经典范例 50 讲系列

北京希望电子出版社　总策划
吴财军　编　写

C#
经典范例50讲

北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

经典范例 50 讲系列

北京希望电子出版社　总策划
吴财军　编　写

C# 经典范例 50 讲

北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

内 容 简 介

本书是介绍面向对象的程序开发语言 C#语言的专题书籍，C#语言能够使程序员在微软提供的.NET 开发平台上快速开发出种类丰富的应用程序。书中通过 50 个经典实例详细介绍了用 C#语言开发应用的方法与技巧。

本书由 50 讲组成，第 1～9 讲介绍使用 C＃语言进行基本的 Windows 应用程序开发，同时还介绍了.NET 开发平台的一些特点以及如何利用.NET 平台进行开发等内容；第 10～14 讲介绍在 C＃中进行多线程应用程序的开发；第 15～19 讲介绍数据库应用程序开发；第 20～26 讲介绍多媒体应用程序开发，详细讲解了在窗体上绘制各种图形，操作图片、声音、视频等高级多媒体效果的技巧与方法；第 27～34 讲介绍利用操作系统提供的功能进行高级的应用程序开发；第 35～41 讲介绍互联网应用程序开发；第 42～46 讲介绍 Web 应用程序开发，这是 C＃的一个突出特点，讲解了直接在 C＃的开发环境中编写 Web 应用程序的方法；第 47～50 讲介绍开发各种分布式应用系统的方法。

本书内容丰富，讲解详细，范例与软件功能紧密结合，具有很强的实用性，能让读者尽快熟悉 C＃编程的特点和优点。本书面向初中级用户和社会 C#培训班。

本版 CD 内容为书中实例源代码。

盘书系列名：经典范例 50 讲系列

盘　书　名：C#经典范例 50 讲

文本著作者：吴财军

责任编辑：周凤明

CD 制作者：希望多媒体开发中心

CD 测试者：希望多媒体测试部

出版、发行者：北京希望电子出版社

地　　址：北京市海淀区知春路甲 63 号卫星大厦三层 100080

网址: www.bhp.com.cn　E-mail: zwb@bhp.com.cn

电话: 010-62520290,62521724,62528991,62630301,62524940,62521921,82610344

（发行），010-82675588-202（门市），010-82675588-501,82675588-201（编辑部）

经　　销：各地新华书店、软件连锁店

排　　版：希望图书输出中心　马伟科

CD 生产者：北京中新联光盘有限责任公司

文本印刷者：北京东升印刷厂

开本 / 规格：787 毫米×1092 毫米　1/16　26.75 印张　625 千字

版次 / 印次：2003 年 4 月第 1 版　2003 年 4 月第 1 次印刷

印　　数：0001～5000 册

本 版 号：ISBN 7-89498-085-4

定　　价：38.00 元（本版 CD）

前　言

本书是一本实践性很强的编程书籍，整本书都是以实例形式向读者介绍C＃编程语言的各方面内容，让读者可以很容易的学习C＃编程。全书精心安排了50个实例，涉及C＃编程的各个方面，每讲中的实例都由浅入深的进行编排，利于读者的学习，满足了不同层次读者开发应用程序的需要。

每个实例都具有较强的针对性，突出讲解C＃编程的特点、开发思想、技术和方法。在讲解中把每个实例分成若干部分，分别详细介绍，利于读者的领会。在每个实例后面附有详细的源代码，对重要的程序都有注释，便于读者理解掌握各个实例所涉及的知识。通过本书的学习，读者将会感受到新一代的编程语言的强大功能以及特殊魅力。

全书共分为八大部分，各部分的大致内容介绍如下。

从第1讲到第9讲主要介绍Windows应用程序开发，展现了如何使用C＃语言进行基本的Windows应用程序开发，同时还介绍了.NET开发平台的一些特点以及如何利用.NET平台进行开发等内容；

