# 学习大纲

|  |
| --- |
|  |

# API用法说明

## GetFileAttributesExA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索指定文件或目录的属性。  若要以事务处理操作的形式执行此操作，请使用 [GetFileAttributesTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getfileattributestransacteda) 函数。 语法 C++复制  BOOL GetFileAttributesExA(  [in] LPCSTR lpFileName,  [in] GET\_FILEEX\_INFO\_LEVELS fInfoLevelId,  [out] LPVOID lpFileInformation  ); 参数 [in] lpFileName  文件或目录的名称。  默认情况下，名称限制为MAX\_PATH个字符。 若要将此限制扩展到 32，767 个宽字符，请在路径前面添加“\？\”。 有关详细信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，可以选择删除MAX\_PATH限制，而无需在前面添加“\\？\”。 有关详细信息，请参阅 [**命名文件、路径和命名空间**](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file) 的“最大路径长度限制”部分。  [in] fInfoLevelId  要检索的属性信息的类。  此参数可以是 [GET\_FILEEX\_INFO\_LEVELS](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ne-minwinbase-get_fileex_info_levels) 枚举中的以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GetFileExInfoStandard** | *lpFileInformation* 参数是[WIN32\_FILE\_ATTRIBUTE\_DATA](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/ns-fileapi-win32_file_attribute_data)结构。 |   [out] lpFileInformation  指向接收属性信息的缓冲区的指针。  存储在此缓冲区中的属性信息类型由 fInfoLevelId 的值决定。 返回值 如果函数成功，则返回值为非零值。  如果函数失败，则返回值为零 (0) 。 要获得更多的错误信息，请调用 GetLastError。 注解 [GetFileAttributes](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfileattributesa) 函数检索文件系统属性信息。 **GetFileAttributesEx** 可以获取其他一组文件或目录属性信息。 目前， **GetFileAttributesEx** 检索一组标准属性，该属性是文件系统属性信息的超集。  在装载文件夹的目录上调用 **GetFileAttributesEx** 函数时，它将返回目录的属性，而不是装载的文件夹与目录关联的卷中根目录的属性。 若要获取关联卷的属性，请调用 [GetVolumeNameForVolumeMountPoint](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getvolumenameforvolumemountpointw) 以获取关联卷的名称。 然后在调用 **GetFileAttributesEx** 时使用生成的名称。 结果是关联卷上根目录的属性。  在Windows 8和Windows Server 2012中，以下技术支持此功能。  展开表   |  |  | | --- | --- | | **技术** | **支持** | | 服务器消息块 (SMB) 3.0 协议 | 是 | | SMB 3.0 透明故障转移 (TFO) | 是 | | 具有横向扩展文件共享的 SMB 3.0 (SO) | 是 | | 群集共享卷文件系统 (CsvFS) | 是 | | 弹性文件系统 (ReFS) | 是 |     符号链接行为 - 如果路径指向符号链接，函数将返回符号链接的属性。 事务处理操作 如果文件在事务中打开进行修改，则在提交事务之前，任何其他线程都不能打开该文件进行修改。 因此，如果事务处理线程首先打开文件，则尝试在提交事务之前修改文件的任何后续线程都会收到共享冲突。 如果非事务处理线程在事务处理线程之前修改了文件，并且当事务尝试打开该文件时文件仍然打开，则事务会收到 **错误ERROR\_TRANSACTIONAL\_CONFLICT**。  **备注**  fileapi.h 标头将 GetFileAttributesEx 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** | fileapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [文件特性常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-attribute-constants)  [文件管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [GetFileAttributes](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfileattributesa)  [GetFileAttributesTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getfileattributestransacteda)  [SetFileAttributes](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfileattributesa)  [符号链接](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/symbolic-links) |

