[1. Windows的消息处理 3](#_Toc465094163)

[1-1. 基本处理 3](#_Toc465094164)

[1-2. Windows 的消息一览 3](#_Toc465094165)

[1-3. PostMessage， Send Message 3](#_Toc465094166)

[2. 在TextBox控件里按键盘，是Form还是TextBox先获得键盘处理事件 3](#_Toc465094167)

[3. GUI的消息处理 4](#_Toc465094168)

[3-1. 通用消息处理 4](#_Toc465094169)

[3-1-1. Application.Run 4](#_Toc465094170)

[3-1-2. ThreadContext.LocalModelMessageLoop 5](#_Toc465094171)

[3-1-3. Control.PreProcessControlMessageInternal 7](#_Toc465094172)

[3-1-4. Control.preProcessMessage 8](#_Toc465094173)

[3-1-5. Control.ProcessCmdKey 9](#_Toc465094174)

[3-1-6. Native 的Dispatch Method 应该是直接到Control.WndProc 去了 10](#_Toc465094175)

[3-1-7. 所有的方法都是返回true的时候结束处理 10](#_Toc465094176)

[3-2. Kyeboard Event 处理 10](#_Toc465094177)

[3-2-1. ProcessCmdKey 方法在所有其他事件处理方法之前被调用 10](#_Toc465094178)

[3-2-2. ProcessKeyMessage 10](#_Toc465094179)

[3-2-3. ProcessKeyPreview 11](#_Toc465094180)

[3-2-4. Form 重载了Control的ProcessKeyPreview 11](#_Toc465094181)

[3-2-5. Form 的KeyPreview 属性 11](#_Toc465094182)

[3-2-6. ProcessKeyEventArgs 12](#_Toc465094183)

[3-2-7. OnKeyDown，OnKeyUp，OnKeyProcess 14](#_Toc465094184)

[3-2-8. KeyPressEventArgs 14](#_Toc465094185)

[3-2-9. KeyEventArgs 15](#_Toc465094186)

[4. 可以认为Control 是WinForm的基类 17](#_Toc465094187)

[4-1. Control 类的基类及子类 17](#_Toc465094188)

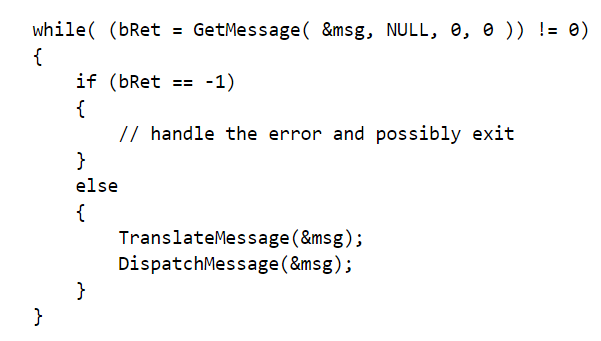
[4-2. Control 继承自Component，但Component 做的事情很少 19](#_Toc465094189)

[4-3. TextBoxBase 和 ButtonBase 19](#_Toc465094190)

[4-4. Form 类继承自ContainerControl 19](#_Toc465094191)

# Windows的消息处理

## 基本处理

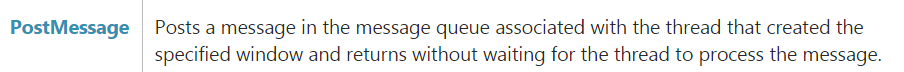


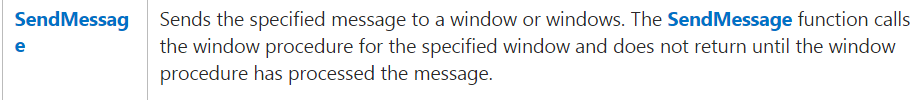
注意，这里是没有PreTranslateMessage 的定义的，PreTranslateMessage是 .NET 自己加的

