要实现一个屏幕键盘，需要监听所有键盘事件，无论窗体是否被激活。因此需要一个全局的钩子，也就是系统范围的钩子。

什么是钩子（Hook） 钩子（Hook）是Windows提供的一种消息处理机制平台，是指在程序正常运行中接受信息之前预先启动的函数，用来检查和修改传给该程序的信息，（钩子）实际上是一个处理消息的程序段，通过系统调用，把它挂入系统。每当特定的消息发出，在没有到达目的窗口前，钩子程序就先捕获该消息，亦即钩子函数先得到控制权。这时钩子函数即可以加工处理（改变）该消息，也可以不作处理而继续传递该消息，还可以强制结束消息的传递。注意：安装钩子函数将会影响系统的性能。监测“系统范围事件”的系统钩子特别明显。因为系统在处理所有的相关事件时都将调用您的钩子函数，这样您的系统将会明显的减慢。所以应谨慎使用，用完后立即卸载。还有，由于您可以预先截获其它进程的消息，所以一旦您的钩子函数出了问题的话必将影响其它的进程。

钩子的作用范围 一共有两种范围（类型）的钩子，局部的和远程的。局部钩子仅钩挂自己进程的事件。远程的钩子还可以将钩挂其它进程发生的事件。远程的钩子又有两种： 基于线程的钩子将捕获其它进程中某一特定线程的事件。简言之，就是可以用来观察其它进程中的某一特定线程将发生的事件。系统范围的钩子将捕捉系统中所有进程将发生的事件消息。 这里列举两个要用到的钩子。

1. WH\_KEYBOARD\_LL Hook WH\_KEYBOARD\_LL Hook监视输入到线程消息队列中的键盘消息。

（2）WH\_MOUSE\_LL Hook WH\_MOUSE\_LL Hook监视输入到线程消息队列中的鼠标消息

class 把 API 调用封装起来以便调用。

安装钩子 使用SetWindowsHookEx函数（API函数），指定一个Hook类型、自己的Hook过程是全局还是局部Hook，同时给出Hook过程的进入点，就可以轻松的安装自己的Hook过程。SetWindowsHookEx总是将你的Hook函数放置在Hook链的顶端。你可以使用CallNextHookEx函数将系统消息传递给Hook链中的下一个函数。 对于某些类型的Hook，系统将向该类的所有Hook函数发送消息，这时，Hook函数中的CallNextHookEx语句将被忽略。全局（远程钩子）Hook函数可以拦截系统中所有线程的某个特定的消息，为了安装一个全局Hook过程，必须在应用程序外建立一个DLL并将该Hook函数封装到其中，应用程序在安装全局Hook过程时必须先得到该DLL模块的句柄。将Dll名传递给LoadLibrary 函数，就会得到该DLL模块的句柄；得到该句柄 后，使用GetProcAddress函数可以得到Hook过程的地址。最后，使用SetWindowsHookEx将 Hook过程的首址嵌入相应的Hook链中，SetWindowsHookEx传递一个模块句柄，它为 Hook过程的进入点，线程标识符置为0，该Hook过程同系统中的所有线程关联。如果是安装局部Hook此时该Hook函数可以放置在DLL中，也可以放置在应用程序的模块段。在C#中通过平台调用来调用API函数。

public void Start(bool installMouseHook, bool installKeyboardHook) {

2 if (hMouseHook == IntPtr.Zero && installMouseHook) {

3 MouseHookProcedure = new HookProc(MouseHookProc);

4 hMouseHook = SetWindowsHookEx(

5 WH\_MOUSE\_LL,

6 MouseHookProcedure,

7 Marshal.GetHINSTANCE(

8 Assembly.GetExecutingAssembly().GetModules()[0]),

9 0

10 );

11

12 if (hMouseHook == IntPtr.Zero) {

13 int errorCode = Marshal.GetLastWin32Error();

14 Stop(true, false, false);

15

16 throw new Win32Exception(errorCode);

17 }

18 }

19

20 if (hKeyboardHook == IntPtr.Zero && installKeyboardHook) {

21 KeyboardHookProcedure = new HookProc(KeyboardHookProc);

22 //install hook

23 hKeyboardHook = SetWindowsHookEx(

24 WH\_KEYBOARD\_LL,

25 KeyboardHookProcedure,

26 Marshal.GetHINSTANCE(

27 Assembly.GetExecutingAssembly().GetModules()[0]),

28 0);

29 // If SetWindowsHookEx fails.

