**深入理解windows 消息机制**

深入理解Windows消息机制

       今天我们来学一学Windows消息机制，我们知道在传统的C语音程序中，当我们需要打开一个文件时，我们可以调用fopen()函数，这个函数最后又会调用操作系统提供的函数以此来打开文件。而在Windows编程中，不仅用户可以调用系统的API函数，反之，系统也可以调用应用程序，而这些调用就是通过Windows的消息机制来实现的。Windows程序设计是一种完全不同于传统的DOS方式的程序设计方法，它是一种事件驱动的程序设计模式，主要是基于消息的。

**一、消息的定义？**

       消息系统对于一个win32程序来说十分重要，它是一个程序运行的动力源泉。一个消息，是系统定义的一个32位的值，他唯一的定义了一个事件，向 Windows发出一个通知，告诉应用程序某个事情发生了。例如，单击鼠标、改变窗口尺寸、按下键盘上的一个键都会使Windows发送一个消息给应用程序的消息队列（下面会讲到）中，然后应用程序再从消息队列中取出消息并进行相应的响应。在这个处理的过程中，操作系统也会给应用程序“发送消息”，而所谓的发送消息--------实际上就是操作系统调用程序中的一个专门负责处理消息的函数，这个函数称为"**窗口过程**"。  
        消息本身是作为一个记录传递给应用程序的，这个记录中包含了消息的类型以及其他信息。例如，对于单击鼠标所产生的消息来说，这个记录中包含了单击鼠标时的坐标。这个记录类型叫做MSG，MSG含有来自windows应用程序消息队列的消息信息，在Windows中MSG结构体定义如下：

typedef struct tagMsg

{

       HWND    hwnd;       //接受该消息的窗口句柄

       UINT    message;    //消息常量标识符，也就是我们通常所说的消息号

       WPARAM  wParam;     //32位消息的特定附加信息，确切含义依赖于消息值

       LPARAM  lParam;     //32位消息的特定附加信息，确切含义依赖于消息值

       DWORD   time;       //消息创建时的时间

       POINT   pt;         //消息创建时的鼠标/光标在屏幕坐标系中的位置

}MSG;

**二、消息队列的定义**

        在Windows编程中，每一个Windows应用程序开始执行后，系统都会为该程序创建一个消息队列，这个消息队列用来存放该应用程序所创建的窗口的信息。例如，当我们按下鼠标右键的时候，这时会产生一个WM\_RBUTTONDOWN消息，系统会自动将这个消息放进当前窗口所属的应用程序的消息队列中，等待应用程序的结束。Windows将产生的消息以此放进消息队列中，应用程序则通过一个消息循环不断的从该消息队列中读取消息，并做出响应（后面会详细讲述消息处理过程..）

**三、消息中的家庭成员**

        通过前面所罗列的MSG结构体，我们是不是会对消息结构里边包含的元素有了一个比较清楚的认识呢？如果还没有，没关系！！呵呵，那么现在再次对那些咋一看就会泪奔的变量做出详细的解释：  
        **hwnd** ：一个窗口句柄（具体位数跟个人PC操作系统位数相关），它表示的是消息所属的窗口。我们通常开发的程序都是窗口应用程序，一般一个消息都是和某个窗口相关联的。比如我们在某个活动窗口按下鼠标右键，此时产生的消息就是发送给该活动窗口的。窗口可以是任何类型的屏幕对象，因为Win32能够维护大多数可视对象的句柄(窗口、对话框、按钮、编辑框等)。补充一下：“句柄”---在Windows程序中，有各种各样的资源，系统在创建这些资源的时候，都会为他们分配内存，并返回标识这些资源的标识号，这个标识号就是句柄）。  
       **message：**一个消息的标识符，用于区别其他消息的常量值，这些常量可以是Windows单元中预定义的常量，也可以是自定义的常量。在Windows中消息是由一个数值表示的，不同的消息对应不同的数值。但由于当这些消息种类多到足以挑战我们的IQ，所以聪明的程序开发者便想到将这些数值定义为WM\_XXX宏的形式。例如，鼠标左键按下的消息--WM\_LBUTTONDOWN，键盘按下消息--WM\_KEYDOWN，字符消息--WM\_CHAR，等等。。。。消息标识符以常量命名的方式指出消息的含义。当窗口过程接收到消息之后，他就会使用消息标识符来决定如何处理消息。例如、WM\_PAINT告诉窗口过程窗体客户区被改变了需要重绘。符号常量指定系统消息属于的类别，其前缀指明了处理解释消息的窗体的类型。  
        **wParam和lParam**：用于指定消息的附加信息。例如，当我们收到一个键盘按下消息的时候，message成员变量的值就是WM\_KEYDOWN，但是用户到底按下的是哪一个按键，我们就得拜托这二位，由他们来告知我们具体的信息。

