# 邮槽和管道类似,都是可以用文件读写方式ReadFile和WriteFie来操作,就像操作文件句柄一样.

# 邮槽是在服务器端建立,在客户端使用,他的传输是单项的,只能由客户端向服务器输送数据,然后服务端去邮槽读取数据

# 邮槽可以用于本机的两个不同进程之间的通信,也可以用于网络里面两个主机之间的通信.

# 邮槽的名称非常特别,必须以\\开头,\\.代表本机,

# 本机邮槽名称以[\\.\mailslot\](file:///\\.\mailslot\)开头,

# 网络邮槽以[\\域名\](file:///\\域名\) 或者 [\\计算机名\](file:///\\计算机名\) 开头

# 广播邮槽以[\\\*\mailslot\](file:///\\*\mailslot\) 开头

# 邮槽不能发送太多内容,一般在420个字节以内

# 邮槽接收数据的时候是阻塞的

# 学习大纲

|  |
| --- |
|  |

# API用法详解

## CreateMailSlot函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 使用指定名称创建 mailslot，并返回 mailslot 服务器可用于对 mailslot 执行操作的句柄。 mailslot 是创建它的计算机的本地位置。 如果已存在具有指定名称的 mailslot，则会发生错误。 语法 C++复制  HANDLE CreateMailslotA(  [in] LPCSTR lpName,  [in] DWORD nMaxMessageSize,  [in] DWORD lReadTimeout,  [in, optional] LPSECURITY\_ATTRIBUTES lpSecurityAttributes  ); 参数 [in] lpName  mailslot 的名称。 此名称必须具有以下格式：  \\.\mailslot\[*path*]*name*  名称字段必须是唯一的。 该名称可能包含由反斜杠分隔的多个伪目录级别。 例如，\\.\mailslot\example\_mailslot\_name 和 \\.\mailslot\abc\def\ghi 都是有效名称。  [in] nMaxMessageSize  可写入 mailslot 的单个邮件的最大大小（以字节为单位）。 若要指定消息可以是任意大小，请将此值设置为零。  [in] lReadTimeout  读取操作在发生超时之前可以等待将邮件写入 mailslot 的时间（以毫秒为单位）。 以下值具有特殊含义。  展开表   |  |  | | --- | --- | | **值** | **含义** | | 0 | 如果没有消息，则立即返回 。 (系统不会将立即返回视为 error。) | | **MAILSLOT\_WAIT\_FOREVER**  ( (DWORD) -1) | 永远等待消息。 |     此超时值适用于所有后续读取操作和所有继承的 mailslot 句柄。  [in, optional] lpSecurityAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针。 结构的 **bInheritHandle** 成员确定返回的句柄是否可以由子进程继承。 如果 *lpSecurityAttributes* 为 **NULL**，则不能继承句柄。 返回值 如果函数成功，则返回值是 mailslot 的句柄，用于服务器 mailslot 操作。 此函数返回的句柄是异步的或重叠的。  如果函数失败，则返回值为 INVALID\_HANDLE\_VALUE。 要获得更多的错误信息，请调用 GetLastError。 注解 在满足以下条件之一之前，mailslot 一直存在：   * 最后一个 (可能继承或复制) 句柄使用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数关闭。 * 拥有最后一个 (的进程可能继承或重复) 句柄退出。   系统使用第二种方法销毁 mailslot。  若要将邮件写入 mailslot，进程使用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 函数，并使用以下格式之一指定 mailslot 名称。  展开表   |  |  | | --- | --- | | **格式** | **使用情况** | | \\.\mailslot\*name* | 检索本地 mailslot 的客户端句柄。 | | \\*computername*\mailslot\*name* | 检索远程 mailslot 的客户端句柄。 | | \\*domainname*\mailslot\*name* | 检索指定域中具有指定名称的所有 mailslot 的客户端句柄。 | | \\\*\mailslot\*name* | 检索系统主域中具有指定名称的所有 mailslot 的客户端句柄。 |     如果 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 指定域或使用星号格式指定系统的主域，则应用程序一次无法向 mailslot 写入超过 424 个字节。 如果应用程序尝试这样做， [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_BAD\_NETPATH**。  使用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 检索 mailslot 的客户端句柄时，应用程序必须指定**FILE\_SHARE\_READ**标志。  如果调用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 来访问不存在的邮件图，则将设置 **ERROR\_FILE\_NOT\_FOUND** 错误代码。 示例 有关示例，请参阅 [创建 Mailslot](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/creating-a-mailslot)。  **备注**  winbase.h 标头将 CreateMailslot 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** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)  [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)  [GetMailslotInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getmailslotinfo)  [Mailslot 函数](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslot-functions)  [Mailslots 概述](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslots)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [SetMailslotInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setmailslotinfo)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

