# 学习大纲

|  |
| --- |
|  |

## DeleteFile的语法

|  |
| --- |
| BOOL DeleteFile(  [in] LPCTSTR lpFileName  ); 参数 [in] lpFileName  要删除的文件的名称。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。 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 （0）。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 如果应用程序尝试删除不存在的文件，**则 DeleteFile** 函数将失败，并显示**ERROR\_FILE\_NOT\_FOUND**。如果文件是只读文件，则函数将失败，并显示 **ERROR\_ACCESS\_DENIED**。  以下列表列出了有关删除、删除或关闭文件的一些提示：   * 要删除只读文件，首先必须删除 read-only 属性。 * 要删除或重命名文件，您必须具有该文件的 delete 权限或 delete child 权限 在父目录中。 * 若要以递归方式删除目录中的文件，请使用 [SHFileOperation](https://learn.microsoft.com/en-us/windows/desktop/api/shellapi/nf-shellapi-shfileoperationa) 函数。 * 要删除空目录，请使用 [RemoveDirectory](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-removedirectorya) 函数。 * 若要关闭打开的文件，请使用 [CloseHandle](https://learn.microsoft.com/en-us/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数。   如果设置的目录具有除 delete 和 delete child 之外的所有访问权限，并且 继承新文件，则可以创建文件但无法删除它。但是，您可以创建 一个文件，然后获取您在创建 文件。  如果您在创建文件时请求删除权限，则可以使用该权限删除或重命名文件 handle，但不能与任何其他 handle.有关更多信息，请参阅[文件安全性和访问权限](https://learn.microsoft.com/en-us/windows/desktop/FileIO/file-security-and-access-rights)。  如果应用程序尝试 删除为正常 I/O 打开了其他句柄的文件或作为内存映射文件 （打开其他句柄时，必须指定 **FILE\_SHARE\_DELETE**）。  **DeleteFile** 函数将文件标记为删除 关闭。因此，在关闭文件的最后一个句柄之前，不会删除文件。后续调用 到 [CreateFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) 以打开文件，但**ERROR\_ACCESS\_DENIED**失败。  符号链接行为 -  如果路径指向符号链接，则删除符号链接，而不是目标。要删除目标，请执行以下操作 必须调用 [CreateFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-createfilea) 并指定 **FILE\_FLAG\_DELETE\_ON\_CLOSE**。 |

## CopyFile的语法

|  |
| --- |
| 将现有文件复制到新文件。  [CopyFileEx](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-copyfileexa) 函数提供两个额外的 能力。**CopyFileEx** 可以调用指定的回调 每次完成部分复制操作时，都可以执行此函数，并且可以在复制操作期间取消 **CopyFileEx**。  要将此操作作为事务处理操作执行，请使用 [CopyFileTransacted](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-copyfiletransacteda) 函数。 语法 BOOL CopyFile(  [in] LPCTSTR lpExistingFileName,  [in] LPCTSTR lpNewFileName,  [in] BOOL bFailIfExists  ); 参数 [in] lpExistingFileName  现有文件的名称。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  如果 *lpExistingFileName* 不存在，**则 CopyFile** 将失败，[并且 GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_FILE\_NOT\_FOUND**。  [in] lpNewFileName  新文件的名称。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in] bFailIfExists  如果此参数为 **TRUE**，并且 *lpNewFileName* 指定的新文件已存在，则函数将失败。如果此参数为 **FALSE** 且新文件已存在，则函数将覆盖现有文件 成功。 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 现有文件的安全资源属性 （**ATTRIBUTE\_SECURITY\_INFORMATION**） 为 复制到新文件。  **Windows 7、Windows Server 2008 R2、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows 8 和 Windows Server 2012。  现有文件的文件属性将复制到新文件中。例如，如果现有文件具有 **FILE\_ATTRIBUTE\_READONLY** file 属性，则通过调用 **CopyFile** 创建的副本也将具有 **FILE\_ATTRIBUTE\_READONLY** file 属性。有关更多信息，请参阅[检索和更改文件属性](https://learn.microsoft.com/en-us/windows/desktop/FileIO/retrieving-and-changing-file-attributes)。  如果目标文件已存在，则此函数将失败，并显示 **ERROR\_ACCESS\_DENIED** 并设置了 **FILE\_ATTRIBUTE\_HIDDEN** 或 **FILE\_ATTRIBUTE\_READONLY** 属性。  当 **CopyFile** 用于复制加密文件时，它会尝试 以使用源文件加密中使用的密钥加密目标文件。如果无法做到这一点， 此函数尝试使用默认密钥加密目标文件。如果 这两种方法都无法完成，**CopyFile** 会失败并显示 **ERROR\_ENCRYPTION\_FAILED** 错误代码。  符号链接行为 - 如果源文件是符号链接，则实际复制的文件为 符号链接的目标。  如果目标文件已存在并且是符号链接，则符号链接的目标将被覆盖 按源文件。 |

