# 这一节我们来学习文件操作是四种方法

|  |
| --- |
|  |

# api用法说明

# 1.>c语言文件函数

## fopen, \_wfopen

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开文件。 这些执行更多参数验证并返回错误代码的函数有更安全的版本可用；请参阅 [fopen\_s, \_wfopen\_s](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-s-wfopen-s?view=msvc-170)。 语法 C复制  FILE \*fopen(  const char \*filename,  const char \*mode  );  FILE \*\_wfopen(  const wchar\_t \*filename,  const wchar\_t \*mode  ); 参数 *filename* 文件名。  *mode* 启用的访问类型。 返回值 这些函数均返回指向打开文件的指针。 一个 null 指针值指示错误。 如果 *filename* 或 *mode* 是 NULL 或空字符串，这些函数则会触发无效的参数处理程序，如 [Parameter Validation](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)。 如果允许执行继续，则这些函数将返回 NULL 并将 errno 设置为 EINVAL。  有关详细信息，请参阅[errno, \_doserrno, \_sys\_errlist和\_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170). 备注 **fopen** 函数打开由 *filename* 指定的文件。 默认情况下，使用 ANSI 代码页 (CP\_ACP) 解释窄 *filename* 字符串。 在 Windows 桌面应用程序中，可以通过使用 [SetFileApisToOEM](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-setfileapistooem) 函数将它更改为 OEM 代码页 (CP\_OEMCP)。 可以使用 [AreFileApisANSI](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-arefileapisansi) 函数来确定是使用 ANSI 还是系统默认的 OEM 代码页来解释 *filename*。 **\_wfopen** 是 **fopen** 的宽字符版本；**\_wfopen** 参数是宽字符串。 除此以外， **\_wfopen** 和 **fopen** 的行为完全相同。 仅使用 **\_wfopen** 不会影响在文件流中使用的编码字符集。  **fopen** 接受执行时在文件系统上有效的路径； **fopen** 还接受 UNC 路径和包含映射的网络驱动器的路径（前提是执行代码的系统在执行时能够访问共享或映射的驱动器）。 为 **fopen** 构造路径时，请确保驱动器、路径或网络共享在执行环境中可用。 可使用斜杠 (/) 或反斜杠 (\) 作为路径中的目录分隔符。  对文件执行任何其他操作前，请始终检查返回值以确定指针是否为 NULL。 如果发生错误，系统将设置全局变量 errno，此变量可用于获取特定错误信息。 有关详细信息，请参阅[errno, \_doserrno, \_sys\_errlist和\_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170).  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此状态，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 Unicode 支持 **fopen** 支持 Unicode 文件流。 若要打开 Unicode 文件，请将指定所需编码的 ccs=encoding 标志传递到 **fopen**，如下所示。  FILE \*fp = fopen("newfile.txt", "rt+, ccs=UTF-8");  **ccs** 编码的允许值为 UNICODE、**UTF-8** 和 **UTF-16LE**。  在 Unicode 模式下打开文件时，输入函数会将从文件中读取的数据转换为存储为 **wchar\_t**类型的 UTF-16 数据。 写入到在 Unicode 模式下打开的文件的函数需要包含存储为 **wchar\_t**类型的 UTF-16 数据的缓冲区。 如果将文件编码为 UTF-8，则在写入它时，UTF-16 数据会转换为 UTF-8。 在读取它时，该文件的 UTF-8 编码的内容会转换为 UTF-16。 尝试在 Unicode 模式下读取或写入奇数个字节会导致 [参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170) 错误。 若要读取或写入在你的程序中存储为 UTF-8 的数据，请使用文本或二进制文件模式，而不是 Unicode 模式。 你应负责所有必需的编码转换。  如果文件已存在并已打开以进行读取或追加，则文件中的任何字节顺序标记 (BOM)将确定编码。 BOM 编码优先于由 **ccs** 标志指定的编码。 只有在没有 BOM 或文件是新文件时，才使用 **ccs** 编码。  **备注**  BOM 检测仅适用于在 Unicode 模式下（即通过传递 **ccs** 标志）打开的文件。  下表汇总的模式用于传递到文件中的 **ccs** 和字节顺序标记的各种 **fopen** 标志。 基于 ccs 标志和 BOM 使用的编码 展开表   | **ccs 标志** | **无 BOM（或新文件）** | **BOM：UTF-8** | **BOM：UTF-16** | | --- | --- | --- | --- | | UNICODE | **UTF-16LE** | **UTF-8** | **UTF-16LE** | | **UTF-8** | **UTF-8** | **UTF-8** | **UTF-16LE** | | **UTF-16LE** | **UTF-16LE** | **UTF-8** | **UTF-16LE** |   在 Unicode 模式下打开以进行写入的文件将自动写入 BOM。  