# 学习大纲

|  |
| --- |
|  |

## GetFileAttributes函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DWORD GetFileAttributesA(  [in] LPCSTR lpFileName  ); 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。 返回值 如果函数成功，则返回值包含指定文件或目录的属性。 有关属性值及其说明的列表，请参阅 下面的[文件属性常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-attribute-constants)。  如果函数失败，则返回值 **INVALID\_FILE\_ATTRIBUTES**。 要获得更多的错误信息，请调用 GetLastError。  文件属性常量   | **常量/值** | **说明** | | --- | --- | | **FILE\_ATTRIBUTE\_READONLY**  1 (0x00000001) | 只读文件。 应用程序可以读取文件，但不能写入或删除该文件。 在目录上不遵循此属性。 有关详细信息，请参阅 [无法在 Windows Server 2003、Windows XP、Windows Vista 或 Windows 7 中查看或更改文件夹的只读或系统属性](https://support.microsoft.com/kb/326549)。 | | **FILE\_ATTRIBUTE\_HIDDEN**  2 (0x00000002) | 文件或目录处于隐藏状态。 它不包括在普通目录列表中。 | | **FILE\_ATTRIBUTE\_SYSTEM**  4 (0x00000004) | 操作系统使用其中的一部分或独占使用的文件或目录。 | | **FILE\_ATTRIBUTE\_DIRECTORY**  16 (0x00000010) | 标识目录的句柄。 | | **FILE\_ATTRIBUTE\_ARCHIVE**  32 (0x00000020) | 作为存档文件或目录的文件或目录。 应用程序通常使用此属性来标记要备份或删除的文件。 | | **FILE\_ATTRIBUTE\_DEVICE**  64 (0x00000040) | 此值保留供系统使用。 | | **FILE\_ATTRIBUTE\_NORMAL**  128 (0x00000080) | 未设置其他属性的文件。 此属性仅在单独使用时才有效。 | | **FILE\_ATTRIBUTE\_TEMPORARY**  256 (0x00000100) | 用于临时存储的文件。 如果有足够的缓存内存可用，文件系统会避免将数据写回到大容量存储，因为通常情况下，应用程序在句柄关闭后会删除临时文件。 在这种情况下，系统可以完全避免写入数据。 否则，在句柄关闭后写入数据。 | | **FILE\_ATTRIBUTE\_SPARSE\_FILE**  512 (0x00000200) | 是稀疏文件的文件。 | | **FILE\_ATTRIBUTE\_REPARSE\_POINT**  1024 (0x00000400) | 具有关联的重新分析点的文件或目录，或作为符号链接的文件。 | | **FILE\_ATTRIBUTE\_COMPRESSED**  2048 (0x00000800) | 压缩的文件或目录。 对于文件，将压缩该文件中的所有数据。 对于目录，压缩是新建文件和子目录的默认设置。 | | **FILE\_ATTRIBUTE\_OFFLINE**  4096 (0x00001000) | 文件的数据不会立即可用。 此属性指示文件数据以物理方式移动到脱机存储。 此属性由远程存储（分层存储管理软件）使用。 应用程序不应随意更改此属性。 | | **FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED**  8192 (0x00002000) | 内容索引服务不会为文件或目录编制索引。 | | **FILE\_ATTRIBUTE\_ENCRYPTED**  16384 (0x00004000) | 加密的文件或目录。 对于文件，文件中的所有数据流都已加密。 