# 学习大纲

|  |
| --- |
|  |

## 1.以一个文件属性数据结构体为例

|  |
| --- |
|  |

## 我们可以通过一个例子来学习这个结构体

## 2.打开vs2010，新建一个c++项目，取名：lesson4，是一个空项目，然后我们添加源码

|  |
| --- |
|  |

## 3.我们要做的就是学习使用windows的API获取文件的属性，我们先创建一个测试文件test.txt

|  |
| --- |
|  |

## 4.我们的s一个功能就是显示文件的创建时间，我们自己定义一个函数来实现

|  |
| --- |
|  |

### 执行效果如下：

|  |
| --- |
|  |

## 5.上面那个自定义函数可以多次使用，我们接着用它来显示访问时间和修改时间，代码如下

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

# 小结：

## 这个实例使用到了3个Window结构体：分别是：

### WIN32\_FILE\_ATTRIBUTE\_DATA

|  |
| --- |
| 语法 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。  有关可能的值及其描述，请参阅 [File Attribute Constants](https://learn.microsoft.com/en-us/windows/desktop/FileIO/file-attribute-constants)。  ftCreationTime  一个 [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构，用于指定文件或 目录。  如果底层文件系统不支持创建时间，则此成员为零。  ftLastAccessTime  [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。  对于文件，该结构指定上次读取或写入文件的时间。  对于目录，结构指定创建目录的时间。  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。 如果底层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/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。  对于文件，结构指定上次写入文件的时间。  对于目录，结构指定创建目录的时间。  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  文件大小的低阶 **DWORD**。  此成员对目录没有意义。 |

### FILETIME结构体

|  |
| --- |
| 语法 C++复制  typedef struct \_FILETIME {  DWORD dwLowDateTime;  DWORD dwHighDateTime;  } FILETIME, \*PFILETIME, \*LPFILETIME; 成员 dwLowDateTime  文件时间的低阶部分。  dwHighDateTime  文件时间的高阶部分。 言论 若要将 **FILETIME** 结构转换为易于向用户显示的时间，请使用 [FileTimeToSystemTime](https://learn.microsoft.com/en-us/windows/desktop/api/timezoneapi/nf-timezoneapi-filetimetosystemtime) 函数。  建议不要从 **FILETIME** 结构中加减值来获取相对时间。相反，您应该将文件时间的低阶和高阶部分复制到 [ULARGE\_INTEGER](https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-ularge_integer-r1) 结构中，对 **QuadPart** 成员执行 64 位算术，并将 **LowPart** 和 **HighPart** 成员复制到 **FILETIME** 结构中。  不要将指向 **FILETIME** 结构的指针强制转换为 **ULARGE\_INTEGER\*** 或 **\_\_int64\*** 值，因为它可能会导致 64 位 Windows 上的对齐错误。  并非所有文件系统都可以记录创建和上次访问时间，也不是所有文件系统都以相同的方式记录它们。例如，在 NT FAT 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（实际上是访问日期）。在 NTFS 上，访问时间的分辨率为 1 小时。因此， [GetFileTime](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getfiletime) 函数可能不会返回使用 [SetFileTime](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 函数设置的相同文件时间信息。此外，FAT 以本地时间记录磁盘上的时间。但是，NTFS 以 UTC 格式在磁盘上记录时间。有关更多信息，请参阅 [File Times](https://learn.microsoft.com/en-us/windows/desktop/SysInfo/file-times)。  使用 **FILETIME** 结构的函数可以允许 **dwLowDateTime** 和 **dwHighDateTime** 成员通常指定的零值或正值之外的值。例如，[SetFileTime](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 函数使用 0xFFFFFFFF 来指定应保留文件的上一个访问时间。有关更多信息，请参阅您正在调用的函数的主题。 |

