# 这一节我们来学习编辑类控件

# 学习大纲

|  |
| --- |
|  |
|  |
|  |
|  |

## SetWindowText函数用法

|  |
| --- |
| 如果指定窗口的标题栏有一个) ，则更改 (的文本。 如果指定的窗口是控件，则会更改控件的文本。 但是， **SetWindowText** 无法更改另一个应用程序中控件的文本。 语法 C++复制  BOOL SetWindowTextA(  [in] HWND hWnd,  [in, optional] LPCSTR lpString  ); 参数 [in] hWnd  类型：**HWND**  要更改其文本的窗口或控件的句柄。  [in, optional] lpString  类型： **LPCTSTR**  新的标题或控件文本。 返回值 类型： **BOOL**  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。 要获得更多的错误信息，请调用 GetLastError。 注解 如果目标窗口由当前进程拥有， **SetWindowText** 会导致 [将WM\_SETTEXT](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-settext) 消息发送到指定的窗口或控件。 但是，如果控件是使用 **WS\_CAPTION** 样式创建的列表框控件， **则 SetWindowText** 将设置控件的文本，而不是列表框条目的文本。  若要在另一个进程中设置控件的文本，请直接发送 [WM\_SETTEXT](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-settext) 消息，而不是调用 **SetWindowText**。  **SetWindowText** 函数不会 (ASCII 代码0x09) 展开制表符。 制表符显示为垂直条 (|) 字符。 示例 有关示例，请参阅 [发送消息](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/using-messages-and-message-queues)。  **备注**  winuser.h 标头将 SetWindowText 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)。 |

## GetWindowText函数用法

|  |
| --- |
| 如果指定窗口的标题栏有一个) 到缓冲区中，则复制指定窗口标题栏的文本 (。 如果指定的窗口是控件，则复制控件的文本。 但是， **GetWindowText** 无法检索另一个应用程序中控件的文本。 语法 C++复制  int GetWindowTextA(  [in] HWND hWnd,  [out] LPSTR lpString,  [in] int nMaxCount  ); 参数 [in] hWnd  类型：**HWND**  包含文本的窗口或控件的句柄。  [out] lpString  类型： **LPTSTR**  将接收文本的缓冲区。 如果字符串长或长于缓冲区，则字符串将被截断并终止为 null 字符。  [in] nMaxCount  类型： **int**  要复制到缓冲区的最大字符数，包括 null 字符。 如果文本超出此限制，则会将其截断。 返回值 类型： **int**  如果函数成功，则返回值为复制的字符串的长度（以字符为单位），不包括终止 null 字符。 如果窗口没有标题栏或文本，如果标题栏为空，或者窗口或控件句柄无效，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。  此函数无法检索另一个应用程序中编辑控件的文本。 注解 如果目标窗口由当前进程拥有， **则 GetWindowText** 将导致 [将WM\_GETTEXT](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-gettext) 消息发送到指定的窗口或控件。 如果目标窗口由另一个进程拥有，并且具有描述文字，**则 GetWindowText 将**检索描述文字文本的窗口。 如果窗口没有描述文字，则返回值为 null 字符串。 此行为是设计使然。 如果拥有目标窗口的进程没有响应，它允许应用程序调用 **GetWindowText** ，而不会变得无响应。 但是，如果目标窗口没有响应，并且它属于调用应用程序， **则 GetWindowText** 将导致调用应用程序变得无响应。  若要在另一个进程中检索控件的文本，请直接发送 [WM\_GETTEXT](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-gettext) 消息，而不是调用 **GetWindowText**。 示例 以下示例代码演示了对 **GetWindowTextA 的**调用。  C++复制  hwndCombo = GetDlgItem(hwndDlg, IDD\_COMBO);  cTxtLen = GetWindowTextLength(hwndCombo);  // Allocate memory for the string and copy  // the string into the memory.  pszMem = (PSTR) VirtualAlloc((LPVOID) NULL,  (DWORD) (cTxtLen + 1), MEM\_COMMIT,  PAGE\_READWRITE);  GetWindowText(hwndCombo, pszMem,  cTxtLen + 1);  若要在上下文中查看此示例，请参阅 [发送邮件](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/using-messages-and-message-queues)。  **备注**  winuser.h 标头将 GetWindowText 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)。 |