如果 *mode* 为某 encoding 值的 **a, ccs=encoding**，**fopen** 将先尝试使用读取和写入访问权限打开文件。 如果此操作成功，此函数将读取 BOM 以确定文件的编码。 如果失败，此函数将使用文件的默认编码。 在任一情况下，**fopen** 使用仅写访问权限重新打开文件。 （此行为仅适用于 **"a"** 模式，不适用于 **"a+"** 模式。） 一般文本例程映射 展开表   | **TCHAR.H 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_tfopen** | **fopen** | **fopen** | **\_wfopen** |   字符串 *mode* 指定为文件请求的访问类型，如下所示。  展开表   | ***mode*** | **Access** | | --- | --- | | **"r"** | 打开以便读取。 如果文件不存在或找不到，**fopen** 调用将失败。 | | **"w"** | 打开用于写入的空文件。 如果给定文件存在，则其内容会被销毁。 | | **"a"** | 在文件末尾打开以进行写入（追加），在新数据写入到文件之前不移除文件末尾 (EOF) 标记。 如果文件不存在，则创建文件。 | | **"r+"** | 打开以便读取和写入。 文件必须存在。 | | **"w+"** | 打开用于读取和写入的空文件。 如果文件存在，则其内容会被销毁。 | | **"a+"** | 打开以进行读取和追加。 追加操作包括在新数据写入文件之前移除 EOF 标记。 写入完成后，EOF 标记不会还原。 如果文件不存在，则创建文件。 |   使用 **"a"** 访问类型或 **"a+"** 访问类型打开文件时，所有写入操作均将在文件末尾进行。 使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 或 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170)可重新定位文件指针，但在执行任何写入操作前，文件指针将始终被移回文件末尾。 因此，无法覆盖现有数据。  在 EOF 标记追加到文件之前，**"a"** 模式不会将其移除。 在追加后，MS-DOS TYPE 命令只显示原始 EOF 标记之前的数据，不显示追加到文件的任何数据。 EOF 标记追加到文件之前， **"a+"** 模式不会将其移除。 在追加后，MS-DOS TYPE 命令显示文件中的所有数据。 需使用 **"a+"** 模式才能附加到通过 CTRL+**Z** EOF 标记终止的流文件。  指定 **"r+"**、 **"w+"**或 **"a+"** 访问类型时，允许读取和写入（文件将处于打开状态以进行“更新”）。 但是，当你从读取切换到写入时，输入操作必须遇到 EOF 标记。 如果没有 EOF，必须使用对文件定位函数的干预调用。 文件定位函数是 **fsetpos**、 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170)和 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170)。 从写入切换到读取时，必须使用对 **fflush** 或文件定位函数的干预调用。  除了前面的值以外，可将以下字符追加到 *mode* 以指定换行符的转换模式。  展开表   | ***mode* 修饰符** | **转换模式** | | --- | --- | | **t** | 在文本（转换）模式下打开。 输入时，回车换行 (CR-LF) 组合将转换为单一的换行 (LF)；输出时，LF 字符将转换为 CR-LF 组合。 CTRL+Z 也将在输入时解释为文件尾字符。 | | **b** | 在二进制（未转换）模式下打开；禁止涉及回车和换行字符的转换。 |   在文本模式下， CTRL+**Z** 解释为输入上的 EOF 字符。 在使用 **"a+"**打开以供读取/写入的文件中，**fopen** 将检查文件末尾的 CTRL+**Z** ，并删除它（如果可能）。 将其删除是因为使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 和 **ftell** 在以 CTRL+**Z** 结尾的文件中移动时，可能导致 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 在文件末尾附近错误运行。  在文本模式下，输入时，回车-换行 (CRLF) 组合将转换为单一的换行 (LF) 字符；输出时，LF 字符将转换为 CRLF 组合。 当 Unicode 流 I/O 函数在文本模式（默认设置）下运行时，源或目标流将假定为一系列多字节字符。 因此，Unicode 流输入函数将多字节字符转换为宽字符（就像调用 **mbtowc** 函数一样）。 出于同一原因，Unicode 流输出函数将宽字符转换为多字节字符（就像调用 **wctomb** 函数一样）。  如果 **t** 或 **b** 在 *mode* 中未给出，则默认转换模式由全局变量 [\_fmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/fmode?view=msvc-170) 定义。 如果 **t** 或 **b** 是该参数的前缀，则函数将失败并返回 NULL。  有关如何在 Unicode 和多字节流 I/O 中使用文本和二进制模式的详细信息，请参阅 [文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170) 和 [文本和二进制模式下的 Unicode 流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/unicode-stream-i-o-in-text-and-binary-modes?view=msvc-170)。  