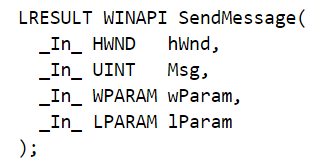
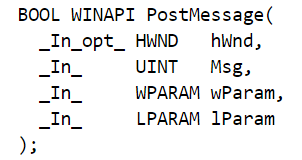
## Windows 的消息一览

[**Windows Message List.pdf**](Windows%20Message%20List.pdf)

## PostMessage， Send Message



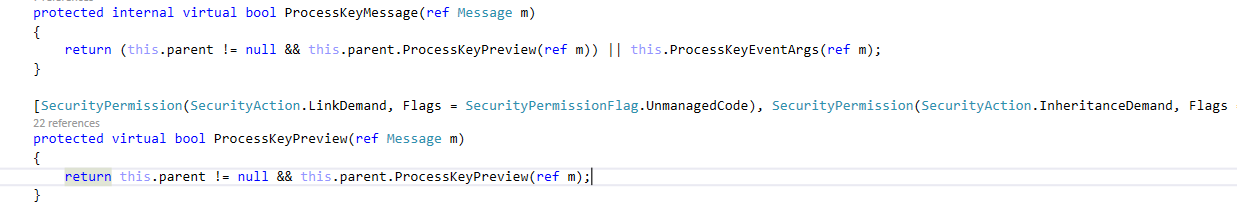


# 在TextBox控件里按键盘，是Form还是TextBox先获得键盘处理事件

按照windows 的消息循环的处理，应该是TextBox 这个hHandle 先获得键盘处理事件。

那为什么在.NET 里好像是Form里的方法先调用呢？ 那是因为.NET 的winform framework通过一个机制把消息从TextBox先通过parent属性一层层传递到了Form，让form先调用，然后才是TextBox自己的默认事件处理方法。其实就是通过Control 基类的ProcessKeyMessage 和ProcessKeyPreview 来实现的。



