基于GDI+的颜色配置组件

The Component of Color Configuration Based on GDI+

何海江 （长沙湖南经济管理干部学院计算机系 410004）

摘 要：介绍了一种基于GDI+的颜色配置组件的开发技术。组件包括模态和非模态两个窗体，其中非模态窗体可同时配置绘图时需要的前景色、背景色、边框色和透明度值。该组件可重用性强。

关键词：GDI+ C# 颜色 组件

GDI+是微软的.NET Framework类库，用于图形编程。在微软以前的开发工具中，使用GDI来进行图形编程，相当复杂，而GDI+是GDI的包装器，大大简化了在此基础上的图形程序设计任务。笔者从事的基于.NET的组态软件工程中，需要开发一个绘图软件，绘制线、矩形、圆和多边形等普通图形，而这些图形的前景色、背景色、边框色和透明度值能方便修改。在工作过程中，编写了一个这样的组件，将复杂的程序设计技巧封装，操作简单而可靠性高。下面将颜色配置组件（后文称为ColorCfg）的关键技术逐一介绍。

1 组件的功能

ColorCfg中有两个窗体，一为非模态(如图1所示)，二为模态(如图2所示)。

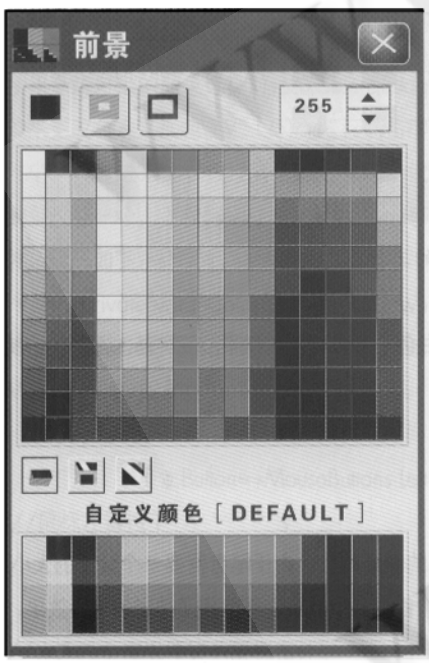

图 1 非模态窗体

非模态窗体功能包括：

（1）前景色、背景色、边框色对应三个模拟按钮，都可在窗体中16*16个颜色中自由选择。

（2）上部分12*16为固定颜色，根据RGB值均匀分布，大致能反映所有颜色。下部分4*16为自定义颜色，可通过两种方式修改：双击，出现操作系统带的颜色对话框，修改自定义颜色中某一个；窗体中三个图形按钮分别为打开调色板文件、保存调色板文件、将调色板重置为缺省颜色值，通过这三个按钮可一次修改所有的4*16自定义颜色。三个图形按钮都带有Tooltip的悬停提示。

