链滴

Win32 汇编学习 (4)：绘制文本

作者：Akkuman
原文链接：https://ld246.com/article/1518196154658
来源网站：链滴

这次，我们将学习如何在窗口的客户区“绘制”字符串。我们还将学习关于“设备环境”的概念。

<!--more-->

理论：

“绘制”字符串

Windows 中的文本是一个GUI（图形用户界面）对象。每一个字符实际上是由许多的像素点组成，些点在有笔画的地方显示出来，这样就会出现字符。这也是为什么我说“绘制”字符，而不是写字符通常您都是在您应用程序的客户区“绘制”字符串（尽管您也可以在客户区外“绘制”）。Windows 下的“绘制”字符串方法和 Dos 下的截然不同，在 Dos 下，您可以把屏幕想象成 85 x 25 的一个平，而 Windows 下由于屏幕上同时有几个应用程序的画面，所以您必须严格遵从规范。Windows 通把每一个应用程序限制在他的客户区来做到这一点。当然客户区的大小是可变的，您随时可以调整。

在您在客户区“绘制”字符串前，您必须从 Windows 那里得到您客户区的大小，确实您无法像在 DS 下那样随心所欲地在屏幕上任何地方“绘制”，绘制前您必须得到 Windows 的允许，然后 Windows 会告诉您客户区的大小，字体，颜色和其它 GUI 对象的属性。您可以用这些来在客户区“绘制”。

设备环境

什么是“设备环境”（DC）呢？ 它其实是由 Windows 内部维护的一个数据结构。一个“设备环境和一个特定的设备相连。像打印机和显示器。对于显示器来说，“设备环境”和一个个特定的窗口相。

“设备环境”中的有些属性和绘图有关，像：颜色，字体等。您可以随时改动那些缺省值，之所以保缺省值是为了方便。您可以把“设备环境”想象成是Windows 为您准备的一个绘图环境，而您可以时根据需要改变某些缺省属性。

当应用程序需要绘制时，您必须得到一个“设备环境”的句柄。通常有几种方法。

- 在 WM_PAINT 消息中使用 call BeginPaint
- 在其他消息中使用 call GetDC
- call CreateDC 建立你自己的 DC

您必须牢记的是，**在处理单个消息后你必须释放“设备环境”句柄**。不要在一个消息处理中获得“设环境”句柄，而在另一个消息处理中在释放它。

我们在Windows 发送 WM_PAINT 消息时处理绘制客户区，Windows 不会保存客户区的内容，它的是方法是“重绘”机制（譬如当客户区刚被另一个应用程序的客户区覆盖），Windows 会把 WM_AINT 消息放入该应用程序的消息队列。重绘窗口的客户区是各个窗口自己的责任，您要做的是在窗过程处理 WM_PAINT 的部分知道绘制什么和何如绘制。

您必须了解的另一个概念是“无效区域”。Windows 把一个最小的需要重绘的正方形区域叫做“无区域”。当 Windows 发现了一个”无效区域“后，它就会向该应用程序发送一个 WM_PAINT 消息在 WM_PAINT 的处理过程中，窗口首先得到一个有关绘图的结构体，里面包括无效区的坐标位置等您可以通过调用 BeginPaint 让“无效区”有效，**如果您不处理 WM_PAINT 消息，至少要调用缺省窗口处理函数 DefWindowProc，或者调用 ValidateRect 让“无效区”有效。否则您的应用程序会收到无穷无尽的 WM_PAINT 消息。**

下面是响应该消息的步骤：

1. 取得“设备环境”句柄
2. 绘制客户区
3. 释放“设备环境”句柄

注意，您无须显式地让“无效区”有效，这个动作由 BeginPaint 自动完成。您可以在 BeginPaint 和 ndpaint 之间，调用所有的绘制函数。几乎所有的 GDI 函数都需要“设备环境”的句柄作为参数。

内容：

我们将写一个应用程序，它会在客户区的中心显示一行 "Win32 汇编非常有意思"

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib

.DATA
ClassName db "SimpleWinClass",0
AppName  db "Our Second Window",0
OurText  db "Win32 汇编非常有意思",0

.DATA?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.CODE
start:
    invoke GetModuleHandle, NULL
    mov     hInstance,eax
    invoke GetCommandLine
    mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hwnd:HWND
    mov   wc.cbSize,SIZEOF WNDCLASSEX
    mov   wc.style, CS_HREDRAW or CS_VREDRAW
    mov   wc.lpfnWndProc, OFFSET WndProc
    mov   wc.cbClsExtra,NULL
    mov   wc.cbWndExtra,NULL
    push  hInst
    pop   wc.hInstance
```

