11. Windows API编程-GDI绘图

1. 楔子

在前面的《Windows API编程-消息驱动》一章中，在键盘消息处理工程Win32KeyMsg中，可以看到窗口回调函数中有处理绘图消息`WM_PAINT`的代码，如下图所示：

```
case WM_PAINT:
    //为指定窗口进行绘图工作的准备，并用将和绘图有关的信息填充到一个PAINTSTRUCT结构中。
    hdc = BeginPaint(hWnd, &ps);
    Ellipse(hdc, x1, y1, x2, y2); //用设备环境句柄绘制椭圆
    EndPaint(hWnd, &ps);     //绘图的结束，释放绘图区
    break;
```

在上图红框里的代码中，小明又发现了若干陌生的类型和函数。

`BeginePaint`和`EndPaint`这一对函数比较容易理解，因为都是API函数，可以在它俩之间添加绘图代码。Ellipse函数也容易理解，是绘制椭圆的API函数。

关键是HDC、PAINTSTRUCT这两个类型是什么？从哪里来的？

此外，上一章中的鼠标消息处理工程Win32Mouse中，窗口回调函数中处理鼠标右键消息`WM_RBUTTONDOWN`的代码，以图形的形式显示字符串，所以有以下三行关键代码：

```
hdc = GetDC(hWnd);     //返回hWnd参数所指定窗口的客户区的设备环境。
TextOut(hdc, x, y, info, _tcslen(info));  //将字符串在窗口客户区以图形形式显示
ReleaseDC(hWnd, hdc);      //函数释放设备环境供其他应用程序使用
```

这里又出现了一对儿API函数：`GetDC`和`ReleaseDC`。以及在屏幕显示字符串的API函数`TextOut`。这是在非`WM_PAINT`消息处理区显示图形。

小明又经过艰苦的调查发现，绘图属于Windows的**GDI编程**。

2. 背景

在前面章节中，小明已经知道操作系统是如何帮助应用程序获得用户消息的，这是操作系统在处理用户输入。

那么，Windows操作系统是如何支持应用程序输出的？Windows窗口应用程序不支持标准输出函数（如printf），应用程序包括文字在内的所有数据均是以图形方式“绘制”到窗口上的。

问题是计算机的外部设备很多，同种类型的设备又有不同厂家、不同型号的产品。这些产品总会有些细微的差别。这就给应用程序的编写带来很大的困难，因为一个应用程序不可能顾及到所有硬件的兼容性。

这就是操作系统发挥重大作用的地方了，操作系统在应用程序和外部设备之间充当一个桥梁，让Windows应用程序做到设备无关性。

3. 设备环境（上下文）

Windows应用程序如何做到设备无关呢？是因为Windows对图形显示设备进行了封装，形成了一个统一的**虚拟图形显示设备**。应用程序可以在这个虚拟设备上进行绘图，而**设备驱动程序**将虚拟设备图形转换为物理设备图形。如此，应用程序不需要直接和硬件打交道，即设备无关。

这个虚拟图形设备用一个包含各种设备属性的**数据结构**来表示，称为**设备环境DC**。设备环境与特定的显示设备（例如显示器或打印机）相关联。对于视频显示，设备环境通常与屏幕上的一个特定的窗口相关联。

从应用程序的角度看，设备环境DC就是windows提供的一个绘画工具箱。

4. GDI

Windows上绘制图形，调用的是**GDI绘图函数**。

GDI是Windows上图形设备接口（Graphics Device Interface）的简写，它是Windows下的一个动态链接库**GDI32.dll**，可以导出几百个可以调用的图形绘制函数，它接受Windows应用程序的绘图请求（表现为GDI函数调用），并将它们传给响应的设备驱动程序，完成特定硬件的输出，比如屏幕输出、打印机输出等。GDI负责Windows的所有的图形输出，包括屏幕上输出像素、再打印机上输出硬拷贝、以及绘制Windows用户界面。