**time和pt**：这俩个变量分别被用来表示消息投递到消息队列中的时间和鼠标当前的位置，一般情况下不怎么使用（但不代表没用）

**四、消息标识符**

        系统保留消息标识符的值在0x0000在0x03ff(WM\_USER-1)范围。这些值被系统定义消息使用。应用程序不能使用这些值给自己的消息。应用程序消息从WM\_USER（0X0400）到0X7FFF，或0XC000到0XFFFF；WM\_USER到 0X7FFF范围的消息由应用程序自己使用；0XC000到0XFFFF范围的消息用来和其他应用程序通信，在此只是罗列一些具有标志性的消息值：

1. WM\_NULL---0x0000    空消息。
2. 0x0001----0x0087    主要是窗口消息。
3. 0x00A0----0x00A9    非客户区消息
4. 0x0100----0x0108    键盘消息
5. 0x0111----0x0126    菜单消息
6. 0x0132----0x0138    颜色控制消息
7. 0x0200----0x020A    鼠标消息
8. 0x0211----0x0213    菜单循环消息
9. 0x0220----0x0230    多文档消息
10. 0x03E0----0x03E8    DDE消息
11. 0x0400             WM\_USER
12. 0x8000             WM\_APP
13. 0x0400----0x7FFF    应用程序自定义私有消息

**五、消息的分类**

        windows中的消息虽然很多，但是种类并不繁杂，大体上有3种： 窗口消息、命令消息、         控件通知消息。  
    **(1)   窗口消息**- - - -大概是系统中最为常见的消息，它是指由操作系统和控制其他窗口的窗口所使用的消息。例如CreateWindow、DestroyWindow和MoveWindow等都会激发窗口消息，还有我们在上面谈到的单击鼠标所产生的消息也是一种窗口消息。  
      **(2） 命令消息**- - - - 这是一种特殊的窗口消息，他用来处理从一个窗口发送到另一个窗口的用户请求，例如按下一个按钮，他就会向主窗口发送一个命令消息。  
       **(3)   控件通知消息**- - - 其实它是这样滴，当一个窗口内的子控件发生了一些事情，而这些是需要通知父窗口的，此刻它就上场啦。通知消息只适用于标准的窗口控件如按钮、列表框、组合框、编辑框，以及Windows公共控件如树状视图、列表视图等。

        例如，单击或双击一个控件、在控件中选择部分文本、操作控件的滚动条都会产生通知消息（她类似于命令消息），那么控件通知消息就会从控件窗口发送到它的主窗口。但是这种消息的存在并不是为了处理用户命令，而是为了让主窗口能够改变控件，例如加载、显示数据。

        再例如，按下一个按钮，他向父窗口发送的消息也可以看作是一个控件通知消息；单击鼠标所产生的消息可以由主窗口直接处理，然后交给控件窗口处理。其中窗口消息及控件通知消息主要由窗口类即直接或间接由CWND类派生类处理。相对窗口消息及控件通知消息而言，命令消息的处理对象范围就广得多，它不仅可以由窗口类处理，还可以由文挡类，文档模板类及应用类所处理。  
       由于控件通知消息很重要的，编程者用的也比较多，但是具体的含义往往令初学者晕头转向，所以我决定把常见的几个列出来供大家参考：

* *按扭控件*

*BN\_CLICKED        用户单击了按钮  
 BN\_DISABLE 按钮被禁止  
 BN\_DOUBLECLICKED  用户双击了按钮  
 BN\_HILITE  用/户加亮了按钮  
 BN\_PAINT  按钮应当重画  
 BN\_UNHILITE 加亮应当去掉*

