# 学习大纲

|  |
| --- |
|  |

# 我们通过一个案例来学习这些API

## 1.打开vs2010新建一个项目，取名lesson5-sysdir，是空项目，然后新建一个cpp文件

|  |
| --- |
|  |

## 2.在这个cpp文件里面我们主要学习获取系统目录，然后写到一个文件中。需要用到4个API，全部代码在上面了。

### GetSystemDirectory函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索系统目录的路径。 系统目录包含系统文件，例如动态链接库和驱动程序。  提供此函数主要用于兼容性。 应用程序应将代码存储在“程序文件”文件夹中，并将持久数据存储在用户配置文件的“应用程序数据”文件夹中。 有关详细信息，请参阅 [ShGetFolderPath](https://learn.microsoft.com/zh-cn/windows/desktop/api/shlobj_core/nf-shlobj_core-shgetfolderpatha)。 语法 C++复制  UINT GetSystemDirectoryA(  [out] LPSTR lpBuffer,  [in] UINT uSize  ); 参数 [out] lpBuffer  指向要接收路径的缓冲区的指针。 除非系统目录是根目录，否则此路径不会以反斜杠结尾。 例如，如果系统目录在驱动器 C 上名为 Windows\System32，则此函数检索的系统目录路径为 C：\Windows\System32。  [in] uSize  缓冲区的最大大小（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 TCHAR 为单位**），不包括终止 null 字符。 如果长度大于缓冲区的大小，则返回值是保存路径所需的缓冲区大小，包括终止 null 字符。  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 应用程序不应在系统目录中创建文件。 如果用户运行的是共享版本的操作系统，则应用程序对系统目录没有写入访问权限。 示例 有关示例，请参阅 [获取系统信息](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/getting-system-information)。  **备注**  sysinfoapi.h 标头将 GetSystemDirectory 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** | sysinfoapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetCurrentDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getcurrentdirectory)  [GetWindowsDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getwindowsdirectorya)  [SetCurrentDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setcurrentdirectory)  [系统信息函数](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/system-information-functions) |