30 if (hKeyboardHook == IntPtr.Zero) {

31 // Returns the error code returned by the last

32 // unmanaged function called using platform invoke

33 // that has the DllImportAttribute.SetLastError flag set.

34 int errorCode = Marshal.GetLastWin32Error();

35 //do cleanup

36 Stop(false, true, false);

37 //Initializes and throws a new instance of the

38 // Win32Exception class with the specified error.

39 throw new Win32Exception(errorCode);

40 }

41 }

42 }

使用完钩子后，要进行卸载，这个可以写在析构函数中。将这个文件编译成一个dll，即可在应用程序中调用。

1

2 public void Stop() {

3 this.Stop(true, true, true);

4 }

5

6 public void Stop(bool uninstallMouseHook, bool uninstallKeyboardHook,

7 bool throwExceptions) {

8 // if mouse hook set and must be uninstalled

9 if (hMouseHook != IntPtr.Zero && uninstallMouseHook) {

10 // uninstall hook

11 bool retMouse = UnhookWindowsHookEx(hMouseHook);

12 // reset invalid handle

13 hMouseHook = IntPtr.Zero;

14 // if failed and exception must be thrown

15 if (retMouse == false && throwExceptions) {

16 // Returns the error code returned by the last unmanaged function

17 // called using platform invoke that has the DllImportAttribute.

18 // SetLastError flag set.

19 int errorCode = Marshal.GetLastWin32Error();

20 // Initializes and throws a new instance of the Win32Exception class

21 // with the specified error.

22 throw new Win32Exception(errorCode);

23 }

24 }

25

26 // if keyboard hook set and must be uninstalled

27 if (hKeyboardHook != IntPtr.Zero && uninstallKeyboardHook) {

28 // uninstall hook

29 bool retKeyboard = UnhookWindowsHookEx(hKeyboardHook);

30 // reset invalid handle

31 hKeyboardHook = IntPtr.Zero;

32 // if failed and exception must be thrown

33 if (retKeyboard == false && throwExceptions) {

34 // Returns the error code returned by the last unmanaged function

35 // called using platform invoke that has the DllImportAttribute.

36 // SetLastError flag set.

37 int errorCode = Marshal.GetLastWin32Error();

38 // Initializes and throws a new instance of the Win32Exception class

39 // with the specified error.

40 throw new Win32Exception(errorCode);

41 }

42 }

43 }

44

通过SendInput API函数提供虚拟键盘代码的调用即可模拟键盘输入。下面的代码模拟一个 KeyDown 和 KeyUp 过程，把他们连接起来就是一次按键过程。

1 private void SendKeyDown(short key) {

2 Input[] input = new Input[1];

3 input[0].type = INPUT.KEYBOARD;

4 input[0].ki.wVk = key;

5 input[0].ki.time = NativeMethods.GetTickCount();

6

7 if (NativeMethods.SendInput((uint)input.Length, input, Marshal.SizeOf(input[0]))

8 < input.Length) {

9 throw new Win32Exception(Marshal.GetLastWin32Error());

10 }

11 }

12

13 private void SendKeyUp(short key) {

14 Input[] input = new Input[1];

15 input[0].type = INPUT.KEYBOARD;

16 input[0].ki.wVk = key;

17 input[0].ki.dwFlags = KeyboardConstaint.KEYEVENTF\_KEYUP;

18 input[0].ki.time = NativeMethods.GetTickCount();

19

20 if (NativeMethods.SendInput((uint)input.Length, input, Marshal.SizeOf(input[0]))

21 < input.Length) {

22 throw new Win32Exception(Marshal.GetLastWin32Error());

23 }

24 }

然后自己实现一个 KeyBoardButton 控件用作按钮，用 Visual Studio 或者 SharpDevelop 为屏幕键盘设计 UI，然后在这些 Button 的 Click 事件里面模拟一个按键过程。