## SendMessage函数用法

|  |
| --- |
| 将指定的消息发送到一个或多个窗口。 **SendMessage** 函数调用指定窗口的窗口过程，在窗口过程处理消息之前不会返回 。  若要发送消息并立即返回，请使用 [SendMessageCallback](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-sendmessagecallbacka) 或 [SendNotifyMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-sendnotifymessagea) 函数。 若要将消息发布到线程的消息队列并立即返回，请使用 [PostMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-postmessagea) 或 [PostThreadMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-postthreadmessagea) 函数。 语法 C++复制  LRESULT SendMessage(  [in] HWND hWnd,  [in] UINT Msg,  [in] WPARAM wParam,  [in] LPARAM lParam  ); 参数 [in] hWnd  类型：**HWND**  窗口的句柄，其窗口过程将接收消息。 如果此参数 **HWND\_BROADCAST** ( (HWND) 0xffff) ，则消息将发送到系统中的所有顶级窗口，包括禁用或不可见的无所有者窗口、重叠窗口和弹出窗口;但消息不会发送到子窗口。  消息发送受 UIPI 约束。 进程线程只能将消息发送到完整性级别较低或相等进程的线程的消息队列。  [in] Msg  类型： **UINT**  要发送的消息。  有关系统提供的消息的列表，请参阅 [系统定义的消息](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/about-messages-and-message-queues)。  [in] wParam  类型：WPARAM  其他的消息特定信息。  [in] lParam  类型：LPARAM  其他的消息特定信息。 返回值 类型： **LRESULT**  返回值指定消息处理的结果;这取决于发送的消息。 注解 当消息被 UIPI 阻止时，使用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 检索的最后一个错误设置为 5 (拒绝访问) 。  需要使用 **HWND\_BROADCAST** 进行通信的应用程序应使用 [RegisterWindowMessage](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-registerwindowmessagea) 函数获取应用程序间通信的唯一消息。  系统仅对系统消息执行封送处理， (范围为 0 到 ([WM\_USER-1](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/wm-user)) ) 。 若要将其他消息 (那些 >= **WM\_USER**) 发送到另一个进程，必须执行自定义封送处理。  如果指定的窗口是由调用线程创建的，则窗口过程将立即作为子例程调用。 如果指定的窗口是由其他线程创建的，则系统会切换到该线程并调用相应的窗口过程。 仅在接收线程执行消息检索代码时，才会处理线程之间发送的消息。 在接收线程处理消息之前，将阻止发送线程。 但是，发送线程将在等待处理其消息时处理传入的非排队消息。 若要防止这种情况，请将 [SendMessageTimeout](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-sendmessagetimeouta) 与SMTO\_BLOCK集结合使用。 有关非排队消息的详细信息，请参阅 [非排队消息](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/about-messages-and-message-queues)。  辅助功能应用程序可以使用 **SendMessage** 将 [WM\_APPCOMMAND](https://learn.microsoft.com/zh-cn/windows/desktop/inputdev/wm-appcommand) 消息发送到 shell 以启动应用程序。 此功能不保证适用于其他类型的应用程序。 示例 有关示例，请参阅 [显示键盘输入](https://learn.microsoft.com/zh-cn/windows/desktop/inputdev/using-keyboard-input)。 |

