MFC 中四种常用绘图类的用法探究

软件工程 II 大作业　参考题目 4.15

在 MFC 中类 CPen、CColorDialog、CFontDialog、

CPrintDialog 的用法

作者：程嘉梁

学号：2016013225

手机号：15652667706

Email：chengjl16@163.com

日期：2017/05/25

目录

一、MFC 中 CPen 类的介绍及用法

1、CPen 类的简介

CPen 画笔是一种用来画线及绘制有形边框的工具，用户可以指定它的颜色及厚度，并且可以根据用途与个人喜好绘制不同形状的线。

CPen 类，该类封装了 Windows 图形设备接口（GDI）画笔，主要通过构造函数来创建绘图对象。

CPen 类的继承层次结构为 CObject->CGdiObject->Cpen。

使用 CPen 类时需调用头文件 afxwin.h。

2、CPen 类的成员简述

名称	功能描述
CPen::CPen	构造一个 CPen 对象。
CPen::CreatePen	创建具有指定样式、宽度和画笔属性的画笔，并将其赋给 CPen 对象。
CPen::CreatePenIndirect	创建具有 LOGPEN 结构中给出的样式、宽度和颜色的画笔，并将其赋给 CPen 对象。
CPen::FromHandle	CPen 给定一个窗口时,返回一个指向对象的指针 HPEN。
CPen::GetExtLogPen	获得 EXTLOGPEN 的底层结构。
CPen::GetLogPen	获得 LOGPEN 的底层结构。
CPen::operator HPEN	返回附加到 CPen 对象的 Windows 句柄。

3、CPen 类的成员函数的用法详解

3.1 CPen::CPen 函数

CPen 类重载了三个构造函数。

```
CPen();

CPen(int nPenStyle,
    int nWidth,
    COLORREF crColor);

CPen(int nPenStyle,
    int nWidth,
    const LOGBRUSH* pLogBrush,
    int nStyleCount = 0,
    const DWORD* lpStyle = NULL);
```

函数功能：构造一个 CPen 类对象。

·第一个构造函数不含任何参数，它构造的是一个未初始化的 CPen 对象。

·第二个构造函数带有 3 个参数，分别对画笔的线形，线宽和颜色进行了初始化。

参数 nPenStyle：指定画笔的风格（样式），也就是画笔的线形。

参数 nPenStyle 的值	
参数值	注释
PS_SOLID	创建一支实心的笔。
PS_DASH	创建虚线笔。仅当笔宽度为 1 或更小时才有效，以设备为单位。
PS_DOT	创建一个虚线笔。仅当笔宽度为 1 或更小时才有效，以设备为单位。
PS_DASHDOT	用交替的虚线和点创建笔。仅当笔宽度为 1 或更小时才有效，以设备为单位。
PS_DASHDOTDOT	用交替的虚线和双点创建笔。仅当笔宽度为 1 或更小时才有效，以设备为单位。
PS_NULL	创建一个空笔。
PS_INSIDEFRAME	创建一个内框线画笔，该画笔可以在 Windows GDI 输出函数定义的矩形边界所生成的封闭状的边框内绘制直线。

参数 nWidth：确定了画线的宽度。

参数 crColor：包含了一个画笔所具有的 RGB 颜色。

·第三个构造函数带有 5 个参数，现一一介绍。

参数 nPenStyle：功能同上。除了具有上一个构造函数中介绍的参数值外，还增加了如下的参数值。

新增的 nPenStyle 参数值	
参数值	注释
PS_GEOMETRIC	创建一个几何画笔。
PS_COSMETIC	创建一个装饰画笔。
PS_ALTERNATE	创建一个设置其他像素的画笔。（此款式仅适用于装饰画笔）。
PS_USERSTYLE	创建一个使用用户提供的样式数组的画笔。
PS_ENDCAP_ROUND	封顶是圆形的。
PS_ENDCAP_SQUARE	封顶是方形的。
PS_ENDCAP_FLAT	封顶是平的。
PS_JOIN_BEVEL	成尖角连接。
PS_JOIN_MITER	通过 SetMiterLimit 函数设置的当前极限值范围内斜接;否则，成尖角连接。
PS_JOIN_ROUND	成圆角连接。

参数 nWidth：指定笔的宽度。

注：nWidth 对于两个构造函数的区别：

对于第二个构造函数，如果此值为 0，则不管映射模式如何，设备单位的宽度始终为 1 像素。而对于第三个构造函数，如果 nPenStyle 是 PS_GEOMETRIC，则以逻辑单位给出宽度。如果 nPenStyle 是 PS_COSMETIC，则宽度必须设置为 1。

参数 pLogBrush：指向一个 LOGBRUSH 结构。如果 nPenStyle 是 PS_COSMETIC，则结构的 lbColor 成员 LOGBRUSH 指定笔的颜色，并且结构的 lbStyle 成员 LOGBRUSH 必须设置为 BS_SOLID。如果 nPenStyle 是 PS_GEOMETRIC，则必须使用所有成员来指定笔的画笔属性。
参数 nStyleCount：指定 lpStyle 数组的双字单位长度。如果 nPenStyle 不是 PS_USERSTYLE，该值必须为零。
参数 lpStyle：指向一组双字值。第一个值指定用户定义样式中第一个虚线的长度，第二个值指定第一个空格的长度，依此类推。如果不是 PS_USERSTYLE，则该指针必须为 NULL。

注：几种构造函数的区别：
1、如果使用第一种不带参数的构造函数，则必须使用 CreatePen，CreatePenIndirect 或 CreateStockObject 函数进行初始化；如果使用后两种构造函数，则不需要进一步的初始化。
2、带有参数的构造函数可能会在遇到错误时抛出异常，而没有参数的构造函数将始终成功。

例：创建宽度为 2 的实心红画笔。
第一种方法：

```
//创建宽度为2的实心红色笔。
CPen myPen1(PS_SOLID, 10, RGB(255, 0, 0));
```

第二种方法：

```
//创建宽度为2的实心红色笔。
LOGBRUSH logBrush;
logBrush.lbStyle = BS_SOLID;
logBrush.lbColor = RGB(255, 0, 0);
CPen myPen2(PS_SOLID|PS_GEOMETRIC|PS_ENDCAP_ROUND,2,&logBrush);
```

3.2 CPen::CreatePen

```
BOOL CreatePen(
    int nPenStyle,
    int nWidth,
    COLORREF crColor);

BOOL CreatePen(
    int nPenStyle,
    int nWidth,
    const LOGBRUSH* pLogBrush,
    int nStyleCount = 0,
    const DWORD* lpStyle = NULL);
```

函数功能：创建具有指定样式，宽度和画笔属性的逻辑装饰或几何笔，并将其附加到 CPen 对象。该函数有两个重载函数
CreatePen 函数的参数含义及用法与 Cpen 构造函数相同。
函数返回值：如果成功则为非零，如果失败则为零。

注：CreatePen 函数注意事项：
1、宽度大于 1 像素的笔应始终具有 PS_NULL，PS_SOLID 或 PS_INSIDEFRAME 样式。
2、如果笔具有 PS_INSIDEFRAME 样式，并且颜色与逻辑颜色表中的颜色不匹配，笔将以抖动颜色绘制。该 PS_SOLID 画笔样式不能用于创建一个带有抖动颜色的笔。如果笔宽小于或等于 1，则样式 PS_INSIDEFRAME 与 PS_SOLID 相同。
3、第二个 CreatePen 函数初始化具有指定样式，宽度和画笔属性的逻辑装饰或几何笔。装饰笔的宽度总是为 1; 几何笔的宽度总是以世界单位指定。应用程序创建逻辑笔后，可以通过调用 CDC :: SelectObject 函数选择该画笔，从而用于绘制线条和曲线。
4、如果 nPenStyle 是 PS_COSMETIC 和 PS_USERSTYLE，则 lpStyle 数组中的条目以样式单位指定破折号和空格的长度。样式单位由用于绘制线的设备定义。
5、如果 nPenStyle 是 PS_GEOMETRIC 和 PS_USERSTYLE，则 lpStyle 数组中的条目以逻辑单位指定破折号和空格的长度。
6、如果 nPenStyle 是 PS_ALTERNATE，则忽略样式单位，并设置为其他像素。
7、当应用程序不再需要一个给定的笔时，它应该调用 CGdiObject :: DeleteObject 成员函数或销毁该 CPen 对象，以使该资源不再使用。在绘图设备中选择笔时，应用程序不应删除笔。

例：用该函数创建宽度为 2 的实心红画笔。
第一种方法：

```
//创建宽度为2的实心红色笔。
CPen myPen1;
myPen1.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
```

第二种方法：

```
//创建宽度为2的实心红色笔。
CPen myPen2;
LOGBRUSH logBrush;
logBrush.lbStyle = BS_SOLID;
logBrush.lbColor = RGB(0, 255, 0);
myPen2.CreatePen(PS_DOT | PS_GEOMETRIC | PS_ENDCAP_ROUND, 2, &logBrush);
```

3.3 CPen::CreatePenIndirect

```
BOOL CreatePenIndirect(LPLOGPEN lpLogPen);
```

函数功能：初始化具有指向的结构中给出的样式，宽度和颜色的笔 lpLogPen。
参数 lpLogPen：指向包含有关笔的信息的 Windows LOGPEN 结构。
函数返回值：如果成功则为非零，如果失败则为零。

