# 这一节我们学习一种使用对话框资源创建窗口的方法

|  |
| --- |
|  |

## 这个实例需要用到一个叫做DialogBox的API，用法如下

|  |
| --- |
| C++  void DialogBox(  [in, optional] **HINSTANCE** hInstance,  [in] **LPCTSTR**lpTemplate,  [in, optional] **HWND** hWndParent,  [in, optional] **DLGPROC** lpDialogFunc  ); 参数 [in, optional] hInstance  类型：**HINSTANCE**  包含对话框模板的模块的句柄。如果此参数为 NULL，则使用当前可执行文件。  [in] lpTemplate  类型： **LPCTSTR**  对话框模板。此参数是指向指定对话框模板名称的以 null 结尾的字符串的指针，或者是指定对话框模板的资源标识符的整数值。如果参数指定了资源标识符，则其高序词必须为零，并且其低序词必须包含该标识符。您可以使用 [MAKEINTRESOURCE](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-makeintresourcea) 宏创建此值。  [in, optional] hWndParent  类型： **HWND**  拥有对话框的窗口的句柄。  [in, optional] lpDialogFunc  类型：**DLGPROC**  指向对话框过程的指针。有关对话框过程的详细信息，请参阅 [DialogProc](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nc-winuser-dlgproc)。 返回值 没有 言论 **DialogBox** 宏使用 [CreateWindowEx](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-createwindowexa) 函数创建对话框。然后**，DialogBox** 向对话框过程发送[一条WM\_INITDIALOG](https://learn.microsoft.com/en-us/windows/desktop/dlgbox/wm-initdialog)消息（如果模板指定了 [DS\_SETFONT](https://learn.microsoft.com/en-us/windows/desktop/dlgbox/about-dialog-boxes) 或 DS\_SHELLFONT 样式，则发送[一条WM\_SETFONT](https://learn.microsoft.com/en-us/windows/desktop/winmsg/wm-setfont)消息）。该函数显示对话框（无论模板是否指定了**WS\_VISIBLE**样式），禁用所有者窗口，并启动自己的消息循环来检索和调度对话框的消息。  当对话框过程调用 [EndDialog](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-enddialog) 函数时，**DialogBox** 会销毁对话框，结束消息循环，启用所有者窗口（如果以前已启用），并返回对话框过程在调用 **EndDialog** 时指定的 *nResult* 参数。 |

## 这个函数需要一个对话框很多函数DLGPROC，用法如下

|  |
| --- |
| C++复制  DLGPROC Dlgproc;  INT\_PTR Dlgproc(  HWND hwnd,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  )  {...} 参数 Hwnd  类型： **HWND**  对话框的句柄。  uMsg  类型： **UINT**  消息。  wParam  类型：**WPARAM**  其他特定于消息的信息。  lParam  类型： **LPARAM**  其他特定于消息的信息。  类型： **INT\_PTR**  通常，如果对话框过程处理了消息，则应返回 **TRUE**，如果未处理，则应返回 **FALSE**。如果对话框过程返回 **FALSE，**则对话框管理器将执行默认对话框操作以响应该消息。  如果对话框过程处理需要特定返回值的消息，则对话框过程应在返回 **TRUE** 之前立即调用 [SetWindowLong](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-setwindowlonga)（*hwndDlg*， **DWL\_MSGRESULT**， *lResult*） 来设置所需的返回值。请注意，必须在返回 **TRUE** 之前立即调用 **SetWindowLong**;提前执行此操作可能会导致 **DWL\_MSGRESULT** 值被嵌套的对话框消息覆盖。  以下消息是上述一般规则的例外情况。有关返回值语义的详细信息，请参阅特定消息的文档。   * [WM\_CHARTOITEM](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-chartoitem) * [WM\_COMPAREITEM](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-compareitem) * [WM\_CTLCOLORBTN](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-ctlcolorbtn) * [WM\_CTLCOLORDLG](https://learn.microsoft.com/en-us/windows/desktop/dlgbox/wm-ctlcolordlg) * [WM\_CTLCOLOREDIT](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-ctlcoloredit) * [WM\_CTLCOLORLISTBOX](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-ctlcolorlistbox) * [WM\_CTLCOLORSCROLLBAR](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-ctlcolorscrollbar) * [WM\_CTLCOLORSTATIC](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-ctlcolorstatic) * [WM\_INITDIALOG](https://learn.microsoft.com/en-us/windows/desktop/dlgbox/wm-initdialog) * [WM\_QUERYDRAGICON](https://learn.microsoft.com/en-us/windows/desktop/winmsg/wm-querydragicon) * [WM\_VKEYTOITEM](https://learn.microsoft.com/en-us/windows/desktop/Controls/wm-vkeytoitem)  返回值 没有 言论 仅当使用对话框的对话框类时，才应使用对话框过程。这是默认类，当对话框模板中未指定显式类时使用。尽管对话框过程类似于窗口过程，但它不得调用 [DefWindowProc](https://learn.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-defwindowproca) 函数来处理不需要的消息。不需要的消息由对话框窗口过程在内部处理。 |

