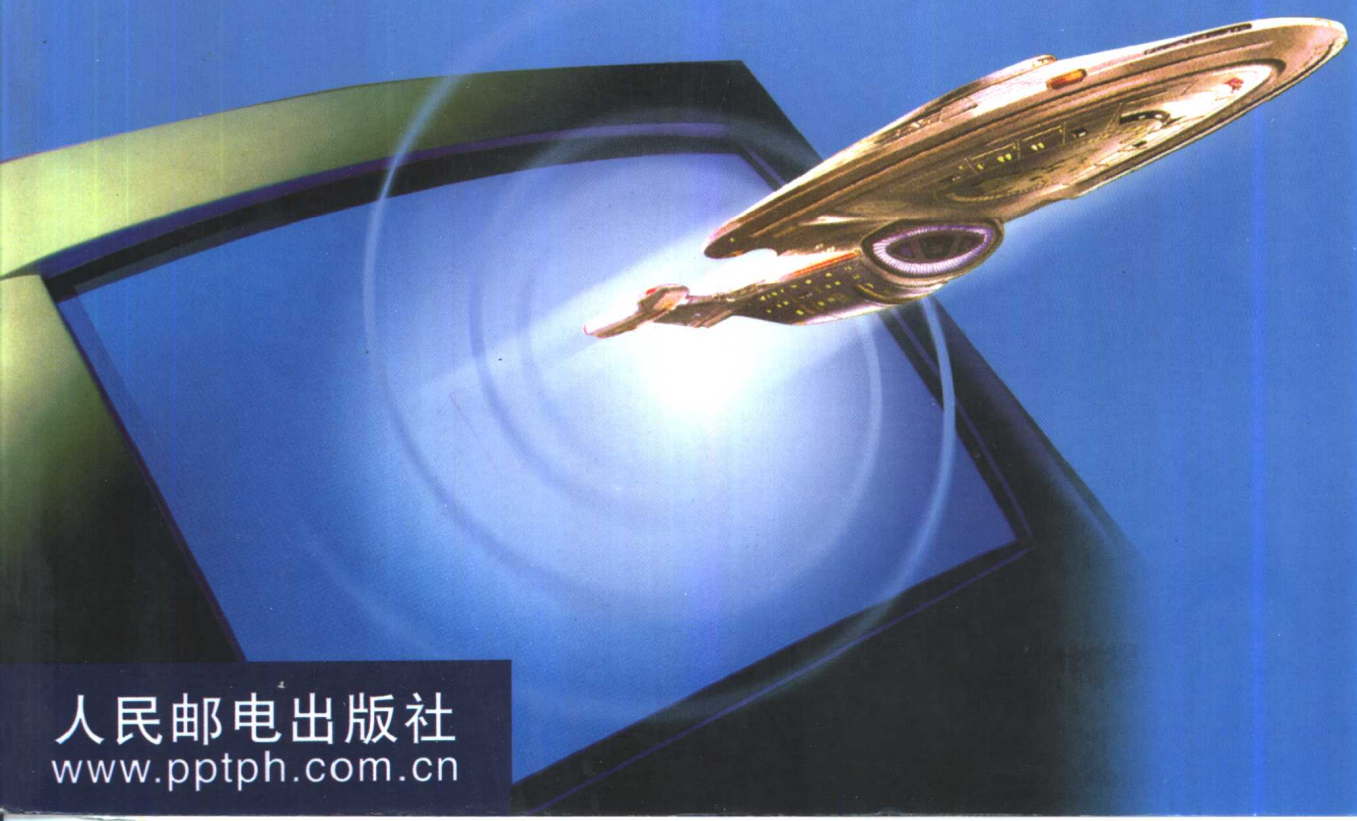
特效视窗——

Visual C++

开发高级界面实例

冯峰　王雪梅　等编著



人民邮电出版社
www.pptph.com.cn

特效视窗

——Visual C++开发高级界面实例

冯　峰　王雪梅　等编著

人 民 邮 电 出 版 社

图书在版编目(CIP)数据

特效视窗：Visual C++开发高级界面实例/冯峰等编著.北京：人民邮电出版社，2000.9
ISBN 7-115-08716-4

I.特… II.冯… III.C 语言-程序设计 IV.TP312

中国版本图书馆 CIP 数据核字(2000)第 42230 号

特效视窗——Visual C++开发高级界面实例

◆ 编　　著　冯　峰　王雪梅　等
责任编辑　王晓明
◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@pptph.com.cn
网址　http://www.pptph.com.cn
北京汉魂图文设计有限公司制作
北京顺义向阳胶印厂印刷
新华书店总店北京发行所经销
◆ 开本：787×1092　1/16
印张：26.5
字数：666 千字　　2000 年 11 月第 1 版
印数：1－4 000 册　　2000 年 11 月北京第 1 次印刷
ISBN 7-115-08716-4/TP·1772

