.NET 2.0 - WinForm Control - DataGridView 编程 36 计（一）收藏

新一篇: .NET 窗体应用中的多线程误区~~

.NET 2.0 - WinForm Control - DataGridView 编程 36 计（一）

目录:

① DataGridView 取得或者修改当前单元格的内容：

当前单元格指的是 **DataGridView** 焦点所在的单元格，它可以通过 **DataGridView** 对象的 **CurrentCell** 属性取得。如果当前单元格不存在的时候，返回 **Nothing(C#是 null)**

[VB.NET]

```
' 取得当前单元格内容
Console.WriteLine(DataGridView1.CurrentCell.Value)
' 取得当前单元格的列 Index
Console.WriteLine(DataGridView1.CurrentCell.ColumnIndex)
```

```
' 取得当前单元格的行 Index
Console.WriteLine(DataGridView1.CurrentCell.RowIndex)
```

```
[C#]
// 取得当前单元格内容
Console.WriteLine(DataGridView1.CurrentCell.Value);
// 取得当前单元格的列 Index
Console.WriteLine(DataGridView1.CurrentCell.ColumnIndex);
// 取得当前单元格的行 Index
Console.WriteLine(DataGridView1.CurrentCell.RowIndex);
```

另外，使用 DataGridView.CurrentCellAddress 属性（而不是直接访问单元格）来确定单元格所在的行：DataGridView.CurrentCellAddress.Y 和列： DataGridView.CurrentCellAddress.X 。这对于避免取消共享行的共享非常有用。

当前的单元格可以通过设定 DataGridView 对象的 CurrentCell 来改变。可以通过 CurrentCell 来设定 DataGridView 的激活单元格。将 CurrentCell 设为 Nothing(null) 可以取消激活的单元格。

```
[VB.NET]
' 设定 (0, 0) 为当前单元格
DataGridView1.CurrentCell = DataGridView1(0, 0)
```

```
[C#]
// 设定 (0, 0) 为当前单元格
DataGridView1.CurrentCell = DataGridView1[0, 0];
```

在整行选中模式开启时，你也可以通过 CurrentCell 来设定选定行。

```
        /// <summary>
        /// 向下遍历
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button4_Click(object sender, EventArgs e)
        {
            int row = this.dataGridView1.CurrentRow.Index + 1;
            if (row > this.dataGridView1.RowCount - 1)
                row = 0;
            this.dataGridView1.CurrentCell = this.dataGridView1[0, row];
```

```
        }

        /// <summary>
        /// 向上遍历
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button5_Click(object sender, EventArgs e)
        {
            int row = this.dataGridView1.CurrentRow.Index - 1;
            if (row < 0)
                row = this.dataGridView1.RowCount - 1;
            this.dataGridView1.CurrentCell = this.dataGridView1[0, row];
        }
```

***** 注意：**this.dataGridView** 的索引器的参数是：**columnIndex, rowIndex** 或是 **columnName, rowIndex**

这与习惯不同。

---

② DataGridView 设定单元格只读：

GO TO TOP

1） 使用 **ReadOnly** 属性

⇒ 如果希望，**DataGridView** 内所有单元格都不可编辑， 那么只要：

```
[VB.NET]
' 设置 DataGridView1 为只读
DataGridView1.ReadOnly = True
```

```
[C#]
// 设置 DataGridView1 为只读
DataGridView1.ReadOnly = true;
```

此时，用户的新增行操作和删除行操作也被屏蔽了。

⇒ 如果希望，**DataGridView** 内某个单元格不可编辑， 那么只要：

```
[VB.NET]
' 设置 DataGridView1 的第 2 列整列单元格为只读
DataGridView1.Columns(1).ReadOnly = True

' 设置 DataGridView1 的第 3 行整行单元格为只读
DataGridView1.Rows(2).ReadOnly = True

' 设置 DataGridView1 的[0，0]单元格为只读
DataGridView1(0, 0).ReadOnly = True
```

```
[C#]
// 设置 DataGridView1 的第 2 列整列单元格为只读
DataGridView1.Columns[1].ReadOnly = true;

// 设置 DataGridView1 的第 3 行整行单元格为只读
DataGridView1.Rows[2].ReadOnly = true;

// 设置 DataGridView1 的[0，0]单元格为只读
DataGridView1[0, 0].ReadOnly = true;
```

2） 使用 EditMode 属性

DataGridView.EditMode 属性被设置为 DataGridViewEditMode.EditProgrammatically 时，用户就不能手动编辑单元格的内容了。

但是可以通过程序，调用 DataGridView.BeginEdit 方法，使单元格进入编辑模式进行编辑。

```
[VB.NET]
DataGridView1.EditMode = DataGridViewEditMode.EditProgrammatically
```

```
[C#]
DataGridView1.EditMode = DataGridViewEditMode.EditProgrammatically;
```

3） 根据条件设定单元格的不可编辑状态

当一个一个的通过单元格坐标设定单元格 ReadOnly 属性的方法太麻烦的时候，你可以通过 CellBeginEdit 事件来取消单元格的编辑。

```
[VB.NET]
'CellBeginEdit 事件处理方法
```

```
Private Sub DataGridView1_CellBeginEdit(ByVal sender As Object, _
        ByVal e As DataGridViewCellCancelEventArgs) _
        Handles DataGridView1.CellBeginEdit
    Dim dgv As DataGridView = CType(sender, DataGridView)
    ' 是否可以进行编辑的条件检查
    If dgv.Columns(e.ColumnIndex).Name = "Column1" AndAlso _
        Not CBool(dgv("Column2", e.RowIndex).Value) Then
        ' 取消编辑
        e.Cancel = True
    End If
End Sub
```

```
[C#]
// CellBeginEdit 事件处理方法
private void DataGridView1_CellBeginEdit(object sender,
    DataGridViewCellCancelEventArgs e)
{
    DataGridView dgv = (DataGridView)sender;
    //是否可以进行编辑的条件检查
    if (dgv.Columns[e.ColumnIndex].Name == "Column1" &&
        !(bool)dgv["Column2", e.RowIndex].Value)
    {
        // 取消编辑
        e.Cancel = true;
    }
}
```