注：CreatePenIndirect 函数注意事项。
1、宽度大于 1 像素的笔应始终具有 PS_NULL，PS_SOLID 或 PS_INSIDEFRAME 样式。
如果笔具有 PS_INSIDEFRAME 样式，并且颜色与逻辑颜色表中的颜色不匹配，笔将以抖动颜色绘制。如果笔宽小于或等于 1，则样式 PS_INSIDEFRAME 与 PS_SOLID 相同。

2、第二个成员变量 lopnWidth 虽然也是用于指定画笔的宽度，但其类型却是 POINT 结构，在该结构中的 y 成员变量不起任何作用，只采用 x 成员变量来表示画笔宽度。
3、在实际使用中，既可以使用 CreatePenIndirect()函数来创建画笔对象，也可以使用 CreatePen()函数来创建，其实这两个函数是可互换的。

例：用该函数创建宽度为 2 的实心红画笔。

```
//创建宽度为2的实心红色笔。
LOGPEN logpen;
CPen   myPen;
logpen.lopnWidth.x = 2;//设置宽度为2
logpen.lopnColor = RGB(255, 0, 0);//设置颜色为红色
logpen.lopnStyle = PS_SOLID;//设置风格为实线
myPen.CreatePenIndirect(&logpen);//使用新的设置创建画笔
```

3.4 CPen::FromHandle

```
static CPen* PASCAL FromHandle(HPEN hPen);
```

函数功能：返回一个指向 Windows GDI 笔的对象句柄的指针。
参数 hpen：HPEN 指向 Windows GDI 笔的句柄。
函数返回值：CPen 如果成功，则返回指向对象的指针; 否则为 NULL。

注：FromHandle 函数注意事项。
如果 CPen 对象未附加到句柄，则 CPen 会创建并附加临时对象。该临时 CPen 对象仅在下一次应用程序在其事件循环中有空闲时间时才有效，此时所有临时图形对象都将被删除。换句话说，临时对象仅在处理一个窗口消息期间有效。

例：将一个 HPEN 句柄转换为 CPen*对象。

```
//将一个HPEN句柄转换为CPen*对象。
HPEN hPen;
CPen* pPen = CPen::FromHandle(hPen);
```

3.5 CPen::GetExtLogPen

```
int GetExtLogPen(EXTLOGPEN* pLogPen);
```

函数功能：获得 EXTLOGPEN 底层结构。
参数 pLogPen：指向包含笔的信息的 EXTLOGPEN 结构。
函数返回值：如果成功则为非零，如果失败则为零。

注：EXTLOGPEN 结构，限定出笔的风格、宽度和刷属性。

例：调用 GetExtLogPen 检索笔的属性，然后创建一个具有相同颜色的新的装饰笔。

```
//创建一个具有相同颜色的新的装饰笔。
CPen oldPen(PS_SOLID, 1, RGB(255, 0, 0));
EXTLOGPEN extlogpen;
oldPen.GetExtLogPen(&extlogpen);
CPen newPen;
LOGBRUSH LogBrush = { extlogpen.elpBrushStyle, extlogpen.elpColor,
    extlogpen.elpHatch };
newPen.CreatePen(PS_COSMETIC, 1, &LogBrush,0,0);
```

3.6 CPen::GetLogPen

```
int GetLogPen(LOGPEN* pLogPen);
```

函数功能：获得 LOGPEN 基础结构。
参数 pLogPen：指向包含有关笔的信息的 LOGPEN 结构。
函数返回值：如果成功则为非零，如果失败则为零。

注：LOGPEN 结构，定义笔的样式，颜色和图案。

例：调用 ExtLogPen 检索现有笔的属性，然后创建一个具有相同风格的新的画笔。

```
//创建一个具有相同风格的新的画笔。
CPen oldPen(PS_SOLID, 2, RGB(255, 0, 0));
LOGPEN logpen;
oldPen.GetLogPen(&logpen);
CPen newPen(logpen.lopnStyle, logpen.lopnWidth.x, logpen.lopnColor);
```

3.7 CPen :: operator HPEN

```
operator HPEN() const;
```

函数功能：获取对象附加的 Windows GDI 句柄。
函数返回值：如果成功，则由对象表示的 Windows GDI 对象的句柄 CPen;否则为 NULL。

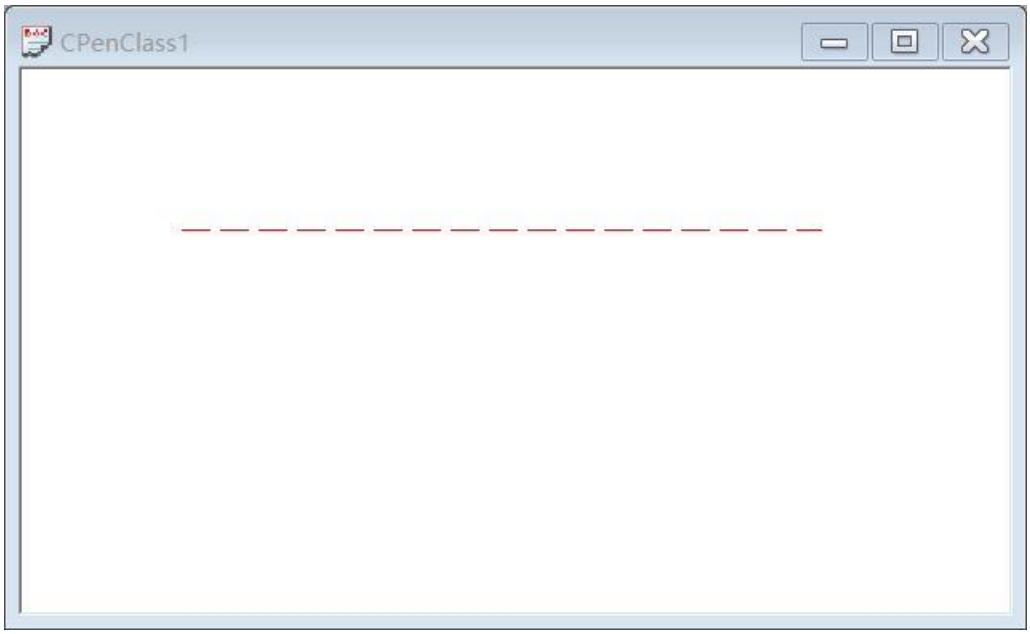
注：这个操作符是一个铸造操作符，它支持直接使用一个 HPEN 对象。

例：获取创建画笔对象的句柄。

```
//创建宽度为2的实心红画笔。
CPen myPen(PS_SOLID, 2, RGB(255, 0, 0));
//获取画笔对象的句柄
HPEN hPen = (HPEN)myPen;
```

4、例程：应用 CPen 类函数画一条红色虚线。

```
void CCPrintDialogClassView::OnDraw(CDC* pDC)
{
    CCPrintDialogClassDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 在此处为本机数据添加绘制代码
    CPen newPen;
    newPen.CreatePen(PS_DASH, 1, RGB(255, 0, 0));
    HGDIOBJ oldPen = pDC->SelectObject(newPen);
    pDC->MoveTo(100, 100);
    pDC->LineTo(500, 100);
    pDC->SelectObject(oldPen);
}
```



二、MFC 中 CColorDialog 类的介绍及用法

1、CColorDialog 类的简介

CColorDialog 对象是用于显示系统定义颜色的列表的对话框。用户可以从列表中选择或创建特定颜色，然后在对话框退出时将其返回给应用程序。

CColorDialog 类的继承层次结构为 CObject -> CCmdTarget ->CWnd ->CDialog -> CCommonDialog->CColorDialog。

使用 CColorDialog 类时需调用头文件 afxdlgs.h。

2、CColorDialog 类的成员简介

类型	名称	描述
构造函数	CColorDialog::CColorDialog	构造一个 CColorDialog 对象。
公有成员函数	CColorDialog::DoModal	显示颜色对话框，允许用户进行选择。
	CColorDialog::GetColor	返回包含所选颜色值的 COLORREF 结构。
	CColorDialog::GetSavedCustomColors	检索用户创建的自定义颜色。
	CColorDialog::SetCurrentColor	将当前颜色选择强制为指定的颜色。
保护型成员函数	CColorDialog::OnColorOK	覆盖以验证输入到对话框中的颜色。
公有数据结构	CColorDialog::m_cc	用于自定义对话框设置的结构。

3、CColorDialog 类的成员的用法详解

3.1 CColorDialog::CColorDialog

```
CColorDialog(
    COLORREF clrInit = 0,
    DWORD dwFlags = 0,
    CWnd* pParentWnd = NULL);
```

函数功能：构造一个 CColorDialog 对象。

参数 clrInit：默认颜色选择。如果未指定值，则默认值为 RGB（0,0,0）（黑色）。

参数 dwFlags：一组自定义对话框的功能和外观的标志。

参数 pParentWnd：指向对话框的父窗口或所有者窗口的指针。

例：创建一个所选颜色为红色的完全打开式对话框。

```
//创建一个所选颜色为红色的完全打开式对话框。
CColorDialog dlg(RGB(255, 0, 0), CC_FULLOPEN);
```