定价：39.00 元

内 容 提 要

由于 Visual C++具有优秀的界面开发特性，因此使用该工具进行高级界面开发已成为编程工作中一个重要的方面。本书主要介绍使用 Visual C++进行应用程序界面开发的方法与技巧。

本书介绍了 7 个综合实例，分别是：实现颜色渐变的 CProgressCtrl 控件、实现在 Windows 任务栏中的动画图标、实现界面上的高级位图操作、高级派生控件的组合使用、OutLook 窗口界面的创建、带有图像的菜单、椭圆形的 CToolTip 控件。这些实例包含了许多非常有用的界面开发内容，可以帮助读者掌握 Viusal C++中的界面高级组合效果的创建技术。

本书可供 Visaul C++编程技术人员在设计开发应用程序界面时参考，也可供刚入门的编程新手在熟悉编程方法时学习使用。

前　言

Microsoft Windows 已经广泛地被人们所接受，人们越来越多地体会到图形界面(GUI)带来的巨大好处。随着 Windows 98 的正式推出，微软公司随即又推出了基于 Windows 98 操作平台的强大编程工具软件 Microsoft Visual C++ 6.0。

Visual C++包含完整的 Windows 应用开发系统。如果人们愿意，仍然可以使用 Windows SDK 所提供的 API 来开发用 C 语言编写的 Windows 程序。Windows SDK 程序设计技术已经广为人知，并且已经有许多书籍对此专门作了介绍。同时人们还可利用 Visual C++所提供的一些新的工具使 Windows 应用程序的编写变得更加方便。

Internet 的流行同样也影响了 Visual C++和 MFC。在 Visual C++ 4.0 的版本里，引进了对 Internet 编程的支持并设计了新的类库。Visual C++ 5.0 又增加了一些新类，对产品的界面也做了很多改动。

1998 年，Microsoft 公司推出的 Visual C++ 6.0，对 Visual C++ 5.0 的界面继续进行了改善。但是，MFC 类库中的界面仍然满足不了广大用户的需求。所以，程序员不得不自己开发特殊环境下的界面。同时，Visual C++在其窗体类和窗体派生类中都保留了应有的虚函数来适应不同的程序员需求。

本书介绍的不是如何用 Visual C++来进行 Windows 程序设计，而是介绍如何用 Visual C++所提供的 MFC 类库来创建特殊的界面效果。这是当前 Windows 系统下编程的潮流和趋势。而且，在使用 Microsoft 基本类库进行编程的同时也使用了 Windows SDK 的界面、资源和消息函数，并将 MFC 类库和 Windows SDK 函数进行了完美的结合。

本书主要由冯峰、王雪梅编写，参加本书编写的还有：王剑峰、冯静、王咏梅、宁宪东、王玮宸、宁子巍、梁俊、刘强、李伟、周晓昱、梁晖、王秀丽、牛鑫瑞、岳敏、李佳音、程卫国、刘艺、赵宇明、秦爱德、李腾、徐昕、朱亚平、刘文国、王爱民、曾东等，他们都为本书的编写做出了很大的贡献。

由于计算机技术发展迅速，加上作者自身水平有限，书中错误在所难免，请读者批评指正。

作　者

2000 年 1 月

目　录

实例1 实现颜色渐变的CProgressCtrl控件

实 例 简 介

? 实例提要

在 Visual C++中可以通过 MFC AppWizard 很方便地实现一个颜色渐变的 CProgressCtrl 控件和颜色井控件。本实例首先通过 MFC AppWizard 创建一个 MFC 应用程序框架，在此基础上通过添加相应的变量和函数，完成一个颜色井控件，并在该控件的基础上，实现颜色渐变的 CProgressCtrl 控件的创建演示程序。程序的运行结果如图 1-1 和图 1-2 所示。