从第10讲到第14讲主要介绍多线程应用程序开发，提出了如何在C＃中进行多线程应用程序的开发，同时还利用C#的强大功能解决了线程同步、线程互斥等问题；

从第15讲到第19讲主要介绍数据库应用程序开发，讲解了如何利用C＃提供的强大的数据库访问能力进行数据库应用程序的开发，这样大大地减轻了程序员的开发难度；

从第20讲到第26讲主要介绍多媒体应用程序开发，介绍了C＃语言如何实现在窗体上绘制各种的图形，如何操作图片、声音、视频等高级的多媒体效果；

从第27讲到第34讲主要介绍Windows系统程序开发，说明了如何在C＃开发环境中开发系统级的应用程序，如何利用操作系统提供的功能进行高级的应用程序开发；

从第35讲到第41讲主要介绍互联网应用程序开发，描述了C＃提供的对互联网的强大支持，虽然这里列举了一些互联网应用实例，但是仅仅用到了C＃互联网功能的一小部分，其强大的互联网功能还需进一步的挖掘；

从第42讲到第46讲主要介绍Web应用程序开发，这是C＃的一个突出特点。程序员可以利用这一特点直接在C＃的开发环境中编写Web应用程序，而不需要再使用其他的工具进行开发。

从第47讲到第50讲主要介绍分布式应用程序开发。作为新一代的编程语言，C＃支持分布式应用程序的开发又是其特点之一。充分利用C＃提供的分布式应用程序开发功能，程序员可以轻松的开发出各种分布式应用系统。

虽然本书制作了大量的编程实例，内容尽可能的覆盖C＃编程的各个方面，但是C＃功能强大，涉及的方面很多，各讲中的实例仅仅是对相关方面编程的一个初步的探索，不可能完全在这些实例中展现其强大开发能力的方方面面，所以希望本书能给读者起到一个指引方向的作用，读者在学习完本书后还需要更多的练习和更深入的学习才能真正的领会C＃编程的魅力。

本书由吴财军执笔编写，此外，郑丹、王瑾、吴浩、李炎、刘伟、刘华刚、朱峰、赵晓燕、李晓、马苍等同志在整理材料方面给予了作者很大的帮助，在此一并致以感谢！

由于编者水平有限，缺点和错误恳请专家和广大读者批评指正。

编　者

2003.2

目 录

Windows 应用程序开发

在 Windows 系统中开发应用程序是很重要的一种软件开发方式。大量的应用程序需要在 Windows 系统中进行编写，以便适应不断发展的需求。在这一讲里将介绍如何使用 Visual Studio .Net 中的 C#语言来开发我们熟悉的 Windows 常用小工具应用程序。通过编写 Windows 下的常用小工具应用程序，初步了解 C#语言编程的过程，熟悉 C#语言的语法，领会 C#的思想。同时，还学习使用 Visual Studio .Net 开发平台提供的众多组件，比如：按钮（Button）、文本框（TextBox）、菜单（MainMenu）等等。另外还将对.Net 开发框架做一个初步的介绍。

后面的第 1 讲到第 9 讲将介绍 Windows 应用程序的开发：第 1 讲介绍如何动态改变应用程序主菜单的显示方式；第 2 讲介绍如何根据当前选择的对象动态地改变鼠标右键菜单；第 3 讲介绍如何在 C#中使用 Timer 组件和 NumericUpDown 组件；第 4 讲介绍如何在 C#中使用按钮组件；第 5 讲介绍如何在 C#中使用 RichTextBox 组件；第 6 讲介绍如何在 C#中进行二进制文件的读写；第 7 讲介绍如何在 C#中进行文本文件的读写；第 8 讲介绍如何在 C#中使用 TreeView 组件、ListView 组件和 Splitter 组件；第 9 讲介绍如何在 C#中进行基本的文档打印操作。

第 1 讲　可改变的主菜单

1.1　实现目标

在很多的 Windows 应用程序中，主菜单是一个很常用的工具，因为它可以为用户提供丰富的功能选项。本实例将制作一个可以改变的主菜单，它能根据用户的不同选择，显示不同的主菜单。这对于开发多语言版本的应用程序来说是非常有用的技术，如图 1-1，图 1-2 所示。