3.2 CColorDialog::DoModal

```
virtual INT_PTR DoModal();
```

函数功能：用于显示 Windows 常用颜色对话框，并允许用户选择颜色。

返回值：IDOK 或 IDCANCEL。如果返回 IDCANCEL，则需调用 Windows CommDlgExtendedError 函数来确定是否发生错误。(IDOK 和 IDCANCEL 是指示用户是否选择了“确定”或“取消”按钮的常量。)

注：如果要通过设置 m_cc 结构的成员来初始化各种颜色对话框选项，则应该在构建对话框对象之后与调用该函数之前进行操作。调用后 DoModal，可以调用其他成员函数，将用户输入的设置或信息检索到对话框中。

例：显示之前创建的所选颜色为红色的完全打开式对话框。

```
//创建一个所选颜色为红色的完全打开式对话框。
CColorDialog dlg(RGB(255, 0, 0), CC_FULLOPEN);
//显示之前创建的所选颜色为红色的完全打开式对话框。
dlg.DoModal();
```

3.3 CColorDialog::GetColor

```
COLORREF GetColor() const;
```

函数功能：在调用 DoModal 函数之后调用该函数，来检索用户所选择的颜色。
返回值：一个包含颜色对话框中选择颜色的 RGB 信息的 COLORREF 值。

例：获取用户选择颜色的 RGB 值。

```
    //创建一个所选颜色为红色的完全打开式对话框
    CColorDialog dlg(RGB(255, 0, 0), CC_FULLOPEN);
    //显示之前创建的所选颜色为红色的完全打开式对话框。
    if (dlg.DoModal() == IDOK)
    {
        COLORREF color = dlg.GetColor();//获取用户所选择颜色的RGB值
        CString a;
        a.Format(_T("所选颜色的RGB值为red = %u, ")
            _T("green = %u, blue = %u\n"),
            GetRValue(color), GetGValue(color), GetBValue(color));
        pDC->TextOut(20, 20, a);//将获取的值输出
    }
}
```

3.4 CColorDialog::GetSavedCustomColors

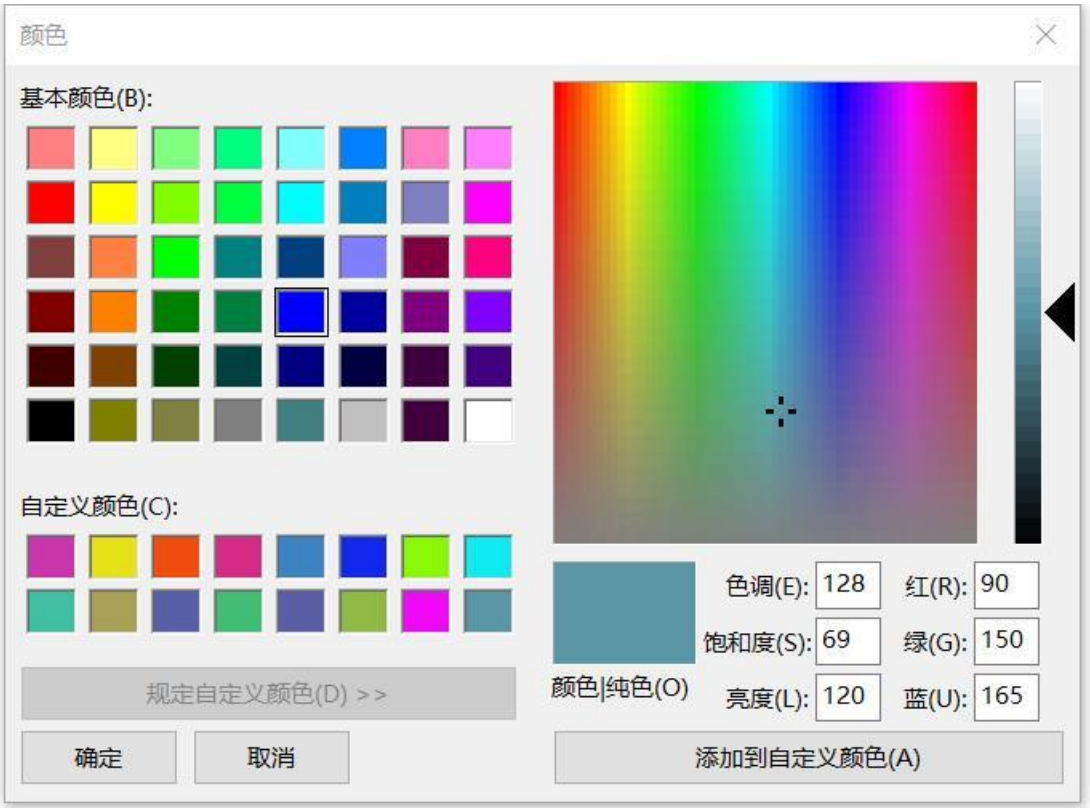
```
COLORREF GetColor() const;
```

函数功能：获取用户自定义的十六种颜色。
返回值：指向 16 个 RGB 颜色值数组的指针，用于存储用户创建的自定义颜色。

注：
1、GetSavedCustomColors 函数提供检索用户自定义颜色的通道。DoModal 函数返回 IDOK 后可以检索这些颜色。
2、返回数组中的 16 个 RGB 值中的每一个都初始化为 RGB（255,255,255）（白色）。用户选择的自定义颜色仅在应用程序中的对话框调用的过程中进行保存。如果要在应用程序调用的过程中保存这些颜色，则必须以其他方式保存它们，例如在初始化（.INI）文件中。

例：获取用户自定义的 16 种颜色的 RGB 值。

```
CColorDialog dlg;
dlg.DoModal();
if (dlg.DoModal() == IDOK)
{
    //获取用户自定义的16种颜色的RGB值
    COLORREF* ccolor = dlg.GetSavedCustomColors();
    CString a;
    for (int i = 0; i < 16; i++)
    {
        a.Format(_T("自定义的第%d种颜色的RGB值：red = %u,green = %u, blue= %u\n"),
            i+1,
            GetRValue(ccolor[i]),
            GetGValue(ccolor[i]),
            GetBValue(ccolor[i]));
        pDC->TextOut(20, 20 + 30 * i, a);//输出获取的值
    }
}
```



自定义的第1种颜色的RGB值：red=201,green=54,blue=171
自定义的第2种颜色的RGB值：red=231,green=225,blue=24
自定义的第3种颜色的RGB值：red=241,green=77,blue=14
自定义的第4种颜色的RGB值：red=213,green=43,blue=132
自定义的第5种颜色的RGB值：red=61,green=131,blue=194
自定义的第6种颜色的RGB值：red=18,green=40,blue=237
自定义的第7种颜色的RGB值：red=139,green=248,blue=7
自定义的第8种颜色的RGB值：red=15,green=235,blue=240
自定义的第9种颜色的RGB值：red=64,green=191,blue=163
自定义的第10种颜色的RGB值：red=168,green=162,blue=87
自定义的第11种颜色的RGB值：red=89,green=95,blue=166
自定义的第12种颜色的RGB值：red=65,green=190,blue=115
自定义的第13种颜色的RGB值：red=91,green=94,blue=164
自定义的第14种颜色的RGB值：red=145,green=185,blue=70
自定义的第15种颜色的RGB值：red=240,green=9,blue=247
自定义的第16种颜色的RGB值：red=90,green=150,blue=165

3.5 CColorDialog::m_cc

```
CHOOSECOLOR m_cc;
```

结构功能：一个 CHOOSECOLOR 类型的结构体，它的成员用于存储对话框的特征和值。

注：构建 CColorDialog 对象后，可以用 m_cc 在调用 DoModal 成员函数之前设置对话框的各个方面。

例：使用m_cc成员变量自定义CColorDialog的设置。CColorDialog将被显示为完全打开，红色显示为所选颜色。

```
//使用m_cc成员变量自定义CColorDialog的设置。
//CColorDialog将被显示为完全打开，红色显示为所选颜色。
CColorDialog dlg;
dlg.m_cc.Flags |= CC_FULLOPEN | CC_RGBINIT;
dlg.m_cc.rgbResult = RGB(255, 0, 0);
dlg.DoModal();
```