### 邮槽创建后需要使用CreateFile函数来打开,就像打开文件一样

## CreateFile函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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) |

### 实例,[创建 Mailslot](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/creating-a-mailslot)。

C++

#include <windows.h>

#include <stdio.h>

HANDLE hSlot;

LPCTSTR SlotName = TEXT("\\\\.\\mailslot\\sample\_mailslot");

BOOL WINAPI MakeSlot(LPCTSTR lpszSlotName)

{

hSlot = CreateMailslot(lpszSlotName,

0, // no maximum message size

MAILSLOT\_WAIT\_FOREVER, // no time-out for operations

(LPSECURITY\_ATTRIBUTES) NULL); // default security

if (hSlot == INVALID\_HANDLE\_VALUE)

{

printf("CreateMailslot failed with %d\n", GetLastError());

return FALSE;

}

else printf("Mailslot created successfully.\n");

return TRUE;

}

void main()

{

MakeSlot(SlotName);

}

若要创建可由子进程继承的 mailslot，应用程序应更改作为 [**CreateMailslot**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winbase/nf-winbase-createmailslota) 的最后一个参数传递的[**SECURITY\_ATTRIBUTES**](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))结构。 为此，应用程序将此结构的 **bInheritHandle** 成员设置为 **TRUE** ， (默认设置为 **FALSE**) 。

## GetMailslotInfo 函数的语法

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索有关指定 mailslot 的信息。 语法 C++复制  BOOL GetMailslotInfo(  [in] HANDLE hMailslot,  [out, optional] LPDWORD lpMaxMessageSize,  [out, optional] LPDWORD lpNextSize,  [out, optional] LPDWORD lpMessageCount,  [out, optional] LPDWORD lpReadTimeout  ); 参数 [in] hMailslot  mailslot 的句柄。 [CreateMailslot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota) 函数必须创建此句柄。  [out, optional] lpMaxMessageSize  此 mailslot 允许的最大邮件大小（以字节为单位）。 此值可以大于或等于创建 [mailslot 的 CreateMailslot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota) 函数的 *cbMaxMsg* 参数中指定的值。 此参数可以为 NULL。  [out, optional] lpNextSize  下一条消息的大小（以字节为单位）。 以下值具有特殊含义。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MAILSLOT\_NO\_MESSAGE**  ( (DWORD) -1) | 没有下一条消息。 |     此参数可以为 NULL。  [out, optional] lpMessageCount  函数返回时等待读取的消息总数。 此参数可以为 NULL。  [out, optional] lpReadTimeout  读取操作在超时之前可以等待邮件写入邮件图的时间量（以毫秒为单位）。 当函数返回时，将填充此参数。 此参数可以为 NULL。 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。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateMailslot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota)  [Mailslot 函数](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslot-functions)  [Mailslots 概述](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslots)  [SetMailslotInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setmailslotinfo) |