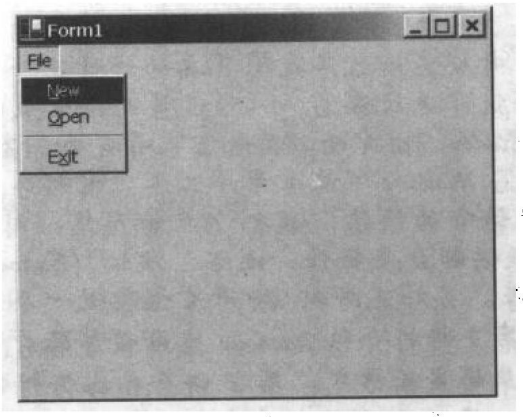

图 1-1　程序的英文主菜单

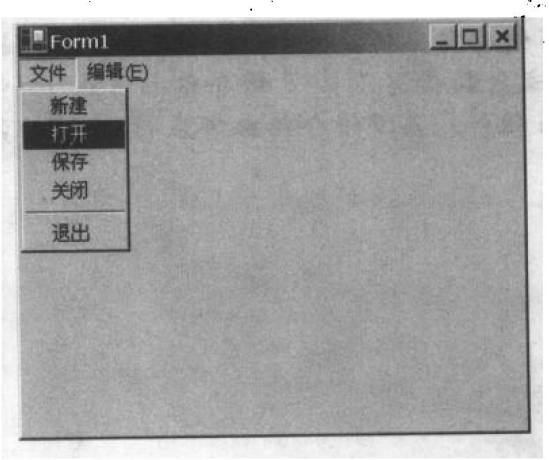

图 1-2　程序的中文主菜单

1.2　背景知识介绍

很多 Windows 应用程序中都需要主菜单，在 Visual Studio.Net 开发环境中提供了 MainMenu 组件来实现对主菜单的操作与控制，这样就方便了编程者动态地添加、删除菜单项。下面列举出 MainMenu 组件常用的属性和方法：

- MainMenu.Name

该属性说明了一个 MainMenu 类的实例名称，在这个程序中就可以直接使用该组件。

- MainMenu.MenuItems

该属性说明了与 MainMenu 相关联的 MenuItems 对象的集合。

- MenuItem

该 MenuItem 对象表示在 MainMenu 或者 ContextMenu 内显示的单个项。

- MenuItem.Add

该方法向与 MenuItem 相关联的项中添加一个新的菜单项，可以说明菜单的名称以及处理菜单项单击事件的函数。

1.3　实现步骤及技术分析

1. 界面设计

（1）启动 Visual Studio，新建一个 Visual C#项目，然后在“模板”窗格中选择“Windows 应用程序”。

（2）设置组件的属性如下：

```
// menuItem1 的属性设置
this.menuItem1.Index = 0;
this.menuItem1.MenuItems.AddRange（new System.Windows.Forms.MenuItem[] {
        this.miNew,
        this.miOpen,
        this.menuItem4,
        this.miExit}）；
this.menuItem1.Text = "&File";
// miNew 菜单项的设置
this.miNew.Index = 0;
this.miNew.Text = "&New";
this.miNew.Click += new System.EventHandler（this.MenuSelect）；
// miOpen 菜单项的设置
this.miOpen.Index = 1;
this.miOpen.Text = "&Open";
this.miOpen.Click += new System.EventHandler（this.MenuSelect）；
// menuItem4 菜单项的设置
this.menuItem4.Index = 2;
this.menuItem4.Text = "-";
// miExit 菜单项的设置
this.miExit.Index = 3;
this.miExit.Text = "E&xit";
this.miExit.Click += new System.EventHandler（this.miExit_Click）；
```

2. 设定私有变量

程序中为了能够动态地转换主菜单的显示内容，定义了一个主菜单类的对象实体，用

来保存新的中文菜单项的内容。另外，还定义了主菜单下面的子菜单类的对象实体，其代码如下。

```
// 新的菜单项变量。
private MainMenu mmNewMenu;
// 文件子菜单项变量
private MenuItem miFile;
// 编辑子菜单项变量
private MenuItem miEdit;
```