# 步骤如下

## 1，用vs2008新建一个c++常规空项目，取名：Lesson7-create-win-with-dlg，点击完成

|  |
| --- |
|  |

## 2.先添加个源文件如lesson7.cpp,然后添加主函数定义

|  |
| --- |
|  |

## 3.然后在资源文件夹里面点击右键->添加-资源，会打开一个对话框，如图

|  |
| --- |
|  |
|  |

## 4.选中Dialog，然后点击新建，会创建一个对话框资源

|  |
| --- |
|  |

## 5.然后我可以根据需要来编辑这个资源，如：修改标题，删除一些按钮

|  |
| --- |
|  |

## 6.我们修改一下对话框的ID，把它改为IDD\_MAIN,然后添加一个图片控件和2给按钮

|  |
| --- |
|  |

## 7.然后我们导入2个图片作为Bitmap资源

|  |
| --- |
|  |

## 8.此时vs2008会自动生成一个resource.h文件，里面有我们的资源的标识定义代码。也生成了一个.rc文件，不过我们最好不要动它，我们需要在我们的cpp文件里面包含整个头文件。

|  |
| --- |
|  |

## 9.然后我们就可以使用我们定义的对话框ID来创建对话框，在创建对话框之前，我们需要先定义一个对话框回调函数，在主函数之前声明，在主函数之后定义

|  |
| --- |
|  |

### 注意：这里有一个问题，为什么我们点击对话框说明的x不能关闭对话框？因为我用错API，关闭对话框的API是EndDialog(对话框句柄，0)

### 对话框窗口过程的代码也可以这么,这样子比较符合规范

|  |
| --- |
| INT\_PTR CALLBACK Dlgproc(  HWND hwnd,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  )  {  BOOL ret = TRUE;  int wmId = LOWORD(wParam);  switch(uMsg)  {  case WM\_COMMAND:  switch(wmId)  {  case IDC\_BUTTON1:  MessageBox(hwnd,"Button1 Clicked","info",0);  break;  case IDC\_BUTTON2:  MessageBox(hwnd,"Button2 Clicked","info",0);  break;  }  break;  case WM\_CLOSE:  EndDialog(hwnd,0);  break;  default:  ret = FALSE;  }  return ret;  } |

### 意思就是：自己感兴趣的消息自己处理，不感兴趣的交给对话框内部处理，注意在这里不能使用DefWindowProc这是和普通文档视图程序不一样的地方

## 10.可以添加鼠标左键按下消息响应代码

|  |
| --- |
| INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  )  {  BOOL ret = TRUE;  int wmId = LOWORD(wParam);  switch(uMsg)  {  case WM\_COMMAND:  switch(wmId)  {  case IDC\_BUTTON1:  MessageBox(hDlg,"Button1 Clicked","info",0);  break;  case IDC\_BUTTON2:  MessageBox(hDlg,"Button2 Clicked","info",0);  break;  }  break;  case WM\_LBUTTONDOWN:  MessageBox(hDlg,"Mouse Clicked","info",0);  break;  case WM\_CLOSE:  EndDialog(hDlg,0);  break;  default:  ret = FALSE;  }  return ret;  } |

# 这一节的学习就到此为止，Lesson7.cpp的完整代码如下

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  //对话框回调函数  INT\_PTR CALLBACK Dlgproc(  HWND hwnd,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  );  int APIENTRY WinMain(HINSTANCE hinstance,HINSTANCE prev,LPSTR lpCmdLine,int nCmdShow)  {  DialogBox(hinstance,MAKEINTRESOURCE(IDD\_MAIN),NULL,(DLGPROC)Dlgproc);  return 0;  }  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  )  {  BOOL ret = TRUE;  int wmId = LOWORD(wParam);  switch(uMsg)  {  case WM\_COMMAND:  switch(wmId)  {  case IDC\_BUTTON1:  MessageBox(hDlg,"Button1 Clicked","info",0);  break;  case IDC\_BUTTON2:  MessageBox(hDlg,"Button2 Clicked","info",0);  break;  }  break;  case WM\_LBUTTONDOWN:  MessageBox(hDlg,"Mouse Clicked","info",0);  break;  case WM\_CLOSE:  EndDialog(hDlg,0);  break;  default:  ret = FALSE;  }  return ret;  } |