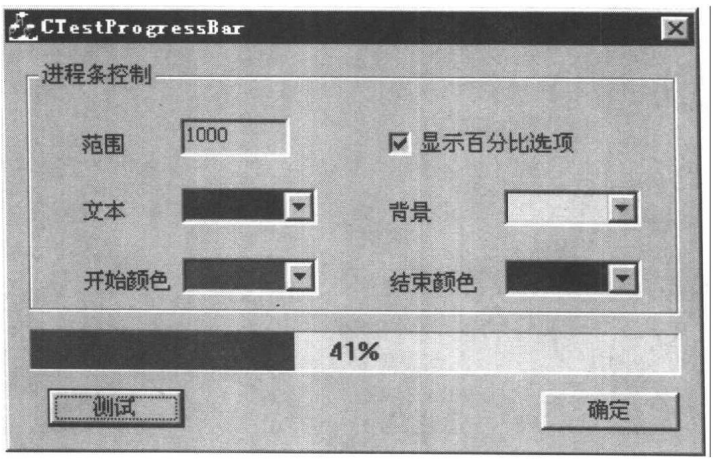


图 1-1 颜色渐变的 CProgressCtrl 控件

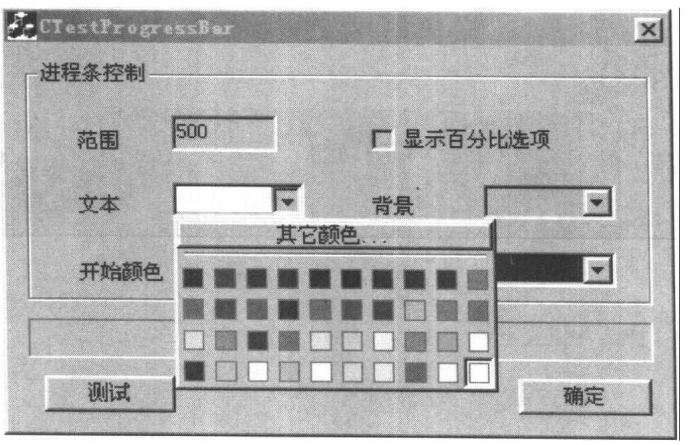


图 1-2 颜色井的弹出对话框

? 技术概要

本实例使用的主要技术如下：

① 使用设备上下文（Device Context，简称 CDC）：CDC 类定义了一类设备环境对象。CDC 对象提供的成员函数和一个设备环境一起工作，例如显示器或打印机，就像成员函数和与窗口客户区域相联系的显示环境一起工作一样。通过 CDC 的成员函数来处理所有的绘画工作。该类为设备环境提供了许多成员函数，这些函数使用通过图形设备接口（GDI）选入的绘画工具、颜色以及调色板。它还提供了许多成员函数用于获取或设置绘画属性、映射方式、视区原点、窗口宽度、比例尺、区域、剪辑、画线、画简单形状、画椭圆以及画多边形。这些成员函数提供了不同字体的文本绘制、脱机使用打印机、滚动、显示图元文件，而这些都是 CDC 类中的基本操作。在本实例中通过结合该类中的基本操作和基类函数来完成与内存设备上下文的对应。

② 由 CButton 类派生并创建特色的颜色井：CButton 类在 Windows 应用程序中的使用十分广泛，通过以 CButton 类为基类派生出一个富有深刻含义的颜色井可以使读者对 MFC 类库有更进一步的了解。读者还可以在该类的基础上派生出效果更强的派生类。

③ 由 CWnd 类派生出颜色板：在 Windows MFC 类库中使用 CWnd 类可为基类派生出如图 1-2 所示的颜色板。

④ 在进程条基类（CProgressCtrl）中添加颜色渐变的效果：在 Windows 应用程序中所见到的进程条控件有两种（对于使用 Visual C++的人员而言），一种是 MFC 类库中的 CProgressCtrl 类，另一种就是派生出来的进程条控件，其形式如图 1-1 所示中的进程条，但其中没有颜色的变化。本实例将涉及以上的所有内容，以这些内容为基础完成一个效果极强的 CprogressCtrl 派生类。

⑤ 在 CProgressCtrl 类中添加进程的百分比显示功能。

步骤之一——实现颜色井

以下的内容将介绍颜色井的创建，主要的添加类是以 CButton 类为基类和以 CWnd 类为基类的两个派生类。在这两个派生类中封装了实现该功能的完全代码，它不仅在教学上是一个很好的实例，而且在工程应用上也是一个很有应用前景的特效界面控制方案。下面，将讨论添加颜色井的步骤。

