事件委托用于定义事件的签名。 通常，特定事件委托与特定事件数据类相对应。 按照约定，.NET Framework 中的事件具有签名 EventName(sender, e)，其中 sender 是提供对触发事件的类或结构的引用的 [Object](https://msdn.microsoft.com/zh-cn/library/system.object%28v=vs.110%29.aspx)，e 是 [EventArgs](https://msdn.microsoft.com/zh-cn/library/system.eventargs%28v=vs.110%29.aspx) 对象或从提供事件数据的 [EventArgs](https://msdn.microsoft.com/zh-cn/library/system.eventargs%28v=vs.110%29.aspx) 派生的对象。 委托定义通常则采用 EventNameHandler(sender, e) 的形式。

下面的示例声明一个名为 AlarmEvent 的事件。 该示例节选自名为 Alarm 的类的示例，其完整的源代码如下所示。 请注意，它具有 AlarmEventHandler 委托的签名。

public event AlarmEventHandler AlarmEvent;

定义事件实现后，您必须确定引发事件的时间。 通过在定义事件的类或派生类中调用受保护的 OnEventName 方法来引发事件。 随后，OnEventName方法引发事件

受保护的 OnEventName方法也允许派生类重写事件，而不必向其附加委托。 派生类必须始终调用基类的 OnEventName方法以确保注册的委托接收到事件。

自定义控件，数组类型字段

private string[] summaryColumns;

[Browsable(true), Category("Summary")]

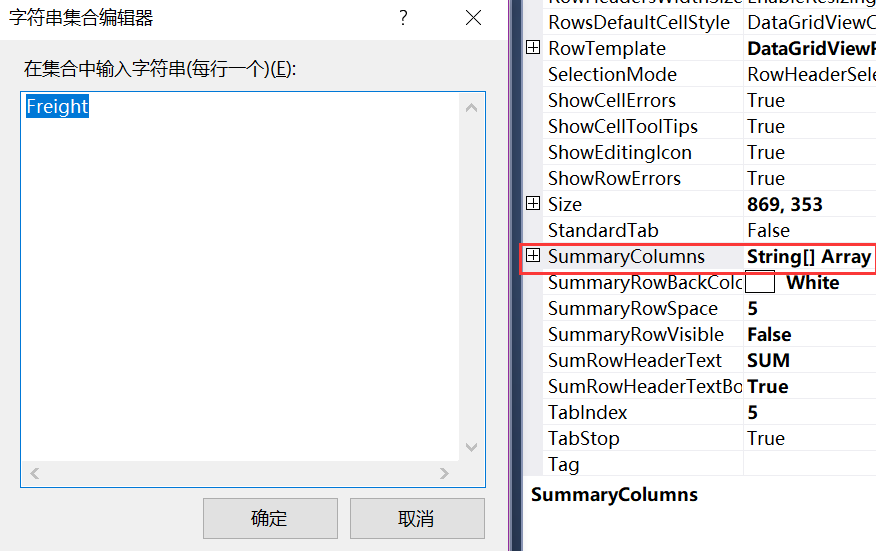
public string[] SummaryColumns

{

get { return summaryColumns; }

set { summaryColumns = value; }

}



未在本地计算机上注册“Microsoft.Jet.OLEDB.4.0”提供程序

如果是64位平台，也可能出现这种情况；解决方法：生成->配置管理器->平台->点击Any Cpu选项卡->新建->新建平台->X86

键的预处理

　　像其他消息一样，键盘消息是在窗体或控件的WndProc方法中处理的。窗体或控件在处理键盘消息之前，PreProcessMessage方法会调用一个或多个方法，这些方法可被重写以处理特殊的字符键和物理按键。

　　其实我们按下一个键时，**KeyDown**和**KeyPress**是都要发生的，而且两个人看上去没有什么区别。但是其实两者还是有实质的区别的。

　　例如在一个多行文本框里面输入字符，为什么我们没有定义一个键盘响应事件，但是键盘却响应了，把字符输入到了文本框呢？其实我觉得因为这个事件就是系统默认的一个消息响应，它就是**KeyPress**。如果你自己还给**KeyPress**定义了自己的响应的话，那么程序会先执行你定义的程序，然后完成系统要求的显示字符事件的响应过程，但是如果你在自己定义**KeyPress**事件如：

private void \_TextBox\_KeyPress(object sender, KeyPressEventArgs e)

