vs.85).aspx>

显示标准窗口。

HRESULT Show (

[in, optional] HWND hwndOwner

);

hwndOwner 类型：HWND 拥有者窗口的句柄。这个值可以为NULL。

返回值 类型：HRESULT 如果成功返回S\_OK;如果失败返回HRESULT错误代码如下：

HRESULT\_FROM\_WIN32(ERROR\_CANCELLED) 用户通过取消操作关闭了窗口。

***IShellItem 接口***

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb761144(v=vs.85).aspx>

包含检索关于Shell项目的方法。IShellItem和IShellItem2在任何新的代码中是首选代表项目的。

IShellItem接口继承自IUnknown接口。

***IsDialogMessage 函数***

该函数决定一个消息是否指定给指定的对话框，如果是，则处理消息。

BOOL IsDialogMessage (

HWND hDlg,

LPMSG IpMsg

);

hDlg：标识对话框。

IpMsg：指向一个含有将被检测的消息的MSG结构。

返回值：如果消息被处理，则返回值为非零值；如果消息没有被处理，则返回值为零。

备注：尽管IsDialogMessage函数是为无模态对话框而扩展的，但可以用含有控制的任何一个窗口来使用它。当 IsDialogMessage处理一个消息时，它检测键盘信息并把它们转变成对响应对话框的选择命令。例如当按下tab时选择下一个控制或控制组，当按下down时选择控制组的下一个控制。因为IsDialogMessage 函数要执行消息所有必要的转变和传送，IsDialogMessage函数处理的消息不必传送给TranslateMessage或 DispatchMessage函数处理。 IsDialogMessage函数把WM\_GETDLGETDLGCODE发送到对话框应用程序，决定应该处理哪个键。 lsDialogMessage函数把DMa\_ETDEFID和DM\_SETDEFID消息发送到窗口。这些消息在WINUSERH头文件中定义为WM\_USER和WM\_USER+1所以就有与应用程序定义的有一样值的消息发生冲突。

检测该消息是否是对应句柄的对话框消息，如果是，则交个user32.dll模块对齐进行处理，实际上user32.dll模块将其转发给对话框所属的窗口函数了，在对话框所属的窗口函数里面对消息进行判断，最终调用消息对应的函数，而消息对应的函数是我们之前映射好的，已经存在于对话框消息映射表了。

***LoadAccelerators 宏***

调入加速键表。该函数调入指定的加速键表。

HACCEL LoadAccelerators（

HINSTANCE hlnstance，

LPCTSTR lpTableName

）；

hlnstance 模块的一个实例的句柄，该模块的可执行文件中包含将要调入的加速键表。

IpTableName 指向一个以空结尾的字符串的指针，该字符串包含了即将调入的加速键表的名字。另一种可选的方案是，该参数可以在加速键表资源的低位字中指定资源标识符，而高位字中全零。MADEINTRESOURCE宏可被用于创建该值。

返回值：若函数调用成功，则返回所加载的加速键表句柄[1] 。若函数调用失败，则返回值为NULL。若要获得更多的错误信息，可以调用GetLastError函数。

备注：若加速键表尚未装入，该函数可从指定的可执行文件中将它装入。从资源中装入的加速键表，在程序结束时可自动释放。Windows CE：资源不被拷贝到RAM中，因而不能被修改。

***LoadIcon 函数***

LoadIcon函数从与hInstance模块相关联的可执行文件中装入lpIconName指定的图标资源，仅当图标资源还没有被装入时该函数才执行装入操作，否则只获取装入的资源句柄。

HICON LoadIcon(HINSTANCE hInstance,LPCTSTR lpIconName);

***LoadString 函数***

从资源里加载字符串资源到CString对象里。

WINUSERAPI int WINAPI LoadStringA(

\_\_in\_opt HINSTANCE hInstance,

\_\_in UINT uID,

\_\_out\_ ecount(cchBufferMax) LPSTR lpBuffer,

\_\_in int nBufferMax

);

参数1: hInstance是应用程序实例句柄。

参数2: uID是资源中的字符串编号。（ID）

参数3: lpBuffer是接收从资源里拷贝字符串出来的缓冲区。（预先定义的字符数组或指针）

参数4: nBufferMax是指明缓冲的大小。

***LOWORD 函数***

WORD LOWORD(

DWORD dwValue

);

dwvalue 设置指定要转换的值。

返回值 返回值是指定值的低位单字

***lstrlen 函数***

该函数返回指定字符串的字节长度（ANSI版）或字符长度（双字节标准版）；其中不包括终止NULL字符。

int lstrlen ( LPCTSTR lpString );

lpString：指向以NULL为终止符的字符串。

返回值 指定字符串中字节（ANSI版）或字符（双字节标准版）的长度。

***MAKEINTRESOURCE 宏***

MAKEINTRESOURCE是一个资源名转换的宏，这个宏是把一个数字类型转换成指针类型的宏，它不存在释放的问题。

***memcpy 函数***

c和c++使用的内存拷贝函数，memcpy函数的功能是从源src所指的内存地址的起始位置开始拷贝n个字节到目标dest所指的内存地址的起始位置中。

void \*memcpy (void \*dest,

const void \*src,

size\_t n );

返回值：函数返回指向dest的指针。

***memcpy\_s 函数， wmemcpy\_s 函数***

<http://technet.microsoft.com/zh-cn/library/wes2t00f(v=vs.110).aspx>

在缓冲区之间复制字节。 这些是 memcpy， wmemcpy 的版本与安全增强的 CRT中的安全功能如中所述。

errno\_t memcpy\_s(

void \*dest,

size\_t numberOfElements,

const void \*src,

size\_t count

);

errno\_t wmemcpy\_s(

wchar\_t \*dest,

size\_t numberOfElements,

const wchar\_t \*src,

size\_t count

);

dest 新的缓冲区。

numberOfElements 目标缓冲区的大小。

src 要复制的缓冲区。

count 要复制的字节长度。

返回值 零，成功;非零，错误代码。

备注:

memcpy\_s 复制 count 个字节从src到 dest; wmemcpy\_s 复制 count 个宽字节 (两个字节)。 如果源和目标重叠， memcpy\_s 行为不确定。 使用 memmove\_s 处理重叠区域。

这些功能验证其参数。 如果 dest 或 src 是 null 指针，或者 numberOfElements 为缓冲区太小，这些函数调用的参数无效处理程序，如 参数验证所述。 如果执行允许继续，这些函数返回 EINVAL 并将 errno 到 EINVAL。

