# 学习大纲

|  |
| --- |
|  |

# API用法说明

# ReadFile函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从指定的文件或输入/输出 (I/O) 设备读取数据。 读取发生在文件指针指定的位置（如果设备支持）。  此函数适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 [ReadFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfileex)。 语法 C++  BOOL ReadFile(  [in] HANDLE hFile,  [out] LPVOID lpBuffer,  [in] DWORD nNumberOfBytesToRead,  [out, optional] LPDWORD lpNumberOfBytesRead,  [in, out, optional] LPOVERLAPPED lpOverlapped  ); 参数 [in] hFile  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) 。  必须已创建具有读取访问权限 *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)。  对于异步读取操作，*hFile* 可以是 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 函数使用 **FILE\_FLAG\_OVERLAPPED** 标志打开的任何句柄，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。  [out] lpBuffer  指向接收从文件或设备读取的数据的缓冲区的指针。  此缓冲区必须在读取操作期间保持有效。 在完成读取操作之前，调用方不得使用此缓冲区。  [in] nNumberOfBytesToRead  要读取的最多字节数。  [out, optional] lpNumberOfBytesRead  指向变量的指针，该变量接收在使用同步 *hFile* 参数时读取的字节数。 **ReadFile** 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，否则可为 **NULL**。  如果使用 **FILE\_FLAG\_OVERLAPPED** 打开 *hFile*，*则 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读取操作已完成。  对于支持字节偏移量的 *hFile* ，如果使用此参数，则必须指定从文件或设备开始读取的字节偏移量。 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** 。  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”** 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;它指定读取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， **ReadFile** 函数将返回 ：   * 读取请求的字节数。 * 写入操作在管道的写入端完成。 * 正在使用异步句柄，并且读取正在异步进行。 * 发生错误。   每当存在过多的未完成异步 I/O 请求时，**ReadFile** 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 - 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 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。  **ReadFile** 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，并且读取操作与锁定部分重叠，则此函数将失败。  在读取操作使用缓冲区时访问输入缓冲区可能会导致读取到该缓冲区的数据损坏。 在读取操作完成之前，应用程序不得读取、写入、重新分配或释放读取操作正在使用的输入缓冲区。 使用异步文件句柄时，这可能会特别出现问题。 有关同步文件句柄与异步文件句柄的其他信息，请参阅 [同步和文件位置](https://learn.microsoft.com/zh-cn/) 部分以及 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 参考主题。  可以使用具有控制台输入句柄的 **ReadFile** 从控制台输入缓冲区读取字符。 控制台模式确定 **ReadFile** 函数的确切行为。 默认情况下，控制台模式 **为ENABLE\_LINE\_INPUT**，指示 **ReadFile** 应在到达回车符之前读取。 如果按 Ctrl+C，调用成功，但 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_OPERATION\_ABORTED**。 有关详细信息，请参阅 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)。  从通信设备读取时，**ReadFile** 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)。  如果 **ReadFile** 尝试从缓冲区太小的 mailslot 读取，则函数返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_INSUFFICIENT\_BUFFER**。  成功处理使用 **FILE\_FLAG\_NO\_BUFFERING 标志通过**[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读取操作已完成。 * *lpNumberOfBytesRead* 参数应设置为 **NULL**。 使用 [GetOverlappedResult](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-getoverlappedresult) 函数获取实际读取的字节数。 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) 函数获取读取的字节数。  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，则对函数（如使用该句柄的 **ReadFile** ）的后续调用通常会立即返回，但对于被阻止的执行，也可以同步运行。 有关更多信息，请参见[**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：   * **ReadFile** 可能会在读取操作完成之前返回。 在此方案中， **ReadFile** 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读取操作时继续。 * *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。   + **ReadFile** 在开始 I/O 操作时将事件重置为未签名状态。   + 当读取操作完成时， [在 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;在该时间之前，读取操作被视为挂起。   + 由于读取操作从 [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 **ReadFile** 可能在系统级读取操作完成之前返回， (读取挂起) ，因此，在向事件 (发出信号之前，应用程序不应修改、释放或重复使用该偏移量或结构的任何其他部分， 读取) 完成。   + 如果在异步操作期间检测到文件结尾 (EOF) ，则调用该操作的 [GetOverlappedResult](https://learn.microsoft.com/zh-cn/windows/desktop/api/ioapiset/nf-ioapiset-getoverlappedresult) 将返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_HANDLE\_EOF**。   使用同步文件句柄时的注意事项：   * 如果 *lpOverlapped* 为 **NULL**，则读取操作从当前文件位置开始， **ReadFile** 在操作完成之前不会返回，并且系统会在 **ReadFile** 返回之前更新文件指针。 * 如果 *lpOverlapped* 不为 **NULL**，则读取操作从 [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， **ReadFile** 在读取操作完成之前不会返回。 在 **ReadFile** 返回之前，系统将更新 **OVERLAPPED** 偏移量和文件指针。 * 如果 *lpOverlapped* 为 **NULL**，则当同步读取操作到达文件末尾时， **ReadFile** 将返回 **TRUE** 并将 设置为 \*lpNumberOfBytesRead 零。 * 如果 *lpOverlapped* 不为 **NULL**，则当同步读取操作到达文件末尾时， **ReadFile** 返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_HANDLE\_EOF**。   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写入句柄已关闭，则 **当 ReadFile** 尝试使用管道的相应读取句柄进行读取时，函数返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_BROKEN\_PIPE**。  如果在消息模式下读取命名管道，并且下一条消息的长度超过 *nNumberOfBytesToRead* 参数指定的时间， **则 ReadFile** 返回 **FALSE** ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_MORE\_DATA**。 消息的其余部分可以通过后续调用 **ReadFile** 或 [PeekNamedPipe](https://learn.microsoft.com/zh-cn/windows/desktop/api/namedpipeapi/nf-namedpipeapi-peeknamedpipe) 函数来读取。  如果**当 ReadFile** 在管道上返回 **TRUE** 时*，lpNumberOfBytesRead* 参数为零，则管道的另一端调用 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 函数，并将 *nNumberOfBytesToWrite* 设置为零。  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 如果存在绑定到文件句柄的事务，则该函数将从文件的事务处理视图中返回数据。 事务处理读取句柄保证在句柄持续时间内显示文件的相同视图。 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)。  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 有关演示如何测试文件结尾的代码示例，请参阅 [测试文件结尾](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/testing-for-the-end-of-a-file)。 有关其他示例，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)。 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)  [文件管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [GetCommTimeouts](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getcommtimeouts)  [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)  [OVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-overlapped)  [PeekNamedPipe](https://learn.microsoft.com/zh-cn/windows/desktop/api/namedpipeapi/nf-namedpipeapi-peeknamedpipe)  [ReadFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfileex)  [SetCommTimeouts](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setcommtimeouts)  [SetErrorMode](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-seterrormode)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