详细的内容可以见下面的分析。

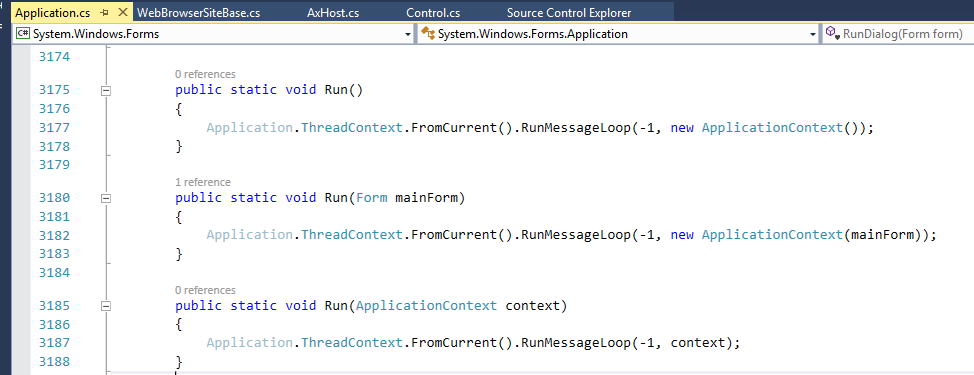
# GUI的消息处理

用IL反编译System.Windows.Forms.dll，可以例子中用到的所有类

## 通用消息处理

### Application.Run

消息循环不在Application 类中，在ThreadContext 类中



### ThreadContext.LocalModelMessageLoop



* **基本处理**

While

{

GetMessage() --Native

If(!PreTranslateMessage) -- .NET

{

TranslateMessage() --Native

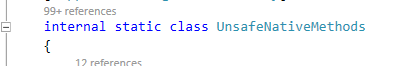
DispatchMessage（）--.NET

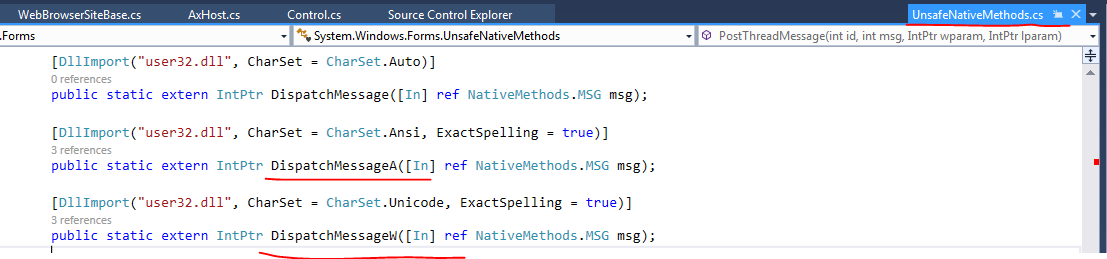
}

}

* **UnSafeNativeMethods**

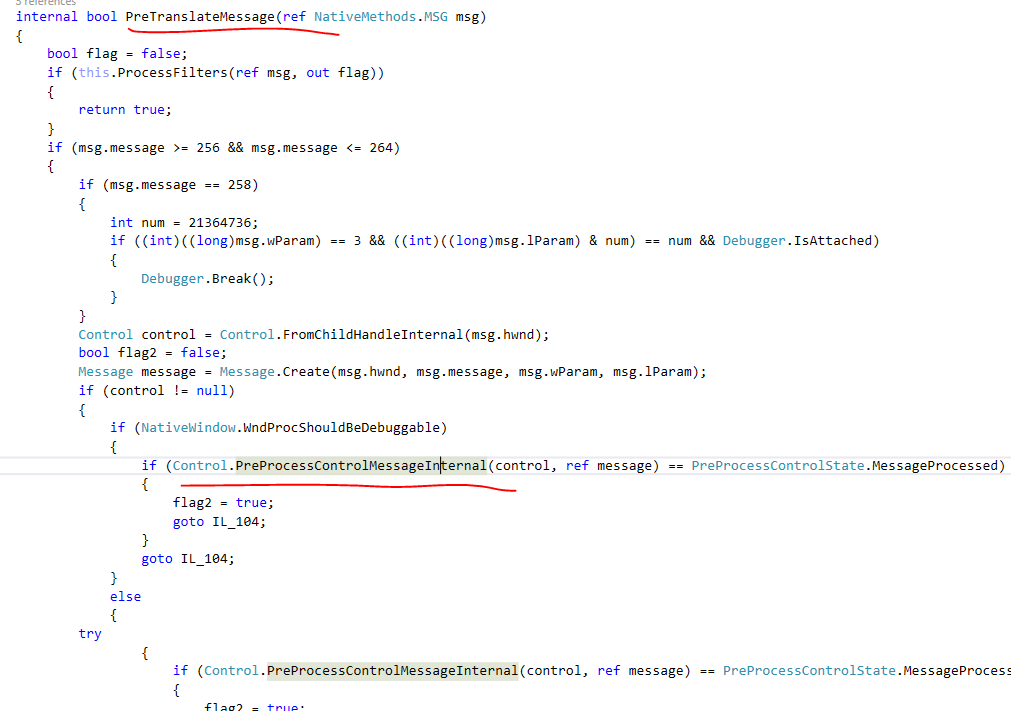
凡是调用这个类中方法的，都直接到Native Code 去了，没有.NET Code 了





