# **[Hook钩子C#实例](http://www.cnblogs.com/ceoliujia/archive/2010/05/20/1740217.html)**

**转过来的文章，出处已经不知道了，但只这篇步骤比较清晰，就贴出来了。**

**一。写在最前**

本文的内容只想以最通俗的语言说明钩子的使用方法，具体到钩子的详细介绍可以参照下面的网址：

[http://www.microsoft.com/china/community/program/originalarticles/techdoc/hook.mspx](http://www.microsoft.com/china/community/program/originalarticles/techdoc/hook.mspx" \t "http://www.cnblogs.com/ceoliujia/archive/2010/05/20/_blank)

**二。了解一下钩子**

从字面上理解，钩子就是想钩住些东西，在程序里可以利用钩子提前处理些Windows消息。

**例子：**有一个Form，Form里有个TextBox，我们想让用户在TextBox里输入的时候，不管敲键盘的哪个键，TextBox里显示的始终为“A”，这时我们就可以利用钩子监听键盘消息，先往Windows的钩子链表中加入一个自己写的钩子监听键盘消息，只要一按下键盘就会产生一个键盘消息，我们的钩子在这个消息传到TextBox之前先截获它，让TextBox显示一个“A”，之后结束这个消息，这样TextBox得到的总是“A”。

**消息截获顺序：**既然是截获消息，总要有先有后，钩子是按加入到钩子链表的顺序决定消息截获顺序。就是说最后加入到链表的钩子最先得到消息。

**截获范围：**钩子分为线程钩子和全局钩子，线程钩子只能截获本线程的消息，全局钩子可以截获整个系统消息。我认为应该尽量使用线程钩子，全局钩子如果使用不当可能会影响到其他程序。

**三。开始通俗**

这里就以上文提到的简单例子做个线程钩子。

**第一步：**声明API函数

使用钩子，需要使用WindowsAPI函数，所以要先声明这些API函数。

// 安装钩子

[DllImport("user32.dll",CharSet=CharSet.Auto, CallingConvention=CallingConvention.StdCall)]

public static extern int SetWindowsHookEx(int idHook, HookProc lpfn, IntPtr hInstance, int threadId);

// 卸载钩子

[DllImport("user32.dll",CharSet=CharSet.Auto, CallingConvention=CallingConvention.StdCall)]

public static extern bool UnhookWindowsHookEx(int idHook);

// 继续下一个钩子

[DllImport("user32.dll",CharSet=CharSet.Auto, CallingConvention=CallingConvention.StdCall)]

public static extern int CallNextHookEx(int idHook, int nCode, Int32 wParam, IntPtr lParam);

// 取得当前线程编号

[DllImport("kernel32.dll")]

static extern int GetCurrentThreadId();

声明一下API函数，以后就可以直接调用了。

**第二步：**声明、定义。

public delegate int HookProc(int nCode, Int32 wParam, IntPtr lParam);

static int hKeyboardHook = 0;

HookProc KeyboardHookProcedure;

先解释一下委托，钩子必须使用标准的钩子子程，钩子子程就是一段方法，就是处理上面例子中提到的让TextBox显示“A”的操作。

钩子子程必须按照HookProc(int nCode, Int32 wParam, IntPtr lParam)这种结构定义，三个参数会得到关于消息的数据。

当使用SetWindowsHookEx函数安装钩子成功后会返回钩子子程的句柄，hKeyboardHook变量记录返回的句柄，如果hKeyboardHook不为0则说明钩子安装成功。

**第三步：**写钩子子程

钩子子程就是钩子所要做的事情。

private int KeyboardHookProc(int nCode, Int32 wParam, IntPtr lParam)

{if (nCode >= 0)

{

textbox1.Text = “A”;

return 1;

}

return CallNextHookEx(hKeyboardHook, nCode, wParam, lParam);

}

我们写一个方法，返回一个int值，包括三个参数。如上面给出的代码，符合钩子子程的标准。

nCode参数是钩子代码，钩子子程使用这个参数来确定任务，这个参数的值依赖于Hook类型。

wParam和lParam参数包含了消息信息，我们可以从中提取需要的信息。

方法的内容可以根据需要编写，我们需要TextBox显示“A”，那我们就写在这里。当钩子截获到消息后就会调用钩子子程，这段程序结束后才往下进行。截获的消息怎么处理就要看子程的返回值了，如果返回1，则结束消息，这个消息到此为止，不再传递。如果返回0或调用CallNextHookEx函数则消息出了这个钩子继续往下传递，也就是传给消息真正的接受者。