## SetMailslotInfo 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 设置指定 mailslot 用于读取操作的超时值。 语法 C++复制  BOOL SetMailslotInfo(  [in] HANDLE hMailslot,  [in] DWORD lReadTimeout  ); 参数 [in] hMailslot  mailslot 的句柄。 [CreateMailslot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota) 函数必须创建此句柄。  [in] lReadTimeout  读取操作在超时之前可以等待邮件写入邮件图的时间（以毫秒为单位）。 以下值具有特殊含义。  展开表   |  |  | | --- | --- | | **值** | **含义** | | 0 | 如果没有消息，则立即返回 。 (系统不会将即时返回视为错误。) | | **MAILSLOT\_WAIT\_FOREVER**  ( (DWORD) -1) | 永远等待消息。 |     此超时值适用于所有后续读取操作和所有继承的邮件图句柄。 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。 注解 创建 mailslot 时，通常由 [CreateMailslot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota) 设置用于读取操作的初始超时值。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateMailslot](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createmailslota)  [GetMailslotInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getmailslotinfo)  [Mailslot 函数](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslot-functions)  [Mailslots 概述](https://learn.microsoft.com/zh-cn/windows/desktop/ipc/mailslots) |

# 写入 Mailslot

写入 mailslot 类似于写入标准磁盘文件。 以下代码使用 [**CreateFile**](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)、 [**WriteFile**](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 和 [**CloseHandle**](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数将短消息放入 mailslot。 邮件将广播到本地计算机上名为“sample\_mailslot”的邮件图服务器。 代码在已创建 mailslot 服务器的情况下运行。

C++复制

#include <windows.h>

#include <stdio.h>

LPCTSTR SlotName = TEXT("\\\\.\\mailslot\\sample\_mailslot");

BOOL WriteSlot(HANDLE hSlot, LPCTSTR lpszMessage)

{

BOOL fResult;

DWORD cbWritten;

fResult = WriteFile(hSlot,

lpszMessage,

(DWORD) (lstrlen(lpszMessage)+1)\*sizeof(TCHAR),

&cbWritten,

(LPOVERLAPPED) NULL);

if (!fResult)

{

printf("WriteFile failed with %d.\n", GetLastError());

return FALSE;

}

printf("Slot written to successfully.\n");

return TRUE;

}

int main()

{

HANDLE hFile;

hFile = CreateFile(SlotName,

GENERIC\_WRITE,

FILE\_SHARE\_READ,

(LPSECURITY\_ATTRIBUTES) NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

(HANDLE) NULL);

if (hFile == INVALID\_HANDLE\_VALUE)

{

printf("CreateFile failed with %d.\n", GetLastError());

return FALSE;

}

WriteSlot(hFile, TEXT("Message one for mailslot."));

WriteSlot(hFile, TEXT("Message two for mailslot."));

Sleep(5000);

WriteSlot(hFile, TEXT("Message three for mailslot."));

CloseHandle(hFile);

return TRUE;

}

# 从 Mailslot 读取

创建 mailslot 的过程可以在调用 [**ReadFile**](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 函数时使用 mailslot 句柄从中读取邮件。 以下示例调用 [**GetMailslotInfo**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winbase/nf-winbase-getmailslotinfo) 函数来确定 mailslot 中是否存在邮件。 如果消息正在等待，则会显示每个消息以及剩余要读取的消息数。

邮件图存在，直到为所有打开的服务器句柄调用 [**CloseHandle**](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数，或者直到拥有 mailslot 句柄的所有服务器进程退出为止。 在这两种情况下，将从 mailslot 中删除所有未读邮件，关闭 mailslot 的所有客户端句柄，并从内存中删除 mailslot 本身。

C++复制

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

#include <strsafe.h>

HANDLE hSlot;

LPCTSTR SlotName = TEXT("\\\\.\\mailslot\\sample\_mailslot");

BOOL ReadSlot()

{

DWORD cbMessage, cMessage, cbRead;

BOOL fResult;

LPTSTR lpszBuffer;

TCHAR achID[80];

DWORD cAllMessages;

HANDLE hEvent;

OVERLAPPED ov;

cbMessage = cMessage = cbRead = 0;

hEvent = CreateEvent(NULL, FALSE, FALSE, TEXT("ExampleSlot"));

if( NULL == hEvent )

return FALSE;

ov.Offset = 0;

ov.OffsetHigh = 0;

ov.hEvent = hEvent;

fResult = GetMailslotInfo( hSlot, // mailslot handle

(LPDWORD) NULL, // no maximum message size

&cbMessage, // size of next message

&cMessage, // number of messages

(LPDWORD) NULL); // no read time-out

if (!fResult)

{

printf("GetMailslotInfo failed with %d.\n", GetLastError());

return FALSE;

}

if (cbMessage == MAILSLOT\_NO\_MESSAGE)

{

printf("Waiting for a message...\n");

return TRUE;

}

cAllMessages = cMessage;

while (cMessage != 0) // retrieve all messages

{

// Create a message-number string.