### CreateFile函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 创建或打开文件或 I/O 设备。 最常用的 I/O 设备如下所示：文件、文件流、目录、物理磁盘、卷、控制台缓冲区、磁带驱动器、通信资源、mailslot 和管道。 该函数返回一个句柄，该句柄可用于访问各种类型的 I/O 的文件或设备，具体取决于文件或设备以及指定的标志和属性。  若要将此操作作为事务处理操作执行，这会导致可用于事务处理 I/O 的句柄，请使用 [CreateFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfiletransacteda) 函数。 语法 C++复制  HANDLE CreateFileA(  [in] LPCSTR lpFileName,  [in] DWORD dwDesiredAccess,  [in] DWORD dwShareMode,  [in, optional] LPSECURITY\_ATTRIBUTES lpSecurityAttributes,  [in] DWORD dwCreationDisposition,  [in] DWORD dwFlagsAndAttributes,  [in, optional] HANDLE hTemplateFile  ); 参数 [in] lpFileName  要创建或打开的文件或设备的名称。 可以在此名称中使用正斜杠 （/） 或反斜杠 （\）。  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 宽字符，请将“\\？\”前面追加到路径。 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，你可以选择加入以删除MAX\_PATH限制，而无需追加“\\？\”。 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。  有关特殊设备名称的信息，请参阅 [定义 MS-DOS 设备名称](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/defining-an-ms-dos-device-name)。  若要创建文件流，请指定文件的名称、冒号，然后指定流的名称。 有关详细信息，请参阅 [文件流](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-streams)。  [in] dwDesiredAccess  请求对文件或设备的访问权限，可以汇总为读取、写入或 0，以指示两者均未提供）。  最常用的值是 **GENERIC\_READ**、**GENERIC\_WRITE**或两者（GENERIC\_READ | GENERIC\_WRITE）。 有关详细信息，请参阅 [通用访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/generic-access-rights)、[文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)、[文件访问权限常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-access-rights-constants)和 [ACCESS\_MASK](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/access-mask)。  如果此参数为零，则应用程序可以在不访问该文件或设备的情况下查询某些元数据（如文件、目录或设备属性），即使拒绝 **GENERIC\_READ** 访问也是如此。  不能请求与共享模式冲突的访问模式，该模式由 *dwShareMode* 参数指定的打开请求中已具有打开句柄。  有关详细信息，请参阅本主题的“备注”部分，[创建和打开文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-opening-files)。  [in] dwShareMode  请求的文件或设备的共享模式，可以读取、写入、删除、所有这些或无（请参阅下表）。 对属性或扩展属性的访问请求不受此标志的影响。  如果此参数为零且 **CreateFile** 成功，则文件或设备无法共享，并且无法在文件或设备的句柄关闭之前再次打开。 有关详细信息，请参阅“备注”部分。  无法请求与具有打开句柄的现有请求中指定的访问模式冲突的共享模式。 **CreateFile** 将失败，[GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_SHARING\_VIOLATION**。  若要使进程能够在另一个进程打开文件或设备时共享文件或设备，请使用以下一个或多个值的兼容组合。 有关此参数与 *dwDesiredAccess* 参数的有效组合的详细信息，请参阅 [创建和打开文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-opening-files)。  **注意** 每个打开句柄的共享选项一直有效，直到该句柄关闭，而不考虑进程上下文。    展开表   |  |  | | --- | --- | | **价值** | **意义** | | **0**  0x00000000 | 如果其他进程请求删除、读取或写入访问权限，则阻止其他进程打开文件或设备。 | | **FILE\_SHARE\_DELETE**  0x00000004 | 启用对文件或设备上的后续打开操作以请求删除访问权限。  否则，如果进程请求删除访问权限，则无法打开文件或设备。  如果未指定此标志，但文件或设备已打开以删除访问权限，则函数将失败。  **注释** 删除访问权限允许删除和重命名操作。 | | **FILE\_SHARE\_READ**  0x00000001 | 允许对文件或设备执行后续打开操作以请求读取访问权限。  否则，如果进程请求读取访问权限，则其他进程无法打开文件或设备。  如果未指定此标志，但已打开文件或设备进行读取访问，则函数将失败。 | | **FILE\_SHARE\_WRITE**  0x00000002 | 允许对文件或设备执行后续打开操作以请求写入访问权限。  否则，如果进程请求写入访问权限，则其他进程无法打开文件或设备。  如果未指定此标志，但已打开文件或设备进行写入访问或具有写入访问权限的文件映射，则函数将失败。 |   [in, optional] lpSecurityAttributes  指向包含两个独立但相关数据成员的 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针：可选的安全描述符，以及一个布尔值，该值确定是否可由子进程继承返回的句柄。  此参数可以 **NULL**。  如果此参数 **NULL**，则应用程序可能创建的任何子进程都无法继承由 **CreateFile** 返回的句柄，并且与返回的句柄关联的文件或设备将获取默认的安全描述符。  结构的 **lpSecurityDescriptor** 成员指定文件或设备的 [SECURITY\_DESCRIPTOR](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-security_descriptor)。 如果此成员 **NULL**，则会为与返回的句柄关联的文件或设备分配一个默认的安全描述符。  **CreateFile** 打开现有文件或设备时忽略 **lpSecurityDescriptor** 成员，但继续使用 **bInheritHandle** 成员。  **bInheritHandle** 结构的成员指定是否可以继承返回的句柄。  有关详细信息，请参阅“备注”部分。  [in] dwCreationDisposition  对存在或不存在的文件或设备执行的操作。  对于文件以外的设备，此参数通常设置为 **OPEN\_EXISTING**。  有关详细信息，请参阅“备注”部分。  此参数必须是以下值之一，不能组合这些值：  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **CREATE\_ALWAYS**  2 | 始终创建新文件。  如果指定的文件存在且可写，则函数将截断文件、函数成功，最后错误代码设置为 **ERROR\_ALREADY\_EXISTS**（183）。  如果指定的文件不存在且路径有效，则会创建一个新文件，该函数会成功，最后一个错误代码设置为零。  有关详细信息，请参阅本主题的“备注”部分。 | | **CREATE\_NEW**  1 | 仅当该文件尚不存在时，才会创建一个新文件。  如果指定的文件存在，函数将失败，最后一个错误代码设置为 **ERROR\_FILE\_EXISTS**（80）。  如果指定的文件不存在，并且是可写位置的有效路径，则会创建一个新文件。 | | **OPEN\_ALWAYS**  4 | 始终打开文件。  如果指定文件存在，则函数成功，最后一个错误代码设置为 **ERROR\_ALREADY\_EXISTS**（183）。  如果指定的文件不存在并且是可写位置的有效路径，该函数将创建一个文件，最后一个错误代码设置为零。 | | **OPEN\_EXISTING**  3 | 仅当文件或设备存在时才打开该文件或设备。  如果指定的文件或设备不存在，该函数将失败，最后一个错误代码设置为 **ERROR\_FILE\_NOT\_FOUND**（2）。  有关设备的详细信息，请参阅“备注”部分。 | | **TRUNCATE\_EXISTING**  5 | 打开一个文件并截断它，使其大小为零字节，仅当它存在时。  如果指定的文件不存在，该函数将失败，最后一个错误代码设置为 **ERROR\_FILE\_NOT\_FOUND**（2）。  调用过程必须打开文件，并将 **GENERIC\_WRITE** 位设置为 *dwDesiredAccess* 参数的一部分。 |   [in] dwFlagsAndAttributes  文件或设备属性和标志，**FILE\_ATTRIBUTE\_NORMAL** 是文件最常见的默认值。  此参数可以包含可用文件属性的任意组合（**FILE\_ATTRIBUTE\_\***）。 所有其他文件属性都替代 **FILE\_ATTRIBUTE\_NORMAL**。  此参数还可以包含标志（**FILE\_FLAG\_\***）的组合，用于控制文件或设备缓存行为、访问模式和其他特殊用途标志。 这些值与任何 **FILE\_ATTRIBUTE\_\*** 值结合使用。  此参数还可以通过指定 **SECURITY\_SQOS\_PRESENT** 标志来包含安全服务质量（SQOS）信息。 下表中显示了与 SQOS 相关的其他标志信息，这些属性和标志表如下。  注释 当 CreateFile 打开现有文件时，它通常会将文件标志与现有文件的文件属性合并，并忽略作为 dwFlagsAndAttributes的 一部分提供的任何文件属性。 [**创建和打开文件**](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-opening-files)中详细介绍了特殊情况。    以下某些文件属性和标志可能仅适用于文件，不一定适用于所有 **CreateFile** 的其他类型的设备。 有关详细信息，请参阅本主题的“备注”部分，[创建和打开文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-opening-files)。  有关对文件属性的更高级访问，请参阅 [SetFileAttributes](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfileattributesa)。 有关所有文件属性及其值和说明的完整列表，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** | **意义** | | **FILE\_ATTRIBUTE\_ARCHIVE**  32 （0x20） | 该文件应存档。 应用程序使用此属性标记文件以供备份或删除。 | | **FILE\_ATTRIBUTE\_ENCRYPTED**  16384 （0x4000） | 文件或目录已加密。 对于文件，这意味着文件中的所有数据都已加密。 对于目录，这意味着加密是新创建的文件和子目录的默认值。 有关详细信息，请参阅 [文件加密](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-encryption)。  如果还指定了 **FILE\_ATTRIBUTE\_SYSTEM**，则此标志无效。  家庭版、家庭高级版、入门版或 ARM 版 Windows 不支持此标志。 | | **FILE\_ATTRIBUTE\_HIDDEN**  2 （0x2） | 文件已隐藏。 不要将其包含在普通目录列表中。 | | **FILE\_ATTRIBUTE\_NORMAL**  128 （0x80） | 该文件没有设置其他属性。 仅当单独使用时，此属性才有效。 | | **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在物理上移动到脱机存储。 此属性由远程存储（分层存储管理软件）使用。 应用程序不应任意更改此属性。 | | **FILE\_ATTRIBUTE\_READONLY**  1 （0x1） | 该文件是只读的。 应用程序可以读取文件，但无法写入或删除该文件。 | | **FILE\_ATTRIBUTE\_SYSTEM**  4 （0x4） | 该文件是操作系统的一部分或独占使用。 | | **FILE\_ATTRIBUTE\_TEMPORARY**  256 （0x100） | 该文件用于临时存储。  有关详细信息，请参阅本主题的 [缓存行为](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea#caching_behavior) 部分。 |     展开表   |  |  | | --- | --- | | **旗** | **意义** | | **FILE\_FLAG\_BACKUP\_SEMANTICS**  0x02000000 | 正在为备份或还原操作打开或创建该文件。 当进程具有 **SE\_BACKUP\_NAME** 和 **SE\_RESTORE\_NAME** 特权时，系统可确保调用进程替代文件安全检查。 有关详细信息，请参阅 [更改令牌](https://learn.microsoft.com/zh-cn/windows/desktop/SecBP/changing-privileges-in-a-token)中的特权。  必须设置此标志才能获取目录的句柄。 目录句柄可以传递给某些函数，而不是文件句柄。 有关详细信息，请参阅“备注”部分。 | | **FILE\_FLAG\_DELETE\_ON\_CLOSE**  0x04000000 | 文件将在关闭所有句柄后立即删除，其中包括指定的句柄和任何其他打开或重复的句柄。  如果文件存在现有的打开句柄，则调用将失败，除非它们都以 **FILE\_SHARE\_DELETE** 共享模式打开。  文件的后续打开请求失败，除非指定了 **FILE\_SHARE\_DELETE** 共享模式。 | | **FILE\_FLAG\_NO\_BUFFERING**  0x20000000 | 文件或设备正在打开，没有数据读取和写入的系统缓存。 此标志不会影响硬盘缓存或内存映射文件。  使用 **FILE\_FLAG\_NO\_BUFFERING** 标志成功处理使用 **CreateFile** 打开的文件有严格的要求，有关详细信息，请参阅 [文件缓冲](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-buffering)。 | | **FILE\_FLAG\_OPEN\_NO\_RECALL**  0x00100000 | 请求文件数据，但它应继续位于远程存储中。 不应将其传输回本地存储。 此标志供远程存储系统使用。 | | **FILE\_FLAG\_OPEN\_REPARSE\_POINT**  0x00200000 | 正常 [重新分析点](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/reparse-points) 处理不会发生;**CreateFile** 将尝试打开重新分析点。 打开文件时，将返回文件句柄，无论控制重新分析点的筛选器是否正常运行。  此标志不能与 **CREATE\_ALWAYS** 标志一起使用。  如果文件不是重新分析点，则忽略此标志。  有关详细信息，请参阅“备注”部分。 | | **FILE\_FLAG\_OVERLAPPED**  0x40000000 | 文件或设备正在为异步 I/O 打开或创建。  在此句柄上完成后续 I/O 操作时，[OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构中指定的事件将设置为信号状态。  如果指定了此标志，则文件可用于同时读取和写入操作。  如果未指定此标志，则会序列化 I/O 操作，即使对读取和写入函数的调用指定了 [重叠](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构也是如此。  有关使用此标志创建的文件句柄时的注意事项，请参阅本主题的 [同步和异步 I/O 句柄](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea#synchronous_and_asynchronous_i_o_handles) 部分。 | | **FILE\_FLAG\_POSIX\_SEMANTICS**  0x01000000 | 访问将按照 POSIX 规则进行。 这包括允许具有名称的多个文件，仅在支持该命名的文件系统时有所不同。 使用此选项时请小心，因为使用此标志创建的文件可能无法由为 MS-DOS 或 16 位 Windows 编写的应用程序访问。 | | **FILE\_FLAG\_RANDOM\_ACCESS**  0x10000000 | 访问旨在随机访问。 系统可以将其用作优化文件缓存的提示。  如果文件系统不支持缓存的 I/O 并 **FILE\_FLAG\_NO\_BUFFERING**，则此标志无效。  有关详细信息，请参阅本主题的 [缓存行为](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea#caching_behavior) 部分。 | | **FILE\_FLAG\_SESSION\_AWARE**  0x00800000 | 正在打开具有会话感知的文件或设备。 如果未指定此标志，则会话中运行的进程无法打开每个会话设备（例如使用 RemoteFX USB 重定向的设备）。 此标志对会话 0 中不具有调用方的影响。 此标志仅在 Windows 的服务器版本上受支持。  **Windows Server 2008 R2 和 Windows Server 2008：**Windows Server 2012 之前不支持此标志。 | | **FILE\_FLAG\_SEQUENTIAL\_SCAN**  0x08000000 | Access 旨在从头到尾的顺序。 系统可以将其用作优化文件缓存的提示。  如果使用读后（即反向扫描），则不应使用此标志。  如果文件系统不支持缓存的 I/O 并 **FILE\_FLAG\_NO\_BUFFERING**，则此标志无效。  有关详细信息，请参阅本主题的 [缓存行为](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea#caching_behavior) 部分。 | | **FILE\_FLAG\_WRITE\_THROUGH**  0x80000000 | 写入操作不会经历任何中间缓存，它们将直接转到磁盘。  有关详细信息，请参阅本主题的 [缓存行为](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-createfilea#caching_behavior) 部分。 |     *dwFlagsAndAttributes* 参数还可以指定 SQOS 信息。 有关详细信息，请参阅 [模拟级别](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/impersonation-levels)。 当调用应用程序将 **SECURITY\_SQOS\_PRESENT** 标志指定为 *dwFlagsAndAttributes*的一部分时，它还可以包含以下一个或多个值。  展开表   |  |  | | --- | --- | | **安全标志** | **意义** | | **SECURITY\_ANONYMOUS** | 在匿名模拟级别模拟客户端。 | | **SECURITY\_CONTEXT\_TRACKING** | 安全跟踪模式是动态的。 如果未指定此标志，则安全跟踪模式是静态的。 | | **SECURITY\_DELEGATION** | 在委派模拟级别模拟客户端。 | | **SECURITY\_EFFECTIVE\_ONLY** | 只有客户端安全上下文的启用方面可供服务器使用。 如果未指定此标志，则客户端安全上下文的所有方面都可用。  这允许客户端限制服务器在模拟客户端时可以使用的组和特权。 | | **SECURITY\_IDENTIFICATION** | 在标识模拟级别模拟客户端。 | | **SECURITY\_IMPERSONATION** | 在模拟级别模拟客户端。 如果未指定其他标志以及 **SECURITY\_SQOS\_PRESENT** 标志，则这是默认行为。 |   [in, optional] hTemplateFile  具有 **GENERIC\_READ** 访问权限的模板文件的有效句柄。 模板文件为正在创建的文件提供文件属性和扩展属性。  此参数可以 **NULL**。  打开现有文件时，**CreateFile** 忽略此参数。  打开新的加密文件时，该文件从其父目录继承自由访问控制列表。 有关详细信息，请参阅 [文件加密](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-encryption)。 返回值 如果函数成功，则返回值是指定文件、设备、命名管道或邮件槽的打开句柄。  如果函数失败，则返回值 **INVALID\_HANDLE\_VALUE**。 若要获取扩展的错误信息，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。 言论 **CreateFile** 最初是为文件交互而开发的，但后来进行了扩展和增强，以包括 Windows 开发人员可用的大多数其他类型的 I/O 设备和机制。 本部分尝试介绍开发人员在不同上下文和不同 I/O 类型中使用 **CreateFile** 时可能会遇到的各种问题。 仅当专门引用存储在文件系统上实际文件中的数据时，文本才会尝试使用单词 文件。 但是，*文件的一些用法* 可能更普遍地引用支持类似文件的机制的 I/O 对象。 由于前面提到的历史原因，此术语 *文件* 在常量名称和参数名称中特别普遍。  使用 **CreateFile**返回的对象句柄完成应用程序后，请使用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数关闭句柄。 这不仅释放了系统资源，还可以对共享文件或设备以及将数据提交到磁盘等内容产生更广泛的影响。 本主题中会相应地说明具体内容。  **Windows Server 2003 和 Windows XP：如果尝试打开文件或目录以在远程计算机上删除，**尝试打开文件或目录以在远程计算机上删除，则当 *dwDesiredAccess* 参数的值是 **DELETE** 访问标志（0x00010000）**OR**“与任何其他访问标志一起”时，会发生共享冲突， 并且远程文件或目录尚未使用 **FILE\_SHARE\_DELETE**打开。 为了避免在此方案中发生共享冲突，请使用 **DELETE 打开远程文件或目录** 访问权限，或者调用 [DeleteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-deletefilea)，而无需先打开文件或目录进行删除。  某些文件系统（如 NTFS 文件系统）支持单个文件和目录的压缩或加密。 在具有具有此支持的装载文件系统的卷上，新文件继承其目录的压缩和加密属性。  不能使用 **CreateFile** 来控制文件或目录上的压缩、解压缩或解密。 有关详细信息，请参阅 [创建和打开文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-opening-files)、[文件压缩和解压缩](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-compression-and-decompression)，以及 [文件加密](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-encryption)。  **Windows Server 2003 和 Windows XP：**出于向后兼容性目的，**CreateFile** 在 *lpSecurityAttributes*中指定安全描述符时，不会应用继承规则。 为了支持继承，以后查询此文件的安全描述符的函数可能会启发式确定并报告继承是否有效。 有关详细信息，请参阅 [可继承 ACE 的自动传播](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/automatic-propagation-of-inheritable-aces)。  如前所述，如果 *lpSecurityAttributes* 参数 **NULL**，则 **CreateFile** 返回的句柄不能由应用程序可能创建的任何子进程继承。 有关此参数的以下信息也适用：   * 如果 **bInheritHandle** 成员变量未 **FALSE**（任何非零值），则可以继承句柄。 因此，如果不希望句柄可继承，则必须将此结构成员正确初始化为 **FALSE**。 * 文件或目录的默认安全描述符中的访问控制列表（ACL）继承自其父目录。 * 目标文件系统必须支持 **lpSecurityDescriptor** 成员对文件和目录的安全性，才能对其产生影响，这可以通过使用 [GetVolumeInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getvolumeinformationa)来确定。   在 Windows 8 和 Windows Server 2012 中，以下技术支持此函数。  展开表   |  |  | | --- | --- | | **科技** | **支持** | | 服务器消息块 （SMB） 3.0 协议 | 是的 | | SMB 3.0 透明故障转移 （TFO） | 请参阅备注 | | 具有横向扩展文件共享的 SMB 3.0 （SO） | 请参阅备注 | | 群集共享卷文件系统 （CsvFS） | 是的 | | 可复原文件系统 （ReFS） | 是的 |     请注意，如果对已打开的备用数据流的文件执行 **，则具有取代处置的 CreateFile** 将失败。 符号链接行为 如果对此函数的调用创建文件，则行为没有变化。 此外，请考虑有关 **FILE\_FLAG\_OPEN\_REPARSE\_POINT**的以下信息：   * 如果指定了 **FILE\_FLAG\_OPEN\_REPARSE\_POINT**：   + 如果打开现有文件并且它是符号链接，则返回的句柄是符号链接的句柄。   + 如果指定了 **TRUNCATE\_EXISTING** 或 **FILE\_FLAG\_DELETE\_ON\_CLOSE**，受影响的文件是符号链接。 * 如果未指定 **FILE\_FLAG\_OPEN\_REPARSE\_POINT**：   + 如果打开现有文件并且它是符号链接，则返回的句柄是目标的句柄。   + 如果指定了 **CREATE\_ALWAYS**、**TRUNCATE\_EXISTING**或 **FILE\_FLAG\_DELETE\_ON\_CLOSE**，受影响的文件就是目标。  缓存行为 **CreateFile** 使用 *dwFlagsAndAttributes* 参数的几个可能值来控制或影响系统缓存与句柄关联的数据的方式。 它们是：   * **FILE\_FLAG\_NO\_BUFFERING** * **FILE\_FLAG\_RANDOM\_ACCESS** * **FILE\_FLAG\_SEQUENTIAL\_SCAN** * **FILE\_FLAG\_WRITE\_THROUGH** * **FILE\_ATTRIBUTE\_TEMPORARY**   如果未指定这些标志，系统将使用默认的常规用途缓存方案。 否则，系统缓存的行为为每个标志指定。  不应合并其中一些标志。 例如，将 **FILE\_FLAG\_RANDOM\_ACCESS** 与 **FILE\_FLAG\_SEQUENTIAL\_SCAN** 相结合是自败。  指定 **FILE\_FLAG\_SEQUENTIAL\_SCAN** 标志可以提高使用顺序访问读取大型文件的应用程序的性能。 对于主要按顺序读取大型文件的应用程序，性能提升可能更为明显，但偶尔会跳过小范围的字节。 如果应用程序移动文件指针进行随机访问，则很可能不会出现最佳缓存性能。 但是，仍可以保证正确的操作。  **FILE\_FLAG\_WRITE\_THROUGH** 和 **FILE\_FLAG\_NO\_BUFFERING** 的标志是独立的，可以组合使用。  如果使用 **FILE\_FLAG\_WRITE\_THROUGH** 但未指定 **FILE\_FLAG\_NO\_BUFFERING**，以便系统缓存生效，则数据将写入系统缓存，但不会延迟地刷新到磁盘。  如果同时指定了 **FILE\_FLAG\_WRITE\_THROUGH** 和 **FILE\_FLAG\_NO\_BUFFERING**，以便系统缓存无效，则数据会立即刷新到磁盘，而无需通过 Windows 系统缓存。 操作系统还向永久性媒体请求硬盘本地硬件缓存的写入。  **注意** 并非所有硬盘硬件都支持此写通功能。    正确使用 **FILE\_FLAG\_NO\_BUFFERING** 标志需要特殊的应用程序注意事项。 有关详细信息，请参阅 [文件缓冲](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-buffering)。  通过 **FILE\_FLAG\_WRITE\_THROUGH** 的写通请求还会导致 NTFS 刷新任何元数据更改，例如时间戳更新或重命名操作，这会导致处理请求。 因此，**FILE\_FLAG\_WRITE\_THROUGH** 标志通常与 **FILE\_FLAG\_NO\_BUFFERING** 标志一起使用，作为每次写入后调用 [FlushFileBuffers](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-flushfilebuffers) 函数的替代项，这可能会导致不必要的性能损失。 将这些标志一起使用可避免这些处罚。 有关文件和元数据缓存的一般信息，请参阅 [文件缓存](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-caching)。  当 **FILE\_FLAG\_NO\_BUFFERING** 与 **FILE\_FLAG\_OVERLAPPED**结合使用时，标志可提供最大的异步性能，因为 I/O 不依赖于内存管理器的同步操作。 但是，某些 I/O 操作需要更多时间，因为缓存中未保存数据。 此外，文件元数据仍可能缓存（例如，创建空文件时）。 若要确保将元数据刷新到磁盘，请使用 [FlushFileBuffers](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-flushfilebuffers) 函数。  指定 **FILE\_ATTRIBUTE\_TEMPORARY** 属性会导致文件系统避免在有足够的缓存内存可用时将数据写回到大容量存储，因为应用程序在关闭句柄后删除临时文件。 在这种情况下，系统可以完全避免写入数据。 尽管它不直接控制数据缓存的方式与前面提到的标志相同，但 **FILE\_ATTRIBUTE\_TEMPORARY** 属性确实会告知系统在系统缓存中尽可能多地保存而不编写数据缓存，因此可能对某些应用程序感到担忧。 文件 如果重命名或删除文件，然后在不久后还原该文件，系统会在缓存中搜索要还原的文件信息。 缓存信息包括其短/长名称对和创建时间。  如果由于对 [DeleteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-deletefilea)的上一次调用而挂起删除的文件调用 **CreateFile**，该函数将失败。 操作系统会延迟文件删除，直到文件的所有句柄都关闭。 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_ACCESS\_DENIED**。  *dwDesiredAccess* 参数可以为零，允许应用程序在没有使用足够安全设置的情况下访问文件属性来查询文件属性。 这可用于测试文件是否存在，而无需打开该文件进行读取和/或写入访问，或获取有关文件或目录的其他统计信息。 请参阅 [获取和设置文件信息](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/obtaining-and-setting-file-information) 和 [GetFileInformationByHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfileinformationbyhandle)。  如果指定了 **CREATE\_ALWAYS** 和 **FILE\_ATTRIBUTE\_NORMAL**，**CreateFile** 将失败，并将文件存在且具有 **FILE\_ATTRIBUTE\_HIDDEN** 或 **FILE\_ATTRIBUTE\_SYSTEM** 属性时将最后一个错误设置为 **ERROR\_ACCESS\_DENIED**。 若要避免此错误，请指定与现有文件相同的属性。  当应用程序跨网络创建文件时，最好将 GENERIC\_READ | GENERIC\_WRITE 用于 *dwDesiredAccess*，而不是单独使用 **GENERIC\_WRITE**。 生成的代码速度更快，因为重定向程序可以使用缓存管理器，并发送更少的 SMB 和更多数据。 这种组合还避免了跨网络写入文件偶尔会返回 **ERROR\_ACCESS\_DENIED**的问题。  有关详细信息，请参阅 [创建和打开文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-opening-files)。 同步和异步 I/O 句柄 **CreateFile** 提供用于创建同步或异步的文件或设备句柄。 同步句柄的行为使使用该句柄的 I/O 函数调用被阻止，直到它们完成，而异步文件句柄使系统能够立即从 I/O 函数调用返回，无论它们是否已完成 I/O 操作。 如前所述，此同步与异步行为是通过在 *dwFlagsAndAttributes* 参数中指定 **FILE\_FLAG\_OVERLAPPED** 来确定的。 使用异步 I/O 时，存在多种复杂性和潜在缺陷;有关详细信息，请参阅 [同步和异步 I/O](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/synchronous-and-asynchronous-i-o)。 文件流 在 NTFS 文件系统上，可以使用 **CreateFile** 在文件中创建单独的流。 有关详细信息，请参阅 [文件流](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-streams)。 目录 应用程序无法使用 **CreateFile**创建目录，因此，对于此用例，只有 **OPEN\_EXISTING** 值对 *dwCreationDisposition* 有效。 若要创建目录，应用程序必须调用 [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)。  若要使用 **CreateFile**打开目录，请将 **FILE\_FLAG\_BACKUP\_SEMANTICS** 标志指定为 *dwFlagsAndAttributes*的一部分。 在没有 **SE\_BACKUP\_NAME** 和 **SE\_RESTORE\_NAME** 特权的情况下使用此标志时，仍适用适当的安全检查。  使用 **CreateFile** 在对 FAT 或 FAT32 文件系统卷进行碎片整理期间打开目录时，请不要指定 **MAXIMUM\_ALLOWED** 访问权限。 如果这样做，则拒绝对目录的访问。 请改为指定 **GENERIC\_READ** 访问权限。  有关详细信息，请参阅 [关于目录管理](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/about-directory-management)。 物理磁盘和卷 对磁盘或卷的直接访问受到限制。  **Windows Server 2003 和 Windows XP：**直接访问磁盘或卷的方式不受限制。  可以使用 **CreateFile** 函数打开物理磁盘驱动器或卷，该驱动器返回可与 [DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 函数一起使用的直接访问存储设备 （DASD） 句柄。 这样，便可以直接访问磁盘或卷，例如分区表等磁盘元数据。 但是，这种类型的访问还会向潜在的数据丢失公开磁盘驱动器或卷，因为使用此机制对磁盘进行不正确的写入可能会使其内容无法访问操作系统。 为了确保数据完整性，请务必熟悉 **DeviceIoControl**，以及其他 API 与直接访问句柄（而不是文件系统句柄）的行为方式不同。  必须满足以下要求才能成功进行此类调用：   * 调用方必须具有管理权限。 有关详细信息，请参阅 [使用特殊特权](https://learn.microsoft.com/zh-cn/windows/desktop/SecBP/running-with-special-privileges)运行。 * *dwCreationDisposition* 参数必须具有 **OPEN\_EXISTING** 标志。 * 打开卷或软盘时，*dwShareMode* 参数必须具有 **FILE\_SHARE\_WRITE** 标志。   注释 dwDesiredAccess 参数 可以为零，允许应用程序在不访问设备的情况下查询设备属性。 这适用于应用程序来确定软盘驱动器的大小及其支持的格式，而无需驱动器中的软盘，例如。 它还可用于读取统计信息，而无需更高级别的数据读取/写入权限。    *x*：打开物理驱动器时，*lpFileName* 字符串应采用以下格式：“\\.\PhysicalDrive*X*”。 硬盘编号从零开始。 下表显示了物理驱动器字符串的一些示例。  展开表   |  |  | | --- | --- | | **字符串** | **意义** | | “\\.\PhysicalDrive0” | 打开第一个物理驱动器。 | | “\\.\PhysicalDrive2” | 打开第三个物理驱动器。 |     若要获取卷的物理驱动器标识符，请打开卷的句柄，并使用 [IOCTL\_VOLUME\_GET\_VOLUME\_DISK\_EXTENTS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-ioctl_volume_get_volume_disk_extents)调用 [DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 函数。 此控制代码返回每个卷的一个或多个盘区的磁盘编号和偏移量;卷可以跨越多个物理磁盘。  有关打开物理驱动器的示例，请参阅 [调用 DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/calling-deviceiocontrol)。  打开卷或可移动媒体驱动器（例如软盘驱动器或闪存内存拇指驱动器）时，*lpFileName* 字符串应采用以下格式：“\\.\*X*：”。 不要使用尾随反斜杠（\），指示驱动器的根目录。 下表显示了驱动器字符串的一些示例。  展开表   |  |  | | --- | --- | | **字符串** | **意义** | | “\\.\A：” | 打开软盘驱动器 A。 | | “\\.\C：” | 打开 C： 卷。 | | “\\.\C：\” | 打开 C： 卷的文件系统。 |     还可以通过引用卷名称来打开卷。 有关详细信息，请参阅 [命名卷](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/naming-a-volume)。  卷包含一个或多个装载的文件系统。 即使 **CreateFile**中未指定非缓存选项，也可以以非缓存方式打开卷句柄。 应假定所有Microsoft文件系统都以非缓存的形式打开卷句柄。 对文件的非缓存 I/O 的限制也适用于卷。  即使数据未缓存，文件系统也可能不需要缓冲区对齐。 但是，如果在打开卷时指定了非缓存选项，则会强制实施缓冲区对齐方式，而不考虑卷上的文件系统。 建议在所有文件系统上以非缓存方式打开卷句柄，并遵循非缓存 I/O 限制。  **注意** 若要读取或写入卷的最后几个扇区，必须调用 [**DeviceIoControl**](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 并指定 [**FSCTL\_ALLOW\_EXTENDED\_DASD\_IO**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_allow_extended_dasd_io)。 这表示文件系统驱动程序不会对分区读取或写入调用执行任何 I/O 边界检查。 相反，边界检查由设备驱动程序执行。   变更程序设备 [DeviceIoControl 的 **IOCTL\_CHANGER\_\*** 控制代码](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol) 接受变更器设备的句柄。 若要打开更改器设备，请使用以下形式的文件名：“\\.\Changer*x*”，其中 *x* 是一个数字，指示要打开的设备，从零开始。 若要在用 C 或C++编写的应用程序中打开 changer 设备零，请使用以下文件名：“\\.\Changer0”。 磁带驱动器 可以使用以下格式的文件名打开磁带驱动器：“\\.\TAPE*x*”，其中 *x* 是指示要打开的驱动器的数字，从磁带驱动器零开始。 若要在以 C 或 C++ 编写的应用程序中打开磁带驱动器零，请使用以下文件名：“\\.\TAPE0”。  有关详细信息，请参阅 [备份](https://learn.microsoft.com/zh-cn/windows/desktop/Backup/backup)。 通信资源 **CreateFile** 函数可以创建通信资源的句柄，例如串行端口 COM1。 对于通信资源，*dwCreationDisposition* 参数必须 **OPEN\_EXISTING**，*dwShareMode* 参数必须为零（独占访问），*hTemplateFile* 参数必须 **NULL**。 可以指定读取、写入或读/写访问权限，并且可以为重叠 I/O 打开句柄。  若要指定大于 9 的 COM 端口号，请使用以下语法：“\\.\COM10”。 此语法适用于允许指定 COM 端口号的所有端口号和硬件。  有关通信的详细信息，请参阅 [通信](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/communications-resources)。 控制台 **CreateFile** 函数可以创建控制台输入的句柄（CONIN$）。 如果进程由于继承或重复而具有打开的句柄，则它还可以为活动屏幕缓冲区（CONOUT$）创建句柄。 调用进程必须附加到继承的控制台或由 [AllocConsole](https://learn.microsoft.com/zh-cn/windows/console/allocconsole) 函数分配的控制台。 对于控制台句柄，请按如下所示设置 **CreateFile** 参数。  展开表   |  |  | | --- | --- | | **参数** | **价值** | | *lpFileName* | 使用 CONIN$ 值指定控制台输入。  使用 CONOUT$ 值指定控制台输出。  CONIN$ 获取控制台输入缓冲区的句柄，即使 [SetStdHandle](https://learn.microsoft.com/zh-cn/windows/console/setstdhandle) 函数也会重定向标准输入句柄。 若要获取标准输入句柄，请使用 [GetStdHandle](https://learn.microsoft.com/zh-cn/windows/console/getstdhandle) 函数。  CONOUT$ 获取活动屏幕缓冲区的句柄，即使 [SetStdHandle](https://learn.microsoft.com/zh-cn/windows/console/setstdhandle) 重定向标准输出句柄也是如此。 若要获取标准输出句柄，请使用 [GetStdHandle](https://learn.microsoft.com/zh-cn/windows/console/getstdhandle)。 | | *dwDesiredAccess* | 首选 GENERIC\_READ | GENERIC\_WRITE，但任一限制访问。 | | *dwShareMode* | 打开 CONIN$时，请指定 **FILE\_SHARE\_READ**。 打开 CONOUT$时，请指定 **FILE\_SHARE\_WRITE**。  如果调用进程继承控制台，或者子进程应能够访问控制台，则必须 FILE\_SHARE\_READ | FILE\_SHARE\_WRITE此参数。 | | *lpSecurityAttributes* | 如果希望继承控制台，[SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的 **bInheritHandle** 成员必须 **TRUE**。 | | *dwCreationDisposition* | 使用 **CreateFile** 打开控制台时，应指定 **OPEN\_EXISTING**。 | | *dwFlagsAndAttributes* | 忽视。 | | *hTemplateFile* | 忽视。 |     下表显示了 *dwDesiredAccess* 和 *lpFileName*的各种设置。  展开表   |  |  |  | | --- | --- | --- | | ***lpFileName*** | ***dwDesiredAccess*** | **结果** | | “CON” | **GENERIC\_READ** | 打开控制台进行输入。 | | “CON” | **GENERIC\_WRITE** | 打开控制台进行输出。 | | “CON” | GENERIC\_READ | GENERIC\_WRITE | 导致 createFile **失败**;[GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_FILE\_NOT\_FOUND**。 |    Mailslots 如果 **CreateFile** 打开 mailslot 的客户端端，则如果 mailslot 客户端尝试在 mailslot 服务器使用 [CreateMailSlot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota) 函数创建本地 mailslot 之前，该函数将返回 **INVALID\_HANDLE\_VALUE**。  有关详细信息，请参阅 [Mailslots](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslots)。 管道 如果 **CreateFile** 打开命名管道的客户端端，该函数将使用处于侦听状态的命名管道的任何实例。 打开过程可以根据需要多次复制句柄，但在打开句柄后，命名管道实例无法由另一个客户端打开。 打开管道时指定的访问必须与 createNamedPipe 函数的 dwOpenMode 参数中指定的访问兼容。  如果在此操作之前未在服务器上成功调用 [CreateNamedPipe](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createnamedpipea) 函数，则管道将不存在，**CreateFile** 将失败，**ERROR\_FILE\_NOT\_FOUND**。  如果至少有一个活动管道实例，但服务器上没有可用的侦听器管道，这意味着所有管道实例当前都已连接，**CreateFile** 失败并 **ERROR\_PIPE\_BUSY**。  有关详细信息，请参阅 [管道](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/pipes)。 例子 以下主题显示了示例文件操作：   * [将一个文件追加到另一个文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/appending-one-file-to-another-file) * [取消挂起的 I/O 操作](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/canceling-pending-i-o-operations) * [使用重定向的输入和输出](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/creating-a-child-process-with-redirected-input-and-output) 创建子进程 * [创建和使用临时文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-using-a-temporary-file) * [FSCTL\_RECALL\_FILE](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_recall_file) * [GetFinalPathNameByHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfinalpathnamebyhandlea) * 在文件 中 锁定和解锁字节范围 * [从文件句柄获取文件名](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/obtaining-a-file-name-from-a-file-handle) * [获取文件系统识别信息](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/obtaining-file-system-recognition-information) * [打开文件以读取或写入](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/opening-a-file-for-reading-or-writing) * [检索 Last-Write 时间](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/retrieving-the-last-write-time) * [SetFileInformationByHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfileinformationbyhandle) * 文件 末尾的 测试 * 使用光纤 * 使用流 * [走更改日记记录的缓冲区](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/walking-a-buffer-of-change-journal-records) * [Wow64DisableWow64FsRedirection](https://learn.microsoft.com/zh-cn/windows/win32/api/wow64apiset/nf-wow64apiset-wow64disablewow64fsredirection) * [Wow64EnableWow64FsRedirection](https://learn.microsoft.com/zh-cn/windows/win32/api/wow64apiset/nf-wow64apiset-wow64enablewow64fsredirection)   以下主题演示了物理设备 I/O：   * [呼叫 DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/calling-deviceiocontrol) * [配置通信资源](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/configuring-a-communications-resource) * [监视通信事件](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/monitoring-communications-events) * [处理删除设备](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/processing-a-request-to-remove-a-device) 的请求   使用命名管道的示例位于 [命名管道客户端](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/named-pipe-client)。  使用 mailslot 会显示 [写入 Mailslot](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/writing-to-a-mailslot)。  可以在创建备份应用程序找到磁带备份代码片段。  **备注**  fileapi.h 标头将 CreateFile 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。 将非中性编码别名与非非编码的代码混合使用可能会导致编译或运行时错误不匹配。 有关详细信息，请参阅函数原型的 约定。 要求  |  |  | | --- | --- | | **标头** | fileapi.h （包括 Windows.h） | | **库** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [关于目录管理](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/about-directory-management)  关于卷管理 的  [备份](https://learn.microsoft.com/zh-cn/windows/desktop/Backup/backup)  [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)  [通信](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/communications-resources)  [CreateDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createdirectorya)  [CreateDirectoryEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createdirectoryexa)  [CreateFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfiletransacteda)  [CreateMailSlot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota)  [CreateNamedPipe](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createnamedpipea)  [创建、删除和维护文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating--deleting--and-maintaining-files)  [DeleteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-deletefilea)  [设备输入和输出控制（IOCTL）](https://learn.microsoft.com/zh-cn/windows/desktop/DevIO/device-input-and-output-control-ioctl-)  [DeviceIoControl](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-deviceiocontrol)  [文件压缩和解压缩](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-compression-and-decompression)  [文件加密](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-encryption)  [文件管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)  [文件流](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-streams)  **Functions**  [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)  [I/O 完成端口](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/i-o-completion-ports)  [I/O 概念](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/i-o-concepts)  [Mailslots](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslots)  [获取和设置文件信息](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/obtaining-and-setting-file-information)  **概述主题**  [管道](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/pipes)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [ReadFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfileex)  使用特殊特权运行的  [SetFileAttributes](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfileattributesa)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)  [WriteFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefileex) |