## 还有一个CopyFileEx函数，语法如下

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CopyFileExA 函数 （winbase.h） 将现有文件复制到新文件，并通过回调通知应用程序其进度 功能。  要将此操作作为事务处理操作执行，请使用 [CopyFileTransacted](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-copyfiletransacteda) 函数。 语法 BOOL CopyFileExA(  [in] LPCSTR lpExistingFileName,  [in] LPCSTR lpNewFileName,  [in, optional] LPPROGRESS\_ROUTINE lpProgressRoutine,  [in, optional] LPVOID lpData,  [in, optional] LPBOOL pbCancel,  [in] DWORD dwCopyFlags  ); 参数 [in] lpExistingFileName  现有文件的名称。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  如果 *lpExistingFileName* 不存在，则 **CopyFileEx** 函数将失败， [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_FILE\_NOT\_FOUND**。  [in] lpNewFileName  新文件的名称。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in, optional] lpProgressRoutine  调用 **LPPROGRESS\_ROUTINE** 类型的回调函数的地址 每次复制文件的另一部分时。此参数可以为 **NULL。**为 有关 Progress 回调函数的更多信息，请参阅 [CopyProgressRoutine](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nc-winbase-lpprogress_routine) 函数。  [in, optional] lpData  要传递给回调函数的参数。此参数可以为 **NULL。**  [in, optional] pbCancel  如果在复制操作期间将此标志设置为 **TRUE**，则取消该操作。 否则，复制操作将继续完成。  [in] dwCopyFlags  指定如何复制文件的标志。此参数可以是以下各项的组合 值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **COPY\_FILE\_ALLOW\_DECRYPTED\_DESTINATION**  0x00000008 | 即使目标副本无法加密，尝试复制加密文件也会成功。 | | **COPY\_FILE\_COPY\_SYMLINK**  0x00000800 | 如果源文件是符号链接，则目标文件也是指向同一文件的符号链接 源符号链接指向的文件。  **Windows Server 2003 和 Windows XP：**不支持此值。 | | **COPY\_FILE\_FAIL\_IF\_EXISTS**  0x00000001 | 如果目标文件已存在，则复制操作将立即失败。 | | **COPY\_FILE\_NO\_BUFFERING**  0x00001000 | 复制操作使用无缓冲的 I/O 执行，绕过系统 I/O 高速缓存资源。推荐 用于非常大的文件传输。  **Windows Server 2003 和 Windows XP：**不支持此值。 | | **COPY\_FILE\_OPEN\_SOURCE\_FOR\_WRITE**  0x00000004 | 将复制该文件，并打开原始文件以进行写入访问。 | | **COPY\_FILE\_RESTARTABLE**  0x00000002 | 如果复制失败，则会在目标文件中跟踪复制的进度。失败的副本可以是 稍后通过为 *lpExistingFileName* 和 *lpNewFileName* 指定与失败的调用中使用的值相同的值来重新启动。这可能会显着减慢 关闭 COPY 操作，因为在 COPY 操作期间可能会多次刷新新文件。 | | **COPY\_FILE\_REQUEST\_COMPRESSED\_TRAFFIC**  0x10000000 | 请求底层传输通道在复制操作期间压缩数据。并非所有媒体都支持该请求，在这种情况下，将忽略该请求。压缩属性和参数（计算复杂度、内存使用量）无法通过此 API 进行配置，并且可能会因不同的操作系统版本而异。  此标志是在 Windows 10 版本 1903 和 Windows Server 2022 中引入的。