---

③ DataGridView 不显示最下面的新行：

GO TO TOP

通常 **DataGridView** 的最下面一行是用户新追加的行（行头显示 ***** ）。如果不想让用户新追加行即不想显示该新行，可以将 **DataGridView** 对象的 **AllowUserToAddRows** 属性设置为 **False**。

```
[VB.NET]
' 设置用户不能手动给 DataGridView1 添加新行
DataGridView1.AllowUserToAddRows = False
```

```
[C#]
// 设置用户不能手动给 DataGridView1 添加新行
DataGridView1.AllowUserToAddRows = false;
```

但是，可以通过程序： **DataGridViewRowCollection.Add** 为 **DataGridView** 追加新行。

补足： 如果 **DataGridView** 的 **DataSource** 绑定的是 **DataView,** 还可以通过设置 **DataView.AllowAdd** 属性为 **False** 来达到同样的效果。

---

④ DataGridView 判断新增行：

GO TO TOP

DataGridView 的 **AllowUserToAddRows** 属性为 **True** 时也就是允许用户追加新行的场合下，**DataGridView** 的最后一行就是新追加的行**(***行)。使用 **DataGridViewRow.IsNewRow** 属性可以判断哪一行是新追加的行。另外，通过 **DataGridView.NewRowIndex** 可以获取新行的行序列号。在没有新行的时候，**NewRowIndex = -1**。

```
[VB.NET]
If DataGridView1.CurrentRow.IsNewRow Then
    Console.WriteLine("当前行为新追加行。")
Else
    Console.WriteLine("当前行不是新追加行。")
End If
```

---

⑤ DataGridView 行的用户删除操作的自定义：

GO TO TOP

1） 无条件的限制行删除操作。

默认时，**DataGridView** 是允许用户进行行的删除操作的。如果设置 **DataGridView** 对象的 **AllowUserToDeleteRows** 属性为 **False** 时， 用户的行删除操作就被禁止了。

```
[VB.NET]
' 禁止 DataGridView1 的行删除操作。
DataGridView1.AllowUserToDeleteRows = False
```

```
[C#]
// 禁止 DataGridView1 的行删除操作。
DataGridView1.AllowUserToDeleteRows = false;
```

但是，通过 DataGridViewRowCollection.Remove 还是可以进行行的删除。

补足： 如果 DataGridView 绑定的是 DataView 的话，通过 DataView.AllowDelete 也可以控制行的删除。

2） 行删除时的条件判断处理。

用户在删除行的时候，将会引发 DataGridView.UserDeletingRow 事件。 在这个事件里，可以判断条件并取消删除操作。

```
[VB.NET]
' DataGridView1 的 UserDeletingRow 事件
Private Sub DataGridView1_UserDeletingRow(ByVal sender As Object, _
        ByVal e As DataGridViewRowCancelEventArgs) _
        Handles DataGridView1.UserDeletingRow
    ' 删除前的用户确认。
    If MessageBox.Show("确认要删除该行数据吗？", "删除确认", _
        MessageBoxButtons.OKCancel, MessageBoxIcon.Question) <> _
            Windows.Forms.DialogResult.OK Then
        ' 如果不是 OK，则取消。
        e.Cancel = True
    End If
End Sub
```

```
[C#]
// DataGridView1 的 UserDeletingRow 事件
private void DataGridView1_UserDeletingRow(
    object sender, DataGridViewRowCancelEventArgs e)
{
    // 删除前的用户确认。
    if (MessageBox.Show("确认要删除该行数据吗？", "删除确认",
        MessageBoxButtons.OKCancel,
        MessageBoxIcon.Question) != DialogResult.OK)
    {
```

```
        // 如果不是 OK，则取消。
        e.Cancel = true;
    }
}
```

---

⑥ DataGridView 行、列的隐藏和删除：

GO TO TOP

1） 行、列的隐藏

```
[VB.NET]
' DataGridView1 的第一列隐藏
DataGridView1.Columns(0).Visible = False
' DataGridView1 的第一行隐藏
DataGridView1.Rows(0).Visible = False
```

```
[C#]
// DataGridView1 的第一列隐藏
DataGridView1.Columns[0].Visible = false;
// DataGridView1 的第一行隐藏
DataGridView1.Rows[0].Visible = false;
```

2） 行头、列头的隐藏

```
[VB.NET]
' 列头隐藏
DataGridView1.ColumnHeadersVisible = False
' 行头隐藏
DataGridView1.RowHeadersVisible = False
```

```
[C#]
// 列头隐藏
DataGridView1.ColumnHeadersVisible = false;
// 行头隐藏
DataGridView1.RowHeadersVisible = false;
```

3） 行和列的删除

[VB.NET]

```
' 删除名为"Column1"的列
DataGridView1.Columns.Remove("Column1")
' 删除第一列
DataGridView1.Columns.RemoveAt(0)
' 删除第一行
DataGridView1.Rows.RemoveAt(0)
```

[C#]

```
' 删除名为"Column1"的列
DataGridView1.Columns.Remove("Column1");
' 删除第一列
DataGridView1.Columns.RemoveAt(0);
' 删除第一行
DataGridView1.Rows.RemoveAt(0);
```

4） 删除选中行

[VB.NET]

```
For Each r As DataGridViewRow In DataGridView1.SelectedRows
    If Not r.IsNewRow Then
        DataGridView1.Rows.Remove(r)
    End If
Next
```

[C#]

```
foreach (DataGridViewRow r in DataGridView1.SelectedRows)
{
    if (!r.IsNewRow)
    {
        DataGridView1.Rows.Remove(r);
```

```
    }
}
```

---

⑦ DataGridView 禁止列或者行的 Resize：

GO TO TOP

1） 禁止所有的列或者行的 **Resize**

[VB.NET]

```
' 禁止用户改变 DataGridView1 的所有列的列宽
DataGridView1.AllowUserToResizeColumns = False

'禁止用户改变 DataGridView1 の所有行的行高
DataGridView1.AllowUserToResizeRows = False
```