* *组合框控件*

*CBN\_CLOSEUP 组合框的列表框被关闭  
 CBN\_DBLCLK 用户双击了一个字符串  
 CBN\_DROPDOWN 组合框的列表框被拉出  
 CBN\_EDITCHANGE 用户修改了编辑框中的文本  
 CBN\_EDITUPDATE 编辑框内的文本即将更新  
 CBN\_ERRSPACE 组合框内存不足  
 CBN\_KILLFOCUS 组合框失去输入焦点  
 CBN\_SELCHANGE 在组合框中选择了一项  
 CBN\_SELENDCANCEL 用户的选择应当被取消  
 CBN\_SELENDOK 用户的选择是合法的  
 CBN\_SETFOCUS 组合框获得输入焦点*

* *编辑框控件*

*EN\_CHANGE 编辑框中的文本己更新  
 EN\_ERRSPACE 编辑框内存不足  
 EN\_HSCROLL 用户点击了水平滚动条  
 EN\_KILLFOCUS 编辑框正在失去输入焦点  
 EN\_MAXTEXT 插入的内容被截断  
 EN\_SETFOCUS 编辑框获得输入焦点  
 EN\_UPDATE 编辑框中的文本将要更新  
 EN\_VSCROLL 用户点击了垂直滚动条消息含义*

* *列表框控件*

*LBN\_DBLCLK 用户双击了一项  
 LBN\_ERRSPACE 列表框内存不够  
 LBN\_KILLFOCUS 列表框正在失去输入焦点  
 LBN\_SELCANCEL 选择被取消  
 LBN\_SELCHANGE 选择了另一项  
 LBN\_SETFOCUS 列表框获得输入焦点*

**六、队列消息和非队列消息**

        从消息的发送途径来看，Windows程序中的消息可以分成2种：队列消息和非队列消息，也有叫“进队消息”和“不进队消息”。

        消息队列可以分成系统消息队列和线程消息队列。系统消息队列由Windows维护，线程消息队列则由每个GUI线程自己进行维护，为避免给non-GUI现成创建消息队列，所有线程产生时并没有消息队列，仅当线程第一次调用GDI函数时系统才给线程创建一个消息队列。（本段内容貌似应该放在消息队列时讲，但个人觉得放在这里很方便理解下面的内容）

（1）、队列消息送到系统消息队列，然后到线程消息队列；

        对于队列消息，最常见的是鼠标和键盘触发的消息，例如WM\_MOUSERMOVE,WM\_CHAR等消息，还有一些其它的消息，例如：WM\_PAINT、 WM\_TIMER和WM\_QUIT。当鼠标、键盘事件被触发后，相应的鼠标或键盘驱动程序就会把这些事件转换成相应的消息，然后输送到系统消息队列，由 Windows系统去进行处理。Windows系统则在适当的时机，从系统消息队列中取出一个消息，根据前面我们所说的MSG消息结构确定消息是要被送往那个窗口，然后把取出的消息送往创建窗口的线程的相应队列，下面的事情就该由线程消息队列操心了，Windows开始忙自己的事情去了。线程看到自己的消息队列中有消息，就从队列中取出来，通过操作系统发送到合适的窗口过程去处理。  
     一般来讲，系统总是将消息Post在消息队列的末尾。这样保证窗口以先进先出的顺序接受消息。然而,WM\_PAINT是一个例外，同一个窗口的多个 WM\_PAINT被合并成一个 WM\_PAINT 消息, 合并所有的无效区域到一个无效区域。合并WM\_PAIN的目的是为了减少刷新窗口的次数。

（2）、非队列消息直接送给目的窗口过程。  
        非队列消息将会绕过系统队列和消息队列，直接将消息发送到窗口过程，。系统发送非队列消息通知窗口，系统发送消息通知窗口。例如,当用户激活一个窗口系统发送WM\_ACTIVATE,WM\_SETFOCUS, and WM\_SETCURSOR。这些消息通知窗口它被激活了。非队列消息也可以由当应用程序调用系统函数产生。例如,当程序调用SetWindowPos系统发送WM\_WINDOWPOSCHANGED消息。一些函数也发送非队列消息，例如下面我们要谈到的函数。