### 这是一种比较多人使用的方法。

# 扩展：我们可以测试当用户点击第一个按钮时把第一幅问题显示到图片控件上面，点击第二个按钮时把第二幅位图显示中图片控件上面

### 具体代码如下

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  //对话框回调函数  INT\_PTR CALLBACK Dlgproc(  HWND hwnd,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  );  HBITMAP bitmap1,bitmap2,com\_bit,bitOld1,bitOld2;  HINSTANCE hinDlg;  int APIENTRY WinMain(HINSTANCE hinstance,HINSTANCE prev,LPSTR lpCmdLine,int nCmdShow)  {  hinDlg = hinstance;  DialogBox(hinstance,MAKEINTRESOURCE(IDD\_MAIN),NULL,(DLGPROC)Dlgproc);  return 0;  }  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  )  {  HWND hwndPic;  HDC memDC,hdcPic;  HBITMAP bitmap1,bitmap2;  BITMAP bm;  BOOL ret = TRUE;  hwndPic = GetDlgItem(hDlg,IDC\_PIC);  RECT rc;  COLORREF bkColor;  HBRUSH hBrush;  int wmId = LOWORD(wParam);  switch(uMsg)  {  case WM\_COMMAND:  switch(wmId)  {  case IDC\_BUTTON1:  //MessageBox(hDlg,"Button1 Clicked","info",0);  hdcPic = GetDC(hwndPic);  memDC = CreateCompatibleDC(hdcPic);  /\*  先把原来的图片擦除  \*/  hBrush = CreateSolidBrush(RGB(255,255,255));  GetClientRect(hwndPic,&rc);  FillRect(hdcPic,&rc,hBrush);  bitmap1 = LoadBitmap(hinDlg,MAKEINTRESOURCE(IDB\_BITMAP1));  if(!bitmap1)  {  MessageBox(hDlg,"Load Bitmap Failed","error",0);  }  GetObject(bitmap1,sizeof(bm),&bm);  SelectObject(memDC,bitmap1);  GetClientRect(hwndPic,&rc);  //BitBlt(hdcPic,rc.left,rc.top,bm.bmWidth,bm.bmHeight,memDC,0,0,SRCCOPY);  BitBlt(hdcPic,rc.left,rc.top,rc.right,rc.bottom,memDC,0,0,SRCCOPY);  DeleteDC(memDC);  DeleteDC(hdcPic);  break;  case IDC\_BUTTON2:  //MessageBox(hDlg,"Button2 Clicked","info",0);  hdcPic = GetDC(hwndPic);  memDC = CreateCompatibleDC(hdcPic);  /\*  先把原来的图片擦除  \*/  hBrush = CreateSolidBrush(RGB(255,255,255));  GetClientRect(hwndPic,&rc);  FillRect(hdcPic,&rc,hBrush);  //再输出新图片  bitmap2 = LoadBitmap(hinDlg,MAKEINTRESOURCE(IDB\_BITMAP2));  if(!bitmap2)  {  MessageBox(hDlg,"Load Bitmap Failed","error",0);  }  GetObject(bitmap2,sizeof(bm),&bm);  SelectObject(memDC,bitmap2);  GetClientRect(hwndPic,&rc);  BitBlt(hdcPic,rc.left,rc.top,rc.right,rc.bottom,memDC,0,0,SRCCOPY);  DeleteDC(memDC);  DeleteDC(hdcPic);  break;  }  break;  case WM\_LBUTTONDOWN:  MessageBox(hDlg,"Mouse Clicked","info",0);  break;  case WM\_CLOSE:  EndDialog(hDlg,0);  break;  default:  ret = FALSE;  }  return ret;  } |