## WIN32\_FILE\_ATTRIBUTE\_DATA 结构

|  |  |  |
| --- | --- | --- |
| 包含文件或目录的属性信息。 [GetFileAttributesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfileattributesexa) 函数使用此结构。 语法 C++  typedef struct \_WIN32\_FILE\_ATTRIBUTE\_DATA {  DWORD dwFileAttributes;  FILETIME ftCreationTime;  FILETIME ftLastAccessTime;  FILETIME ftLastWriteTime;  DWORD nFileSizeHigh;  DWORD nFileSizeLow;  } WIN32\_FILE\_ATTRIBUTE\_DATA, \*LPWIN32\_FILE\_ATTRIBUTE\_DATA; 成员 dwFileAttributes  文件或目录的文件系统属性信息。  有关可能的值及其说明，请参阅 [文件属性常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-attribute-constants)。  ftCreationTime  [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构，指定何时创建文件或目录。  如果基础文件系统不支持创建时间，则此成员为零。  ftLastAccessTime  [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。  对于文件， 结构指定上次读取或写入文件时。  对于目录，结构指定何时创建目录。  对于文件和目录，指定的日期是正确的，但一天中的时间始终设置为午夜。 如果基础文件系统不支持上次访问时间，则此成员为零。  ftLastWriteTime  [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。  对于文件， 结构指定上次写入该文件的日期。  对于目录，结构指定何时创建目录。  如果基础文件系统不支持上次写入时间，则此成员为零。  nFileSizeHigh  文件大小的高阶 **DWORD** 。  此成员对目录没有意义。  nFileSizeLow  文件大小的低序 **DWORD** 。  此成员对目录没有意义。 注解 并非所有文件系统都可以记录创建和上次访问时间，并非所有文件系统都以相同的方式记录它们。 例如，在 FAT 文件系统上，创建时间的分辨率为 10 毫秒，写入时间的分辨率为 2 秒，访问时间的分辨率为 1 天。 在 NTFS 文件系统上，访问时间的分辨率为 1 小时。 有关详细信息，请参阅[文件时间](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/file-times)。 要求  |  |  | | --- | --- | | **标头** | fileapi.h (包括 Windows.h、WinBase.h) |  另请参阅 [文件特性常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-attribute-constants)  [GetFileAttributesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfileattributesexa) |

# 这一节我们来学习用MFC来创建对话框窗口

## 1.打开vs2010，新建-》项目-》c++-》MFC-MFC应用程序，项目名称：Lesson8-MFC-Dialog，点击完成

|  |
| --- |
|  |

## 2.然后会弹出创建项目向导，next-》选择基于对话框的应用程序

|  |
| --- |
|  |

## 3.点击完成，就会生成一个对话框应用程序

|  |
| --- |
|  |

## 4.我们把对话框默认的2个按钮输出，然后给对话框添加一个图片框控件，然后再添加2个按钮，点击每一个按钮在图片框中显示不一样的位图，注意，有一个问题非常坑爹，就是使用图片控件的SetBitmap设置图片程序老是崩溃，使用我们使用曲线救国，定义一个位图画刷，然后用整个画刷来填充图片框，完整代码如下

