第六课　绘图

一、绘图基础

1. 设备上下文(Device Context, DC)

HDC - 设备上下文句柄，表示绘图设备。

```
HDC hDC = BeginPaint (hWnd, ...);
                在这个窗口里绘图
HDC hDC = BeginPaint (hPrinter, ...);
                在这个打印机上绘图
TextOut (hDC, ...);
      在这里写字
Ellipse (hDC, ...);
      在这里画圆
EndPaint (hWnd, ...); // 结束绘图
```

2. 图形设备接口(Graphics Device Interface, GDI)

Win32提供的绘图APIs，封装在gdi32.dll动态链接库中。

二、颜色空间

1. 计算机三原色：红(R)、绿(G)、蓝(B)

R：0 - 255
G：0 - 255
B：0 - 255

2. 颜色深度：位/像素

```
 1位/像素：2种颜色，黑白图
 4位/像素：16种颜色，黑、白、深/浅灰、深/浅红、深/浅绿、
           深/浅蓝、深/浅黄、深/浅青、深/浅洋红
 8位/像素：256种颜色，调色板
16位/像素：R5-G5-B6，增强色
24位/像素：R8-G8-B8，真彩色
32位/像素：A8-R8-G8-B8，增加透明度
```

3. 颜色处理

1) COLORREF - DWORD - unsigned long - 32位无符号整型 - 0x00BBGGRR

```
COLORREF cr = 0x00000000; // 黑色
COLORREF cr = 0x000000FF; // 纯红
COLORREF cr = 0x0000FF00; // 纯绿
COLORREF cr = 0x00FF0000; // 纯蓝
```

2) 颜色组合

```
COLORREF cr = RGB (255, 0, 0); // 纯红
```

3) 颜色分解

```
GetRValue/GetGValue/GetBValue

BYTE byRed = GetRValue (cr); // 获取红色分量
```

BYTE - unsigned char

三、点
---

1. 获取点的颜色

```
COLORREF GetPixel (
    HDC hdc,   // 设备上下文句柄
    int nXPos, // 像素点水平坐标
    int nYPos  // 像素点垂直坐标
);
```

返回像素点的颜色。

2. 设置点的颜色

```
COLORREF SetPixel (
    HDC      hdc,    // 设备上下文句柄
    int      X,      // 像素点水平坐标
    int      Y,      // 像素点垂直坐标
    COLORREF crColor // 期望颜色
);
```

成功返回该像素点的实际颜色(有可能与期望颜色不同，
但尽量接近)，失败返回-1。

四、直线
---

1. 从当前点移动到目标点，目标点成为新的当前点

```
BOOL MoveToEx (
```

```
    HDC hdc,         // 设备上下文句柄
    int X,           // 目标点水平坐标
    int Y,           // 目标点垂直坐标
    LPPOINT lpPoint // 移动前的当前点(输出参数)，可为NULL
);
```

成功返回TRUE，失败返回FALSE。

2. 从当前点到目标点画直线，目标点成为新的当前点

```
BOOL LineTo (
    HDC hdc,   // 设备上下文句柄
    int nXEnd, // 目标点水平坐标
    int nYEnd  // 目标点垂直坐标
);
```

成功返回TRUE，失败返回FALSE。

五、(椭)圆弧
---

1. 画弧

```
BOOL Arc (
    HDC hdc,          // 设备上下文句柄
    int nLeftRect,    // 所在椭圆外切矩形左上角水平坐标
    int nTopRect,     // 所在椭圆外切矩形左上角垂直坐标
    int nRightRect,   // 所在椭圆外切矩形右下角水平坐标
    int nBottomRect, // 所在椭圆外切矩形右下角垂直坐标
    int nXStartArc,  // 起点水平坐标 \
    int nYStartArc,  // 起点垂直坐标  \
                                      > 起点到终点按逆时针截取
    int nXEndArc,     // 终点水平坐标  /
    int nYEndArc      // 终点水平坐标 /
);
```