☛ 建立一个新项目 CTestProgressBar

选择 File 菜单中的 New 命令，在弹出的 New 对话框中选择 Project 标签页，然后选择列表中的 MFC AppWizard (exe)，在 Project name 编辑框中输入工程名 CTestProgressBar。在第一步骤（Step1）中选择 Dialog based 选项，如图 1-3 所示。

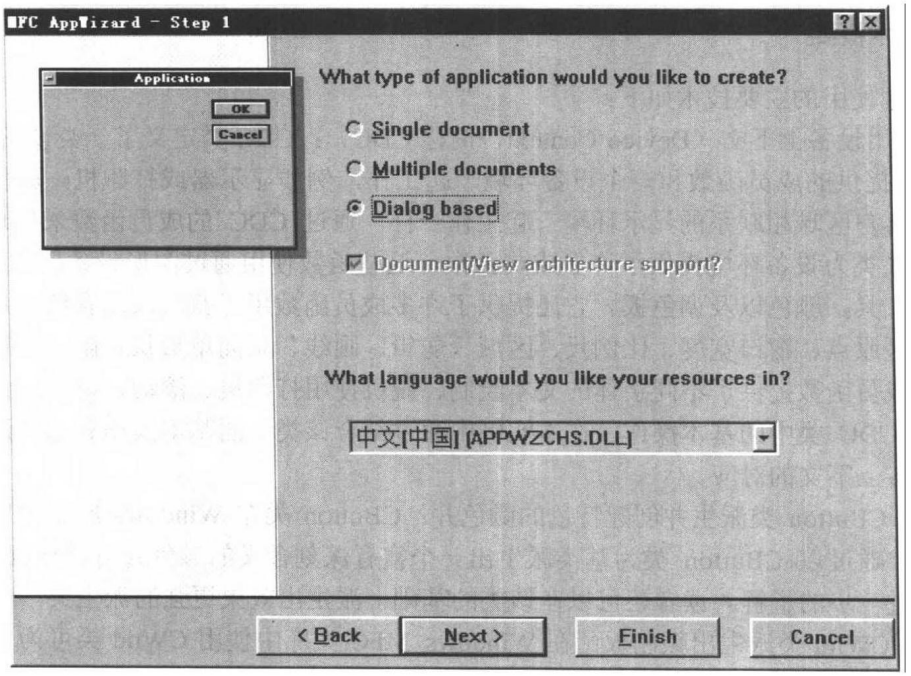


图 1-3　创建基于对话框的项目 CTestProgressBar

按下 Finish 按钮，AppWizard 将自动创建其它相关类，其相应的信息如下：

Application type of CTestProgressBar:

Dialog-Based Application targeting:

Win32

Classes to be created:

Application: CCTestProgressBarApp in CTestProgressBar.h and CTestProgressBar.cpp

Dialog: CCTestProgressBarDlg in CTestProgressBarDlg.h and CTestProgressBarDlg.cpp

Features:

+ About box on system menu
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:

中文[中国]