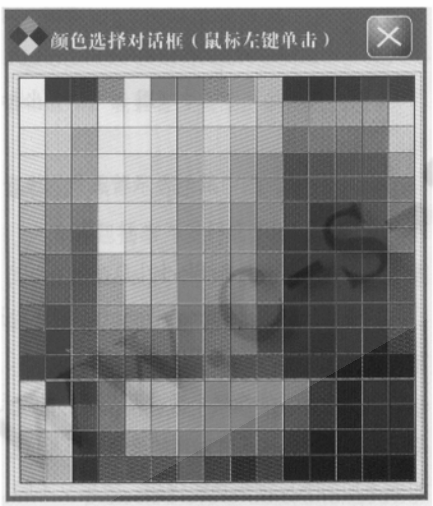

图 2 模态窗体

（3）提供透明度值（后文称为Alpha）来美化图形，该值的修改使用VS.NET开发环境中的NumericUpdown控件实现。

（4）订阅事件，当非模态窗体中上述四个值变化后，可实时动态地更新用户所绘制的图形外观。

模态窗体只有一个功能，从固定颜色和自定义颜色中选定一个颜色，类似于系统自带的颜色选择对话框。绘图软件中许多情况下需要这样模态的颜色选择窗体。

2 组件结构

ColorCfg是一个Windows类库，命名空间为ColorSelect，全部使用C#语言开发，包含四个文件：

（1）Color_Define.cs：定义固定颜色和自定义颜色，并提供颜色的查询功能，成员皆为静态，该对象一直处于内存中且无需实例。关键代码及其注释如下：

```
internal class RefOfColor
    {
        public static readonly Color[] FixRef;//固定颜色值.
        private static readonly Color[] DefaultRef;//缺省的自定义颜色值.
        public static Color[] DefRef;//自定义的颜色.
        static RefOfColor() //构造器，初始化所有成员变量。
            //将自定义的颜色重置为缺省的自定义颜色。
        public static void ResetToDefault()
        /* 从行和列取对应的颜色值。16*16个颜色值 row=0...15 col =0...15*/
        public static Color GetColorFrom16_16(int row,int col)
```

public static bool GetRowColFromColor(Color cor,out int row,out int col)//从颜色值取到对应的行和列。找到cor，返回true，否则返回false。

/*从调色板文件中读出自定义颜色。读成功返回true,否则false*/

public static bool ReadFromPalFile(string filename)

/*向调色板文件中写入自定义颜色。成功返回true,否则false。将配置好的颜色保存到一个文件，通过“打开调色板文件”按钮调用ReadFromPalFile 随时读出。*/

public static bool WriteToPalFile(string filename)

}

（2）ColorModalSelect.cs：实现模态窗体。该对象从System.Windows.Forms.Form继承，包含两个属性：当前选定的颜色CorCurrent，所选颜色是否变化BcolorIsChanged（布尔量），该值为真时，才需要做进一步的更新操作。

（3）ColorCfg.cs：实现非模态窗体。该对象从System.Windows.Forms.Form继承，关键代码文章中3、4、5节介绍。使用该对象的父进程只需要一个ColorCfg实例，一旦创建后不会消除，除非父进程显示使用其Dispose方法从内存中卸载。实现方法是重载其Closing事件，添加如下代码：

```
Hide();
e.Cancel = true;
```

这样当按下关闭窗口的按钮或ALT+F4，只隐藏窗口，不删除窗体对象。

（4）ColorChangedEvent.cs：事件类型属性类，第6节事件中用到，关键代码及其注释如下：

```
public enum WhichChangedType//定义事件枚举类型
{
    FrontChanged = 1,//前景色改变了
    BackChanged,    //背景色改变了
    FrameChanged,   //边框色改变了
    AlphaChanged    //透明度值改变了
}
public delegate void ColorChangedCallback(object sender, ColorChangedEventArgs E);
//事件代理
public class ColorChangedEventArgs//事件参数类，从System.EventArgs继承，只有一个事件枚举类型的属性，当上述四种事件发生后，事件订阅者知道是何种事件。
```

3 模拟按钮的实现

如图1左上角三个模拟按钮对应前景色、背景色和边框色，任意时刻一个按钮按下，另两个弹起，主要用到Graphics类（GDI+的绘图模板，对应GDI的CDC）的三个方法：画线DrawLine、画空心矩形DrawRectangle和画实心矩形FillRectangle。