### SYSTEMTIME结构体

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 语法 C++复制  typedef struct \_SYSTEMTIME {  WORD wYear;  WORD wMonth;  WORD wDayOfWeek;  WORD wDay;  WORD wHour;  WORD wMinute;  WORD wSecond;  WORD wMilliseconds;  } SYSTEMTIME, \*PSYSTEMTIME, \*LPSYSTEMTIME; 成员 wYear  这一年。此成员的有效值为 1601 到 30827。  wMonth  月份。此成员可以是以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | 1 | 一月 | | 2 | 二月 | | 3 | 三月 | | 4 | 四月 | | 5 | 五月 | | 6 | 六月 | | 7 | 七月 | | 8 | 八月 | | 9 | 九月 | | 10 | 十月 | | 11 | 十一月 | | 12 | 十二月 |   wDayOfWeek  星期几。此成员可以是以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | 0 | 星期日 | | 1 | 星期一 | | 2 | 星期二 | | 3 | 星期三 | | 4 | 星期四 | | 5 | 星期五 | | 6 | 星期六 |   wDay  该月的日期。此成员的有效值为 1 到 31。  wHour  小时。此成员的有效值为 0 到 23。  wMinute  分钟。此成员的有效值为 0 到 59。  wSecond  第二个。此成员的有效值为 0 到 59。  wMilliseconds  毫秒。此成员的有效值为 0 到 999。 |

## 可以看到，直接得到的文件时间我们很难看懂，需要使用2次时间转换，这里用到了3个API函数

### GetFileAttributesEx

|  |
| --- |
| GetFileAttributesEx The **GetFileAttributesEx** function obtains attribute information about a specified file or directory.  This function is similar to the [**GetFileAttributes**](mk:@MSITStore:C:\Program%20Files%20(x86)\Microsoft%20Visual%20Studio\MSDN98\98VS\2052\winbase.chm::/devdoc/live/pdwbase/filesio_9pgz.htm) function. **GetFileAttributes** returns a set of FAT-style attribute information. **GetFileAttributesEx** is designed to obtain other sets of file or directory attribute information. Currently, **GetFileAttributeEx** obtains a set of standard attributes that is a superset of the FAT-style attribute information.  **BOOL GetFileAttributesEx(**  **LPCTSTR** *lpFileName***,** // pointer to string that specifies a  // file or directory  **GET\_FILEEX\_INFO\_LEVELS** *fInfoLevelId***,**  // value that specifies the type of  // attribute information to obtain  **LPVOID** *lpFileInformation* // pointer to buffer to receive  // attribute information  **);**   Parameters *lpFileName*  Pointer to a null-terminated string that specifies a file or directory.  By default, this string is limited to MAX\_PATH characters. The limit is related to how the **GetFileAttributesEx** function parses paths. An application can transcend this limit and send in paths longer than MAX\_PATH characters by calling the wide (W) version of **GetFileAttributesEx** and prepending "\\?\" to the path. However, each component in the path cannot be more than MAX\_PATH characters long. The "\\?\" tells the function to turn off path parsing. This technique also works with UNC names. The "\\?\" is ignored as part of the path. For example, "\\?\C:\myworld\private" is seen as "C:\myworld\private", and "\\?\UNC\peanuts\hotstuff\coolapps" is seen as "\\peanuts\hotstuff\coolapps".  *fInfoLevelId*  Specifies a [**GET\_FILEEX\_INFO\_LEVELS**](mk:@MSITStore:C:\Program%20Files%20(x86)\Microsoft%20Visual%20Studio\MSDN98\98VS\2052\winbase.chm::/devdoc/live/pdwbase/filesio_4ws2.htm) enumeration type that gives the set of attribute information to obtain.  *lpFileInformation*  Pointer to a buffer that receives the attribute information. The type of attribute information stored into this buffer is determined by the value of *fInfoLevelId*. Return Values If the function succeeds, the return value is a nonzero value.  If the function fails, the return value is zero. To get extended error information, call [**GetLastError**](mk:@MSITStore:C:\Program%20Files%20(x86)\Microsoft%20Visual%20Studio\MSDN98\98VS\2052\winbase.chm::/devdoc/live/pdwbase/errors_3uwi.htm). |