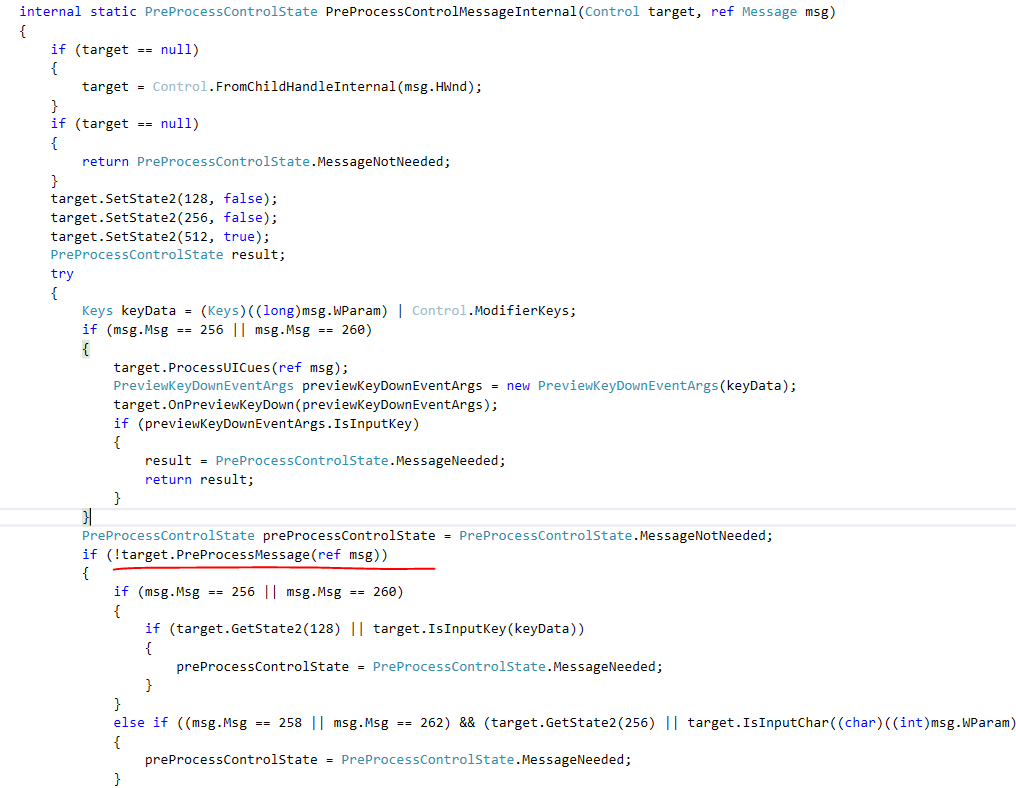
* **在Windows的Native的TranslateMessage之前，.net 加了自己的PreTranslateMessage**

这里直接调到Control类的PreProcessControlMessageInternal 去了



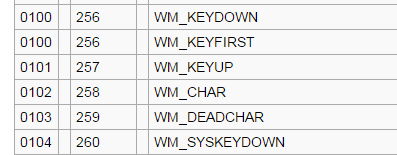
### Control.PreProcessControlMessageInternal

然后到了PreProcessMessage



### Control.preProcessMessage

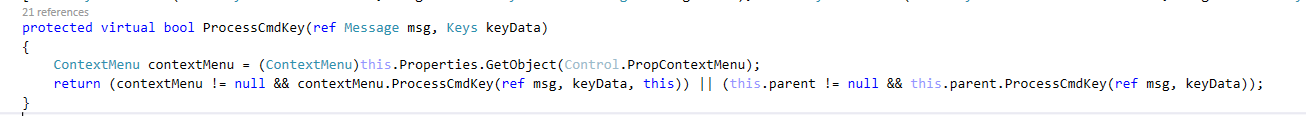
如果是键盘的KeyDown消息，那么调用Control.ProcessCmdKey()