对于目录，加密是新创建的文件和子目录的默认加密。 | | **FILE\_ATTRIBUTE\_INTEGRITY\_STREAM**  32768 (0x00008000) | 目录或用户数据流配置了完整性 (仅在 ReFS 卷) 上受支持。 它不包括在普通目录列表中。 如果文件已重命名，完整性设置将一直保留。 如果复制了某个文件，则如果源文件或目标目录设置了完整性，则目标文件将具有完整性设置。 **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在Windows Server 2012之前，不支持此标志。 | | **FILE\_ATTRIBUTE\_VIRTUAL**  65536 (0x00010000) | 此值保留供系统使用。 | | **FILE\_ATTRIBUTE\_NO\_SCRUB\_DATA**  131072 (0x00020000) | 后台数据完整性扫描程序不读取的用户数据流 (AKA 清理器) 。 在目录上设置时，它仅提供继承。 此标志仅在 存储空间 和 ReFS 卷上受支持。 它不包括在普通目录列表中。 **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在Windows 8和Windows Server 2012之前，不支持此标志。 | | **FILE\_ATTRIBUTE\_EA**  262144 (0x00040000) | 具有扩展属性的文件或目录。  **重要：** 此常量仅供内部使用。 | | **FILE\_ATTRIBUTE\_PINNED**  524288 (0x00080000) | 此属性指示用户意图，即即使未主动访问文件或目录，也应在本地保持完全存在。 此属性用于分层存储管理软件。 | | **FILE\_ATTRIBUTE\_UNPINNED**  1048576 (0x00100000) | 此属性指示，除非主动访问文件或目录，否则不应在本地完全存在。 此属性用于分层存储管理软件。 | | **FILE\_ATTRIBUTE\_RECALL\_ON\_OPEN**  262144 (0x00040000) | 此属性仅出现在目录枚举类 (FILE\_DIRECTORY\_INFORMATION、FILE\_BOTH\_DIR\_INFORMATION等) 中。 设置此属性时，这意味着文件或目录在本地系统上没有物理表示形式;项是虚拟的。 打开该项目的成本将比平常要高，例如，它会导致至少从远程存储中提取其中一部分。 | | **FILE\_ATTRIBUTE\_RECALL\_ON\_DATA\_ACCESS**  4194304 (0x00400000) | 如果设置了此属性，则意味着文件或目录在本地未完全存在。 对于表示并非所有数据都位于本地存储 (的文件，例如，它可能稀疏，某些数据仍位于远程存储) 中。 对于目录，这意味着某些目录内容正在从另一个位置虚拟化。 读取文件/枚举目录的成本将比平常要高，例如，这会导致至少从远程存储中提取一些文件/目录内容。 只有内核模式调用方可以设置此位。  文件系统微型筛选器低于 180000 – 189999高度范围 (FSFilter HSM 加载顺序组) 不得对设置了此属性的文件发出目标缓存读取或写入。 这可能会导致缓存污染和潜在的文件损坏。 有关详细信息，请参阅 [处理占位符](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ifs/placeholders_guidance)。 |  注解 在装载文件夹的目录上调用 **GetFileAttributes** 时，它将返回目录的文件系统属性，而不是装载的文件夹与目录关联的卷中根目录的属性。 若要获取关联卷的文件属性，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 **GetFileAttributes** 时使用生成的名称。 结果是关联卷上根目录的属性。  如果为网络共享调用 **GetFileAttributes** ，则函数将失败， [并且 GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_BAD\_NETPATH**。 必须指定该共享上的子文件夹的路径。 |

