第 1 章　常用控件使用详解

控件是 Visual C#编程的基础，也是重要的可视化编程工具。在 Windows 对话框中经常会遇到按钮、列表框这样的控件，Microsoft 将这些流行控件作为 Windows 95/98/2000/XP 操作系统的一部分，使 Windows 程序员不必再白手起家来创建这些控件。

Control 类是所有窗体控件(包括窗体)的基类，因此它的属性、方法和事件也适用于所有窗体控件(除非被重载)。需要注意的是，控件类中定义的属性和属性窗口中显示的属性名称并不一定完全相同。例如，控件类中定义的 TextAlign 属性，在属性窗口中将显示为 Text Align。但这两者之间的关系还是显而易见的。因此本章虽然主要介绍控件类中的属性定义形式，但读者仍然能容易地在属性窗口中找到对应项。

本章要点：

- ❖ 控件基础功能支持
- ❖ 常用控件的属性、方法和事件
- ❖ 常用控件的使用方法
- ❖ 窗体控件集合的管理

1.1　控件基础功能支持

Control 类定义了控件(具有可视化外观的组件)的基类。本类实现了类需要向用户显示的基本功能。例如，处理键盘和鼠标输入。管理消息发送和安全；定义控件的位置和大小(虽然没有实现绘制)；提供 Windows 句柄(hWnd)等。

Control 类的构造的函数定义如下：

```
public Control();
public Control(string text);
public Control(Control parent,string text);
public Control(string text,int left,int top,int width,int height);
public Control (Control parent,string text,int left,int top,int width,int
height);
```

参数：

text——控件文本。

parent——本控件的父控件。

left——在控件容器中，控件左上角的 x 坐标(以像素为单位)。

top——在控件容器中，控件左上角的 y 坐标(以像素为单位)。

width——控件的宽度(以像素为单位)。

height——控件的高度(以像素为单位)。

Control 类是 Windows 窗体应用程序中使用的所有控件的基类。由于该类一般不用于创建实例，因此构造函数通常由派生类调用。

1.1.1 基础控件属性

Control 类中定义的基础控件属性如下(需要注意的是，这些属性与其属性窗口中的名称并不完全一致，但依然存在明显的对应关系，例如 AllowDrop 属性对应于 Allow Drop)：

- AllowDrop 属性

使用 AllowDrop 属性，以确定控件是否可接受用户拖动。如果本控件允许拖动，则本属性的值为 true，否则为 false(默认值)。对于 RichTextBox 控件，本属性总是为 false。因为，该控件不允许执行拖动操作。

- Anchor 属性

使用 Anchor 属性，以获取或设置控件的哪个边缘停靠在容器边缘。本属性的值为 AnchorStyles 枚举值之一。

- BackColor 属性

使用 BackColor 属性，以获取或设置本控件的背景颜色。本属性为环境属性，因此总是返回非空值。

- BackgroundImage 属性

使用 BackgroundImage 属性，以获取或设置控件中显示的背景图片。

- BindingContext 属性

使用 BindingContext 属性，以获取或设置对象的 BindingContext。控件的 BindingContext 对象用于为其中包含的所有数据绑定控件返回一个 BindingManagerBase 对象。BindingManagerBase 对象使绑定到同一数据源的所有控件保持同步。例如，通过设置该对象的 Position 属性，可以指定所有数据绑定控件指向的底层列表项。

- Bottom 属性

使用 Bottom 属性，以获取本控件下边缘与容器客户区上边缘之间的距离。本属性的值等于 Top 属性与 Height 属性值之和。

- Bounds 属性

使用 Bounds 属性，以获取或设置本控件的边界矩形。

- CanFocus 属性

使用 CanFocus 属性，以确定控件是否能接收焦点。如果控件可以接收焦点，则本属性的值为 true，否则为 false。要使控件能接收输入焦点，控件必须具有句柄，并且其 Visible 和 Enabled 属性必须为 true。

- CanSelect 属性

使用 CanSelect 属性，以确定本控件是否可被选择。如果控件可被选择，则本属性的值为 true，否则为 false。如果控件的 ControlStyles. Selectable 被设置为 true，并且它的容器

控件和所有父控件都可见并且被启用，则本属性将返回 true。

下面给出 CanSelect 属性为 false 的 Windows 窗体控件：Panel、GroupBox、PictureBox、ProgressBar、Splitter、Label、LinkLabel(当控件中不存在链接时)。需要注意的是，派生自这些控件的控件也不能被选择。