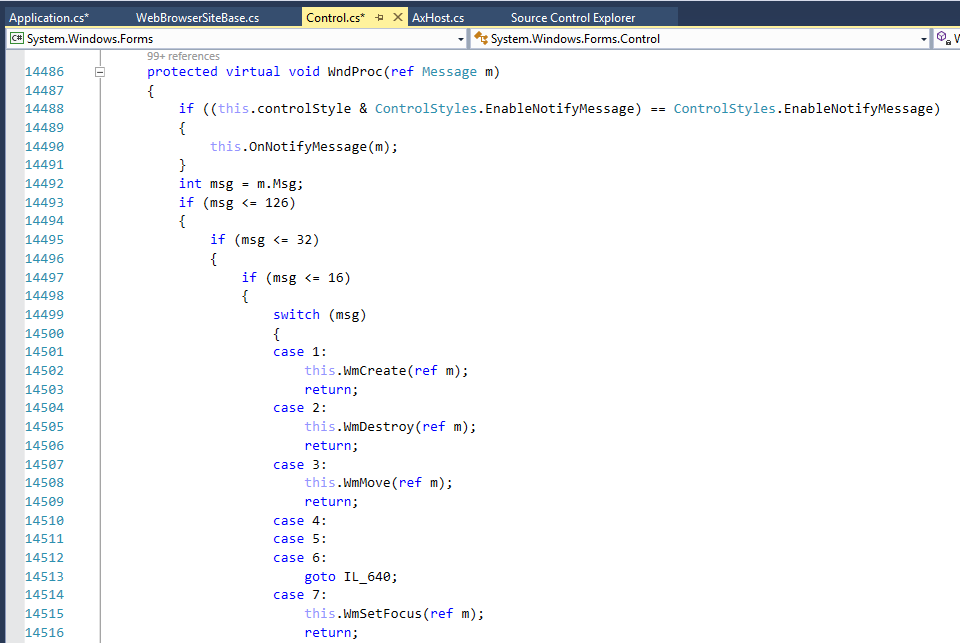


### Control.ProcessCmdKey

可以看到这个ProcessCmd一层层跑到顶层Form上去处理去了，各个子控件应该不需要处理这个方法



### Native 的Dispatch Method 应该是直接到Control.WndProc 去了



### 所有的方法都是返回true的时候结束处理

跟一般的处理有点不一样的是，所有的处理方法都是返回为true的时候，后续的其他处理链就不处理了

跟一般的方法的返回值的处理不一样

## Kyeboard Event 处理

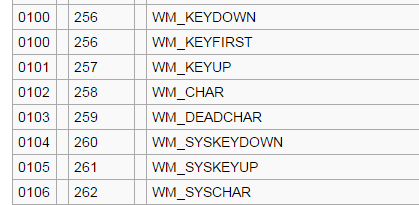
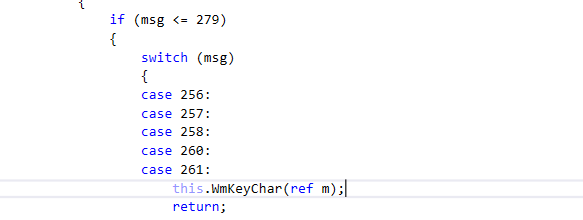
### ProcessCmdKey 方法在所有其他事件处理方法之前被调用

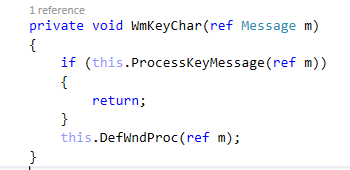
因为ProcessCmdKey 是在preTranslateMessage()的时候就已经被调用了

### ProcessKeyMessage

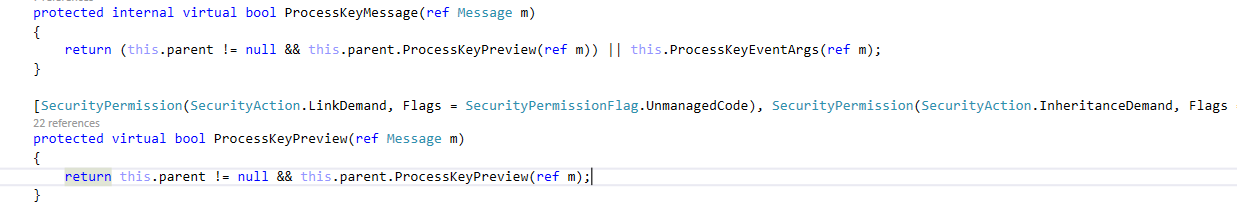
Control 类的wndproc 调用WmKeyChar，KeyChar 又调用了ProcessKeyMessage 方法。