|  |
| --- |
| // MfcDlgDlg.cpp: 实现文件  //  #include "pch.h"  #include "framework.h"  #include "MfcDlg.h"  #include "MfcDlgDlg.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  #ifdef AFX\_DESIGN\_TIME  enum { IDD = IDD\_ABOUTBOX };  #endif  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(IDD\_ABOUTBOX)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CMfcDlgDlg 对话框  CMfcDlgDlg::CMfcDlgDlg(CWnd\* pParent /\*=nullptr\*/)  : CDialogEx(IDD\_MFCDLG\_DIALOG, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CMfcDlgDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_PIC\_STATIC, m\_picCtrl);  }  BEGIN\_MESSAGE\_MAP(CMfcDlgDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BUTTON2, &CMfcDlgDlg::OnBnClickedButton2)  ON\_BN\_CLICKED(IDC\_BUTTON1, &CMfcDlgDlg::OnBnClickedButton1)  ON\_WM\_CREATE()  END\_MESSAGE\_MAP()  // CMfcDlgDlg 消息处理程序  BOOL CMfcDlgDlg::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 != nullptr)  {  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 CMfcDlgDlg::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 CMfcDlgDlg::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  {  CClientDC dc(&m\_picCtrl);  CRect rc;  m\_picCtrl.GetClientRect(&rc);  CBrush\* brush = CBrush::FromHandle(CreateSolidBrush(RGB(255, 255, 255)));  dc.FillRect(&rc, brush);  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 CMfcDlgDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  void CMfcDlgDlg::OnBnClickedButton2()  {  // TODO: 在此添加控件通知处理程序代码  CClientDC dc(&m\_picCtrl);  CRect rc;  m\_picCtrl.GetClientRect(&rc);  CBrush\* brush = nullptr;  if (!brush)  brush = CBrush::FromHandle(CreateSolidBrush(RGB(255, 255, 255)));  dc.FillRect(&rc, brush);  CBitmap bitmap;  bitmap.LoadBitmapW(IDB\_BITMAP2);  CBrush bitBrush(&bitmap);  dc.FillRect(&rc, &bitBrush);  }  void CMfcDlgDlg::OnBnClickedButton1()  {  // TODO: 在此添加控件通知处理程序代码  CClientDC dc(&m\_picCtrl);  CRect rc;  m\_picCtrl.GetClientRect(&rc);  CBrush\* brush = nullptr;  if(!brush)  brush = CBrush::FromHandle(CreateSolidBrush(RGB(255, 255, 255)));  dc.FillRect(&rc, brush);  CBitmap bitmap;  bitmap.LoadBitmapW(IDB\_BITMAP1);  CBrush bitBrush(&bitmap);  dc.FillRect(&rc, &bitBrush);    } |

# 下面是vs2010的版本

|  |
| --- |
|  |

## Lesson8-MFCdlgDlg.cpp的完整代码如下

|  |
| --- |
| // Lesson8-MFCdlgDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "Lesson8-MFCdlg.h"  #include "Lesson8-MFCdlgDlg.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 CDialog  {  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() : CDialog(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialog::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialog)  END\_MESSAGE\_MAP()  // CLesson8MFCdlgDlg 对话框  CLesson8MFCdlgDlg::CLesson8MFCdlgDlg(CWnd\* pParent /\*=NULL\*/)  : CDialog(CLesson8MFCdlgDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CLesson8MFCdlgDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialog::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_PIC\_STATIC, m\_picCtrl);  }  BEGIN\_MESSAGE\_MAP(CLesson8MFCdlgDlg, CDialog)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  //}}AFX\_MSG\_MAP  ON\_BN\_CLICKED(IDC\_BUTTON1, &CLesson8MFCdlgDlg::OnBnClickedButton1)  ON\_BN\_CLICKED(IDC\_BUTTON2, &CLesson8MFCdlgDlg::OnBnClickedButton2)  END\_MESSAGE\_MAP()  // CLesson8MFCdlgDlg 消息处理程序  BOOL CLesson8MFCdlgDlg::OnInitDialog()  {  CDialog::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)  {  CString strAboutMenu;  strAboutMenu.LoadString(IDS\_ABOUTBOX);  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 CLesson8MFCdlgDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialog::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 CLesson8MFCdlgDlg::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  {  CClientDC dc(&m\_picCtrl);  CRect rc;  m\_picCtrl.GetClientRect(&rc);  CBrush\* brush = CBrush::FromHandle(CreateSolidBrush(RGB(255, 255, 255)));  dc.FillRect(&rc, brush);  CDialog::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 CLesson8MFCdlgDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  void CLesson8MFCdlgDlg::OnBnClickedButton1()  {  // TODO: 在此添加控件通知处理程序代码  CClientDC dc(&m\_picCtrl);  CRect rc;  m\_picCtrl.GetClientRect(&rc);  CBrush\* brush = NULL;  if(!brush)  brush = CBrush::FromHandle(CreateSolidBrush(RGB(255, 255, 255)));  dc.FillRect(&rc, brush);  CBitmap bitmap;  bitmap.LoadBitmapW(MAKEINTRESOURCE(IDB\_BITMAP1));  CBrush bitBrush(&bitmap);  dc.FillRect(&rc, &bitBrush);  }  void CLesson8MFCdlgDlg::OnBnClickedButton2()  {  // TODO: 在此添加控件通知处理程序代码  CClientDC dc(&m\_picCtrl);  CRect rc;  m\_picCtrl.GetClientRect(&rc);  CBrush\* brush =NULL;  if(!brush)  brush = CBrush::FromHandle(CreateSolidBrush(RGB(255, 255, 255)));  dc.FillRect(&rc, brush);  CBitmap bitmap;  bitmap.LoadBitmapW(MAKEINTRESOURCE(IDB\_BITMAP2));  CBrush bitBrush(&bitmap);  dc.FillRect(&rc, &bitBrush);  } |