***MessageBox 函数***

MessageBox显示一个模态对话框，其中包含一个系统图标、 一组按钮和一个简短的特定于应用程序消息，如状态或错误的信息。消息框中返回一个整数值，该值指示用户单击了哪个按钮。

int WINAPI MessageBox (

HWND hWnd,

LPCTSTR lpText,

LPCTSTR lpCaption,

UINT uType

);

hWnd：消息框的所属窗口句柄。如果为NULL，则消息框没有拥有窗口。

lpText：消息框的内容。如果使用了Unicode库，则把文本变成：TEXT(//sometext)

lpCaption：消息框的标题。如果使用了Unicode库，则把文本变成：TEXT(//sometext)

uType：指定一个决定对话框的内容和行为的位标志集。此参数可以为下列标志组中标志的组合。指定下列标志中的一个来显示消息框中的按钮以及图标。

MB\_OK 默认值。有一个确认按钮在里面。

MB\_YESNO 有是和否在里面。

MB\_ABORTRETRYIGNORE 有Abort（放弃），Retry（重试）和Ignore（跳过）

MB\_YESNOCANCEL 消息框含有三个按钮：Yes，No和Cancel

MB\_RETRYCANCEL 有Retry（重试）和Cancel（取消）

MB\_OKCANCEL 消息框含有两个按钮：OK和Cancel

返回值：

IDOK（1） OK

IDCANCEL（2） CANCEL

IDABORT（3） ABORT

IDRETRY（4） RETRY

IDIGNORE（5） IGNORE

IDYES（6） YES

IDNO（7） NO

***MSG 结构***

在Windows程序中，消息是由MSG结构体来表示的。MSG结构体的定义如下（参见MSDN）：

typedef struct tagMSG {

HWND hwnd;

UINT message;

WPARAM wParam;

LPARAM lParam;

DWORD time;

POINT pt;

} MSG;

该[结构体](http://baike.baidu.com/view/204974.htm)中各[成员变量](http://baike.baidu.com/view/684821.htm)的含义如下：

第一个[成员变量](http://baike.baidu.com/view/684821.htm)hwnd表示消息所属的窗口。我们通常开发的程序都是窗口[应用程序](http://baike.baidu.com/view/330120.htm)，一个消息一般都是与某个窗口相关联的。例如，在某个[活动窗口](http://baike.baidu.com/view/1988070.htm)中按下鼠标左键，产生的按键消息就是发给该窗口的。在Windows程序中，用HWND类型的[变量](http://baike.baidu.com/view/296689.htm)来标识窗口。

第二个[成员变量](http://baike.baidu.com/view/684821.htm)message指定了消息的[标识符](http://baike.baidu.com/view/390932.htm)。在Windows中，消息是由一个数值来表示的，不同的消息对应不同的数值。但是由于数值不便于记忆，所以Windows将消息对应的数值定义为WM\_XXX宏（WM是Window Message的缩写）的形式，XXX对应某种消息的英文拼写的大写形式。例如，鼠标左键按下消息是[WM\_LBUTTONDOWN](http://baike.baidu.com/view/1514116.htm)，键盘按下消息是WM\_KEYDOWN，字符消息是WM\_CHAR，等等。在程序中我们通常都是以WM\_XXX宏的形式来使用消息的。

提示：如果想知道WM\_XXX消息对应的具体数值，可以在[Visual C++](http://baike.baidu.com/view/100377.htm)[开发环境](http://baike.baidu.com/view/4831305.htm)中选中WM\_XXX，然后单击鼠标右键，在弹出菜单中选择goto definition，即可看到该宏的具体定义。跟踪或查看某个[变量](http://baike.baidu.com/view/296689.htm)的定义，都可以使用这个方法。

第三、第四个[成员变量](http://baike.baidu.com/view/684821.htm)wParam和lParam，用于指定消息的附加信息。例如，当我们收到一个[字符](http://baike.baidu.com/view/263416.htm)消息的时候，message[成员变量](http://baike.baidu.com/view/684821.htm)的值就是WM\_CHAR，但用户到底输入的是什么字符，那么就由wParam和lParam来说明。wParam、lParam表示的信息随消息的不同而不同。如果想知道这两个[成员变量](http://baike.baidu.com/view/684821.htm)具体表示的信息，可以在MSDN中关于某个具体消息的说明文档查看到。读者可以在VC++的开发环境中通过goto definition查看一下WPARAM和LPARAM这两种类型的定义，可以发现这两种类型实际上就是unsigned int和long。

最后两个[变量](http://baike.baidu.com/view/296689.htm)分别表示消息投递到[消息队列](http://baike.baidu.com/view/262473.htm)中的时间和鼠标的当前位置。

hwnd

Identifies the window whose window procedure receives the message.

message

Specifies the message number.

wParam

Specifies additional information about the message. The exact meaning depends on the value of the message member.

lParam

Specifies additional information about the message. The exact meaning depends on the value of the message member.

time

Specifies the time at which the message was posted.

pt

Specifies the cursor position, in screen coordinates, when the message was posted.

***MsgWaitForMultipleObjects 函数***

等候单个对象或一系列对象发出信号---标志着规定的超时已经过去，或特定类型的消息已抵达线程的输入队列。如返回条件已经满足，则立即返回

WINUSERAPI DWORD WINAPI MsgWaitForMultipleObjects(

\_In\_ DWORD nCount,

\_In\_reads\_opt\_(nCount) CONSTHANDLE \*pHandles,

\_In\_ BOOL fWaitAll,

\_In\_ DWORD dwMilliseconds,

\_In\_ DWORD dwWakeMask

);

nCount Long，指定列表中的句柄数量

pHandles Long，指定对象句柄组合中的第一个元素

fWaitAll Long，如果为TRUE，表示除非对象同时发出信号，否则就等待下去。如果为FALSE，表示任何对象发出信号即可。

dwMilliseconds Long，指定要等待的毫秒数。

dwWakeMask Long，带有QS\_？？前缀的一个或多个常数，用于标识特定的消息类型。

返回值，Long，如fWaitAll设为TRUE，则下述任何一个常数都标志着成功：

WAIT\_ABANDONED\_0：所有对象都发出消息，而且其中一个或多个属于互斥体（一但拥有它门的进程中止，就会发出信号）。

WAIT\_TIMEOUT：对象保持未发信号的状态，但规定的等待超时时间已经超过

WAIT\_OBJECT\_0：所有的对象都发出信号

WAIT\_TO\_COMPLETION（仅适用于WaitForSingleObjectEx），由于一个I/O完成操作已准备好执行，从而造成了函数的返回。

返回WAIT\_FAILED表示函数执行失败。会设置GetLastError

如fWaitAll设为FALSE，那返回结果与前面说的相似，只是可能还会返回相对于WAIT\_ABANDONED\_0或WAIT\_OBJECT\_0的一个正偏移量，指出哪个对象是被抛弃还是发出信号。

如果是由于dwWakeMask指定的，符合特殊标准的一条消息的到达而造成了函数的返回，则返回WAIT\_OBJECT\_0 + nCount

***PAINTSTRUCT 结构***

该结构体包含了某应用程序用来绘制它所拥有的窗口客户区所需要的信息。

typedef struct tag PAINTSTRUCT {

HDC hdc;

BOOL fErase;

RECT rcPaint;

BOOL fRestore;

BOOL fIncUpdate;

BYTE rgbReserved[32];

} PAINTSTRUCT, \*PPAINTSTRUCT;