所有如果重载了这个方法，那么最先被执行。



ProcessKeyMessage 并不立即处理消息，而是通过ProcessKeyPreivew 方法，先一层一层的把键盘事件传递给了父类。



而且如果父类已经处理了，那么按照.net || 的执行逻辑，自己真正的ProcessKeyEventArgs 方法就不执行了。

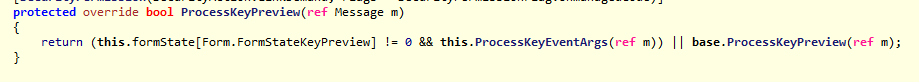
由于这个方法包含了消息的向上传递链，所以一般还是不直接重载这个方法为好。

### ProcessKeyPreview

这个就是用来处理消息从底层传递到上层的。Control 的默认的处理就是如果有父控件，直接调用父控件的此方法。

由于这个方法是Control 类定义的，所有继承自Control的控件都有此虚方法。**这个方法也是消息链传递的关键。**

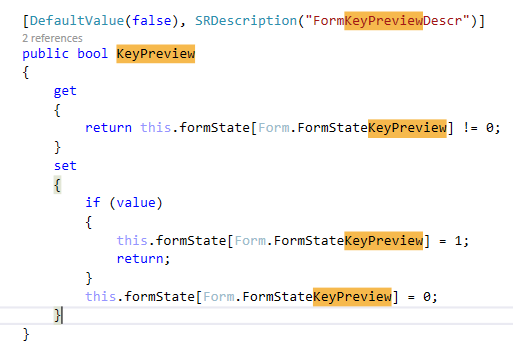
### Form 重载了Control的ProcessKeyPreview

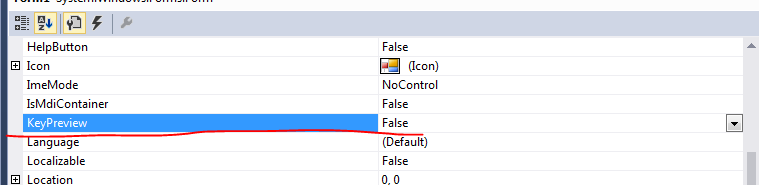


由于Form 就是顶层控件，所有再用parent 不为null，调用parent的ProcessKeyPreview 就没意义了。

### Form 的KeyPreview 属性

根据Form的ProcessKeyPreview 方法， 只有当这个属性设置为true的时候，Form才会先触发自己的键盘事件处理函数ProcessKeyEventArgs方法，即所谓的Preivew



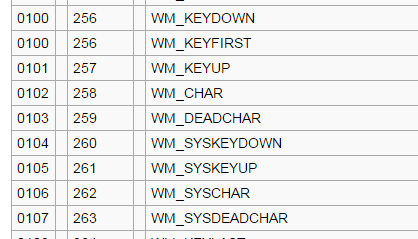
### ProcessKeyEventArgs

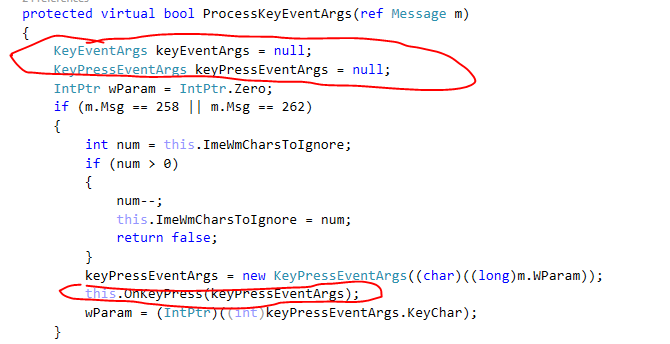
这个名字看起来像是一个参数类，但确实一个处理方法。可见微软也有不靠谱的时候。

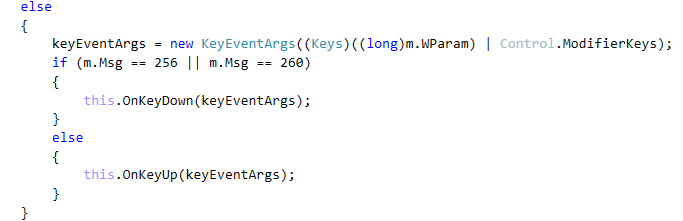
这个是键盘处理的核心函数，所有事件的触发都是通过这个函数实现的。如果重载了这个函数，那么所有的Keyup，KeyDown，KeyPress 就都可以忽略了。

