.NET框架程序员参考手册·用户界面篇

吉尚戎　等编著

国防工业出版社

·北京·

内 容 简 介

本书详细介绍了.NET 框架中有关用户界面的内容。全书共 12 章，主要内容包括：基础控件功能支持，Windows 窗体和应用程序对象，文本显示控件，图像显示和存储控件，命令控件，选项列表控件，值设置控件，文本编辑控件，分组控件，菜单控件，数据库数据显示控件和其他常用控件等。

图书在版编目(CIP)数据

.NET 框架程序员参考手册·用户界面篇/吉尚戎等编著 .—北京：国防工业出版社，2002.1
ISBN 7-118-02772-3

Ⅰ.N... Ⅱ.吉... Ⅲ.计算机网络－程序设计
Ⅳ.TP393

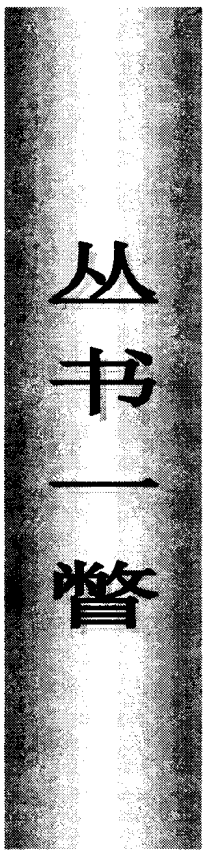
中国版本图书馆 CIP 数据核字(2001)第 091790 号

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号)
(邮政编码 100044)
北京奥隆印刷厂印刷
新华书店经售
*
开本 787×1092 1/16 印张 36 839 千字
2002 年 1 月第 1 版 2002 年 1 月北京第 1 次印刷
印数：1—4000 册 定价：52.00 元

Visual Studio.NET 7.0 是微软推出的新一代可视化集成开发环境，其中的.NET 就是指.NET 框架（.NET Framework）。.NET 框架是一种用于构建、配置、运行 Web 服务和应用程序的多语言环境，它主要由统一的编程类库、通用语言运行库（Common Language Runtime）和 ASP.NET（Active Server Pages.NET）三个部分组成。.NET 框架不但提供了诸如自动内存管理之类的很多强有力的功能，而且它的引入使得多语言间的无缝互用成为现实。

从某种程度上说，新版本的 Visual Studio 就是以.NET 框架为中心的：新引入的 C#语言本身并无类库，而是充分利用.NET 框架提供的功能；Visual Basic 7.0 语言上做了很大修改，而这些修改正是为了实现与.NET 框架的无缝兼容；Visual C++ 7.0 中提供了受控代码（Managed Extensions）编程，使得由 C++编写的代码也依然能够使用.NET 框架的服务。

.NET 框架中的类型有很多的功能，例如，封装数据结构、执行 I/O 操作、访问数据、控制服务器、获取类信息以及激活安全检查等。.NET 框架中既包括比较抽象的基类，也包括由基类派生的、具有实际功能的类。这些派生类已经提供足够强大的功能，但是如果需要，程序员依然可以通过继续派生提供更强大的功能。.NET 框架还包括接口及其默认实现。要使用接口的功能，程序员可以自己实现这些接口，也可以直接使用（或派生）运行库中的接口实现类。



由于.NET 框架类库的内容非常浩繁，为了向读者提供更具针对性的参考信息，《.NET 框架程序员参考手册》丛书分为以下 6 册：

- **框架基础篇**

本篇主要包括整个.NET 框架的根名称空间：System，以及微软的两个服务名称空间：Microsoft.ComServices 和 Microsoft.Win32。这些名称空间为用户提供了底层功能和服务支持，同时也是开发高级功能的基础。

- **数据访问篇**

本篇主要包括为数据 I/O 提供支持的名称空间：System.IO；为数据库访问提供支持的名称空间：System.Data 及其下属的三级名称空间。通过这些名称空间，用户能够方便地进行数据存取、数据库事务和 XML 编档。

- **网络编程篇**

本篇主要包括为进行网络和 Web 服务提供支持的名称空间：System.Security，System.Net，System.Web 及其下属的三级名称空间等。

- **用户界面篇**

本篇主要包括为用户界面提供支持的名称空间：System.Windows.Forms 等。

- **常规操作篇**

本篇主要包括为 Windows 下的常用操作提供支持的名称空间：System.Configration，System.Globalization，System.ServiceProcess，System.Text 和 System.Timers 等。

- **组件模型篇**

本篇主要包括为组件模型提供支持的名称空间：System.ComponentModel，System.Collections，System.Resources，System.Core 和 System.Threading 等。

