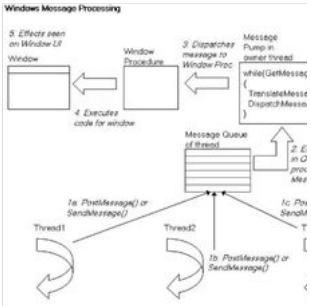
消息机制

编辑

在Windows中发生的一切都可以用消息来表示，消息用于告诉操作系统发生了什么，所有的Windows应用程序都是消息驱动的。 一个消息是由消息的名称(UINT)和两个参数（WPARAM, LPARAM）组成。消息的参数中包含有重要的信息。例如对鼠标消息而言，LPARAM中一般包含鼠标的位置信息，而WPARAM参数中包含了发生该消息时，SHIFT、CTRL等键的状态信息，对于不同的消息类型来说，两个参数也都相应地具有明确意义。

中文名	消息机制	源　于	Windows应用程序
组　成	名称和两个参数WPARAM, LPARAM	概　述	消息用于告诉操作系统发生了什么



消息机制图册

消息机制

目录

程序核心

编辑

在Windows中，不同的消息由应用程序的不同部分进行处理。MFC库将很多底层的消息都屏蔽了，使用户更加方便、简易地处理消息。例如，用户接收到诸如移动鼠标键（WM_MOUSEMOVE）消息或单击鼠标键（WM_LRBUTTONDOWN）消息时不必处理窗口和鼠标的重画工作，MFC及应用程序框架会替用户做这些工作。在使用MFC进行编程时，用户只需处理一些高层的消息，例如，“用户在单击窗口中的OK按扭”，“用户现在选中了下拉列表框中的第五项”等等，这样就大大减轻了程序员的负担。

输入焦点

编辑

Windows是一个以消息为导向的系统，应用程序只能被动地等待用户按键的消息，不能主动地去读键盘的状态，也就是说，每当键盘上有个键被按下，系统就会发出一个按键消息给窗口，告诉它某个键被按下去了，只要鼠标移动一下，系统也会发出相应的消息，并把鼠标的坐标信息传给窗口。

Windows可以同时执行许多程序，但键盘只有一个，怎么判断由哪个窗口接收键盘及鼠标的消息呢？采用“输入焦点”（input focus）技术可以解决这个问题。只要某个窗口取得输入焦点，它不但会被提升到屏幕的最前面，颜色也会有所不同，所有的键盘消息就会导向该窗口，该窗口也成为“活动窗口”。

窗口如何取得输入焦点？通常被鼠标单击的窗口会得到输入焦点，除此之外，程序本身也可以利用SetFocus()来指定哪个窗口拥有输入焦点。

CWnd* CWnd::SetFocus();

如果调用某窗口的SetFocus()成员函数，该窗口就可以取得输入焦点，该函数返回前一个拥有输入焦点的窗口。

如果某个窗口的输入焦点被抢走，Windows系统就会发出WM_KILLFOCUS消息给这个失去输入焦点的窗口，同时还会告诉该窗口下一个取得输入焦点的窗口的指针。而获得输入焦点的窗口则会收到WM_SETFOCUS消息。

消息响应函数分别为：

afx_msg void OnKillFocus(CWnd* pNewWnd);

其中的参数为得到输入焦点的窗口的指针。

Afx_msg void OnSetFocus(CWnd* pOldWnd);

其中的参数为失去输入焦点的窗口的指针。

消息分类

编辑

相关术语

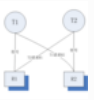

事件驱动模型

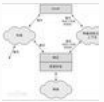

多线程

进程间


handler机制

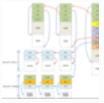
事件驱动


死锁

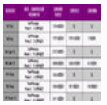

句柄

相关函数

钩子函数


回调函数


sendmes


peekmessa...


winmain


createpr

分享

词条统计

浏览次数：38362次

编辑次数：19次历史版本

最近更新：2016-04-15

创建者：ctu_85

Windows系统预定义了许多消息，每个消息都拥有一个宏定义，即用形象的字符串来标识消息，一系列#define 语句将消息与特定数值联系起来，可以在头文件WinUser.h中找到这些宏定义，例如