### 文件属性常量

|  |
| --- |
| #define FILE\_ATTRIBUTE\_READONLY 0x00000001  #define FILE\_ATTRIBUTE\_HIDDEN 0x00000002  #define FILE\_ATTRIBUTE\_SYSTEM 0x00000004  #define FILE\_ATTRIBUTE\_DIRECTORY 0x00000010  #define FILE\_ATTRIBUTE\_ARCHIVE 0x00000020  #define FILE\_ATTRIBUTE\_DEVICE 0x00000040  #define FILE\_ATTRIBUTE\_NORMAL 0x00000080  #define FILE\_ATTRIBUTE\_TEMPORARY 0x00000100  #define FILE\_ATTRIBUTE\_SPARSE\_FILE 0x00000200  #define FILE\_ATTRIBUTE\_REPARSE\_POINT 0x00000400  #define FILE\_ATTRIBUTE\_COMPRESSED 0x00000800  #define FILE\_ATTRIBUTE\_OFFLINE 0x00001000  #define FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED 0x00002000  #define FILE\_ATTRIBUTE\_ENCRYPTED 0x00004000  #define FILE\_ATTRIBUTE\_INTEGRITY\_STREAM 0x00008000  #define FILE\_ATTRIBUTE\_VIRTUAL 0x00010000  #define FILE\_ATTRIBUTE\_NO\_SCRUB\_DATA 0x00020000  #define FILE\_ATTRIBUTE\_EA 0x00040000  #define FILE\_ATTRIBUTE\_PINNED 0x00080000  #define FILE\_ATTRIBUTE\_UNPINNED 0x00100000  #define FILE\_ATTRIBUTE\_RECALL\_ON\_OPEN 0x00040000  #define FILE\_ATTRIBUTE\_RECALL\_ON\_DATA\_ACCESS 0x00400000 |

### FileTimeToLocalFileTime

|  |
| --- |
| 语法 C++复制  BOOL FileTimeToLocalFileTime(  [in] const FILETIME \*lpFileTime,  [out] LPFILETIME lpLocalFileTime  ); 参数 [in] lpFileTime  指向 [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构的指针，该结构包含要转换为本地文件时间的基于 UTC 的文件时间。  [out] lpLocalFileTime  指向 [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构的指针，用于接收转换后的本地文件时间。此参数不能与 *lpFileTime* 参数相同。 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)。 言论 要在将文件时间转换为本地时间时考虑夏令时，请使用以下函数序列来代替 **FileTimeToLocalFileTime**：   1. [FileTimeToSystemTime （文件时间到系统时间）](https://learn.microsoft.com/en-us/windows/desktop/api/timezoneapi/nf-timezoneapi-filetimetosystemtime) 2. [SystemTimeToTzSpecificLocalTime](https://learn.microsoft.com/en-us/windows/desktop/api/timezoneapi/nf-timezoneapi-systemtimetotzspecificlocaltime) 3. [SystemTimeToFileTime （系统时间到文件时间）](https://learn.microsoft.com/en-us/windows/desktop/api/timezoneapi/nf-timezoneapi-systemtimetofiletime) |

### FileTimeToSystemTime

|  |
| --- |
| 语法 C++复制  BOOL FileTimeToSystemTime(  [in] const FILETIME \*lpFileTime,  [out] LPSYSTEMTIME lpSystemTime  ); 参数 [in] lpFileTime  指向包含该文件的 [FILETIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构的指针 要转换为系统 （UTC） 日期和时间格式的时间。  此值必须小于 0x8000000000000000。否则，函数将失败。  [out] lpSystemTime  指向 [SYSTEMTIME](https://learn.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-systemtime) 结构的指针，用于接收 转换文件时间。 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)。 |