## GetFileAttributesEx函数的语法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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。 |

## SetFileAttributes函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BOOL SetFileAttributesA(  [in] LPCSTR lpFileName,  [in] DWORD dwFileAttributes  ); 参数 [in] lpFileName  要设置其属性的文件的名称。  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] dwFileAttributes  为文件设置的文件属性。  此参数可以是一个或多个值，使用按位 OR 运算符进行组合。 但是，所有其他值都替代 **FILE\_ATTRIBUTE\_NORMAL**。  此函数并非支持所有属性。 有关详细信息，请参见“备注”部分。  下面是支持的属性值列表。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_ATTRIBUTE\_ARCHIVE**  32 (0x20) | 作为存档文件或目录的文件或目录。 应用程序通常使用此属性来标记要备份或删除的文件。 | | **FILE\_ATTRIBUTE\_HIDDEN**  2 (0x2) | 文件或目录处于隐藏状态。 它不包括在普通目录列表中。 | | **FILE\_ATTRIBUTE\_NORMAL**  128 (0x80) | 未设置其他属性的文件。 此属性仅在单独使用时才有效。 | | **FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED**  8192 (0x2000) | 内容索引服务不会为文件或目录编制索引。 | | **FILE\_ATTRIBUTE\_OFFLINE**  4096 (0x1000) | 文件的数据不会立即可用。 此属性指示文件数据以物理方式移动到脱机存储。 此属性由远程存储（分层存储管理软件）使用。 应用程序不应随意更改此属性。 | | **FILE\_ATTRIBUTE\_READONLY**  1 (0x1) | 只读文件。 应用程序可以读取文件，但不能写入或删除该文件。 在目录上不遵循此属性。 有关详细信息，请参阅“无法在 Windows Server 2003、Windows XP 或 Windows Vista 中查看或更改文件夹的只读或系统属性。 | | **FILE\_ATTRIBUTE\_SYSTEM**  4 (0x4) | 操作系统使用其中的一部分或独占使用的文件或目录。 | | **FILE\_ATTRIBUTE\_TEMPORARY**  256 (0x100) | 用于临时存储的文件。 如果有足够的缓存内存可用，文件系统会避免将数据写回到大容量存储，因为通常情况下，应用程序在句柄关闭后会删除临时文件。 在这种情况下，系统可以完全避免写入数据。 否则，在句柄关闭后写入数据。 |  返回值 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 下表介绍了如何设置无法使用 **SetFileAttributes** 设置的属性。 有关所有文件属性值及其说明的完整列表，请参阅 [文件属性常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-attribute-constants)。  展开表   |  |  | | --- | --- | | **属性** | **如何设置** | | **FILE\_ATTRIBUTE\_COMPRESSED**  0x800 | 若要设置文件的压缩状态，请将 [DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 函数与 [FSCTL\_SET\_COMPRESSION](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_set_compression) 操作一起使用。 | | **FILE\_ATTRIBUTE\_DEVICE**  0x40 | 保留;请勿使用。 | | **FILE\_ATTRIBUTE\_DIRECTORY**  0x10 | 文件无法转换为目录。 若要创建目录，请使用 [CreateDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createdirectorya) 或 [CreateDirectoryEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createdirectoryexa) 函数。 | | **FILE\_ATTRIBUTE\_ENCRYPTED**  0x4000 | 若要创建加密文件，请将 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 函数与 **FILE\_ATTRIBUTE\_ENCRYPTED** 属性一起使用。 若要将现有文件转换为加密文件，请使用 [EncryptFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-encryptfilea) 函数。 | | **FILE\_ATTRIBUTE\_REPARSE\_POINT**  0x400 | 若要将重新分析点与文件或目录相关联，请将 [DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 函数与 [FSCTL\_SET\_REPARSE\_POINT](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_set_reparse_point) 操作一起使用。 | | **FILE\_ATTRIBUTE\_SPARSE\_FILE**  0x200 | 若要设置文件的稀疏属性，请将 [DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 函数与 [FSCTL\_SET\_SPARSE](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_set_sparse) 操作一起使用。 |    事务处理操作 如果文件在事务中打开以供修改，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修改文件之前修改了文件，并且当事务尝试打开该文件时该文件仍处于打开状态，则事务 **会收到错误ERROR\_TRANSACTIONAL\_CONFLICT**。 |

## FileTimeToLocalFileTime函数的语法

|  |
| --- |
| 语法 BOOL FileTimeToLocalFileTime(  [in] const FILETIME \*lpFileTime,  [out] LPFILETIME lpLocalFileTime  ); 参数 [in] lpFileTime  指向 [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构的指针，其中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/zh-cn/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 若要考虑将文件时间转换为本地时间时的夏令时，请使用以下函数序列代替使用 **FileTimeToLocalFileTime**：   1. [FileTimeToSystemTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/timezoneapi/nf-timezoneapi-filetimetosystemtime) 2. [SystemTimeToTzSpecificLocalTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/timezoneapi/nf-timezoneapi-systemtimetotzspecificlocaltime) 3. [SystemTimeToFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/timezoneapi/nf-timezoneapi-systemtimetofiletime) |

## FileTimeToSystemTime函数的语法

|  |
| --- |
| 语法 BOOL FileTimeToSystemTime(  [in] const FILETIME \*lpFileTime,  [out] LPSYSTEMTIME lpSystemTime  ); 参数 [in] lpFileTime  指向 [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构的指针，其中包含要转换为系统的文件时间 (UTC) 日期和时间格式。  此值必须小于 0x8000000000000000。 否则，该函数将失败。  [out] lpSystemTime  指向 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime) 结构的指针，用于接收转换的文件时间。 返回值 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。 |