PAINTSTRUCT 结构体包含了用于绘制窗口客户区的信息。例如要更新的客户区的矩形区域的大小等等，MFC里的CPaintDC与之对应；

BeginPaint可以得到客户区设备描述表的句柄，GetDC也可以得到，MFC里的CClientDC与之对应。

hdc是用于绘制的句柄，

fErase如果为非零值则擦除背景，否则不擦除背景，

rcPaint 通过制定左上角和右下角的坐标确定一个要绘制的矩形范围，该矩形单位相对于客户区左上角，

后面三个参数都是系统预留的，编程一般用不到。

***PeekMessage 函数***

PeekMessage是一个Windows API函数。该函数为一个消息检查线程消息队列，并将该消息（如果存在）放于指定的结构。

BOOL PeekMessage (

LPMSG IpMsg,

HWND hWnd,

UINT wMSGfilterMin,

UINT wMsgFilterMax,

UINT wRemoveMsg

);

lpMsg 接收消息信息的MSG结构指针。

hWnd 其消息被检查的窗口句柄。

wMsgFilterMin 指定被检查的消息范围里的第一个消息。

wMsgFilterMax 指定被检查的消息范围里的最后一个消息。

wRemoveMsg 确定消息如何被处理。此参数可取下列值之一：

PM\_NOREMOVE PeekMessage 处理后，消息不从队列里除掉。

PM\_REMOVE PeekMessage 处理后，消息从队列里除掉。

PM\_NOYIELD 此标志使系统不释放等待调用程序空闲的线程。可将PM\_NOYIELD随意组合到PM\_NOREMOVE或PM\_REMOVE。

缺省地，处理所有类型的消息。若只处理某些消息，指定一个或多个下列值[1] ：

PM\_QS\_INPUT Windows NT 5.0和Windows 98：处理鼠标和键盘消息。

PM\_QS\_PAINT Windows NT 5.0和Windows 98：处理画图消息。

PM\_QS\_POSTMESSAGE Windows NT 5.0和Windows 98：处理所有被寄送的消息，包括计时器和热键。

PM\_QS\_SENDMESSAGE Windows NT 5.0和Windows 98：处理所有发送消息。

返回值 编辑如果消息可得到，返回非零值；如果没有消息可得到，返回值是零。

备注: 和函数GetMessage不一样的是，GetMessage:从系统获取消息，将消息从系统中移除，属于阻塞函数。当系统无消息时，GetMessage会等待下一条消息。而函数PeekMesssge是以查看的方式从系统中获取消息，可以不将消息从系统中移除，是非阻塞函数；当系统无消息时，返回FALSE，继续执行后续代码。

PeekMesssge只得到那些与参数hWnd标识的窗口相联系的消息或被lsChild确定为其子窗口相联系的消息，并且该消息要在由参数wMsgFiterMin和wMsgFiherMax确定的范围内。如果hWnd为NULL，则PeekMessage接收属于当前调用线程的窗口的消息（PeekMessage不接收属于其他线程的窗口的消息）。如果hWnd为-1，PeekMessage只返回hWnd值为NULL的消息，该消息由函数PostThreadMessage寄送。如果wMsgFilterMin和wMsgFilterMax都为零，PeekMessage返回所有可得的消息（即，无范围过滤）。

常数WM\_KEYFIRST和WM\_KEYLAST可作为过滤值取得所有键盘消息；常数WM\_MOUSEFIRST和WM\_MOUSELAST可用来接收所有的鼠标消息。

PeekMessage通常不从队列里清除WM\_PAINT消息。该消息将保留在队列里直到处理完毕。但如果WM\_PAINT消息有一个空更新区，PeekMessage将从队列里清除WM\_PAINT消息。

Windows CE：有一个NULL更新区的WM\_PAINT消息不从队列里清除。

***PostQuitMessage 函数***

PostMessage是Windows API(应用程序接口) 中的一个常用函数，用于将一条消息放入到消息队列中。

该函数将一个消息放入（寄送）到与指定窗口创建的线程相联系消息队列里，不等待线程处理消息就返回，是异步消息模式。消息队列里的消息通过调用GetMessage和PeekMessage取得。

BOOL WINAPI PostMessage (

HWND hWnd,

UINT Msg,

WPARAM wParam,

LPARAM lParam

);

hWnd：其窗口程序接收消息的窗口的句柄。可取有特定含义的两个值：

HWND\_BROADCAST：消息被寄送到系统的所有顶层窗口，包括无效或不可见的非自身拥有的窗口、 被覆盖的窗口和弹出式窗口。消息不被寄送到子窗口

NULL：此函数的操作和调用参数dwThread设置为当前线程的标识符PostThreadMessage函数一样

Msg：指定被寄送的消息。

wParam：指定附加的消息特定的信息。

IParam：指定附加的消息特定的信息。

返回值：如果函数调用成功，返回非零，函数调用返回值为零

***PostQuitMessage 函数***

该函数向系统表明有个线程有终止请求。通常用来响应WM\_DESTROY消息。

void PostQuitMessage（

int nExitCode

）

参数：nExitCode：指定应用程序退出代码。此值被用作消息WM\_QUIT的wParam参数。

返回值：无。

PostQuitMessage寄送一个WM\_QUIT消息给线程的消息队列并立即返回；此函数向系统表明有个线程请求在随后的某一时间终止。

当线程从消息队列里取得WM\_QUIT消息时，应当退出消息循环并将控制返回给系统。返回给系统的退出值必须是消息WM\_QUIT的wParam参数。

速查：Windows NT：3.1及以上版本；Windows：95及以上版本；Windows CE：1.0及以上版本；头文件：winuser.h；输入库：user32.lib：Uhicode：在Wihdows NT环境下以Unicode和ANSI方式实现。

***QueryPerformanceCounter 函数***

函数QueryPerformanceCounter用于得到高精度计时器的值(如果存在这样的计时器)

BOOL QueryPerformanceCounter (

LARGE\_INTEGER \*ipPerformanceCount

);

LARGE\_INTEGER \*ipProformanceCount 为一个指针变量用于函数传值,即指向现时计数器的值.如果安装的硬件不支持高精度计时器,该参数将返回0,

返回值:如果安装的硬件支持高精度计时器,函数将返回非0值.如果安装的硬件不支持高精度计时器,函数将返回0.

***QueryPerformanceFrequency 函数***

<http://msdn.microsoft.com/en-us/library/ms644905(VS.85).aspx>

检查高精度计时器的频率。性能计数器的频率固定在系统引导并且所有处理器是相同的。因此，这个频率只需要在程序初始化时询问，载入到缓存中。

BOOL WINAPI QueryPerformanceFrequency (

\_Out\_  LARGE\_INTEGER \*lpFrequency

);

*lpFrequency* [out]

一个指针来接收正确的高精度计时器频率，每秒数量。如果安装的硬件不支持高精度计时器，这个参数将为0（在能运行windowsXP以上的系统中不会出现）。

返回值：硬件支持高精度计时器返回非零，不支持返回0.