### GetFileSize 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索指定文件的大小（以字节为单位）。  建议使用 [GetFileSizeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfilesizeex)。 语法 C++复制  DWORD GetFileSize(  [in] HANDLE hFile,  [out, optional] LPDWORD lpFileSizeHigh  ); 参数 [in] hFile  文件的句柄。  [out, optional] lpFileSizeHigh  指向变量的指针，其中返回了文件大小的高阶双字。 如果应用程序不需要高阶双字，此参数可以为 **NULL** 。 返回值 如果函数成功，则返回值为文件大小的低位双字，如果 *lpFileSizeHigh* 为非 **NULL**，则该函数会将文件大小的高阶双字放入该参数指向的变量中。  如果函数失败且 *lpFileSizeHigh* 为 **NULL**，则返回值 **INVALID\_FILE\_SIZE**。 要获得更多的错误信息，请调用 GetLastError。 当 *lpFileSizeHigh* 为 **NULL** 时，为大型文件返回的结果不明确，您将无法确定文件的实际大小。 建议改用 [GetFileSizeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfilesizeex) 。  如果函数失败，并且 *lpFileSizeHigh* 为非 **NULL**，则返回值 **INVALID\_FILE\_SIZE** 并且 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **除NO\_ERROR**以外的值。 备注 不能将 **GetFileSize** 函数与非视设备（如管道或通信设备）的句柄一起使用。 若要确定 *hFile* 的文件类型，请使用 [GetFileType](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfiletype) 函数。  **GetFileSize** 函数检索文件的未压缩大小。 若要获取文件的压缩大小，请使用 GetCompressedFileSize 函数。  请注意，如果返回值 **INVALID\_FILE\_SIZE** (0xffffffff) ，则应用程序必须调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 来确定函数是成功还是失败。 函数在未显示时可能失败的原因是 *lpFileSizeHigh* 可能为非 **NULL** ，或者文件大小可能0xffffffff。 在这种情况下， **GetLastError** 将在成功后返回 **NO\_ERROR** (0) 。 由于此行为，建议改用 [GetFileSizeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfilesizeex) 。  **事务处理操作：**如果有绑定到文件句柄的事务，则该函数将返回独立文件视图的信息。  在 Windows 8 和 Windows Server 2012 中，此函数由以下技术支持。  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 (CSV) | 是 | | 弹性文件系统 (ReFS) | 是 |    示例 有关示例，请参阅 [在文件中创建视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-view-within-a-file)。 要求  |  |  | | --- | --- | | **标头** | fileapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [文件管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [GetCompressedFileSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getcompressedfilesizea)  [GetFileSizeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfilesizeex)  [GetFileType](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfiletype) |

### 还有一个枚举值：GetFileExInfoStandard

|  |
| --- |
| typedef enum \_GET\_FILEEX\_INFO\_LEVELS {  GetFileExInfoStandard,  GetFileExMaxInfoLevel  } GET\_FILEEX\_INFO\_LEVELS; |

# 扩展,这里扩展了上面的实例,代码如下

|  |
| --- |
|  |

### app.cpp

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  DWORD OutputFileTime(LPFILETIME lpft);  BOOL CreateNewFile(LPCTSTR fileName);  void ShowFileInfo(LPCTSTR fileName);  void PrintFileAttrib(DWORD attr);  void main()  {  ShowFileInfo(TEXT("demo.txt"));  //获取文件显示  system("pause");  }  DWORD OutputFileTime(LPFILETIME lpft)  {  FILETIME tmLocal;  //1.世界设计转化为当地时间  FileTimeToLocalFileTime(lpft, &tmLocal);  //2.把当地文件时间转化为系统时间  SYSTEMTIME st;  FileTimeToSystemTime(&tmLocal, &st);  wprintf(TEXT("%d-%#02d-%#02d %#02d:%#02d:%#02d\n"),  st.wYear, st.wMonth, st.wDay, st.wHour, st.wMinute, st.wSecond);  return 0;  }  BOOL CreateNewFile(LPCTSTR fileName)  {  HANDLE hFile;  DWORD dwWritten;  hFile = CreateFile(fileName, GENERIC\_WRITE, 0, 0, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE)  {  wprintf(TEXT("Create File Failed,error code:%d"), GetLastError());  CloseHandle(hFile);  return FALSE;  }  if (!WriteFile(hFile, TEXT("Hello,file"), lstrlen(TEXT("Hello,file")) \* sizeof(TCHAR), &dwWritten, NULL))  {  wprintf(TEXT("Write File Failed,error code:%d"), GetLastError());  CloseHandle(hFile);  return FALSE;  }  CloseHandle(hFile);  return TRUE;  }  void ShowFileInfo(LPCTSTR fileName)  {  WIN32\_FIND\_DATA wfd;  WIN32\_FILE\_ATTRIBUTE\_DATA wfad;  HANDLE hFind;  hFind = FindFirstFile(fileName, &wfd);  if (hFind == INVALID\_HANDLE\_VALUE)//在显示文件属性之前先查找文件  {  CreateNewFile(fileName);//如果找不到就创建它  }  //显示文件属性  if (!GetFileAttributesEx(fileName, GetFileExInfoStandard, &wfad))  {  wprintf(TEXT("Get File atrributes Failed,error code:%d"), GetLastError());  CloseHandle(hFind);  }  wprintf(TEXT("File information\n===========================================================\n"));  wprintf(TEXT("Creation time:\n"));  OutputFileTime(&wfad.ftCreationTime);  wprintf(TEXT("Access time:\n"));  OutputFileTime(&wfad.ftLastAccessTime);  wprintf(TEXT("Write time:\n"));  OutputFileTime(&wfad.ftLastWriteTime);  PrintFileAttrib(wfad.dwFileAttributes);  wprintf(TEXT("File Size:%d Bytes\n"),wfad.nFileSizeLow);  CloseHandle(hFind);  }  void PrintFileAttrib(DWORD attr)  {  wprintf(TEXT("File Attribute:"));  switch (attr)  {  case FILE\_ATTRIBUTE\_ARCHIVE:  wprintf(TEXT("Archived.\n"));  break;  case FILE\_ATTRIBUTE\_DIRECTORY:  wprintf(TEXT("Directory.\n"));  break;  case FILE\_ATTRIBUTE\_READONLY:  wprintf(TEXT("Read Only file.\n"));  break;  }  } |