[C#]

```
// 禁止用户改变 DataGridView1 的所有列的列宽
DataGridView1.AllowUserToResizeColumns = false;

//禁止用户改变 DataGridView1 の所有行的行高
DataGridView1.AllowUserToResizeRows = false;
```

但是可以通过 **DataGridViewColumn.Width** 或者 **DataGridViewRow.Height** 属性设定列宽和行高。

2） 禁止指定行或者列的 **Resize**

[VB.NET]

```
' 禁止用户改变 DataGridView1 的第一列的列宽
DataGridView1.Columns(0).Resizable = DataGridViewTriState.False

' 禁止用户改变 DataGridView1 的第一列的行宽
DataGridView1.Rows(0).Resizable = DataGridViewTriState.False
```

[C#]

```
// 禁止用户改变 DataGridView1 的第一列的列宽
```

```
DataGridView1.Columns[0].Resizable = DataGridViewTriState.False;


// 禁止用户改变 DataGridView1 的第一列的行宽
DataGridView1.Rows[0].Resizable = DataGridViewTriState.False;
```

⇒ 关于 **NoSet**

当 **Resizable** 属性设为 **DataGridViewTriState.NotSet** 时， 实际上会默认以 **DataGridView** 的 **AllowUserToResizeColumns** 和 **AllowUserToResizeRows** 的属性值进行设定。比如： **DataGridView.AllowUserToResizeColumns = False** 且 **Resizable** 是 **NoSet** 设定时，**Resizable = False** 。

判断 **Resizable** 是否是继承设定了 **DataGridView** 的 **AllowUserToResizeColumns** 和 **AllowUserToResizeRows** 的属性值， 可以根据 **State** 属性判断。如果 **State** 属性含有 **ResizableSet**，那么说明没有继承设定。

3） 列宽和行高的最小值的设定

```
[VB.NET]
' 第一列的最小列宽设定为 100
DataGridView1.Columns(0).MinimumWidth = 100


' 第一行的最小行高设定为 50
DataGridView1.Rows(0).MinimumHeight = 50
```

```
[C#]
// 第一列的最小列宽设定为 100
DataGridView1.Columns[0].MinimumWidth = 100;


// 第一行的最小行高设定为 50
DataGridView1.Rows[0].MinimumHeight = 50;
```

4) 禁止用户改变行头的宽度以及列头的高度

```
[VB.NET]
'  禁止用户改变列头的高度
DataGridView1.ColumnHeadersHeightSizeMode = _
    DataGridViewColumnHeadersHeightSizeMode.DisableResizing


' 禁止用户改变行头的宽度
DataGridView1.RowHeadersWidthSizeMode = _
    DataGridViewRowHeadersWidthSizeMode.EnableResizing
```

```
[C#]
// 禁止用户改变列头的高度
DataGridView1.ColumnHeadersHeightSizeMode =
    DataGridViewColumnHeadersHeightSizeMode.DisableResizing;

// 禁止用户改变行头的宽度
DataGridView1.RowHeadersWidthSizeMode =
    DataGridViewRowHeadersWidthSizeMode.EnableResizing;
```

---

⑧ DataGridView 列宽和行高自动调整的设定：

GO TO TOP

1) 设定行高和列宽自动调整

```
[VB.NET]
' 设定包括 Header 和所有单元格的列宽自动调整
DataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells

' 设定包括 Header 和所有单元格的行高自动调整
DataGridView1.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells
```

```
[C#]
// 设定包括 Header 和所有单元格的列宽自动调整
DataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

// 设定包括 Header 和所有单元格的行高自动调整
DataGridView1.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;
```

AutoSizeColumnsMode 属性的设定值枚举请参照 **msdn** 的 **DataGridViewAutoSizeRowsMode** 说明。

2）指定列或行自动调整

```
[VB.NET]
' 第一列自动调整
DataGridView1.Columns(0).AutoSizeMode = _
    DataGridViewAutoSizeColumnMode.DisplayedCells
```

```
[C#]
// 第一列自动调整
DataGridView1.Columns[0].AutoSizeMode =
    DataGridViewAutoSizeColumnMode.DisplayedCells;
```

AutoSizeMode 设定为 **NotSet** 时， 默认继承的是 **DataGridView.AutoSizeColumnsMode** 属性。

3) 设定列头的高度和行头的宽度自动调整

```
[VB.NET]
' 设定列头的宽度可以自由调整
DataGridView1.ColumnHeadersHeightSizeMode = _
    DataGridViewColumnHeadersHeightSizeMode.AutoSize

' 设定行头的宽度可以自由调整
DataGridView1.RowHeadersWidthSizeMode = _
    DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders
```

```
[C#]
// 设定列头的宽度可以自由调整
DataGridView1.ColumnHeadersHeightSizeMode =
    DataGridViewColumnHeadersHeightSizeMode.AutoSize;

// 设定行头的宽度可以自由调整
DataGridView1.RowHeadersWidthSizeMode =
    DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders;
```

4） 随时自动调整

a， 临时的，让列宽自动调整，这和指定 **AutoSizeColumnsMode** 属性一样。

```
[VB.NET]
' 让 DataGridView1 的所有列宽自动调整一下。
DataGridView1.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells)

' 让 DataGridView1 的第一列的列宽自动调整一下。
DataGridView1.AutoResizeColumn(0, DataGridViewAutoSizeColumnMode.AllCells)
```

```
[C#]
// 让 DataGridView1 的所有列宽自动调整一下。
DataGridView1.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);

// 让 DataGridView1 的第一列的列宽自动调整一下。
DataGridView1.AutoResizeColumn(0, DataGridViewAutoSizeColumnMode.AllCells);
```

上面调用的 **AutoResizeColumns** 和 **AutoResizeColumn** 当指定的是 **DataGridViewAutoSizeColumnMode.AllCells** 的时候，参数可以省略。即：