在 Windows 10 上，驻留在 SMB 共享上的文件支持该标志，其中协商的 SMB 协议版本为 SMB v3.1.1 或更高版本。 |  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)。  如果 *lpProgressRoutine* **PROGRESS\_CANCEL**由于 用户取消操作时，**CopyFileEx** 将返回零 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_REQUEST\_ABORTED**。在这种情况下，部分复制的目标文件为 删除。  如果 *lpProgressRoutine* **PROGRESS\_STOP**由于 用户停止操作时，**CopyFileEx** 将返回零 [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_REQUEST\_ABORTED**。在这种情况下，将保留部分复制的目标文件 完整。 言论 此函数保留扩展属性、OLE 结构化存储、NTFS 文件系统备用数据流、 security 资源属性和 file 属性。  **Windows 7、Windows Server 2008 R2、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows 8 之前，现有文件的安全资源属性 （**ATTRIBUTE\_SECURITY\_INFORMATION**） 不会复制到新文件 和 Windows Server 2012。  现有文件的安全资源属性 （**ATTRIBUTE\_SECURITY\_INFORMATION**） 为 复制到新文件。  **Windows 7、Windows Server 2008 R2、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows 8 和 Windows Server 2012。  如果目标文件已存在，则此函数将失败，并显示 **ERROR\_ACCESS\_DENIED** 并设置了 **FILE\_ATTRIBUTE\_HIDDEN** 或 **FILE\_ATTRIBUTE\_READONLY** 属性。  使用 **CopyFileEx** 复制加密文件时， 函数尝试使用源文件加密中使用的密钥加密目标文件。如果 这是无法完成的，此函数尝试使用默认密钥加密目标文件。如果两者都是 方法，**则 CopyFileEx** 会失败，并显示 **ERROR\_ENCRYPTION\_FAILED** 错误代码。如果希望 **CopyFileEx** 完成复制操作，即使 目标文件无法加密，请在调用 **CopyFileEx** 时将 **COPY\_FILE\_ALLOW\_DECRYPTED\_DESTINATION** 作为 *dwCopyFlags* 参数的值。  如果指定了 **COPY\_FILE\_COPY\_SYMLINK**，则以下规则适用：   * 如果源文件是符号链接，则复制符号链接，而不是目标文件。 * 如果源文件不是符号链接，则行为不会发生变化。 * 如果目标文件是现有的符号链接，则符号链接将被覆盖，而不是目标 文件。 * 如果还指定了 **COPY\_FILE\_FAIL\_IF\_EXISTS**，并且目标文件是 existing symbolic link 的实例，则操作在所有情况下都会失败。   如果未指定 **COPY\_FILE\_COPY\_SYMLINK**，则适用以下规则：   * 如果还指定了 **COPY\_FILE\_FAIL\_IF\_EXISTS**，并且目标文件是 existing symbolic link，则仅当符号链接的目标存在时，操作才会失败。 * 如果未指定 **COPY\_FILE\_FAIL\_IF\_EXISTS**，则行为不会发生变化。   **Windows 7、Windows Server 2008 R2、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**如果要编写的应用程序正在优化 LAN 中的文件复制操作，请考虑使用 Windows 套接字中的 [TransmitFile](https://learn.microsoft.com/en-us/windows/desktop/api/mswsock/nf-mswsock-transmitfile) 函数 （Winsock） 的 Shell。**TransmitFile** 支持高性能 网络传输，并提供一个简单的接口来将文件内容发送到远程计算机。要使用 **TransmitFile**，您必须编写一个 Winsock 客户端 从源计算机发送文件的应用程序，以及使用 其他 Winsock 函数在远程计算机上接收文件。 |