# 演练

## 1，新建有文件夹Lesson27，在里面新建一个常规空项目，然后加载对话框资源手动创建对话框应用程序，起名：ctrldemo1，然后右击资源文件夹-》添加-资源-Dialog添加一个对话框资源

|  |
| --- |
|  |

## 2.把取消按钮删除，然后添加一个组框，标题是：控件使用演示，然后在里面添加2个编辑框控件，改为多行，有水平和垂直滚动条

|  |
| --- |
|  |

## 3.新建一个cpp源文件，取名ctrldemo1.cpp,需要包含Windows.h和开发工具自动生成的resources.h,先在WinMain函数外面对应关于对话框窗口过程函数DlgProc，这是一个回调函数，然后在WinMain函数里面创建一个对话框窗口，不这个DlgProc函数作为最后一个参数

|  |
| --- |
|  |

## 4.然后外面在DlgProc函数里面代码代码，我们先添加用户点击x按钮也就是WM\_CLOSE消息的处理代码和IDOK也就是用户点击退出按钮的事件处理代码

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT message,  WPARAM wParam,  LPARAM lParam  )  {  BOOL Ret = TRUE;  int cmdId;  switch(message)  {  case WM\_COMMAND:  cmdId = LOWORD(wParam);  switch(cmdId)  {  case IDOK:  if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  {  //PostQuitMessage(0);  EndDialog(hDlg,0);  }  break;  case IDC\_BTN\_CLEAR:  break;  }  break;  case WM\_CLOSE:  if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  {  //PostQuitMessage(0);  EndDialog(hDlg,0);  }  break;  default:  Ret = FALSE;  break;  }  return Ret;  }  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmdLine,int nCmdShow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,(DLGPROC)Dlgproc);  return 0;  } |

### 效果，点击或者退出按钮，会弹出确认消息框，点击确定退出程序

|  |
| --- |
|  |

## 5.然后我们来编写清除按钮的代码，可以使用SendMessage函数来做，不过它只能够清除选中是内容，我们这里就是这个意思，如果想清空全部，可以使用SetWindowText(“”)

|  |
| --- |
|  |

## 6.编辑框控件也能够向父窗口发送消息，我们来处理一下第一个编辑框的消息，为了方便看到结果，我们添加两个静态文本，一个的内容是：Edit1状态，另外一个没有文本，设置边框属性，把它的ID设置为IDC\_STATIC\_STATE,当编辑框1的状态发生改变，我们就可以设置有边框的静态文本的内容为编辑框的状态

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT message,  WPARAM wParam,  LPARAM lParam  )  {  static HWND hwndEdit1,hwndEdit2,hwndStatic; //有坑，这里使用静态变量  BOOL Ret = TRUE;  int cmdId;  hwndEdit1 = GetDlgItem(hDlg,IDC\_EDIT1);//获取第1个编辑框的句柄  hwndEdit2 = GetDlgItem(hDlg,IDC\_EDIT2);//获取第2个编辑框的句柄  hwndStatic = GetDlgItem(hDlg,IDC\_STATIC\_STATE);  TCHAR szContent[MAX\_PATH];  switch(message)  {  case WM\_COMMAND:  cmdId = LOWORD(wParam);  switch(cmdId)  {  case IDC\_EDIT1:  switch(HIWORD(wParam))  {  case EN\_SETFOCUS://不要再获取焦点时使用MessageBox函数，会得不到焦点  SetWindowText(hwndStatic,"得到焦点");  break;  case EN\_KILLFOCUS:  SetWindowText(hwndStatic,"失去焦点");  break;  case EN\_UPDATE:  GetWindowText(hwndEdit1,szContent,MAX\_PATH);  //SetWindowText(hDlg,szContent);  SetWindowText(hwndEdit2,szContent);  SetWindowText(hwndStatic,"文本更新");  break;  }  break;  case IDOK:  if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  {  //PostQuitMessage(0);  EndDialog(hDlg,0);  }    break;  case IDC\_BTN\_CLEAR:  //MessageBox(hDlg,"button clicked","info",MB\_OKCANCEL);  SendMessage(hwndEdit1,WM\_CLEAR,0,0);//清除选中的内容  //SetWindowText(hwndEdit1,TEXT(" ")); //清除所有内容  SetWindowText(hDlg,"Dialog");  break;  default:  Ret = FALSE;  break;  }  break;  case WM\_CLOSE:  //if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  //{  // //PostQuitMessage(0);  // EndDialog(hDlg,0);  //}  //这里不需要询问，因为用都已经点击了x，他是想退出程序  EndDialog(hDlg,0);  break;  default:  Ret = FALSE;  break;  }  return Ret;  }  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmdLine,int nCmdShow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,(DLGPROC)Dlgproc);  return 0;  } |