**七、一个简单的Win32程序**

        现在让我们通过这个程序来更进一步的理解Windows消息！！！

1. /\*\*
2. 一个简单的Win32应用程序，通过这个简单的实例讲解Windows消息是如何传递的
3. \*/
4. #include <windows.h>
6. //声明窗口过程函数
7. LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
9. //定义一个全局变量，作为窗口类名
10. TCHAR szClassName[] = TEXT("SimpleWin32");
12. //应用程序主函数
13. int WINAPI WinMain (HINSTANCE hInstance,
14. HINSTANCE hPrevInstance,
15. LPSTR szCmdLine,
16. int iCmdShow){
17. /\*\*\*\*\*\*\*\*\*\*\*注意以下几步是windows窗口创建的流程\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
18. //1.设计一个窗口类
19. //（说明：在这里需要自己查一下 \_WNDCLASS结构体，不过里边的成员就是以下被初始化的那些变量）
20. WNDCLASS wndclass;
21. wndclass.style = CS\_HREDRAW|CS\_VREDRAW; //当窗口水平方向的宽度和垂直方向的高度变化时重绘整个窗口
22. wndclass.lpfnWndProc = WndProc;//关联窗口过程函数
23. wndclass.cbClsExtra = 0;
24. wndclass.cbWndExtra = 0;
25. wndclass.hInstance = hInstance;//实例句柄
26. wndclass.hIcon = LoadIcon(NULL,IDI\_APPLICATION);//图标
27. wndclass.hCursor = LoadCursor(NULL,IDC\_ARROW);//光标
28. wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);//画刷
29. wndclass.lpszMenuName = NULL;//菜单
30. wndclass.lpszClassName = szClassName;//类名称
31. //};
33. //2.注册窗口类
34. if(!RegisterClass (&wndclass)){
35. MessageBox (NULL, TEXT ("RegisterClass Fail!"), szClassName, MB\_ICONERROR);
36. return 0;
37. }
39. //3.创建一个窗口
40. HWND hwnd;
41. hwnd = CreateWindow(szClassName,//窗口类名称
42. TEXT ("The Simple Win32 Application"),//窗口标题
43. WS\_OVERLAPPEDWINDOW,//窗口风格,即通常我们使用的windows窗口样式
44. CW\_USEDEFAULT,//指定窗口的初始水平位置,即屏幕坐标系的窗口的左上角的X坐标
45. CW\_USEDEFAULT,//指定窗口的初始垂直位置,即屏幕坐标系的窗口的左上角的Y坐标
46. CW\_USEDEFAULT,//窗口的宽度
47. CW\_USEDEFAULT,//窗口的高度
48. NULL,//父窗口句柄
49. NULL,//窗口菜单句柄
50. hInstance,//实例句柄
51. NULL);
53. //4.显示窗口
54. ShowWindow(hwnd,iCmdShow);
56. //5.更新窗口
57. UpdateWindow(hwnd);

60. //消息循环
61. MSG msg;
62. while(GetMessage(&msg,NULL,0,0))//从消息队列中取消息
63. {
64. TranslateMessage (&msg); //转换消息
65. DispatchMessage (&msg); //派发消息
66. }
67. return msg.wParam;
68. }
70. /\*\*
71. \*消息处理函数
72. \*参数:窗口句柄，消息，消息参数，消息参数
73. \*/
74. LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
75. {
76. //处理感兴趣的消息
77. switch (message)
78. {
79. case WM\_DESTROY:
80. //当用户关闭窗口，窗口销毁，程序需结束，发退出消息，以退出消息循环
81. PostQuitMessage(0);
82. return 0;
83. }
84. //其他消息交给由系统提供的缺省处理函数
85. return ::DefWindowProc (hwnd, message, wParam, lParam);
86. }

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**八、程序说明**

        上面是一个非常简单的Win32小程序，编译运行会显示一个窗口，关闭窗口程序会结束运行。这段代码涉及GetMessage，TranslateMessage，DispatchMessage这三个函数，相关函数还有PeekMessage,WaitMessage等。在此，我们先对这些与消息相关的函数进行简单讲解。