## MoveFile函数的语法参数

|  |
| --- |
| MoveFile 函数 （winbase.h） 移动现有文件或目录，包括其子项。  若要指定如何移动文件，请使用 [MoveFileEx](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-movefileexa) 或 [MoveFileWithProgress](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-movefilewithprogressa) 函数。  若要将此操作作为事务处理操作执行，请使用 [MoveFileTransacted](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-movefiletransacteda) 函数。 语法 C++复制  BOOL MoveFile(  [in] LPCTSTR lpExistingFileName,  [in] LPCTSTR lpNewFileName  ); 参数 [in] lpExistingFileName  本地计算机上文件或目录的当前名称。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in] lpNewFileName  文件或目录的新名称。新名称必须不存在。新文件可能位于 不同的文件系统或驱动器。新目录必须位于同一驱动器上。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。 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 **MoveFile** 函数将移动（重命名）文件或 目录（包括其子目录）位于同一目录或跨目录中。需要注意的是，当 destination 位于不同的卷上。  如果跨卷移动文件，**则 MoveFile** 不会移动 包含文件的安全描述符。该文件将在 destination 目录中。  **MoveFile** 函数将其操作与 Link Tracking 服务，因此可以在链接源移动时对其进行跟踪。 |

## MoveFileEx，语法如下

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MoveFileExA 函数 （winbase.h） 使用各种移动选项移动现有文件或目录，包括其子项。  [MoveFileWithProgress](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-movefilewithprogressa) 函数是等效的 添加到 **MoveFileEx** 函数中，但 **MoveFileWithProgress** 允许您提供 接收进度通知的 callback 函数。 语法 BOOL MoveFileExA(  [in] LPCSTR lpExistingFileName,  [in, optional] LPCSTR lpNewFileName,  [in] DWORD dwFlags  ); 参数 [in] lpExistingFileName  本地计算机上文件或目录的当前名称。  如果 *dwFlags* 指定 **MOVEFILE\_DELAY\_UNTIL\_REBOOT**，则 文件不能存在于远程共享上，因为延迟操作是在网络 可用。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in, optional] lpNewFileName  本地计算机上文件或目录的新名称。  移动文件时，目标可以位于不同的文件系统或卷上。如果目标处于开启状态 另一个驱动器，则必须在 *dwFlags* 中设置 **MOVEFILE\_COPY\_ALLOWED** 标志。  移动目录时，目标必须位于同一驱动器上。  如果 *dwFlags* 指定 **MOVEFILE\_DELAY\_UNTIL\_REBOOT** 且 *lpNewFileName* 为 **NULL，则 MoveFileEx** 会注册要在系统重启时删除的 *lpExistingFileName* 文件。如果 *lpExistingFileName* 引用目录，则系统会在重新启动时删除该目录 仅当目录为空时。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in] dwFlags  此参数可以是以下一个或多个值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MOVEFILE\_COPY\_ALLOWED**  2 （0x2） | 如果要将文件移动到其他卷，该函数将使用 [CopyFile](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-copyfile) 和 [DeleteFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-deletefilea) 函数模拟移动。  如果文件已成功复制到其他卷，但无法删除原始文件， 该函数成功，使源文件保持不变。  此值不能与 **MOVEFILE\_DELAY\_UNTIL\_REBOOT** 一起使用。 | | **MOVEFILE\_CREATE\_HARDLINK**  16 （0x10） | 保留供将来使用。 | | **MOVEFILE\_DELAY\_UNTIL\_REBOOT**  4 （0x4） | 在重新启动操作系统之前，系统不会移动文件。