# 第一个win32项目

|  |
| --- |
|  |

# 第一个win32窗口应用程序代码win32app.cpp如下

|  |
| --- |
| // win32app.cpp : 定义应用程序的入口点  //  #include "stdafx.h"  #include "win32app.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ˉ全局字符串  LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);  LoadString(hInstance, IDC\_WIN32APP, 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\_WIN32APP));  // 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ˉ数簓: MyRegisterClass()  //  //  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\_ICON1));  //wcex.hIcon = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_WIN32APP));  //wcex.hIcon = LoadIcon(NULL, MAKEINTRESOURCE(IDI\_ERROR));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  //wcex.hCursor = LoadCursor(NULL, IDC\_CROSS);  //wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.hbrBackground = (HBRUSH)GetStockObject(GRAY\_BRUSH);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_WIN32APP);  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  //  //  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 - 绘制主窗口的消息  // 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;  switch (message)  {  case WM\_COMMAND:  wmId = LOWORD(wParam);  wmEvent = HIWORD(wParam);  // 分?析?菜?单蹋?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ú此?添?加ó任?意癮绘?图?代洙?码?...  EndPaint(hWnd, &ps);  break;  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  }  // “°关?于?”±框ò的?消?息￠处鋦理え?程ì序ò。￡  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;  } |

# 消息和消息队列

|  |
| --- |
|  |

# 句柄

|  |
| --- |
|  |

# 从变量的类型区分变量的用途

|  |
| --- |
|  |

# WinMain函数，他是win32应用程序的入口函数，新一点的vs使用的是\_tWinMain

|  |
| --- |
|  |

## 可以设置你的win32应用程序接收一个命令行参数，vc++6的界面如下，操作：点击项目=》设置或者项目属性-》调试，在命令参数一栏填写，

|  |
| --- |
|  |

## vc++ 2010的界面是这样子的

|  |
| --- |
|  |

## 然后可以在WinMain函数代码开始的地方插入一个断点来看看有没有接收到这个参数

|  |
| --- |
|  |

## 然后点击调试运行按钮。在vs2010中是按钮，然后程序就好在断点处停下来，我们查看局部变量，发现这个参数是接收到了的

|  |
| --- |
|  |

# 窗口创建的步骤

|  |
| --- |
|  |

# 窗口类结构体

|  |
| --- |
|  |

## vs2010使用的是WNDCLASSEX

|  |
| --- |
|  |

# 窗口样式的组合方法

|  |
| --- |
|  |

## 可以在一个样式上面点击右键-》转到定义，就可以查看他的定义

|  |
| --- |
|  |
|  |

# 窗口过程函数，他是一个回调函数，也就是他会自动调用

|  |
| --- |
|  |

# 窗口样式

|  |
| --- |
|  |

# CreateWindow函数，有11个参数

## 参数依次是窗口类名称，窗口名称，窗口样式，窗口起点x坐标，窗口起点y坐标，窗口宽度，窗口高度，

## 父窗口句柄，菜单句柄，应用程序示例句柄，lpParam

|  |
| --- |
|  |

## Windows32程序的运行流程：定义窗口类-》注册窗口类-》创建窗口=》显示窗口=》更新窗口=》消息循环

# GetMessage函数

|  |
| --- |
|  |

## 第一个参数的说明的前面有一个【out】，说明我们不需要给这个参数赋值，他是用来接收结果的，第三个参数可以用来获取第一个消息，用WM\_KEYFIRST 获取第一个键盘消息，用WM\_MOUSEFIRST 来获取第一个鼠标消息。第四个参数可以用来获取最后一个消息用WM\_KEYLAST 来获取最后一个键盘消息用WM\_MOUSELAST 来获取最后一个鼠标消息。如果需接收所有消息，把第三个和第四个参数都设置为0。在一个win32窗口应用程序中，通常需要一个while循环来获取消息，翻译消息，然后调用DispatchMessage函数来把消息传给操作系统，操作系统再调用我们的WndProc回调函数来处理消息。

# WndProc函数，其实名字可以任意起，不过传递个窗口类时，需要使用一样的名字。否则会有问题，如下面把它叫做WinSunProc

|  |
| --- |
|  |

## 注意：1.虽然BeginPaint函数和GetDC函数都能够获取HDC实例，都是BeginPaint函数只能够在WM\_PAINT消息的处理代码里面使用，同样道理，GetDC函数不能够在WM\_PAINT消息处理代码里面使用，只能够在别的地方使用，2.还有BeginPaint函数需要搭配EndPaint函数使用，GetDC函数需要搭配ReleaseDC函数来使用，获取了hdc在使用完成后需要释放内存。3.WM\_CLOSE消息是在用户点击了窗口的关闭按钮时发送给窗口的，然后我们在他的处理代码里面调用了DestroyWindow函数，这个函数会发送WM\_DESTROY消息，我们可以对这个消息进行处理。在上面的老师案例中，如果我们把信息框的代码放在WM\_DESTROY消息的处理代码中，如图，是会有问题的。

|  |
| --- |
|  |

## 其实是不行的，当你在信息框里面点击yes，程序会退出，点击no是保护退出的。这样子就好出现一种现象，就是窗口被销毁了，但是应用程序没有退出

|  |
| --- |
|  |

## 原因也很简单，你不是调用了销毁窗口函数吗，那么窗口就好被销毁，但是程序没有退出。因为点击No按钮的时候PostQuitMessage函数根本就没有执行，程序不会退出，这个坑需要注意。

## 还有一个问题，就是在switch最后的default语句里面的默认窗口过程的代码必须写，如果注释了，窗口不能显示

|  |
| --- |
|  |

## 但是程序正在运行

|  |
| --- |
|  |

## 这个坑也是需要注意的。

# WndProc函数的定义中的CALLBACK是什么意思

## 可以在CALLBACK上面点击右键-》转到定义，内容如下，

|  |
| --- |
|  |

### 他其实就是一个\_\_stdcall,是一个标准调用约定他和c语言的调用约定\_\_cdecl是不一样的。他们的不同主要体现在参数传递顺序右键堆栈的清除方式

# 调用约定其实是可以设置的，vs2010的设置方法点击项目-》属性-》c/c++ -》高级，第一栏就可以设置了

|  |
| --- |
|  |