成功返回TRUE，失败返回FALSE。

2. 设置截取方向

```
int SetArcDirection (
    HDC hdc,          // 设备上下文句柄
    int ArcDirection // AD_CLOCKWISE/AD_COUNTERCLOCKWISE,
                      // 按顺时针/逆时针截取
);
```

成功返回原截取方向，失败返回0。

六、封闭图形
---

1. 封闭

能被画刷填充的图形是封闭图形。由直线围成的图形不属于封闭图形。

2. 矩形

```
BOOL Rectangle (
    HDC hdc,         // 设备上下文句柄
    int nLeftRect,   // 矩形左上角水平坐标
    int nTopRect,    // 矩形左上角垂直坐标
    int nRightRect, // 矩形右下角水平坐标
    int nBottomRect // 矩形右下角垂直坐标
);
```

成功返回TRUE，失败返回FALSE。

3. (椭)圆

```
BOOL Ellipse (
    HDC hdc,         // 设备上下文句柄
    int nLeftRect,   // 外切矩形左上角水平坐标
    int nTopRect,    // 外切矩形左上角垂直坐标
    int nRightRect, // 外切矩形右下角水平坐标
    int nBottomRect // 外切矩形右下角垂直坐标
};
```

成功返回TRUE，失败返回FALSE。

4. 圆角矩形

```
BOOL RoundRect (
    HDC hdc,          // 设备上下文句柄
    int nLeftRect,    // 矩形左上角水平坐标
    int nTopRect,     // 矩形左上角垂直坐标
    int nRightRect,   // 矩形右下角水平坐标
    int nBottomRect, // 矩形右下角垂直坐标
    int nWidth,       // 圆角所在椭圆的宽度
    int nHeight       // 圆角所在椭圆的高度
);
```

成功返回TRUE，失败返回FALSE。

范例：WinPaint

七、画笔
---

1. 画笔的作用

线型、线宽和颜色。

HPEN - 画笔句柄。

2. 创建画笔

```
HPEN CreatePen (
    int      fnPenStyle, // 线型
    int      nWidth,     // 线宽
    COLORREF crColor     // 颜色
);
```

成功返回画笔句柄，失败返回NULL。

fnPenStyle取值：
```
    PS_SOLID       - 实线
    PS_DASH        - 虚线
    PS_DOT         - 点线
    PS_DASHDOT     - 点划线
    PS_DASHDOTDOT  - 双点划线
    PS_NULL        - 不可见
```

除PS_SOLID可通过nWidth指定线宽外，
其它线型只支持一像素线宽，
若nWidth>1，则按一律按实线画。

3. 将画笔应用到设备上下文中

```
HGDIOBJ SelectObject (
    HDC hdc,         // 设备上下文句柄
    HGDIOBJ hgdiobj // GDI对象句柄
);
```

成功返回原GDI对象句柄，失败返回NULL。

4. 恢复原画笔

以第一次调用SelectObject函数的返回值作为参数再次调用该函数。

5. 释放画笔

```
BOOL DeleteObject (
    HGDIOBJ hObject // GDI对象句柄
);
```

成功返回TRUE，失败返回FALSE。

注意：不能删除设备上下文正在使用的GDI对象，
因此必须先恢复原画笔，再删除新画笔。
严格遵循“建->进->用->出->删”的操作步骤。

八、画刷
---

1. 画刷的作用

封闭图形的填充颜色和图案。

HBRUSH - 画刷句柄。

2. 创建画刷

1) 单色画刷

```
HBRUSH CreateSolidBrush (
    COLORREF crColor // 填充颜色
);
```

成功返回画刷句柄，失败返回NULL。

2) 阴影线画刷

```
HBRUSH CreateHatchBrush (
    int fnStyle,    // 阴影线风格
    COLORREF clrref // 阴影线颜色
);
```

成功返回画刷句柄，失败返回NULL。

fnStyle取值：
    HS_HORIZONTAL - 水平阴影(-)
    HS_VERTICAL   - 垂直阴影(|)
    HS_CROSS   - 正交阴影(+)
    HS_BDIAGONAL - 正斜阴影(/)
    HS_FDIAGONAL - 反斜阴影(\)
    HS_DIAGCROSS - 斜交阴影(X)