系统移动文件 在执行 AUTOCHK 之后，但在创建任何分页文件之前。因此，此参数 启用该函数以从以前的启动中删除分页文件。  仅当进程位于属于管理员的用户的上下文中时，才能使用此值 group 或 LocalSystem 帐户。  此值不能与 **MOVEFILE\_COPY\_ALLOWED** 一起使用。 | | **MOVEFILE\_FAIL\_IF\_NOT\_TRACKABLE**  32 （0x20） | 如果源文件是链接源，但在移动后无法跟踪该文件，则函数将失败。 如果目标是使用 FAT 文件系统格式化的卷，则可能会出现这种情况。 | | **MOVEFILE\_REPLACE\_EXISTING**  1 （0x1） | 如果存在名为 *lpNewFileName* 的文件，该函数会将其内容替换为 *lpExistingFileName* 文件的内容，前提是安全性要求 满足访问控制列表 （ACL）。有关详细信息，请参阅此 的 “备注 ”部分 主题。  如果 *lpNewFileName* 命名现有目录，则会报告错误。 | | **MOVEFILE\_WRITE\_THROUGH**  8 （0x8） | 在文件实际移动到磁盘上之前，该函数不会返回。  设置此值可保证作为复制和删除操作执行的移动将刷新到磁盘 在函数返回之前。刷新发生在复制操作结束时。  如果设置了 **MOVEFILE\_DELAY\_UNTIL\_REBOOT**，则此值不起作用。 |  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 （0）。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 *dwFlags* 参数指定 **MOVEFILE\_DELAY\_UNTIL\_REBOOT**，则 **MoveFileEx** 在无法访问注册表时失败。这 函数将重新启动时要重命名的文件的位置存储在以下注册表值中：**HKEY\_LOCAL\_MACHINE**\**SYSTEM**\**CurrentControlSet**\**Control**\**Session Manager**\**PendingFileRenameOperations**  此注册表值为 **REG\_MULTI\_SZ** 类型。每个重命名操作都会存储一个 的字符串，具体取决于重命名是否为 DELETE：   * *szSrcFile*\0\0 * *szSrcFile*\0*szDstFile*\0   字符串 *szSrcFile*\0\0 表示将在重新启动时删除文件 *szSrcFile*。字符串 *szSrcFile*\0*szDstFile*\0 表示在重新启动时将 *szSrcFile* 重命名为 *szDstFile*。  **注意**尽管 \0\0 在技术上不允许在 **REG\_MULTI\_SZ** 节点中使用，但它可以，因为 该文件被视为已重命名为 null 名称。    系统使用这些注册表项在重新启动时按与它们相同的顺序完成操作 发出。例如，以下代码片段创建注册表项，这些注册表项在重新启动时删除 *szSrcFile* 并将 *szSrcFile* 重命名为 *szDstFile*：  C++复制  MoveFileEx(szSrcFile, NULL, MOVEFILE\_DELAY\_UNTIL\_REBOOT);  MoveFileEx(szSrcFile, szDstFile, MOVEFILE\_DELAY\_UNTIL\_REBOOT);  因为使用 **MOVEFILE\_DELAY\_UNTIL\_REBOOT** 标志指定的实际移动和删除操作发生在调用应用程序停止之后 运行，则返回值无法反映移动或删除文件的成功或失败。相反，它反映了 将适当的条目放入注册表中的成功或失败。  仅当目录为空时，系统才会删除使用 **MOVEFILE\_DELAY\_UNTIL\_REBOOT** 标志标记为删除的目录。要确保删除目录， 在尝试删除目录之前，先移动或删除目录中的所有文件。文件可能在启动时位于 目录中 时间，但必须先删除或移动它们，然后系统才能删除目录。  移动和删除操作在引导时执行，其顺序与 调用 application。要在引导时删除包含文件的目录，请首先删除这些文件。  如果跨卷移动文件，**则 MoveFileEx** 不会 将安全描述符与文件一起移动。该文件在 destination 目录中。  **MoveFileEx** 函数与 [链接跟踪](https://learn.microsoft.com/en-us/windows/desktop/FileIO/distributed-link-tracking-and-object-identifiers)服务， 因此，可以在链接源移动时对其进行跟踪。  要删除或重命名文件，您必须具有该文件的 delete 权限或 delete child 权限 父目录。如果您设置的目录具有除 delete 和 delete child 之外的所有访问权限，并且 ACL 的 ACL 新文件被继承，那么您应该能够创建文件，但无法删除它。但是，您 然后，可以创建一个文件，并获取您在 您创建文件。如果您在创建文件时请求删除权限，则可以删除或重命名 具有该 handle 的文件，但不包含任何其他 handle 的文件。有关更多信息，请参阅[文件安全性和访问权限](https://learn.microsoft.com/en-us/windows/desktop/FileIO/file-security-and-access-rights)。 |