3. 初始化中文菜单

在此实例的设计中，静态设计的是英文主菜单，当程序初始化的时候再动态生成一个不可见的中文主菜单，当用户单击了英文菜单中的 New 或者 Open 菜单项时，才显示相应的中文主菜单。下面的代码说明了如何动态地生成一个中文的主菜单项。

```
public Form1（）
{
    // Windows 窗体设计器支持所必需的
    InitializeComponent（）；

    // TODO: 在 InitializeComponent 调用后添加任何构造函数代码
    // 建立主菜单。
    mmNewMenu = new MainMenu（）；
    // 建立第一个文件子菜单。
    miFile = new MenuItem（"文件"）；
    // 建立子菜单项中的每一个菜单项，并且关联其对应的菜单命令处理函数。
    miFile.MenuItems.Add（"新建",new System.EventHandler（miNew_Click））；
    miFile.MenuItems.Add（"打开",new System.EventHandler（miOpen_Click））；
    miFile.MenuItems.Add（"保存",new System.EventHandler（miSave_Click））；
    miFile.MenuItems.Add（"关闭",new System.EventHandler（miClose_Click））；
    miFile.MenuItems.Add（"-"）；
    miFile.MenuItems.Add（"退出",new System.EventHandler（miExit_Click））；
    // 建立第二个编辑子菜单
    miEdit = new MenuItem（"编辑（&E）"）；
    miEdit.MenuItems.Add（"剪切　（&T）"）；
    miEdit.MenuItems.Add（"复制　（&C）"）；
    miEdit.MenuItems.Add（"粘贴　（&P）"）；
    mmNewMenu.MenuItems.Add（miFile）；
    mmNewMenu.MenuItems.Add（miEdit）；
}
```

4. New 与 Open 菜单项的处理过程

当单击了该实例程序的英文菜单项 New 或者 Open 的时候，生成一个对话框来表示执行了相应的操作，然后加载中文主菜单进行显示，其生成的代码如下。

```
// 响应单击菜单项的事件。
private void MenuSelect（object sender, System.EventArgs e）
{
    MessageBox.Show（"单击了一个菜单项.....", "提示框"）;
    LoadSecondMenu（）;
}
// 加载新的菜单。
private void LoadSecondMenu（）
{
    this.Menu = mmNewMenu;
}
```

5. 中文菜单项的处理

当进入到中文菜单项之后，对于每一个子菜单项都可以定义与之相关的菜单命令处理函数，在这里只是简单地显示了一个对话框来表示执行了相关的菜单命令。当单击了中文子菜单项“关闭”之后，将再次回到英文菜单中。其源代码如下。

```
private void miSave_Click（object sender, System.EventArgs e）
{
    MessageBox.Show（"单击了'保存'菜单项.....", "提示框"）;
}
private void miClose_Click（object sender, System.EventArgs e）
{
    // 重新显示英文的主菜单项。
    this.Menu = mainMenu1;
}
```

附程序完整源代码，如下所示：

程序清单

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ChMenu
```

```
{
    /// <summary>
    /// 动态切换主菜单。
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem menuItem4;
        private System.Windows.Forms.MenuItem miNew;
        private System.Windows.Forms.MenuItem miOpen;
        private System.Windows.Forms.MenuItem miExit;
        /// <summary>
        /// 必需的设计器变量。
        /// </summary>
        private System.ComponentModel.Container components = null;
        // 新的菜单项变量。
        private MainMenu mmNewMenu;
        private MenuItem miFile;
        private MenuItem miEdit;
        public Form1 ()
        {
            // Windows 窗体设计器支持所必需的
            InitializeComponent ();
            mmNewMenu = new MainMenu ();
            miFile = new MenuItem ("文件");
            miFile.MenuItems.Add ("新建",new System.EventHandler (miNew_Click));
            miFile.MenuItems.Add ("打开",new System.EventHandler (miOpen_Click));
            miFile.MenuItems.Add ("保存",new System.EventHandler (miSave_Click));
            miFile.MenuItems.Add ("关闭",new System.EventHandler (miClose_Click));
            miFile.MenuItems.Add ("-");
            miFile.MenuItems.Add ("退出",new System.EventHandler (miExit_Click));
            miEdit = new MenuItem ("编辑 (&E)");
            miEdit.MenuItems.Add ("剪切 (&T)");
            miEdit.MenuItems.Add ("复制 (&C)");
            miEdit.MenuItems.Add ("粘贴 (&P)");
            mmNewMenu.MenuItems.Add (miFile);
            mmNewMenu.MenuItems.Add (miEdit);
```