StringCchPrintf((LPTSTR) achID,

80,

TEXT("\nMessage #%d of %d\n"),

cAllMessages - cMessage + 1,

cAllMessages);

// Allocate memory for the message.

lpszBuffer = (LPTSTR) GlobalAlloc(GPTR,

lstrlen((LPTSTR) achID)\*sizeof(TCHAR) + cbMessage);

if( NULL == lpszBuffer )

return FALSE;

lpszBuffer[0] = '\0';

fResult = ReadFile(hSlot,

lpszBuffer,

cbMessage,

&cbRead,

&ov);

if (!fResult)

{

printf("ReadFile failed with %d.\n", GetLastError());

GlobalFree((HGLOBAL) lpszBuffer);

return FALSE;

}

// Concatenate the message and the message-number string.

StringCbCat(lpszBuffer,

lstrlen((LPTSTR) achID)\*sizeof(TCHAR)+cbMessage,

(LPTSTR) achID);

// Display the message.

\_tprintf(TEXT("Contents of the mailslot: %s\n"), lpszBuffer);

GlobalFree((HGLOBAL) lpszBuffer);

fResult = GetMailslotInfo(hSlot, // mailslot handle

(LPDWORD) NULL, // no maximum message size

&cbMessage, // size of next message

&cMessage, // number of messages

(LPDWORD) NULL); // no read time-out

if (!fResult)

{

printf("GetMailslotInfo failed (%d)\n", GetLastError());

return FALSE;

}

}

CloseHandle(hEvent);

return TRUE;

}

BOOL WINAPI MakeSlot(LPCTSTR lpszSlotName)

{

hSlot = CreateMailslot(lpszSlotName,

0, // no maximum message size

MAILSLOT\_WAIT\_FOREVER, // no time-out for operations

(LPSECURITY\_ATTRIBUTES) NULL); // default security

if (hSlot == INVALID\_HANDLE\_VALUE)

{

printf("CreateMailslot failed with %d\n", GetLastError());

return FALSE;

}

return TRUE;

}

void main()

{

MakeSlot(SlotName);

while(TRUE)

{

ReadSlot();

Sleep(3000);

}

}

# 演练

## 1.新建一个文件夹Lesson71-proc-comm-mailslot,然后新建一个MFC客户端程序前面:MaiSlotServer,然后在这个解决方案里面添加一个MFC对话框项目,取名:MaiSlotClient,先把这两个项目中默认控件删除,然后重新PreTranslateMessage函数屏蔽回车键和esc键