DataGridView1.AutoResizeColumn(0) 和 **DataGridView1.AutoResizeColumns()**

b，临时的，让行高自动调整

```
[VB.NET]
' 让 DataGridView1 的所有行高自动调整一下。
DataGridView1.AutoResizeRows(DataGridViewAutoSizeRowsMode.AllCells)

' 让 DataGridView1 的第一行的行高自动调整一下。
DataGridView1.AutoResizeRow(0, DataGridViewAutoSizeRowMode.AllCells)
```

```
[C#]
// 让 DataGridView1 的所有行高自动调整一下。
DataGridView1.AutoResizeRows(DataGridViewAutoSizeRowsMode.AllCells);

//让 DataGridView1 的第一行的行高自动调整一下。
DataGridView1.AutoResizeRow(0, DataGridViewAutoSizeRowMode.AllCells);
```

上面调用的 **AutoResizeRows** 和 **AutoResizeRow** 当指定的是 **DataGridViewAutoSizeRowMode.AllCells** 的时候， 参数可以省略。即：**DataGridView1.AutoResizeRow（0）** 和 **DataGridView1.AutoResizeRows()**

c，临时的，让行头和列头自动调整

```
[VB.NET]
' 列头高度自动调整
DataGridView1.AutoResizeColumnHeadersHeight()

' 行头宽度自动调整
DataGridView1.AutoResizeRowHeadersWidth( _
    DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders)
```

```
[C#]
// 列头高度自动调整
DataGridView1.AutoResizeColumnHeadersHeight();

// 行头宽度自动调整
DataGridView1.AutoResizeRowHeadersWidth(
    DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders);
```

关于性能：

通过 **AutoSizeColumnsMode** 或者 **AutoSizeRowsMode** 属性所指定的单元格进行自动调整时，如果调整次数过于多那么将可能导致性能下降，尤其是在行和列数比较多的情况下。在这时用 **DisplayedCells** 代替 **AllCells** 能减少非所见的单元格的调整，从而提高性能。

---

⑨ DataGridView 冻结列或行

GO TO TOP

1） 列冻结

DataGridViewColumn.Frozen 属性为 **True** 时， 该列左侧的所有列被固定， 横向滚动时固定列不随滚动条滚动而左右移动。这对于重要列固定显示很有用。

```
[VB.NET]
' DataGridView1 的左侧 2 列固定
DataGridView1.Columns(1).Frozen = True
[C#]
// DataGridView1 的左侧 2 列固定
DataGridView1.Columns[1].Frozen = true;
```

但是，**DataGridView.AllowUserToOrderColumns = True** 时，固定列不能移动到非固定列， 反之亦然。

2） 行冻结

DataGridViewRow.Frozen 属性为 **True** 时， 该行上面的所有行被固定， 纵向滚动时固定行不随滚动条滚动而上下移动。

```
[VB.NET]
' DataGridView1 的上 3 行固定
DataGridView1.Rows(2).Frozen = True
```

```
[C#]
// DataGridView1 的上 3 行固定
DataGridView1.Rows[2].Frozen = true;
```

---

⑩ DataGridView 列顺序的调整

GO TO TOP

设定 **DataGridView** 的 **AllowUserToOrderColumns** 为 **True** 的时候， 用户可以自由调整列的顺序。

当用户改变列的顺序的时候，其本身的 **Index** 不会改变，但是 **DisplayIndex** 改变了。你也可以通过程序改变 **DisplayIndex** 来改变列的顺序。 列顺序发生改变时会引发 **ColumnDisplayIndexChanged** 事件：

```
[VB.NET]
' DataGridView1 的 ColumnDisplayIndexChanged 事件处理方法
Private Sub DataGridView1_ColumnDisplayIndexChanged(ByVal sender As Object, _
        ByVal e As DataGridViewColumnEventArgs) _
        Handles DataGridView1.ColumnDisplayIndexChanged
    Console.WriteLine("{0} 的位置改变到 {1} 。", _
        e.Column.Name, e.Column.DisplayIndex)
End Sub
```

```
[C#]
// DataGridView1 的 ColumnDisplayIndexChanged 事件处理方法
private void DataGridView1_ColumnDisplayIndexChanged(object sender,
    DataGridViewColumnEventArgs e)
{
    Console.WriteLine("{0} 的位置改变到 {1} ",
        e.Column.Name, e.Column.DisplayIndex);
}
```

⑪ DataGridView 行头列头的单元格

GO TO TOP

```
[VB.NET]
' DataGridView1 的第一列列头内容
DataGridView1.Columns(0).HeaderCell.Value = "第一列"

' DataGridView1 的第一行行头内容
DataGridView1.Rows(0).HeaderCell.Value = "第一行"

' DataGridView1 的左上头部单元内容
DataGridView1.TopLeftHeaderCell.Value = "左上"
```

```
[C#]
// 改变 DataGridView1 的第一列列头内容
DataGridView1.Columns[0].HeaderCell.Value = "第一列";

// 改变 DataGridView1 的第一行行头内容
DataGridView1.Rows[0].HeaderCell.Value = "第一行";

// 改变 DataGridView1 的左上头部单元内容
DataGridView1.TopLeftHeaderCell.Value = "左上";
```

另外你也可以通过 **HeaderText** 来改变他们的内容。

```
[VB.NET]
' 改变 DataGridView1 的第一列列头内容
DataGridView1.Columns(0).HeaderText = "第一列"
```

```
[C#]
// 改变 DataGridView1 的第一列列头内容
DataGridView1.Columns[0].HeaderText = "第一列";
```

⑫ DataGridView 剪切板的操作

GO TO TOP

DataGridView.ClipboardCopyMode 属性被设定为 **DataGridViewClipboardCopyMode.Disable** 以外的情况时，「**Ctrl + C**」按下的时候，被选择的单元格的内容会拷贝到系统剪切板内。格式有： **Text**， **UnicodeText**，**Html**， **CommaSeparatedValue**。可以直接粘贴到 **Excel** 内。