可将以下选项追加到 *mode* 以指定更多行为。  展开表   | ***mode* 修饰符** | **行为** | | --- | --- | | **x** | 如果 *filename* 已存在，则强制函数失败。 只能与“w”或“w+”说明符一起使用。 | | **c** | 启用关联 *filename* 的提交标志，以便在调用 **fflush** 或 **\_flushall** 时将文件缓冲区的内容直接写入磁盘。 | | **n** | 将关联的 *filename* 的提交标志重置为“no-commit”。此标志为默认值。 如果将程序显式链接到 COMMODE.OBJ，它还将重写全局提交标志。 除非将程序显式链接到 COMMODE.OBJ，否则全局提交标志默认为 "no-commit"（请参阅[链接选项](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/link-options?view=msvc-170)）。 | | **N** | 指定文件不由子进程继承。 | | **S** | 指定缓存针对（但不限于）从磁盘的顺序访问进行优化。 | | **R** | 指定缓存针对（但不限于）从磁盘的随机访问进行优化。 | | **T** | 指定一个文件，除非内存压力需要它，否则不会写入磁盘。 | | **D** | 指定在关闭最后一个指向该文件的指针时删除的临时文件。 | | **ccs=encoding** | 指定要用于此文件的编码字符集（**UTF-8**、**UTF-16LE** 或 UNICODE）。 如果需要 ANSI 编码，请不要指定此字符集。 此标志与前面的标志用逗号 (,) 分隔。 例如：FILE \*f = fopen("newfile.txt", "rt+, ccs=UTF-8"); |   **fopen** 和 **\_fdopen** 中使用的 *mode* 字符串的有效字符与 [\_open](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) 和 [\_sopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sopen-wsopen?view=msvc-170) 中使用的 *oflag* 参数对应，如下所示。  展开表   | **字符串中的 *mode* 字符** | **\_open/\_sopen 的等效 *oflag* 值** | | --- | --- | | **a** | \_O\_WRONLY | \_O\_APPEND （通常为 \_O\_WRONLY | \_O\_CREAT | \_O\_APPEND ） | | **a+** | \_O\_RDWR | \_O\_APPEND （通常为 \_O\_RDWR | \_O\_APPEND | \_O\_CREAT ） | | **r** | \_O\_RDONLY | | **r+** | \_O\_RDWR | | **w** | \_O\_WRONLY （通常为 \_O\_WRONLY | \_O\_CREAT | \_O\_TRUNC ） | | **w+** | \_O\_RDWR （通常为 \_O\_RDWR | \_O\_CREAT | \_O\_TRUNC ） | | **b** | \_O\_BINARY | | **t** | \_O\_TEXT（翻译） | | **x** | \_O\_EXCL | | **c** | 无 | | **n** | 无 | | **S** | \_O\_SEQUENTIAL | | **R** | \_O\_RANDOM | | **T** | \_O\_SHORTLIVED | | **D** | \_O\_TEMPORARY | | **ccs=UNICODE** | \_O\_WTEXT | | \**ccs=UTF-8\** | \_O\_UTF8 | | **ccs=UTF-16LE** | \_O\_UTF16 |   如果你使用 **rb** 模式，不必移植代码，而如果希望读取大文件中的大部分内容或不担心网络性能，你可能还要考虑是否使用内存映射的 Win32 文件方式。  关于 T 和 D：   * 只要内存压力不需要，T 就避免将文件写入磁盘。 有关详细信息，请参阅[文件属性常量](https://learn.microsoft.com/zh-cn/windows/win32/fileio/file-attribute-constants)中的 FILE\_ATTRIBUTE\_TEMPORARY，以及此博客文章[它只是暂时性的](https://learn.microsoft.com/archive/blogs/larryosterman/its-only-temporary)。 * D 指定写入磁盘的常规文件。 区别在于它在关闭时会自动删除。 可以组合 TD 来获取这两种语义。   **c**、、**nR**、**S**、**t**和 **T***mode* **D**选项Microsoft扩展fopen\_wfopen，并且不应在需要 ANSI 可移植性时使用。 要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fopen** | <stdio.h> | | **\_wfopen** | <stdio.h> 或 <wchar.h> |   **\_wfopen** 是 Microsoft 扩展。 有关兼容性的更多信息，请参见 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。  **c**、**n**、**t**、**S**、**R**和 **T***mode* **D**选项Microsoft **fopen** **\_fdopen**扩展，并且不应在需要 ANSI 可移植性的情况下使用。 示例 1 以下程序打开两个文件。 它使用 **fclose** 关闭第一个文件，使用 **\_fcloseall** 关闭所有剩余文件。  