## MoveFileWithProgressA函数，语法如下

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MoveFileWithProgressA 函数 （winbase.h） 移动文件或目录，包括其子项。您可以提供一个回调函数，该函数接收 进度通知。 语法 BOOL MoveFileWithProgressA(  [in] LPCSTR lpExistingFileName,  [in, optional] LPCSTR lpNewFileName,  [in, optional] LPPROGRESS\_ROUTINE lpProgressRoutine,  [in, optional] LPVOID lpData,  [in] DWORD dwFlags  ); 参数 [in] lpExistingFileName  本地计算机上现有文件或目录的名称。  如果 *dwFlags* 指定 **MOVEFILE\_DELAY\_UNTIL\_REBOOT**，则 文件不能存在于远程共享上，因为延迟操作是在网络 可用。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in, optional] lpNewFileName  本地计算机上文件或目录的新名称。  移动文件时， *lpNewFileName* 可以位于不同的文件系统或卷上。如果 *lpNewFileName* 位于另一个驱动器上，则必须在 *dwFlags* 中设置 **MOVEFILE\_COPY\_ALLOWED** 标志。  移动目录时， *lpExistingFileName* 和 *lpNewFileName* 必须位于同一驱动器上。  如果 *dwFlags* 指定 **MOVEFILE\_DELAY\_UNTIL\_REBOOT** 且 *lpNewFileName* 为 **NULL，则 MoveFileWithProgress** 会在系统重启时注册要删除的 *lpExistingFileName*。如果函数 无法访问注册表以存储有关删除操作的信息。如果 *lpExistingFileName* 引用目录，则系统会在重新启动时删除该目录 仅当目录为空时。  默认情况下，名称限制为 MAX\_PATH 个字符。要将此限制扩展到 32,767 个宽字符，请在路径前面加上 “\\？\”。有关更多信息，请参见[命名文件、路径和命名空间](https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，您可以选择在不附加“\\？\”的情况下删除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。  [in, optional] lpProgressRoutine  指向 [CopyProgressRoutine](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nc-winbase-lpprogress_routine) 回调的指针 每次移动文件的其他部分时都会调用该函数。回调函数可以是 如果您提供显示操作进度的用户界面，则很有用。此参数可以为 **NULL。**  [in, optional] lpData  要传递给 [CopyProgressRoutine](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nc-winbase-lpprogress_routine) 回调函数的参数。这 parameter 可以为 **NULL。**  [in] dwFlags  移动选项。此参数可以是以下一个或多个值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MOVEFILE\_COPY\_ALLOWED**  2 （0x2） | 如果要将文件移动到其他卷，该函数将使用 [CopyFile](https://learn.microsoft.com/en-us/windows/desktop/api/winbase/nf-winbase-copyfile) 和 [DeleteFile](https://learn.