- Capture 属性

使用 Capture 属性，以确定控件是否被鼠标捕获。如果控件被鼠标捕获，则本属性的值为 true，否则为 false(默认值)。当控件被鼠标捕获后，它将接收鼠标输入，而无论光标是否处于它的边界内。

- CausesValidation 属性

使用 CausesValidation 属性，以确定进入控件是否会导致所有需要校验的控件都被校验。如果进入控件会导致所有需要校验的控件都被校验，则本属性的值为 true，否则为 false(默认值)。

对于“帮助”按钮等控件，一般都将本属性设置为 false。

- ClientRectangle 属性

使用 ClientRectangle 属性，以获取代表控件客户区的矩形。由于客户坐标是相对于控件客户区左上角的，因此由本属性返回的矩形左上角坐标为(0，0)。在使用. NET 框架工具绘制控件表面时，需要使用本属性以确定绘制区域。

- ClientSize 属性

使用 ClientSize 属性，以获取控件客户区的宽度和高度。返回的 Size 对象的 Width 属性代表客户区的宽度，Height 属性代表客户区的高度。

- CompanyName 属性

使用 CompanyName 属性，以获取包含控件的应用程序的创建者(公司)名。

- ContainsFocus 属性

使用 ContainsFocus 属性，以确定控件或其子控件当前是否具有焦点。如果控件或其子控件当前具有焦点，则本属性的值为 true，否则为 false。要确定控件本身是否具有焦点，则应使用 Focusd 属性。要给予控件输入焦点，则应使用 Focus 或 Select 方法。

- ContextMenu 属性

使用 ContextMenu 属性，以获取或设置与本控件相关的快捷菜单。

- Controls 属性

使用 Controls 属性，以获取或设置控件中包含的控件集合。

- Created 属性

使用 Created 属性，以确定控件是否已被创建。如果控件已被创建，则本属性的值为 true，否则为 false。

- CreateParams 属性

使用 CreateParams 属性，以获取创建本控件时使用的参数。本属性的值为控件句柄被创建时包含必需参数的 CreateParams 对象。当在派生类中重载 CreateParams 时，应确保通过创建基类的 CreateParams 实例来扩展 CreateParams 对象——添加或修改属性值。

- Cursor 属性

使用 Cursor 属性，以获取或设置当用户将鼠标移动到控件上方时光标的形状。

- DataBindings 属性

使用 DataBindings 属性，以获取控件的数据绑定。本属性的值为包含控件的 Binding 对象的 ControlBindingsCollection。通过将 Binding 对象加入到本集合中，可以将控件的任何属性绑定到对象属性上。

- DefaultBackColor 属性

使用 DefaultBackColor 属性，以获取控件的默认背景色。本属性的默认值为 SystemColors. Control。但在派生类中可指定其他默认颜色。

- DefaultFont 属性

使用 DefaultFont 属性，以获取控件的默认字体。本属性的值为控件的默认背景字体，其默认值为 8 点字体。如果系统未安装指定字体，则控件试图使用如下字体之一：CordiaUPC、MS PGothic、SimSun、Gulim、MingLiU 和 Arial。

- DefaultForeColor 属性

使用 DefaultForeColor 属性，以获取控件的默认前景色。本属性的默认值为 SystemColors. ControlText。

- DefaultImeMode 属性

使用 DefaultImeMode 属性，以获取本控件支持的默认输入法编辑器(IME，Input Method Editor)。在 Control 类中，本属性将总是返回 ImeMode. Inherit。这表示指定的 IME 模式继承自父控件。

- DefaultSize 属性

使用 DefaultSize 属性，以获取控件的默认尺寸。

- DisplayRectangle 属性

使用 DisplayRectangle 属性，以获取代表控件显示区域的矩形。对于基础控件类，本属性等于客户矩形。然而，派生控件可能会修改客户区域，使其与显示区域具有不同的值。

- Disposing 属性

使用 Disposing 属性，以确定控件是否处于清除进程中。如果控件正处于清除进程，则本属性的值为 true，否则为 false。在控件被清除后，它就不能再作为有效的 Windows 控件引用了。虽然控件实例已被清除，它还将继续存留于内存中，直到冗码收集器将其删除。

- Dock 属性

使用 Dock 属性，以获取或设置控件停靠的父容器边界。本属性的值为 DockStyle 枚举值之一，其默认值为 None。控件将被停靠到父容器中的指定边界。例如，如果将本属性设置为 DockStyle. Left，则控件的左边界将被停靠到其父控件的左边界。此外，控件的停靠边界还将重置以匹配容器控件。