3.6CColorDialog::OnColorOK

```
virtual BOOL OnColorOK();
```

函数功能：覆盖并确认输入到对话框中的颜色。

注：

1、只有当用户在颜色对话框中提供自定义颜色的自定义验证时，才会重写此功能。

2、用户可以通过以下两种方法之一选择一种颜色：

a、单击调色板上的颜色。所选颜色的 RGB 值然后反映在相应的 RGB 编辑框中。

b、 在 RGB 编辑框中输入值。

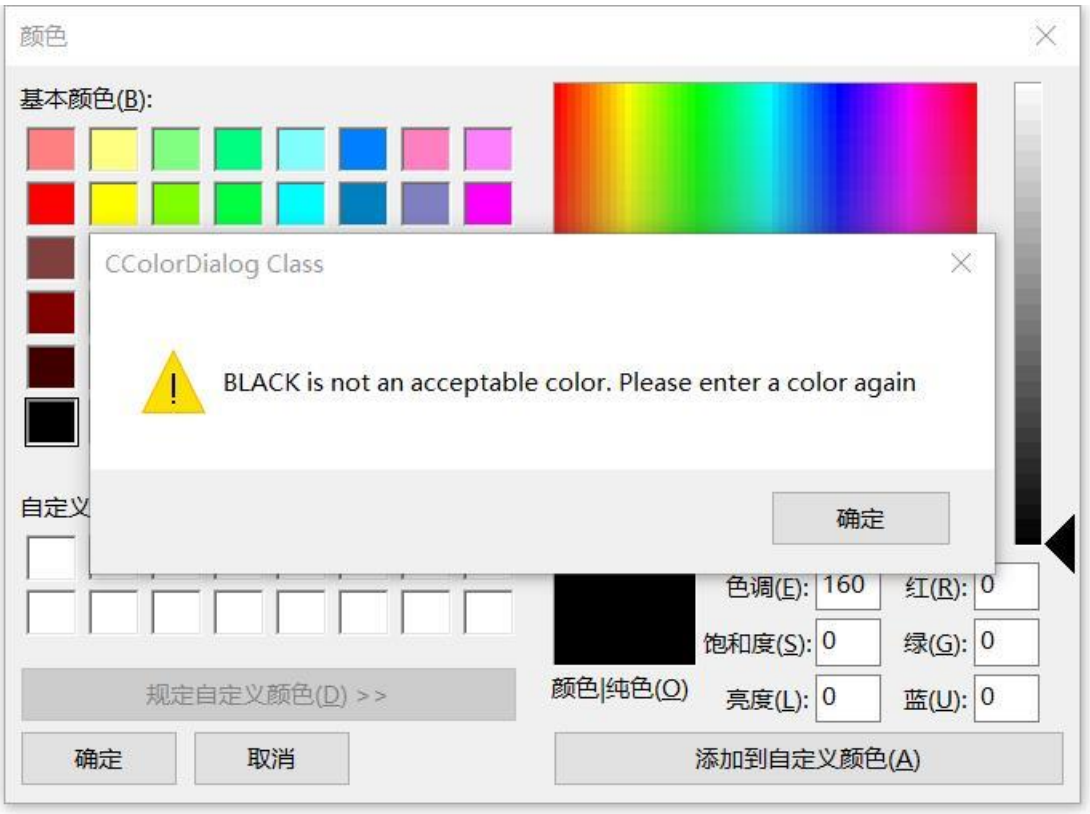
3、该函数允许你根据具体应用，拒绝用户输入到常见颜色对话框中的颜色。

4、通常不需要使用此函数，因为框架提供颜色的默认验证，并在输入无效颜色时显示消息框。

5、您可以从 OnColorOK 内部调用 SetCurrentColor 来强制进行颜色选择。一旦 OnColorOK 被触发（即用户单击确定以接受颜色更改），您可以调用 GetColor 获取新颜色的 RGB 值。

例：用该函数验证输入的颜色，如果为黑色，则强制当前颜色选择为创建对话框时最初的颜色。

```
class cMyColorDlg :public CColorDialog
{
public:
    cMyColorDlg(){}
    ~cMyColorDlg(){}

    //OnColorOK函数覆盖输入道红色，绿色和蓝色编辑控件的颜色。
    //如果颜色是黑色（RGB(0,0,0)），
    //则强制当前颜色为创建对话框时最初选择的颜色。
    //颜色对话框不会关闭。
    //用户可以输入新的颜色。
    BOOL OnColorOK()
    {
        COLORREF clrref = GetColor();
        if (clrref == RGB(0, 0, 0))
        {
            AfxMessageBox(_T("BLACK is not an acceptable color.")
                _T("Please enter a color again"));

            //GetColor()返回最初选择的颜色。
            SetCurrentColor(GetColor());

            //不会关闭颜色对话框。
            return TRUE;
        }

        //确定关闭颜色对话框。
        return FALSE;
    }
};

    //用该函数验证输入的颜色，如果为黑色，
    //则强制当前颜色选择为创建对话框时最初的颜色。
    cMyColorDlg dlg;
    dlg.m_cc.Flags |= CC_FULLOPEN | CC_RGBINIT;
    dlg.m_cc.rgbResult = RGB(255, 0, 0);
    dlg.DoModal();
    dlg.OnColorOK();
```



3.7CColorDialog :: SetCurrentColor

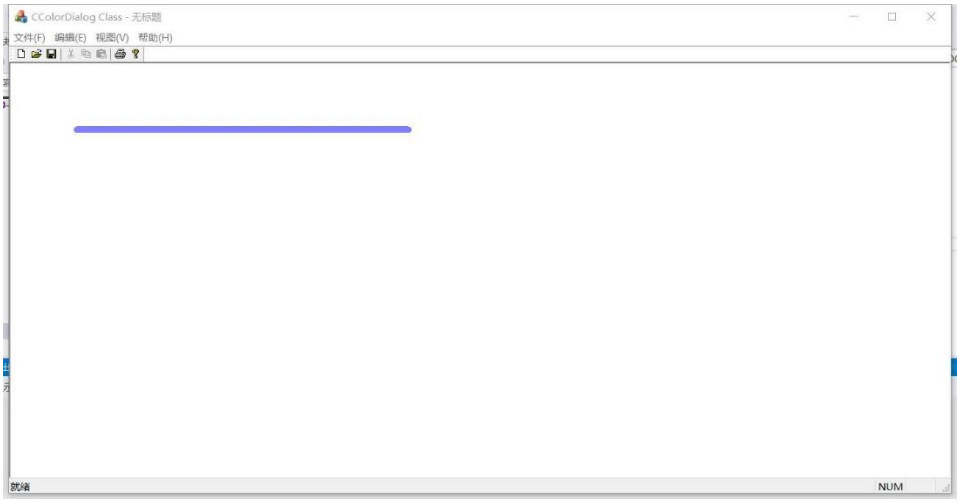
```
void SetCurrentColor(COLORREF clr);
```

函数功能 :在调用 DoModal 函数之后调用该函数, 将当前颜色选择强制为指定的颜色值 clr。
参数 clr：RGB 颜色值。

注：这个函数是从消息处理程序或者 OnColorOk 函数中调用。对话框将根据 clr 的参数值自动更新用户的颜色选择。

4、例程：应用 CColorDialog 类，从(100,100)向(600,100)绘制一条宽度为 10 像素的直线。颜色由用户自行选择，初始值为红色，且最终不能为黑色。

```
void CCPrintDialogClassView::OnDraw(CDC* pDC)
{
    CCPrintDialogClassDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 在此处为本机数据添加绘制代码

    //应用CColorDialog类，从(100,100)向(600,100)绘制一条宽度为10像素的直线。
    //颜色由用户自行选择，初始值为红色，且最终不能为黑色。

    CPen newPen;//构造CPen类对象newPen。
    cMyColorDlg dlg;//构造cMyColorDlg对象dlg。
    COLORREF color;//画笔选择颜色。

    dlg.m_cc.Flags |= CC_FULLOPEN | CC_RGBINIT;
    dlg.m_cc.rgbResult = RGB(255, 0, 0);
    dlg.DoModal();
    dlg.OnColorOK();
    color = dlg.GetColor();//获取用户选择的颜色。

    newPen.CreatePen(PS_SOLID, 10, color);//创造相应颜色的画笔。
    HGDIOBJ oldPen = pDC->SelectObject(newPen);//记录原有画笔。
    pDC-> SelectObject(newPen);
    pDC->MoveTo(100, 100);
    pDC->LineTo(600, 100);//从（100，100）向（600，100）画线。
    pDC->SelectObject(oldPen);//恢复原有画笔。
}
```



三、MFC 中 CFontDialog 类的介绍及用法

1、CFontDialog 类的简介

CFontDialog 对象是与当前系统中安装的字体列表对应的对话框。用户可以从列表中选择特定的字体，然后将该选择报告回应用程序。

CColorDialog 类的继承层次结构为 CObject -> CCmdTarget ->CWnd ->CDialog -> CCommonDialog->CFontDialog。

使用 CFontDialog 类时需调用头文件 afxdlgs.h。

2、CFontDialog 类的成员简介

类型	名称	描述
构造函数	CFontDialog::CFontDialog	构造一个 CFontDialog 对象。
公有成员函数	CFontDialog::DoModal	显示对话框，允许用户进行选择。
	CFontDialog::GetCharFormat	检索所选字体的字符格式。
	CFontDialog::GetColor	返回所选字体的颜色。
	CFontDialog::GetCurrentFont	将当前所选字体的特征分配给 LOGFONT 结构。
	CFontDialog::GetFaceName	返回所选字体的封面名称。
	CFontDialog::GetSize	返回所选字体的大小。
	CFontDialog::GetStyleName	返回所选字体的样式名称。
	CFontDialog::GetWeight	返回所选字体的重量。
	CFontDialog::IsBold	确定字体是否为粗体。
	CFontDialog::IsItalic	确定字体是否是斜体。
	CFontDialog::IsStrikeOut	确定字体是否显示与删除。
	CFontDialog::IsUnderline	确定字体是否带下划线。
公有数据结构	CFontDialog::m_cf	用于定制 CFontDialog 对象的结构。

