# 这一节我们来学习目录和模块

# 学习大纲

|  |
| --- |
|  |

## CreateDirectory函数的语法如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 语法 BOOL CreateDirectory(  [in] LPCTSTR lpPathName,  [in, optional] LPSECURITY\_ATTRIBUTES lpSecurityAttributes  ); 参数 [in] lpPathName  需要创建的目录的路径。  对于此函数的 ANSI 版本，路径的默认字符串大小限制为 248 个字符（**MAX\_PATH** - 足以容纳 8.3 文件名）。要将此限制扩展到 32,767 个宽字符，请调用该函数的 Unicode 版本并在其前面加上 “\？”有关更多信息，请参阅[命名文件](https://learn.microsoft.com/en-us/windows/desktop/FileIO/naming-a-file)  **提示**从 Windows 10 版本 1607 开始，对于此函数的 Unicode 版本 （**CreateDirectoryW**），可以选择删除 248 个字符的限制，而无需预置“\\？\”。每个路径段 255 个字符的限制仍然适用。有关详细信息[**，请参阅命名文件、路径和命名空间**](https://learn.microsoft.com/en-us/windows/desktop/FileIO/naming-a-file)的“最大路径长度限制”部分。    [in, optional] lpSecurityAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针。结构的 **lpSecurityDescriptor** 成员指定安全性 描述符。如果 *lpSecurityAttributes* 为 **NULL，**则目录将获取默认安全描述符。默认 目录的安全描述符继承自其父目录。  目标文件系统必须支持文件和目录的安全性，此参数才能生效。 （当 [GetVolumeInformation](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getvolumeinformationa) 返回 **FS\_PERSISTENT\_ACLS**.） 返回值 如果函数成功，则返回值为非零。  如果函数失败，则返回值为零。要获取扩展错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。可能的错误包括 以后。  展开表   |  |  | | --- | --- | | **返回代码** | **描述** | | **ERROR\_ALREADY\_EXISTS** | 指定的目录已存在。 | | **ERROR\_PATH\_NOT\_FOUND** | 一个或多个中间目录不存在;此函数只会在 路径。 |  言论 某些文件系统（如 NTFS 文件系统）支持对单个文件进行压缩或加密，并且 目录。在为此类文件系统格式化的卷上，新目录将继承压缩和加密 属性。  应用程序可以通过调用设置了 **FILE\_FLAG\_BACKUP\_SEMANTICS** 标志的 [CreateFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) 来获取目录的句柄。有关代码示例，请参阅 **CreateFile**。 |

## GetCurrentDirectory函数的语法如下

|  |
| --- |
| 语法 C++复制  DWORD GetCurrentDirectory(  [in] DWORD nBufferLength,  [out] LPTSTR lpBuffer  ); 参数 [in] nBufferLength  当前目录字符串的缓冲区长度，以 **TCHAR** 为单位。这 缓冲区长度必须包含用于终止 null 字符的空间。  [out] lpBuffer  指向接收当前目录字符串的缓冲区的指针。此以 null 结尾的字符串指定 当前目录的绝对路径。  要确定所需的缓冲区大小，请将此参数设置为 **NULL**，并将 *nBufferLength* 参数设置为 0。 返回值 如果函数成功，则返回值指定写入缓冲区的字符数。 不包括终止 null 字符。  如果函数失败，则返回值为零。要获取扩展错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。  如果 *lpBuffer* 指向的缓冲区不够大，则返回值 指定缓冲区的所需大小（以字符为单位），包括 null 终止字符。 言论 每个进程都有一个 current 目录，该目录由两部分组成：   * 磁盘指示符，可以是驱动器号后跟冒号，也可以是服务器名称后跟共享 名称 （\\*servername*\*sharename*) * 磁盘指示符上的目录   若要设置当前目录，请使用 [SetCurrentDirectory](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-setcurrentdirectory) 函数。  多线程应用程序和共享库代码不应使用 **GetCurrentDirectory** 函数， 而应避免使用相对路径名称。[SetCurrentDirectory](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-setcurrentdirectory) 函数写入的当前目录状态在每个进程中存储为全局变量，因此多线程应用程序无法可靠地使用此值，而不会因可能也在读取或设置此值的其他线程而损坏数据。此限制也适用于 **SetCurrentDirectory** 和 [GetFullPathName](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getfullpathnamea) 函数。例外情况是，当保证应用程序在单个线程中运行时，例如，在创建任何其他线程之前，从主线程的命令行参数字符串解析文件名。在多线程应用程序或共享库代码中使用相对路径名可能会产生不可预知的结果，因此不受支持。 |