#define WM_PAINT 120

可以在程序中通过消息名“WM_PAINT”来访问它。其他消息如：

#define WM_MOUSEMOVE 0x0200

#defineWM_LBUTTONDOWN 0x0201

#define WM_LBUTTONUP 0x0202

#defineWM_LBUTTONDBLCLK 0x0203

#defineWM_RBUTTONDOWN 0x0204

#define WM_RBUTTONUP 0x0205

#define WM_RBUTTONDBLCLK 0x0206

#define WM_MBUTTONDOWN 0x0207

#define WM_MBUTTONUP 0x0208

系统定义的消息有不同的前缀，不同的前缀有不同的含义。

标准的消息

除了WM_COMMAND消息，所有以WM_为前缀的消息都是标准的Windows消息，如窗口、鼠标移动、窗口大小改变等，程序启动或退出甚至每一段固定的时间都会产生标准Windows消息。如

1） 键盘消息

对于窗口而言，来自用户的按键输入可分为两类，一类是系统键（system key），另一类则是非系统键。凡是ALT和其它键一同按下的组合称为“系统键”，窗口收到系统键之后，会自动地将它解释成系统事件，或者查阅键盘加速表，将系统键翻译成加速表指定的信息。如：ALT+F4的组合会迫使窗口关闭，“ALT+字母”的组合可能会拉下某个菜单。

当用户按下某个键时，Windows系统会先发出WM_KEYDOWN消息给窗口，这个消息的意思是“按键被压下去”。接着Windows系统会发出WM_CHAR给同一个窗口，这个消息代表的意义是“系统送来某个字符”，如果用户放开此键，Windows系统会发出WM_KEYUP消息，表示“按键被放开”。如果用户一直按住某个键不放，经过一段时间之后会产生“连发”的效果，造成Windows系统不停地发出WM_KEYDOWN与WM_CHAR消息。

计算机内部以ASCII码的规则来记录所有的英文字母和数字符号。不过不是键盘上每个按键都可以对应成ASCII码中的字符，如大小写键、CTRL键、F1到F12键等。

每个按键都有对应的扫描码，PC BIOS收到键盘的中断消息后，会自动将扫描码翻译成ASCII码，但有些控制键无法译成ASCII码，如Page UP、Page Down等。Windows定义了一套与硬件无关的“虚拟键码”来表示键盘上所有的按键，如A键就是VK_A、ESC键就是VK_ESC、F1键是VK_F1、ALT键是VK_MENU等。因为“虚拟键码”定义的规则与硬件无关，所以有些虚拟键在通常的键盘上根本就找不着。

#define VK_LBUTTON 0x01

#define VK_RBUTTON 0x02

#define VK_CANCEL 0x03

#define VK_MBUTTON 0x04 /* NOT contiguous with L & RBUTTON */

#define VK_BACK 0x08

#define VK_TAB 0x09

#define VK_CLEAR 0x0C

#define VK_RETURN 0x0D

#define VK_SHIFT 0x10

#define VK_CONTROL 0x11

#define VK_MENU 0x12

#define VK_PAUSE 0x13

#define VK_CAPITAL 0x14

#define VK_F1 0x70

#define VK_F2 0x71

#define VK_F3 0x72

#define VK_F4 0x73

#define VK_F5 0x74

#define VK_F6 0x75

#define VK_F7 0x76

#define VK_F8 0x77

#define VK_F9 0x78

#define VK_F10 0x79

#

#define WM_CHAR 0x0102 //字符消息

息WM_CHAR)，含有此窗口键盘上哪个键被按下了。该消息的处理函数为OnChar()。具体形式为：

afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)

各参数含义为：

nChar: 键盘所输入的ASCII码。

nRepCnt: 按键的重复次数，当用户按下某个键不放时，该参数将持续增加。

nFlag: 用于传递按键的其它一些信息，如扫描码，上一次按键状态等。具体如下：

字节 说明

0-7 键盘扫描码

8 此按键为扩充按键，如F1，F12等功能键，此字节等于1时为真

9-12 保留

13 此字节为1表示按下键的同时，ALT键也被按住了

14 前一个按键状态。此字节为1代表信息在按键被按下之前就送出来了

15 此字节为1表示这个按键已经被放开了，反之就表示还被按着

此外还有两个常用的键盘消息：WM_KEYDOWN和WM_KEYUP.