3、CFontDialog 类的成员的用法详解

3.1CFontDialog::CFontDialog

```
CFontDialog(
    LPLOGFONT lplfInitial = NULL,
    DWORD dwFlags = CF_EFFECTS | CF_SCREENFONTS,
    CDC* pdcPrinter = NULL,
    CWnd* pParentWnd = NULL);

CFontDialog(
    const CHARFORMAT& charformat,
    DWORD dwFlags = CF_SCREENFONTS,
    CDC* pdcPrinter = NULL,
    CWnd* pParentWnd = NULL);
```

函数功能：构造一个 CFontDialog 对象。
参数 lplfInitial：指向 LOGFONT 数据结构的指针，可以让您设置字体的某些特性。
参数 charformat：指向 CHARFORMAT 数据结构的指针，允许您在丰富的编辑控件中设置某些字体的特征。
参数 dwFlags：指定一个或多个选择字体标志。可以使用按位或运算符组合一个或多个预设值。（注：如果您修改 m_cf.Flags 结构成员，请确保在更改中使用按位或运算符，以保持默认行为不变。）
参数 pdcPrinter：指向打印机设备上下文的指针。如果提供，则该参数指向要在其上选择字体的打印机的打印机设备上下文。
参数 pParentWnd：指向字体对话框的父窗口或所有者窗口的指针。

注：
1、构造函数会自动填充结构的成员 CHOOSEFONT。如果想要一个不同于默认值的字体对话框，则应该只更改这些对话框。
2、此函数的第一个重载函数只有在没有丰富的编辑控件支持时才存在。

例：构造以 12 点“Times New Roman”的字体作为所选字体的对话框实例对象。

```
//构造以12点“Times New Roman”的字体作为所选字体的对话框实例对象。
LOGFONT lf;
memset(&lf, 0, sizeof(LOGFONT));//将lf的数据清零。
//假定存在一个被初始化的CWnd
CClientDC dc(this);
//将字体高度转换为显示器对应分辨率下12点的高度。
lf.lfHeight = -MulDiv(12, dc.GetDeviceCaps(LOGPIXELSX), 72);
//将lf对应字体的名字显示为“Times New Roman”
_tcscpy_s(lf.lfFaceName, LF_FACESIZE, _T("Times New Roman"));
//创建初始化字体为lf对应类型的字体对话框对象。
CFontDialog fdlg(&lf);
```

3.2CFontDialog::DoModal

```
virtual INT_PTR DoModal();
```

函数功能：调用此功能显示 Windows 常用字体对话框，并允许用户选择字体。
返回值：IDOK 或 IDCANCEL。如果返回 IDCANCEL，则需调用 Windows CommDlgExtendedError 函数来确定是否发生错误。(IDOK 和 IDCANCEL 是指示用户是否选择了“确定”或“取消”按钮的常量。)

注：如果要通过设置 m_cf 结构的成员来初始化各种字体对话框控件，那么需要在调用 DoModal 函数之前与在构建对话框对象之后执行此操作。调用 DoModal 之后，可以调用其他成员函数，将用户输入的设置或信息检索到对话框中。

例：显示之前所创建的对话框对象。

```
//显示所构造的以12点“Times New Roman”的字体作为所选字体的对话框实例对象。
LOGFONT lf;
memset(&lf, 0, sizeof(LOGFONT));//将lf的数据清零。
//假定存在一个被初始化的CWnd
CClientDC dc(this);
//将字体高度转换为显示器对应分辨率下12点的高度。
lf.lfHeight = -MulDiv(12, dc.GetDeviceCaps(LOGPIXELSX), 72);
//将lf对应字体的名字显示为“Times New Roman”
_tcscpy_s(lf.lfFaceName, LF_FACESIZE, _T("Times New Roman"));
//创建初始化字体为lf对应类型的字体对话框对象。
CFontDialog fdlg(&lf);
```

3.3CFontDialog::GetCharFormat

```
void GetCharFormat(CHARFORMAT& cf) const;
```

函数功能：检索所选字体的字符格式。
参数 cf：一个包含所选字体字符格式信息的 CHARFORMAT 结构。

3.4CFontDialog::GetColor

```
COLORREF GetColor() const;
```

函数功能：调用此函数来检索所选的字体颜色。

返回值：所选字体的颜色。

例：获取所选字体的颜色。

```
//获取所选字体的颜色。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    COLORREF color = dlg.GetColor();
    CString s;
    s.Format(_T("所选字体的颜色的RGB值为（%d,%d,%d）"),
        GetRValue(color), GetGValue(color), GetBValue(color));
    pDC->TextOutW(100, 100, s);
}
```

3.5CFontDialog::GetCurrentFont

```
void GetCurrentFont(LPLOGFONT lplf);
```

函数功能：调用此函数将当前选定字体的特征分配给 LOGFONT 结构的成员。
参数 lplf：一个指向 LOGFONT 的指针。

注：如果在调用 DoModal 函数期间调用此函数，它将返回当前选择（用户在对话框中看到或已更改的内容）。如果在调用 DoModal 函数之后调用此函数(只有 DoModal 返回 IDOK)，则返回用户实际选择的内容。

例：获取当前选定字体的特征。

```
//获取所选字体的特征。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    LOGFONT lf;
    dlg.GetCurrentFont(&lf);
    CString s;
    s.Format(_T("所选字体的名称为："), lf.lfFaceName);
    pDC->TextOutW(100, 100, s);
     }
```

3.6CFontDialog::GetFaceName

```
CString GetFaceName() const;
```

函数功能：调用此函数检索所选字体的封面名称。
返回值：在 CFontDialog 对话框中选择的字体的封面名称。

例：获取所选字体的封面名称。

```
//获取所选字体的封面名称。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    CString facename = dlg.GetFaceName();
    pDC->TextOutW(100, 75, _T("所选字体名称为："));
    pDC->TextOutW(100, 100, facename);
}
```

3.7CFontDialog::GetSize

```
int GetSize() const;
```

函数功能：调用此函数来检索所选字体的大小。
返回值：字体的大小，以十分之一点为单位。

例：获取所选字体的大小。

```
//获取所选字体的大小。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    int size = dlg.GetSize();
    CString s;
    s.Format(_T("所选字体大小为：%d"), size);
    pDC->TextOutW(100, 100, s);
}
```

3.8CFontDialog::GetStyleName

```
CString GetStyleName() const;
```

函数功能：调用此函数来检索所选字体的样式名称。
返回值：字体的样式名称。

例：获取所选字体的样式名称。

```
//获取所选字体的样式名称。
CFontDialog dlg;
dlg.m_cf.Flags |= CF_USESTYLE;
if (dlg.DoModal() == IDOK)
```

```
{
    CString stylename = dlg.GetStyleName();
    pDC->TextOutW(100, 75, _T("所选字体的样式为："));
    pDC->TextOutW(100, 100, stylename);
}
```

3.9CFontDialog::GetWeight

```
int GetWeight() const;
```

函数功能：调用此函数来检索所选字体的权重。
返回值：所选字体权重。

例：获取所选字体权重。

```
//获取所选字体的权重。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    int weight;
    CString s;
    s.Format(_T("所选字体的权重为：%d"), weight);
    pDC->TextOutW(100, 100, s);
}
```

3.10CFontDialog::IsBold

```
BOOL IsBold() const;
```

函数功能：调用此函数确定所选字体是否为粗体。
返回值：如果所选字体的 Bold 特性已启用，则为非零;否则 0。

例：判断所选字体是否为粗体。

```
//判断所选字体是否为粗体。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    BOOL bold = dlg.IsBold();
    CString s;
    if(bold==1) s.Format(_T("所选字体是粗体"));
    else s.Format(_T("所选字体不是粗体"));
    pDC->TextOutW(100, 100, s);
}
```

3.11CFontDialog::IsItalic

```
BOOL IsItalic() const;
```

函数功能：调用此函数确定所选字体是否为斜体。
返回值：如果所选字体的 Italic 特性已启用，则为非零; 否则 0。

例：判断所选字体是否为斜体。

```
//判断所选字体是否为斜体。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    BOOL italic = dlg.IsItalic();
    CString s;
    if(italic ==1) s.Format(_T("所选字体是斜体。"));
    else s.Format(_T("所选字体不是斜体。"));
    pDC->TextOutW(100, 100, s);
}
```

3.12CFontDialog::IsStrikeOut

```
BOOL IsStrikeOut() const;
```

函数功能：调用此函数确定所选字体是否有删除符。
返回值：如果所选字体的 StrikeOut 特性已启用，则为非零; 否则 0。

例：判断所选字体是否有删除符。

```
//判断所选字体是否有删除符。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    BOOL strikeout = dlg.IsStrikeOut();
    CString s;
    if(strikeout==1) s.Format(_T("所选字体有删除符。"));
    else s.Format(_T("所选字体没有删除符。"));
    pDC->TextOutW(100, 100, s);
}
```

3.13CFontDialog::IsUnderLine

```
BOOL IsUnderline() const;
```

函数功能：调用此函数确定所选字体是否有下划线。
返回值：如果所选字体的 UnderLine 特性已启用，则为非零; 否则 0。

例：判断所选字体是否有下划线。

```
//判断所选字体是否有下划线。
CFontDialog dlg;
if (dlg.DoModal() == IDOK)
{
    BOOL underline = dlg.IsUnderline();
    CString s;
    if(underline==1) s.Format(_T("所选字体有下划线。"));
    else s.Format(_T("所选字体没有下划线。"));
    pDC->TextOutW(100, 100, s);
}
```

