# 学习大纲

|  |
| --- |
|  |

## GetTextExtentPoint32函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GetTextExtentPoint32 函数计算指定文本字符串的宽度和高度。 BOOL GetTextExtentPoint32A(  [in] HDC hdc,  [in] LPCSTR lpString,  [in] int c,  [out] LPSIZE psizl  ); 参数 [in] hdc  设备上下文的句柄。  [in] lpString  指向指定文本字符串的缓冲区的指针。 字符串不需要以 null 结尾，因为 *c* 参数指定字符串的长度。  [in] c  *lpString* 指向[的字符串的长度](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/specifying-length-of-text-output-string)。  [out] psizl  指向 [SIZE](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-size) 结构的指针，该结构接收字符串的维度（以逻辑单元为单位）。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 **GetTextExtentPoint32** 函数使用当前所选字体来计算字符串的尺寸。 以逻辑单位表示的宽度和高度在计算时不考虑任何剪裁。  由于某些设备包含字符，因此字符串中字符范围的总和可能不等于字符串的范围。  计算的字符串宽度考虑由 [SetTextCharacterExtra](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextcharacterextra) 函数设置的字符间距和 [SetTextJustification](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextjustification) 设置的理由。 这适用于在屏幕上显示和打印。 但是，如果在 [ExtTextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-exttextouta) 中设置 *lpDx*，**则 GetTextExtentPoint32** 不会考虑字符间间距或理由。 此外，对于 EMF，打印结果始终同时考虑字符间间距和理由。  处理屏幕上显示的文本时，计算的字符串宽度会考虑 [SetTextCharacterExtra](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextcharacterextra) 函数设置的字符间距和 [SetTextJustification](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextjustification) 设置的理由。 但是，如果在 [ExtTextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-exttextouta) 中设置 *lpDx*，**则 GetTextExtentPoint32** 不会考虑字符间间距或理由。 但是，使用 EMF 打印时：   * 尽管 **GetTextExtentPoint32** 考虑了字符间间距，但打印结果会忽略字符间间距。 * 尽管 **GetTextExtentPoint32** 忽略了打印结果，但打印结果会考虑理由。   当此函数返回文本范围时，它假定文本是水平范围，即转义程始终为 0。 对于文本的水平和垂直度量，这都是如此。 即使使用指定非零转义的字体，此函数在计算文本范围时也不会使用角度。 应用必须显式转换它。 但是，当图形模式设置为 [GM\_ADVANCED](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setgraphicsmode) 并且字符方向与打印方向为 90 度时，此函数返回的值不遵循此规则。 当给定字符串的字符方向和打印方向匹配时，此函数返回 [SIZE](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-size) 结构中字符串的尺寸为 { cx ： 116， cy ： 18 }。 当同一字符串的字符方向和打印方向相距 90 度时，此函数返回 **SIZE** 结构中字符串的尺寸为 { cx ： 18， cy ： 116 }。  **GetTextExtentPoint32** 在计算文本字符串的高度时，不考虑“\n” (新行) 或“\r\n”， (回车符和新行) 字符。 示例 有关示例，请参阅 [在同一行上从不同字体绘制文本](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/drawing-text-from-different-fonts-on-the-same-line)。  **备注**  wingdi.h 标头将 GetTextExtentPoint32 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [ExtTextOut](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-exttextouta)  [字体和文本函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/font-and-text-functions)  [字体和文本概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/fonts-and-text)  [SIZE](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-size)  [SetTextCharacterExtra](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextcharacterextra)  [SetTextJustification](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-settextjustification) |

# 演练

## 1.新建一个文件夹叫做Lesson48-dll1，然后在里面新建一个win32项目，取名drawutil,注意，此时不能直接点击完成，需要点击下一步，然后在应用程序类型里面选择DLL，然后在下面勾选空项目一栏

|  |
| --- |
|  |

### 然后点击完成

|  |
| --- |
|  |

## 2.然后我们在Lesson38文件夹里面新建一个win32应用程序，用来使用这个dll文件，取名test-drawutil，这个是window应用程序。点击完成

|  |
| --- |
|  |

## 3.下面我们开始写dll代码，我们在drawutil里面添加一个drawutil.c源文件和一个drawutil.h头文件，dll项目一定要有头文件。在drawutil.c里面需要定义一个DllMain函数，这是Dll文件的入口。这个函数可以在dll被加载是时候自动调用，可以做一些初始化操作。这个函数必须要有，即使你在dll文件初始化的时候不需要做任何东西，也需要一句return TRUE.这里我们就不需要做初始化。

|  |
| --- |
|  |

## 4.然后我们定义一个函数叫做DwuCenterText，需要注意的是，windows系统里面的函数一般有2个版本，一个A，一个W版本，我们的函数也需要这么做.还有，这两个函数名字前面需要加CALLBACK，而且在函数的最前面必须写extern "C" \_\_declspec(dllexport),也就是说你必须把你的函数导出，其实extern "C"可省。如果没有导出声明，函数只能在dll里面使用，在外部无法使用。如果你的函数是给c++使用的话，前面的extern "C"不能省去，为了保险我们还是写全了。