```
    mov   wc.hbrBackground,COLOR_WINDOW+1
    mov   wc.lpszMenuName,NULL
    mov   wc.lpszClassName,OFFSET ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
          hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
       .WHILE TRUE
              invoke GetMessage, ADDR msg,NULL,0,0
              .BREAK .IF (!eax)
              invoke TranslateMessage, ADDR msg
              invoke DispatchMessage, ADDR msg
       .ENDW
       mov     eax,msg.wParam
       ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    LOCAL hdc:HDC
    LOCAL ps:PAINTSTRUCT
    LOCAL rect:RECT
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_PAINT
       invoke BeginPaint,hWnd, ADDR ps
       mov    hdc,eax
       invoke GetClientRect,hWnd, ADDR rect
       invoke DrawText, hdc,ADDR OurText,-1, ADDR rect, \
              DT_SINGLELINE or DT_CENTER or DT_VCENTER
       invoke EndPaint,hWnd, ADDR ps
    .ELSE
       invoke DefWindowProc,hWnd,uMsg,wParam,lParam
       ret
    .ENDIF
    xor   eax, eax
    ret
WndProc endp
end start
```

分析：

这里的大多数代码和Win32汇编学习(3)：简单的窗口中的一样。我只解释其中一些不相同的地方。

```
LOCAL hdc：HDC
LOCAL ps：PAINTSTRUCT
```

```
LOCAL rect：RECT
```

这些局部变量由处理 WM_PAINT 消息中的 GDI 函数调用。hdc 用来存放调用 BeginPaint 返回的“备环境”句柄。ps 是一个 PAINTSTRUCT 数据类型的变量。通常您不会用到其中的许多值，它由 Windows 传递给 BeginPaint，在结束绘制后再原封不动的传递给 EndPaint。rect 是一个 RECT 结构体型参数，它的定义如下：

```
RECT Struct left LONG ?
top LONG ?
right LONG ?
bottom LONG ?
RECT ends
```

left 和 top 是正方形左上角的坐标。right 和 bottom 是正方形右下角的坐标。客户区的左上角的坐是 x=0，y=0，这样对于 x=0，y=10 的坐标点就在它的下面。

```
invoke BeginPaint，hWnd， ADDR ps
mov hdc，eax
invoke GetClientRect，hWnd， ADDR rect
invoke DrawText， hdc，ADDR OurText，-1， ADDR rect， \
DT_SINGLELINE or DT_CENTER or DT_VCENTER
invoke EndPaint，hWnd， ADDR ps
```

在处理 WM_PAINT 消息时，您调用BeginPaint函数，传给它一个窗口句柄和未初始化的 PAINTSTRCT 型参数。调用成功后在 eax 中返回“设备环境”的句柄。下一次，调用 GetClientRect 以得到客区的大小，大小放在 rect 中，然后把它传给 DrawText。DrawText 的语法如下：

```
DrawText proto hdc：HDC， lpString：DWORD， nCount：DWORD， lpRect：DWORD， uFormat：DWORD
```

DrawText是一个高层的调用函数。它能自动处理像换行、把文本放到客户区中间等这些杂事。所以只管集中精力“绘制”字符串就可以了。让我们来看一看该函数的参数：

- hdc： “设备环境”的句柄。
- lpString：要显示的文本串，该文本串要么以NULL结尾，要么在nCount中指出它的长短。
- nCount：要输出的文本的长度。若以NULL结尾，该参数必须是-1。
- lpRect： 指向要输出文本串的正方形区域的指针，该方形必须是一个裁剪区，也就是说超过该区域字符将不能显示。
- uFormat：指定如何显示。我们可以用 or 把以下标志或到一块：
  - DT_SINGLELINE：是否单行显示。
  - DT_CENTER：是否水平居中。
  - DT_VCENTER ：是否垂直居中。

结束绘制后，必须调用 EndPaint 释放“设备环境”的句柄。 好了，现在我们把“绘制”文本串的要总结如下：

1. 必须在开始和结束处分别调用 BeginPaint 和 EndPaint；
2. 在 BeginPaint 和 EndPaint 之间调用所有的绘制函数；
3. 如果在其它的消息处理中重新绘制客户区，您可以有两种选择：

- 用 GetDC和ReleaseDC代替BeginPaint和EndPaint；
- 调用 InvalidateRect或UpdateWindow让客户区无效，这将迫使WINDOWS把WM_PAINT放应用程序消息队列，从而使得客户区重绘。