3) 位图画刷

```
HBRUSH CreatePatternBrush (
    HBITMAP hbmp // 位图句柄
);
```

成功返回画刷句柄，失败返回NULL。

3. 将画刷应用到设备上下文中

```
HGDIOBJ hbrushOld = SelectObject (hDC, hbrushNew);
```

4. 恢复原画刷

```
SelectObject (hDC, hbrushOld);
```

5. 释放画刷

```
DeleteObject (hbrushNew);
```

九、库存对象

```
HGDIOBJ GetStockObject (
    int fnObject // 库存对象类型
);
```

成功返回库存对象句柄，否则返回NULL。

fnObject取值：
    HOLLOW_BRUSH
    NULL_BRUSH
    BLACK_BRUSH
    WHITE_BRUSH
    NULL_PEN
    BLACK_PEN
    WHITE_PEN
    ...

库存对象不需要通过DeleteObject释放。

范例：WinPaint

十、位图
---

1. 基本知识

计算机图像分为：

光栅图像：记录图像中每一个像素点的颜色和透明度信息。
矢量图像：记录图像的渲染算法、绘图指令等信息。

Windows中的位图(Bitmap)是一种光栅图像，通过位图句柄标识——HBITMAP。

2. 位图的使用

1) 在资源中添加位图资源

2) 从资源中加载位图

```
HBITMAP LoadBitmap(
    HINSTANCE hInstance,   // 应用程序实例句柄
    LPCTSTR   lpBitmapName // 位图资源名
);
```

成功返回位图句柄，失败返回NULL。

3) 创建一个与当前设备上下文相匹配的内存设备上下文

```
HDC CreateCompatibleDC (
    HDC hdc // 当前设备上下文句柄，
            // 可为NULL，使用屏幕设备上下文
);
```

成功返回内存设备上下文句柄，失败返回NULL。

4) 将位图选入内存设备上下文

```
HGDIOBJ hbmpOld = SelectObject (hdcMem, hbmpNew);
```

5) 位图成像

A. 原样成像：将内存设备上下文中的位图按原样成像到窗口设备上下文中

```
BOOL BitBlt (
    HDC   hdcDest, // 目标设备上下文句柄
    int   nXDest,  // 目标图像左上角水平坐标
    int   nYDest,  // 目标图像左上角垂直坐标
    int   nWidth,  // 图像宽度
    int   nHeight, // 图像高度
    HDC   hdcSrc,  // 源设备上下文句柄
    int   nXSrc,   // 源图像左上角水平坐标
    int   nYSrc,   // 源图像左上角垂直坐标
    DWORD dwRop    // 光栅操作(ROP)码，源像素与目标像素的位运算
);
```

成功返回TRUE，失败返回FALSE。

B. 缩放成像：将内存设备上下文中的位图按一定比例缩放成像到窗口设备上下文中

```
BOOL StretchBlt (
    HDC   hdcDest,      // 目标设备上下文句柄
    int   nXOriginDest, // 目标图像左上角水平坐标
    int   nYOriginDest, // 目标图像左上角垂直坐标
    int   nWidthDest,   // 目标图像宽度
    int   nHeightDest,  // 目标图像高度
    HDC   hdcSrc,       // 源设备上下文句柄
    int   nXOriginSrc,  // 源图像左上角水平坐标
    int   nYOriginSrc,  // 源图像左上角垂直坐标
    int   nWidthSrc,    // 源图像宽度
    int   nHeightSrc,   // 源图像高度
    DWORD dwRop         // 光栅操作(ROP)码
);
```