```
        // TODO: 在 InitializeComponent 调用后添加任何构造函数代码
    }
    /// <summary>
    /// 清理所有正在使用的资源。
    /// </summary>
    protected override void Dispose ( bool disposing )
    {
        if ( disposing )
        {
            if (components != null)
            {
                components.Dispose ();
            }
        }
        base.Dispose ( disposing );
    }
    #region Windows Form Designer generated code
    /// <summary>
    /// 设计器支持所需的方法 - 不要使用代码编辑器修改此方法的内容。
    /// </summary>
    private void InitializeComponent ()
    {
        this.mainMenu1 = new System.Windows.Forms.MainMenu ();
        this.menuItem1 = new System.Windows.Forms.MenuItem ();
        this.miNew = new System.Windows.Forms.MenuItem ();
        this.miOpen = new System.Windows.Forms.MenuItem ();
        this.menuItem4 = new System.Windows.Forms.MenuItem ();
        this.miExit = new System.Windows.Forms.MenuItem ();
        // mainMenu1
        this.mainMenu1.MenuItems.AddRange (new System.Windows.Forms.MenuItem[]
{
                this.menuItem1});
        // menuItem1
        this.menuItem1.Index = 0;
        this.menuItem1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
                this.miNew,
                this.miOpen,
                this.menuItem4,
```

```
            this.miExit});
        this.menuItem1.Text = "&File";
        // miNew
        this.miNew.Index = 0;
        this.miNew.Text = "&New";
        this.miNew.Click += new System.EventHandler(this.MenuSelect);
        // miOpen
        this.miOpen.Index = 1;
        this.miOpen.Text = "&Open";
        this.miOpen.Click += new System.EventHandler(this.MenuSelect);
        // menuItem4
        this.menuItem4.Index = 2;
        this.menuItem4.Text = "-";
        // miExit
        this.miExit.Index = 3;
        this.miExit.Text = "E&xit";
        this.miExit.Click += new System.EventHandler(this.miExit_Click);
        // Form1
        this.AutoScaleBaseSize = new System.Drawing.Size(8, 18);
        this.ClientSize = new System.Drawing.Size(376, 288);
        this.Menu = this.mainMenu1;
        this.Name = "Form1";
        this.Text = "Form1";
    }
    #endregion
    /// <summary>
    ///应用程序的主入口点。
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.Run(new Form1());
    }
    // 加载新的菜单。
    private void LoadSecondMenu()
    {
        this.Menu = mmNewMenu;
    }
```