# 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 展开表   |  |  | | --- | --- | | **最低受支持的客户端** | Windows XP [桌面应用 | UWP 应用] | | **最低受支持的服务器** | Windows Server 2003 [桌面应用 | UWP 应用] | | **目标平台** | Windows | | **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) |

# 1.定义宽字符串

|  |
| --- |
|  |

# 2关于MessageBox函数的第一个参数的赋值问题

## 如果给他赋值一个窗口句柄，那个句柄对于的窗口就是消息框的所有者，消息框始终覆盖在窗口上面，如果传递NULL，那么消息框的所有者的桌面。有时候会很不方便。建议传递窗口句柄，这样子只有你销毁了一个消息框，才会有第二个消息框产生。

# 3.下面我们来学习一下读取文件内容的API

## 1.还是我们的Project1项目，我们把原来的代码注释了

|  |
| --- |
|  |

## 2.然后我们在源文件所在的目录下面新建一个txt文件

|  |
| --- |
|  |
|  |

## 3.我们添加下面的读取代码，在读取文件之前，先要创建一个文件句柄，调用CreateFile函数创建新文件或者打开一个现存的文件

|  |
| --- |
|  |

### 注意：CreateFile既可以创建文件，又可以打开文件.我们的程序运行的时候需要把文件拷贝到exe文件所在的目录否则找不到文件。打开文件后，需要判断一下是否打开成功，不成功程序就需要返回。

## 4.打开成功后我们来读取，先定义一个读取缓冲区的大小

|  |
| --- |
|  |

## 5.读取文件的代码如下

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 注意：文本文件的字符编码需要改为ANSI否则有很多乱码。

# app.c的代码如下

|  |
| --- |
| #include<stdio.h>  #include<Windows.h>  /\*\*  在命令行里面使用windows数据类型  \*/  #define BUF\_SIZE 4096  int main()  {  //读取文件  HANDLE hFile = CreateFile(TEXT("test1.txt"), GENERIC\_READ| GENERIC\_WRITE, FILE\_SHARE\_READ,  NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL,NULL);  //判断是否打开成功，如果失败提出消息框，并且在命令行显示具体错误信息  if (hFile == INVALID\_HANDLE\_VALUE)  {  MessageBox(NULL, TEXT("read file failed..."), TEXT("read error"), MB\_OK);  printf("file error:%x",GetLastError());  return -1;  }  //打开文件成功，就可以读取了，需要使用循环  DWORD nIn;  CHAR buffer[BUF\_SIZE];  memset(buffer, 0, BUF\_SIZE);  while (ReadFile(  hFile,  buffer,  BUF\_SIZE,  &nIn,  NULL  ) && nIn > 0)  {  printf("%s\n", buffer);  }  //读完文件记得要关闭句柄  CloseHandle(hFile);  system("pause");  return 0;  } |