WM_KEYDOWN消息是当用户按下一个非系统键时产生的，非系统键就是不按下ALT键时的按键。

WM_KEYUP 消息是当用户释放一个非系统键时产生的。

鼠标消息

① #define WM_MOUSEMOVE 0x0200 //鼠标移动消息

当鼠标在某个窗口内移动时，Windows会不断地发出鼠标移动消息WM_MOUSEMOVE，并把鼠标的最新位置传给该窗口。如果在窗口的范围内按下鼠标左键，系统就会发出“按下左键”的WM_LBUTTONDOWN消息给该窗口，等到用户放开按键后，再发出“放开左键”的WM_LBUTTONUP消息给该窗口。

鼠标移动消息的消息响应函数为：

afx_msg voidOnMouseMove(UINT nFlags， CPoint point)

其中的参数含义如下：

UINT nFlag：此事件发生时，鼠标按键、键盘控制键的状态，可以是以下值的任意组合：

当用户按下CTRL键时，nFlags设置为MK_CONTROL。

当用户按下鼠标左键时，nFlags设置为MK_LBUTTON。

当用户按下鼠标中键时，nFlags设置为MK_MBUTTON

浅析消息机制

Windows系统是一个消息驱动的OS，什么是消息呢？我很难说得清楚，也很难下一个定义，我下面从不同的几个方面讲解一下，希望大家看了后有一点了解。

消息的组成

一个消息由一个消息名称（UINT），和两个参数（WPARAM，LPARAM）组成。当用户进行了输入或是窗口的状态发生改变时系统都会发送消息到某一个窗口。例如当菜单转中之后会有WM_COMMAND消息发送，WPARAM的高字中（HIWORD(wParam)）是命令的ID号，对菜单来讲就是菜单ID。当然用户也可以定义自己的消息名称，也可以利用自定义消息来发送通知和传送数据。

谁将收到消息

一个消息必须由一个窗口接收。在窗口的过程（WNDPROC）中可以对消息进行分析，对自己感兴趣的消息进行处理。例如你希望对菜单选择进行处理那么你可以定义对WM_COMMAND进行处理的代码，如果希望在窗口中进行图形输出就必须对WM_PAINT进行处理。

未处理的消息

M$为窗口编写了默认的窗口过程，这个窗口过程将负责处理那些你不处理消息。正因为有了这个默认窗口过程我们才可以利用Windows的窗口进行开发而不必过多关注窗口各种消息的处理。例如窗口在被拖动时会有很多消息发送，而我们都可以不予理

窗口句柄

说到消息就不能不说窗口句柄，系统通过窗口句柄来在整个系统中唯一标识一个窗口，发送一个消息时必须指定一个窗口句柄表明该消息由那个窗口接收。而每个窗口都会有自己的窗口过程，所以用户的输入就会被正确的处理。例如有两个窗口共用一个窗口过程代码，你在窗口一上按下鼠标时消息就会通过窗口一的句柄被发送到窗口一而不是窗口二。

示例

下面有一段伪代码演示如何在窗口过程中处理消息

```
LONG yourWndProc(HWND hWnd,UINT uMessageType,WPARAM wP,LPARAM)
{
switch(uMessageType)
{//使用SWITCH语句将各种消息分开
case(WM_PAINT):
doYourWindow(...);//在窗口需要重新绘制时进行输出
break;
case(WM_LBUTTONDOWN):
doYourWork(...);//在鼠标左键被按下时进行处理
break;
default:
callDefaultWndProc(...);//对于其它情况就让系统自己处理
break;
}
}
```

接下来谈谈什么是消息机制：系统将会维护一个或多个消息队列，所有产生的消息都会被放入或是插入队列中。系统会在队列中取出每一条消息，根据消息的接收句柄而将该消息发送给拥有该窗口的程序的消息循环。每一个运行的程序都有自己的消息循环，在循环中得到属于自己的消息并根据接收窗口的句柄调用相应的窗口过程。而在没有消息时消息循环就将控制权交给系统所以Windows可以同时进行多个任务。下面的伪代码演示了消息循环的用法：

```
while(1)
{
id=getMessage(...);
if(id == quit)
break;
translateMessage(...);
}
```

当该程序没有消息通知时getMessage就不会返回，也就不会占用系统的CPU时间。 图示消息投递模式

在16位的系统中系统中只有一个消息队列，所以系统必须等待当前任务处理消息后才可以发送下一消息到相应程序，如果一个程序陷如死循环或是耗时操作时系统就会得不到控制权。这种多任务系统也就称为协同式的多任务系统。Windows3.X就是这种系统。