### GetWindowsDirectoryA 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索 Windows 目录的路径。  提供此函数主要用于与旧版应用程序的兼容性。 新应用程序应将代码存储在“程序文件”文件夹中，并将持久数据存储在用户配置文件的“应用程序数据”文件夹中。 有关详细信息，请参阅 [ShGetFolderPath](https://learn.microsoft.com/zh-cn/windows/desktop/api/shlobj_core/nf-shlobj_core-shgetfolderpatha)。 语法 C++复制  UINT GetWindowsDirectoryA(  [out] LPSTR lpBuffer,  [in] UINT uSize  ); 参数 [out] lpBuffer  指向接收路径的缓冲区的指针。 除非 Windows 目录是根目录，否则此路径不会以反斜杠结尾。 例如，如果 Windows 目录在驱动器 C 上名为 Windows，则此函数检索到的 Windows 目录的路径为 C：\Windows。 如果系统安装在驱动器 C 的根目录中，则检索的路径为 C：。  [in] uSize  *lpBuffer* 参数指定的缓冲区的最大大小（以 **TCHAR 为单位**）。 此值应设置为 **MAX\_PATH**。 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 **TCHAR** 为单位），不包括终止 null 字符。  如果长度大于缓冲区的大小，则返回值是保存路径所需的缓冲区的大小。  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 Windows 目录是某些旧版应用程序在其中存储初始化和帮助文件的目录。 新应用程序不应将文件存储在 Windows 目录中;相反，它们应将系统范围的数据存储在应用程序的安装目录中，并将特定于用户的数据存储在用户的配置文件中。  如果用户运行的是共享版本的系统，则保证 Windows 目录对于每个用户都是私有的。  如果应用程序创建希望按用户存储的其他文件，则应将它们放在 HOMEPATH 环境变量指定的目录中。 如果管理员通过用户管理器管理工具指定，则每个用户的此目录将有所不同。 HOMEPATH 始终指定用户的主目录（保证每个用户是专用目录）或默认目录 (例如 C：\USERS\DEFAULT) 用户将拥有所有访问权限。  **终端服务：**如果应用程序在终端服务环境中运行，则每个用户都有一个专用的 Windows 目录。 系统还有一个共享的 Windows 目录。 如果应用程序是终端服务感知 (在映像标头) 中设置了 **IMAGE\_DLLCHARACTERISTICS\_TERMINAL\_SERVER\_AWARE** 标志，则此函数将返回系统 Windows 目录的路径，就像 [GetSystemWindowsDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemwindowsdirectorya) 函数一样。 否则，它将检索用户的专用 Windows 目录的路径。 示例 有关示例，请参阅 [获取系统信息](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/getting-system-information)。  **备注**  sysinfoapi.h 标头将 GetWindowsDirectory 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** | sysinfoapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetCurrentDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getcurrentdirectory)  [GetSystemDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemdirectorya)  [GetSystemWindowsDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemwindowsdirectorya)  [系统信息函数](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/system-information-functions) |