3.14CFontDialog::m_cf

```
CHOOSEFONT m_cf;
```

结构功能：存储对话框对象的特征。

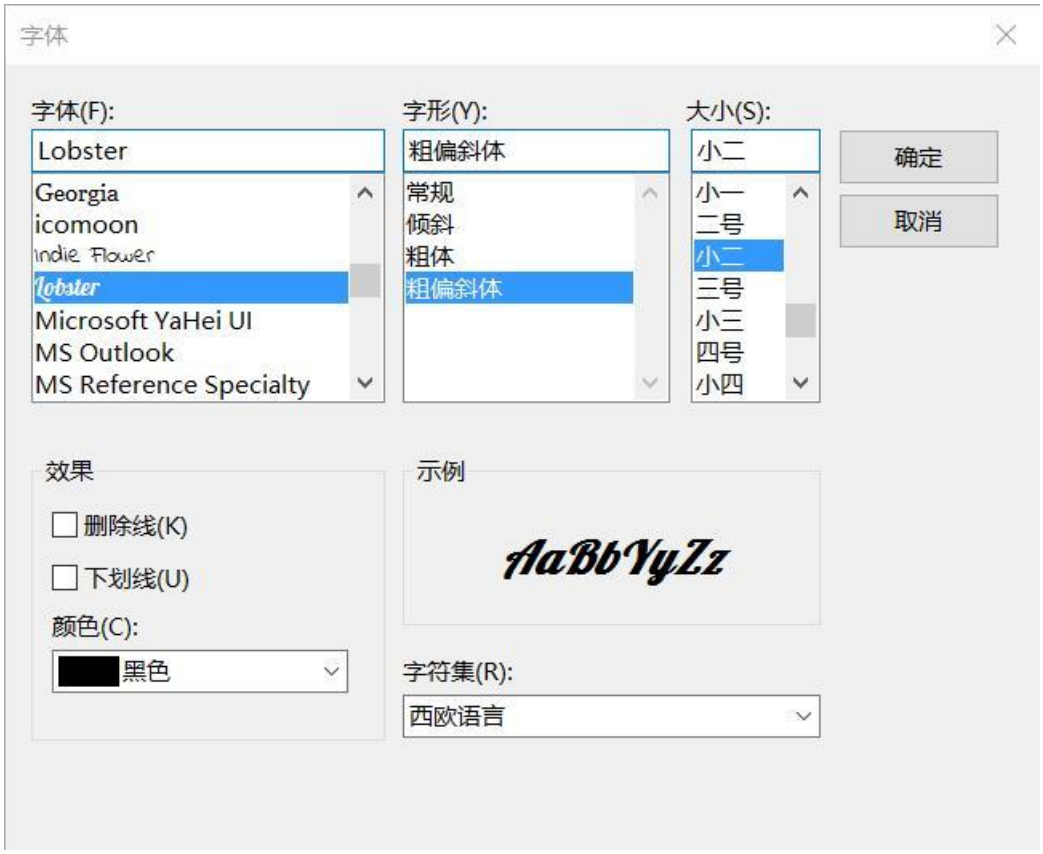
注：构建 CFontDialog 对象后，可以在调用 DoModal 成员函数之前用 m_cf 修改对话框的各个方面。

例：用 m_cf 构建一个字体对话框对象，使得字体初始颜色为红色。

```
//用m_cf构建一个字体对话框对象，使得字体初始颜色为红色。
CFontDialog dlg;
dlg.m_cf.Flags |= CF_APPLY;
dlg.m_cf.rgbColors = RGB(255, 0, 0);
dlg.DoModal();
```

4、例程：应用 CFontDialog 类，让用户自行选择一种字体，输出“Good programmers have sailed!”。

```
void CCPrintDialogClassView::OnDraw(CDC* pDC)
{
    CCPrintDialogClassDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 在此处为本机数据添加绘制代码

    //应用CFontDialog类，让用户自行选择一种字体，
    //输出“Good programmers have sailed!”。
    CFontDialog dlg;//构造CFontDialog类对象
    CFont font;//构造CFont类对象
    LOGFONT lf;//构造字体参数结构
    dlg.DoModal();//打开对话框
    memset(&lf, 0, sizeof(LOGFONT));//将lf结构归零
    dlg.GetCurrentFont(&lf);//获取用户选择的字体参数
    VERIFY(font.CreateFontIndirect(&lf));//创造该字体

    CClientDC dc(this);
    CFont* def_font = dc.SelectObject(&font);//选取该字体
    dc.TextOut(5, 5, _T("Good programmers have sailed!"));
    dc.SelectObject(def_font);

    font.DeleteObject();//销毁该字体
}
```


字体
字体(F):
Lobster
Georgia
icomoon
Indie Flower
Lobster
Microsoft YaHei UI
MS Outlook
MS Reference Specialty
字形(Y):
粗偏斜体
常规
倾斜
粗体
粗偏斜体
大小(S):
小二
小一
二号
小二
三号
小三
四号
小四
确定
取消
效果
删除线(K)
下划线(U)
颜色(C):
黑色
示例
AaBbYyZz
字符集(R):
西欧语言


CFontDialog Class - 无标题
文件(F) 编辑(E) 视图(V) 帮助(H)
Good programmers have sailed!
就绪
NUM

四、MFC 中 CPrintDialog 类的介绍及用法

1、CPrintDialog 类的简介

CPrintDialog 的功能是提供打印设置和页面设置的通用对话框，可以通过对话框来处理应用程序的打印过程的许多方面。

CPrintDialog 类的继承层次结构为 CObject -> CCmdTarget ->CWnd ->CDialog -> CCommonDialog->CPrintDialog。

使用 CPrintDialog 类时需调用头文件 afxdlgs.h。

2、CPrintDialog 类的成员简介

类型	名称	描述
构造函数	CPrintDialog::CPrintDialog	构造一个 CPrintDialog 对象。
公有成员函数	CPrintDialog::CreatePrinterDC	创建打印机设备描述表，而不显示打印对话框。
	CPrintDialog::DoModal	显示对话框，允许用户进行选择。
	CPrintDialog::GetCopies	检索所要求的份数。
	CPrintDialog::GetDefaults	检索设备默认值，而不显示对话框。
	CPrintDialog::GetDeviceName	检索当前选择的打印机设备的名称。
	CPrintDialog::GetDevMode	检索 DEVMODE 结构。
	CPrintDialog::GetDriverName	检索当前选择的打印机驱动程序的名称。
	CPrintDialog::GetFromPage	检索打印范围的起始页。
	CPrintDialog::GetPortName	检索当前选择的打印机端口的名称。
	CPrintDialog::GetPrinterDC	检索打印机设备上下文的句柄。
	CPrintDialog::GetToPage	检索打印范围的结束页。
	CPrintDialog::PrintAll	确定是否打印文档的所有页面。
	CPrintDialog::PrintCollate	确定是否请求已整理的副本。
	CPrintDialog::PrintRange	确定是否仅打印指定范围的页面。
	CPrintDialog::PrintSelection	确定是否只打印当前选择的项目。

公有数据结构	CPrintDialog::m_pd	用于定制 CPrintDialog 对象的结构。

3、CPrintDialog 类的成员的用法详解

3.1 CPrintDialog::CPrintDialog

```
CPrintDialog(
    BOOL bPrintSetupOnly,
    DWORD dwFlags = PD_ALLPAGES | PD_USEDEVMODECOPIES | PD_NOPAGENUMS |
                    PD_HIDEPRINTTOFILE | PD_NOSELECTION,
    CWnd* pParentWnd = NULL);
```

函数功能：构造一个 Windows 打印或打印设置对话框对象。
参数 bPrintSetupOnly：指定是否显示标准 Windows 打印对话框或打印设置对话框。将此参数设置为 TRUE 以显示标准 Windows 打印设置对话框。将其设置为 FALSE 以显示 Windows 打印对话框。如果 bPrintSetupOnly 为 FALSE，打印设置选项按钮仍显示在打印对话框中。
参数 dwFlags：可以使用一个或多个标志来自定义对话框的设置，使用按位或运算符组合。PD_ALLPAGES 标志将默认打印范围设置为文档的所有页面。
参数 pParentWnd：指向对话框的父窗口或所有者窗口的指针。

注：当你调用该函数时将 bPrintSetupOnly 设置为 FALSE，将会自动调用 PD_RETURNDC 的属性。在调用 DoModal，GetDefaults 或者 GetPrinterDC 之后，打印机的 DC（设备描述表）将会返回到 m_pd.hDC。只有调用者调用 DeleteDC 才能将该 DC 释放。

例：创造打印对话框，选择最初选择的“Selection”按钮。“All”单选按钮被启用，但是“Pages”单选按钮被禁用。

```
//创造打印对话框，选择最初选择的“Selection”按钮。
//“All”单选按钮被启用，但是“Pages”单选按钮被禁用。
CPrintDialog dlg(FALSE, PD_SELECTION | PD_USEDEVMODECOPIES);
```