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

## 2.先来做邮槽客户端布局,添加一个静态文本,内容为输入信息,然后添加一个编辑框控件和一个按钮,id和文本如图

|  |
| --- |
|  |
|  |

## 3.给编辑框绑定一个CEdit成员变量,方便判断内容为空时为编辑框获取输入焦点

|  |
| --- |
|  |
|  |

## 4.然后我们来给邮槽服务端做布局

|  |
| --- |
|  |
|  |

## 5.给这个按钮绑定一个成员变量

|  |
| --- |
|  |
|  |

## 6.双击按钮进入点击事件处理函数.我们来写代码,这个代码比较特别,会一直阻塞直到由数据

|  |
| --- |
| void CMailSlotServerDlg::OnBnClickedBtnRecv()  {  // TODO: 在此添加控件通知处理程序代码  m\_btn\_recv.EnableWindow(FALSE);//因为下面设置了邮槽会一直等待消息的到来,这里需要把按钮禁用;  m\_btn\_recv.SetWindowText(\_T("真正等等消息,请稍后"));//通知用户真正等待消息  HANDLE hmSlot;  hmSlot = CreateMailslot(\_T("\\\\.\\mailslot\\mymailslot"),0,MAILSLOT\_WAIT\_FOREVER,NULL);  if(INVALID\_HANDLE\_VALUE == hmSlot)  {  MessageBox(\_T("创建邮槽失败"));  m\_btn\_recv.EnableWindow(TRUE);  m\_btn\_recv.SetWindowText(\_T("接收邮槽数据"));//创建邮槽失败后还是要把按钮恢复原来的样子并且启用它  return;  }  //定义一个字符缓冲区来接收数据,大小不能超过424字节  TCHAR buf[420];  ZeroMemory(buf,420);  DWORD dwRead;  //读取信息到缓冲区  if(!ReadFile(hmSlot,buf,420,&dwRead,NULL))  {  MessageBox(\_T("读取邮槽信息失败"));  m\_btn\_recv.EnableWindow(TRUE);  m\_btn\_recv.SetWindowText(\_T("接收邮槽数据"));//接收数据失败后还是要把按钮恢复原来的样子并且启用它  CloseHandle(hmSlot);  return;  }  SetDlgItemText(IDC\_EDIT\_MSG,buf);  CloseHandle(hmSlot);  //接收并且显示完数据后启用按钮  m\_btn\_recv.EnableWindow(TRUE);  m\_btn\_recv.SetWindowText(\_T("接收邮槽数据"));//把文本设置为原来的文本  } |

### 测试一下,效果如图

|  |
| --- |
|  |

## 7.然后我们来给客户端也就是发送端的按钮添加点击事件处理函数,先添加一些长度控制代码,因为邮槽是由长度限制的,发送消息长度不能超过424字节

|  |
| --- |
| void CMailSlotClientDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  CString str;  GetDlgItemText(IDC\_EDIT\_MSG,str);  if(str.GetLength() == 0)//发送内容不能为空  {  MessageBox(\_T("请输入需要发送的内容!!"));  m\_edit\_msg.SetFocus();  return;  }  if(str.GetLength() >= 424)//发送内容不能超过424字节  {  MessageBox(\_T("发送的内容长度不能大于424字节!!"));  m\_edit\_msg.SetFocus();  return;  }  } |

### 编译运行测试一下,效果如下

|  |
| --- |
|  |

### 两个发送的文本太长,又会弹出消息框

|  |
| --- |
|  |

## 8.然后就可以编写正常发送的代码

|  |
| --- |
| void CMailSlotClientDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  HANDLE hmSlot;  CString str;  TCHAR buf[420];  DWORD dwWritten;  GetDlgItemText(IDC\_EDIT\_MSG,str);  if(str.GetLength() == 0)//发送内容不能为空  {  MessageBox(\_T("请输入需要发送的内容!!"));  m\_edit\_msg.SetFocus();  return;  }    //客户端在发送消息之前需要先打开邮槽,邮槽的名称必须和服务器创建的邮槽的名称一致  hmSlot = CreateFile(\_T("\\\\.\\mailslot\\mymailslot"),GENERIC\_WRITE,FILE\_SHARE\_READ,  NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(INVALID\_HANDLE\_VALUE == hmSlot)  {  MessageBox(\_T("打开邮槽失败!!"));  return;  }  //发送消息    ZeroMemory(buf,420);  GetDlgItemText(IDC\_EDIT\_MSG,buf,420);  if(!WriteFile(hmSlot,buf,420,&dwWritten,NULL))  {  MessageBox(\_T("写入邮槽失败!!"));  CloseHandle(hmSlot);  return;  }  CloseHandle(hmSlot);  } |