### GetCurrentDirectory 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索当前进程的当前目录。 语法 C++  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**。 缓冲区长度必须包含终止 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。  若要确定所需的缓冲区大小，请将此参数设置为 **NULL** ，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。 要获得更多的错误信息，请调用 GetLastError。  如果 *lpBuffer* 指向的缓冲区不够大，则返回值以字符为单位指定缓冲区的所需大小，包括 null 终止符。 注解 每个进程都有一个包含两个部分的当前目录：   * 驱动器号后跟冒号的磁盘指示符，或后跟冒号的服务器名称 (\\*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/zh-cn/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 和 应避免使用相对路径名称。 [SetCurrentDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setcurrentdirectory) 函数写入的当前目录状态存储在每个进程中作为全局变量，因此多线程应用程序无法可靠地使用此值，而不会从可能读取或设置此值的其他线程中损坏数据。 此限制也适用于 **SetCurrentDirectory** 和 [GetFullPathName](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-getfullpathnamea) 函数。 例外情况是保证应用程序在单个线程中运行，例如，在创建任何其他线程之前，从main线程中的命令行参数字符串分析文件名。 在多线程应用程序或共享库代码中使用相对路径名称可能会产生不可预知的结果，因此不受支持。  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 [更改当前目录](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/changing-the-current-directory)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createdirectorya)  [目录管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/directory-management-functions)  [GetSystemDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemdirectorya)  [GetWindowsDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getwindowsdirectorya)  [RemoveDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-removedirectorya)  [SetCurrentDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setcurrentdirectory) |