- Enabled 属性

使用 Enabled 属性，以确定控件是否被启用。如果控件当前被启用，则本属性的值为 true，否则为 false。

- Focused 属性

使用 Focused 属性，以确定控件是否具有输入焦点。如果控件具有输入焦点，则本属性的值为 true，否则为 false。

- Font 属性

使用 Font 属性，以获取或设置控件的当前字体。

- FontHeight 属性

使用 FontHeight 属性，以获取或设置控件 Font 属性的高度。本属性的值为控件字体的像素高度。

- ForeColor 属性

使用 ForeColor 属性，以获取或设置控件的前景色。本属性的默认值为 DefaultForeColor。

- Handle 属性

使用 Handle 属性，以获取绑定到本控件的窗口句柄。

- HasChildren 属性

使用 HasChildren 属性，以确定控件中是否包含子控件。如果控件中包含子控件，则本属性的值为 true，否则为 false。如果控件集合的 Count 属性大于 0，则 HasChildren 属性将返回 true。

- Height 属性

使用 Height 属性，以获取本控件的高度。

- ImeMode 属性

使用 ImeMode 属性，以获取或设置本控件支持的输入方法编辑器。

- InvokeRequired 属性

使用 InvokeRequired 属性，以确定调用本控件的方法时，是否需要激活调用者。如果控件句柄与方法调用不在同一线程中，则标志应通过激活方法来调用其他方法，此时本属性的值为 true。

Windows 窗体的控件被绑定到指定线程(非线程安全)。因此，如果从其他线程调用控件方法时，必须使用控件的激活方法来将调用配置到正确的线程上。本属性可用于确定是否需要调用激活方法，这在不知道控件属于哪个线程时尤其有用。控件有 4 个方法可由任何线程调用：Invoke、BeginInvoke、EndInvoke 和 CreateGraphics。对于所有其他方法调用，都应使用激活方法。

- IsDisposed 属性

使用 IsDisposed 属性，以确定控件是否已被清除。如果控件已被清除，则本属性的值为 true，否则为 false。

在控件被清除后，它就不能再作为有效的 Windows 控件引用了。虽然控件实例已被清除，它还将继续存留于内存中，直到冗码收集器将其删除。

- IsHandleCreated 属性

使用 IsHandleCreated 属性，以确定控件是否有与之相关的句柄。如果控件有与之相关的句柄，则本属性的值为 true，否则为 false。

- Left 属性

使用 Left 属性，以获取本控件左边界的 x 坐标(以像素为单位)。

- Location 属性

使用 Location 属性，以获取本控件的左上角坐标(相对于其容器的左上角)。

- ModifierKeys 属性

使用 ModifierKeys 属性，以确定 Shift、Ctrl 和 Alt 修饰符的当前状态。

- MouseButtons 属性

使用 MouseButtons 属性，以确定鼠标键的当前状态。

- MousePosition 属性

使用 MousePosition 属性，以获取鼠标指针的当前位置(屏幕坐标)。

- Name 属性

使用 Name 属性，以获取或设置控件的名称。本属性的默认值为空字符串。在运行时，可使用 Name 属性引用对象。由于 Name 属性为 String 类型，因此它能用于分支逻辑语句中。

- Parent 属性

使用 Parent 属性，以获取或设置本控件的父对象。

- ProductName 属性

使用 ProductName 属性，以获取包含控件的应用程序的产品名。

- ProductVersion 属性

使用 ProductVersion 属性，以获取包含控件的应用程序的版本。

- RecreatingHandle 属性

使用 RecreatingHandle 属性，以确定控件是否正在重新创建它的句柄。如果控件正在重新创建它的句柄，则本属性的值为 true，否则为 false。

- Region 属性

使用 Region 属性，以获取或设置与本控件相关的区域(椭圆或多边形)。本属性定义了控件的轮廓和边界。

- ResizeRedraw 属性

使用 ResizeRedraw 属性，以确定控件是否应在重设尺寸时被重新绘制。如果控件在重设尺寸时应被重新绘制，则本属性的值为 true，否则为 false。

- Right 属性

使用 Right 属性，以获取控件右边界与其容器左边界之间的距离。

- RightToLeft 属性

使用 RightToLeft 属性，以获取或设置控件元素的对齐方式是否能被颠倒，从而支持使用从右到左字体的地区。

本属性由国际化应用程序使用，以适应于从右到左的顺序书写和阅读的语言。当本属性被设置为 RightToLeft. Yes 时，包含文本的控件元素将被反向以从右向左显示，例如在使用希伯来和阿拉伯字体时。