|  |
| --- |
|  |

### 注意，如果你的函数需要在window窗口里面使用，就必须添加CALLBACK如果在命令行使用，就不需要

## 5.光怎么写还不行，我们需要在drawutil.h里面判断一下是否是c++程序来调用函数对这个声明做适当的调整

|  |
| --- |
|  |

## 6.然后我们就可以把drawutil.c的函数定义改为下面的写法

|  |
| --- |
| #include<Windows.h>  #include"drawutil.h"  int WINAPI DllMain(HINSTANCE hInstance,DWORD reason,PVOID reserved)  {  return TRUE;  }  //通用调用方法，注意EXPORT是在头文件声明的  EXPORT BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr)  {  }  EXPORT BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr)  {  }  //c++的调用方法  //extern "C" \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr)  //{  //  //}  //  //extern "C" \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr)  //{  //  //}  //下面的写法只有c语言可以调用，c++不行  // \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr)  //{  //  //}  //  // \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr)  //{  //  //} |

## 7.我们需在头文件里面声明这两个函数，否则外部无法使用。

|  |
| --- |
|  |

## 8.然后，我们需要判断程序是否使用UNICODE来决定调用哪一个版本的函数

|  |
| --- |
|  |

## 9.下面我们来完成函数功能的编写

|  |
| --- |
|  |

## 10.在项目上面点击右键-》生成，就好编译生成项目，生成成功后如图

|  |
| --- |
|  |

### 我们进入Debug目录，生成的文件如图所示

|  |
| --- |
|  |

### 在这里对我们有用的是drawutil.dll和到入库drawutil.lib

## 11.我们把这两个文件和drawutil.h头文件复制粘贴到test-drawutil项目的源码目录里面

|  |
| --- |
|  |

## 12.然后在test-drawutil项目的头文件文件夹上面点右键-》添加现有项，选择我们刚刚粘贴的drawutil.h头文件

|  |
| --- |
|  |
|  |

## 13.在test-drawutil.cpp文件中把这个头文件包含进来

|  |
| --- |
|  |

## 14.然后在WM\_PAINT的消息处理代码中调用这个函数，注意需要先获取窗口客户区的矩形大小

|  |
| --- |
|  |

## 15.右击test-drawutil项目，点击属性-》链接器-》输入，在右边的附加依赖项里面点击编辑，在弹出的窗口中添加drawutil.lib，然后点击确定

|  |
| --- |
|  |

### 然后点击应用，再点击确定

|  |
| --- |
|  |

### 编译运行程序，效果如下

|  |
| --- |
|  |

# 这一节的学习到此为止，完整代码

## dll项目

## drwautil/drawutil/drawutil.h

|  |
| --- |
| #pragma once  #ifdef \_\_cplusplus  #define EXPORT extern "C" \_\_declspec(dllexport) //如果是c++就定义这个  #else  #define EXPORT \_\_declspec(dllexport)  #endif  EXPORT BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr);  EXPORT BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr);  //根据程序是否使用UNICODE来决定调用哪一个版本的函数，也就是定义一个DwuCenterText宏  #ifdef UNICODE  #define DwuCenterText DwuCenterTextW  #else  #define DwuCenterText DwuCenterTextA  #endif |

## drwautil/drawutil/drawutil.c

|  |
| --- |
| #include<Windows.h>  #include"drawutil.h"  int WINAPI DllMain(HINSTANCE hInstance,DWORD reason,PVOID reserved)  {  return TRUE;  }  //通用调用方法，注意EXPORT是在头文件声明的  EXPORT BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr)  {  int length,centerx,centery;  SIZE sz;  length = lstrlenA(pstr);//获取字符串里面的字符有多少个  GetTextExtentPoint32A(hdc,pstr,length,&sz);//GetTextExtentPoint32 函数计算指定文本字符串的宽度和高度。  centerx = (prc->right - prc->left-sz.cx)/2;  centery = (prc->bottom - prc->top-sz.cy)/2;  return TextOutA(hdc,centerx,centery,pstr,length);  }  EXPORT BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr)  {  int length,centerx,centery;  SIZE sz;  length = lstrlenW(pstr);//获取字符串里面的字符有多少个  GetTextExtentPoint32W(hdc,pstr,length,&sz);//GetTextExtentPoint32 函数计算指定文本字符串的宽度和高度。  centerx = (prc->right - prc->left-sz.cx)/2;  centery = (prc->bottom - prc->top-sz.cy)/2;  return TextOutW(hdc,centerx,centery,pstr,length);  }  //c++的调用方法  //extern "C" \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr)  //{  //  //}  //  //extern "C" \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr)  //{  //  //}  //下面的写法只有c语言可以调用，c++不行  // \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextA(HDC hdc,PRECT prc,PCSTR pstr)  //{  //  //}  //  // \_\_declspec(dllexport) BOOL CALLBACK DwuCenterTextW(HDC hdc,PRECT prc,PCWSTR pstr)  //{  //  //} |