### GetTempPathA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索为临时文件指定的目录的路径。 语法 C++复制  DWORD GetTempPathA(  [in] DWORD nBufferLength,  [out] LPSTR lpBuffer  ); 参数 [in] nBufferLength  *lpBuffer* 标识的字符串缓冲区的大小（以 **TCHAR 为单位**）。  [out] lpBuffer  指向字符串缓冲区的指针，该缓冲区接收指定临时文件路径的以 null 结尾的字符串。 返回的字符串以反斜杠结尾，例如“C：\TEMP\”。 返回值 如果函数成功，则返回值为复制到 *lpBuffer* 的字符串的长度（以 **TCHAR 为单位**），不包括终止 null 字符。 如果返回值大于 *nBufferLength*，则返回值是保存路径所需的缓冲区的长度（ **以 TCHAR** 为单位）。  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。  可能的最大返回值为 **MAX\_PATH**+1 (261) 。 备注 **备注**  应用应调用 [**GetTempPath2**](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-gettemppath2a) 而不是 **GetTempPath**。  **GetTempPath** 函数按以下顺序检查是否存在环境变量，并使用找到的第一个路径：   1. TMP 环境变量指定的路径。 2. TEMP 环境变量指定的路径。 3. USERPROFILE 环境变量指定的路径。 4. Windows 目录。   请注意，函数不会验证路径是否存在，也不会测试当前进程是否对路径具有任何类型的访问权限。 **GetTempPath** 函数返回格式正确的字符串，该字符串根据之前指定的环境变量搜索顺序指定完全限定的路径。 在对文件 I/O 操作进行任何使用之前，应用程序应验证路径是否存在和对路径的充分访问权限。  符号链接行为 - 如果路径指向符号链接，则临时路径名称将保留任何符号链接。  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 [创建和使用临时文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-using-a-temporary-file)。  **备注**  fileapi.h 标头将 GetTempPath 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 [文件管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [GetTempFileName](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-gettempfilenamea)  [符号链接](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/symbolic-links) |