C复制  // crt\_fopen.c  // compile with: /W3  // This program opens two files. It uses  // fclose to close the first file and  // \_fcloseall to close all remaining files.  #include <stdio.h>  FILE \*stream, \*stream2;  int main( void )  {  int numclosed;  // Open for read (will fail if file "crt\_fopen.c" does not exist)  if( (stream = fopen( "crt\_fopen.c", "r" )) == NULL ) // C4996  // Note: fopen is deprecated; consider using fopen\_s instead  printf( "The file 'crt\_fopen.c' was not opened\n" );  else  printf( "The file 'crt\_fopen.c' was opened\n" );  // Open for write  if( (stream2 = fopen( "data2", "w+" )) == NULL ) // C4996  printf( "The file 'data2' was not opened\n" );  else  printf( "The file 'data2' was opened\n" );  // Close stream if it is not NULL  if( stream)  {  if ( fclose( stream ) )  {  printf( "The file 'crt\_fopen.c' was not closed\n" );  }  }  // All other files are closed:  numclosed = \_fcloseall( );  printf( "Number of files closed by \_fcloseall: %u\n", numclosed );  }  Output复制  The file 'crt\_fopen.c' was opened  The file 'data2' was opened  Number of files closed by \_fcloseall: 1 示例 2 以下程序在具有 Unicode 编码的文本模式下创建文件（或在文件存在时覆盖文件）。 然后，它将两个字符串写入文件并关闭文件。 输出是名为 *\_wfopen\_test.xml* 的文件，其中包含输出部分中的数据。  C复制  // crt\_\_wfopen.c  // compile with: /W3  // This program creates a file (or overwrites one if  // it exists), in text mode using Unicode encoding.  // It then writes two strings into the file  // and then closes the file.  #include <stdio.h>  #include <stddef.h>  #include <stdlib.h>  #include <wchar.h>  #define BUFFER\_SIZE 50  int main(int argc, char\*\* argv)  {  wchar\_t str[BUFFER\_SIZE];  size\_t strSize;  FILE\* fileHandle;  // Create an the xml file in text and Unicode encoding mode.  if ((fileHandle = \_wfopen( L"\_wfopen\_test.xml",L"wt+,ccs=UNICODE")) == NULL) // C4996  // Note: \_wfopen is deprecated; consider using \_wfopen\_s instead  {  wprintf(L"\_wfopen failed!\n");  return(0);  }  // Write a string into the file.  wcscpy\_s(str, sizeof(str)/sizeof(wchar\_t), L"<xmlTag>\n");  strSize = wcslen(str);  if (fwrite(str, sizeof(wchar\_t), strSize, fileHandle) != strSize)  {  wprintf(L"fwrite failed!\n");  }  // Write a string into the file.  wcscpy\_s(str, sizeof(str)/sizeof(wchar\_t), L"</xmlTag>");  strSize = wcslen(str);  if (fwrite(str, sizeof(wchar\_t), strSize, fileHandle) != strSize)  {  wprintf(L"fwrite failed!\n");  }  // Close the file.  if (fclose(fileHandle))  {  wprintf(L"fclose failed!