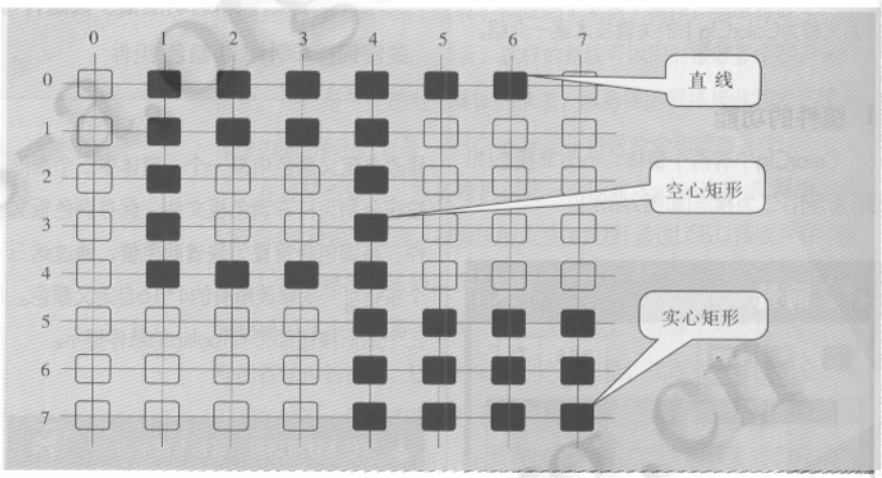

图3 8*8坐标下的三种基本图形

如图3所示，图中黑色表示着黑色的像素，白色表示未着色的像素。三种基本图形的绘制方法分别为：

直线DrawLine(Pens.Black,1,0,6,0)

空心矩形DrawRectangle(Pens.Black,1,1,3,3)

实心矩形FillRectangle(Pens.Black,4,5,4,3)

请注意直线的像素个数，两个矩形的宽和高的像素个数。由于笔者需要一个像素的大小，每个方法中坐标和尺寸对应的像素大不相同。整个组件开发过程中都使用了缺省的锯齿效果，如果添加如下反锯齿功能代码，效果又不同。

Graphics.SmoothingMode = SmoothingMode.AntiAlias

模拟按钮的外观关键在边框如何突出阴影效果，使用白、灰和浅灰三种颜色就能实现。下面的代码为按下状态的画边框部分，其中g为Graphics类的实例，start为按钮的左上角坐标，按钮宽和高皆为Len。

```
g.FillRectangle( new SolidBrush(this.BackColor),new Rectangle(start,new Size(Len,Len)));//先清除原来弹起状态留下的垃圾。
g.DrawLine(Pens.White,start+new Size(Len,0),start+new Size(Len,Len));
g.DrawLine(Pens.White,start+new Size(0,Len),start+new Size(Len,Len));
g.DrawLine(Pens.Gray,start,start+new Size(Len,0));
g.DrawLine(Pens.Gray,start,start+new Size(0,Len));
```

如何识别用户点击的是固定颜色、自定义颜色，还是三个模拟按钮，根据这16*16+3个矩形的坐标和宽和高来识别。如某一个矩形的左上角坐标为(left,top),宽为width,高为height,则构造此矩形rect = new Rectangle(new Point(left,top),new Size(width, height)),使用方法rect.Contains(new Point(x,y))判断鼠标点击时的坐标（x,y）是否落在该矩形内。当用户点击某一模拟按钮，该按钮对应的颜色用白色框突出显示，用户点击16*16细小颜色框，按下的模拟按钮的颜色则相应变化。

三个模拟按钮的当前颜色值使用三个属性记录，分别为CorFront、CorBackground和CorFrame，皆具有get和set存取功能。

4 鼠标双击

鼠标左键双击4*16个自定义颜色框，可使用系统自带的颜色选择对话框ColorDialog类来更改自定义颜色。VS.NET的鼠标双击事件中，传递进来的参数没有鼠标坐标，要识别更改哪一个，需要特别处理MouseDown事件，代码如下：

```
private void ColorCfg_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
    {
        if( e.Button==MouseButtons.Left )//鼠标左键按下
        {
            ProcessFrameMouseDown(e.X,e.Y); //处理三个模拟按钮的鼠标事件。
            ProcessColorMouseDown(e.X,e.Y);//处理16*16颜色框的鼠标事件。
            bLeftDown = true;//设标志，鼠标双击中要用到。
            iMouseX = e.X; iMouseY = e.Y; //记录鼠标点击时的位置。
        }
        else //鼠标其他键的单击则清除前面的标志。
            bLeftDown = false;
    }
private void ColorCfg_DoubleClick(object sender, System.EventArgs e)
    {
        if( !bLeftDown )//双击前没有鼠标左键按下标志，则不是左键双击。
            return;
        //使用第三节介绍的方法识别在哪一个自定义颜色框中双击，如果找到，则使用ColorDialog 类来设置新的自定义颜色。
        （代码略.....）
    }
```