1

2 private void ButtonOnClick(object sender, EventArgs e) {

3 KeyboardButton btnKey = sender as KeyboardButton;

4 if (btnKey == null) {

5 return;

6 }

7

8 SendKeyCommand(btnKey);

9 }

10

11 private void SendKeyCommand(KeyboardButton keyButton) {

12 short key = keyButton.VKCode;

13 if (combinationVKButtonsMap.ContainsKey(key)) {

14 if (keyButton.Checked) {

15 SendKeyUp(key);

16 } else {

17 SendKeyDown(key);

18 }

19 } else {

20 SendKeyDown(key);

21 SendKeyUp(key);

22 }

23 }

其中 combinationVKButtonsMap 是一个 IDictionary<short, IList<KeyboardButton>>, key 存储的是VK\_SHIFT, VK\_CONTROL 等组合键的键盘码。左右两个按钮对应同一个键盘码，因此需要放在一个 List 里。标准键盘上的每一个键都有虚拟键码( VK\_CODE)与之对应。还有一些其他的常量，把它写在一个静态 class 里。 1 // KeyboardConstaint.cs 2 internal static class KeyboardConstaint { 3 internal static readonly short VK\_F1 = 0x70; 4 internal static readonly short VK\_F2 = 0x71; 5 internal static readonly short VK\_F3 = 0x72; 6 internal static readonly short VK\_F4 = 0x73; 7 internal static readonly short VK\_F5 = 0x74; 8 internal static readonly short VK\_F6 = 0x75; 9 internal static readonly short VK\_F7 = 0x76; 10 internal static readonly short VK\_F8 = 0x77; 11 internal static readonly short VK\_F9 = 0x78; 12 internal static readonly short VK\_F10 = 0x79; 13 internal static readonly short VK\_F11 = 0x7A; 14 internal static readonly short VK\_F12 = 0x7B; 15 16 internal static readonly short VK\_LEFT = 0x25; 17 internal static readonly short VK\_UP = 0x26; 18 internal static readonly short VK\_RIGHT = 0x27; 19 internal static readonly short VK\_DOWN = 0x28; 20 21 internal static readonly short VK\_NONE = 0x00; 22 internal static readonly short VK\_ESCAPE = 0x1B; 23 internal static readonly short VK\_EXECUTE = 0x2B; 24 internal static readonly short VK\_CANCEL = 0x03; 25 internal static readonly short VK\_RETURN = 0x0D; 26 internal static readonly short VK\_ACCEPT = 0x1E; 27 internal static readonly short VK\_BACK = 0x08; 28 internal static readonly short VK\_TAB = 0x09; 29 internal static readonly short VK\_DELETE = 0x2E; 30 internal static readonly short VK\_CAPITAL = 0x14; 31 internal static readonly short VK\_NUMLOCK = 0x90; 32 internal static readonly short VK\_SPACE = 0x20; 33 internal static readonly short VK\_DECIMAL = 0x6E; 34 internal static readonly short VK\_SUBTRACT = 0x6D; 35 36 internal static readonly short VK\_ADD = 0x6B; 37 internal static readonly short VK\_DIVIDE = 0x6F; 38 internal static readonly short VK\_MULTIPLY = 0x6A; 39 internal static readonly short VK\_INSERT = 0x2D; 40 41 internal static readonly short VK\_OEM\_1 = 0xBA; // ';:' for US 42 internal static readonly short VK\_OEM\_PLUS = 0xBB; // '+' any country 43 44 internal static readonly short VK\_OEM\_MINUS = 0xBD; // '-' any country 45 46 internal static readonly short VK\_OEM\_2 = 0xBF; // '/?' for US 47 internal static readonly short VK\_OEM\_3 = 0xC0; // '`~' for US 48 internal static readonly short VK\_OEM\_4 = 0xDB; // '[{' for US 49 internal static readonly short VK\_OEM\_5 = 0xDC; // '\|' for US 50 internal static readonly short VK\_OEM\_6 = 0xDD; // ']}' for US 51 internal static readonly short VK\_OEM\_7 = 0xDE; // ''"' for US 52 internal static readonly short VK\_OEM\_PERIOD = 0xBE; // '.>' any country 53 internal static readonly short VK\_OEM\_COMMA = 0xBC; // ',<' any country 54 internal static readonly short VK\_SHIFT = 0x10; 55 internal static readonly short VK\_CONTROL = 0x11; 56 internal static readonly short VK\_MENU = 0x12; 57 internal static readonly short VK\_LWIN = 0x5B; 58 internal static readonly short VK\_RWIN = 0x5C; 59 internal static readonly short VK\_APPS = 0x5D; 60 61 internal static readonly short VK\_LSHIFT = 0xA0; 62 internal static readonly short VK\_RSHIFT = 0xA1; 63 internal static readonly short VK\_LCONTROL = 0xA2; 64 internal static readonly short VK\_RCONTROL = 0xA3; 65 internal static readonly short VK\_LMENU = 0xA4; 66 internal static readonly short VK\_RMENU = 0xA5; 67 68 internal static readonly short VK\_SNAPSHOT = 0x2C; 69 internal static readonly short VK\_SCROLL = 0x91; 70 internal static readonly short VK\_PAUSE = 0x13; 71 internal static readonly short VK\_HOME = 0x24; 72 73 internal static readonly short VK\_NEXT = 0x22; 74 internal static readonly short VK\_PRIOR = 0x21; 75 internal static readonly short VK\_END = 0x23; 76 77 internal static readonly short VK\_NUMPAD0 = 0x60; 78 internal static readonly short VK\_NUMPAD1 = 0x61; 79 internal static readonly short VK\_NUMPAD2 = 0x62; 80 internal static readonly short VK\_NUMPAD3 = 0x63; 81 internal static readonly short VK\_NUMPAD4 = 0x64; 82 internal static readonly short VK\_NUMPAD5 = 0x65; 83 internal static readonly short VK\_NUMPAD5NOTHING = 0x0C; 84 internal static readonly short VK\_NUMPAD6 = 0x66; 85 internal static readonly short VK\_NUMPAD7 = 0x67; 86 internal static readonly short VK\_NUMPAD8 = 0x68; 87 internal static readonly short VK\_NUMPAD9 = 0x69; 88 89 internal static readonly short KEYEVENTF\_EXTENDEDKEY = 0x0001; 90 internal static readonly short KEYEVENTF\_KEYUP = 0x0002; 91 92 internal static readonly int GWL\_EXSTYLE = -20; 93 internal static readonly int WS\_DISABLED = 0X8000000; 94 internal static readonly int WM\_SETFOCUS = 0X0007; 95 }