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-deletefilea) 函数模拟移动。  如果文件已成功复制到其他卷，但无法删除原始文件， 该函数成功，使源文件保持不变。  此值不能与 **MOVEFILE\_DELAY\_UNTIL\_REBOOT** 一起使用。 | | **MOVEFILE\_CREATE\_HARDLINK**  16 （0x10） | 保留供将来使用。 | | **MOVEFILE\_DELAY\_UNTIL\_REBOOT**  4 （0x4） | 在重新启动操作系统之前，系统不会移动文件。系统移动文件 在执行 AUTOCHK 之后，但在创建任何分页文件之前。因此，此参数 启用该函数以从以前的启动中删除分页文件。  仅当进程位于属于管理员的用户的上下文中时，才能使用此值 group 或 LocalSystem 帐户。  此值不能与 **MOVEFILE\_COPY\_ALLOWED** 一起使用。 | | **MOVEFILE\_FAIL\_IF\_NOT\_TRACKABLE**  32 （0x20） | 如果源文件是链接源，但在移动后无法跟踪该文件，则函数将失败。 如果目标是使用 FAT 文件系统格式化的卷，则可能会出现这种情况。 | | **MOVEFILE\_REPLACE\_EXISTING**  1 （0x1） | 如果存在名为 *lpNewFileName* 的文件，该函数会将其内容替换为 *lpExistingFileName* 文件的内容。  如果 *lpNewFileName* 或 *lpExistingFileName* 命名目录，则不能使用此值。 | | **MOVEFILE\_WRITE\_THROUGH**  8 （0x8） | 在文件实际移动到磁盘上之前，该函数不会返回。  设置此值可保证作为复制和删除操作执行的移动将刷新到磁盘 在函数返回之前。刷新发生在复制操作结束时。  如果设置了 **MOVEFILE\_DELAY\_UNTIL\_REBOOT**，则此值不起作用。 |  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)。  跨卷移动文件时，如果 *lpProgressRoutine* 由于用户取消操作而返回 **PROGRESS\_CANCEL**， **MoveFileWithProgress** 将返回零， [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_REQUEST\_ABORTED**。现有文件保持不变。  跨卷移动文件时，如果 *lpProgressRoutine* 由于用户停止操作而返回 **PROGRESS\_STOP**， **则 MoveFileWithProgress** 将返回零， [GetLastError](https://learn.microsoft.com/en-us/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_REQUEST\_ABORTED**。现有文件保持不变。 言论 **MoveFileWithProgress** 函数将其操作与链接协调 tracking 服务，因此可以在链接源移动时对其进行跟踪。  要删除或重命名文件，您必须具有该文件的 delete 权限或 delete child 权限 父目录。如果您设置的目录具有除 delete 和 delete child 之外的所有访问权限，并且 ACL 为 new 文件被继承，那么您应该能够创建文件但无法删除它。但是，您可以 然后创建一个文件，您将获得在返回给您的句柄上请求的所有访问权限 创建文件。如果您在创建文件时请求了删除权限，则可以删除或重命名 具有该句柄的文件，但不能使用任何其他句柄的文件。 |