而32位的系统中每一运行的程序都会有一个消息队列，所以系统可以在多个消息队列中转换而不必等待当前程序完成消息处

理就可以得到控制权。这种多任务系统就称为抢先式的多任务系统。Windows95/NT就是这种系统。

消息队列

编辑

Windows中有一个系统消息队列，对于每一个正在执行的Windows应用程序,系统为其建立一个“消息队列”，即应用程序队列，用来存放该程序可能创建的各种窗口的消息。应用程序中含有一段称作“消息循环”的代码，用来从消息队列中检索这些消息并把它们分发到相应的窗口函数中。

消息循环

编辑

Windows为当前执行的每个Windows程序维护一个「消息队列」。在发生输入事件之后，Windows将事件转换为一个「消息」并将消息放入程序的消息队列中。程序通过执行一块称之为「消息循环」的程序代码从消息队列中取出消息：

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


msg变量是型态为MSG的结构，型态MSG在WINUSER.H中定义如下：

```
typedef struct tagMSG
{
HWND hwnd ;
UINT message ;
WPARAM wParam ;
LPARAM lParam ;
DWORD time ;
POINT pt ;
}
MSG, * PMSG ;
```

POINT数据型态也是一个结构，它在WINDEF.H中定义如下：

```
typedef struct tagPOINT
{
LONG x ;
LONG y ;
}
POINT, * PPOINT;
```

TranslateMessage(&msg); 将msg结构传给Windows，进行一些键盘转换。（关于这一点，我们将在第六章中深入讨论。）

DispatchMessage(&msg);又将msg结构回传给Windows。然后，Windows将该消息发送给适当的窗口消息处理程序，让它进行处理。这也就是说，Windows将呼叫窗口消息处理程序。在HELLOWIN中，这个窗口消息处理程序就是WndProc函数。处理完消息之后，WndProc传回到Windows。此时，Windows还停留在DispatchMessage呼叫中。在结束DispatchMessage呼叫的处理之后，Windows回到HELLOWIN程序中，并且接着从下一个GetMessage呼叫开始消息循环。

消息分类

编辑

消息能够被分为「队列化的」和「非队列化的」。

队列化消息

队列化的消息是由Windows放入程序消息队列中的。在程序的消息循环中，重新传回并分配给窗口消息处理程序。非队列化的消息在Windows呼叫窗口时直接送给窗口消息处理程序。也就是说，队列化的消息被「发送」给消息队列，而非队列化的消息则「发送」给窗口消息处理程序。任何情况下，窗口消息处理程序都将获得窗口所有的消息--包括队列化的和非队列化的。窗口消息处理程序是窗口的「消息中心」。队列化消息基本上是使用者输入的结果，以击键（如WM_KEYDOWN和WM_KEYUP消息）、击键产生的字符（WM_CHAR）、鼠标移动（WM_MOUSEMOVE）和鼠标按钮（WM_LBUTTONDOWN）的形式给出。队列化消息还包含时钟消息（WM_TIMER）、更新消息（WM_PAINT）和退出消息（WM_QUIT）。

非队列化消息

非队列化消息则是其它消息。在许多情况下，非队列化消息来自呼叫特定的Windows函数。例如，当WinMain呼叫CreateWindow时，Windows将建立窗口并在处理中给窗口消息处理程序发送一个WM_CREATE消息。当WinMain呼叫ShowWindow时，Windows将给窗口消息处理程序发送WM_SIZE和WM_SHOWWINDOW消息。当WinMain呼叫UpdateWindow时，Windows将给窗口消息处理程序发送WM_PAINT消息。键盘或鼠标输入时发出的队列化消息信号，也能在非队列化消息中出现。例如，用键盘或鼠标选择了一个菜单项时，键盘或鼠标消息就是队列化的，而说明菜单项已选中的WM_COMMAND消息则可能就是非队列化的。

函数区别

SendMessage()与PostMessage()的区别

它们两者是用于向应用程序发送消息的。PostMessagex()将消息直接加入到应用程序的消息队列中，不等程序返回就退出。

函数peekmessage和getmessage的区别

两个函数主要有以下两个区别:

1.GetMessage将等到有合适的消息时才返回,而PeekMessage只是撇一下消息队列。

2.GetMessage会将消息从队列中删除,而PeekMessage可以设置最后一个参数wRemoveMsg来决定是否将消息保留在队列中。

词条标签： 计算机学