**（一）消息发送关键函数说明**

        把一个消息发送到窗口有3种方式：发送、寄送和广播。

* *发送消息的函数：*
  + *SendMessage*
  + *SendMessageCallback*
  + *SendNotifyMessage*
  + *SendMessageTimeout*
* *寄送消息的函数：*
  + *PostMessage*
  + *PostThreadMessage*
  + *PostQuitMessage*
* *广播消息的函数：*
  + *BroadcastSystemMessage*
  + *BroadcastSystemMessageEx*

**1、SendMessage**

LRESULT SendMessage(HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam)，

        这个函数主要是向一个或多个窗口发送一条消息，一直等到消息被处理之后才会返回。不过需要注意的是，如果接收消息的窗口是同一个应用程序的一部分，那么这个窗口的窗口函数就被作为一个子程序马上被调用；如果接收消息的窗口是被另外的线程所创建的，那么窗口系统就切换到相应的线程并且调用相应的窗口函数，这条消息不会被放进目标应用程序队列中。函数的返回值是由接收消息的窗口的窗口函数返回，返回的值取决于被发送的消息。  
**2、PostMessage**

BOOL PostMessage(HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam)，

        该函数把一条消息放置到创建hWnd窗口的线程的消息队列中，该函数不等消息被处理就马上将控制返回。需要注意的是，如果hWnd参数为 HWND\_BROADCAST，那么，消息将被寄送给系统中的所有的重叠窗口和弹出窗口，但是子窗口不会收到该消息；如果hWnd参数为NULL，则该函数类似于将dwThreadID参数设置成当前线程的标志来调用PostThreadMEssage函数。

        从上面的这２个具有代表性的函数，我们可以看出消息的发送方式和寄送方式的区别所在：被发送的消息会被立即处理，处理完毕后函数才会返回；被寄送的消息不会被立即处理，他被放到一个先进先出的队列中，一直等到应用程序空线的时候才会被处理，不过函数放置消息后立即返回。  
　　实际上，发送消息到一个窗口处理过程和直接调用窗口处理过程之间并没有太大的区别，他们直接的唯一区别就在于你可以要求操作系统截获所有被发送的消息，但是不能够截获对窗口处理过程的直接调用。  
　　以寄送方式发送的消息通常是与用户输入事件相对应的，因为这些事件不是十分紧迫，可以进行缓慢的缓冲处理，例如鼠标、键盘消息会被寄送，而按钮等消息则会被发送。

**3、BroadcastSystemMessage**

long BroadcastSystemMessage(DWORDdwFlags,LPDWORD lpdwRecipients,UINT uiMessage,WPARAM wParam,LPARAM lParam);

        该函数可以向指定的接收者发送一条消息，这些接收者可以是应用程序、可安装的驱动程序、网络驱动程序、系统级别的设备驱动消息和他们的任意组合。需要注意的是，如果dwFlags参数是BSF\_QUERY并且至少一个接收者返回了BROADCAST\_QUERY\_DENY，则返回值为０，如果没有指定BSF\_QUERY，则函数将消息发送给所有接收者，并且忽略其返回值。

**（二）消息的接收关键函数说明**

*消息的接收主要有３个函数：*

*GetMessage*

*PeekMessage*

*WaitMessage。*

**1、GetMessage**

BOOL GetMessage(LPMSG lpMsg,HWND hWnd,UINTwMsgFilterMin,UINT wMsgFilterMax);

        该函数用来获取与hWnd参数所指定的窗口相关的且wMsgFilterMin和wMsgFilterMax参数所给出的消息值范围内的消息。需要注意的是，如果hWnd为NULL，则GetMessage获取属于调用该函数应用程序的任一窗口的消息，如果 wMsgFilterMin和wMsgFilterMax都是０，则GetMessage就返回所有可得到的消息。函数获取之后将删除消息队列中的除 WM\_PAINT消息之外的其他消息，至于WM\_PAINT则只有在其处理之后才被删除。

**2、PeekMessage**

BOOL PeekMessage(LPMSG lpMsg,HWNDhWnd,UINT wMsgFilterMin,UINT wMsgFilterMax,UINT wRemoveMsg）；