## WIN32\_FILE\_ATTRIBUTE\_DATA结构体

|  |
| --- |
| C++复制  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)。 |

### 注意：保存在这个结构体里面的文件大小信息被保存在两个字段里面，需要用他们来计算计算才能得到真正的文件大小

## FILETIME结构体

|  |
| --- |
| typedef struct \_FILETIME {  DWORD dwLowDateTime;  DWORD dwHighDateTime;  } FILETIME, \*PFILETIME, \*LPFILETIME; 成员 dwLowDateTime  文件时间的低序部分。  dwHighDateTime  文件时间的高阶部分。 注解 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/zh-cn/windows/desktop/api/timezoneapi/nf-timezoneapi-filetimetosystemtime) 函数。  建议不要从 **FILETIME** 结构中添加和减去值来获取相对时间。 相反，应将文件时间的低位和高阶部分复制到 [ULARGE\_INTEGER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-ularge_integer-r1) 结构，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，因为这可能会导致 64 位 Windows 上的对齐错误。  并非所有文件系统都可以记录创建时间和上次访问时间，并非所有文件系统都以相同的方式记录它们。 例如，在 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， (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/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfiletime) 函数可能不会返回使用 [SetFileTime](https://learn.microsoft.com/zh-cn/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 格式记录磁盘上的时间。 有关详细信息，请参阅[文件时间](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/file-times)。  使用 **FILETIME** 结构的函数可以允许在 **dwLowDateTime** 和 **dwHighDateTime** 成员通常指定的零个值或正值之外的值。 例如， [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 函数使用 0xFFFFFFFF 来指定应保留文件的先前访问时间。 有关详细信息，请参阅要调用的函数的主题。 |

## SYSTEMTIME结构体

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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  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   |  |  | | --- | --- | | **值** | **含义** | | 1 | 1 月 | | 2 | February | | 3 | 3 月 | | 4 | April | | 5 | 五月 | | 6 | June | | 7 | 7 月 | | 8 | 8 月 | | 9 | 9 月 | | 10 | 10 月 | | 11 | 11 月 | | 12 | 12 月 |   wDayOfWeek  星期几。 此成员可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | 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  每月的日期。 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  秒钟。 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。 注解 **备注**  **SYSTEMTIME** 不检查查看表示的日期是否为实际有效日期。 使用此 API 时，应确保其有效性，尤其是在闰年方案中。 有关详细信息 [**，请参阅闰日就绪情况**](https://techcommunity.microsoft.com/t5/azure-developer-community-blog/it-s-2020-is-your-code-ready-for-leap-day/ba-p/1157279) 。  建议不要从 **SYSTEMTIME** 结构中添加和减去值来获取相对时间。 相反，你应该   * 将 **SYSTEMTIME** 结构转换为 [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。 * 将生成的 [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构复制到 [ULARGE\_INTEGER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-ularge_integer-r1) 结构。 * 对 [ULARGE\_INTEGER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-ularge_integer-r1) 值使用正常的 64 位算术。   系统可以通过与时间源同步来定期刷新时间。 由于系统时间可以向前或向后调整，因此请勿比较系统时间读数以确定已用时间。 请改用 [Windows 时间](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/windows-time)中所述的方法之一。 示例 以下示例演示 [GetSystemTime 和 GetLocalTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemtime) 函数检索的时间值之间的差异。  C++复制  #include <windows.h>  #include <stdio.h>  void main()  {  SYSTEMTIME st, lt;    GetSystemTime(&st);  GetLocalTime(&lt);    printf("The system time is: %02d:%02d\n", st.wHour, st.wMinute);  printf(" The local time is: %02d:%02d\n", lt.wHour, lt.wMinute);  }  syntax复制  // Sample output  The system time is: 19:34  The local time is: 12:34 |