## 7.下面，我们再添加2个按钮来学习一下复制和粘贴功能，添加两个按钮，一个复制一个粘贴，ID分别是IDC\_BTN\_COPY和IDC\_BTN\_PASTE

|  |
| --- |
|  |

## 8.然后我们添加两个按钮的点击事件代码，需要在WM\_COMMAND处理代码中获取他们的ID然后根据不同的id做不同的操作，注意只有选中的文本才能复制粘贴

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT message,  WPARAM wParam,  LPARAM lParam  )  {  static HWND hwndEdit1,hwndEdit2,hwndStatic; //有坑，这里使用静态变量  BOOL Ret = TRUE;  int cmdId;  hwndEdit1 = GetDlgItem(hDlg,IDC\_EDIT1);//获取第1个编辑框的句柄  hwndEdit2 = GetDlgItem(hDlg,IDC\_EDIT2);//获取第2个编辑框的句柄  hwndStatic = GetDlgItem(hDlg,IDC\_STATIC\_STATE);  TCHAR szContent[MAX\_PATH];  switch(message)  {  case WM\_COMMAND:  cmdId = LOWORD(wParam);  switch(cmdId)  {  case IDC\_EDIT1:  switch(HIWORD(wParam))  {  case EN\_SETFOCUS://不要再获取焦点时使用MessageBox函数，会得不到焦点  SetWindowText(hwndStatic,"得到焦点");  break;  case EN\_KILLFOCUS:  SetWindowText(hwndStatic,"失去焦点");  break;  case EN\_UPDATE:  //GetWindowText(hwndEdit1,szContent,MAX\_PATH);  //SetWindowText(hDlg,szContent);  //SetWindowText(hwndEdit2,szContent);  SetWindowText(hwndStatic,"文本更新");  break;  }  break;  case IDOK:  if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  {  //PostQuitMessage(0);  EndDialog(hDlg,0);  }    break;  case IDC\_BTN\_CLEAR:  //MessageBox(hDlg,"button clicked","info",MB\_OKCANCEL);  SendMessage(hwndEdit1,WM\_CLEAR,0,0);//清除选中的内容  //SetWindowText(hwndEdit1,TEXT(" ")); //清除所有内容  SetWindowText(hDlg,"Dialog");  break;  case IDC\_BTN\_COPY:  switch(HIWORD(wParam))  {  case BN\_CLICKED://拷贝第一个文本框的内容  SendMessage(hwndEdit1,WM\_COPY,0,0);  break;  default:  Ret = FALSE;  }  break;  case IDC\_BTN\_PASTE:  switch(HIWORD(wParam))  {  case BN\_CLICKED://粘贴到第二个文本框里面  SendMessage(hwndEdit2,WM\_PASTE,0,0);  break;  default:  Ret = FALSE;  }  break;  default:  Ret = FALSE;  break;  }  break;  case WM\_CLOSE:  //if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  //{  // //PostQuitMessage(0);  // EndDialog(hDlg,0);  //}  //这里不需要询问，因为用都已经点击了x，他是想退出程序  EndDialog(hDlg,0);  break;  default:  Ret = FALSE;  break;  }  return Ret;  }  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmdLine,int nCmdShow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,(DLGPROC)Dlgproc);  return 0;  } |