成功返回TRUE，失败返回FALSE。

6) 恢复原位图

```
SelectObject (hdcMem, hbmpOld);
```

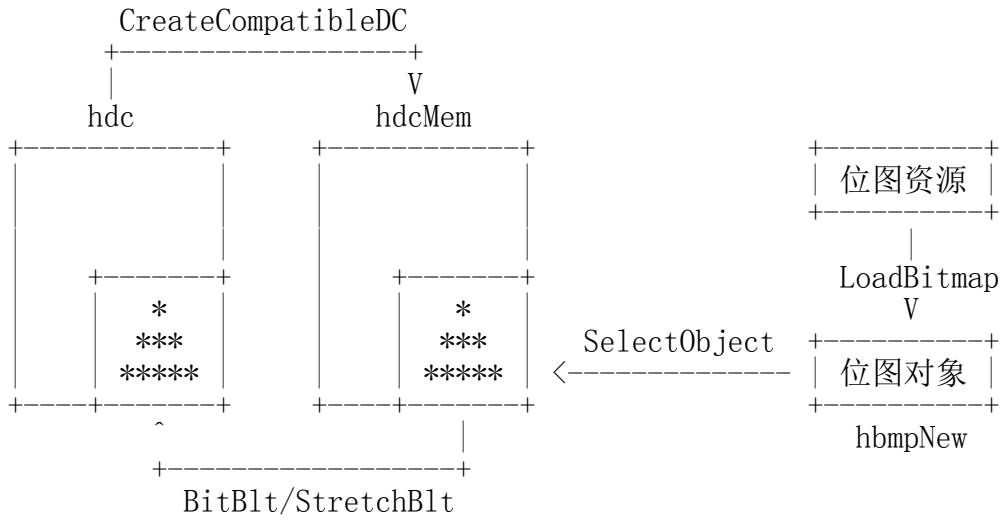
7) 释放位图

```
DeleteObject (hbmpNew);
```

8) 释放内存设备上下文

```
BOOL DeleteDC (
    HDC hdc // 设备上下文句柄
);
```

成功返回TRUE，失败返回FALSE。



一次性成像：不经由内存设备上下文，
直接将位图资源选入窗口设备上下文中，
会显露出渲染过程。

防内存泄漏：释放内存设备上下文的同时也释放了与渲染过程有关的内存资源。

3. 位图的大小

```
int GetObject (
    HGDIOBJ hgdiobj,  // GDI对象句柄
    int     cbBuffer, // 缓冲区字节数
    LPVOID  lpvObject // 缓冲区地址
);
```

若lpvObject非空，则返回实际存入缓冲区的字节数，
若lpvObject为空，则返回需要的缓冲区字节数，失败返回0。

```
BITMAP bitmap;
GetObject (hBmp, sizeof (bitmap), &bitmap);

typedef struct tagBITMAP {
    LONG   bmType;       // 位图类型，只能为0
    LONG   bmWidth;      // 位图宽度(以像素为单位)
    LONG   bmHeight;     // 位图高度(以像素为单位)
    LONG   bmWidthBytes; // 位图宽度(以字节为单位)
    WORD   bmPlanes;     // 颜色平面数
    WORD   bmBitsPixel;  // 位/每像素
    LPVOID bmBits;       // 位图数据
}   BITMAP, *PBITMAP;
```

范例：WinPaint

十一、坐标系
---

1. 坐标系分类

1) 设备坐标系：显示器设备以像素为单位，
以左上角为原点，X轴向右为正，Y轴向下为正。

A. 屏幕坐标系：以屏幕左上角为坐标原点。
B. 窗口坐标系：以窗口左上角为坐标原点。
C. 客户区坐标系：以窗口客户区左上角为坐标原点。

一个像素的大小与屏幕的尺寸和显示分辨率都有关：
比如：17寸显示器(对角线长17英寸)，4:3比率，1024*768分辨率
像素宽度=17*25.4*4/5/1024=0.337(毫米)
像素高度=17*25.4*3/5/768 =0.337(毫米)