### 效果

|  |
| --- |
|  |
|  |

### 注意两个按钮代码的实现，这里需要解决一个问题，就是每一次点击按钮，首先需要做的就是把之前的图片擦除，然后再显示新图片，否则两张图片叠加在一起好不好看。擦除的办法就是使用FillRect函数填充图片框，使用白色刷子。

## BitBlt函数的语法如下

|  |
| --- |
| BOOL BitBlt(  [in] HDC hdc,  [in] int x,  [in] int y,  [in] int cx,  [in] int cy,  [in] HDC hdcSrc,  [in] int x1,  [in] int y1,  [in] DWORD rop  ); 参数 [in] hdc  目标设备上下文的句柄。  [in] x  目标矩形左上角的 x 坐标（以逻辑单位表示）。  [in] y  目标矩形左上角的 y 坐标（以逻辑单位表示）。  [in] cx  源矩形和目标矩形的宽度（以逻辑单位表示）。  [in] cy  源矩形和目标矩形的高度（以逻辑单位表示）。  [in] hdcSrc  源设备上下文的句柄。  [in] x1  源矩形左上角的 x 坐标（以逻辑单位表示）。  [in] y1  源矩形左上角的 y 坐标（以逻辑单位表示）。  [in] rop  光栅操作代码。这些代码定义如何将源矩形的颜色数据与目标矩形的颜色数据相结合，以获得最终颜色。 |

### 以下列表显示了一些常见的栅格操作代码。

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| **BLACKNESS** | 使用与物理调色板中的索引 0 关联的颜色填充目标矩形。（对于默认物理调色板，此颜色为黑色。 |
| **CAPTUREBLT** | 包括结果图像中位于窗口顶部的任何窗口。默认情况下，图像仅包含您的窗口。请注意，这通常不能用于打印设备上下文。 |
| **DSTINVERT** | 反转目标矩形。 |
| **MERGECOPY** | 通过使用布尔 AND 运算符，将源矩形的颜色与当前在 *hdcDest* 中选择的画笔合并。 |
| **MERGEPAINT** | 使用布尔 OR 运算符将反转源矩形的颜色与目标矩形的颜色合并。 |
| **NOMIRRORBITMAP** | 防止位图被镜像。 |
| **NOTSRCCOPY** | 将反转的源矩形复制到目标。 |
| **NOTSRCERASE** | 使用布尔 OR 运算符组合源矩形和目标矩形的颜色，然后反转结果颜色。 |
| **PATCOPY** | 将当前在 *hdcDest* 中选择的画笔复制到目标位图中 |
| **PATINVERT** | 使用布尔 XOR 运算符将当前在 *hdcDest* 中选择的画笔的颜色与目标矩形的颜色组合在一起 |
| **PATPAINT** | 通过使用布尔 OR 运算符，将当前在 *hdcDest* 中选择的画笔的颜色与反转源矩形的颜色组合在一起。此操作的结果通过使用布尔 OR 运算符与目标矩形的颜色相结合. |
| **SRCAND** | 使用布尔 AND 运算符组合源矩形和目标矩形的颜色。 |
| **SRCCOPY** | 将源矩形直接复制到目标矩形。 |
| **SRCERASE** | 使用布尔 AND 运算符将目标矩形的反转颜色与源矩形的颜色组合在一起。 |
| **SRCINVERT** | 使用布尔 XOR 运算符组合源矩形和目标矩形的颜色 |
| **SRCPAINT** | 使用布尔 OR 运算符组合源矩形和目标矩形的颜色 |
| **WHITENESS** | 使用与物理调色板中的索引 1 关联的颜色填充目标矩形。（对于默认物理调色板，此颜色为白色。 |

## Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror).

## Remarks

**BitBlt** only does clipping on the destination DC.

If a rotation or shear transformation is in effect in the source device context, **BitBlt** returns an error. If other transformations exist in the source device context (and a matching transformation is not in effect in the destination device context), the rectangle in the destination device context is stretched, compressed, or rotated, as necessary.

If the color formats of the source and destination device contexts do not match, the **BitBlt** function converts the source color format to match the destination format.

When an enhanced metafile is being recorded, an error occurs if the source device context identifies an enhanced-metafile device context.

Not all devices support the **BitBlt** function. For more information, see the RC\_BITBLT raster capability entry in the [GetDeviceCaps](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-getdevicecaps) function as well as the following functions: [MaskBlt](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-maskblt), [PlgBlt](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-plgblt), and [StretchBlt](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-stretchblt).