{

　　　　e.Handled=true;

}

　　那么系统要求的显示字符响应过程就不会被执行，原因就是这个语句的意思就是这个消息响应已经完成。但是如果你在**KeyDown**,**KeyUp**里面加这条语句，显示字符串等消息还是会发生，这说明系统的显示字符串和**KeyDown**,**KeyUp**是没有关系的,而对于回车和退格消息响应则是由**KeyDown**控制,增加那条语句则回车换行不会执行，退格响应也不会执行。

　　综上：要屏蔽键盘上的一个字符的输入，可以在**KeyPress**里面进行增加语句，而要屏蔽回车和退格则可以在**KeyDown**里面进行增加语句。

//KeyCode、KeyData、KeyValue对于字母键只记录大写的值

//没有KeyChar

//必须先将KeyPreview的属性设置为true

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Alt && e.Control && e.KeyCode == Keys.F2)

{

MessageBox.Show("You press the Alt and Ctrl and F2 buttons!");

}

if (e.KeyData == Keys.Up)

{

MessageBox.Show("You press the Up buttons!");

}

if (e.KeyValue == 27)

{

MessageBox.Show("You press the Esc buttons!");

}

}

//只能返回一个字符的ASCII码

//不能处理功能键、编辑键、组合键

//KeyChar能区分大小写

//KeyChar字存在于KeyPress中

private void Form1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == 65)

{

MessageBox.Show("You press the A buttons!");

}

if (e.KeyChar == 97)

{

MessageBox.Show("You press the a buttons!");

}

//KeyChar是不区分数字是否在大小哪个键盘的

if (e.KeyChar == 48)

{

MessageBox.Show("You press the 0 buttons!");

}

}

2，按键盘的Ctrl+A，KeyDwon和KeyUp个触发两次。

第一次**KeyDown**里的KeyEventArgs的值如下：

KeyCode：Keys.ContrlKey

KeyData：Keys.ContrlKey

KeyValue:17

第二次**KeyDown**里的KeyEventArgs的值如下：

KeyCode：Keys.A

KeyData：Keys.ContrlKey + Keys.A

KeyValue:65

**5） 某些特殊键在控件上被默认处理的解决办法**

比如，TextBox上的TAB键就被默认处理的，在TextBox上按Tab键，将不会触发KeyDown已经后面的KeyPress和KeyUp事件，解决的办法是重新TextBox控件的IsInputKey方法，这个在MSDN上有说明，特实践证明了下，确实可行。

1. class TextBoxEx :System.Windows.Forms.TextBox
2. {
4. protected override bool IsInputKey(System.Windows.Forms.Keys keyData)
5. {
6. if (keyData == System.Windows.Forms.Keys.Tab)
7. {
8. return true;
9. }
10. return base.IsInputKey(keyData);
11. }
12. }
13. // 摘要:

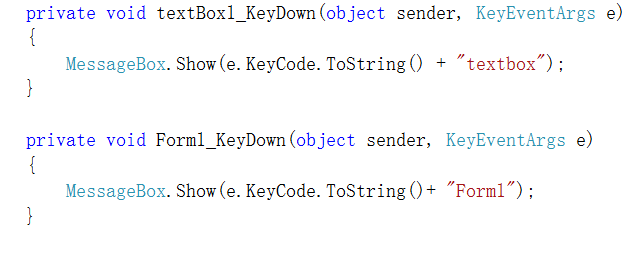
// 获取或设置一个值，该值指示在将键事件传递到具有焦点的控件前，窗体是否将接收此键事件。

//

// 返回结果:

// 如果窗体将接收所有键事件，则为 true；如果窗体上当前选定控件接收键事件，则为 false。默认值为 false。

public bool KeyPreview { get; set; }



KeyPreview设置为true时，先响应form，然后textbox

输入焦点离开控件时发生

When you change the focus by using the keyboard (TAB, SHIFT+TAB, and so on), by calling the [Select](https://msdn.microsoft.com/zh-cn/library/7wt11hea(v=vs.110).aspx) or [SelectNextControl](https://msdn.microsoft.com/zh-cn/library/system.windows.forms.control.selectnextcontrol(v=vs.110).aspx) methods, or by setting the [ContainerControl.ActiveControl](https://msdn.microsoft.com/zh-cn/library/system.windows.forms.containercontrol.activecontrol(v=vs.110).aspx) property to the current form, focus events occur in the following order:

protected override void OnLeave(EventArgs e)