当 RightToLeft 属性被设置为 RightToLeft. Yes 时，控件元素将可能受到如下影响：垂

直滚动条将显示在窗体、多行文本框等控件的左侧；水平滚动条中的滑块出现在最右侧；复选框将显示在与 CheckAlign 属性设置相反的一侧；窗体标题栏中的文本将右对齐显示。不过图标和控件框将依然保持它们的原位；列表框、组合框和下拉控件中的条目将右对齐；NumericUpDown 和 DomainUpDown 控件的上下按钮将出现在控件的左侧；菜单和菜单项将右对齐显示；工具栏按钮不会受到 RightToLeft 属性的影响；AxHost 支持从右到左的对齐方式；然而对 ActiveX 控件的影响则取决于开发者实现的程度。

需要注意的是，当 RightToLeft 属性被设置为 RightToLeft. Yes 时，控件元素的水平对齐方式将反向，但对应的属性值却不会改变。例如，在 TextAlign 属性为 HorizontalAlignment. Left 的 TextBox 控件中，文本将被右对齐显示，但是 TextAlign 属性的值将依然为 HorizontalAlignment. Left。但是，如果 RightToLeft 属性被设置为 RightToLeft. Yes，而 TextAlign 属性被设置为 HorizontalAlignment. Right，那么文本将左对齐显示。

- ShowFocusCues 属性

使用 ShowFocusCues 属性，以确定用户界面是否处于显示或隐藏焦点矩形的状态。如果用户界面处于显示或隐藏焦点矩形的状态，则本属性的值为 true，否则为 false。

- ShowKeyboardCues 属性

使用 ShowKeyboardCues 属性，以确定用户界面是否处于显示或隐藏键盘加速键的状态。如果用户界面处于显示或隐藏键盘加速键的状态，则本属性的值为 true，否则为 false。

- Size 属性

使用 Size 尺寸，以获取控件的宽度和高度。

- TabIndex 属性

使用 TabIndex 属性，以获取本控件在容器中的 Tab 键顺序。Tab 键索引可以为任意大于或等于 0 的整数。如果容器中存在多个 Tab 键索引相同的控件，那么将使用 z 顺序确定选择控件的顺序。

- TabStop 属性

使用 TabStop 属性，以确定是否能通过 Tab 键使本控件获得焦点。如果使用 Tab 键能使 Tab 键获得焦点，则本属性的值为 true，否则为 false。

- Tag 属性

使用 Tag 属性，以获取或设置包含与控件相关的数据的对象。本属性的默认值为空引用。可将任意派生自 Object 的对象赋予本属性。如果通过 Windows 窗体设计器设置本属性，则只能为其赋予文本。

- Text 属性

使用 Text 属性，以获取或设置与本控件相关的文本。

- Top 属性

使用 Top 属性，以获取控件的顶坐标。

- TopLevelControl 属性

使用 TopLevelControl 属性，以获取包含当前控件的顶层控件。

- Visible 属性

使用 Visible 属性，以确定控件是否可见。如果控件可见，则本属性的值为 true，否则为 false。

- Width 属性

使用 Width 属性，以获取本控件的宽度。

1.1.2 基础控件方法

Control 类中定义的基础控件方法如下：

- BeginInvoke 方法

调用 BeginInvoke 方法，以执行控件基础句柄所在线程中的指定 Delegate，其定义如下：

```
public IAsyncResult BeginInvoke(Delegate method);
public IAsyncResult BeginInvoke(Delegate method,object[] args);
```

参数：

method——Delegate 方法。

args——代表指定方法参数的 Object 数组。如果不需要使用参数，则应将本参数设置为空。

返回值：

代表 BeginInvoke 操作结果的 IAsyncResult 接口。

Delegate 将被异步调用，而本方法将立即返回。本方法可由任何线程调用，这包括拥有控件句柄的线程。如果控件句柄不存在，则将在控件层次中向上寻找，直到找到一个有句柄的控件或窗体。如果找不到合适的句柄，则 BeginInvoke 将抛出异常。Delegate 方法中的异常被看作是未被捕捉的，并将会发送给应用程序的异常处理函数。