## SetCurrentDirectory函数的语法如下

|  |
| --- |
| 语法 BOOL SetCurrentDirectory(  [in] LPCTSTR lpPathName  ); 参数 [in] lpPathName  新当前目录的路径。此参数可以指定相对路径或完整路径。在任何一种情况下，都会计算指定目录的完整路径并将其存储为当前目录。  有关更多信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  默认情况下，名称限制为 **MAX\_PATH** 个字符。  **提示**  从 Windows 10 版本 1607 开始，您可以选择取消 **MAX\_PATH** 限制。有关详细信息，请参阅[**命名文件、路径和命名空间**](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)的最大路径长度限制部分。  **重要**  将当前目录设置为超过 **MAX\_PATH**会导致 [**CreateProcessW**](https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw) 失败。  null 字符前面的最后一个字符必须是反斜杠 （'\'）。如果您未指定反斜杠，则将为您添加反斜杠。因此，除非包含尾部反斜杠，否则请为路径指定 **>MAX\_PATH-2** 个字符;在这种情况下，请为路径指定 **MAX\_PATH-1** 个字符。 返回值 如果函数成功，则返回值为非零。  如果函数失败，则返回值为零。要获取扩展错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror)。 言论 每个进程都有一个 current 目录，由两部分组成：   * 磁盘指示符，可以是驱动器号后跟冒号，也可以是服务器名称和共享名称 （\\*servername*\*sharename*) * 磁盘指示符上的目录   当前目录由进程的所有线程共享： 如果一个线程更改了当前目录，则会影响所有线程 在这个过程中。  多线程应用程序和共享库代码应避免 调用 **SetCurrentDirectory** 函数，因为存在 影响其他线程执行的相对路径计算。 相反  多线程应用程序和共享库代码应避免 使用相对路径，以便它们不受 current 目录。  **注意**  在执行进程时，进程的当前目录将被锁定。这可以防止目录被删除、移动或重命名。 |