**第四步：**安装钩子、卸载钩子

准备工作都完成了，剩下的就是把钩子装入钩子链表。

我们可以写两个方法在程序中合适位置调用。代码如下：

// 安装钩子

public void HookStart()

{

if(hMouseHook == 0)

{

// 创建HookProc实例

MouseHookProcedure = new HookProc(MouseHookProc);

// 设置线程钩子

hMouseHook = SetWindowsHookEx( 2, KeyboardHookProcedure, IntPtr.Zero,

GetCurrentThreadId());

// 如果设置钩子失败

if(hMouseHook == 0 )

{

HookStop();

throw new Exception("SetWindowsHookEx failed.");

}

}

}

// 卸载钩子

public void HookStop()

{

bool retKeyboard = true;

if(hKeyboardHook != 0)

{

retKeyboard = UnhookWindowsHookEx(hKeyboardHook);

hKeyboardHook = 0;

}

if (!(retMouse && retKeyboard)) throw new Exception("UnhookWindowsHookEx

failed.");

}

安装钩子和卸载钩子关键就是SetWindowsHookEx和UnhookWindowsHookEx方法。

SetWindowsHookEx (int idHook, HookProc lpfn, IntPtr hInstance, int threadId) 函数将钩子加入到钩子链表中，说明一下四个参数：

**idHook** 钩子类型，即确定钩子监听何种消息，上面的代码中设为2，即监听键盘消息并且是线程钩子，如果是全局钩子监听键盘消息应设为13，线程钩子监听鼠标消息设为7，全局钩子监听鼠标消息设为14。

**lpfn** 钩子子程的地址指针。如果dwThreadId参数为0 或是一个由别的进程创建的线程的标识，lpfn必须指向DLL中的钩子子程。 除此以外，lpfn可以指向当前进程的一段钩子子程代码。钩子函数的入口地址，当钩子钩到任何消息后便调用这个函数。

**hInstance**应用程序实例的句柄。标识包含lpfn所指的子程的DLL。如果threadId 标识当前进程创建的一个线程，而且子程代码位于当前进程，hInstance必须为NULL。可以很简单的设定其为本应用程序的实例句柄。

**threaded** 与安装的钩子子程相关联的线程的标识符。如果为0，钩子子程与所有的线程关联，即为全局钩子。

上面代码中的SetWindowsHookEx方法安装的是线程钩子，用GetCurrentThreadId()函数得到当前的线程ID，钩子就只监听当前线程的键盘消息。

UnhookWindowsHookEx (int idHook) 函数用来卸载钩子，卸载钩子与加入钩子链表的顺序无关，并非后进先出。

**四。节外生枝**

**安装全局钩子**

上文使用的是线程钩子，如果要使用全局钩子在钩子的安装上略有不同。如下：

SetWindowsHookEx( 13,KeyboardHookProcedure,

Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly().GetModules()[0]),0)

这条语句即定义全局钩子。

**子程消息处理**

钩子子程可以得到两个关于消息信息的参数wPrama、lParam。怎么将这两个参数转成我们更容易理解的消息呢。

对于鼠标消息，我们可以定义下面这个结构：

public struct MSG

{

public Point p;

public IntPtr HWnd;

public uint wHitTestCode;

public int dwExtraInfo;

}

对于键盘消息，我们可以定义下面这个结构：

public struct KeyMSG

{

public int vkCode;

public int scanCode;

public int flags;

public int time;

public int dwExtraInfo;

}

然后我们可以在子程里用下面语句将lParam数据转换成MSG或KeyMSG结构数据

MSG m = (MSG) Marshal.PtrToStructure(lParam, typeof(MSG));

KeyMSG m = (KeyMSG) Marshal.PtrToStructure(lParam, typeof(KeyMSG));

这样可以更方便的得到鼠标消息或键盘消息的相关信息，例如p即为鼠标坐标，HWnd即为鼠标点击的控件的句柄，vkCode即为按键代码。

**注：**这条语句对于监听鼠标消息的线程钩子和全局钩子都可以使用，但对监听键盘消息的线程钩子使用会出错，目前在找原因。

如果是监听键盘消息的线程钩子，我们可以根据lParam值的正负确定按键是按下还是抬起，根据wParam值确定是按下哪个键。

// 按下的键

Keys keyData = (Keys)wParam;

if(lParam.ToInt32() > 0)

{

// 键盘按下

}

if(lParam.ToInt32() < 0)

{

// 键盘抬起

}

如果是监听键盘消息的全局钩子，按键是按下还是抬起要根据wParam值确定。

wParam = = 0x100 // 键盘按下

wParam = = 0x101 // 键盘抬起