### 测试,重新生成所有项目,然后启动两个程序,先按下服务器的接收按钮,让服务器创建邮槽和等待消息,然后在客户端里面输入信息大家发送按钮,效果如下

|  |
| --- |
|  |

# 这一节的学习到此为止,完整代码如下,这里只给出对话框类的代码,因为应用程序类的代码基本不用写

## MailSlotClientDlg.h

|  |
| --- |
| // MailSlotClientDlg.h : 头文件  //  #pragma once  #include "afxwin.h"  // CMailSlotClientDlg 对话框  class CMailSlotClientDlg : public CDialogEx  {  // 构造  public:  CMailSlotClientDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_MAILSLOTCLIENT\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  // 生成的消息映射函数  virtual BOOL OnInitDialog();  afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);  afx\_msg void OnPaint();  afx\_msg HCURSOR OnQueryDragIcon();  DECLARE\_MESSAGE\_MAP()  public:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  // 绑定编辑框的成员变量  CEdit m\_edit\_msg;  afx\_msg void OnBnClickedBtnSend();  }; |

## MailSlotClientDlg.cpp

|  |
| --- |
| // MailSlotClientDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "MailSlotClient.h"  #include "MailSlotClientDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CMailSlotClientDlg 对话框  CMailSlotClientDlg::CMailSlotClientDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CMailSlotClientDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CMailSlotClientDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_EDIT\_MSG, m\_edit\_msg);  }  BEGIN\_MESSAGE\_MAP(CMailSlotClientDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BTN\_SEND, &CMailSlotClientDlg::OnBnClickedBtnSend)  END\_MESSAGE\_MAP()  // CMailSlotClientDlg 消息处理程序  BOOL CMailSlotClientDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CMailSlotClientDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CMailSlotClientDlg::OnPaint()  {  if (IsIconic())  {  CPaintDC dc(this); // 用于绘制的设备上下文  SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);  // 使图标在工作区矩形中居中  int cxIcon = GetSystemMetrics(SM\_CXICON);  int cyIcon = GetSystemMetrics(SM\_CYICON);  CRect rect;  GetClientRect(&rect);  int x = (rect.Width() - cxIcon + 1) / 2;  int y = (rect.Height() - cyIcon + 1) / 2;  // 绘制图标  dc.DrawIcon(x, y, m\_hIcon);  }  else  {  CDialogEx::OnPaint();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CMailSlotClientDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CMailSlotClientDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if(pMsg->message == WM\_KEYDOWN)  {  switch(pMsg->wParam)  {  case VK\_RETURN:  return TRUE;  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void CMailSlotClientDlg::OnBnClickedBtnSend()  {  // TODO: 在此添加控件通知处理程序代码  HANDLE hmSlot;  CString str;  TCHAR buf[420];  DWORD dwWritten;  GetDlgItemText(IDC\_EDIT\_MSG,str);  if(str.GetLength() == 0)//发送内容不能为空  {  MessageBox(\_T("请输入需要发送的内容!!"));  m\_edit\_msg.SetFocus();  return;  }    //客户端在发送消息之前需要先打开邮槽,邮槽的名称必须和服务器创建的邮槽的名称一致  hmSlot = CreateFile(\_T("\\\\.\\mailslot\\mymailslot"),GENERIC\_WRITE,FILE\_SHARE\_READ,  NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(INVALID\_HANDLE\_VALUE == hmSlot)  {  MessageBox(\_T("打开邮槽失败!!"));  return;  }  //发送消息    ZeroMemory(buf,420);  GetDlgItemText(IDC\_EDIT\_MSG,buf,420);  if(!WriteFile(hmSlot,buf,420,&dwWritten,NULL))  {  MessageBox(\_T("写入邮槽失败!!"));  CloseHandle(hmSlot);  return;  }  CloseHandle(hmSlot);  } |

## MailSlotServerDlg.h

|  |
| --- |
| // MailSlotServerDlg.h : 头文件  //  #pragma once  #include "afxwin.h"  // CMailSlotServerDlg 对话框  class CMailSlotServerDlg : public CDialogEx  {  // 构造  public:  CMailSlotServerDlg(CWnd\* pParent = NULL); // 标准构造函数  // 对话框数据  enum { IDD = IDD\_MAILSLOTSERVER\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  HICON m\_hIcon;  // 生成的消息映射函数  virtual BOOL OnInitDialog();  afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);  afx\_msg void OnPaint();  afx\_msg HCURSOR OnQueryDragIcon();  DECLARE\_MESSAGE\_MAP()  public:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  // 绑定按钮的成员变量  CButton m\_btn\_recv;  afx\_msg void OnBnClickedBtnRecv();  }; |