5 Alpha 值的使用

非模态窗体的三个颜色属性都只有RGB分量，没有Alpha分量。以填充刷子为例，在一个使用该组件的父进程中，这样来使用Alpha值：

```
SolidBrush brush=new SolidBrush(Color.FromArgb(cfg.AlphaOfFront , cfg.CorFront));
```

其中cfg为类ColorSelect.ColorCfg的实例，AlphaOfFront 是对应Alpha值的属性。加入Alpha值后，能显著改善图形的美观程度。

6 事件

事件包括事件发行者(Publisher)和事件订阅者(Subscriber)两个角色。非模态窗体作为事件发行者包括三项工作：

（1）定义delegate类型，在第二节ColorChangedEvent.cs中已有介绍。

（2）定义事件。

```
protected event ColorChangedCallback ColorChangedDone=null;
```

（3）触发事件，以透明度值变化为例。

```
private void alphaUpDown_ValueChanged(object sender, System.EventArgs e)
    {
        AlphaOfFront = Decimal.ToInt32(alphaUpDown.Value);
        OnColorChanged( WhichChangedType.AlphaChanged );
    }
```

其他三个颜色属性的变化产生类似的事件，只有事件参数不同。为增强组件的功能，另提供下面的方法ReSetupConfig，父进程可调用该方法来修改这些属性。为保证用户不会误操作，增加一个私有成员：事件是否执行标志，代码 "private bool bResetupNoEvent=false; "，这样当父进程调用ReSetupConfig修改颜色和Alpha值四个属性时不触发事件。

```
public void ReSetupConfig(int alpha,Color forColor,Color backColor,
    Color frameColor)
    {
    bResetupNoEvent = true;//不触发事件。
    CorFront = forColor;
        CorBackground = backColor;
            .....
    }
```

OnColorChanged方法中包装了事件的触发。

```
protected virtual void OnColorChanged(WhichChangedType type)
    {
     if( ColorChangedDone==null)
        return;
     if( bResetupNoEvent )//当执行ReSetupConfig时不触发事件。
        return;
        ColorChangedDone(this,new ColorChangedEventArgs(type));
        }
```

另外，为方便父进程关联和移除事件处理程序，增加两个包装 "+="和 "-="的成员。

```
public void AddOnColorChangedDone(ColorChangedCallback handler)
    {
    if( this.ColorChangedDone==null )
        this.ColorChangedDone = handler;
        else
         this.ColorChangedDone += handler;
    }
    public void RemoveOnColorChanged
```

```
Done(ColorChangedCallback handler)
        {
    if( this.ColorChangedDone==null )
                return;
        this.ColorChangedDone -= handler;
        }
```

使用非模态窗体的父进程作为事件的订阅者，包括两项工作：

① 编写事件处理过程。

```
private void HandleColorChange(object
sender, ColorSelect.ColorChangedEventArgs E)
        {
            ......
    }
```

② 订阅事件。

```
cfg.AddOnColorChangedDone(new
ColorSelect.ColorChangedCallback
(HandleColorChange));
```

事件处理过程可以有多个，方法AddOnColorChangedDone将父进程的事件处理过程和事件关联，方法RemoveOnColorChangedDone从事件中移除已经关联的事件处理过程。父进程将那些颜色和透明度值变化后需要做的事情写进事件处理过程，每当四个属性变化后，父进程规定的任务自动执行，不需要其他同步处理技术。

7 结束语

ColorCfg组件经过严格测试，在使用过程中一直稳定可靠，可在其他使用COM组件和.NET组件的绘图程序中使用。需要详细代码的读者可与笔者联系。

参考文献

1 Eric White著，杨浩、张哲峰译，GDI+程序设计，清华大学出版社，2002。

2 Microsoft·Microsoft Visual Studio.NET Msdn Library，2003。

3 孙三才、张智凯、许薰尹著，C#与.NET，中国青年出版社，2002。