***ReadFile 函数***

从文件指针指向的位置开始将数据读出到一个文件中， 且支持同步和异步操作，

如果文件打开方式没有指明FILE\_FLAG\_OVERLAPPED的话，当程序调用成功时，它将实际读出文件的字节数保存到lpNumberOfBytesRead指明的地址空间中。

FILE\_FLAG\_OVERLAPPED 允许对文件进行重叠操作

如果文件要交互使用的话，当函数调用完毕时要记得调整文件指针。

从文件中读出数据。与fread函数相比，这个函数要明显灵活的多。该函数能够操作通信设备、管道、套接字以及邮槽。

BOOL ReadFile (

HANDLE hFile, //文件的句柄

LPVOID lpBuffer, //用于保存读入数据的一个缓冲区

DWORD nNumberOfBytesToRead, //要读入的字节数

LPDWORD lpNumberOfBytesRead, //指向实际读取字节数的指针

LPOVERLAPPED lpOverlapped //如文件打开时指定了FILE\_FLAG\_OVERLAPPED，那么必须，用这个参数引用一个特殊的结构。 //该结构定义了一次异步读取操作。否则，应将这个参数设为NULL );

HANDLE hFile, 需要读入数据的文件指针,这个指针指向的文件必须是GENERIC\_READ 访问属性的文件。

LPVOID lpBuffer，接收数据的缓冲区。

DWORD nNumberOfBytesToRead，指定要读取的字节数。

LPDWORD lpNumberOfBytesRead，指向一个DWORD类型变量的指针，用来接收读取的字节数。如果下一个参数为NULL，那么一定要传入这个参数。

LPOVERLAPPED lpOverlapped OVERLAPPED结构体指针,如果文件是以FILE\_FLAG\_OVERLAPPED方式打开的话,那么这个指针就不能为NULL。

FILE\_FLAG\_OVERLAPPED 允许对文件进行重叠操作

文件或设备被打开或创建异步I / O。

当后续的I / O操作完成这个句柄，OVERLAPPED结构中指定的事件 将被设置为有信号状态。

如果这个标志被指定，该文件可用于同时读取和写入操作。

如果没有指定这个标志，然后被序列化I / O操作，即使调用读写函数指定一个OVERLAPPED结构

返回值 调用成功,返回非0；调用不成功,返回为0。会设置GetLastError。如启动的是一次异步读操作，则函数会返回零值，并将ERROR\_IO\_PENDING设置成GetLastError的结果。如结果不是零值，但读入的字节数小于nNumberOfBytesToRead参数指定的值，表明早已抵达了文件的结尾。

***Rect结构***

rect结构定义了一个矩形框左上角以及右下角的坐标，RECT结构通常用于Windows编程。

typedef struct \_RECT {

LONG left;

LONG top;

LONG right;

LONG bottom;

} RECT, \*PRECT;

left ： 指定矩形框左上角的x坐标

top： 指定矩形框左上角的y坐标

right： 指定矩形框右下角的x坐标

bottom：指定矩形框右下角的y坐标

备注：

当rect传递给fillrect函数时，该矩形框将被填充满，但不包括右栏和底行的像素。

该结构与rectl结构是同一结构。

***RegisterClass函数***

该函数注册在随后调用CreateWindow函数和CreateWindowEx函数中使用的窗口类。 RegisterClass函数己经由函数RegisterClassEx函数来代替，但是，如果不需要设置类的小目标则仍然可以使用RegisterClass函数。

ATOM RegisterClass( \_\_in CONST WNDCLASS \*lpWndClass);

lpWndClass：指向一个WNDCLASS结构的指针。在将它传递给函数之前，必须在该结构中填充适当的类属性。

如果函数成功，返回值是唯一标识已注册的类的一个原子；如果函数失败，返回值为0。若想获得更多错误信息，请调用GetLastError函数。

如果使用RegisterClassA来注册窗口，应用程序通知系统被注册类的窗口的消息使用ANSI字符集的文本和字符参数；如果使用RegisterClassW来注册窗口，应用程序需要系统以Unicode来传递消息的文本参数。lsWindowUnicode函数使应用程序可以查询每一个窗口的字符特征。参看Win32API中的ANSI和Unicode函数，请查阅Functiont prototype。

应用程序注册的所有的窗口类在应用程序中止后都为未注册的类。

Windows 95：所有由DLL注册的类在DLL卸载后均未注册的类。

Windows NT：所有由DLL注册的类在DLL卸载后仍为已注册的类。

Windows 95：如果WNDCLASSEX结构中的cbWndExtra或cbClsExtra单元包含字节数超过40个字节，则RegisterClassEx将失败。

Windows CE：由lpWndClass参数指向的WNDCLAS结构不支持lpszMenuName域，因为WindowsCE不支持缺省菜单。

除非使用了WindowsCE的lconcurs组件（这个组件提供了在适当目标平台上的鼠标支持），否则不能使用由lpWndClass指向的WNDCLASS结构中的hCursor域。

速查：Windows NT：3.1以上版本；Windows：95以上版本：WindowsCE：1.0以上版本；头文件：Winuser.h;库文件：user32.lib；Unicode：在Windows NT上实现为Unicodee和ANSI两种版本。

***RegisterClassEx函数***

该函数为随后在调用Createwindow函数和CreatewindowEx函数中使用的窗口注册一个窗口类。

ATOM RegisterClassEx（CONST WNDCLASSEX \*Ipwcx）；

***reinterpret\_cast 强制类型转换***

reinterpret\_cast是C++里的强制类型转换符。

reinterpret\_cast<type-id> (expression)

type-id 必须是一个指针、引用、算术类型、函数指针或者成员指针。它可以把一个指针转换成一个整数，也可以把一个整数转换成一个指针（先把一个指针转换成一个整数，再把该整数转换成原类型的指针，还可以得到原先的指针值）。

***SafeRelease 函数***

一些在这个文档中的实例代码使用这个函数来释放COM接口指针。

template <class T> void SafeRelease(T \*\*ppT)

{

if (\*ppT)

{

(\*ppT)->Release();

\*ppT = NULL;

}

}

这个函数不是在一个SDK的头文件中定义的。要使用这个函数，你必须用你自己的代码定义它。这个函数释放ppT的指针并将它设置为NULL。另一个选择是使用一个巧妙的指针函数，比如CComPtr，它在ATL中定义。

***SetWindowLong 函数（老）***

SetWindowLong是一个Windows API函数。该函数用来改变指定窗口的属性．函数也将指定的一个32位值设置在窗口的额外存储空间的指定偏移位置。

LONG SetWindowLong (

HWND hWnd,

int nlndex,

LONG dwNewLong

);

hWnd 窗口句柄及间接给出的窗口所属的类。

nlndex 指定将设定的大于等于0的偏移值。有效值的范围从0到额外类的存储空间的字节数减4：例如若指定了12或多于12个字节的额外窗口存储空间，则应设索引位8来访问第三个4字节，同样设置0访问第一个4字节，4访问第二个4字节。要设置其他任何值，可以指定下面值之一：