### 使用SetCurrentDirectory来设置当前目录，这个目录必须存在，否则设置失败

## GetModuleFileName函数的语法如下

|  |
| --- |
| GetModuleFileName函数 （libloaderapi.h） 检索包含指定模块的文件的完全限定路径。该模块必须已由当前进程加载。  要查找由另一个进程加载的模块的文件，请使用 [GetModuleFileNameEx](https://learn.microsoft.com/en-us/windows/desktop/api/psapi/nf-psapi-getmodulefilenameexa) 函数。 语法 C++复制  DWORD GetModuleFileNameA(  [in, optional] HMODULE hModule,  [out] LPSTR lpFilename,  [in] DWORD nSize  ); 参数 [in, optional] hModule  正在请求其 path 的已加载模块的句柄。如果此参数为 **NULL，则 GetModuleFileName** 将检索当前进程的可执行文件的路径。  **GetModuleFileName** 函数不会检索使用 **LOAD\_LIBRARY\_AS\_DATAFILE** 标志加载的模块的路径。有关更多信息，请参阅 [LoadLibraryEx](https://learn.microsoft.com/en-us/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibraryexa)。  [out] lpFilename  指向接收模块的完全限定路径的缓冲区的指针。如果路径的长度小于 *nSize* 参数指定的大小，则函数成功，路径将作为以 null 结尾的字符串返回。  如果路径的长度超过 *nSize* 参数指定的大小，则函数成功，字符串将被截断为 *nSize* 字符，包括终止 null 字符。  **Windows XP：**字符串被截断为 *nSize* 字符，并且不以 null 结尾。  返回的字符串将使用加载模块时指定的相同格式。因此，路径可以是长文件名或短文件名，并且可以使用 prefix 。有关更多信息，请参阅[命名文件](https://learn.microsoft.com/en-us/windows/desktop/FileIO/naming-a-file)。\\?\  [in] nSize  *lpFilename* 缓冲区的大小，以 **TCHAR** 为单位。 返回值 如果函数成功，则返回值是复制到缓冲区的字符串的长度（以字符为单位），不包括终止 null 字符。如果缓冲区太小而无法保存模块名称，则字符串将被截断为 *nSize* 字符，包括终止 null 字符，函数返回 *nSize*，并且该函数将最后一个错误设置为 **ERROR\_INSUFFICIENT\_BUFFER**。  **Windows XP：**如果缓冲区太小而无法容纳模块名称，则函数将返回 *nSize*。最后一个错误代码仍为 **ERROR\_SUCCESS**。如果 *nSize* 为零，则返回值为零，最后一个错误代码为 **ERROR\_SUCCESS**。  如果函数失败，则返回值为 0（零）。要获取扩展错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。 言论 如果 DLL 在两个进程中加载，则其在一个进程中的文件名可能与另一个进程中的文件名不同。  全局变量会自动初始化为可执行文件的完整路径，并可用于检索可执行文件的完整路径名。\_pgmptr 例子 有关示例，请参阅 [安装服务](https://learn.microsoft.com/en-us/windows/desktop/Services/installing-a-service)。  **注意**  libloaderapi.h 标头将 GetModuleFileName 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 encoding-neutral 别名与非编码中性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。有关更多信息，请参见[**函数原型的约定**](https://learn.microsoft.com/en-us/windows/win32/intl/conventions-for-function-prototypes)。 |