### 效果

|  |
| --- |
|  |

# 扩展2.还可以做出命令行参数的版本,注意此时必须把main改为wmain

|  |
| --- |
|  |

## app-argv.cpp

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  //文件时间和属性-命令行参数版本  DWORD OutputFileTime(LPFILETIME lpft);  BOOL CreateNewFile(LPCTSTR fileName);  void ShowFileInfo(LPCTSTR fileName);  void PrintFileAttrib(DWORD attr);  int wmain(int argc,TCHAR\* argv[])//宽字符版本的带参数main需要使用wmain否则运行不正常  {  if (argc < 2)  {  wprintf(TEXT("Usage:%s <filename>\n"), argv[0]);  return -1;  }  ShowFileInfo(argv[1]);  //获取文件显示  system("pause");  return 0;  }  DWORD OutputFileTime(LPFILETIME lpft)  {  FILETIME tmLocal;  //1.世界设计转化为当地时间  FileTimeToLocalFileTime(lpft, &tmLocal);  //2.把当地文件时间转化为系统时间  SYSTEMTIME st;  FileTimeToSystemTime(&tmLocal, &st);  wprintf(TEXT("%d-%#02d-%#02d %#02d:%#02d:%#02d\n"),  st.wYear, st.wMonth, st.wDay, st.wHour, st.wMinute, st.wSecond);  return 0;  }  BOOL CreateNewFile(LPCTSTR fileName)  {  HANDLE hFile;  DWORD dwWritten;  hFile = CreateFile(fileName, GENERIC\_WRITE, 0, 0, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE)  {  wprintf(TEXT("Create File Failed,error code:%d\n"), GetLastError());  CloseHandle(hFile);  return FALSE;  }  if (!WriteFile(hFile, TEXT("Hello,file"), lstrlen(TEXT("Hello,file")) \* sizeof(TCHAR), &dwWritten, NULL))  {  wprintf(TEXT("Write File Failed,error code:%d\n"), GetLastError());  CloseHandle(hFile);  return FALSE;  }  CloseHandle(hFile);  return TRUE;  }  void ShowFileInfo(LPCTSTR fileName)  {  WIN32\_FIND\_DATA wfd;  WIN32\_FILE\_ATTRIBUTE\_DATA wfad;  HANDLE hFind;  hFind = FindFirstFile(fileName, &wfd);  if (hFind == INVALID\_HANDLE\_VALUE)//在显示文件属性之前先查找文件  {  CreateNewFile(fileName);//如果找不到就创建它  }  //显示文件属性  if (!GetFileAttributesEx(fileName, GetFileExInfoStandard, &wfad))  {  wprintf(TEXT("Get File atrributes Failed,error code:%d"), GetLastError());  CloseHandle(hFind);  }  wprintf(TEXT("File information\n===========================================================\n"));  wprintf(TEXT("Creation time:\n"));  OutputFileTime(&wfad.ftCreationTime);  wprintf(TEXT("Access time:\n"));  OutputFileTime(&wfad.ftLastAccessTime);  wprintf(TEXT("Write time:\n"));  OutputFileTime(&wfad.ftLastWriteTime);  PrintFileAttrib(wfad.dwFileAttributes);  wprintf(TEXT("File Size:%d Bytes\n"),wfad.nFileSizeLow);  CloseHandle(hFind);  }  void PrintFileAttrib(DWORD attr)  {  wprintf(TEXT("File Attribute:"));  switch (attr)  {  case FILE\_ATTRIBUTE\_ARCHIVE:  wprintf(TEXT("Archived.\n"));  break;  case FILE\_ATTRIBUTE\_DIRECTORY:  wprintf(TEXT("Directory.\n"));  break;  case FILE\_ATTRIBUTE\_READONLY:  wprintf(TEXT("Read Only file.\n"));  break;  }  } |