GWL\_EXSTYLE -20 设定一个新的扩展风格。

GWL\_HINSTANCE -6 设置一个新的应用程序实例句柄。

GWL\_ID -12 设置一个新的窗口标识符。

GWL\_STYLE -16 设定一个新的窗口风格。

GWL\_USERDATA -21 设置与窗口有关的32位值。每个窗口均有一个由创建该窗口的应用程序使用的32位值。

GWL\_WNDPROC -4 为窗口过程设定一个新的地址。

GWL\_HWNDPARENT -8 改变子窗口的父窗口,应使用SetParent函数。

当hWnd参数标识了一个对话框时，也可使用下列值：

DWL\_DLGPROC DWLP\_MSGRESULT + sizeof(LRESULT) (值:4) 设置对话框过程的新地址。

DWL\_MSGRESULT 0 设置在对话框过程中处理的消息的返回值。

DWL\_USER DWLP\_DLGPROC + sizeof(DLGPROC) (值:8) 设置的应用程序私有的新的额外信息，例如一个句柄或指针。

dwNewLong 指定的替换值。

dwNewLong中可以使用以下一些列表控件的专用风格：

LVS\_ICON LVS\_SMALLICON LVS\_LIST LVS\_REPORT

这四种风格决定控件的外观，同时只可以选择其中一种，分别对应：

大图标显示，小图标显示，列表显示，详细报表显示

LVS\_EDITLABELS 结点的显示字符可以被编辑，对于报表风格来讲可编辑的只为第一列。

LVS\_SHOWSELALWAYS 在失去焦点时也显示当前选中的结点

LVS\_SINGLESEL 同时只能选中列表中一项

返回值 编辑如果函数成功，返回值是指定的32位整数的原来的值。如果函数失败，返回值为0。若想获得更多错误信息，请调用GetLastError函数。

如果指定32位整数的原来的值为0，并且函数成功，则返回值为0，但是函数并不清除最后的错误信息，这就很难判断函数是否成功。这时，就应在调用SetWindowLong之前调用callingSetLastError（0）函数来清除最后的错误信息。这样，如果函数失败就会返回0，并且GetLastError。也返回一个非零值。

***SendDlgItemMessage 函数***

该函数把一个消息发送给指定的对话框中的控件。

LONG SendDlgItemMessage (

HWND hDlg,

int nIDDlgItem,

UINT Msg,

WPARAM wParam,

LPARAM lParam

);

hDlg:指定含有控件的对话框。

nIDDigItem:指定接收消息的控件的标识符。

Msg：指定将被发送的消息。

wParam:指定消息特定的其他信息。

lParam:指定消息特定的其他信息。

返回值：返回值指定消息处理的结果，且依赖于发送的消息。

备注：SendDlgItemMessage函数直到消息已经被处理时才返回。

使用SendDlgItemMessage函数同从一个指定的控件中检索句柄和调用SendMessagge函数一样。

***ShowWindow 函数***

该函数设置指定窗口的显示状态。

BOOL ShowWindow （

HWND hWnd,

int nCmdShow ）；

hWnd:指窗口句柄。

nCmdShow：指定窗口如何显示。如果发送应用程序的程序提供了STARTUPINFO结构，则应用程序第一次调用ShowWindow时该参数被忽略。否则，在第一次调用ShowWindow函数时，该值应为在函数WinMain中nCmdShow参数。在随后的调用中，该参数可以为下列值之一：

SW\_FORCEMINIMIZE：在WindowNT5.0中最小化窗口，即使拥有窗口的线程被挂起也会最小化。在从其他线程最小化窗口时才使用这个参数。

SW\_HIDE：隐藏窗口并激活其他窗口。

SW\_MAXIMIZE：最大化指定的窗口。

SW\_MINIMIZE：最小化指定的窗口并且激活在Z序中的下一个顶层窗口。

SW\_RESTORE：激活并显示窗口。如果窗口最小化或最大化，则系统将窗口恢复到原来的尺寸和位置。在恢复最小化窗口时，应用程序应该指定这个标志。

SW\_SHOW：在窗口原来的位置以原来的尺寸激活和显示窗口。

SW\_SHOWDEFAULT：依据在STARTUPINFO结构中指定的SW\_FLAG标志设定显示状态，STARTUPINFO 结构是由启动应用程序的程序传递给CreateProcess函数的。

SW\_SHOWMAXIMIZED：激活窗口并将其最大化。

SW\_SHOWMINIMIZED：激活窗口并将其最小化。

SW\_SHOWMINNOACTIVE：窗口最小化，激活窗口仍然维持激活状态。

SW\_SHOWNA：以窗口原来的状态显示窗口。激活窗口仍然维持激活状态。

SW\_SHOWNOACTIVATE：以窗口最近一次的大小和状态显示窗口。激活窗口仍然维持激活状态。

SW\_SHOWNORMAL：激活并显示一个窗口。如果窗口被最小化或最大化，系统将其恢复到原来的尺寸和大小。应用程序在第一次显示窗口的时候应该指定此标志。

应用程序第一次调用ShowWindow时，应该使用WinMain函数的nCmdshow参数作为它的nCmdShow参数。在随后调用ShowWindow函数时，必须使用列表中的一个给定值，而不是由WinMain函数的nCmdSHow参数指定的值。

正如在nCmdShow参数中声明的，如果调用应用程序的程序使用了在STARTUPINFO结构中指定的信息来显示窗口，则在第一次调用ShowWindow函数时nCmdShow参数就被忽略。在这种情况下，ShowWindow函数使用STARTUPINFO结构中的信息来显示窗口。在随后的调用中，应用程序必须调用ShowWindow 函数（将其中nCmdShow参数设为SW\_SHOWDEFAULT）来使用由程序调用该应用程序时提供的启动信息。这个处理在下列情况下发生：

应用程序通过调用带WS\_VISIBLE标志的函数来创建它们的主窗口函数；

应用程序通过调用清除了WS\_VISIBLE标志的CreateWindow函数来创建主窗口函数，并且随后调用带SW\_SHOW标志的ShowWindow函数来显示窗口；

Windows CE：nCmdShow参数不支持下列值：

SW\_MAXIMINZE；SW\_MINIMIZE；SW\_RESTORE；SW\_SHOWDEFAULT

SW\_SHOWMAXIMIZED；SW\_SHOWMINIMIZED；SW\_SHOWMININOACTIVATE

速查：Windows NT：3.1以上版本；Windows：95以上版本：Windows CE：1.0以上版本；头文件：winuw库文件：user32.lib。

***sprintf 函数***

字符串格式化命令，主要功能是把格式化的数据写入某个字符串中。

int sprintf (

char \*buffer,

const char \*format,

[ argument] …

);

buffer：char型指针，指向将要写入的字符串的缓冲区。

format：格式化字符串。

[argument]...：可选参数，可以是任何类型的数据。

返回值：字符串长度（strlen）

***TCHAR 宏***

TCHAR是通过define定义的字符串宏。

因为C++支持两种字符串，即常规的ANSI编码（使用""包裹）和Unicode编码（使用L""包裹），这样对应的就有了两套字符串处理函数，比如：strlen和wcslen，分别用于处理两种字符串。

微软将这两套字符集及其操作进行了统一，通过条件编译（通过\_UNICODE和UNICODE宏）控制实际使用的字符集，这样就有了\_T("")这样的字符串，对应的就有了\_tcslen这样的函数

为了存储这样的通用字符，就有了TCHAR：

当没有定义\_UNICODE宏时，TCHAR = char，\_tcslen =strlen

当定义了\_UNICODE宏时，TCHAR = wchar\_t ， \_tcslen = wcslen[1]