### WriteFile函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将数据写入指定的文件或输入/输出 (I/O) 设备。  此函数设计用于同步和异步操作。 有关专为异步操作设计的类似函数，请参阅 [WriteFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefileex)。 语法 C++复制  BOOL WriteFile(  [in] HANDLE hFile,  [in] LPCVOID lpBuffer,  [in] DWORD nNumberOfBytesToWrite,  [out, optional] LPDWORD lpNumberOfBytesWritten,  [in, out, optional] LPOVERLAPPED lpOverlapped  ); 参数 [in] hFile  文件或 I/O 设备的句柄 (例如文件、文件流、物理磁盘、卷、控制台缓冲区、磁带驱动器、套接字、通信资源、mailslot 或管道) 。  必须已创建具有写入访问权限的 *hFile* 参数。 有关详细信息，请参阅 [通用访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/generic-access-rights) 和 [文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)。  对于异步写入操作，*hFile* 可以是使用 **FILE\_FLAG\_OVERLAPPED** 标志的 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 函数打开的任何句柄，也可以是套接字或 [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 函数返回的[套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket)句柄。  [in] lpBuffer  指向缓冲区的指针，该缓冲区包含要写入文件或设备的数据。  此缓冲区必须在写入操作期间保持有效。 在完成写入操作之前，调用方不得使用此缓冲区。  [in] nNumberOfBytesToWrite  要写入文件或设备的字节数。  值为零指定 null 写入操作。 null 写入操作的行为取决于基础文件系统或通信技术。  **Windows Server 2003 和 Windows XP：**网络上的管道写入操作在每次写入的大小方面受到限制。 金额因平台而异。 对于 x86 平台，为 63.97 MB。 对于 x64 平台，为 31.97 MB。 对于 Itanium，为 63.95 MB。 有关管道的详细信息，请参阅“备注”部分。  [out, optional] lpNumberOfBytesWritten  指向变量的指针，该变量接收使用同步 *hFile* 参数时写入的字节数。 **WriteFile** 在执行任何工作或错误检查之前将此值设置为零。 如果这是异步操作，请对此参数使用 **NULL** ，以避免潜在的错误结果。  仅当 *lpOverlapped* 参数不为 **NULL** 时，此参数才能为 **NULL**。  **Windows 7：**此参数不能为 **NULL**。  有关详细信息，请参见“备注”部分。  [in, out, optional] lpOverlapped  如果使用 **FILE\_FLAG\_OVERLAPPED** 打开 *hFile* 参数，则需要指向 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构的指针，否则此参数可以为 **NULL**。  对于支持字节偏移量的 *hFile* ，如果使用此参数，则必须指定开始写入文件或设备的字节偏移量。 此偏移量是通过设置 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构的 **Offset** 和 **OffsetHigh** 成员指定的。 对于不支持字节偏移量的 *hFile* ， **将忽略 Offset** 和 **OffsetHigh** 。  若要写入文件末尾，请将 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构的 **Offset** 和 **OffsetHigh** 成员指定为0xFFFFFFFF。 这在功能上等效于之前调用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 函数以使用**FILE\_APPEND\_DATA**访问打开 *hFile*。  有关 *lpOverlapped* 和 **FILE\_FLAG\_OVERLAPPED**的不同组合的详细信息，请参阅“备注”部分和 [“同步和文件位置”](https://learn.microsoft.com/zh-cn/) 部分。 返回值 如果函数成功，则返回值为非零 (**TRUE**) 。  如果函数失败或正在异步完成，则返回值为零 (**FALSE**) 。 若要获得更多的错误信息，请调用 GetLastError 函数。  **注意**[**GetLastError**](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 代码**ERROR\_IO\_PENDING**不是失败;它指定写入操作正在异步等待完成。 有关详细信息，请参阅“备注”。   注解 发生以下情况之一时， **WriteFile** 函数将返回 ：   * 写入请求的字节数。 * 如果写入被阻止) ，读取操作会释放管道 (读取端的缓冲区空间。 有关详细信息，请参阅 [管道](https://learn.microsoft.com/zh-cn/) 部分。 * 正在使用异步句柄，并且写入正在异步进行。 * 发生错误。   每当存在过多的未完成异步 I/O 请求时， **WriteFile** 函数可能会失败并出现 **ERROR\_INVALID\_USER\_BUFFER** 或 **ERROR\_NOT\_ENOUGH\_MEMORY** 。  若要取消所有挂起的异步 I/O 操作，请使用以下任一函数：   * [CancelIo](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/cancelio) - 此函数仅取消由指定文件句柄的调用线程发出的操作。 * CancelIoEx - 此函数取消由指定文件句柄的线程发出的所有操作。   使用 [CancelSynchronousIo](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/cancelsynchronousio-func) 函数取消挂起的同步 I/O 操作。  取消的 I/O 操作已完成， **ERROR\_OPERATION\_ABORTED**错误。  **WriteFile** 函数可能会失败**并ERROR\_NOT\_ENOUGH\_QUOTA**，这意味着调用进程的缓冲区无法锁定页。 有关详细信息，请参阅 [SetProcessWorkingSetSize](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-setprocessworkingsetsize)。  如果文件的一部分被另一个进程锁定，并且写入操作与锁定部分重叠， **WriteFile** 将失败。  写入文件时，在关闭用于写入的所有句柄之前，最后一次写入时间不会完全更新。 因此，若要确保准确的上次写入时间，请在写入文件后立即关闭文件句柄。  在写入操作使用缓冲区时访问输出缓冲区可能会导致从该缓冲区写入的数据损坏。 在写入操作完成之前，应用程序不得写入、重新分配或释放写入操作正在使用的输出缓冲区。 使用异步文件句柄时，这可能会特别出现问题。 稍后可在同步 [和文件位置](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-writefile#synchronization-and-file-position) 部分以及 [同步和异步 I/O 中找到有关同步文件句柄与异步文件](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/synchronous-and-asynchronous-i-o)句柄的其他信息。  请注意，可能无法为远程文件正确更新时间戳。 若要确保结果一致，请使用无缓冲区 I/O。  系统将要写入的零个字节解释为指定 null 写入操作， **WriteFile** 不会截断或扩展文件。 若要截断或扩展文件，请使用 [SetEndOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setendoffile) 函数。  可以使用具有控制台输出句柄的 **WriteFile** 将字符写入屏幕缓冲区。 函数的确切行为由控制台模式确定。 数据将写入当前光标位置。 光标位置在写入操作后更新。 有关控制台句柄的详细信息，请参阅 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)。  写入通信设备时，**WriteFile** 的行为由当前通信超时确定，该超时是使用 [SetCommTimeouts 和 GetCommTimeouts](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setcommtimeouts) 函数设置和检索的。 如果未能设置超时值，可能会出现不可预知的结果。 有关通信超时的详细信息，请参阅 [COMMTIMEOUTS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-commtimeouts)。  尽管单扇区写入是原子性的，但不能保证多扇区写入是原子的，除非使用事务 (即创建的句柄是事务处理句柄;例如，使用 [CreateFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfiletransacteda)) 创建的句柄。 缓存的多扇区写入可能不会始终立即写入磁盘;因此，请在 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 中指定**FILE\_FLAG\_WRITE\_THROUGH**，以确保将整个多扇区写入磁盘，而不会造成潜在的缓存延迟。  如果直接写入具有已装载文件系统的卷，则必须先获取该卷的独占访问权限。 否则，可能会导致数据损坏或系统不稳定，因为应用程序的写入操作可能与来自文件系统的其他更改冲突，并使卷的内容处于不一致状态。 为防止这些问题，Windows Vista 及更高版本中进行了以下更改：   * 如果卷没有装载的文件系统，或者满足以下条件之一，则对卷句柄的写入将成功：   + 要写入的扇区是启动扇区。   + 要写入的扇区驻留在文件系统空间之外。   + 已使用 [FSCTL\_LOCK\_VOLUME](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_lock_volume) 或 [FSCTL\_DISMOUNT\_VOLUME](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_dismount_volume) 显式锁定或卸载卷。   + 该卷没有实际的文件系统。 (换句话说，它具有 RAW 文件系统 mounted.) * 如果满足以下条件之一，则磁盘句柄上的写入将成功：   + 要写入的扇区不在卷的范围内。   + 要写入的扇区位于已装载卷内，但已使用 [FSCTL\_LOCK\_VOLUME](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_lock_volume) 或 [FSCTL\_DISMOUNT\_VOLUME](https://learn.microsoft.com/zh-cn/windows/desktop/api/winioctl/ni-winioctl-fsctl_dismount_volume) 显式锁定或卸载卷。   + 要写入到的扇区位于没有除 RAW 以外的已装载文件系统的卷内。   使用 **FILE\_FLAG\_NO\_BUFFERING** 成功处理使用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 打开的文件有严格的要求。 有关详细信息，请参阅 [文件缓冲](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-buffering)。  如果使用 **FILE\_FLAG\_OVERLAPPED** 打开 *hFile*，则以下条件有效：   * *lpOverlapped* 参数必须指向有效且唯一[的 OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构，否则函数可能会错误地报告写入操作已完成。 * *lpNumberOfBytesWritten* 参数应设置为 **NULL**。 若要获取写入的字节数，请使用 [GetOverlappedResult](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-getoverlappedresult) 函数。 如果 *hFile* 参数与 I/O 完成端口相关联，则还可以通过调用 [GetQueuedCompletionStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-getqueuedcompletionstatus) 函数来获取写入的字节数。   在 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 (CSV) | 是 | | 弹性文件系统 (ReFS) | 是 |    同步和文件位置 如果使用 **FILE\_FLAG\_OVERLAPPED** 打开 *hFile*，则它是异步文件句柄;否则，它是同步的。 如前所述，每个使用 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构的规则略有不同。  **注意** 如果为异步 I/O 打开文件或设备，则使用该句柄对函数（如 **WriteFile** ）的后续调用通常会立即返回，但也会在被阻止的执行方面以同步方式运行。 有关详细信息，请参阅 [**http://support.microsoft.com/kb/156932**](https://support.microsoft.com/kb/156932)。    使用异步文件句柄时的注意事项：   * **WriteFile** 可能会在写入操作完成之前返回。 在此方案中， **WriteFile** 返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数返回 **ERROR\_IO\_PENDING**，这允许调用进程在系统完成写入操作时继续。 * *lpOverlapped* 参数不能为 **NULL**，并且应考虑到以下事实：   + 尽管在 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构中指定的事件由系统自动设置和重置，但在 **OVERLAPPED** 结构中指定的偏移量不会自动更新。   + **WriteFile** 在开始 I/O 操作时将事件重置为非对齐状态。   + 在 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构中指定的事件在写入操作完成时设置为信号状态;在该时间之前，写入操作被视为挂起。   + 由于写入操作从 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构中指定的偏移量开始，并且 **WriteFile** 可能在系统级写入操作完成之前返回， (写入挂起) ，因此，在事件发出信号之前，应用程序不应修改、释放或重复使用该结构的任何其他部分，直到事件发出信号 (， 写入) 完成。   使用同步文件句柄时的注意事项：   * 如果 *lpOverlapped* 为 **NULL**，则写入操作从当前文件位置开始， **并且 WriteFile** 在操作完成之前不会返回，并且系统会在 **WriteFile** 返回之前更新文件指针。 * 如果 *lpOverlapped* 不为 **NULL**，则写入操作在 [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped) 结构中指定的偏移量处开始， **并且 WriteFile** 在写入操作完成之前不会返回。 在 **WriteFile** 返回之前，系统将更新 **OVERLAPPED** Internal 和 InternalHigh 字段以及文件指针。   有关详细信息，请参阅 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 和 [同步和异步 I/O](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/synchronous-and-asynchronous-i-o)。 管道 如果使用匿名管道且已关闭读取句柄，当 **WriteFile** 尝试使用管道的相应写入句柄进行写入时，函数返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_BROKEN\_PIPE**。  如果应用程序使用 **WriteFile** 函数写入管道时管道缓冲区已满，则写入操作可能不会立即完成。 当使用 [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 函数 (读取操作) 为管道提供更多系统缓冲区空间时，将完成写入操作。  写入缓冲区空间不足的非阻塞的字节模式管道句柄时， **WriteFile** 返回 **TRUE** ，其中包含 \**lpNumberOfBytesWritten*<*nNumberOfBytesToWrite*。  有关管道的详细信息，请参阅 [管道](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/pipes)。 事务处理操作 如果存在绑定到句柄的事务，则会事务处理文件写入。 有关详细信息，请参阅 [关于事务 NTFS](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/about-transactional-ntfs)。 示例 有关一些示例，请参阅[创建和使用临时文件和](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/creating-and-using-a-temporary-file)[打开文件进行读取或写入](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/opening-a-file-for-reading-or-writing)。  下面的 C++ 示例演示如何对齐未缓冲区文件写入的扇区。 *Size* 变量是你有兴趣写入文件的原始数据块的大小。 有关未缓冲文件 I/O 的其他规则，请参阅 [文件缓冲](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-buffering)。  C++复制  #include <windows.h>  #define ROUND\_UP\_SIZE(Value,Pow2) ((SIZE\_T) ((((ULONG)(Value)) + (Pow2) - 1) & (~(((LONG)(Pow2)) - 1))))  #define ROUND\_UP\_PTR(Ptr,Pow2) ((void \*) ((((ULONG\_PTR)(Ptr)) + (Pow2) - 1) & (~(((LONG\_PTR)(Pow2)) - 1))))  int main()  {  // Sample data  unsigned long bytesPerSector = 65536; // obtained from the GetFreeDiskSpace function.  unsigned long size = 15536; // Buffer size of your data to write.    // Ensure you have one more sector than Size would require.  size\_t sizeNeeded = bytesPerSector + ROUND\_UP\_SIZE(size, bytesPerSector);    // Replace this statement with any allocation routine.  auto buffer = new uint8\_t[SizeNeeded];    // Actual alignment happens here.  auto bufferAligned = ROUND\_UP\_PTR(buffer, bytesPerSector);  // ... Add code using bufferAligned here.    // Replace with corresponding free routine.  delete buffer;  } 要求  |  |  | | --- | --- | | **标头** | fileapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CancelIo](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/cancelio)  [CancelIoEx](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/cancelioex-func)  [CancelSynchronousIo](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/cancelsynchronousio-func)  [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)  [CreateFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfiletransacteda)  [文件管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)  [GetOverlappedResult](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-getoverlappedresult)  [GetQueuedCompletionStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-getqueuedcompletionstatus)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SetEndOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setendoffile)  [WriteFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefileex) |