本书约定

由于.NET 框架涉及面极宽，为了节省篇幅，使读者能以最少的费用得到最广泛的信息，本书有以下几点约定：

- 在介绍每个名称空间时，都会将其中定义的成员以表格形式给出。读者可以根据这些表格给出的信息，迅速确定将查阅的内容。而.NET 框架命名规则也使确定成员功能变得更加容易，例如 DirectorySeparatorChar 的意思为目录分隔符字符（Directory-Separator-Char）。
- 凡是“待提供”的名称空间成员，都只是在列表中简述，而不做进一步详细说明。
- 对于那些功能、形式类似的名称空间成员，只是选择其中最具代表性的进行介绍，其他的只列表简述。读者若需得知这些简述成员的详细形式，只要参考对应的详述成员即可。
- 另外，类库的每个层次都会自动继承其基层次中的所有成员，因此在介绍时也不再介绍这些继承所得的成员，而只是指出其派生层次。这样读者即可根据其派生层次确定其中包含有哪些继承成员。当然，对于那些被重载的继承成员，我们还是会加以详细介绍。

目　录

第 1 章　基础控件功能支持

在窗体上用于输入、输出信息的图形或文字符号称为控件。控件是可视化编程的重要工具，也是面向对象编程和代码重用的典范。在.NET 框架中提供了许多有用的控件模板，供用户选择使用，如命令按钮、标签、文本框等。这些控件多数都是 Windows 的资源，这在某种意义上也体现了.NET 框架的标准性。本章将向读者介绍.NET 框架中实现了控件基础功能的类。另外需要注意的是，本书所介绍的所有控件都来自 System.Windows.Forms 名称空间，并能用于 Windows 98、Windows NT 4.0、Windows Millennium Edition、Windows 2000 和 Windows XP 等操作系统。与 System.Windows.Forms 名称空间对应的组件为：System.Windows.Forms.dll。

1.1　控 件 基 类

在.NET 框架中，Control 类是所有其他控件（包括 Windows 窗体）的基类，其中实现了 Windows 控件的通用功能。

原型：

[Visual Basic]

```
Public Class Control Inherits MarshalByRefComponent Implements IWin32Window
```

[C#]

```
public class Control : MarshalByRefComponent, IWin32Window
```

[C++]

```
public __gc class Control : public MarshalByRefComponent,IWin32Window
```

[JScript]

```
public class Control extends MarshalByRefComponent, IWin32Window
```

用途：

Control 类定义了控件（具有可视化外观的组件）的基类。

说明：

Control 类是 Object/MarshalByRefObject/MarshalByRefComponent 的子层次。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。创建自己的控件类时，一般无需继承 Control 类，而应继承 UserControl 类。

成员：

构造函数

原型：

[Visual Basic]

```
Public Sub New()
Public Sub New(ByVal text As String)
Public Sub New(ByVal parent As Control,ByVal text As String)
Public Sub New(ByVal text As String,ByVal left As Integer,ByVal top As Integer,ByVal width As Integer,ByVal height As Integer)
Public Sub New(ByVal parent As Control,ByVal text As String,ByVal left As Integer,ByVal top As Integer,ByVal width As Integer,ByVal height As Integer)
```

[C#]

```
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);
```

[C++]

```
public: Control();
public: Control(String* text);
public: Control(Control* parent,String* text);
public: Control(String* text,nt left,int top,int width,int height);
public: Control(Control* parent,String* text, int left,int top,int width, int height);
```

[JScript]

```
public function Control();
public function Control(text : String);
public function Control(parent : Control,text : String);
public function Control(text : String,left : int,top : int,width : int,height : int);
public function Control(parent : Control,text : String, left : int, top : int, width : int, height : int);
```

用途：调用 Control 类的构造函数，以初始化该类的一个新实例。

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）。

说明：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。

AccessibilityObject 属性

原型：

[Visual Basic]

```
Public ReadOnly Property AccessibilityObject As AccessibleObject
[C#]
public AccessibleObject AccessibilityObject {get;}
[C++]
public: __property AccessibleObject* get_AccessibilityObject();
[JScript]
public function get AccessibilityObject() : AccessibleObject;
```

用途：使用 AccessibilityObject 属性，以获取赋予控件的 AccessibleObject。

AccessibleDefaultActionDescription 属性

原型：

```
[Visual Basic]
Public Property AccessibleDefaultActionDescription As String
[C#]
public string AccessibleDefaultActionDescription {get; set;}
[C++]
public: __property String* get_AccessibleDefaultActionDescription();
public: __property void set_AccessibleDefaultActionDescription(String*);
[JScript]
public function get AccessibleDefaultActionDescription() : String;
public function set AccessibleDefaultActionDescription(String);
```