# 演练

### 我们要实现的功能是：

|  |  |
| --- | --- |
| fileutil –d a.txt | 删除a.txt文件 |
| fileutil –c a.txt b.txt | 把a.txt复制一份，改名b.txt |
| fileutil –m d:\text.txt e:\text.txt | 移动文件，其实也可以重命名文件 |

## 1.新建一个c++常规空项目，取名：fileutil,然后新建一个cpp文件叫做del-copy-move.cpp,这一次，我们想做一个命令行工具可以复制，移动和删除文件，所有我们需要给main函数传递参数，然后我们先写一些代码来测试一下是否可以正确获取命令行参数。

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 2.把测试代码注释了，然后我们来实现功能，需要做对-d ,-c和-m参数的判断和参数个数的判断

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  int main(int argc,PCHAR argv[])  {  /\*printf("arg cout: %d\n",argc);  printf("first arg : %s\n",argv[0]);  printf("second arg : %s\n",argv[1]);  printf("third arg : %s\n",argv[2]);\*/  //如果第二个参数是-d，我们就做删除  if(0==lstrcmp("-d",argv[1]) && argc==3)//判断是是否是-d,而且需要有另外一个参数就是需要删除的文件名  {  DeleteFile(argv[2]);  //MessageBox(NULL,"Delete Succeeded","result",0);  printf("Delete Succeeded");  } else if(0==lstrcmp("-c",argv[1]) && argc==4){ //复制需要在命令后面跟个参数  CopyFile(argv[2],argv[3],FALSE);  } else if(0==lstrcmp("-m",argv[1]) && argc==4){//移动也需要在命令后面跟个参数  MoveFile(argv[2],argv[3]);  } else {  printf("usage: fileutil -d ||-c ||-m file1 [file2]\n");  }    system("pause");  return 0;  } |

## 3.扩展，为我们的程序添加执行api成功和失败的提示信息

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  int main(int argc,PCHAR argv[])  {  /\*printf("arg cout: %d\n",argc);  printf("first arg : %s\n",argv[0]);  printf("second arg : %s\n",argv[1]);  printf("third arg : %s\n",argv[2]);\*/  //如果第二个参数是-d，我们就做删除  if(0==lstrcmp("-d",argv[1]) && argc==3)//判断是是否是-d,而且需要有另外一个参数就是需要删除的文件名  {  if(!DeleteFile(argv[2])){  printf("delete error: %d\n",GetLastError());  }else{  //MessageBox(NULL,"Delete Succeeded","result",0);  printf("Delete Succeeded\n");  }  } else if(0==lstrcmp("-c",argv[1]) && argc==4){ //复制需要在命令后面跟个参数  if(!CopyFile(argv[2],argv[3],FALSE))  {  printf("Copy error: %d\n",GetLastError());  }else{  printf("Copy Succeeded\n");  }  } else if(0==lstrcmp("-m",argv[1]) && argc==4){//移动也需要在命令后面跟个参数  if(!MoveFile(argv[2],argv[3]))  {  printf("Move error: %d\n",GetLastError());  } else{  printf("Move Succeeded\n");  }  } else {  printf("usage: fileutil -d ||-c ||-m file1 [file2]\n");  }    system("pause");  return 0;  } |

## 4，扩展2.在复制文件的时候如果目标文件存在询问用户是否覆盖，如果用户输入’y’,就覆盖，此时需要把CopyFile函数的第三个参数改为TRUE，我们修改一下代码。del-copy-move.cpp完整代码如下

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  int main(int argc,PCHAR argv[])  {  /\*printf("arg cout: %d\n",argc);  printf("first arg : %s\n",argv[0]);  printf("second arg : %s\n",argv[1]);  printf("third arg : %s\n",argv[2]);\*/  //如果第二个参数是-d，我们就做删除  if(0==lstrcmp("-d",argv[1]) && argc==3)//判断是是否是-d,而且需要有另外一个参数就是需要删除的文件名  {  if(!DeleteFile(argv[2])){  printf("delete error: %d\n",GetLastError());  }else{  //MessageBox(NULL,"Delete Succeeded","result",0);  printf("Delete Succeeded\n");  }  } else if(0==lstrcmp("-c",argv[1]) && argc==4){ //复制需要在命令后面跟个参数  if(!CopyFile(argv[2],argv[3],TRUE))  {  if(GetLastError() == 0x50)  {  printf("target file %s already exists,over write?(y/n): ",argv[3]);  if('y' == getchar())  {  if(! CopyFile(argv[2],argv[3],FALSE))//覆盖目标文件  {  printf("Copy Failed\n");  } else {  printf("Copy Succeeded\n");  return 0;  }  }  }  }else {  printf("Copy Succeeded\n");  }  } else if(0==lstrcmp("-m",argv[1]) && argc==4){//移动也需要在命令后面跟个参数  if(!MoveFile(argv[2],argv[3]))  {  printf("Move error: %d\n",GetLastError());  } else{  printf("Move Succeeded\n");  }  } else {  printf("usage: fileutil -d ||-c ||-m file1 [file2]\n");  }    system("pause");  return 0;  } |

### 运行效果

|  |
| --- |
|  |