### 写法和vs2019有点不一样

### 效果

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

# 小结：

## 1.如果需要在对话框初始化后就在一些绘图之类的操作，如填充背景，需要在对话框的OnPaint函数里面添加代码，如，我们想给图片框填充白色背景

|  |
| --- |
|  |

# 扩展:我们新建一个MFC对话框程序,如何在上面练习第4课的获取文件属性功能,当用户点击一个按钮,可以选择有文件,然后文件就获取这个文件的属性然后显示在对话框控件上面

## 1.新建一个MFC对话框项目取名:Lesson8-mfc-dlg-file-attribs

|  |
| --- |
|  |

## 2.把对话框上面一些不需要的控件删除,保留确定按钮,然后需要重写PreTranslateMessage函数来防止用户按下回车键程序就退出

|  |
| --- |
|  |
| BOOL Clesson8mfcdlgfileattribDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if (pMsg->message == WM\_KEYDOWN)  {  switch (pMsg->wParam)  {  case VK\_RETURN:  return TRUE;  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  } |

## 3.然后我们给打开按钮添加消息处理函数,代码如下

|  |
| --- |
| void Clesson8mfcdlgfileattribDlg::OnBnClickedBtnOpen()  {  // TODO: 在此添加控件通知处理程序代码  WIN32\_FILE\_ATTRIBUTE\_DATA wfad;//保存文件属性的结构体  CString filePath;  CFileDialog dlg(TRUE);  if (IDOK == dlg.DoModal())  {  filePath = dlg.GetPathName();  if (GetFileAttributesEx(filePath, GetFileExInfoStandard, &wfad))  {  SetDlgItemInt(IDC\_SIZE, wfad.nFileSizeLow);  SetDlgItemText(IDC\_CTIME, GetFileTimeString(&wfad.ftCreationTime));  SetDlgItemText(IDC\_ATIME, GetFileTimeString(&wfad.ftLastAccessTime));  SetDlgItemText(IDC\_WTIME, GetFileTimeString(&wfad.ftLastWriteTime));    switch (wfad.dwFileAttributes)  {  case FILE\_ATTRIBUTE\_ARCHIVE:  SetDlgItemText(IDC\_ATTRIB, \_T("Arhived"));  break;  case FILE\_ATTRIBUTE\_DIRECTORY:  SetDlgItemText(IDC\_ATTRIB, \_T("Directory"));  break;  case FILE\_ATTRIBUTE\_READONLY:  SetDlgItemText(IDC\_ATTRIB, \_T("Read Only"));  break;  case FILE\_ATTRIBUTE\_SYSTEM:  SetDlgItemText(IDC\_ATTRIB, \_T("System"));  break;  case FILE\_ATTRIBUTE\_TEMPORARY:  SetDlgItemText(IDC\_ATTRIB, \_T("Temp File"));  break;  }  }  }  } |

## 4.这里用到了一个自定义函数GetFileTimeString,代码如下

|  |
| --- |
| CString GetFileTimeString(FILETIME\* ft)  {  FILETIME ftLocal;  SYSTEMTIME st;  CString strTime("");  FileTimeToLocalFileTime(ft, &ftLocal);  FileTimeToSystemTime(&ftLocal, &st);  strTime.Format(\_T("%d-%#02d-%#02d %#02d:%#02d:%#02d"), st.wYear, st.wMonth, st.wDay, st.wHour, st.wMinute, st.wSecond);  return strTime;  } |

### 效果如下

|  |
| --- |
|  |
|  |

# 这一节的学习到此为止