        该函数用于查看应用程序的消息队列，如果其中有消息就将其放入lpMsg所指的结构中，不过，与GetMessage不同的是，PeekMessage函数不会等到有消息放入队列时才返回。同样，如果hWnd为NULL，则PeekMessage获取属于调用该函数应用程序的任一窗口的消息，如果hWnd=-1，那么函数只返回把hWnd参数为NULL的PostAppMessage函数送去的消息。如果 wMsgFilterMin和wMsgFilterMax都是０，则PeekMessage就返回所有可得到的消息。函数获取之后将视最后一个参数来决定是否删除消息队列中的除 WM\_PAINT消息之外的其他消息，至于WM\_PAINT则只有在其处理之后才被删除。

**3、WaitMessage**

BOOL WaitMessage();

        当一个应用程序无事可做时，该函数就将控制权交给另外的应用程序，同时将该应用程序挂起，直到一个新的消息被放入应用程序的队列之中才返回。

**（三）消息循环关键代码说明**

while(GetMessage(&msg, NULL, 0, 0))

{

       if(!TranslateAccelerator(msg.hWnd, hAccelTable, &msg))

      {

            TranslateMessage(&msg);

            DispatchMessage(&msg);

       }

}

        首先，GetMessage从进程的主线程的消息队列中获取一个消息并将它复制到MSG结构，如果队列中没有消息，则GetMessage函数将等待一个消息的到来以后才返回。如果你将一个窗口句柄作为第二个参数传入GetMessage，那么只有指定窗口的的消息可以从队列中获得。

        GetMessage也可以从消息队列中过滤消息只接受消息队列中落在范围内的消息。这时候就要利用GetMessage／PeekMessage指定一个消息过滤器。这个过滤器是一个消息标识符的范围或者是一个窗体句柄，或者两者同时指定。当应用程序要查找一个后入消息队列的消息是很有用。WM\_KEYFIRST 和 WM\_KEYLAST 常量用于接受所有的键盘消息。 WM\_MOUSEFIRST 和 WM\_MOUSELAST 常量用于接受所有的鼠标消息。

        然后TranslateAccelerator判断该消息是不是一个按键消息并且是一个加速键消息，如果是，则该函数将把几个按键消息转换成一个加速键消息传递给窗口的回调函数。处理了加速键之后，函数TranslateMessage将把两个按键消息WM\_KEYDOWN和WM\_KEYUP转换成一个 WM\_CHAR，不过需要注意的是，消息WM\_KEYDOWN,WM\_KEYUP仍然将传递给窗口的回调函数。

        处理完之后，DispatchMessage函数将把此消息发送给该消息指定的窗口中已设定的回调函数。如果消息是WM\_QUIT，则 GetMessage返回0，从而退出循环体。应用程序可以使用PostQuitMessage来结束自己的消息循环。通常在主窗口的 WM\_DESTROY消息中调用。

下面我们举一个常见的小例子来说明这个消息泵的运用：

if (::PeekMessage(&msg, m\_hWnd, WM\_KEYFIRST,WM\_KEYLAST, PM\_REMOVE))  
{  
          if (msg.message == WM\_KEYDOWN && msg.wParam == VK\_ESCAPE)．．．  
}

  这里我们接受所有的键盘消息，所以就用WM\_KEYFIRST 和 WM\_KEYLAST作为参数。最后一个参数可以是PM\_NOREMOVE 或者 PM\_REMOVE，表示消息信息是否应该从消息队列中删除。                   
   所以这段小代码就是判断是否按下了Esc键，如果是就进行处理。

**（四）消息处理关键函数说明**

        （也就是文章开头提到的窗口过程）  
        窗口过程是一个用于处理所有发送到这个窗口的消息的函数。任何一个窗口类都有一个窗口过程。同一个类的窗口使用同样的窗口过程来响应消息。系统发送消息给窗口过程将消息数据作为参数传递给他，消息到来之后，按照消息类型排序进行处理，其中的参数则用来区分不同的消息，窗口过程使用参数产生合适行为。