☛ **添加新类 CColourPopup**

类 CColourPopup 的主要功能是实现如图 1-4 所示的颜色板。

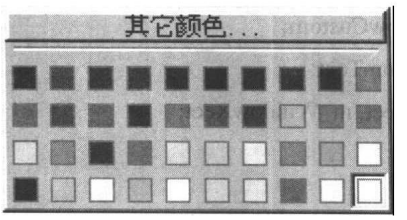


图 1-4　由 CWnd 类派生出来的特效颜色板

首先，选择 Insert 菜单中的 New Class 命令，来添加新类 CColourPopup，如图 1-5 所示。

类名：CColourPopup

基类：generic CWnd

文件：ColourPopup.h 和 ColourPopup.cpp

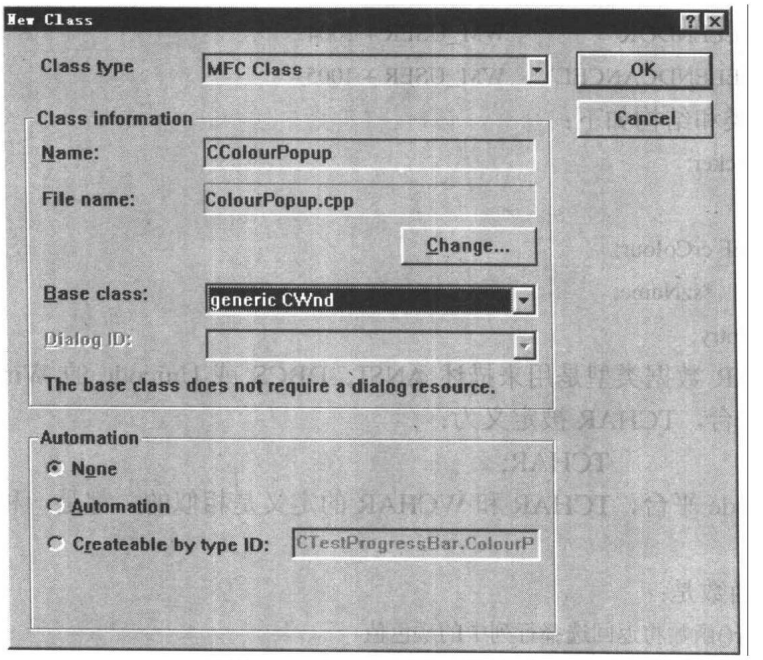


图 1-5　添加新类 CColourPopup 的对话框

☛ 添加和修改代码

1. 添加成员变量

在 ColourPopup.h 头文件中添加如下成员变量：

```
protected:
    static      ColourTableEntry m_crColours[];
    int         m_nNumColours;
    int         m_nNumColumns, m_nNumRows;   //设定颜色板的行数和列数
    int         m_nBoxSize, m_nMargin;
    int         m_nCurrentRow, m_nCurrentCol;
    int         m_nChosenColourRow, m_nChosenColourCol;
```

```
    BOOL            m_bShowCustom;
    CString         m_strCustomText;
    CRect           m_TextRect, m_WindowRect;
    CFont           m_Font;
    CPalette        m_Palette;
COLORREF        m_crInitialColour, m_crColour;
//使用 CToolTipCtrl 控件
    CToolTipCtrl    m_ToolTip;
    CWnd*           m_pParent;
```

2. 在 ColourPopup.h 头文件中添加两个嵌入函数及定义的变量

添加的变量如下：

```
#define CPN_SELCHANGE       WM_USER + 1001
#define CPN_SELENDOK        WM_USER + 1004
#define CPN_SELENDCANCEL    WM_USER + 1005
```

添加定义的类和结构如下：

```
class CColourPicker;
typedef struct {
    COLORREF crColour;
    TCHAR    *szName;
} ColourTableEntry;
```

其中，TCHAR 数据类型是用来描述 ANSI、DBCS 或 Unicode 的 Win32 字符串。对于 ANSI 和 DBCS 平台，TCHAR 被定义为：

```
typedef char            TCHAR;
```

而对于 Unicode 平台，TCHAR 和 WCHAR 的定义是相似的，都是一种 16 位的 Unicode 编码。

添加的嵌入函数是：

```
    //GetColour()函数将返回选择行列中的颜色值
COLORREF GetColour(int row, int col) { return m_crColours[row*m_nNumColumns + col].crColour; }
//GetColourName()函数返回由 CToolTipCtrl 控件显示的颜色说明文字
LPCTSTR GetColourName(int row, int col)
    {
        return m_crColours[row*m_nNumColumns+ col].szName;
    }
```