Control 类中有 4 个可从任何线程调用的方法：Invoke、BeginInvoke、EndInvoke 和 CreateGraphics。对于所有其他方法调用，都应使用某个激活方法以将调用配置到控件线程中。

- BringToFront 方法

调用 BringToFront 方法，以将本控件置于 z 顺序的前景中，其定义如下：

```
public void BringToFront();
```

- Contains 方法

调用 Contains 方法，以确定指定控件是否为本控件的子项，其定义如下：

```
public bool Contains(Control ctl);
```

参数：

ctl——将检测的控件。

返回值：

如果指定控件为本控件的子项，则返回 true，否则返回 false。

- CreateControl 方法

调用 CreateControl 方法，以强制创建控件(这包括创建句柄及其子控件)，其定义如下：

```
public void CreateControl();
```

- CreateControlsInstance 方法

调用 CreateControlsInstance 方法，以创建本控件的新 Controls 集合，其定义如下：

```
protected virtual ControlCollection CreateControlsInstance();
```

返回值：

赋予本控件的新 ControlCollection 实例。

- CreateGraphics 方法

调用 CreateGraphics 方法，以创建控件的 Graphics 对象，其定义如下：

```
public Graphics CreateGraphics();
public Graphics CreateGraphics(IntPtr dc);
```

参数：

dc——将为其创建 Graphics 的对象的设备上下文句柄。

返回值：

控件的 Graphics 对象。

控件的画刷、字体、前景色和背景色成为 Graphics 对象的默认值。当不再需要返回的 Graphics 对象时，必须用 Dispose 方法清除。Graphics 对象只在当前 Windows 消息中有效。

- CreateHandle 方法

调用 CreateHandle 方法，以创建本控件的句柄，其定义如下：

```
protected virtual void CreateHandle();
```

如果对象处于清除状态，则方法将抛出 ObjectDisposedException 异常。

当在派生类中重载时，应确保调用基类的 CreateHandle 方法。

- DefWndProc 方法

调用 DefWndProc 方法，以向默认的窗口进程发送消息，其定义如下：

```
protected virtual void DefWndProc(ref Message m);
```

参数：

m——Win32 消息。

- DestroyHandle 方法

调用 DestroyHandle 方法，以销毁与本控件相关的句柄，其定义如下：

```
protected virtual void DestroyHandle();
```

当在派生类中重载时，应确保调用基类的 DestroyHandle 方法。

- Dispose 方法

调用 Dispose 方法，以释放由控件使用的非受控资源，并选择性地释放受控资源。该

方法的定义如下：

```
protected override void Dispose(bool disposing);
```

参数：

disposing——如果为 true，则将同时释放受控和非受控资源；否则，将只释放非受控资源。

本方法由公共型 Dispose 和 Finalize 方法调用。Dispose 方法激活了保护型 Dispose (Boolean)方法，并将 disposing 参数设置为 true。Finalize 调用 Dispose 方法时，将 disposing 参数设置为 false。

当 disposing 参数为 true 时，本方法将释放由本 Control 引用的所有受控对象的所有资源。本方法将调用每个被引用对象的所有 Dispose 方法。

Dispose 方法可由其他对象多次调用。当重载 Dispose(Boolean)方法时，不应引用已被以前的 Dispose 方法调用清除的被引用对象。

- DoDragDrop 方法

调用 DoDragDrop 方法，以开始拖动操作，其定义如下：

```
public DragDropEffects DoDragDrop (object data, DragDropEffects allowedEf-
fects);
```

参数：

data——将被拖动的数据。

allowedEffects——DragDropEffects 值之一。

返回值：

代表拖动操作效果的 DragDropEffects 枚举值。

allowedEffects 参数决定了将发生的拖动操作。如果需要与其他进程的应用程序交互，那么数据应为基础受控类(String、Bitmap 或 Metafile)，或实现了 ISerializable 或 IDataObject 的对象。

- EndInvoke 方法

调用 EndInvoke 方法，以获取由 IAsyncResult 接口代表的异步操作的返回值。该方法的定义如下：

```
public object EndInvoke(IAsyncResult asyncResult);
```

参数：

asyncResult——由调用 BeginInvoke 返回的指定被激活的异步操作。

返回值：

由异步调用生成的对象。

如果 asyncResult 参数为空引用，则方法将抛出 ArgumentNullException 异常。如果 asyncResult 不是由相同控件的 BeginInvoke 方法创建，则方法将抛出 ArgumentException 异常。