        一个窗口过程不经常忽略消息，如果他不处理，它会将消息传回到执行默认的处理。窗口过程通过调用DefWindowProc来做这个处理。窗口过程必须 return一个值作为它的消息处理结果。大多数窗口只处理小部分消息和将其他的通过DefWindowProc传递给系统做默认的处理。窗口过程被所有属于同一个类的窗口共享，能为不同的窗口处理消息。下面我们来看一下具体的实例：

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

 int wmId, wmEvent;

 PAINTSTRUCT ps;

 HDC hdc;

 TCHAR szHello[MAX\_LOADSTRING];

 LoadString(hInst, IDS\_HELLO, szHello, MAX\_LOADSTRING);

 switch (message)

 {

  case WM\_COMMAND:

         wmId    = LOWORD(wParam);

         wmEvent = HIWORD(wParam);

         // Parse the menu selections:

         switch (wmId)

         {

          case IDM\_ABOUT:

             DialogBox(hInst, (LPCTSTR)IDD\_ABOUTBOX, hWnd, (DLGPROC)About);

             break;

          case IDM\_EXIT:

             DestroyWindow(hWnd);

             break;

          default:

             return DefWindowProc(hWnd, message, wParam, lParam);

         }

   break;

  case WM\_PAINT:

         hdc = BeginPaint(hWnd, &ps);

         // TODO: Add any drawing code here

         RECT rt;

         GetClientRect(hWnd, &rt);

         DrawText(hdc, szHello, strlen(szHello), &rt, DT\_CENTER);

         EndPaint(hWnd, &ps);

         break;

  case WM\_DESTROY:

         PostQuitMessage(0);

         break;

  default:

         return DefWindowProc(hWnd, message, wParam, lParam);

  }

  return 0;

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**（五）消息分流器**

　　通常的窗口过程是通过一个switch语句来实现的，这个事情很烦，有没有更简便的方法呢？有，那就是消息分流器，利用消息分流器，我们可以把switch语句分成更小的函数，每一个消息都对应一个小函数，这样做的好处就是对消息更容易管理。  
　　之所以被称为消息分流器，就是因为它可以对任何消息进行分流。下面我们做一个函数就很清楚了：

void MsgCracker(HWND hWnd,int id,HWND hWndCtl,UINT codeNotify){

    switch(id){

case ID\_A:

            if(codeNotify==EN\_CHANGE)

                break;

      case ID\_B:

            if(codeNotify==BN\_CLICKED)

break;

             .

       }

}

然后我们修改一下窗口过程：

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

       switch(message)

      {

             HANDLE\_MSG(hWnd,WM\_COMMAND,MsgCracker);

             HANDLE\_MSG(hWnd,WM\_DESTROY,MsgCracker);

           default:

                    return DefWindowProc(hWnd, message, wParam, lParam);

  　}

  return 0;

}

上面的 HANDLE\_MSG 是一个定义在WindowsX.h中的宏：

1. #define HANDLE\_MSG(hwnd,msg,fn) \
2. switch(msg): return HANDLE\_##msg((hwnd),(wParam),(lParam),(fn));

*实际上，HANDLE\_WM\_XXXX都是宏，例如：*

1. *HANDLE\_MSG(hWnd,WM\_COMMAND,MsgCracker);将被转换成如下定义：*
3. *#define HANDLE\_WM\_COMMAND(hwnd,wParam,lParam,fn)\*
4. *((fn)((hwnd),(int)(LOWORD(wParam)),(HWND)(lParam),(UINT)HIWORD(wParam)),0L);*

        好了，事情到了这一步，应该一切都明朗了。  
        不过，我们发现在windowsx.h里面还有一个宏：FORWARD\_WM\_XXXX，我们还是以WM\_COMMAND为例，进行分析：

1. #define FORWARD\_WM\_COMMAND(hwnd, id, hwndCtl, codeNotify, fn) \
2. (void)(fn)((hwnd), WM\_COMMAND, MAKEWPARAM((UINT)(id),(UINT)(codeNotify)), (LPARAM)(HWND)(hwndCtl))

        所以实际上，FORWARD\_WM\_XXXX将消息参数进行了重新构造，生成了wParam &&lParam，然后调用了我们定义的函数。