ClipboardCopyMode 还可以设定 **Header** 部分是否拷贝： **EnableAlwaysIncludeHeaderText** 拷贝 **Header** 部分、**EnableWithoutHeaderText** 则不拷贝。默认是 **EnableWithAutoHeaderText** ， **Header** 如果选择了的话，就拷贝。

1） 编程方式实现剪切板的拷贝

```
Clipboard.SetDataObject(DataGridView1.GetClipboardContent())
```

2) DataGridView 的数据粘贴

实现剪切板的拷贝比较容易，但是实现 **DataGridView** 的直接粘贴就比较难了。「**Ctrl + V**」按下进行粘贴时，**DataGridView** 没有提供方法，只能自己实现。

以下，是粘贴时简单的事例代码，将拷贝数据粘贴到以选择单元格开始的区域内。

[VB.NET]

```
' 当前单元格是否选择的判断
If DataGridView1.CurrentCell Is Nothing Then
    Return
End If
Dim insertRowIndex As Integer = DataGridView1.CurrentCell.RowIndex

' 获取剪切板的内容，并按行分割
Dim pasteText As String = Clipboard.GetText()
If String.IsNullOrEmpty(pasteText) Then
    Return
End If
pasteText = pasteText.Replace(vbCrLf, vbLf)
```

```
pasteText = pasteText.Replace(vbCr, vbLf)
pasteText.TrimEnd(New Char() {vbLf})
Dim lines As String() = pasteText.Split(vbLf)

Dim isHeader As Boolean = True
For Each line As String In lines
    ' 是否是列头
    If isHeader Then
        isHeader = False
    Else
        ' 按 Tab 分割数据
        Dim vals As String() = line.Split(ControlChars.Tab)
        ' 判断列数是否统一
        If vals.Length - 1 <> DataGridView1.ColumnCount Then
            Throw New ApplicationException("粘贴的列数不正确。")
        End If
        Dim row As DataGridViewRow = DataGridView1.Rows(insertRowIndex)
        ' 行头设定
        row.HeaderCell.Value = vals(0)
        ' 单元格内容设定
        Dim i As Integer
        For i = 0 To row.Cells.Count - 1
            row.Cells(i).Value = vals((i + 1))
        Next i

        ' DataGridView 的行索引+1
        insertRowIndex += 1
    End If
Next line
```

```
[C#]
//当前单元格是否选择的判断
if (DataGridView1.CurrentCell == null)
```

```
    return;
int insertRowIndex = DataGridView1.CurrentCell.RowIndex;

// 获取剪切板的内容，并按行分割
string pasteText = Clipboard.GetText();
if (string.IsNullOrEmpty(pasteText))
    return;
pasteText = pasteText.Replace(" ", " ");
pasteText = pasteText.Replace(' ', ' ');
pasteText.TrimEnd(new char[] { ' ' });
string[] lines = pasteText.Split(' ');

bool isHeader = true;
foreach (string line in lines)
{
    // 是否是列头
    if (isHeader)
    {
        isHeader = false;
        continue;
    }

    // 按 Tab 分割数据
    string[] vals = line.Split(' ');
    // 判断列数是否统一
    if (vals.Length - 1 != DataGridView1.ColumnCount)
        throw new ApplicationException("粘贴的列数不正确。");
    DataGridViewRow row = DataGridView1.Rows[insertRowIndex];
    // 行头设定
    row.HeaderCell.Value = vals[0];
    // 单元格内容设定
    for (int i = 0; i < row.Cells.Count; i++)
    {
```

```
        row.Cells[i].Value = vals[i + 1];
    }

    // DataGridView 的行索引+1
    insertRowIndex++;
}
```

---

⑬ DataGridView 单元格的 ToolTip 的设置

GO TO TOP

DataGridView.ShowCellToolTips = True 的情况下， 单元格的 ToolTip 可以表示出来。对于单元格窄小，无法完全显示的单元格，ToolTip 可以显示必要的信息。

1） 设定单元格的 ToolTip 内容

```
[VB.NET]
' 设定单元格的 ToolTip 内容
DataGridView1(0, 0).ToolTipText = "该单元格的内容不能修改"

' 设定列头的单元格的 ToolTip 内容
DataGridView1.Columns(0).ToolTipText = "该列只能输入数字"

' 设定行头的单元格的 ToolTip 内容
DataGridView1.Rows(0).HeaderCell.ToolTipText = "该行单元格内容不能修改"
```

```
[C#]
// 设定单元格的 ToolTip 内容
DataGridView1[0, 0].ToolTipText = "该单元格的内容不能修改";

// 设定列头的单元格的 ToolTip 内容
DataGridView1.Columns[0].ToolTipText = "该列只能输入数字";

// 设定行头的单元格的 ToolTip 内容
DataGridView1.Rows[0].HeaderCell.ToolTipText = "该行单元格内容不能修改";
```

2） CellToolTipTextNeeded 事件

在批量的单元格的 **ToolTip** 设定的时候，一个一个指定那么设定的效率比较低， 这时候可以利用 **CellToolTipTextNeeded** 事件。当单元格的 **ToolTipText** 变化的时候也会引发该事件。但是，当 **DataGridView** 的 **DataSource** 被指定且 **VirualMode=True** 的时候，该事件不会被引发。

```
[VB.NET]
' CellToolTipTextNeeded 事件处理方法
Private Sub DataGridView1_CellToolTipTextNeeded(ByVal sender As Object, _
        ByVal e As DataGridViewCellToolTipTextNeededEventArgs) _
        Handles DataGridView1.CellToolTipTextNeeded
    e.ToolTipText = e.ColumnIndex.ToString() + ", " + e.RowIndex.ToString()
End Sub
```

```
[C#]
// CellToolTipTextNeeded 事件处理方法
private void DataGridView1_CellToolTipTextNeeded(object sender,
    DataGridViewCellToolTipTextNeededEventArgs e)
{
    e.ToolTipText = e.ColumnIndex.ToString() + ", " + e.RowIndex.ToString();
}
```