## GetModuleFileNameEX函数的语法如下，这个模块由另外一个进程加载，

### 注意这个函数效率没有GetModuleFileName高。

|  |
| --- |
| GetModuleFileNameExA 函数 （psapi.h） 检索包含指定模块的文件的完全限定路径。 语法 C++复制  DWORD GetModuleFileNameExA(  [in] HANDLE hProcess,  [in, optional] HMODULE hModule,  [out] LPSTR lpFilename,  [in] DWORD nSize  ); 参数 [in] hProcess  包含模块的进程的句柄。  该 handle 必须具有 **PROCESS\_QUERY\_INFORMATION** 和 **PROCESS\_VM\_READ** 访问权限。有关详细信息，请参阅[进程安全性和访问权限](https://learn.microsoft.com/en-us/windows/desktop/ProcThread/process-security-and-access-rights)。  **Windows 10 及更高版本、Windows Server 2016 及更高版本**：如果 *hModule* 参数为 NULL，则句柄只需要 **PROCESS\_QUERY\_LIMITED\_INFORMATION** 访问权限。  **GetModuleFileNameEx** 函数不会检索使用 **LOAD\_LIBRARY\_AS\_DATAFILE** 标志加载的模块的路径。有关更多信息，请参阅 [LoadLibraryEx](https://learn.microsoft.com/en-us/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibraryexa)。  [in, optional] hModule  模块的句柄。如果此参数为 NULL，**则 GetModuleFileNameEx** 返回 *hProcess* 中指定的进程的可执行文件的路径。  [out] lpFilename  指向缓冲区的指针，该缓冲区接收模块的完全限定路径。如果文件名的大小大于 *nSize* 参数的值，则函数成功，但文件名被截断并以 null 结尾。  [in] nSize  *lpFilename* 缓冲区的大小（以字符为单位）。 返回值 如果函数成功，则返回值指定复制到缓冲区的字符串的长度。  如果函数失败，则返回值为零。要获取扩展错误信息，请调用 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。 言论 **GetModuleFileNameEx** 函数主要供必须从另一个进程提取模块信息的调试器和类似应用程序使用。如果目标进程中的模块列表已损坏或尚未初始化，或者如果模块列表在函数调用期间由于加载或卸载 DLL 而发生更改，**则 GetModuleFileNameEx** 可能会失败或返回不正确的信息。  若要检索当前进程中模块的名称，请使用 [GetModuleFileName](https://learn.microsoft.com/en-us/windows/desktop/api/libloaderapi/nf-libloaderapi-getmodulefilenamea) 函数。这比使用当前进程的句柄调用 **GetModuleFileNameEx** 更高效、更可靠。  若要检索远程进程的主可执行模块的名称，请使用 [GetProcessImageFileName](https://learn.microsoft.com/en-us/windows/desktop/api/psapi/nf-psapi-getprocessimagefilenamea) 或 [QueryFullProcessImageName](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-queryfullprocessimagenamea) 函数。这比使用 NULL 模块句柄调用 **GetModuleFileNameEx** 函数更高效、更可靠。  从 Windows 7 和 Windows Server 2008 R2 开始，Psapi.h 建立 PSAPI 函数的版本号。PSAPI 版本号会影响用于调用函数的名称，并且 程序必须加载的库。  如果 **PSAPI\_VERSION** 为 2 或更大，则此函数在 Psapi.h 中定义为 **K32GetModuleFileNameEx** 并在 kernel32.lib 和 Kernel32.dll。如果 **PSAPI\_VERSION** 为 1，则 this 函数**在** Psapi.h 并导出为 Psapi.lib 并Psapi.dll为调用 **K32GetModuleFileNameEx** 的包装器。  必须在 Windows 早期版本上运行的程序 以及 Windows 7 及更高版本应始终将此函数称为 **GetModuleFileNameEx**。要确保符号的正确解析， 将 Psapi.lib 添加到 **TARGETLIBS** 宏并使用 **-DPSAPI\_VERSION=1** 编译程序。要使用运行时动态链接，请加载 Psapi.dll。 例子 有关示例，请参阅 [枚举进程的所有模块](https://learn.microsoft.com/en-us/windows/desktop/psapi/enumerating-all-modules-for-a-process)。  **注意**  psapi.h 标头将 GetModuleFileNameEx 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 encoding-neutral 别名与非编码中性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。有关更多信息，请参见[**函数原型的约定**](https://learn.microsoft.com/en-us/windows/win32/intl/conventions-for-function-prototypes)。 |

# 演练，注意vs2010粘贴内容到word文档

## 老师实例

|  |
| --- |
|  |

## 1.新建一个MFC项目，取名Lesson18-dir-op，选择基于对话框的应用程序，点击完成，然后我们给对话框添加一个组框，取名新建文件夹，在组框里面托放一个静态并且，文本为需要创建的文件，然后在他的旁边拖放一个编辑框，他的ID用默认的就行了，然后在右边拖放一个按钮，ID改为IDC\_BTN\_CREATE.然后在这三个控件的下面拖放一个静态文本，ID改为IDC\_RESULT，文本为空

