MFC绘图基础 .

from：http://blog.csdn.net/leolee82/article/details/6992590

先认识一下MFC中的一些和绘图有关的结构体和类
1.点
(1)点结构POINT点数据结构POINT用来表示一点的x、y坐标：

```
typedef struct tagPOINT {
LONG x;
LONG y;
} POINT;
```

(2)点类CPoint
点类CPoint为一个没有基类的独立类，封装了POINT结构，有成员变量x和y
其构造函数有5种：

```
CPoint( );
CPoint( int initX, int initY );
CPoint( POINT initPt );
CPoint( SIZE initSize );
CPoint( LPARAM dwPoint );// 低字设为x、高字设为y
```

CPoint类还定义了4个平移和设置函数：

```
void Offset(int xOffset, int yOffset);
void Offset(POINT point);
void Offset(SIZE size);
void SetPoint(int X, int Y);
```

CPoint类还重载了+、-、+=、-=、==、!=等运算符来支持CPoint对象和CPoint、POINT、SIZE对象之间的运算。

2. 大小
(1)大小结构SIZE
大小(size尺寸)结构SIZE用来表示矩形的宽cx和高cy：

```
typedef struct tagSIZE {
LONG cx;
LONG cy;
} SIZE
```

(2)大小类CSize
大小类CSize也为一个没有基类的独立类，封装了SIZE结构，有成员变量cx和cy
其构造函数也有5种：

```
CSize( );
CSize( int initCX, int initCY );
CSize( SIZE initSize );
CSize( POINT initPt );
CSize( DWORD dwSize ); // 低字设为cx、高字设为cy
```

CSizet类也重载了+、-、+=、-=、==、!=等运算符来支持CSize对象和CSize、POINT、SIZE、RECT对象之间的运算

3. 矩形
(1)矩形结构RECT
矩形结构RECT定义了矩形的左上角与右下角的坐标：

```
typedef struct tagRECT {
LONG left;
LONG top;
LONG right;
LONG bottom;
} RECT;
```

(2)矩形类CRect
矩形类CRect也为一个没有基类的独立类，封装了RECT结构，有成员变量left、top、right和bottom
其构造函数有6种：

```
CRect( );
CRect( int l, int t, int r, int b );
CRect( const RECT& srcRect );
CRect( LPCRECT lpSrcRect );
CRect( POINT point, SIZE size );
CRect( POINT topLeft, POINT bottomRight );
```

CRect类重载了=，+、-，+=、-=，==、!=，&、|，& =、|=等运算符来支持CRect对象和CRect、POINT、SIZE、RECT对象之间的运算。

还定义了转换符LPCRECT和LPRECT来自动完成CRect对象到矩形结构和类指针LPCRECT和LPRECT的转换。

判断点是否在矩形中有时需要判断某点（如鼠标位置）是否在某一矩形区域中，这可以调用CRect类的PtInRect函数来做：
BOOL PtInRect(POINT point) const;
该函数当点point在其矩形区域内时，返回真。注意，该矩形区域不包括矩形的右边界和底边界

然后就是关于 设备描述表 的获取和释放了，前一文章提到过， 这里就不提了
只说一个特别点的概念“安全DC句柄”
也可以用CDC类的成员函数：
HDC GetSafeHdc();
来获取CD所对应窗口（如客户区）的安全DC句柄，该句柄在窗口存在期间一直是有效的。
例如，可先定义类变量HDC m_hDC;，再在适当的地方给它赋值m_hDC = GetDC()->GetSafeHdc();，然后就可以放心地使用了

因为不能简单的将这一指针保存在类的数据成员中，而是应该借助GetSafeHdc成员函数将它转换为windows句柄 （唯一能够持久存在的GDI标示）

例如，可以使用CDC类的成员函数
BOOL Attach(HDC hDC); // 成功返回非0
来将CDC对象与DC句柄连接在一起

4.设置绘图颜色
(1)颜色
Windows中的颜色一般用4个字节表示，Win32 API中定义了一个专门表示颜色索引值的变量类型COLORREF：(windef.h)
typedef DWORD COLORREF; // 0x00bbggrr
和一个由红绿蓝三原色构造颜色值的宏RGB：(wingdi.h)
#define RGB(r,g,b) ((COLORREF)(((BYTE)(r)|((WORD)((BYTE)(g))<<8))|(((DWORD)(BYTE)(b))<<16)))

其中，r、g、b为字节变量，取值范围为0~255。其函数说明为：
COLORREF RGB(
BYTE bRed, // 红分量
BYTE bGreen, // 绿分量
BYTE bBlue // 蓝分量
);

例如：
COLORREF red, gray;
red = RGB(255, 0, 0);
gray = RGB(128, 128,128);

在API中还定义了由COLORREF变量获取各个颜色分量的宏Get?Value：(wingdi.h)
#define GetRValue(rgb) (LOBYTE(rgb))
#define GetGValue(rgb) (LOBYTE(((WORD)(rgb)) >> 8))
#define GetBValue(rgb) (LOBYTE((rgb)>>16))