{

base.OnLeave(e);

Hide();

}

private void Form1\_Click(object sender, EventArgs e)

{

this.ActiveControl = label1;

}

DataGridViewComboBoxColumn绑定数据源

数据库字段是number类型的，要指定列是decimal

DataGridViewComboBoxColumn column2 = new DataGridViewComboBoxColumn();

column2.HeaderText = "分组";

DataTable dtgroup = new DataTable();

dtgroup.Columns.Add("groupnum",typeof(decimal));

for(int i=1;i<11;i++)

{

dtgroup.Rows.Add(i);

// column2.Items.Add((decimal)i);

字符类型的要用tostring

// column2.Items.Add(i.ToString());

}

column2.DisplayMember = "groupnum";

column2.ValueMember = "groupnum";

column2.DataSource = dtgroup;

dataGridView1.Columns.Add(column2);

Control.IsInputKey 方法

确定指定的键是常规输入键还是需要预处理的特殊键。

调用 **IsInputKey** 方法以确定 keyData 参数指定的键是否是控件需要的输入键。此方法在窗口消息预处理过程中调用，以确定指定的输入键是应经过预处理还是直接发送到控件。如果 **IsInputKey** 返回 **true**，则将指定的键直接发送到控件。但是，如果 **IsInputKey** 返回 **false**，则指定的键需要预处理，且仅当预处理阶段不使用时，才能发送到控件。

交给Control.ProcessDialogKey 方法预处理

预处理的键包括 Tab、Return、Esc 以及向上键、向下键、向左键和向右键。

DataGridView用RichTextBox做编辑控件。实现IDataGridViewEditingControl接口时。回车键要输入键使用

protected override bool IsInputKey(Keys keyData)

{

Keys keys = keyData & Keys.KeyCode;

if (keys == Keys.Return)

{

return this.Multiline;

}

return base.IsInputKey(keyData);

}

Control.ProcessDialogKey 方法

在消息预处理过程中调用此方法，以处理对话框字符，比如 Tab、Return、Esc 和箭头键。只有当 [IsInputKey](https://msdn.microsoft.com/zh-cn/library/system.windows.forms.control.isinputkey(v=vs.80).aspx) 方法指示控件未在处理该键时，才调用此方法。

DataGridView中回车移动光标到下个cell

protected override bool ProcessDialogKey(Keys keyData)

{

//if the key pressed is "return" then tell the datagridview to move to the next cell

if (keyData == Keys.Enter)

{

MoveToNextCell();

return true;

}

else

return base.ProcessDialogKey(keyData);

}

public void MoveToNextCell()

{

int CurrentColumn, CurrentRow;

//get the current indicies of the cell

CurrentColumn = this.CurrentCell.ColumnIndex;

CurrentRow = this.CurrentCell.RowIndex;

//if cell is at the end move it to the first cell of the next row

//other with move it to the next cell

if (CurrentColumn == this.Columns.Count - 1 && CurrentRow != this.Rows.Count - 1)

{

base.ProcessDataGridViewKey(new KeyEventArgs(Keys.Home));

base.ProcessDataGridViewKey(new KeyEventArgs(Keys.Down));

}

else

base.ProcessDataGridViewKey(new KeyEventArgs(Keys.Right));

}

DataGridViewTextBoxEditingControl.EditingControlWantsInputKey 方法 (Keys, Boolean)

Determines whether the specified key is a regular input key that the editing control should process or a special key that the [DataGridView](https://msdn.microsoft.com/zh-cn/library/system.windows.forms.datagridview(v=vs.110).aspx) should process.

public bool EditingControlWantsInputKey(Keys keyData, bool dataGridViewWantsInputKey)

{

switch ((keyData & Keys.KeyCode))

{

case Keys.Return:

if ((((keyData & (Keys.Alt | Keys.Control | Keys.Shift)) == Keys.Shift) && this.Multiline))

{

return true;

}

break;

case Keys.Left:

case Keys.Right:

case Keys.Up:

case Keys.Down:

return true;

}

return !dataGridViewWantsInputKey;

}

**WM\_NCCALCSIZE message**

**https://msdn.microsoft.com/en-us/library/ms632634(VS.85).aspx**

Sent when the size and position of a window's client area must be calculated. By processing this message, an application can control the content of the window's client area when the size or position of the window changes.