---

⑭ DataGridView 的右键菜单（ContextMenuStrip）

GO TO TOP

DataGridView，DataGridViewColumn，DataGridViewRow，DataGridViewCell 有 **ContextMenuStrip** 属性。可以通过设定 **ContextMenuStrip** 对象来控制 **DataGridView** 的右键菜单的显示。**DataGridViewColumn** 的 **ContextMenuStrip** 属性设定了 除了列头以外的单元格的右键菜单。 **DataGridViewRow** 的 **ContextMenuStrip** 属性设定了除了行头以外的单元格的右键菜单。**DataGridViewCell** 的 **ContextMenuStrip** 属性设定了指定单元格的右键菜单。

```
[VB.NET]
' DataGridView 的 ContextMenuStrip 设定
DataGridView1.ContextMenuStrip = Me.ContextMenuStrip1


' 列的 ContextMenuStrip 设定
```

```
DataGridView1.Columns(0).ContextMenuStrip = Me.ContextMenuStrip2
' 列头的 ContextMenuStrip 设定
DataGridView1.Columns(0).HeaderCell.ContextMenuStrip = Me.ContextMenuStrip2

' 行的 ContextMenuStrip 设定
DataGridView1.Rows(0).ContextMenuStrip = Me.ContextMenuStrip3

' 单元格的 ContextMenuStrip 设定
DataGridView1(0, 0).ContextMenuStrip = Me.ContextMenuStrip4
[C#]
// DataGridView 的 ContextMenuStrip 设定
DataGridView1.ContextMenuStrip = this.ContextMenuStrip1;

// 列的 ContextMenuStrip 设定
DataGridView1.Columns[0].ContextMenuStrip = this.ContextMenuStrip2;
// 列头的 ContextMenuStrip 设定
DataGridView1.Columns[0].HeaderCell.ContextMenuStrip = this.ContextMenuStrip2;

// 行的 ContextMenuStrip 设定
DataGridView1.Rows[0].ContextMenuStrip = this.ContextMenuStrip3;

// 单元格的 ContextMenuStrip 设定
DataGridView1[0, 0].ContextMenuStrip = this.ContextMenuStrip4;
```

对于单元格上的右键菜单的设定，优先顺序是： **Cell > Row > Column > DataGridView**

⇒ **CellContextMenuStripNeeded**、**RowContextMenuStripNeeded** 事件

利用 **CellContextMenuStripNeeded** 事件可以设定单元格的右键菜单，尤其但需要右键菜单根据单元格值的变化而变化的时候。比起使用循环遍历，使用该事件来设定右键菜单的效率更高。但是，在 **DataGridView** 使用了 **DataSource** 绑定而且是 **VirtualMode** 的时候，该事件将不被引发。

```
[VB.NET]
'  CellContextMenuStripNeeded 事件处理方法
Private Sub DataGridView1_CellContextMenuStripNeeded( _
    ByVal sender As Object, _
    ByVal e As DataGridViewCellContextMenuStripNeededEventArgs) _
    Handles DataGridView1.CellContextMenuStripNeeded
  Dim dgv As DataGridView = CType(sender, DataGridView)
```

```
    If e.RowIndex < 0 Then
        ' 列头的 ContextMenuStrip 设定
        e.ContextMenuStrip = Me.ContextMenuStrip1
    ElseIf e.ColumnIndex < 0 Then
        ' 行头的 ContextMenuStrip 设定
        e.ContextMenuStrip = Me.ContextMenuStrip2
    ElseIf TypeOf (dgv(e.ColumnIndex, e.RowIndex).Value) Is Integer Then
        ' 如果单元格值是整数时
        e.ContextMenuStrip = Me.ContextMenuStrip3
    End If
End Sub
```

```
[C#]
//  CellContextMenuStripNeeded 事件处理方法
private void DataGridView1_CellContextMenuStripNeeded(object sender,
    DataGridViewCellContextMenuStripNeededEventArgs e)
{
    DataGridView dgv = (DataGridView)sender;
    if (e.RowIndex < 0)
    {
        //  列头的 ContextMenuStrip 设定
        e.ContextMenuStrip = this.ContextMenuStrip1;
    }
    else if (e.ColumnIndex < 0)
    {
        //  行头的 ContextMenuStrip 设定
        e.ContextMenuStrip = this.ContextMenuStrip2;
    }
    else if (dgv[e.ColumnIndex, e.RowIndex].Value is int)
    {
        //  如果单元格值是整数时
        e.ContextMenuStrip = this.ContextMenuStrip3;
```

```
    }
}
```

同样，可以通过 **RowContextMenuStripNeeded** 事件来设定行的右键菜单。

```
[VB.NET]
'  RowContextMenuStripNeeded 事件处理方法
Private Sub DataGridView1_RowContextMenuStripNeeded( _
        ByVal sender As Object, _
        ByVal e As DataGridViewRowContextMenuStripNeededEventArgs) _
        Handles DataGridView1.RowContextMenuStripNeeded
    Dim dgv As DataGridView = CType(sender, DataGridView)
    '  当"Column1"列是 Bool 型且为 True 时、设定其的 ContextMenuStrip
    Dim boolVal As Object = dgv("Column1", e.RowIndex).Value
    Console.WriteLine(boolVal)
    If TypeOf boolVal Is Boolean AndAlso CBool(boolVal) Then
        e.ContextMenuStrip = Me.ContextMenuStrip1
    End If
End Sub
[C#]
//  RowContextMenuStripNeeded 事件处理方法
private void DataGridView1_RowContextMenuStripNeeded(object sender,
    DataGridViewRowContextMenuStripNeededEventArgs e)
{
    DataGridView dgv = (DataGridView)sender;
    //  当"Column1"列是 Bool 型且为 True 时、设定其的 ContextMenuStrip
    object boolVal = dgv["Column1", e.RowIndex].Value;
    Console.WriteLine(boolVal);
    if (boolVal is bool && (bool)boolVal)
    {
        e.ContextMenuStrip = this.ContextMenuStrip1;
    }
}
```

CellContextMenuStripNeeded 事件处理方法的参数中、「**e.ColumnIndex=-1**」表示行头、「**e.RowIndex=-1**」表示列头。**RowContextMenuStripNeeded** 则不存在「**e.RowIndex=-1**」的情况。