(2)点色（像素）
在Windows中，像素(pixel)的颜色是直接由设备上下文类CDC的成员函数SetPixel来设置的
该函数的原型为：
COLORREF SetPixel(int x, int y, COLORREF crColor);
COLORREF SetPixel(POINT point, COLORREF crColor);
其中，x与y分别为像素点的横坐标与纵坐标，crColor为像素的颜色值。

例如：
pDC->SetPixel(10, 10, RGB(0, 255, 0));
另外，也可以用CDC的成员函数
COLORREF GetPixel(int x, int y) const;
COLORREF GetPixel(POINT point) const;
来获得指定点(x, y)或point的颜色。

例如：
COLORREF col;
col = pDC->GetPixel(10, 10);

(3)线色（笔）
在Windows中，线状图必须用笔(pen)来画，所以线的颜色就由笔色来确定。
在MFC中，笔的属性和功能由CPen类提供（CPen是CGDIObject的派生类）。
笔的创建与使用的步骤为：

#1创建笔对象：
创建笔类CPen对象的方法有如下两种：
CPen(int nPenStyle, int nWidth, COLORREF crColor);
其中：
nPenStyle为笔的风格，可取值：PS_SOLID, PS_DASH, PS_DOT, PS_DASHDOT, PSDASHDOTDOT
注意：1~4号笔风格只是在笔宽=0或1时有效，笔宽>1时总为实心的

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

#2将笔对象选入设备上下文：
为了能使用我们所创建的笔对象，必须先将它选入设备上下文，这可以调用设备上下文类CDC的成员函数SelectObject来完成：
CPen* SelectObject(CPen* pPen);返回值为指向原来笔对象的指针（一般将其保存下来，供下次再装入时使用）。
如pOldPen = pDC->SelectObject(&pen);

另外，Windows中有一些预定义的笔对象，可用CDC的另一成员函数SelectStockObject将其选入DC
其函数原型为：virtual CGdiObject* SelectStockObject(int nIndex);
预定义的笔对象有BLACK_PEN（黑色笔）、WHITE_PEN （白色笔）、NULL_PEN（空笔/无色笔）
例如：pDC->SelectStockObject(BLACK_PEN);

#3使用设备上下文画线状图：
画线状图以及面状图的边线，所使用的是当前设备上下文中的笔对象。
线状图有直线、折线、矩形、（椭）圆（弧）等，详见

#4将笔对象从设备上下文中放出：
为了能删除使用过的笔对象，必须先将它从设备上下文中释放出来后，然后才能删除。
释放的方法是装入其他的笔对象（一般是重新装入原来的笔对象）。例如
pDC->SelectObject(pOldPen);

#5删除笔对象：
为了能删除笔对象，必须先将其从设备上下文中释放。删除方法有如下几种：
调用笔类CDC的成员函数DeleteObject删除笔的当前内容（但是未删除笔对象，以后可再用成员函数CreatePen在笔对象中继续创建新的笔内容）。
如
pen.DeleteObject();
使用删除运算符delete将笔对象彻底删除，如delete pen;
自动删除：若笔对象为局部变量，则在离开其作用域时，会被系统自动删除

(4)面色（刷）
在Windows中，面状图必须用刷(brush)来填充，所以面色是由刷色来确定的。
MFC中的刷类为CBrush（它也是CGDIObject的派生类），刷的创建与使用的步骤与笔的相似。
构造函数有4个：
CBrush( ); // 创建一个刷的空对象
CBrush(COLORREF crColor); // 创建颜色为crColor的实心刷
CBrush(int nIndex, COLORREF crColor); // 创建风格由nIndex指定且颜色为crColor的条纹(hatch)刷，其中nIndex可取条纹风格(Hatch Styles)值：
符号常量 数字常量 风格
HS_HORIZONTAL 0 水平线
HS_VERTICAL 1 垂直线
HS_FDIAGONAL 2 正斜线
HS_BDIAGONAL 3 反斜线
HS_CROSS 4 十字线(正网格)
HS_DIAGCROSS 5 斜十字线(斜网格)

CBrush(CBitmap* pBitmap); // 创建位图为pBitmap的图案刷
如：pDC->FillRect(&rect, new CBrush(RGB(r, g, b)));

与构造函数相对应，有多个创建不同类型刷的成员函数：
BOOL CreateSolidBrush(COLORREF crColor);
BOOL CreateHatchBrush(int nIndex, COLORREF crColor);
BOOL CreatePatternBrush(CBitmap* pBitmap);
BOOL CreateDIBPatternBrush(HGLOBAL hPackedDIB, UINT nUsage);
BOOL CreateDIBPatternBrush(const void* lpPackedDIB, UINT nUsage);
BOOL CreateBrushIndirect(const LOGBRUSH* lpLogBrush);
BOOL CreateSysColorBrush(int nIndex);