### 需要注意：如果先在控制台里面显示中文，需要包含Locale.h头文件，而且需要在程序的开始部分调用setlocale(LC\_ALL,””)

# 演练

## 项目1.获取文件属性

## 1.新建一个c++常规空项目，取名Lesson21-file-attr-time，并且新建一个cpp源文件，取名： file-attr-time.cpp

|  |
| --- |
|  |

## 2.先搭建文件骨架代码如下

|  |
| --- |
|  |

## 3.完整代码如下

### file-attr-time.cpp

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  DWORD DisplayFileTime(LPFILETIME lpfileTime)  {  FILETIME localFileTime;  SYSTEMTIME sysTime;  if(!FileTimeToLocalFileTime(lpfileTime,&localFileTime))  {  printf("Convert File Time To Local Time Failed: %d\n",GetLastError());  return 1;  }  if(!FileTimeToSystemTime(&localFileTime,&sysTime))  {  printf("Convert File Time To System Time Failed: %d\n",GetLastError());  return 1;  }  printf("%d年%#02d月%#02d日 %#02d:%#02d:%#02d\n",sysTime.wYear,sysTime.wMonth,sysTime.wDay,sysTime.wHour,sysTime.wMinute,sysTime.wSecond);  return 0;  }  DWORD ShowFileSize(LPWIN32\_FILE\_ATTRIBUTE\_DATA lpw32fad)  {  ULONGLONG lgSize;//64位  DWORD hiSize,lwSize;  hiSize = lpw32fad->nFileSizeHigh;//  lwSize = lpw32fad->nFileSizeLow;  lgSize = hiSize;//先接收高字段的数据  lgSize <<= 32;//然后左移32位，也就是移动到高32位  lgSize += lwSize; //把低字段的值添加进来  printf("%I64u bytes\n",lgSize);  return 0;  }  DWORD ShowFileAttributs(LPTSTR szPath)  {  WIN32\_FILE\_ATTRIBUTE\_DATA w32fad;    if(!GetFileAttributesEx(szPath,GetFileExInfoStandard,&w32fad))  {  printf("Get File Attributes Failed: %x\n",GetLastError());  return 1;  }  else  { printf("========File Report Of %s======== \n",szPath);    //printf("%d\n",w32fad.dwFileAttributes);    printf("Creation Time: ");  DisplayFileTime(&w32fad.ftCreationTime);  printf("Access Time: ");  DisplayFileTime(&w32fad.ftLastAccessTime);  printf("Last Modify Time: ");  DisplayFileTime(&w32fad.ftLastWriteTime);  printf("File Size: ");  ShowFileSize(&w32fad);  //下面的代码不要改为switch语句，没有效果或者效果不对  if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_NORMAL)  {  printf("%s is a normal file\n",szPath);  }  else if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_READONLY)  {  printf("%s is a readonly file\n",szPath);  }  else if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_HIDDEN)  {  printf("%s is a hidden file\n",szPath);  }  else if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_ARCHIVE)  {  printf("%s has been Archived...\n",szPath);  }  else if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_COMPRESSED)  {  printf("%s has been Compressed...\n",szPath);  }  else if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_ENCRYPTED)  {  printf("%s has been Encrypted...\n",szPath);  }  else if(w32fad.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)  {  printf("%s is a directory\n",szPath);  }    }    return 0;  }  int main()  {  ShowFileAttributs(TEXT("Debug"));  //ShowFileAttributs(TEXT("test.txt"));  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

## 宽字符版本

|  |
| --- |
|  |
| #include<Windows.h>  #include<iostream>  void OutputFileTime(LPFILETIME lpft)  {  FILETIME ftLocal;  SYSTEMTIME st;  if (!FileTimeToLocalFileTime(lpft, &ftLocal))  {  wprintf\_s(L"Convert File Time To local time failed ,code:%d", GetLastError());  return ;  }  if (!FileTimeToSystemTime(&ftLocal,&st))  {  wprintf\_s(L"Convert Local Time To system time failed ,code:%d", GetLastError());  return;  }  wprintf\_s(L"%d-%#02d-%#02d %#02d:%#02d:%#02d\n", st.wYear,st.wMonth,st.wDay,st.wHour,st.wMinute,st.wSecond);  }  void MyGetFileSize(WIN32\_FILE\_ATTRIBUTE\_DATA wfad,DWORD\* lpdwSize)  {  ULONGLONG lgSize;  DWORD hiSize, lwSize;  hiSize = wfad.nFileSizeHigh;  lwSize = wfad.nFileSizeLow;  lgSize = hiSize;  lgSize <<= 32; //64位的数字左移32位  lgSize += lwSize;//接收低32位  \*lpdwSize = lgSize;  }  void ShowFileAtrribStr(DWORD attr)  {  WCHAR szAttr[256];  memset(szAttr, 0, 256);  switch (attr)  {  case FILE\_ATTRIBUTE\_ARCHIVE:  wprintf\_s(L"File Attribute:Archived\n");  break;  case FILE\_ATTRIBUTE\_COMPRESSED:  wprintf\_s(L"File Attribute:Compressed\n");  break;  case FILE\_ATTRIBUTE\_READONLY:  wprintf\_s(L"File Attribute:Read Only\n");  break;  case FILE\_ATTRIBUTE\_HIDDEN:  wprintf\_s(L"File Attribute:Hidden\n");  break;  case FILE\_ATTRIBUTE\_ENCRYPTED:  wprintf\_s(L"File Attribute:Encrypted\n");  break;  case FILE\_ATTRIBUTE\_DIRECTORY:  wprintf\_s(L"File Attribute:Directory\n");  break;  }  }  DWORD ShowFileAttrib(const WCHAR\* fileName)  {  WIN32\_FILE\_ATTRIBUTE\_DATA wfad;  DWORD dwSize;  if (!GetFileAttributesEx(fileName, GetFileExInfoStandard, &wfad))  {  wprintf\_s(L"Get File Attributes failed ,code:%d", GetLastError());  return -1;  }  wprintf\_s(L"File Atributes:\n================================\n");  wprintf\_s(L"Creation Time:");  OutputFileTime(&wfad.ftCreationTime);  wprintf\_s(L"Last Access Time:");  OutputFileTime(&wfad.ftLastAccessTime);  wprintf\_s(L"Last Write Time:");  OutputFileTime(&wfad.ftLastWriteTime);  MyGetFileSize(wfad, &dwSize);  wprintf\_s(L"File Size:%I64uBytes\n",(unsigned \_\_int64)dwSize);  ShowFileAtrribStr(wfad.dwFileAttributes);  return 0;  }  int wmain()  {  ShowFileAttrib(TEXT("Debug"));  system("pause");  return 0;  } |