如果异步操作尚未完成，则本方法将阻塞直到结果可用。

- FindForm 方法

调用 FindForm 方法，以获取控件所在的窗体(控件的父对象不一定是窗体)，其定义如下：

```
public Form FindForm();
```

返回值：

包含控件的窗体。

- Focus 方法

调用 Focus 方法，以为控件设置焦点，其定义如下：

```
public bool Focus();
```

返回值：

如果成功为控件设置了焦点，则返回 true，否则返回 false。

如果控件成功收到输入焦点，则 Focus 方法返回 true。控件可以具有输入焦点，而不显示任意可视化提示。

如果其 ControlStyles. Selectable 风格位为 true，并且所有父对象都可见和被启用，则控件可被选择和接收输入焦点。

- FromChildHandle 方法

调用 FromChildHandle 方法，以返回包含指定句柄的控件，其定义如下：

```
public static Control FromChildHandle(int handle);
```

参数：

handle——将搜索的 HWND。

返回值：

与句柄相关的 Control。如果找不到相应控件，则返回空引用。

本方法将向上检索 HWND 父对象链，直到找到与控件相关句柄。FromChildHandle 方法比 FromHandle 方法更加稳定，因为它能返回与句柄相关的正确控件。

- FromHandle 方法

调用 FromHandle 方法，以返回当前与句柄相关的控件，其定义如下：

```
public static Control FromHandle(int handle);
```

参数：

handle——将检索的 HWND。

返回值：

与 handle 相关的控件。如果没有找到相应控件，则返回空引用。

- GetChildAtPoint 方法

调用 GetChildAtPoint 方法，以获取指定坐标处的子控件，其定义如下：

```
public Control GetChildAtPoint(Point pt);
```

参数：

pt——客户区坐标。

返回值：

指定点处的控件。如果指定点处无除本控件外的其他控件，则返回空引用。

- GetContainerControl 方法

调用 GetContainerControl 方法，以返回父控件链中的下一个 ContainerControl，其定义如下：

```
public virtual IContainerControl GetContainerControl();
```

- GetNextControl 方法

调用 GetNextControl 方法，以获取子控件 Tab 顺序中的下一个控件。该方法的定义如下：

```
public Control GetNextControl(Control ctl,bool forward);
```

参数：

ctl——搜索的起始控件。

forward——确定是否应在 Tab 顺序中向前搜索。

返回值：

Tab 顺序中的下一个控件。

- GetStyle 方法

调用 GetStyle 方法，以确定本控件的指定控件风格位是否被设置，其定义如下：

```
protected bool GetStyle(ControlStyles flag);
```

参数：

flag——将获取其值的 ControlStyles 位。

返回值：

如果本控件的指定风格位被设置，则返回 true，否则返回 false。

- GetTopLevel 方法

调用 GetTopLevel 方法，以确定控件是否为顶层控件，其定义如下：

```
protected bool GetTopLevel();
```

返回值：

如果本控件为顶层控件，则返回 true，否则返回 false。

- Hide 方法

调用 Hide 方法，以通过将控件的 Visible 属性设置为 false 从而隐藏控件，其定义如下：

```
public void Hide();
```

- InitLayout 方法

调用 InitLayout 方法，以在控件被加入到其他容器后初始化布局，其定义如下：

```
protected virtual void InitLayout();
```

- Invalidate 方法

调用 Invalidate 方法，以使控件的指定区域无效，并将绘制消息发送给控件。该方法的定义如下：

```
public void Invalidate();
public void Invalidate(Rectangle rc);
public void Invalidate(bool invalidateChildren);
public void Invalidate(Rectangle rc,bool invalidateChildren);
public void Invalidate(Region region);
public void Invalidate(Region region,bool invalidateChildren);
```

参数：

rc——代表将无效的矩形区域。

invalidateChildren——如果为 true，则控件的子控件也将无效。

region——代表将无效的 Region。

调用本方法并不会强制进行同步绘制。要强制进行同步绘制，则应在调用 Invalidate 方法后，接着调用 Update 方法。如果调用本方法的无参数版本，则整个客户区都将被加入到更新区域中。

- Invoke 方法

调用 Invoke 方法，以执行控件基础句柄所在线程中的指定 Delegate，其定义如下：

```
public object Invoke(Delegate method);
public object Invoke(Delegate method,object[] args);
```