键盘处理事件的参数KeyEventArgs 和KeyPressEventArgs 就是在这里初始化的。

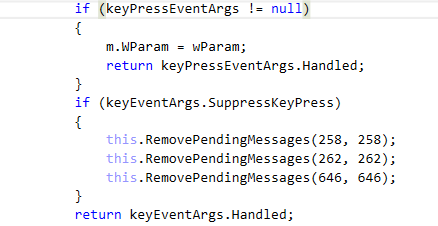
如果是键盘事件产生了字符，调用OnKeyProcess







通过KeyEventArgs 和KeyPressEventArgs 的Handled 来控制事件链的进一步传递



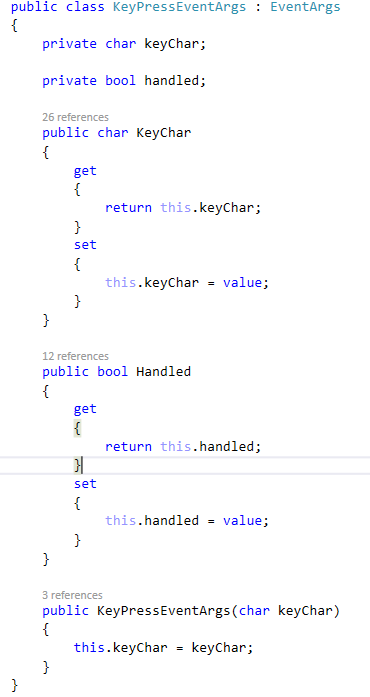
### OnKeyDown，OnKeyUp，OnKeyProcess

对具体的事件处理函数的调用



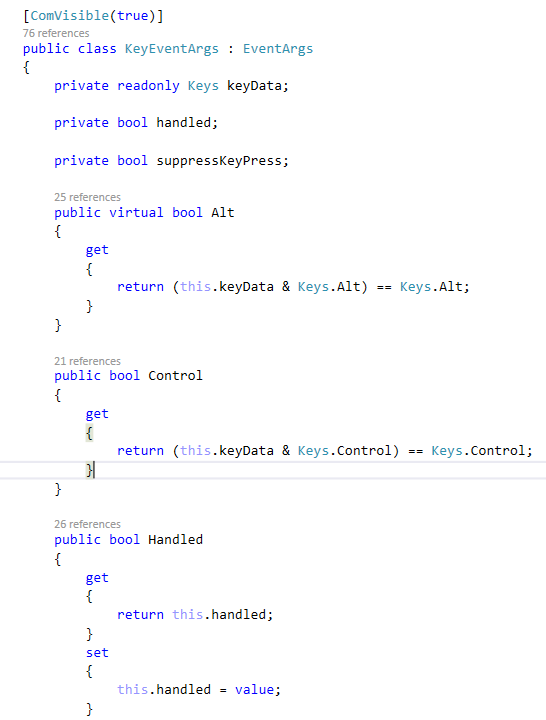
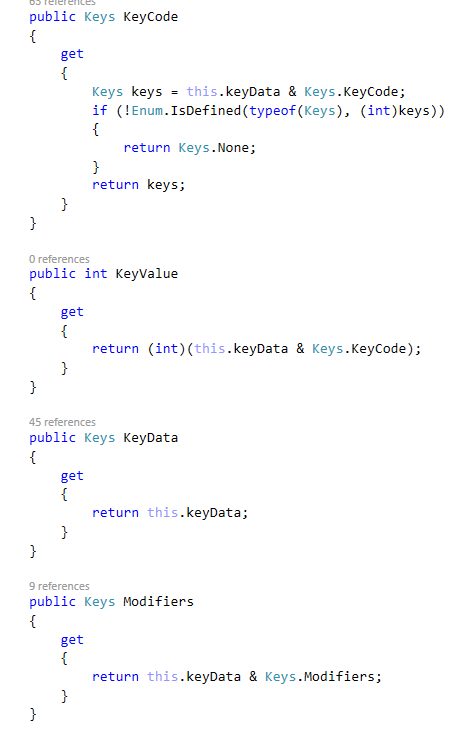
### KeyPressEventArgs