## 项目2，设置文件属性

## 1，新建一个c++常规空项目，取名：Lesson21\_2-set-attr，然后新建一个cpp文件取名：set-attr.cpp

|  |
| --- |
|  |

## 2.输入下面的代码

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  DWORD doSetFileAtrribute(LPTSTR szPath)  {  DWORD dwAttr;  dwAttr = GetFileAttributes(szPath); //设置文件属性不能直接设置，需要先获取文件属性然后把它和新属性进行位或运算，再设置结果到文件属性  //将原来的属性和需要设置的属性进行或存在  /\*dwAttr |= FILE\_ATTRIBUTE\_HIDDEN;\*///添加隐藏属性  //dwAttr -= FILE\_ATTRIBUTE\_HIDDEN;//去除隐藏属性  //dwAttr |= FILE\_ATTRIBUTE\_READONLY;  dwAttr -= FILE\_ATTRIBUTE\_READONLY;  if(!SetFileAttributes(szPath,dwAttr))  {  printf("Set File Attributes Failed: %d",GetLastError());  return -1;  }  else  {  printf("Set File Attributes Successfully\n");  }  //MessageBox(NULL,TEXT("Set File Attributes Successfully"),TEXT("Result"),MB\_OK);  return 0;  }  int main()  {  doSetFileAtrribute(TEXT("hi.txt"));  system("pause");  return 0;  } |

### 注意：设置属性需要先获取文件原来有哪些属性，然后需要添加的属性使用位或操作|=，去除属性可以使用-=来去除

# 扩展：可以把这个下面做成和前面的fileutil项目类似的功能,如：+r表示添加只读属性，-r表示去除只读属性，其他属性如此类推

|  |
| --- |
|  |
| #include<Windows.h>  #include<iostream>  int wmain(int argc, PWCHAR argv[])  {  wprintf\_s(L"%s\n",argv[2]);  if (argc != 3)  {  wprintf\_s(L"Usage:%s +r||-r||+h||-h filename\n ",argv[1]);  return -1;  }  DWORD attr;  attr = GetFileAttributes(argv[2]);  if (lstrcmp(argv[1], L"+r") == 0)  {  attr |= FILE\_ATTRIBUTE\_READONLY;  }  else if (lstrcmp(argv[1], L"-r") == 0)  {  attr -= FILE\_ATTRIBUTE\_READONLY;  }  else if (lstrcmp(argv[1], L"+h") == 0)  {  attr |= FILE\_ATTRIBUTE\_HIDDEN;  }  else if (lstrcmp(argv[1], L"-h") == 0)  {  attr -= FILE\_ATTRIBUTE\_HIDDEN;  }  if (!SetFileAttributes(argv[2], attr))  {  wprintf\_s(L"Set File Attributes Failed: %d\n", GetLastError());  return -1;  }  wprintf\_s(L"Set File Attributes with success\n");  return 0;  } |

## 效果

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

# 这一节的学习到此为止