参数：

method——控件线程上下文中包含将被调用方法的 Delegate。

args——代表指定方法的参数的 Object 数组。如果方法不使用参数，则应将本参数设置为空。

返回值：

被调用的 Delegate 的返回值。如果 Delegate 无返回值，则返回空引用。

不能在控件所属的线程中调用本方法。如果控件句柄尚不存在，则本方法将在控件的父对象链中向上寻找，直到找到一个具有窗口句柄的控件或窗体。如果找不到合适的句柄，则 Invoke 方法将抛出异常，该异常将被传递给调用者。

Delegate 可以为 EventHandler 实例，它的 sender 参数中将包含本控件，而 event 参数中将包含 EventArgs. Empyt。Delegate 还可以是 MethodInvoker 实例，或其他类型的 Delegate。对 EventHandler 或 MethodInvoker Delegate 的调用速度将比对其他类型 Delegate 的调用速度快。

- InvokeGotFocus 方法

调用 InvokeGotFocus 方法，以激发 GotFocus 事件，其定义如下：

```
protected void InvokeGotFocus(Control toInvoke,EventArgs e);
```

参数：

toInvoke——与事件相关的控件。

e——包含事件数据的 EventArgs。

- InvokeLostFocus 方法

调用 InvokeLostFocus 方法，以激发 LostFocus 事件，其定义如下：

```
protected void InvokeLostFocus(Control toInvoke,EventArgs e);
```

参数：

toInvoke——与事件相关的控件。

e——包含事件数据的 EventArgs。

- InvokeOnClick 方法

调用 InvokeOnClick 方法，以为指定控件发出 Click 事件，其定义如下：

```
protected void InvokeOnClick(Control toInvoke,EventArgs e);
```

参数：

toInvoke——与事件相关的控件。

e——包含事件数据的 EventArgs。

- InvokePaint 方法

调用 InvokePaint 方法，以为指定控件发出 Paint 事件，其定义如下：

```
protected void InvokePaint(Control c,PaintEventArgs e);
```

参数：

toInvoke——与事件相关的控件。

e——包含事件数据的 EventArgs。

- InvokePaintBackground 方法

调用 InvokePaintBackground 方法，以为指定控件发出 PaintBackground 事件，其定义如下：

```
protected void InvokePaintBackground(Control c,PaintEventArgs e);
```

参数：

toInvoke——与事件相关的控件。

e——包含事件数据的 EventArgs。

- IsInputChar 方法

调用 IsInputChar 方法，以确定 charCode 是否为控件能辨认的输入字符。该方法的定义如下：

```
protected virtual bool IsInputChar(char charCode);
```

参数：

charCode——将测试的字符。

返回值：

如果字符应被直接发送给控件而无需预处理，则返回 true，否则返回 false。

在 Windows 消息预处理中，将调用本方法以确定指定的输入字符是否需要预处理。如

果 IsInputChar 返回 true，则指定字符将被直接发送给控件；否则，字符将被预处理，并只当被成功处理后才被发送给控件。字符的预处理包括检查字符是否为另一控件的助记符等。

- IsInputKey 方法

调用 IsInputKey 方法，以确定指定键是正常输入键还是需要预处理的特殊键。该方法的定义如下：

```
protected virtual bool IsInputKey(Keys keyData);
```

参数：

keyData——Keys 枚举值之一。

返回值：

如果键应被直接发送给控件，则返回 true，否则返回 false。

在 Windows 消息预处理中，将调用本方法以确定指定的输入键是否需要预处理。如果 IsInputKey 返回 true，则指定键将被直接发送给控件；否则，键将被预处理，并只当被成功处理后才被发送给控件。键的预处理包括检查键是否为 Tab、Esc 等。

- IsMnemonic 方法

调用 IsMnemonic 方法，以确定字符是否为字符串中的助记符，其定义如下：

```
public static bool IsMnemonic(char charCode,string text);
```

参数：

charCode——将测试的字符。

text——将搜索的字符串。

返回值：

如果 charCode 为助记符，则返回 true，否则返回 false。

助记符就是字符串中第一个 & 后的字符。

- PerformLayout 方法

调用 PerformLayout 方法，以强制控件对它的子控件执行布局逻辑，其定义如下：

```
public void PerformLayout();
public void PerformLayout(Control affectedControl,string affectedProperty);
```

参数：

affectedControl——最近被修改的控件。

affectedProperty——最近被修改的控件属性。

- PointToClient 方法

调用 PointToClient 方法，以计算指定屏幕位置的客户坐标，其定义如下：

```
public virtual Point PointToClient(Point p);
```