当我们定义了UNICODE宏，就相当于告诉了编译器：我准备采用UNICODE版本。这个时候，TCHAR就会摇身一变，变成了wchar\_t。而未定义UNICODE宏时，TCHAR摇身一变，变成了unsignedchar。这样就可以很好的切换宽窄字符集。

tchar可用于双字节字符串，使程序可以用于中日韩等国 语言文字处理、显示。使编程方法简化。

***TranslateAccelerator 函数***

TranslateAccelerator，函数功能：翻译加速键表。该函数处理菜单命令中的加速键。该函数将一个WM\_KEYDOWN或WM\_SYSKEYDOWN消息翻译成一个WM\_COMMAND或WM\_SYSCOMMAND消息（如果在给定的加速键表中有该键的入口），然后将WM\_COMMAND或WM\_SYSCOMMAND消息直接送到相应的窗口处理过程。

TranslateAccelerator直到窗口过程处理完消息后才返回。

int TranslateAccelerator（

HWND hWnd,

HACCEL hAccTable，

LPMSG IpMsg

）;

hWnd:窗口句柄，该窗口的消息将被翻译。

hAccTable:加速键表句柄。加速键表必须由LoadAccelerators函数调用装入或由CreateAccd\_eratorTable函数调用创建。

LpMsg:MSG结构指针，MSG结构中包含了从使用GetMessage或PeekMessage函数调用线程消息队列中得到的消息内容。

返回值：若函数调用成功，则返回非零值；若函数调用失败，则返回值为零。若要获得更多的错误信息，可调用GetLastError函数。

为了将该函数发送的消息与菜单或控制发送的消息区别开来，使WM\_COMMAND或WM\_SYSCOMMAND消息的wParam参数的高位字值为1。用于从窗口菜单中选择菜单项的加速键组合被翻译成WM\_SYSCOMMAND消息：所有其他的加速键组合被翻译成WM\_COMMAND。若TransLateAccelerator返回非零值且消息已被翻译，应用程序就不能调用TranslateMessage函数对消息再做处理。每个加速键不一定都对应于菜单命令。若加速键命令对应于菜单项，则WM\_INITMENU和WM\_INITMENUPOPUP消息将被发送到应用程序，就好像用户正试图显示该菜单。然而，如下的任一条件成立时，这些消息将不被发送：窗口被禁止，菜单项被禁止。

***TranslateMessage 函数***

TranslateMessage函数用于将虚拟键消息转换为字符消息。该函数将虚拟键消息转换为字符消息。字符消息被寄送到调用线程的消息队列里，当下一次线程调用函数GetMessage或PeekMessage时被读出。

BOOL TranslateMessage(

CONST MSG \*lpMsg

);

IpMsg：指向含有消息的MSG结构的指针，该结构里含有用函数GetMessage或PeekMessage从调用线程的消息队列里取得的消息信息。

返回值：如果消息被转换（即，字符消息被寄送到调用线程的消息队列里），返回非零值。如果消息是WM\_KEYDOWN，WM\_KEYUP WM\_SYSKEYDOWN或WM\_SYSKEYUP，返回非零值，不考虑转换。如果消息没被转换（即，字符消息没被寄送到调用线程的消息队列里），返回值是零。

备注：此函数不修改由参数IpMsg指向的消息。

WM\_KEYDOWN和WM\_KEYUP组合产生一个WM\_CHAR或WM\_DEADCHAR消息。

WM\_SYSKEYDOWN和WM\_SYSKEYUP组合产生一个WM\_SYSCHAR或 WM\_SYSDEADCHAR消息。TranslateMessage为那些由键盘驱动器映射为ASCll字符的键产生WM\_CHAR消息。

如果应用程序为其他用途处理虚拟键消息，不应调用TranslateMessage。例如，如果件TranslateAccelerator返回一个非零值，应用程序不应调用TranslateMessage。

Windows CE：Windows CE不支持扫描码或扩展键标志，因此，不支持由TranslateMessage产生的WM\_CHAR消息中的IKeyData参数（IParam）取值16-24。

TranslateMessage只能用于转换调用GetMessage或PeekMessage接收的消息。

***UNREFERENCED\_PARAMETER 宏***

避免出现未引用的警告。

***UpdateWindow 函数***

如果窗口更新的区域不为空，UpdateWindow函数通过发送一个WM\_PAINT消息来更新指定窗口的客户区。函数绕过应用程序的消息队列，直接发送WM\_PAINT消息给指定窗口的窗口过程，如果更新区域为空，则不发送消息。

BOOL UpdateWindow(

HWND hWnd // 窗口的句柄

);

hWnd [输入] 要更新的窗口的句柄.

返回值：编辑如果函数调用成功，返回值为非零值。如果函数调用不成功，返回值为零。

在Windows NT/2000/XP中，我们可以使用API 函数 GetLastError 来得到扩展的错误信息。

使用要求：编辑Windows NT/2000/XP： 包括 Windows NT 3.1 及以后版本。

Windows 95/98/Me： 包括 Windows 95 及以后版本。

需要的头文件： 需要包含 Windows.h，在 Winuser.h 中声明了此函数。

需要的库文件: Use User32.lib。

***WaitForSingleObject 函数***

当等待仍在挂起状态时，句柄被关闭，那么函数行为是未定义的。该句柄必须具有 SYNCHRONIZE 访问权限。

DWORD WaitForSingleObject (

HANDLE hHandle,

DWORD dwMilliseconds

);

hHandle[in]对象句柄。可以指定一系列的对象，如Event、Job、Memory resource notification、Mutex、Process、Semaphore、Thread、Waitable timer等。

dwMilliseconds[in]定时时间间隔，单位为milliseconds（毫秒）.如果指定一个非零值，函数处于等待状态直到hHandle标记的对象被触发，或者时间到了。如果dwMilliseconds为0，对象没有被触发信号，函数不会进入一个等待状态，它总是立即返回。如果dwMilliseconds为INFINITE，对象被触发信号后，函数才会返回。

返回值：

WAIT\_ABANDONED 0x00000080：当hHandle为mutex时，如果拥有mutex的线程在结束时没有释放核心对象会引发此返回值。

WAIT\_OBJECT\_0 0x00000000 ：核心对象已被激活

WAIT\_TIMEOUT 0x00000102：等待超时

WAIT\_FAILED 0xFFFFFFFF ：出现错误，可通过GetLastError得到错误代码

***Winmain 函数***

WinMain是一个函数，该函数的功能是被系统调用，作为一个32位应用程序的入口点。WinMain函数应初始化应用程序，显示主窗口，进入一个消息接收一发送循环，这个循环是应用程序执行的其余部分的顶级控制结构。

|  |  |
| --- | --- |
| int WINAPI WinMain  {  HINSTANCE hInstance;  HINSTANCE hPrevInstance;  LPSTR lpCmdLine;  int nCmdShow;  } | |
| 1  2 | 在 Visual Studio 2005/2013 下则为int APIENTRY WinMain...  在 Visual Studio 2012 下则为int CALLBACK WinMain... |

hInstance：[应用程序](http://baike.baidu.com/view/330120.htm)当前实例的句柄。

hPrevlnstance：[应用程序](http://baike.baidu.com/view/330120.htm)的先前实例的句柄。对于同一个程序打开两次，出现两个窗口第一次打开的窗口就是先前实例的窗口。对于一个32位程序，该参数总为NULL。