3. 添加 Initialize()函数

该函数用于初始化弹出颜色板的初始窗体界面。代码如下：

```
void CColourPopup::Initialize()
{
    m_nNumColours       = sizeof(m_crColours)/sizeof(ColourTableEntry);
    ASSERT(m_nNumColours <= MAX_COLOURS);
    if (m_nNumColours > MAX_COLOURS)
```

```
        m_nNumColours = MAX_COLOURS;
    m_nNumColumns          = 0;
m_nNumRows             = 0;
//显示的颜色井中各个颜色框的大小
    m_nBoxSize             = 18;
    m_nMargin              = ::GetSystemMetrics(SM_CXEDGE);
    m_nCurrentRow          = -1;
    m_nCurrentCol          = -1;
    m_nChosenColourRow     = -1;
    m_nChosenColourCol     = -1;
    m_strCustomText        = _T("其它颜色...");
    m_bShowCustom          = TRUE;
    m_pParent              = NULL;
m_crColour             = m_crInitialColour = RGB(0,0,0);
//
//设定了颜色板中颜色区域的面积，其最小值为 7+2×m_nMargin
if (m_nBoxSize - 2*m_nMargin - 2 < 5) m_nBoxSize = 5 + 2*m_nMargin + 2;
//
//在创建的颜色板中使用非客户区的字体
//以下代码创建了 NONCLIENTMETRICS 结构，要了解详细情况请查阅 MSDN 在线帮助
NONCLIENTMETRICS ncm;
    ncm.cbSize = sizeof(NONCLIENTMETRICS);
VERIFY(SystemParametersInfo(SPI_GETNONCLIENTMETRICS, sizeof(NONCLIENTMETRICS),
                                &ncm, 0));
    //通过得到系统的非客户区信息，创建一种与 MessageBox 中字体相同的字体
    m_Font.CreateFontIndirect(&(ncm.lfMessageFont));
    //创建应用颜色板结构
struct
{
    //逻辑颜色板
        LOGPALETTE     LogPalette;
        PALETTEENTRY   PalEntry[MAX_COLOURS];
    } pal;
    //
    //颜色板的入口数为定义的颜色数
    pLogPalette->palNumEntries = m_nNumColours;
    //
    LOGPALETTE* pLogPalette = (LOGPALETTE*) &pal;
    for (int i = 0; i < m_nNumColours; i++)
{
```

```
//以下的三个函数将分别得到和赋予逻辑颜色板的三种颜色值
        pLogPalette->palPalEntry[i].peRed   = GetRValue(m_crColours[i].crColour);
        pLogPalette->palPalEntry[i].peGreen = GetGValue(m_crColours[i].crColour);
        pLogPalette->palPalEntry[i].peBlue  = GetBValue(m_crColours[i].crColour);
        pLogPalette->palPalEntry[i].peFlags = 0;
}
//以创建的逻辑颜色板为模板来生成 MFC 的 CPalette 类
    m_Palette.CreatePalette(pLogPalette);
}
```

在该函数中涉及的结构较多，比较生疏的是以下的几个结构，下面分别介绍一下。

其中的 LOGPALETTE 逻辑颜色板结构定义如下：

```
typedef struct tagLOGPALETTE { // lgpl
    WORD        palVersion;
    WORD        palNumEntries;
    PALETTEENTRY palPalEntry[1];
} LOGPALETTE;
```

参数：

palVersion：描述了系统的版本号。

palNumEntries：指示了逻辑颜色板的入口个数。

palPalEntry：定义了 PALETTEENTRY 结构的数组。

PALETTEENTRY 结构的定义如下：

```
typedef struct tagPALETTEENTRY
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

PALETTEENTRY 结构的定义如下：

```
typedef struct tagPALETTEENTRY
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

其中：peRed、peGreen 、peBlue 分别定义了红、绿和蓝色的数值。

对 peFlags 参数请参阅帮助文件。

4. 编辑 CColourPopup 构造函数和析构函数

函数代码如下：

```
CColourPopup::CColourPopup()
{
```

```
    //构造函数仅仅调用 Initialize()函数
Initialize();
}
CColourPopup::~CColourPopup()
{
//在析构函数中删除系统中动态产生的对象，以避免内存的泄漏
m_Font.DeleteObject();
m_Palette.DeleteObject();
}
```

5. 添加 CColourPopup 重载的构造函数

函数代码如下：

```
CColourPopup::CColourPopup(CPoint p, COLORREF crColour, CWnd* pParentWnd, UINT nID,
                           LPCTSTR szCustomText /* = NULL */)
{
     //调用初始化函数
    Initialize();
    m_crColour = m_crInitialColour = crColour;
    if (szCustomText != NULL)
    {
        m_bShowCustom     = TRUE;
        m_strCustomText = szCustomText;
}
else
        m_bShowCustom = FALSE;
        m_pParent = pParentWnd;
    CColourPopup::Create(p, crColour, pParentWnd, nID, szCustomText);
}
```

在 ColourPopup.h 头文件中添加重载构造函数的原型：

```
public:
    CColourPopup(CPoint p, COLORREF crColour, CWnd* pParentWnd, UINT nID,
                 LPCTSTR szCustomText = NULL);
```