注意：窗口管理模块**USER32.dll**是GDI函数的最大用户，它用GDI函数绘制窗口框架、菜单、标题栏、滚动条、按钮等等。

5. DC+GDI---Windows绘图原理

Windows是如何绘制图形的？先想想一个人类画家，作画需要哪些东西呢？

首先要有一支笔、各种颜料和画布。——对！这些东西就相当于DC！

设备环境DC是一个数据结构，它定义了一系列图形对象及其相关的属性，以及会影响输出结果的绘图方式。这些图形对象包括：画笔（用于画直线），笔刷（用于绘图和填充），位图（用于屏幕的拷贝或滚动），调色板（用于定义可用的颜色集），剪裁区（用于剪裁和其他操作），路径（用于绘图和画图操作）。

GDI定义了很多绘图函数，比如用`TextOut`显示一些文本，只是告诉系统要写字了，但用什么样的笔（HPEN），字是什么颜色（SetTextColor），画在哪张“纸”（HBITMAP）上，则需要从一个由系统定义的数据结构中去读取（默认值）。这个系统数据结构就是DC（Device Context）。

换句话说，GDI函数只是绘画的动作，而DC则保存了绘画所需的材料和工具。

6. 句柄HDC

在win32编程中，设备环境是用HDC来标识。H的意思Handle句柄的意思。

当我们想在一个图形输出设备上绘图时，必须首先获取**设备环境句柄hdc**，并以此为参数调用GDI函数。

当程序完成了对客户区的绘制后，它必须释放设备环境句柄。在程序释放句柄之后，这个句柄不再有效并且不能再被使用。程序必须在处理同一条消息的过程中获取句柄和释放句柄，不能再两条消息中间传递一个设备句柄。唯一的例外是通过CreateDC函数创建的设备环境句柄。

Windows应用程序需要在屏幕上绘图时，通常有两种方法获取设备环境句柄。

方法一：通过BeginPaint和EndPaint函数

这种方法可以在处理WM_PAINT消息时使用。这两个函数需要两个参数：一个是窗口句柄，这是窗口消息处理过程的参数；另一个是一个类型为PAINTSTRUCT结构的变量的地址，程序通常将这个变量命名为ps，在窗口过程中定义如下：

```
PAINTSTRUCT ps;
```

该结构体包含一个名为rcPaint的RECT矩形结构成员，rcPaint定义一个包围窗口客户区无效范围的矩形。

在处理WM_PAINT消息时，窗口过程首先调用BeginPaint函数。这个函数通常会擦去无效区域的背景以便绘图。它同时还会填充ps结构的各个字段。函数的返回值就是设备环境句柄。通常将它保存在一个名为hdc的变量中。这个变量在窗口过程中定义如下：

```
HDC hdc;
```

HDC数据类型定义为32位无符号整数。此后程序可以调用任何一个需要设备环境句柄的GDI函数了，例如TextOut。

调用一次EndPaint函数将释放设备环境句柄。

方法二：通过GetDC函数来获取得相应窗口的HDC

虽然最好让程序在处理`WM_PAINT`消息时才更新整个客户区，但有时也会发现在处理非`WM_PAINT`消息时绘制部分客户区也是很有用的。有些时候，还需要设备环境句柄用作其他用途，例如获取设备环境的信息。这时可以调用GetDC函数来获得窗口客户区的设备环境句柄，在使用完以后，调用ReleaseDC函数将它释放：

```
hdc = GetDC(hwnd);
 [使用GDI函数]
ReleaseDC(hwnd, hdc);
```

与BeginPaint和EndPaint类似，GetDC和ReleaseDC必须成对使用。在处理一条消息时，当调用GetDC函数后，应该在退出窗口过程之前调用ReleaseDC。不要在处理一条消息时调用GetDC，然后在处理另一条消息时调用ReleaseDC。

与从BeginPaint函数返回的设备环境句柄不同，从GetDC返回的设备环境句柄中的裁剪矩形是整个客户区。

除了以上两种常用的获得设备环境句柄的方法之外，还有几个不太常用的:

```
hdc=GetWindowDC(hWnd);   //得到整个窗口（包括菜单栏、标题栏）DC句柄
```

```
hdc=CreateDC(设备驱动名,设备名称,NULL,NULL);    //获取指定设备的句柄
```

7. GDI常用函数

GDI的功能太多了，所以我们须要一种办法对Win32 GDI API的函数分类，以便理解GDI的结构，MSDN库将GDI API分成17个领域，清楚地描写叙述了GDI的功能。