如果需要检测另外一个实例是否已经存在，则使用CreateMutex函数创建一个独一无二的名字。即使互斥名已经存在，CreateMutex函数也是成功的，但是GetLastError函数将返回 ERROR\_ALREADY\_EXISTS，这就表明[应用程序](http://baike.baidu.com/view/330120.htm)有另外一个实例存在，因为它首先创建了互斥名。

lpCmdLine：指向[应用程序](http://baike.baidu.com/view/330120.htm)命令行的字符串的[指针](http://baike.baidu.com/view/159417.htm)，不包括执行文件名。获得整个命令行，参看GetCommandLine。

第三个参数lpCmdLine是一个以空终止的字符串，指定传递给[应用程序](http://baike.baidu.com/view/330120.htm)的[命令行参数](http://baike.baidu.com/view/1646320.htm)。例如：在D盘下有一个sunxin.txt文件，当我们用鼠标双击这个文件时将启动[记事本](http://baike.baidu.com/view/152865.htm)程序（notepad.exe），此时系统会将D:\sunxin.txt作为[命令行参数](http://baike.baidu.com/view/1646320.htm)传递给记事本程序的WinMain函数，记事本程序在得到这个文件的全路径名后，就在窗口中显示该文件的内容。要在VC++[开发环境](http://baike.baidu.com/view/4831305.htm)中向[应用程序](http://baike.baidu.com/view/330120.htm)传递参数，可以单击[菜单](http://baike.baidu.com/view/102616.htm)【Project】→【Settings】，选择“Debug”选项卡，在“Program arguments”编辑框中输入你想传递给应用程序的参数。

nCmdShow：指明窗口如何显示。该参数可以是下列值之一：

SW\_HIDE：隐藏窗口并且激活另外一个窗口。

SW\_MINIMIZE：最小化指定的窗口，并且激活在系统表中的顶层窗口。

SW\_RESTORE：激活并显示窗口。如果窗口已经最小化或最大化，系统将以恢复到原来的尺寸和位置显示窗口（与SW\_SHOWNORMAL相同）。

SW\_SHOW：激活一个窗口并以原来的尺寸和位置显示窗口。

SW\_SHOWMAXIMIZED：激活窗口并且将其最大化。

SW\_SHOWMINIMIZED：激活窗口并将其最小化（以图标显示）。

SW\_SHOWMINNOACTIVE：将一个窗口显示为图标。激活窗口维持活动状态。

SW\_SHOWNA：以窗口的当前状态显示窗口。激活窗口保持活动状态。

SW\_SHOWNOACTIVATE：以窗口的最近一次的尺寸和位置显示窗口。激活窗口维持激活状态。

SW\_SHOWNORMAL：激活并显示窗口。如果窗口最大化或最小化，系统将其恢复到原来的尺寸和位置（与SW\_RESTORE相同）。

如果函数成功，当它接收到一个WM\_QUIT消息时就中止，函数应该返回在该消息的wParam参数的退出值。如果函数在进入[消息循环](http://baike.baidu.com/view/3068833.htm)前退出，应该返回零

当接收到一个WM\_QUIT消息时，程序就中止。这时，WinMain函数应退出[应用程序](http://baike.baidu.com/view/330120.htm)，并且返回传递给WM\_QUIT消息的wParam参数的值。如果由于调用PostQuitMessage函数而接收到WM\_QUIT消息，wParam的值是PostQuitMessage函数的nExitCode的值。请参看“创建一个窗口循环”。

ANSI[应用程序](http://baike.baidu.com/view/330120.htm)可以使用WinMain函数的lpCmdLine参数进入命令行字符串（除了程序名之外）。WinMain不能返回Unicode字符串的原因是IpCmdLine使用的是LPSTR[数据类型](http://baike.baidu.com/view/675645.htm)，而不是LPTSTR类型。GetCommandLine函数可以用于进入命令行的Unicode字符串，因为它使用的是LPTSTR类型。

Windows CE：Windows CE不支持下列 nCmdLine参数值：

SW\_MINIMIZE；SW\_RESTORE；SW\_RESTORE；SW\_SHOWMAXMIZED

SW\_SHOWMINIMIZED；SW\_SHOWMINNOACTIVE

***WNDCLASS 结构（老）***

结构WNDCLASS包含一个窗口类的全部信息，也是Windows编程中使用的基本数据结构之一，应用程序通过定义一个窗口类确定窗口的属性

typedef struct \_WNDCLASS

{

UINT style;// 窗口类型

WNDPROC lpfnWndProc;//窗口处理函数

int cbClsExtra;//窗口扩展

int cbWndExtra;//窗口实例扩展

HINSTANCE hInstance;//实例句柄

HICON hIcon;//窗口的最小化图标

HCURSOR hCursor;//窗口鼠标光标

HBRUSH hbrBackground;//窗口背景色

LPCTSTR lpszMenuName;//窗口菜单

LPCTSTR lpszClassName;// 窗口类名

} WNDCLASS, \*LPWNDCLASS;

style:指定类风格。这些风格可通过按位或操作组合起来。风格如下:

CS\_BYTEALIGNCLIENT: 在字节边界上（在x方向上）定位窗口的用户区域的位置

CS\_BYTEALIGNWINDOW: 在字节边界上（在x方向上）定位窗口的位置

CS\_CLASSDC: 该窗口类的所有窗口实例都共享一个窗口类DC

CS\_DBLCLKS: 允许向窗口发送双击鼠标键的消息

CS\_GLOBALCLASS: 当调用CreateWindow 或 CreateWindowEx 函数来创建窗口时允许它的hInstance参数和注册窗口类时传递给RegisterClass 的 hInstance参数不同。如果不指定该风格，则这两个 hInstance 必须相同。

CS\_HREDRAW: 当水平长度改变或移动窗口时，重画整个窗口

CS\_NOCLOSE: 禁止系统菜单的关闭选项

CS\_OWNDC: 给予每个窗口实例它本身的DC。注意，尽管这样是很方便，但它必须慎重使用，因为每个DC大约要占800个字节的内存。

CS\_PARENTDC: 将子窗口的裁剪区域设置到父窗口的DC中去，这样子窗口便可以在父窗口上绘制自身。注意，这是子窗口还是从系统缓存中获取DC，而不是使用父窗口的DC。使用该风格可以提高系统性能。

CS\_SAVEBITS: 以位图形式保存被该窗口遮挡的屏幕部分，这样当给窗口移动以后，系统便可以用该保存的位图恢复屏幕移动的相应部分，从而系统不用向被该窗口遮挡的窗口发送 WM\_PAINT 消息。该特性对于菜单类型的窗口比较合适，因为它通常是简短的显示一下之后便消失。设置该特性将增加显示该窗口的时间，因为它通常要先分配保存位图的内存。