|  |
| --- |
|  |

## 2.给这个按钮添加点击事件处理代码，如下

|  |
| --- |
| void CLesson18diropDlg::OnBnClickedBtnCreate()  {  // TODO: 在此添加控件通知处理程序代码  //先判断用户有没有输入内容，如果没有，提示用户输入  TCHAR curDir[MAX\_PATH];  CString s;  GetDlgItemText(IDC\_EDIT1,s);  if(s == \_T(""))  {  MessageBox(\_T("请输入需要创建的目录"));  SetDlgItemText(IDC\_RESULT,\_T(""));  m\_editNewDir.SetFocus();  }  else if(s.GetAt(1)==':') //如果有盘符的  {  if(!CreateDirectory(s,NULL))  {  MessageBox(\_T("创建新文件夹失败！！"));  }  SetDlgItemText(IDC\_RESULT,s);  SetDlgItemText(IDC\_EDIT1,\_T(""));  MessageBox(\_T("创建新文件夹成功"));  }  else  {  //在当前目录里面创建目录，输入的目录名称没有盘符的情况  if(!CreateDirectory(s,NULL))  {  MessageBox(\_T("创建新文件夹失败！！"));  }  GetCurrentDirectory(MAX\_PATH,curDir);  s = curDir + CString("\\") + s;  SetDlgItemText(IDC\_RESULT,s);    SetDlgItemText(IDC\_EDIT1,\_T(""));  MessageBox(\_T("创建新文件夹成功"));  }    } |

### 运行程序，效果如下

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

### 上面的按钮的功能是，点击按钮，如果文本框里面的内容有:,说明它有盘符，是一个绝对路径，我们就在指定的盘符下面创建目录，如果没有:,说明是相对目录，我们在当前默认当前目录，这里是cpp源码所在的目录新建一个目录，如果用户没有输入，会提示用户输入

## 3.我们再添加一个组框，取名当前目录，然后拖放一个按钮，ID为IDC\_BTN\_GET，按钮文本：获取，然后拖放一个编辑框，把它禁用，这是用来显示ID用默认的

|  |
| --- |
|  |

## 4.在这个按钮的点击事件中获取当前目录并且显示在文本框里面，代码如下

|  |
| --- |
| void CLesson18diropDlg::OnBnClickedBtnGet()  {  // TODO: 在此添加控件通知处理程序代码  TCHAR curDir[MAX\_PATH];  DWORD ret;  ret = GetCurrentDirectory(MAX\_PATH,curDir);  if(ret == 0)  {  MessageBox(\_T("获取当前文件夹失败"));  }  CString s(curDir);  SetDlgItemText(IDC\_EDIT2,s);  } |

### 效果如下：

|  |
| --- |
|  |

## 5.然后我们再添加一个组框，取名：设置当前目录，然后在里面一个静态文本，内容为：输入新当前目录，然后添加一个编辑框，ID是IDC\_EDIT3，添加一个按钮，文本是：设置，ID是IDC\_BTN\_SET，在添加一个静态文本，内容是：新的当前目录和一个编辑框，编辑框的ID是IDC\_EDIT4,把这个编辑框设置为禁用状态。

|  |
| --- |
|  |

## 6.给设置按钮添加事件响应代码，注意编辑框里面的目录必须存在否则设置失败

|  |
| --- |
| void CLesson18diropDlg::OnBnClickedBtnSet()  {  // TODO: 在此添加控件通知处理程序代码  CString s;  GetDlgItemText(IDC\_EDIT3,s);  if(s == \_T(""))  {  MessageBox(\_T("请输入需要设置的目录"));  SetDlgItemText(IDC\_RESULT,\_T(""));  m\_editNewDir.SetFocus();  }else{  if(!SetCurrentDirectory(s))  {  MessageBox(\_T("输入的目录不存在，请先创建目录再设置当前目录！"));  return;  }  else  {  SetDlgItemText(IDC\_EDIT4,s);  }  }  } |