\n");  }  return 0;  } 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [多字节字符序列的解释](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/interpretation-of-multibyte-character-sequences?view=msvc-170) [fclose, \_fcloseall](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fclose-fcloseall?view=msvc-170) [\_fdopen, \_wfdopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fdopen-wfdopen?view=msvc-170) [ferror](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ferror?view=msvc-170) [\_fileno](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fileno?view=msvc-170) [freopen, \_wfreopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/freopen-wfreopen?view=msvc-170) [\_open, \_wopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) [\_setmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/setmode?view=msvc-170) [\_sopen, \_wsopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sopen-wsopen?view=msvc-170) |

## fopen\_s, \_wfopen\_s

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开文件。 这些版本的 [fopen、\_wfopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) 具有安全性增强功能，如 [CRT 中的安全性功能](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/security-features-in-the-crt?view=msvc-170)中所述。 语法 C复制  errno\_t fopen\_s(  FILE\*\* pFile,  const char \*filename,  const char \*mode  );  errno\_t \_wfopen\_s(  FILE\*\* pFile,  const wchar\_t \*filename,  const wchar\_t \*mode  ); 参数 *pFile* 指向文件指针的指针，该指针接收指向打开的文件的指针。  *filename* 要打开的文件的名称。  *mode* 允许的访问类型。 返回值 如果成功，则为零；如果失败，则为错误代码。 有关这些错误代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 错误条件 展开表   | ***pFile*** | ***filename*** | ***mode*** | **返回值** | ***pFile* 的内容** | | --- | --- | --- | --- | --- | | NULL | any | 任意 | EINVAL | 未更改 | | any | NULL | 任意 | EINVAL | 未更改 | | any | 任意 | NULL | EINVAL | 未更改 |  备注 **fopen\_s** 和 **\_wfopen\_s** 函数无法打开用于共享的文件。 如果需要共享文件，请将 [\_fsopen 或 \_wfsopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fsopen-wfsopen?view=msvc-170) 与适当的共享模式常量配合使用（例如，将 \_SH\_DENYNO 用于读/写共享）。  **fopen\_s** 函数打开由 *filename* 指定的文件。 **\_wfopen\_s** 是 **fopen\_s** 的宽字符版本，**\_wfopen\_s** 的参数是宽字符字符串。 除此以外，**\_wfopen\_s** 和 **fopen\_s** 的行为完全相同。  **fopen\_s** 接受执行时文件系统上有效的路径；**fopen\_s** 接受 UNC 路径以及包含所映射网络驱动器的路径，前提是执行代码的系统在执行时能够访问共享或映射的网络驱动器。 为 **fopen\_s** 构造路径时，请勿做出有关驱动器、路径或执行环境中网络共享的可用性假设。 可使用斜杠 (/) 或反斜杠 (\) 作为路径中的目录分隔符。  这些函数验证其参数。 如果 *pFile*、*filename* 或 *mode* 为 null 指针，则这些函数将生成无效的参数异常，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。  请始终先检查返回值，确定该函数是否成功，再对文件执行任何进一步的操作。 如果出现错误，则将返回错误代码并且将设置全局变量 errno。 有关详细信息，请参阅[errno, \_doserrno, \_sys\_errlist和\_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170).  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此状态，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 Unicode 支持 **fopen\_s** 支持 Unicode 文件流。 若要打开新的或现有的 Unicode 文件，请将指定所需编码的 **ccs** 标志传递到 **fopen\_s**，例如：  **fopen\_s(&fp, "newfile.txt", "w+, ccs=utf-8");**  **ccs** 标志的允许值为 UNICODE、**UTF-8** 和 **UTF-16LE**。 如果未为 **ccs** 指定值，则 **fopen\_s** 将使用 ANSI 编码。  如果文件已存在并已打开以进行读取或追加，则字节顺序标记 (BOM)（如果文件中存在）将确定编码。 