## 9.然后我们可以再添加两个按钮，一个获取文本ID是IDC\_BTN\_GET，一个设置文本ID是IDC\_BTN\_SET

|  |
| --- |
|  |

## 10.然后方便给他们添加处理代码，注意需要把保存文本的变量设置为全局变量，否则可能有乱码

|  |
| --- |
| #include<Windows.h>  #include"resource.h"  TCHAR szContent[MAX\_PATH]; //保存文本内容的全局变量，方便获取文本和设置文本操作  INT\_PTR CALLBACK Dlgproc(  HWND hDlg,  UINT message,  WPARAM wParam,  LPARAM lParam  )  {  static HWND hwndEdit1,hwndEdit2,hwndStatic; //有坑，这里使用静态变量  BOOL Ret = TRUE;  int cmdId;  hwndEdit1 = GetDlgItem(hDlg,IDC\_EDIT1);//获取第1个编辑框的句柄  hwndEdit2 = GetDlgItem(hDlg,IDC\_EDIT2);//获取第2个编辑框的句柄  hwndStatic = GetDlgItem(hDlg,IDC\_STATIC\_STATE);    switch(message)  {  case WM\_COMMAND:  cmdId = LOWORD(wParam);  switch(cmdId)  {  case IDC\_EDIT1:  switch(HIWORD(wParam))  {  case EN\_SETFOCUS://不要再获取焦点时使用MessageBox函数，会得不到焦点  SetWindowText(hwndStatic,"得到焦点");  break;  case EN\_KILLFOCUS:  SetWindowText(hwndStatic,"失去焦点");  break;  case EN\_UPDATE:  //GetWindowText(hwndEdit1,szContent,MAX\_PATH);  //SetWindowText(hDlg,szContent);  //SetWindowText(hwndEdit2,szContent);  SetWindowText(hwndStatic,"文本更新");  break;  }  break;  case IDOK:  if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  {  //PostQuitMessage(0);  EndDialog(hDlg,0);  }    break;  case IDC\_BTN\_CLEAR:  //MessageBox(hDlg,"button clicked","info",MB\_OKCANCEL);  SendMessage(hwndEdit1,WM\_CLEAR,0,0);//清除选中的内容  //SetWindowText(hwndEdit1,TEXT(" ")); //清除所有内容  SetWindowText(hDlg,"Dialog");  break;  case IDC\_BTN\_COPY:  switch(HIWORD(wParam))  {  case BN\_CLICKED://拷贝第一个文本框的内容  SendMessage(hwndEdit1,WM\_COPY,0,0);  break;  default:  Ret = FALSE;  }  break;  case IDC\_BTN\_PASTE:  switch(HIWORD(wParam))  {  case BN\_CLICKED://粘贴到第二个文本框里面  SendMessage(hwndEdit2,WM\_PASTE,0,0);  break;  default:  Ret = FALSE;  }  break;  case IDC\_BTN\_GET:  ZeroMemory(szContent,MAX\_PATH);  GetWindowText(hwndEdit1,szContent,MAX\_PATH);//获取文本需要3个参数  MessageBox(hDlg,szContent,"info",MB\_OKCANCEL);  break;  case IDC\_BTN\_SET:  //SetWindowText(hwndEdit2,szContent);//设置文本只需要2个参数ok  //SetWindowText(hDlg,szContent);  SendMessage(hwndEdit2,WM\_SETTEXT,0,(LPARAM)szContent);//ok  break;  default:  Ret = FALSE;  break;  }  break;  case WM\_CLOSE:  //if(IDOK == MessageBox(hDlg,"退出程序？","确认退出",MB\_OKCANCEL))  //{  // //PostQuitMessage(0);  // EndDialog(hDlg,0);  //}  //这里不需要询问，因为用都已经点击了x，他是想退出程序  EndDialog(hDlg,0);  break;  default:  Ret = FALSE;  break;  }  return Ret;  }  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmdLine,int nCmdShow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,(DLGPROC)Dlgproc);  return 0;  } |