屏幕键盘必须是一个不能获得输入焦点的窗体，在这个窗体的构造函数里，可以安装 一个全局鼠标钩子，再通过调用 SetWindowLong API 函数完成。

1UserActivityHook hook = new UserActivityHook(true, true);

2hook.MouseActivity += HookOnMouseActivity;

3

4private void HookOnMouseActivity(object sener, HookEx.MouseExEventArgs e) {

5 Point location = e.Location;

6

7 if (e.Button == MouseButtons.Left) {

8 Rectangle captionRect = new Rectangle(this.Location, new Size(this.Width,

9 SystemInformation.CaptionHeight));

10 if (captionRect.Contains(location)) {

11 NativeMethods.SetWindowLong(this.Handle, KeyboardConstaint.GWL\_EXSTYLE,

12 (int)NativeMethods.GetWindowLong(this.Handle, KeyboardConstaint.GWL\_EXSTYLE)

13 & (~KeyboardConstaint.WS\_DISABLED));

14 NativeMethods.SendMessage(this.Handle, KeyboardConstaint.WM\_SETFOCUS, IntPtr.Zero, IntPtr.Zero);

15 } else {

16 NativeMethods.SetWindowLong(this.Handle, KeyboardConstaint.GWL\_EXSTYLE,

17 (int)NativeMethods.GetWindowLong(this.Handle, KeyboardConstaint.GWL\_EXSTYLE) |

18 KeyboardConstaint.WS\_DISABLED);

19 }

20 }

21}

鼠标单击标题栏，让屏幕键盘可以接收焦点，并激活，单击其他部分则不激活窗体（如果激活了，其他程序必然取消激活，输入就无法进行了），这样才可以进行输入，并且保证了可以拖动窗体到其他位置。