## MailSlotServerDlg.cpp

|  |
| --- |
| // MailSlotServerDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "MailSlotServer.h"  #include "MailSlotServerDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CMailSlotServerDlg 对话框  CMailSlotServerDlg::CMailSlotServerDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CMailSlotServerDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CMailSlotServerDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_BTN\_RECV, m\_btn\_recv);  }  BEGIN\_MESSAGE\_MAP(CMailSlotServerDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_BTN\_RECV, &CMailSlotServerDlg::OnBnClickedBtnRecv)  END\_MESSAGE\_MAP()  // CMailSlotServerDlg 消息处理程序  BOOL CMailSlotServerDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CMailSlotServerDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CMailSlotServerDlg::OnPaint()  {  if (IsIconic())  {  CPaintDC dc(this); // 用于绘制的设备上下文  SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);  // 使图标在工作区矩形中居中  int cxIcon = GetSystemMetrics(SM\_CXICON);  int cyIcon = GetSystemMetrics(SM\_CYICON);  CRect rect;  GetClientRect(&rect);  int x = (rect.Width() - cxIcon + 1) / 2;  int y = (rect.Height() - cyIcon + 1) / 2;  // 绘制图标  dc.DrawIcon(x, y, m\_hIcon);  }  else  {  CDialogEx::OnPaint();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CMailSlotServerDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CMailSlotServerDlg::PreTranslateMessage(MSG\* pMsg)  {  // TODO: 在此添加专用代码和/或调用基类  if(pMsg->message == WM\_KEYDOWN)  {  switch(pMsg->wParam)  {  case VK\_RETURN:  return TRUE;  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void CMailSlotServerDlg::OnBnClickedBtnRecv()  {  // TODO: 在此添加控件通知处理程序代码  m\_btn\_recv.EnableWindow(FALSE);//因为下面设置了邮槽会一直等待消息的到来,这里需要把按钮禁用;  m\_btn\_recv.SetWindowText(\_T("真正等等消息,请稍后"));//通知用户真正等待消息  HANDLE hmSlot;  hmSlot = CreateMailslot(\_T("\\\\.\\mailslot\\mymailslot"),0,MAILSLOT\_WAIT\_FOREVER,NULL);  if(INVALID\_HANDLE\_VALUE == hmSlot)  {  MessageBox(\_T("创建邮槽失败"));  m\_btn\_recv.EnableWindow(TRUE);  m\_btn\_recv.SetWindowText(\_T("接收邮槽数据"));//创建邮槽失败后还是要把按钮恢复原来的样子并且启用它  return;  }  //定义一个字符缓冲区来接收数据,大小不能超过424字节  TCHAR buf[420];  ZeroMemory(buf,420);  DWORD dwRead;  //读取信息到缓冲区  if(!ReadFile(hmSlot,buf,420,&dwRead,NULL))  {  MessageBox(\_T("读取邮槽信息失败"));  m\_btn\_recv.EnableWindow(TRUE);  m\_btn\_recv.SetWindowText(\_T("接收邮槽数据"));//接收数据失败后还是要把按钮恢复原来的样子并且启用它  CloseHandle(hmSlot);  return;  }  SetDlgItemText(IDC\_EDIT\_MSG,buf);  CloseHandle(hmSlot);  //接收并且显示完数据后启用按钮  m\_btn\_recv.EnableWindow(TRUE);  m\_btn\_recv.SetWindowText(\_T("接收邮槽数据"));//把文本设置为原来的文本  } |

# 扩展,思考一下能不能在服务器刚刚创建完成后就创建邮槽?

## 可能不行因为邮槽是阻塞的创建邮槽后点击按钮可能就不会有反应.

# 扩展2,可以尝试用邮槽实现微软里面2台主机之间的通信