### 效果

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

### 改进：我们把取消按钮删除，把确定按钮的文本改为退出

|  |
| --- |
|  |

### 修改当前目录只在程序运行的时候才有效，程序退出重新运行，以前的设置就丢失了，

### 这个问题在实际开发中需要解决

### 注意：新建好一个对话框项目，切记不要把ok按钮删除，需要保留，并且把他的代码改动一下，如图

|  |
| --- |
|  |

### 注意：这里需要使用平台sdk的MessageBox才有用

### 如果你把OK按钮直接删除了，当你的程序运行时，你会发现，只要你一按下回车，程序就会退出，非常不好

### 有时候还真不能盲目跟随老师，老师就是一上来就把按钮删除，这是不对的

## 模块信息

模块是可执行文件或 DLL。 每个进程都由一个或多个模块组成。 可以通过调用 [**EnumProcessModules**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/nf-psapi-enumprocessmodules) 函数来检索进程的模块句柄列表。 此函数使用指定进程的模块句柄填充 **HMODULE** 值的数组。 第一个模块是可执行文件。 请记住，这些模块句柄很可能来自其他某个进程，因此不能将它们与 [**GetModuleFileName**](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-getmodulefilenamea) 等函数一起使用。 但是，可以使用 [PSAPI 函数](https://learn.microsoft.com/zh-cn/windows/win32/psapi/psapi-functions) 从另一个进程获取有关模块的信息。

以下过程介绍如何从另一个进程获取模块信息。

**从另一个进程获取模块信息**

1. 调用 [**GetModuleBaseName**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/nf-psapi-getmodulebasenamea) 函数。 此函数采用进程句柄和模块句柄作为输入，并使用模块 (的基名称填充缓冲区，例如，Kernel32.dll) 。 相关函数 [**GetModuleFileNameEx**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/nf-psapi-getmodulefilenameexa) 采用与输入相同的参数，但返回模块 (的完整路径，例如，C:\Windows\System32\Kernel32.dll) 。
2. 调用 [**GetModuleInformation**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/nf-psapi-getmoduleinformation) 函数。 此函数采用进程句柄和模块句柄，并使用模块的加载地址、它占用的线性地址空间的大小以及指向其入口点的指针填充 [**MODULEINFO**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/ns-psapi-moduleinfo) 结构。

如果应用程序需要当前进程的模块信息，则应使用 [**GetModuleFileName**](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-getmodulefilenamea) 函数而不是 PSAPI 模块函数。 这可通过两种方式帮助应用程序性能： **GetModuleFileName** 函数比 PSAPI 模块函数更高效;如果应用程序不使用任何 PSAPI 函数，应用程序可以避免加载psapi.dll。

[**GetModuleBaseName**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/nf-psapi-getmodulebasenamea) 和 [**GetModuleFileNameEx**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Psapi/nf-psapi-getmodulefilenameexa) 函数主要用于调试器和必须从另一个进程提取模块信息的类似应用程序。 如果目标进程中的模块列表已损坏或尚未初始化，或者模块列表在函数调用期间因加载或卸载 DLL 而更改，则这些函数可能会失败或返回不正确的信息。

## 7.下面我们来演练一下获取模块文件名称，在windows编程或者MFC中，一个模块是指一个可执行文件或者是一个dll文件，我们给对话框添加一个按钮，文本为获取模块信息，ID为IDC\_MODULE,然后在他旁边添加一个编辑框，使用默认ID：IDC\_EDIT5，然后把它禁用掉，然后我们给这个按钮添加点击事件代码

|  |
| --- |
| void CLesson18diropDlg::OnBnClickedModule()  {  // TODO: 在此添加控件通知处理程序代码  //获取主模块需要把第一个参数改为NULL  TCHAR szModule[MAX\_PATH];  if(!GetModuleFileName(NULL,szModule,MAX\_PATH))  {  MessageBox(\_T("获取主模块信息失败！"));  return;  }  else  {  SetDlgItemText(IDC\_EDIT5,szModule);  }  } |