2) 逻辑坐标系：坐标单位、坐标原点、
坐标轴的正方向等，均可人为设定。

在GDI绘图中，使用的都是逻辑坐标。

默认情况下，1逻辑=1设备，对显示器而言，1设备=1像素，
对打印机而言，1设备=1/1440英寸(25.4/1440=0.018毫米)。

2. 坐标系映射

1) 映射模式

A. 逻辑坐标系和设备坐标系之间的映射关系。

B. 设备坐标系的单位由设备决定，大小不可改变。

C. 逻辑坐标系的单位，可以通过程序设置。

```
int SetMapMode (
    HDC hdc,       // 设备上下文句柄
    int fnMapMode // 映射模式
);
```

成功返回原映射模式，失败返回0。

fnMapMode取值：
MM_TEXT - 1逻辑=1设备，原点左上角，X向右，Y向下。默认模式。
MM_LOENGLISH - 1逻辑=0.01英寸，原点左上角，X向右，Y向上。
MM_HIENGLISH - 1逻辑=0.001英寸，原点左上角，X向右，Y向上。
MM_LOMETRIC - 1逻辑=0.1毫米，原点左上角，X向右，Y向上。
MM_HIMETRIC - 1逻辑=0.01毫米，原点左上角，X向右，Y向上。
MM_TWIPS - 1逻辑=1/1440英寸，原点左上角，X向右，Y向上。
MM_ISOTROPIC - 1逻辑=自定义，原点自定义，X和Y正方向自定义。
MM_ANISOTROPIC - X的1逻辑=自定义1，Y的1逻辑=自定义2，原点自定义，
X和Y正方向自定义。
X的1逻辑=5设备，X向右。
X的1逻辑=-5设备，X向左。
Y的1逻辑=5设备，Y向下。
Y的1逻辑=-5设备，Y向上。

对于MM_ISOTROPIC和MM_ANISOTROPIC两种映射模式，必须通过以下两个函数为逻辑单位、原点和坐标轴的正方向提供定义：

```
BOOL SetViewportExtEx (
    HDC    hdc,      // 设备上下文句柄
    int    nXExtent, // 设备单位X比例
    int    nYExtent, // 设备单位Y比例
    LPSIZE lpSize    // 设备单位原比例(输出，可为NULL)
);
```

成功返回TRUE，失败返回FALSE。

```
BOOL SetWindowExtEx (
    HDC    hdc,      // 设备上下文句柄
    int    nXExtent, // 逻辑单位X比例
    int    nYExtent, // 逻辑单位Y比例
    LPSIZE lpSize    // 逻辑单位原比例(输出，可为NULL)
);
```

成功返回TRUE，失败返回FALSE。

```
SetWindowExtEx   (hDC, 1, 1, NULL);
SetViewportExtEx (hDC, 2, 3, NULL);
```

X的1逻辑=2设备=2像素
Y的1逻辑=3设备=3像素

等价于：

```
SetWindowExtEx   (hDC, 1000, 1000, NULL);
SetViewportExtEx (hDC, 2000, 3000, NULL);
```

对于MM_ISOTROPIC：

```
SetWindowExtEx   (hDC, 1, 1, NULL);
SetViewportExtEx (hDC, 2, 3, NULL);
```

1逻辑=2设备=2像素

```
SetWindowExtEx   (hDC, 1, 1, NULL);
SetViewportExtEx (hDC, 3, 2, NULL);
```

1逻辑=2设备=2像素

逻辑/设备取大者应用于两个方向。

范例：WinPaint

十二、文字

1. 绘制单行文字

```
BOOL TextOut (
    HDC     hdc,      // 设备上下文句柄
```

```
    int     nXStart,  // 文本行水平位置
    int     nYStart,  // 文本行垂直位置
    LPCTSTR lpString, // 文本字符串
    int     cbString  // 文本字符数
);
```

成功返回TRUE，失败返回FALSE。

2. 在矩形区域内绘制文字

```
int DrawText (
    HDC     hDC,      // 设备上下文句柄
    LPCTSTR lpString, // 文本字符串
    int     nCount,   // 文本字符数
    LPRECT  lpRect,   // 矩形区域
    UINT    uFormat   // 文本格式
);
```