CS\_VREDRAW: 当垂直长度改变或移动窗口时，重画整个窗口

lpfnWndProc: 指向窗口过程

cbClsExtra: 指定紧随在 WNDCLASS 数据结构后分配的字节数。系统将其初始化为零。

cbWndExtra: 指定紧随在窗口实例之后分配的字节数，系统将其初始化为零。如果应用程序正在用WNDCLASS结构注册一个在RC资源描述文件中用CLASS指令创建的对话框时，它必须设置这个字段为 DLGWINDOWEXTRA。

hInstance: 标识了该窗口类的窗口过程所在的模块实例的句柄，不能为NULL。

hIcon: 标识了该窗口类的图标。hIcon字段必须是一个图标的句柄；若hIcon字段为NULL，那么系统将提供一个默认的图标。

hCursor: 标识该窗口类的光标，hCursor必须是一个光标资源的句柄。若hCursor字段为NULL，则无论何时鼠标移到应用程序窗口时，应用程序必须显式设置光标形状。

hbrBackground: 标识了该窗口类的背景画刷。hbrBackground字段必须是用于绘制背景的物理刷子的句柄，或者是一个颜色的值。如果给出一个颜色的值，它必须是下面列出的标准系统颜色之一（系统将对所选颜色加1）。如果给出了颜色值，它必须是转换成下列的HBRUSH类型之一的颜色:

COLOR\_ACTIVEBORDER

COLOR\_ACTIVECAPTION

COLOR\_APPWORKSPACE

COLOR\_BACKGROUND

COLOR\_BTNFACE

COLOR\_BTHSHADOW

COLOR\_BTNTEXT

COLOR\_CAPTIONTEXT

COLOR\_GRAYTEXT

COLOR\_HIGHLIGHT

COLOR\_HIGHLIGHTTEXT

COLOR\_INACTIVEBORDER

COLOR\_INACTIVECAPTION

COLOR\_MENU

COLOR\_MENUTEXT

COLOR\_SCROLLBAR

COLOR\_WINDOW

COLOR\_WINDOWFRAME

COLOR\_WINDOWTEXT

当hbrBackground字段为NULL时，每当需要绘制其用户区域时，应用程序必须自己来绘制其背景。应用程序可以通过处理WM\_ERASEBKGND 消息或检查由 BeginPaint 函数填写的 PAINTSTRUCT 结构的fErase 字段来确定背景什么时候需要着色。

lpszMenuName :指向NULL结束的字符串，该字符串描述菜单的资源名，如同在资源文件里显示的名字一样。若使用一个整数标识菜单，可以使用MAKEINTRESOURCE宏。如果lpszMenuName为NULL，那么该窗口类的窗口将没有默认菜单。

lpszClassName :指向NULL结束的字符串，或者是一个原型(atom)。若该参数是一个原型，它必须是一个有先前调用RegisterClass或者 RegisterClassEx函数产生的类原型。类原型必须作为lpszClassName的低字，高字必须为0.若lpszClassName是一个字符串，它描述了窗口类名。这个类名可以是由RegisterClass或RegisterClassEx注册的名字，或者是任何预定义的控件类名。

***WNDCLASSEX***

WNDCLASSEX属于一个窗台类。

WNDCLASSEX 中最重要的成员莫过于lpfnWndProc了。前缀 lpfn 表示该成员是一个指向函数的长指针。在 Win32中由于内存模式是 FLAT 型，所以没有 near 或 far 的区别。每一个窗口类必须有一个窗口过程，当 Windows 把属于特定窗口的消息发送给该窗口时，该窗口的窗口类负责处理所有的消息，如键盘消息或鼠标消息。由于窗口过程差不多智能地处理了所有的窗口消息循环，所以您只要在其中加入消息处理过程即可。

typedef struct WNDCLASSEX {

UINT cbSize;

UINT style;

WNDPROC lpfnWndProc;

int cbClsExtra;

int cbWndExtra;

HINSTANCE hInstance;

HICON hIcon;

HCURSOR hCursor;

HBRUSH hbrBackground;

LPCTSTR lpszMenuName;

LPCTSTR lpszClassName;

HICON hIconSm;

} WNDCLASSEX, \*PWNDCLASSEX;

cbSize：WNDCLASSEX 的大小。我们可以用sizeof（WNDCLASSEX）来获得准确的值。

style：从这个窗口类派生的窗口具有的风格。您可以用“or”操作符来把几个风格或到一起。

lpfnWndProc：窗口处理函数的指针。（回调函数）

cbClsExtra：指定紧跟在窗口类结构后的附加字节数。

cbWndExtra：指定紧跟在窗口实例的附加字节数。如果一个应用程序在资源中用CLASS伪指令注册一个对话框类时，则必须把这个成员设成DLGWINDOWEXTRA。

hInstance：本模块的实例句柄。

hIcon：图标的句柄。

hCursor：光标的句柄。

hbrBackground：背景画刷的句柄。

pszMenuName：指向菜单的指针。

lpszClassName：指向类名称的指针。

hIconSm：和窗口类关联的小图标。如果该值为NULL。则把hIcon中的图标转换成大小合适的小图标。

***WriteFile函数***

从文件指针指向的位置开始将数据写入到一个文件中, 且支持同步和异步操作,

如果文件打开方式没有指明FILE\_FLAG\_OVERLAPPED的话,当程序调用成功时,它将实际写入文件的字节数保存到lpNumberOfBytesWriten指明的地址空间中

如果文件要交互使用的话,当函数调用完毕时要记得调整文件指针

BOOL WriteFile (

HANDL EhFile, //文件句柄

LPCVOID lpBuffer, //数据缓存区指针

DWOR DnNumberOfBytesToWrite, //你要写的字节数

LPDWORD lpNumberOfBytesWritten, //用于保存实际写入字节数的存储区域的指针

LPOVERLAPPED lpOverlapped //OVERLAPPED结构体指针

);

hFile Long，一个文件的句柄

lpBuffer Any，参数类型:指针,指向将写入文件的 数据缓冲区

nNumberOfBytesToWrite Long，要写入数据的字节数量。如写入零字节，表示什么都不写入，但会更新文件的“上一次修改时间”。针对位于远程系统的命名管道，限制在65535个字节以内

lpNumberOfBytesWritten Long，实际写入文件的字节数量（此变量是用来返回的 ）

lpOverlapped OVERLAPPED，倘若在指定FILE\_FLAG\_OVERLAPPED的前提下打开文件，这个参数就必须引用一个特殊的结构。那个结构定义了一次异步写操作。否则，该参数应置为空（将声明变为ByVal As Long，并传递零值）

***ZeroMemory 宏***

ZeroMemory，是美国微软公司的软件开发包SDK中的一个宏。 其作用是用0来填充一块内存区域。ZeroMemory只是将指定的内存块清零。使用结构前清零，而不让结构的成员数值具有不确定性，是一个好的编程习惯。

void ZeroMemory(

PVOID Destination,

SIZE\_T Length

);

Destination :指向一块准备用0来填充的内存区域的开始地址。

Length :准备用0来填充的内存区域的大小，按字节来计算。

返回值：无