3.2CPrintDialog::CreatePrinterDC

```
HDC CreatePrinterDC();
```

函数功能：从 DEVMODE 和 DEVNAMES 结构创建一个打印机设备描述表。
返回值：指向新创建的设备描述表的句柄。

注：该 DC 被认为是当前打印机的设备描述表，并且之前获得的打印机 DC 必须被用户删除。可以不显示“打印”对话框调用此函数并使用最终的 DC。

例：根据用户提供的信息创建打印机设备描述表。

```
//根据用户提供的信息创建打印机设备描述表。
CPrintDialog dlg(FALSE);
if (dlg.DoModal() == IDOK)
{
    HDC hdc = dlg.CreatePrinterDC();
    ASSERT(hdc);
}
```

3.3CPrintDialog::DoModal

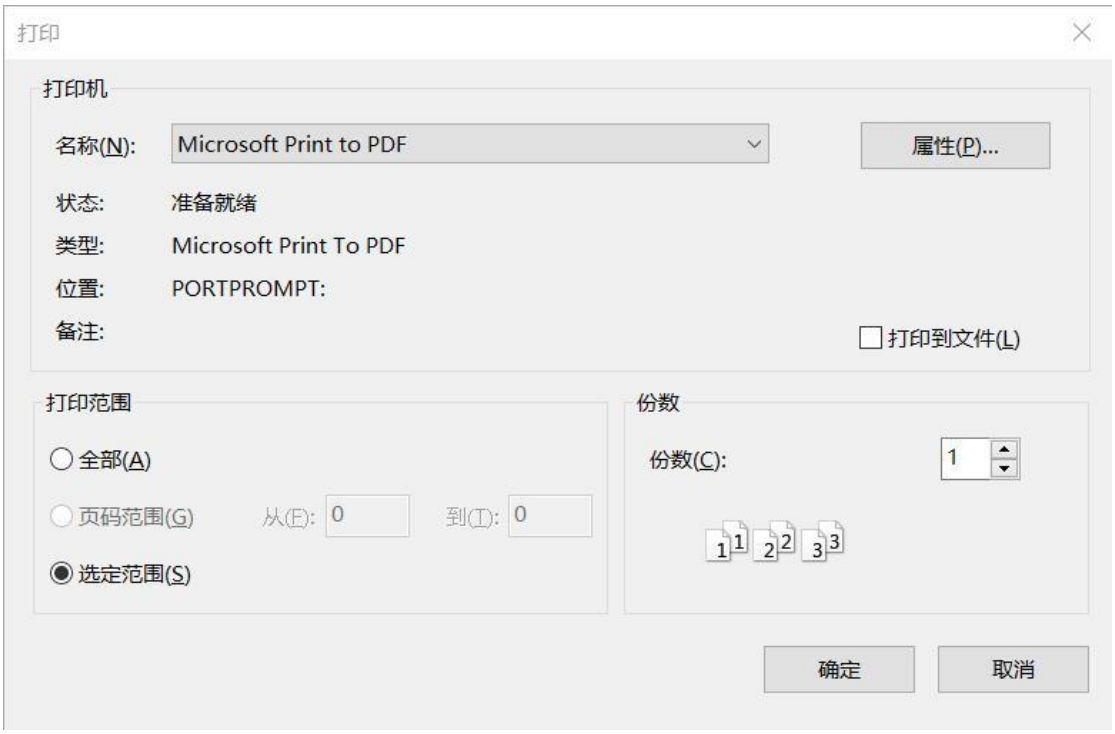
```
virtual INT_PTR DoModal();
```

函数功能：显示 Windows 常用打印对话框，并允许用户选择各种打印选项，如份数，页面范围以及是否应将文档整理。
返回值：IDOK 或 IDCANCEL。如果返回 IDCANCEL，则需调用 Windows 的 CommDlgExtendedError 函数来确定是否发生错误。

注：如果要通过设置 m_pd 数据结构来初始化各种打印对话框选项，则应在调用 DoModal 函数之前与构建对话框对象之后执行此操作。调用 DoModal 函数之后，可以调用其他成员函数，将用户输入的设置或信息检索到对话框中。

例：显示 3.1 例中构建的打印对话框。

```
//创造打印对话框，选择最初选择的“Selection”按钮。
//“All”单选按钮被启用，但是“Pages”单选按钮被禁用。
CPrintDialog dlg(FALSE, PD_SELECTION | PD_USEDEVMODECOPIES);
dlg.DoModal();
```



3.4CPrintDialog::GetCopies

```
int GetCopies() const;
```

函数功能：检索所要求的份数。
返回值：所要求的份数。
注：在调用 DoModal 函数之后调用此函数以检索所请求的副本数。

3.5CPrintDialog::GetDefaults

```
BOOL GetDefaults();
```

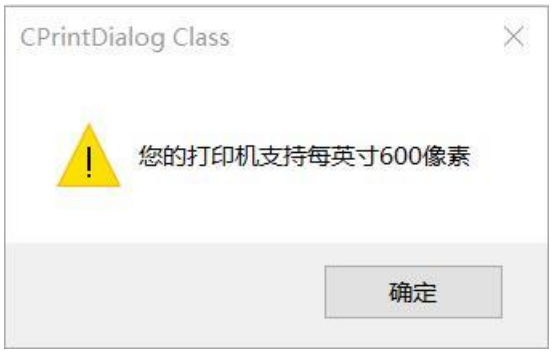
函数功能：检索默认打印机的设备默认值，而不显示对话框。
返回值：如果成功则为非 0，否则为 0。

注：
1、检索到的值存储在 m_pd 数据结构中。
2、在某些情况下，调用此函数将会同时调用构造函数并将 bPrintSetupOnly 的值设置为 FALSE。在这些情况中，打印机的设备描述表和 hDevNames 和 hDevMode 两个句柄将会自动分配。当结束使用该对话框时，调用者需要自行释放打印机的设备描述表（DC）。

例：获取默认打印机的设备描述表，并向用户返回打印机的分辨率（DPI）。

```
//获取默认打印机的设备描述表，并向用户返回打印机的分辨率（DPI）。
CPrintDialog dlg(FALSE);
if (!dlg.GetDefaults())
{
    AfxMessageBox(_T("你没有默认打印机！"));
}
else
{
    //获取我们所获得的设备描述表
    CDC dc;
    dc.Attach(dlg.m_pd.hDC);
    //请求测量
    int nHorz = dc.GetDeviceCaps(LOGPIXELSX);
    int nVert = dc.GetDeviceCaps(LOGPIXELSY);
    //大部分时候两个方向是相同的，但有时不同
    CString str;
    if (nHorz == nVert)
    {
        str.Format(_T("您的打印机支持每英寸%d像素"), nHorz);
    }
```

```
        else
        {
            str.Format(_T("您的打印机支持每英寸")
                _T("%d像素水平分辨率和%d像素垂直分辨率")
                , nHorz, nVert);
        }

        //告知用户
        AfxMessageBox(str);
    }
```



3.6CPrintDialog::GetDeviceName

```
CString GetDeviceName() const;
```

函数功能：检索当前选择打印机设备的名称。
返回值：当前选择的打印机的名称。

注：在调用 DoModal 函数之后调用该函数检索当前选择的打印机名称，或者在调用 GetDefaults 函数之后检索默认打印机的默认设备名称。

例：显示用户的默认打印机名称及其连接的端口，以及打印机使用的假脱机程序名称。

```
    //显示用户的默认打印机名称及其连接的端口，以及打印机使用的假脱机程序名称。
    CPrintDialog dlg(FALSE);
    if (!dlg.GetDefaults())
        AfxMessageBox(_T("你没有默认打印机！"));
    else
    {
        CString strDescription;
        strDescription.Format(_T("你的默认打印机是 %s 在 %s 使用 %s."),
            (LPCTSTR)dlg.GetDeviceName(),
            (LPCTSTR)dlg.GetPortName(),
            (LPCTSTR)dlg.GetDriverName());
        AfxMessageBox(strDescription);
    }
```

3.7CPrintDialog::GetDevMode

```
LPDEVMODE GetDevMode() const;
```

函数功能：检索 DEVMODE 结构。
返回值：DEMOVE 的数据结构，包含打印驱动程序的设备初始化和环境的信息。

注：在调用 DoModal 函数或 DetDefaults 函数之后调用该函数检索有关打印设备信息。必须使用 Windows GlobalUnlock 函数解锁此结构所占用的内存。

3.8CPrintDialog::GetDriverName

```
CString GetDriverName() const;
```

函数功能：检索当前选择的打印机驱动程序的名称。
返回值：一个定义驱动程序名称的 CString。

注：在调用 DoModal 函数或 GetDefaults 函数之后调用此函数来检索系统定义的打印机设备驱动程序的名称。

3.9CPrintDialog::GetFromPage

```
int GetFromPage() const;
```

函数功能：检索打印范围的起始页。
返回值：要打印的页面范围内的起始页码。

注：调用 DoModal 函数之后调用此函数，在要打印的页面范围内检索起始页码。

3.10CPrintDialog::GetPortName

```
CString GetPortName() const;
```

函数功能：检索当前选择的打印机端口的名称。
返回值：当前选择的打印机端口的名称。

注：在调用 DoModal 函数或 GetDefaults 函数之后调用此函数来检索当前选择的打印机端口的名称。

3.11CPrintDialog::GetPrinterDC

```
HDC GetPrinterDC() const;
```

函数功能：检索打印机设备描述表句柄。
返回值：若成功则返回打印机设备描述表句柄，否则为 NULL。

注 :如果构造函数 CPrintDialog 的参数 bPrintSetupOnly 为 FALSE(表示显示"打印"对话框)，则 GetPrinterDC 返回打印机设备描述表的句柄。使用完成后，必须调用 Windows 的 DeleteDC 函数删除设备描述表。