预定义的刷对象有BLACK_BRUSH（黑刷）、DKGRAY_BRUSH（暗灰刷）、GRAY_BRUSH（灰刷）、HOLLOW_BRUSH（空刷）、LTGRAY_BRUSH（亮灰刷）、NULL_BRUSH（空刷）、WHITE_BRUSH（白刷）
缺省的刷为空刷

与笔一样，可以用函数SelectObject或SelectStockObject将自定义的刷或预定义的刷选入DC中，供绘面状图时使用。

(5)文本色
可使用CDC类的成员函数SetTextColor和SetBkColor来分别设置输出文本的前景色和背景色：
（缺省的前景色为黑色，背景色空）
COLORREF GetTextColor( ) const;
virtual COLORREF SetTextColor(COLORREF crColor);
COLORREF GetBkColor( ) const;
virtual COLORREF SetBkColor(COLORREF crColor);

例如：

```
pDC->TextOut(10, 10, "Test text");
pDC->SetTextColor(RGB(0, 128, 0));
pDC->TextOut(10, 30, "Test text");
pDC->SetBkColor(RGB(0, 0, 128));
pDC->TextOut(10, 50, "Test text");
```

5.清屏

Windows没有提供专门的清屏函数，可以调用CWnd的下面两个函数调用来完成该功能：

```
void Invalidate(BOOL bErase = TRUE);void UpdateWindow( );
```

或调用CWnd的函数BOOL RedrawWindow(

```
LPCRECT lpRectUpdate = NULL,
CRgn* prgnUpdate = NULL,
UINT flags = RDW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE
);
```

来完成。

5 设置绘图属性除了映射模式外，还有许多绘图属性可以设置

如背景、绘图方式、多边形填充方式、画弧方向、刷原点等。

#1. 背景

1）背景色当背景模式为不透明时，背景色决定线状图的空隙颜色（如虚线中的空隙、条纹刷的空隙和文字的空隙）

可以使用CDC类的成员函数GetBkColor和SetBkColor来获得和设置当前的背景颜色：

```
COLORREF GetBkColor( ) const; // 返回当前的背景色
virtual COLORREF SetBkColor( COLORREF crColor ); // 返回先前的背景色
// 若出错返回0x800000002）
```

背景模式背景模式影响有空隙的线状图的空隙（如虚线中的空隙、条纹刷的空隙和文字的空隙）用什么办法填充。

可以使用CDC类的成员函数GetBkMode和SetBkMode来获得和设置当前的背景模式：

```
int GetBkMode( ) const; // 返回当前背景模式
int SetBkMode( int nBkMode ); // 返回先前背景模式背景模式的取值nBkMode值 名称 作用
```

OPAQUE 不透明的（缺省值） 空隙用背景色填充

TRANSPARENT 透明的 空隙处保持原背景图不变

2. 绘图模式绘图模式(drawing mode)指前景色的混合方式，

它决定新画图的笔和刷的颜色(pbCol)如何与原有图的颜色(scCol)相结合而得到结果像素色(pixel)。

1）设置绘图模式可使用CDC类的成员函数SetROP2 （ROP = Raster OPeration光栅操作）来设置绘图模式：

```
int SetROP2( int nDrawMode );
```

其中，nDrawMode可取值：绘图模式nDrawMode的取值

符号常量 作用 运算结果

R2_BLACK 黑色 pixel = black

R2_WHITE 白色 pixel = white

R2_NOP 不变 pixel = scCol

R2_NOT 反色 pixel = ~scCol

R2_COPYPEN 覆盖 pixel = pbCol

R2_NOTCOPYPEN 反色覆盖 pixel = ~pbCol

R2_MERGEPENNOT 反色或 pixel = ~scCol | pbCol

R2_MERGENOTPEN 或反色 pixel = scCol | ~pbCol

R2_MASKNOTPEN 与反色 pixel = scCol & ~pbCol

R2_MERGEPEN 或 pixel = scCol | pbCol

R2_NOTMERGEPEN 或非 pixel = ~(scCol | pbCol)

R2_MASKPEN 与 pixel = scCol & pbCol

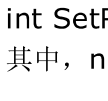
R2_NOTMASKPEN 与非 pixel = ~(scCol & pbCol)

R2_XORPEN 异或 pixel = scCol ^ pbCol

R2_NOTXORPEN 异或非 pixel = ~(scCol ^ pbCol)

其中，R2_COPYPEN（覆盖）为缺省绘图模式，R2_XORPEN（异或）较常用。

分类: VC/MFC



DoubleLi
关注 - 22
粉丝 - 652
+加关注

« 上一篇：HDC，CDC，CWindowDC，CClientDC，CPaintDC基础 .
» 下一篇：windows编程 全屏窗口的创建总结 .

posted on 2012-07-17 16:35 DoubleLi 阅读(6277) 评论(0) 编辑 收藏