---

⑮ DataGridView 的单元格的边框、 网格线样式的设定

GO TO TOP

1) DataGridView 的边框线样式的设定

DataGridView 的边框线的样式是通过 **DataGridView.BorderStyle** 属性来设定的。 **BorderStyle** 属性设定值是一个 **BorderStyle** 枚举： **FixedSingle**（单线，默认）、**Fixed3D**、**None**。

2) 单元格的边框线样式的设定

单元格的边框线的样式是通过 **DataGridView.CellBorderStyle** 属性来设定的。 **CellBorderStyle** 属性设定值是 **DataGridViewCellBorderStyle** 枚举。（详细参见 **MSDN**）

另外，通过 **DataGridView.ColumnHeadersBorderStyle** 和 **RowHeadersBorderStyle** 属性可以修改 **DataGridView** 的头部的单元格边框线样式。 属性设定值是 **DataGridViewHeaderBorderStyle** 枚举。（详细参见 **MSDN**）

3） 单元格的边框颜色的设定

单元格的边框线的颜色可以通过 **DataGridView.GridColor** 属性来设定的。默认是 **ControlDarkDark** 。但是只有在 **CellBorderStyle** 被设定为 **Single**、**SingleHorizontal**、**SingleVertical** 的条件下才能改变其边框线的颜色。同样，**ColumnHeadersBorderStyle** 以及 **RowHeadersBorderStyle** 只有在被设定为 **Single** 时，才能改变颜色。

4） 单元格的上下左右的边框线式样的单独设定

CellBorderStyle 只能设定单元格全部边框线的式样。要单独改变单元格某一边边框式样的话，需要用到 **DataGridView.AdvancedCellBorderStyle** 属性。如示例：

```
[VB.NET]
' 单元格的上边和左边线设为二重线
' 单元格的下边和右边线设为单重线
DataGridView1.AdvancedCellBorderStyle.Top = _
    DataGridViewAdvancedCellBorderStyle.InsetDouble
DataGridView1.AdvancedCellBorderStyle.Right = _
    DataGridViewAdvancedCellBorderStyle.Inset
DataGridView1.AdvancedCellBorderStyle.Bottom = _
    DataGridViewAdvancedCellBorderStyle.Inset
```

```
DataGridView1.AdvancedCellBorderStyle.Left = _
    DataGridViewAdvancedCellBorderStyle.InsetDouble
```

同样，设定行头单元格的属性是： **AdvancedRowHeadersBorderStyle**， 设定列头单元格属性是：**AdvancedColumnHeadersBorderStyle**。

---

⑯ DataGridView 单元格表示值的自定义

GO TO TOP

通过 **CellFormatting** 事件，可以自定义单元格的表示值。（比如：值为 **Error** 的时候，单元格被设定为红色）

下面的示例：将"**Colmn1**"列的值改为大写。

```
[VB.NET]
'CellFormatting 事件处理方法
Private Sub DataGridView1_CellFormatting(ByVal sender As Object, _
        ByVal e As DataGridViewCellFormattingEventArgs) _
        Handles DataGridView1.CellFormatting
    Dim dgv As DataGridView = CType(sender, DataGridView)

    ' 如果单元格是"Column1"列的单元格
    If dgv.Columns(e.ColumnIndex).Name = "Column1" AndAlso _
            TypeOf e.Value Is String Then
        ' 将单元格值改为大写
        Dim str As String = e.Value.ToString()
        e.Value = str.ToUpper()
        '  应用该 Format，Format 完毕。
        e.FormattingApplied = True
    End If
End Sub
```

```
[C#]
//CellFormatting 事件处理方法
private void DataGridView1_CellFormatting(object sender,
    DataGridViewCellFormattingEventArgs e)
{
```

```
    DataGridView dgv = (DataGridView)sender;

    // 如果单元格是"Column1"列的单元格
    if (dgv.Columns[e.ColumnIndex].Name == "Column1" && e.Value is string)
    {
        // 将单元格值改为大写
        string str = e.Value.ToString();
        e.Value = str.ToUpper();
        // 应用该 Format，Format 完毕。
        e.FormattingApplied = true;
    }
}
```

CellFormatting 事件的 DataGridViewCellFormattingEventArgs 对象的 Value 属性一开始保存着未被格式化的值。当 Value 属性被设定表示用的文本之后，把 FormattingApplied 属性做为 True，告知 DataGridView 文本已经格式化完毕。如果不这样做的话，DataGridView 会根据已经设定的 Format，NullValue，DataSourceNullValue，FormatProvider 属性会将 Value 属性会被重新格式化一遍。

---

⑰ DataGridView 用户输入时，单元格输入值的设定

GO TO TOP

通过 DataGridView.CellParsing 事件可以设定用户输入的值。下面的示例：当输入英文文本内容的时候，立即被改变为大写。

```
[VB.NET]
'CellParsing 事件处理方法
Private Sub DataGridView1_CellParsing(ByVal sender As Object, _
        ByVal e As DataGridViewCellParsingEventArgs) _
        Handles DataGridView1.CellParsing
    Dim dgv As DataGridView = CType(sender, DataGridView)

    ' 单元格列为"Column1"时
    If dgv.Columns(e.ColumnIndex).Name = "Column1" AndAlso _
            e.DesiredType Is GetType(String) Then
        ' 将单元格值设为大写
        e.Value = e.Value.ToString().ToUpper()
```

```
        ' 解析完毕
        e.ParsingApplied = True
    End If
End Sub
```

```
[C#]
//CellParsing 事件处理方法
private void DataGridView1_CellParsing(object sender,
    DataGridViewCellParsingEventArgs e)
{
    DataGridView dgv = (DataGridView)sender;

    //单元格列为"Column1"时
    if (dgv.Columns[e.ColumnIndex].Name == "Column1" &&
        e.DesiredType == typeof(string))
    {
        //将单元格值设为大写
        e.Value = e.Value.ToString().ToUpper();
        //解析完毕
        e.ParsingApplied = true;
    }
}
```