用途：使用 AccessibleDefaultActionDescription 属性，以获取或设置控件的默认动作描述。

AccessibleDescription 属性

原型：

```
[Visual Basic]
Public Property AccessibleDescription As String
[C#]
public string AccessibleDescription {get; set;}
[C++]
public: __property String* get_AccessibleDescription();
public: __property void set_AccessibleDescription(String*);
[JScript]
public     function get AccessibleDescription() : String;
public function set AccessibleDescription(String);
```

用途：使用 AccessibleDescription 属性，以获取或设置控件的辅助描述。

说明：本属性的默认值为空引用。

AccessibleName 属性

原型：

[Visual Basic]

```
Public Property AccessibleName As String
```

[C#]

```
public string AccessibleName {get; set;}
```

[C++]

```
public: __property String* get_AccessibleName();
public: __property void set_AccessibleName(String*);
```

[JScript]

```
public function get AccessibleName() : String;
public function set AccessibleName(String);
```

用途：使用 AccessibleName 属性，以获取或设置控件的辅助名称。

说明：本属性的默认值为空引用。

AccessibleRole 属性

原型：

[Visual Basic]

```
Public Property AccessibleRole As AccessibleRole
```

[C#]

```
public AccessibleRole AccessibleRole {get; set;}
```

[C++]

```
public: __property AccessibleRole get_AccessibleRole();
public: __property void set_AccessibleRole(AccessibleRole);
```

[JScript]

```
public function get AccessibleRole() : AccessibleRole;
public function set AccessibleRole(AccessibleRole);
```

用途：使用 AccessibleRole 属性，以获取或设置控件的辅助角色。

说明：本属性的默认值为 Default。

AllowDrop 属性

原型：

[Visual Basic]

```
Overridable Public Property AllowDrop As Boolean
```

[C#]

```
public virtual bool AllowDrop {get; set;}
```

[C++]

```
public: __property virtual bool get_AllowDrop();
public: __property virtual void set_AllowDrop(bool);
```

[JScript]

```
public    function get AllowDrop() : Boolean;
public function set AllowDrop(Boolean);
```

用途：使用 AllowDrop 属性，以确定控件是否可接受用户拖拽于其中的数据。

说明：如果本控件允许拖拽，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拽操作。

Anchor 属性

原型：

[Visual Basic]

```
Overridable Public Property Anchor As AnchorStyles
```

[C#]

```
public virtual AnchorStyles Anchor {get; set;}
```

[C++]

```
public: __property virtual AnchorStyles get_Anchor();
public: __property virtual void set_Anchor(AnchorStyles);
```

[JScript]

```
public function get Anchor() : AnchorStyles;
public function set Anchor(AnchorStyles);
```

用途：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。

说明：本属性的值为 AnchorStyles 枚举值之一。

BackColor 属性

原型：

[Visual Basic]

```
Overridable Public Property BackColor As Color
```

[C#]

```
public virtual Color BackColor {get; set;}
```

[C++]

```
public: __property virtual Color get_BackColor();
public: __property virtual void set_BackColor(Color);
```

[JScript]

```
public function get BackColor() : Color;
public function set BackColor(Color);
```

用途：使用 BackColor 属性，以获取或设置本控件的背景颜色。

说明：本属性为环境属性，因此总是返回非空值。

BackgroundImage 属性

原型：

[Visual Basic]

```
Overridable Public Property BackgroundImage As Image
```

[C#]

```
public virtual Image BackgroundImage {get; set;}
```

[C++]

```
public: __property virtual Image* get_BackgroundImage();
public: __property virtual void set_BackgroundImage(Image*);
```

```
[JScript]
public function get BackgroundImage() : Image;
public function set BackgroundImage(Image);
```

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

BindingContext 属性

原型：

```
[Visual Basic]
Overridable Public Property BindingContext As BindingContext
[C#]
public virtual BindingContext BindingContext {get; set;}
[C++]
public: __property virtual BindingContext* get_BindingContext();
public: __property virtual void set_BindingContext(BindingContext*);
[JScript]
public function get BindingContext() : BindingContext;
public function set BindingContext(BindingContext);
```