### CloseHandle函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 关闭打开的对象句柄。 语法 C++复制  BOOL CloseHandle(  [in] HANDLE hObject  ); 参数 [in] hObject  打开对象的有效句柄。 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。  如果应用程序在调试器下运行，则函数在收到无效的句柄值或伪句柄值时将引发异常。 如果关闭句柄两次，或者对 [FindFirstFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-findfirstfilea) 函数返回的句柄调用 **CloseHandle** 而不是调用 [FindClose](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-findclose) 函数，则可能会发生这种情况。 注解 **CloseHandle** 函数关闭以下对象的句柄：   * 访问令牌 * 通信设备 * 控制台输入 * 控制台屏幕缓冲区 * 事件 * 文件 * 文件映射 * I/O 完成端口 * 作业 * Mailslot * 内存资源通知 * Mutex * 命名管道 * 管道 * 进程 * Semaphore * 线程 * 事务 * 可等待计时器   创建这些对象的函数的文档指出，完成对象后应使用 **CloseHandle** ，以及关闭句柄后对对象挂起的操作会发生什么情况。 通常， **CloseHandle** 会使指定的对象句柄失效，减少对象的句柄计数，并执行对象保留检查。 关闭对象的最后一个句柄后，将从系统中删除该对象。 有关这些对象的创建者函数的摘要，请参阅 [内核对象](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/kernel-objects)。  通常，应用程序应为每个打开的句柄调用 **CloseHandle** 一次。 如果使用句柄的函数因ERROR\_INVALID\_HANDLE而失败，通常不需要调用 **CloseHandle** ，因为此错误通常表示句柄已无效。 但是，某些函数使用 ERROR\_INVALID\_HANDLE 来指示对象本身不再有效。 例如，如果网络连接断开，尝试使用对网络上文件的句柄的函数可能会失败并ERROR\_INVALID\_HANDLE，因为文件对象不再可用。 在这种情况下，应用程序应关闭句柄。  如果处理句柄，则应在提交事务之前关闭绑定到事务的所有句柄。 如果使用 FILE\_FLAG\_DELETE\_ON\_CLOSE 标志调用 [CreateFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfiletransacteda) 打开了事务处理句柄，则在应用程序关闭句柄并调用 [CommitTransaction](https://learn.microsoft.com/zh-cn/windows/desktop/api/ktmw32/nf-ktmw32-committransaction) 之前，不会删除该文件。 有关事务处理对象的详细信息，请参阅 [使用事务](https://learn.microsoft.com/zh-cn/windows/desktop/Ktm/programming-model)。  关闭线程句柄不会终止关联的线程或删除线程对象。 关闭进程句柄不会终止关联的进程或删除进程对象。 若要删除线程对象，必须终止该线程，然后关闭该线程的所有句柄。 有关详细信息，请参阅 [终止线程](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/terminating-a-thread)。 若要删除进程对象，必须终止进程，然后关闭进程的所有句柄。 有关详细信息，请参阅 [终止进程](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/terminating-a-process)。  即使仍有文件视图处于打开状态，关闭文件映射的句柄也会成功。 有关详细信息，请参阅 [关闭文件映射对象](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/closing-a-file-mapping-object)。  请勿使用 **CloseHandle** 函数关闭套接字。 请改用 [closesocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-closesocket) 函数，该函数释放与套接字关联的所有资源，包括套接字对象的句柄。 有关详细信息，请参阅 [套接字关闭](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/graceful-shutdown-linger-options-and-socket-closure-2)。  请勿使用 **CloseHandle** 函数关闭打开的注册表项的句柄。 请改用 [RegCloseKey](https://learn.microsoft.com/zh-cn/windows/desktop/api/winreg/nf-winreg-regclosekey) 函数。 **CloseHandle** 不会关闭注册表项的句柄，但不返回指示此失败的错误。 示例 C++复制  dwPriorityClass = 0;  hProcess = OpenProcess( PROCESS\_ALL\_ACCESS, FALSE, pe32.th32ProcessID );  if( hProcess == NULL )  printError( TEXT("OpenProcess") );  else  {  dwPriorityClass = GetPriorityClass( hProcess );  if( !dwPriorityClass )  printError( TEXT("GetPriorityClass") );  CloseHandle( hProcess );  }  若要在上下文中查看此示例，请参阅 [拍摄快照和查看进程](https://learn.microsoft.com/zh-cn/windows/desktop/ToolHelp/taking-a-snapshot-and-viewing-processes)。 要求  |  |  | | --- | --- | | **标头** | handleapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)  [CreateFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfiletransacteda)  [DeleteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-deletefilea)  [FindClose](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-findclose)  [FindFirstFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-findfirstfilea)  [句柄和对象函数](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/handle-and-object-functions)  [内核对象](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/kernel-objects)  [对象接口](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/object-interface) |