**BitBlt** returns an error if the source and destination device contexts represent different devices. To transfer data between DCs for different devices, convert the memory bitmap to a DIB by calling [GetDIBits](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-getdibits). To display the DIB to the second device, call [SetDIBits](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-setdibits) or [StretchDIBits](https://learn.microsoft.com/en-us/windows/desktop/api/wingdi/nf-wingdi-stretchdibits).

## 也可以尝试使用StretchBlt函数，语法如下

|  |
| --- |
|  |

### 用法和BitBlt类似就是多了两个参数，原图片的宽高

## 修改后的代码

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  //对话框回调函数  INT\_PTR CALLBACK Dlgproc(  HWND hwnd,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  );  HBITMAP bitmap1,bitmap2,com\_bit,bitOld1,bitOld2;  HINSTANCE hinDlg;  int APIENTRY WinMain(HINSTANCE hinstance,HINSTANCE prev,LPSTR lpCmdLine,int nCmdShow)  {  hinDlg = hinstance;  DialogBox(hinstance,MAKEINTRESOURCE(IDD\_MAIN),NULL,(DLGPROC)Dlgproc);  return 0;  }  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT uMsg,  WPARAM wParam,  LPARAM lParam  )  {  HWND hwndPic;  HDC memDC,hdcPic;  HBITMAP bitmap1,bitmap2;  BITMAP bm;  BOOL ret = TRUE;  hwndPic = GetDlgItem(hDlg,IDC\_PIC);  RECT rc;  COLORREF bkColor;  HBRUSH hBrush;  int wmId = LOWORD(wParam);  switch(uMsg)  {  case WM\_COMMAND:  switch(wmId)  {  case IDC\_BUTTON1:  //MessageBox(hDlg,"Button1 Clicked","info",0);  hdcPic = GetDC(hwndPic);  memDC = CreateCompatibleDC(hdcPic);  /\*  先把原来的图片擦除  \*/  hBrush = CreateSolidBrush(RGB(255,255,255));  GetClientRect(hwndPic,&rc);  FillRect(hdcPic,&rc,hBrush);  bitmap1 = LoadBitmap(hinDlg,MAKEINTRESOURCE(IDB\_BITMAP1));  if(!bitmap1)  {  MessageBox(hDlg,"Load Bitmap Failed","error",0);  }  GetObject(bitmap1,sizeof(bm),&bm);  SelectObject(memDC,bitmap1);  GetClientRect(hwndPic,&rc);  //BitBlt(hdcPic,rc.left,rc.top,bm.bmWidth,bm.bmHeight,memDC,0,0,SRCCOPY);  //BitBlt(hdcPic,rc.left,rc.top,rc.right,rc.bottom,memDC,0,0,SRCCOPY);  StretchBlt(hdcPic,rc.left,rc.top,rc.right-rc.left,rc.bottom-rc.top,memDC,0,0,bm.bmWidth,bm.bmHeight ,SRCCOPY);  DeleteDC(memDC);  DeleteDC(hdcPic);  break;  case IDC\_BUTTON2:  //MessageBox(hDlg,"Button2 Clicked","info",0);  hdcPic = GetDC(hwndPic);  memDC = CreateCompatibleDC(hdcPic);  /\*  先把原来的图片擦除  \*/  hBrush = CreateSolidBrush(RGB(255,255,255));  GetClientRect(hwndPic,&rc);  FillRect(hdcPic,&rc,hBrush);  //再输出新图片  bitmap2 = LoadBitmap(hinDlg,MAKEINTRESOURCE(IDB\_BITMAP2));  if(!bitmap2)  {  MessageBox(hDlg,"Load Bitmap Failed","error",0);  }  GetObject(bitmap2,sizeof(bm),&bm);  SelectObject(memDC,bitmap2);  GetClientRect(hwndPic,&rc);    //BitBlt(hdcPic,rc.left,rc.top,rc.right,rc.bottom,memDC,0,0,SRCCOPY);  StretchBlt(hdcPic,rc.left,rc.top,rc.right-rc.left,rc.bottom-rc.top,memDC,0,0,bm.bmWidth,bm.bmHeight ,SRCCOPY);  DeleteDC(memDC);  DeleteDC(hdcPic);  break;  }  break;  case WM\_LBUTTONDOWN:  MessageBox(hDlg,"Mouse Clicked","info",0);  break;  case WM\_CLOSE:  EndDialog(hDlg,0);  break;  default:  ret = FALSE;  }  return ret;  } |

### 效果

|  |
| --- |
|  |
|  |

### 如果图片够清晰的话效果比较好。可以填满整个图片框