6. 添加 Create()函数

函数代码如下：

```
BOOL CColourPopup::Create(CPoint p, COLORREF crColour, CWnd *pParentWnd, UINT nID, LPCTSTR
szCustomText)
{
    ASSERT(pParentWnd && ::IsWindow(pParentWnd->GetSafeHwnd()));
    //检验该类的父类是否为 CColourPicker
    ASSERT(pParentWnd->IsKindOf(RUNTIME_CLASS(CColourPicker)));
    m_pParent   = pParentWnd;
```

```
    m_crColour = m_crInitialColour = crColour;
    // 创建窗体并注册类名
Cstring szClassName =
        AfxRegisterWndClass(CS_CLASSDC|CS_SAVEBITS|CS_HREDRAW|CS_VREDRAW,
                0, (HBRUSH)GetStockObject(LTGRAY_BRUSH),0);
if (!CWnd::CreateEx(0, szClassName,_T(""), WS_VISIBLE|WS_POPUP,
                        p.x, p.y, 100, 100,
                        pParentWnd->GetSafeHwnd(), 0, NULL))
        return FALSE;
    //
    if (szCustomText != NULL)
        m_strCustomText = szCustomText;
    //设定窗体的尺寸
    SetWindowSize();
    // 创建 ToolTips 显示文字
    CreateToolTips();
// 由初始颜色值来确定相应的颜色单元
// 主要功能由 FindCellFromColour()函数完成
    FindCellFromColour(crColour);
    SetCapture();
return TRUE;
}
```

7. 添加 ChangeSelection()函数

函数代码如下：

```
void CColourPopup::ChangeSelection(int row, int col)
{
    CClientDC dc(this);
    //绘制由颜色值得到的颜色选项
    if ((m_nCurrentRow >= 0 && m_nCurrentRow < m_nNumRows &&
        m_nCurrentCol >= 0 && m_nCurrentCol < m_nNumColumns) ||
        (m_nCurrentCol == TEXT_BOX_VALUE))          //TEXT_BOX_VALUE = -2
    {
        int OldRow = m_nCurrentRow;
        int OldCol = m_nCurrentCol;
        m_nCurrentRow = m_nCurrentCol = -1;
        //绘制颜色单元
        DrawCell(&dc, OldRow, OldCol);
    }
    //绘制选中的颜色单元
    m_nCurrentRow = row; m_nCurrentCol = col;
```

```
DrawCell(&dc, m_nCurrentRow, m_nCurrentCol);
    // 将选中的颜色单元保存
if (m_nCurrentRow == TEXT_BOX_VALUE && m_nCurrentCol == TEXT_BOX_VALUE)
        //如果鼠标未选中颜色
        //则返回初始颜色值
        //调用 CColourPicker 类中的自定义消息响应函数
        m_pParent->SendMessage(CPN_SELCHANGE, (WPARAM) m_crInitialColour, 0);
    else
{
    //得到选中的颜色值
        m_crColour = GetColour(m_nCurrentRow, m_nCurrentCol);
        m_pParent->SendMessage(CPN_SELCHANGE, (WPARAM) m_crColour, 0);
    }
}
```

8. 添加 CreateToolTips()函数

该函数的功能是实现如图 1-6 所示的红色显示文字。其函数代码如下：

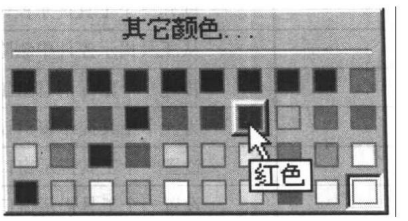


图 1-6　实现的 ToolTips 功能

```
void CColourPopup::CreateToolTips()
{
    if (!m_ToolTip.Create(this)) return;
    // 为每个颜色选项设定一个 ToolTips 显示文字
    for (int row = 0; row < m_nNumRows; row++)
        for (int col = 0; col < m_nNumColumns; col++)
        {
            CRect rect;
            if (!GetCellRect(row, col, rect)) continue;
            //将颜色名得到并添加到相应的 ToolTips 显示文字中
            m_ToolTip.AddTool(this, GetColourName(row, col), rect, 1);
        }
}
```

由于在该函数中使用了 CtoolTipCtrl 类，因此下面将对该类介绍一下。

CToolTipCtrl 类的派生结构如图 1-7 所示。

CToolTipCtrl 类封装了“工具提示并通过文本显示”的性能。CToolTipCtrl 控件是一个弹出窗口，它用来显示一行描述应用程序中某个工具目的的文本。