### 效果

|  |
| --- |
|  |

## 8.上面是主模块信息，如果我们的程序调用了一个dll文件，这个dll文件也叫做我们的模块，只不过它不是住模块，我们把上面的按钮的文本改一下，然后我们再添加一个按钮和一个文本框，如图

|  |
| --- |
|  |

## 9.然后，我们给按钮的点击事件添加处理代码如下，注意如果不是当作主模块第一个参数不能是NULL必须先调用一个dll，然后把它返回的句柄作为参数传递给GetModleFileName函数

|  |
| --- |
| void CLesson18diropDlg::OnBnClickedModule2()  {  // TODO: 在此添加控件通知处理程序代码  //获取调用模块不能把第一个参数改为NULL，需要指定模块句柄  //也就是需要先去调用一个模块，然后接受返回的模块句柄再把它作为参数传递给GetModuleFileName的第一个参数  HMODULE hKernel32;  TCHAR szModule[MAX\_PATH];  //调用kernel32.dll  hKernel32 = LoadLibrary(\_T("kernel32.dll"));  //返回的模块句柄再把它作为参数传递给GetModuleFileName的第一个参数  if(!GetModuleFileName(hKernel32,szModule,MAX\_PATH))  {  MessageBox(\_T("获取模块信息失败！"));  return;  }  else  {  SetDlgItemText(IDC\_EDIT6,szModule);  }  } |