参数：

p——将被转换的屏幕坐标。

返回值：

计算所得的客户坐标。

- PointToScreen 方法

调用 PointToScreen 方法，以计算指定客户位置的屏幕坐标，其定义如下：

```
public virtual Point PointToScreen(Point p);
```

参数：

p——将被转换的客户坐标。

返回值：

转换所得的屏幕坐标。

- PreProcessMessage 方法

调用 PreProcessMessage 方法，以在分遣消息前对其进行预处理，其定义如下：

```
public virtual bool PreProcessMessage(MSG msg);
```

参数：

msg——代表将被预处理的消息。

返回值：

如果消息由控件处理，则返回 true，否则返回 false。

- ProcessCmdKey 方法

调用 ProcessCmdKey 方法，以预处理命令键，其定义如下：

```
protected virtual bool ProcessCmdKey(MSG msg,Keys keyData);
```

参数：

msg——代表将被处理的消息。

keyData——用户按下的键代码。

返回值：

如果字符由控件处理，则返回 true，否则返回 false。

在处理命令键的消息预处理过程中将调用本方法。命令键就是比常规输入键具有更高优先级的键，例如菜单快捷键。

ProcessCmdKey 方法首先检查控件是否为快捷菜单，如果是，则允许 ContextMenu 处理命令键。如果命令键并非菜单快捷键，并且控件有父对象，则键将被传递给父对象的 ProcessCmdKey 方法。除了输入键外，keyData 还能代表同时按下的修饰键（Shift、Ctrl 和 Alt）。

当在派生类中重载 ProcessCmdKey 方法时，控件应返回 true 以标志它处理了键。对于没有由控件处理的键，应返回调用基类的 ProcessCmdKey 方法的返回值。控件很少重载本方法。

- ProcessDialogChar 方法

调用 ProcessDialogChar 方法，以处理对话框字符，其定义如下：

```
protected virtual bool ProcessDialogChar(char charCode);
```

参数：

charCode——将处理的字符。

返回值：

如果字符由控件处理，则返回 true，否则返回 false(以进行进一步处理)。

在处理对话框字符的过程中，将调用本方法。只当 IsInputChar 方法标志控件不处理字符时，本方法才会被调用。ProcessDialogChar 方法只是将字符发送给父对象的 ProcessDialogChar 方法；如果控件无父对象，则返回 false。Form 类重载本方法以执行对话框字符的实际处理。

当在派生类中重载 ProcessDialogChar 方法时，控件应返回 true 以标志它已经处理了字符。对于不由控件处理的字符，则返回值应由调用基类 ProcessDialogChar 方法的结果决定。

- ProcessDialogKey 方法

调用 ProcessDialogKey 方法，以处理对话框键，其定义如下：

```
protected virtual bool ProcessDialogKey(Keys keyData);
```

参数：

keyData——代表将被处理的键的 Keys 值。

返回值：

如果键由控件处理，则返回 true，否则返回 false。

在处理对话框键(例如 Tab、Enter、Esc 和箭头键)的过程中，将调用本方法。只当 IsInputKey 方法标志控件不处理键符时，本方法才会被调用。ProcessDialogKey 方法只是将字符发送给父对象的 ProcessDialogKey 方法；如果控件无父对象，则返回 false。Form 类重载本方法以执行对话框字符的实际处理。

当在派生类中重载 ProcessDialogKey 方法时，控件应返回 true 以标志它已经处理了键。对于不由控件处理的键，则返回值应由调用基类 ProcessDialogKey 方法的结果决定。

- ProcessKeyEventArgs 方法

调用 ProcessKeyEventArgs 方法，以处理键消息并生成合适的控件事件，其定义如下：

```
protected virtual bool ProcessKeyEventArgs(ref Message m);
```

参数：

m——将处理的 Windows 消息。

返回值：

如果消息由控件处理，则返回 true，否则返回 false。

当控件收到键盘消息时将调用本方法。ProcessKeyEventArgs 方法通过调用 OnKeyPress、OnKeyDown 或 OnKeyUp 方法，来为消息生成合适的键事件。m 参数中包含必须被处理的 Windows 消息，可能的 Message. Msg 属性值包括 WM_CHAR、WM_KEYDOWN、WM_SYSKEYDOWN、WM_KEYUP、WM_SYSKEYUP 和 WM_IMECHAR。

当在派生类中重载 ProccssKeyEventArgs 方法时，控件应返回 true 以标志它已经处理了键。对于不由控件处理的键，则返回值应由调用基类 ProcessKeyEventArgs 方法的结果