(1)位图：处理创建、绘制设备相关位图(DDB)、设备无关位图(DIB)、DIB段、像素和区域填充的函数。

(2)画刷：处理创建、改动GDI画刷对象的函数。

(3)剪裁：处理设备上下文可绘制区域的函数。

(4)颜色：调色板管理。

(5)坐标和变换：处理映射模式、设备坐标映射逻辑和通用变换矩阵的函数。

(6)设备环境：创建设备上下文，查询、设置其属性，及选择GDI对象的函数。

(7)填充形状：绘制闭合区域及其周线的函数。

(8)字体和文本：在系统中安装和枚举字体，并用它们绘制文本字符串的函数。

(9)直线和曲线：绘制直线、椭圆曲线和贝赛尔曲线的函数。

(10)元文件：处理Windows格式的元文件或增强型元文件的生成和回放的函数。

(11)多显示监视器：同意在一个系统中使用多个显示监视器的函数。这些函数实际上是从uer32.dll导出的。

(12)画图和绘图：负责画图消息管理和窗体已画图区域的函数。当中一些函数实际上是从uer32.dll导出的。

(13)路径：负责将一系列直线和曲线组成名为路径的GDI对象，并用它来绘制的函数。

(14)画笔：处理直线绘制属性的函数。

(15)打印和打印池：负责将GDI画图命令发送到硬拷贝设备(如行式打印机和画图仪)并平滑地管理这些任务的。打印池函数是由Win32打印池提供的，包含几个系统提供的DLL和销售自己定义的模块。

(16)矩形：user32.dll提供的处理RECT结构的函数。

(17)区域：负责用区域GDI对象描写叙述一个点集的函数，并对该点集进行操作。

8、实验：图形绘制

这里给一些示例代码，大家可以模仿它们，画各种图形。

```
DrawText(ps.hdc, _T("你好！"), 3, &(ps.rcPaint), DT_CENTER);
HBRUSH hb = CreateSolidBrush(RGB(0, 0, 255));   //创建带颜色的画刷
HBRUSH orgBrs = (HBRUSH)SelectObject(ps.hdc, hb); //选新画刷为当前画刷，返回原来的画刷
Ellipse(ps.hdc, 0, 0, 100, 100); //画圆
SelectObject(ps.hdc, orgBrs); //选回原来的画刷
Ellipse(ps.hdc, 100, 100, 200, 200);//再画一个正圆
Rectangle(hdc, 200, 200, 300, 300);  //画矩形
MoveToEx(hdc, 100, 100, NULL);    //跟下一行语句一起画线段
LineTo(hdc, 150, 230);
DeleteObject(hb);    //删除这里创建的画刷对象
```

9. 平台与软件

我们还是按照“平台与软件”的角度，来总结今天所学的内容：

1. 平台干了什么？win32平台为应用程序提供了设备无关的图形输出方法。
2. 程序员干什么？在应用程序代码中调用GDI函数，绘制出想要的图形效果。
3. 平台与软件的交互机制时什么？应用程序可以调用GDI函数（win32 API的一部分）。

附录：GDI+与GDI的比较

1. GDI+是GDI的下一个版本，它进行了很好的改进，并且易用性更好。GDI的一个好处就是你不必知道任何关于数据怎样在设备上渲染的细节，GDI+更好的实现了这个优点。也就是说，GDI是一个中低层API，你还可能要知道一些设备和细节；而GDI+是一个高层的API，你不必知道设备。
2. GDI是**有状态**的，而GDI+是**无状态**的。
3. GDI绘图要使用设备环境(**DC**)和句柄(**Handle**)；而GDI+全部交由Graphics类管理。
4. GDI绘图时可以使用SelectObject频繁切换图形对象，而GDI+的图形对象是独立的。
5. GDI中存在一个“当前位置”（全局区），目的是提高绘图性能；而GDI+取消了它，以避免绘图时不确定这个“当前位置”而带来非预期的错误。
6. GDI总是将画笔和画刷绑定在一起，即使不需要填充一个区域也必须指定一个画刷；而GDI+则可以使用不同的函数分开使用画笔和画刷。