A window receives this message through its [**WindowProc**](https://msdn.microsoft.com/en-us/library/ms633573%28v=vs.85%29.aspx) function.

# Component 类

<https://msdn.microsoft.com/zh-cn/library/system.componentmodel.component(VS.80).aspx>

**Component** 是公共语言运行库中按引用封送的所有组件的基类。**Component** 可以进行远程处理，并且派生自 [MarshalByRefObject](https://msdn.microsoft.com/zh-cn/library/system.marshalbyrefobject%28v=vs.80%29.aspx) 类。**Component** 提供 **IComponent** 接口的实现。[MarshalByValueComponent](https://msdn.microsoft.com/zh-cn/library/system.componentmodel.marshalbyvaluecomponent%28v=vs.80%29.aspx) 提供按值封送的 **IComponent** 的实现。

可以在任何实现 [IContainer](https://msdn.microsoft.com/zh-cn/library/system.componentmodel.icontainer%28v=vs.80%29.aspx) 接口的对象中承载 **Component**，并且可以从其容器中查询和获取服务。容器会为它所包含的每个 **Component** 创建一个 [ISite](https://msdn.microsoft.com/zh-cn/library/system.componentmodel.isite%28v=vs.80%29.aspx)。容器使用该站点来管理 **Component**，而 **Component** 则使用该站点与它的容器进行通信。

**Component** 应通过调用其 [Dispose](https://msdn.microsoft.com/zh-cn/library/system.componentmodel.component.dispose%28v=vs.80%29.aspx) 方法来显式释放资源，而不通过隐式调用 [Finalize](https://msdn.microsoft.com/zh-cn/library/system.object.finalize%28v=vs.80%29.aspx) 方法来等待自动内存管理。当释放一个 [Container](https://msdn.microsoft.com/zh-cn/library/system.componentmodel.container%28v=vs.80%29.aspx) 时，同时会释放该 **Container** 中的所有组件

严重性 代码 说明 项目 文件 行 禁止显示状态

错误 Resx 文件无效。无法加载位于行 1691、位置 5 的数据中的类型 ，原因是它在构造时引发下列异常:此方法显式使用的 CAS 策略已被 .NET Framework 弃用。若要出于兼容性原因而启用 CAS 策略，请使用 NetFx40\_LegacySecurityPolicy 配置开关。有关详细信息，请参见 http://go.microsoft.com/fwlink/?LinkID=155570。 JGWL.WindowsForm C:\wms2015\JGWL.WindowsForm\Form\_Sys\_Home.resx

将那个<data……></data>删除

[此方法显式使用的 CAS 策略已被 .NET Framework 弃用。若要出于兼容性原因而启用 CAS 策略，请使用 NetFx40\_LegacySecurityPolicy 配置开关](http://www.cnblogs.com/zeroone/archive/2012/03/17/2403925.html)

<runtime>

<NetFx40\_LegacySecurityPolicy enabled="true"/>

</runtime>

混合模式程序集是针对“v2.0.50727”版的运行时生成的，在没有配置其他信息的情况下，无法在 4.0 运行时中加载该程序集。

http://blog.csdn.net/dongzengwu/article/details/50428155

**<startup useLegacyV2RuntimeActivationPolicy="true">  
 <supportedRuntime version="v4.0" sku = ".NETFramework,Version=v4.0"/>  
 <supportedRuntime version="v2.0.50727"/>  
 </startup>**

[DefaultProperty("GlobalPaletteMode")]

DefaultProperty     是设置控件默认的属性的。这里是其Text属性。就是当你选择这个控件的时候，在属性窗口中自动被选中的是Text属性。

[ComVisible(true)]

[ComVisibleAttribute] 指示应用该属性的对象是否对COM可见   
COM = Component Object Model，微软的上一代编程模型

[EditorBrowsable(EditorBrowsableState.Never)]

EditorBrowsableState.Never的枚举说明是：该属性或方法始终不能在编辑器中浏览。

意思就是说，让使用者在调用的时候无法智能感知出Object默认的方法