### 效果：

|  |
| --- |
|  |

## 10.还有一个小问题，就是我们的程序点击x按钮不会关闭，我们可以给他一个WM\_CLOSE信息处理函数，内容很简单就是调用CDialogEx的OnOK函数

|  |
| --- |
|  |

## Lesson18-dir-opDlg.cpp的完整代码如下

### Lesson18-dir-opDlg.cpp

|  |
| --- |
| // Lesson18-dir-opDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "Lesson18-dir-op.h"  #include "Lesson18-dir-opDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CLesson18diropDlg 对话框  CLesson18diropDlg::CLesson18diropDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CLesson18diropDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CLesson18diropDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_EDIT1, m\_editNewDir);  }  BEGIN\_MESSAGE\_MAP(CLesson18diropDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDOK, &CLesson18diropDlg::OnBnClickedOk)  ON\_BN\_CLICKED(IDC\_BTN\_CREATE, &CLesson18diropDlg::OnBnClickedBtnCreate)  ON\_BN\_CLICKED(IDC\_BTN\_GET, &CLesson18diropDlg::OnBnClickedBtnGet)  ON\_BN\_CLICKED(IDC\_BTN\_SET, &CLesson18diropDlg::OnBnClickedBtnSet)  ON\_BN\_CLICKED(IDCANCEL, &CLesson18diropDlg::OnBnClickedCancel)  ON\_BN\_CLICKED(IDC\_MODULE, &CLesson18diropDlg::OnBnClickedModule)  ON\_BN\_CLICKED(IDC\_MODULE2, &CLesson18diropDlg::OnBnClickedModule2)  ON\_WM\_CLOSE()  END\_MESSAGE\_MAP()  // CLesson18diropDlg 消息处理程序  BOOL CLesson18diropDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CLesson18diropDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CLesson18diropDlg::OnPaint()  {  if (IsIconic())  {  CPaintDC dc(this); // 用于绘制的设备上下文  SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);  // 使图标在工作区矩形中居中  int cxIcon = GetSystemMetrics(SM\_CXICON);  int cyIcon = GetSystemMetrics(SM\_CYICON);  CRect rect;  GetClientRect(&rect);  int x = (rect.Width() - cxIcon + 1) / 2;  int y = (rect.Height() - cyIcon + 1) / 2;  // 绘制图标  dc.DrawIcon(x, y, m\_hIcon);  }  else  {  CDialogEx::OnPaint();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CLesson18diropDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  void CLesson18diropDlg::OnBnClickedOk()  {  // TODO: 在此添加控件通知处理程序代码  if(IDOK == ::MessageBox(this->m\_hWnd,\_T("退出程序？"),\_T("退出确认"),MB\_OKCANCEL))  {  CDialogEx::OnOK();  }  }  void CLesson18diropDlg::OnBnClickedBtnCreate()  {  // TODO: 在此添加控件通知处理程序代码  //先判断用户有没有输入内容，如果没有，提示用户输入  TCHAR curDir[MAX\_PATH];  CString s;  GetDlgItemText(IDC\_EDIT1,s);  if(s == \_T(""))  {  MessageBox(\_T("请输入需要创建的目录"));  SetDlgItemText(IDC\_RESULT,\_T(""));  m\_editNewDir.SetFocus();  }  else if(s.GetAt(1)==':') //如果有盘符的  {  if(!CreateDirectory(s,NULL))  {  MessageBox(\_T("创建新文件夹失败！！"));  }  SetDlgItemText(IDC\_RESULT,s);  SetDlgItemText(IDC\_EDIT1,\_T(""));  MessageBox(\_T("创建新文件夹成功"));  }  else  {  //在当前目录里面创建目录，输入的目录名称没有盘符的情况  if(!CreateDirectory(s,NULL))  {  MessageBox(\_T("创建新文件夹失败！！"));  }  GetCurrentDirectory(MAX\_PATH,curDir);  s = curDir + CString("\\") + s;  SetDlgItemText(IDC\_RESULT,s);    SetDlgItemText(IDC\_EDIT1,\_T(""));  MessageBox(\_T("创建新文件夹成功"));  }    }  void CLesson18diropDlg::OnBnClickedBtnGet()  {  // TODO: 在此添加控件通知处理程序代码  TCHAR curDir[MAX\_PATH];  DWORD ret;  ret = GetCurrentDirectory(MAX\_PATH,curDir);  if(ret == 0)  {  MessageBox(\_T("获取当前文件夹失败"));  }  CString s(curDir);  SetDlgItemText(IDC\_EDIT2,s);  }  void CLesson18diropDlg::OnBnClickedBtnSet()  {  // TODO: 在此添加控件通知处理程序代码  CString s;  GetDlgItemText(IDC\_EDIT3,s);  if(s == \_T(""))  {  MessageBox(\_T("请输入需要设置的目录"));  SetDlgItemText(IDC\_RESULT,\_T(""));  m\_editNewDir.SetFocus();  }else{  if(!SetCurrentDirectory(s))  {  MessageBox(\_T("输入的目录不存在，请先创建目录再设置当前目录！"));  return;  }  else  {  SetDlgItemText(IDC\_EDIT4,s);  }  }  }  void CLesson18diropDlg::OnBnClickedCancel()  {  // TODO: 在此添加控件通知处理程序代码  //CDialogEx::OnCancel();  }  void CLesson18diropDlg::OnBnClickedModule()  {  // TODO: 在此添加控件通知处理程序代码  //获取主模块需要把第一个参数改为NULL  TCHAR szModule[MAX\_PATH];  if(!GetModuleFileName(NULL,szModule,MAX\_PATH))  {  MessageBox(\_T("获取主模块信息失败！"));  return;  }  else  {  SetDlgItemText(IDC\_EDIT5,szModule);  }  }  void CLesson18diropDlg::OnBnClickedModule2()  {  // TODO: 在此添加控件通知处理程序代码  //获取调用模块不能把第一个参数改为NULL，需要指定模块句柄  //也就是需要先去调用一个模块，然后接受返回的模块句柄再把它作为参数传递给GetModuleFileName的第一个参数  HMODULE hKernel32;  TCHAR szModule[MAX\_PATH];  //调用kernel32.dll  hKernel32 = LoadLibrary(\_T("kernel32.dll"));  //返回的模块句柄再把它作为参数传递给GetModuleFileName的第一个参数  if(!GetModuleFileName(hKernel32,szModule,MAX\_PATH))  {  MessageBox(\_T("获取模块信息失败！"));  return;  }  else  {  SetDlgItemText(IDC\_EDIT6,szModule);  }  }  void CLesson18diropDlg::OnClose()  {  // TODO: 在此添加消息处理程序代码和/或调用默认值  CDialogEx::OnOK();  CDialogEx::OnClose();  } |

## 扩展：其实这个按钮的处理代码也可以弹出选择文件对话框，然后我们加载用文件对话框选择是文件然后删除他的信息

# 这一节的学习到此为止