# 扩展3.做一个MFC对话框程序来显示文件属性

|  |
| --- |
|  |
|  |
| #define IDC\_BTN\_OPEN 1000  #define IDC\_ST\_SIZE 1001  #define IDC\_ST\_ATTRIB2 1002  #define IDC\_ST\_ATTRIB 1002  #define IDC\_ST\_ATIME 1004  #define IDC\_ST\_MTIME 1005  #define IDC\_CREATE 1006 |

## lesson4-file-attribDlg.cpp源码如下

|  |
| --- |
| // lesson4-file-attribDlg.cpp: 实现文件  //  #include "pch.h"  #include "framework.h"  #include "lesson4-file-attrib.h"  #include "lesson4-file-attribDlg.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()  // Clesson4fileattribDlg 对话框  Clesson4fileattribDlg::Clesson4fileattribDlg(CWnd\* pParent /\*=nullptr\*/)  : CDialogEx(IDD\_LESSON4FILEATTRIB\_DIALOG, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void Clesson4fileattribDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(Clesson4fileattribDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BTN\_OPEN, &Clesson4fileattribDlg::OnBnClickedBtnOpen)  END\_MESSAGE\_MAP()  // Clesson4fileattribDlg 消息处理程序  BOOL Clesson4fileattribDlg::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 Clesson4fileattribDlg::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 Clesson4fileattribDlg::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 Clesson4fileattribDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL Clesson4fileattribDlg::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:  case VK\_ESCAPE:  return TRUE;  default:  break;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  CString GetFileTImeString(LPFILETIME lpft)  {  FILETIME ftLocal;  FileTimeToLocalFileTime(lpft, &ftLocal);  SYSTEMTIME st;  FileTimeToSystemTime(&ftLocal, &st);  CString str;  str.Format(L"%d-%#02d-%#02d %#02d:%#02d:%#02d", st.wYear, st.wMonth, st.wDay,  st.wHour, st.wMinute, st.wSecond);  return str;  }  void Clesson4fileattribDlg::OnBnClickedBtnOpen()  {  // TODO: 在此添加控件通知处理程序代码  HANDLE hFile;  CString fileName,attrStr;  WIN32\_FILE\_ATTRIBUTE\_DATA wfad;  CFileDialog dlg(TRUE);  if (IDOK == dlg.DoModal())  {  fileName = dlg.GetPathName();  if (!GetFileAttributesEx(fileName, GetFileExInfoStandard, &wfad))  {  MessageBox(L"获取文件属性失败");  return;  }  //获取文件大小,需要一个文件句柄,也就是说需要先打开文件  hFile = CreateFile(fileName, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  SetDlgItemInt(IDC\_ST\_SIZE, GetFileSize(hFile, NULL));  CloseHandle(hFile);  SetDlgItemText(IDC\_CREATE, GetFileTImeString(&wfad.ftCreationTime));  SetDlgItemText(IDC\_ST\_ATIME, GetFileTImeString(&wfad.ftLastAccessTime));  SetDlgItemText(IDC\_ST\_MTIME, GetFileTImeString(&wfad.ftLastWriteTime));  switch (wfad.dwFileAttributes)  {  case FILE\_ATTRIBUTE\_ARCHIVE:  attrStr = L"Archived";  break;  case FILE\_ATTRIBUTE\_READONLY:  attrStr = L"Read Only";  break;  case FILE\_ATTRIBUTE\_HIDDEN:  attrStr = L"Hidden File";  break;  case FILE\_ATTRIBUTE\_DIRECTORY:  attrStr = L"Directory";  break;  case FILE\_ATTRIBUTE\_SYSTEM:  attrStr = L"System";  break;  case FILE\_ATTRIBUTE\_TEMPORARY:  attrStr = L"Temporary File";  break;  }  SetDlgItemText(IDC\_ST\_ATTRIB,attrStr);  }  } |