用途：使用 BindingContext 属性，以获取或设置对象的 BindingContext。

说明：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，就指定了所有数据绑定控件指向的底层列表项。

下面给出使用 BindingContext 属性的示例：

```
[Visual Basic]
Protected Sub BindControls()

    ' 为两个 TextBox 控件创建两个 Binding 对象，其中 Text 属性与数据绑定
    ' 数据源是一个数据集（ds），其数据成员为一个字符串：RelationName.ColumnName
    text1.DataBindings.Add(New Binding ("Text", ds, "customers.custName"))
    text2.DataBindings.Add(New Binding ("Text", ds, "customers.custID"))

    ' 将 DateTimePicker 绑定到 TableName.RelationName.ColumnName
    DateTimePicker1.DataBindings.Add(New Binding ("Value", ds, "customers.CustToOrders.OrderDate"))

    ' 为新的 Binding 对象添加 Parse 和 Format 事件的 Delegate，并将对象加入到第三个 TextBox 控件的 BindingCollection 中
    ' Delegate 必须在将 Binding 对象加入到集合前被加入；否则将不会格式化，直到数据源的 BindingManagerBase 的 Current 对象发生变化
    Dim b As Binding = New Binding ("Text", ds, "customers.custToOrders.OrderAmount")
    AddHandler b.Parse,  New ConvertEventHandler(AddressOf CurrencyStringToDecimal)
    AddHandler b.Format, New ConvertEventHandler(AddressOf DecimalToCurrencyString)
    text3.DataBindings.Add(b)
```

```
    ' 获取 Customers 表的 BindingManagerBase
    bmCustomers = Me.BindingContext(ds, "Customers")

    ' 使用 RelationName，获取 Orders 表的 BindingManagerBase
    bmOrders = Me.BindingContext(ds, "customers.CustToOrders")

    ' 将第四个 TextBox 控件的 Text 属性绑定到第三个控件的 Text 属性
    text4.DataBindings.Add("Text", text3, "Text")

End Sub
```

Bottom 属性

原型：

```
[Visual Basic]
Public ReadOnly Property Bottom As Integer
[C#]
public int Bottom {get;}
[C++]
public: __property int get_Bottom();
[JScript]
public function get Bottom() : int;
```

用途：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。

说明：本属性的值等于 Top 属性与 Height 属性值之和。

Bounds 属性

原型：

```
[Visual Basic]
Public Property Bounds As Rectangle
[C#]
public Rectangle Bounds {get; set;}
[C++]
public: __property Rectangle get_Bounds();
public: __property void set_Bounds(Rectangle);
[JScript]
public function get Bounds() : Rectangle;
public function set Bounds(Rectangle);
```

用途：使用 Bounds 属性，以获取或设置本控件的边界矩形。

CanFocus 属性

原型：

```
[Visual Basic]
Public ReadOnly Property CanFocus As Boolean
[C#]
public bool CanFocus {get;}
```

```
[C++]
public: __property bool get_CanFocus();
[JScript]
public function get CanFocus() : Boolean;
```

用途：使用 CanFocus 属性，以确定控件是否能接收焦点。

说明：如果控件可以接收焦点，则本属性的值为 true，否则为 false。要是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

原型：

```
[Visual Basic]
Public ReadOnly Property CanSelect As Boolean
[C#]
public bool CanSelect {get;}
[C++]
public: __property bool get_CanSelect();
[JScript]
public function get CanSelect() : Boolean;
```

用途：使用 CanSelect 属性，以确定本控件是否可被选择。

说明：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

原型：

```
[Visual Basic]
Public Property Capture As Boolean
[C#]
public bool Capture {get; set;}
[C++]
public: __property bool get_Capture();
public: __property void set_Capture(bool);
[JScript]
public function get Capture() : Boolean;
public function set Capture(Boolean);
```

用途：使用 Capture 属性，以确定控件是否被鼠标捕获。

说明：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

原型：

[Visual Basic]

```
Public Property CausesValidation As Boolean
```

[C#]

```
public bool CausesValidation {get; set;}
```

[C++]

```
public: __property bool get_CausesValidation();
public: __property void set_CausesValidation(bool);
```

[JScript]

```
public function get CausesValidation() : Boolean;
public function set CausesValidation(Boolean);
```

用途：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。

说明：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

原型：

[Visual Basic]

```
Public ReadOnly Property ClientRectangle As Rectangle
```

[C#]

```
public Rectangle ClientRectangle {get;}
```

[C++]

```
public: __property Rectangle get_ClientRectangle();
```

[JScript]

```
public function get ClientRectangle() : Rectangle;
```

用途：使用 ClientRectangle 属性，以获取代表控件客户区的矩形。

说明：由于客户坐标是相对于控件客户区的左上角的，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

原型：

[Visual Basic]

```
Overridable Public Property ClientSize As Size
```

[C#]

```
public Size ClientSize {virtual get; virtual set;}
```

[C++]

```
public: __property virtual Size get_ClientSize();
public: __property virtual void set_ClientSize(Size);
```