例：该函数的使用模板。

```
//CPrintDialog::GetPrinterDC函数使用模板。
CPrintDialog dlg(FALSE);
if (dlg.DoModal() == IDOK)
{
    // 获取打印机设备描述表句柄
    HDC hdc = dlg.GetPrinterDC();
    ASSERT(hdc);

    //使用句柄进行某些操作

    //清除句柄
    CDC::FromHandle(hdc)->DeleteDC();
}
```

3.12CPrintDialog::GetToPage

```
int GetToPage() const;
```

函数功能：检索打印范围的结束页。
返回值：要打印的页面范围内的结束页码。

注：在调用 DoModal 函数之后调用该函数，从而在要打印的页面范围内检索结束页码。

3.13CPrintDialog::m_pd

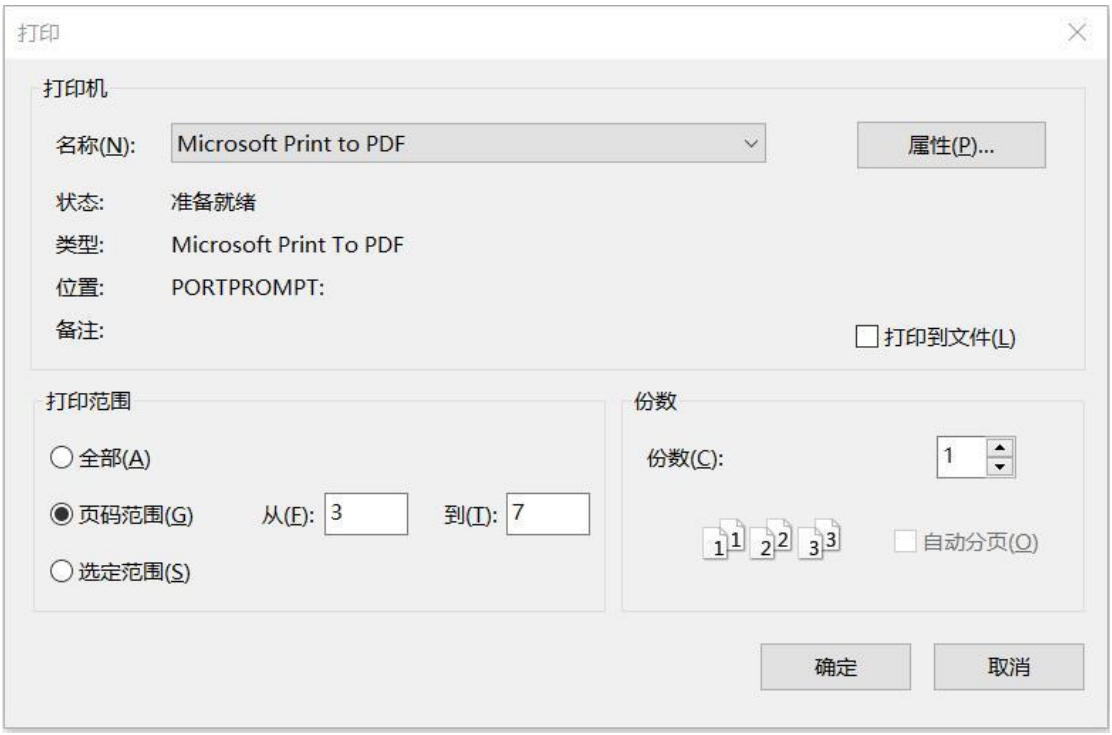
```
PRINTDLG& m_pd;
```

结构功能：存储打印对话框对象的特征。

注：构建 CPrintDialog 对象后，可以在调用 DoModal 函数之前用 m_pd 设置对话框的各个方面。如果 m_pd 直接修改数据成员，将覆盖任何默认行为。

例：显示选择“页面”单选按钮为初始选择的打印对话框。“全部”和“页面”单选按钮均被启用。并且用 m_pd 数据结构确定打印的页面范围，汇报从开始到结束的页码。

```
// 显示选择“页面”单选按钮为初始选择的打印对话框。
//“全部”和“页面”单选按钮均被启用。
//并且用m_pd数据结构确定打印的页面范围，汇报从开始到结束的页码。
CPrintDialog dlg(FALSE, PD_PAGENUMS | PD_USEDEVMODECOPIES);
dlg.m_pd.nMinPage = dlg.m_pd.nFromPage = 1;//确定最小页数为1
dlg.m_pd.nMaxPage = dlg.m_pd.nToPage = 10;//确定最大页数为10
if (dlg.DoModal() == IDOK)
{
    //确定打印的页面范围，从开始到结束的页码
    int from_page = -1, to_page = -1;
    if (dlg.PrintAll())//打印文档中所有页面
    {
        from_page = dlg.m_pd.nMinPage;
        to_page = dlg.m_pd.nMaxPage;
    }
    else if (dlg.PrintRange()) //打印文档中某一范围的页面
    {
        from_page = dlg.GetFromPage();
        to_page = dlg.GetToPage();
    }
    else if (dlg.PrintSelection())//仅打印当前选择的页面

    {
        from_page = to_page = -1;//-1表示未知
    }
    CString str;
    str.Format(_T("从%d打印到%d"), from_page, to_page);
    AfxMessageBox(str);
}
```



3.14CPrintDialog::PrintAll

```
BOOL PrintAll() const;
```

函数功能：确定是否打印文档的所有页面。
返回值：若打印所有页面，则为非 0，否则为 0。

注：调用 DoModal 函数之后调用此函数确定是否打印文档中所有页面。

3.15CPrintDialog::PrintCollate

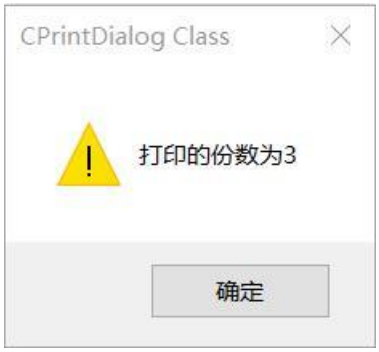
```
BOOL PrintCollate() const;
```

函数功能：确定是否打印副本。
返回值：如果用户在对话框中选择了逐份打印复选框则为非 0，否则为 0。

注：调用 DoModal 函数之后调用此函数确定打印机是否打印文档的所有副本。

例：显示打印对话框并选中逐份打印对话框，告知用户打印的份数。

```
//显示打印对话框并选中逐份打印对话框，告知用户打印的份数。
CPrintDialog dlg(FALSE, PD_ALLPAGES | PD_COLLATE | PD_NOPAGENUMS |
    PD_HIDEPRINTTOFILE);
if (dlg.DoModal() == IDOK)
{
    CString str;
    //如果选中了逐份打印复选框，则GetCopies（）将返回
    //打印的份数。否则，GetCopies（）总是返回1。
    //然后，可以从打印设备的DEVMODE结构中找到打印的份数。
    if (dlg.PrintCollate())
    {
        int num = dlg.GetCopies();
        str.Format(_T("打印的份数为%d\n"), num);
        AfxMessageBox(str);
    }
    else
    {
        LPDEVMODE devmode = dlg.GetDevMode();
        str.Format(_T("Number of copies printed = %d\n"), devmode->dmCopies);
        AfxMessageBox(str);
    }
}
```



3.16CPrintDialog::PrintRange

```
BOOL PrintRange() const;
```

函数功能：确定是否只打印指定范围的页面。
返回值：如果用户只打印文档的一个页面范围则为非 0，否则为 0。

注：调用 DoModal 函数之后，调用此函数确定是否只打印文档的一个页面范围。

3.17CPrintDialog::PrintSelection

```
BOOL PrintSelection() const;
```

函数功能：确定是否只打印当前选择的项目。
返回值：如果用户只打印所选择项目则为非 0，否则为 0。

注：调用 DoModal 函数之后，调用此函数确定是否只打印当前选择的项目。

五、资料来源

1、CPen 类参考资料

1、MSDN-CPen Class
https://msdn.microsoft.com/en-us/library/czhcb94f.aspx#CPen::CPen
2、百度百科-CPen
http://baike.baidu.com/link?url=x1dv9AOCD79Tam1oWaVhr7lJgKnoKqjMY0N7iSDW5dprxx6Rr42E5amswlCDmGBgqzqPgMoP-QKepbavLyNPsK
3、CSDN-CPen 类
http://blog.csdn.net/wuyuan2011woaini/article/details/7556123

2、CColorDialog 类参考资料

MSDN-CColorDialog Class
https://msdn.microsoft.com/en-us/library/6w6cd538.aspx

3、CFontDialog 类参考资料

MSDN-CFontDialog Class
https://msdn.microsoft.com/en-us/library/kck77523.aspx

4、CPrintDialog 类参考资料

MSDN-CPrintDialog Class
https://msdn.microsoft.com/en-us/library/sk61115a.aspx