---

⑱ DataGridView 新加行的默认值的设定

GO TO TOP

需要指定新加行的默认值的时候，可以在 **DataGridView.DefaultValuesNeeded** 事件里处理。在该事件中处理除了可以设定默认值以外，还可以指定某些特定的单元格的 **ReadOnly** 属性等。

```
[VB.NET]
' DefaultValuesNeeded 事件处理方法
Private Sub DataGridView1_DefaultValuesNeeded(ByVal sender As Object, _
        ByVal e As DataGridViewRowEventArgs) _
```

```
        Handles DataGridView1.DefaultValuesNeeded
    ' 设定单元格默认值
    e.Row.Cells("Column1").Value = 0
    e.Row.Cells("Column2").Value = "-"
End Sub
```

```
[C#]
// DefaultValuesNeeded 事件处理方法
private void DataGridView1_DefaultValuesNeeded(object sender,
    DataGridViewRowEventArgs e)
{
    // 设定单元格的默认值
    e.Row.Cells["Column1"].Value = 0;
    e.Row.Cells["Column2"].Value = "-";
}
```

DataGridView 编程 36 计（二）

2008-02-26 09:47:39

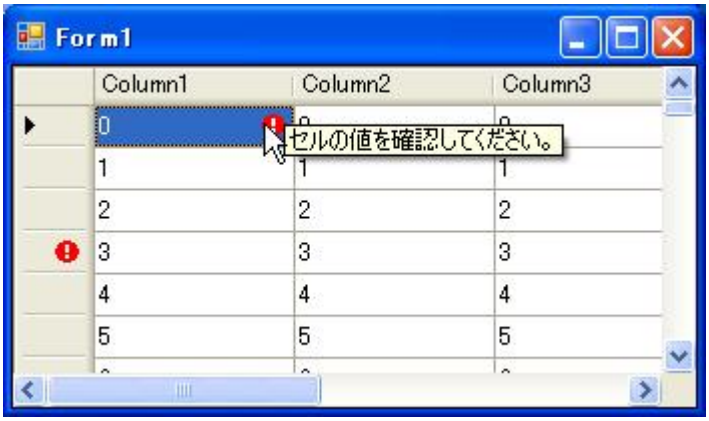
标签：编程 DataGridView [推送到技术圈]

目录：

① DataGridView Error 图标表示的设定：

GO TO TOP

为了提醒用户注意，DataGridView 可以使用 Error 图标来突出显示。如下图：



Error 图标可以在单元格和行头内表示，但不能在列头上显示。

1） ErrorText 属性

当设定单元格/行的 ErrorText 属性的内容后，单元格/行的 Error 图标就会被表示出来。另外，只有在 DataGridView.ShowCellErrors = True 时，Error 图标才能显示。（默认即时 True）

```
[VB.NET]
' 设定 (0, 0) 的单元格表示 Error 图标
DataGridView1(0, 0).ErrorText = "请确认单元格的值。"

' 设定第 4 行(Index=3)的行头显示 Error 图标
DataGridView1.Rows(3).ErrorText = "不能输入负值。"
```

2） CellErrorTextNeeded、RowErrorTextNeeded 事件

即时输入时的 Error 图标的表示，可以使用 CellErrorTextNeeded 事件

同时，在大量的数据处理时，需要进行多处的内容检查并显示 Error 图标的应用中。遍历单元格设定 ErrorText 的方法是效率低下的，应该使用 CellErrorTextNeeded 事件。行的 Error 图标的设定则应该用 RowErrorTextNeeded 事件。

但是，需要注意的是当 DataSource 属性设定了 VirtualMode=True 时，上述事件则不会被引发。

```
[VB.NET]
'CellErrorTextNeeded 事件处理方法
Private Sub DataGridView1_CellErrorTextNeeded(ByVal sender As Object, _
```

```
        ByVal e As DataGridViewCellErrorTextNeededEventArgs) _
        Handles DataGridView1.CellErrorTextNeeded
    Dim dgv As DataGridView = CType(sender, DataGridView)
    ' 单元格值为负整数时，Error 图标被表示。
    Dim cellVal As Object = dgv(e.ColumnIndex, e.RowIndex).Value
    If TypeOf cellVal Is Integer AndAlso CInt(cellVal) < 0 Then
        e.ErrorText = "不能输入负整数。"
    End If
End Sub

'RowErrorTextNeeded 事件处理方法
Private Sub DataGridView1_RowErrorTextNeeded(ByVal sender As Object, _
        ByVal e As DataGridViewRowErrorTextNeededEventArgs) _
        Handles DataGridView1.RowErrorTextNeeded
    Dim dgv As DataGridView = CType(sender, DataGridView)
    If dgv("Column1", e.RowIndex).Value Is DBNull.Value AndAlso _
        dgv("Column2", e.RowIndex).Value Is DBNull.Value Then
        e.ErrorText = _
            "Column1 和 Column2 必须输入一个值。"
    End If
End Sub
```

```
[C#]
// CellErrorTextNeeded 事件处理方法
private void DataGridView1_CellErrorTextNeeded(object sender,
    DataGridViewCellErrorTextNeededEventArgs e)
{
    DataGridView dgv = (DataGridView)sender;
    // 单元格值为负整数时，Error 图标被表示。
    object cellVal = dgv[e.ColumnIndex, e.RowIndex].Value;
    if (cellVal is int && ((int)cellVal) < 0)
    {
        e.ErrorText = "不能输入负整数。";
    }
}

// RowErrorTextNeeded 事件处理方法
private void DataGridView1_RowErrorTextNeeded(object sender,
    DataGridViewRowErrorTextNeededEventArgs e)
{
    DataGridView dgv = (DataGridView)sender;
    if (dgv["Column1", e.RowIndex].Value == DBNull.Value &&
        dgv["Column2", e.RowIndex].Value == DBNull.Value)
    {
        e.ErrorText =
            "Column1 和 Column2 必须输入一个值。";
    }
}
```