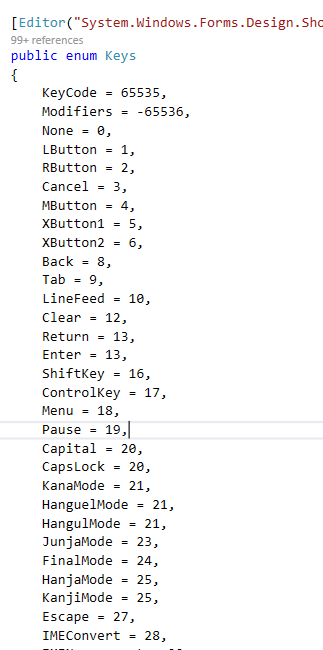
里面的属性是一个KeyChar Char，所以能触发这个事件的，必定是按键产生了一个字符。而功能键比如F1是不会产生这个事件的。



### KeyEventArgs

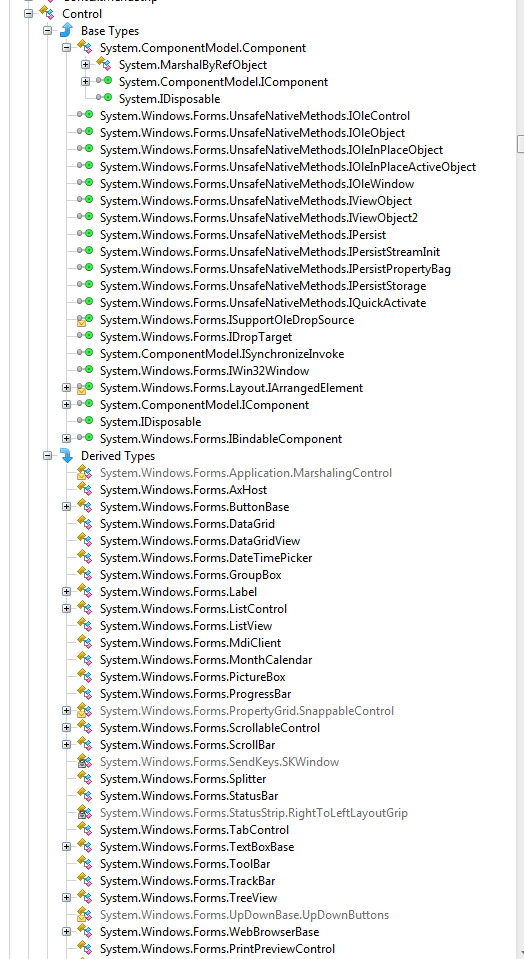
里面用的最多的应该是他的KeyCode 属性

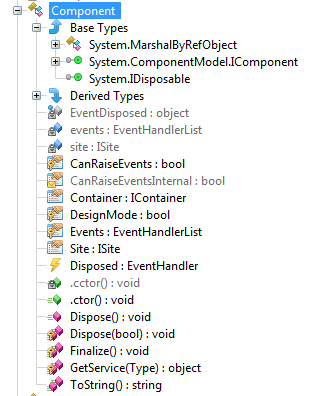


# 可以认为Control 是WinForm的基类

## Control 类的基类及子类



## Control 继承自Component，但Component 做的事情很少



## TextBoxBase 和 ButtonBase

这两个都是常用的TextBox和Button的基类

## Form 类继承自ContainerControl

最终也是继承自Control