## 注意：vs2010有时候又一个非常坑爹的地方，就是变量的定义一定要在开头，后面定义会报一些奇怪的错误如：DWORD dwWritten;

# 扩展:用MFC对话框程序来获取系统目录,windows安装目录,当前目录和临时文件目录

## 1.新建一个MFC对话框程序,取名: lesson5Dirctories注意:用vs2019创建对话框项目时项目名称不要又连接符-或下划线\_,否则会报一个奇怪的错误.先把默认的框架删除,然后查询PreTranslateMessage函数来屏蔽回车和esc键.

|  |
| --- |
|  |
|  |

## 2.给对话框布局如下

|  |
| --- |
|  |
|  |

## 3.然后我们给按钮添加点击事件处理函数,代码如下

|  |
| --- |
| void Clesson5DirctoriesDlg::OnBnClickedBtnGet()  {  // TODO: 在此添加控件通知处理程序代码  WCHAR szInfo[1024];  memset(szInfo, 0, 1024);  //SYSTEM dir  GetSystemDirectory(szInfo, 1024);  SetDlgItemText(IDC\_ST\_SYSDIR, szInfo);  //WINDOW dir  memset(szInfo, 0, 1024);  GetWindowsDirectory(szInfo, 1024);  SetDlgItemText(IDC\_ST\_WINDIR, szInfo);  //Current dir  memset(szInfo, 0, 1024);  GetCurrentDirectory(1024, szInfo);  SetDlgItemText(IDC\_ST\_CURDIR, szInfo);  //Temp dir  memset(szInfo, 0, 1024);  GetTempPath(1024, szInfo);  SetDlgItemText(IDC\_ST\_TMPDIR, szInfo);  } |

### 编译运行程序,点击按钮,效果如下

|  |
| --- |
|  |