```
        private void miNew_Click（object sender, System.EventArgs e）
        {
            MessageBox.Show（"单击了'新建'菜单项.....", "提示框"）；
        }
        private void miOpen_Click（object sender, System.EventArgs e）
        {
            MessageBox.Show（"单击了'打开'菜单项.....", "提示框"）；
        }
        private void miSave_Click（object sender, System.EventArgs e）
        {
            MessageBox.Show（"单击了'保存'菜单项.....", "提示框"）；
        }
        private void miClose_Click（object sender, System.EventArgs e）
        {
            this.Menu = mainMenu1;
        }
        // 响应单击菜单项的事件。
        private void MenuSelect（object sender, System.EventArgs e）
        {
            MessageBox.Show（"单击了一个菜单项.....", "提示框"）；
            LoadSecondMenu（）；
        }
        // 退出应用程序。
        private void miExit_Click（object sender, System.EventArgs e）
        {
            Close（）；
        }
    }
}
```

1.4　总结

应用程序的主菜单提供了良好的操作界面，用户可以通过主菜单使用应用程序提供的丰富的功能。但是随着国际化的加，越来越多的应用程序需要在不同的国家和地区使用，为不同的国家和地区维护不同的代码就显得很费时费事。在 C#中，可以使用动态主菜单的技术来为不同国家和地区的用户提供不同界面的相同的应用程序，这样就大大减少了编程者的负担。本实例仅仅是展示了如何实现动态菜单的功能，读者可以在此基础上添加自己的新功能。

第 2 讲　动态上下文菜单

2.1　实现目标

本实例将制作一个动态的上下文菜单，它能根据当前焦点在不同的组件对象上，从而显示不同的上下文菜单，这样就充分地展示了应用程序的智能性。如图 2-1，图 2-2 所示是应用程序的效果图。

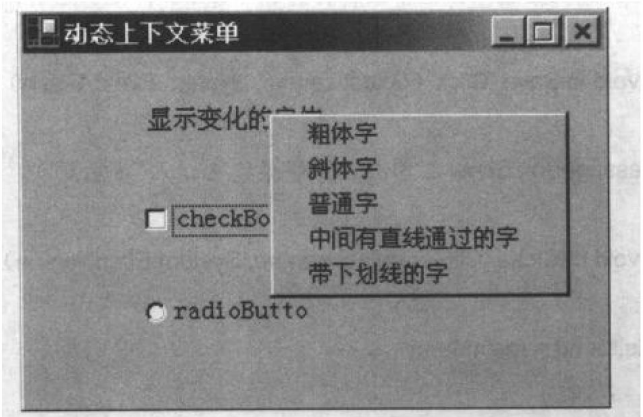

图 2-1　Label 组件的上下文菜单

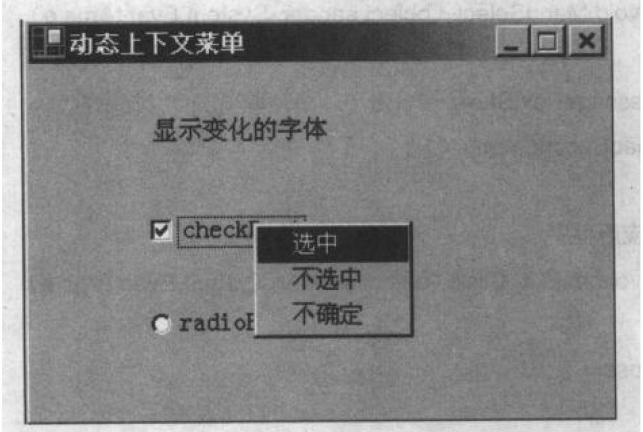

图 2-2　CheckBox 组件的上下文菜单

2.2　背景知识介绍

Windows 应用程序中有很多地方提供上下文菜单，通过上下文菜单，用户可以很方便的操作软件。若要节省创建应用程序所需的时间并减少代码量，可以让多个组件共享单个上下文菜单对象。利用一个只包含该组件必需菜单项的“动态”上下文菜单（或快捷方式菜单），可以减少应用程序中组件所需的上下文菜单总数。下面列举出 ContextMenu 组件常用的属性和方法：

- ContextMenu.Name

该属性说明了一个 ContextMenu 类的实例的名称，在这个程序中就可以直接使用该组件。

- ContextMenu. Popup

该方法说明了 ContextMenu 如何处理右键单击事件。

- CheckBox.Name

该属性说明了一个 CheckBox 类的实例的名称，在这个程序中就可以直接使用该组件。

- CheckBox.ThreeState

该属性说明是否允许 CheckBox 存在三态，即：checked、unchecked、Indeterminate。

- CheckBox.ContextMenu

该属性说明了当该组件有鼠标右键单击事件的时候，使用哪个上下文菜单组件来处理。

注意：RadioButton 和 Label 两个组件的属性设置基本上和 CheckBox 一样，此处就不再列举出来了，读者可以参考后面的代码进行分析。

2.3　实现步骤及技术分析

1. 界面设计

（1）启动 Visual Studio，新建一个 Visual C#项目，然后在“模板”窗格中选择“Windows 应用程序”。

（2）设置组件的属性如下：

```
// ContextMenu 组件的 Popup 事件
contextMenu1.Popup += new System.EventHandler（this.contextMenu1_Popup）;
// Label 组件的属性
label1.Name = "label1";
label1.ContextMenu = this.contextMenu1;
label1.Text = "显示变化的字体";
// CheckBox 组件的属性
checkBox1.Name = "checkBox1";
checkBox1.ContextMenu = this.contextMenu1;
checkBox1.Text = "checkBox1";
checkBox1.ThreeState = true;
// RadioButton 组件的属性
radioButton1.Name = "radioButton1";
radioButton1.ContextMenu = this.contextMenu1;
radioButton1.Text = "radioButton1";
```

2. ContextMenu 的处理过程

将 CheckBox、RadioButton 和 Label 三个组件放到窗体上以后，设置这三个组件的 ContextMenu 属性为 contextMenu1，这样当这三个组件中的某个组件有鼠标右键单击事件发生时，ContextMenu 组件的 Popup 方法就可以对其进行处理了。在该代码中根据当前是哪一个组件产生的事件动态地生成上下文菜单，同时注册对应菜单项的处理函数，其生成的代码如下。

```
private void contextMenu1_Popup（object sender, System.EventArgs e）
```