成功返回文本高度，失败返回0。

uFormat取值：

```
    DT_LEFT           - 靠左
    DT_CENTER         - 水平居中
    DT_RIGHT          - 靠右
    DT_TOP            - 靠上
    DT_VCENTER        - 垂直居中(仅对单行有效)
    DT_BOTTOM         - 靠下(仅对单行有效)
    DT_SINGLELINE     - 单行
    DT_WORDBREAK      - 在矩形边界处按单词折行
    DT_WORD_ELLIPSIS - 显示不下的内容用省略号代替
    DT_NOCLIP         - 超出矩形部分不裁剪(无此，裁剪)
    DT_NOPREFIX       - 不转义&
                        无此，转义&为其后字符加下划线，
                        转义&&为&
```

3. 设置字间距

```
BOOL ExtTextOut (
    HDC         hdc,        // 设备上下文句柄
    int         X,          // 文本行水平位置
    int         Y,          // 文本行垂直位置
    UINT        fuOptions, // 文本输出选项
    CONST RECT* lprc,       // 矩形区域
    LPCTSTR     lpString,   // 文本字符串
    UINT        cbCount,    // 文本字符数
    CONST INT*  lpDx        // 字间距表
);
```

成功返回TRUE，失败返回FALSE。

注意：

1) 字间距是指从一个字符的左边界到下一个字符的左边界之间的距离，
   而非字空。

2）一个汉字占两个字节，需要给出这两个字节的间距——0。

```
C   语    言
-  -  -  -  -
 10  0 10  0
```

范例：WinPaint

练习：WinClock，电子时钟

十三、文字的颜色与背景

1. 文字颜色

```
COLORREF SetTextColor (
    HDC      hdc,    // 设备上下文句柄
    COLORREF crColor // 文字颜色
);
```

成功返回原文字颜色，失败返回CLR_INVALID。

2. 背景颜色

```
COLORREF SetBkColor (
    HDC      hdc,    // 设备上下文句柄
    COLORREF crColor // 背景颜色
);
```

成功返回原背景颜色，失败返回CLR_INVALID。

3. 背景模式

```
int SetBkMode (
    HDC hdc,    // 设备上下文句柄
    int iBkMode // 背景模式
);
```

成功返回原背景模式，失败返回0。

```
iBkMode取值：
    OPAQUE      - 不透明
    TRANSPARENT - 透明
```

范例：WinText

十四、字体

1. Windows常用字体格式为TrueType字体

字体名 - 字体的唯一标识，不一定和字体文件名一样，
　　双击打开看第一行。
HFONT - 字体对象句柄。

C:\windows\fonts目录下存放字体文件。

2. 字体的使用

1) 创建字体

```
HFONT CreateFont (
    int     nHeight,            // 字体高度 \
                                                > 指定一个即可，
    int     nWidth,             // 字体宽度 /  系统自动计算另一个值
    int     nEscapement,        // 倾斜角度
    int     nOrientation,       // 旋转角度，无效果，置0即可
    int     fnWeight,           // 字体粗细
    DWORD   fdwItalic,          // 是否斜体
    DWORD   fdwUnderline,       // 是否带下划线
    DWORD   fdwStrikeOut,       // 是否带删除线
    DWORD   fdwCharSet,         // 字符集标识，用GB2312_CHARSET即可
    DWORD   fdwOutputPrecision, // 输出精度，置0即可
    DWORD   fdwClipPrecision,   // 剪切精度，置0即可
    DWORD   fdwQuality,         // 输出质量，置0即可
    DWORD   fdwPitchAndFamily,  // 匹配字体，若lpszFace所指字体不存在，
                                // 则取此参数所指字体。
                                // 若此参数所指字体亦不存在，则取缺省字体
    LPCTSTR lpszFace            // 字体名
);
```

成功返回字体句柄，失败返回NULL。

2) 应用字体

```
HFONT hfntOld = SelectObject (hDC, hfontNew);
```

3) 绘制文字

TextOut/DrawText/ExtTextOutEx

4) 恢复字体

```
SelectObject (hDC, hfontOld);
```

5) 删除字体

```
DeleteObject (hfontNew);
```

范例：WinFont