## dll测试项目

## test-drawutil/test-drawutil/ test-drawutil.cpp

|  |
| --- |
| // test-drawutil.cpp : 定义应用程序的入口点。  //  #include "stdafx.h"  #include "test-drawutil.h"  #include"drawutil.h"  #define MAX\_LOADSTRING 100  // 全局变量:  HINSTANCE hInst; // 当前实例  TCHAR szTitle[MAX\_LOADSTRING]; // 标题栏文本  TCHAR szWindowClass[MAX\_LOADSTRING]; // 主窗口类名  // 此代码模块中包含的函数的前向声明:  ATOM MyRegisterClass(HINSTANCE hInstance);  BOOL InitInstance(HINSTANCE, int);  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);  INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);  int APIENTRY \_tWinMain(HINSTANCE hInstance,  HINSTANCE hPrevInstance,  LPTSTR lpCmdLine,  int nCmdShow)  {  UNREFERENCED\_PARAMETER(hPrevInstance);  UNREFERENCED\_PARAMETER(lpCmdLine);  // TODO: 在此放置代码。  MSG msg;  HACCEL hAccelTable;  // 初始化全局字符串  LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);  LoadString(hInstance, IDC\_TESTDRAWUTIL, szWindowClass, MAX\_LOADSTRING);  MyRegisterClass(hInstance);  // 执行应用程序初始化:  if (!InitInstance (hInstance, nCmdShow))  {  return FALSE;  }  hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_TESTDRAWUTIL));  // 主消息循环:  while (GetMessage(&msg, NULL, 0, 0))  {  if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  }  return (int) msg.wParam;  }  //  // 函数: MyRegisterClass()  //  // 目的: 注册窗口类。  //  // 注释:  //  // 仅当希望  // 此代码与添加到 Windows 95 中的“RegisterClassEx”  // 函数之前的 Win32 系统兼容时，才需要此函数及其用法。调用此函数十分重要，  // 这样应用程序就可以获得关联的  // “格式正确的”小图标。  //  ATOM MyRegisterClass(HINSTANCE hInstance)  {  WNDCLASSEX wcex;  wcex.cbSize = sizeof(WNDCLASSEX);  wcex.style = CS\_HREDRAW | CS\_VREDRAW;  wcex.lpfnWndProc = WndProc;  wcex.cbClsExtra = 0;  wcex.cbWndExtra = 0;  wcex.hInstance = hInstance;  wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_TESTDRAWUTIL));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_TESTDRAWUTIL);  wcex.lpszClassName = szWindowClass;  wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));  return RegisterClassEx(&wcex);  }  //  // 函数: InitInstance(HINSTANCE, int)  //  // 目的: 保存实例句柄并创建主窗口  //  // 注释:  //  // 在此函数中，我们在全局变量中保存实例句柄并  // 创建和显示主程序窗口。  //  BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)  {  HWND hWnd;  hInst = hInstance; // 将实例句柄存储在全局变量中  hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,  CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);  if (!hWnd)  {  return FALSE;  }  ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);  return TRUE;  }  //  // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)  //  // 目的: 处理主窗口的消息。  //  // WM\_COMMAND - 处理应用程序菜单  // WM\_PAINT - 绘制主窗口  // WM\_DESTROY - 发送退出消息并返回  //  //  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  {  int wmId, wmEvent;  PAINTSTRUCT ps;  HDC hdc;  RECT rect;  switch (message)  {  case WM\_COMMAND:  wmId = LOWORD(wParam);  wmEvent = HIWORD(wParam);  // 分析菜单选择:  switch (wmId)  {  case IDM\_ABOUT:  DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);  break;  case IDM\_EXIT:  DestroyWindow(hWnd);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  break;  case WM\_PAINT:  hdc = BeginPaint(hWnd, &ps);  // TODO: 在此添加任意绘图代码...  GetClientRect(hWnd,&rect);  //调用我们编写的dll函数，注意需要导入头文件,还要在链接选项的输入选项里面添加drawutil.lib  DwuCenterText(hdc,&rect,\_T("Welcome to my c++ Lessons!!!"));  EndPaint(hWnd, &ps);  break;  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  }  // “关于”框的消息处理程序。  INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)  {  UNREFERENCED\_PARAMETER(lParam);  switch (message)  {  case WM\_INITDIALOG:  return (INT\_PTR)TRUE;  case WM\_COMMAND:  if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)  {  EndDialog(hDlg, LOWORD(wParam));  return (INT\_PTR)TRUE;  }  break;  }  return (INT\_PTR)FALSE;  } |