BOM 编码优先于 **ccs** 标志指定的编码。 仅在不存在 BOM 或如果文件是新文件时，才使用 **ccs** 编码。  **备注**  BOM 检测仅适用于在 Unicode 模式下打开的文件；即，通过传递 **ccs** 标志打开的文件。  下表汇总了提供给 **fopen\_s** 的各种 **ccs** 标志值的模式以及文件中的 BOM 的模式。 基于 ccs 标志和 BOM 使用的编码 展开表   | **ccs 标志** | **无 BOM（或新文件）** | **BOM：UTF-8** | **BOM：UTF-16** | | --- | --- | --- | --- | | UNICODE | **UTF-8** | **UTF-8** | **UTF-16LE** | | **UTF-8** | **UTF-8** | **UTF-8** | **UTF-16LE** | | **UTF-16LE** | **UTF-16LE** | **UTF-8** | **UTF-16LE** |   在 Unicode 模式下打开用于写入的文件将自动在其中写入 BOM。  如果 *mode* 为 **"a, ccs=UNICODE"**、**"a, ccs=UTF-8"** 或 **"a, ccs=UTF-16LE"**，**fopen\_s** 将先尝试使用读取和写入访问权限打开文件。 如果成功，此函数将读取 BOM 以确定文件的编码；如果失败，此函数将使用文件的默认编码。 在任一情况下，**fopen\_s** 随后均将使用只写访问权限重新打开文件。 （此行为仅适用于 **a** 模式，不适用于 **a+** 模式。）  字符串 *mode* 指定对文件请求的访问类型，如下所示。  展开表   | ***mode*** | **Access** | | --- | --- | | **"r"** | 打开以便读取。 如果文件不存在或找不到，**fopen\_s** 调用将失败。 | | **"w"** | 打开用于写入的空文件。 如果给定文件存在，则其内容会被销毁。 | | **"a"** | 在文件末尾打开以进行写入（追加），在新数据写入到文件之前不移除文件末尾 (EOF) 标记。 如果文件不存在，则创建文件。 | | **"r+"** | 打开以便读取和写入。 文件必须存在。 | | **"w+"** | 打开用于读取和写入的空文件。 如果文件存在，则其内容会被销毁。 | | **"a+"** | 打开以进行读取和追加。 追加操作包括在新数据写入文件之前移除 EOF 标记。 写入完成后，EOF 标记不会还原。 如果文件不存在，则创建文件。 |   通过使用 **"a"** 或 **"a+"** 访问类型打开文件时，所有写入操作均将在文件末尾进行。 使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 或 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170) 可重新定位文件指针，但在执行任何写入操作前，文件指针将始终被移回文件末尾，以确保不会覆盖现有数据。  在 EOF 标记追加到文件之前，**"a"** 模式会将其删除。 追加后，MS-DOS TYPE 命令仅显示原始 EOF 标记之前的数据，不显示追加到文件的任何数据。 在 EOF 标记追加到文件之前，**"a+"** 模式会将其删除。 在追加后，MS-DOS TYPE 命令显示文件中的所有数据。 需使用 **"a+"** 模式才能附加到通过 CTRL+**Z** EOF 标记终止的流文件。  当指定 **"r+"**、**"w+"** 或 **"a+"** 访问类型时，允许读取和写入。 （文件据说已经可以进行“更新”。）但是，当你从读取切换到写入时，输入操作必定会遇到 EOF 标记。 如果没有 EOF 标记，必须使用对文件定位函数的干预调用。 文件定位函数是 **fsetpos**、[fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 和 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170)。 从写入切换到读取时，必须使用对 **fflush** 或文件定位函数的干预调用。  从 C11 开始，可以将 **"x"** 追加到 **"w"**，或追加 **"w+"** 以使函数在该文件存在时失败，而不是覆盖该文件。  除了上述值之外，可以在 *mode* 中包含以下字符以指定换行符的转换模式：  展开表   | ***mode* 修饰符** | **转换模式** | | --- | --- | | **t** | 在文本（转换）模式下打开。 输入时，回车换行 (CR-LF) 组合将转换为单一的换行 (LF)；输出时，LF 字符将转换为 CR-LF 组合。 CTRL+Z 将在输入时解释为文件尾字符。 | | **b** | 在二进制（未转换）模式下打开；禁止涉及回车和换行字符的转换。 |   在文本（已转换）模式中，CTRL+**Z** 将在输入时解释为文件结尾字符。 在打开使用 **"a+"** 进行读取/写入的文件中，**fopen\_s** 将检查文件末尾的 CTRL+**Z** 并在可能的情况下将其删除。 将其删除是因为使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 和 [ftell](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 在以 CTRL+**Z** 结尾的文件中移动时，可能导致 **fseek** 在文件末尾附近错误运行。  另外，在文本模式下，输入时，回车/换行 (CRLF) 组合将转换为单一的换行 (LF) 字符；输出时，LF 字符将转换为 CRLF 组合。 当 Unicode 流 I/O 函数在文本模式（默认设置）下运行时，源或目标流将假定为一系列多字节字符。 Unicode 流输入函数将多字节字符转换为宽字符（就像调用 **mbtowc** 函数一样）。 出于同一原因，Unicode 流输出函数将宽字符转换为多字节字符（就像调用 **wctomb** 函数一样）。  如果 **t** 或 **b** 在 *mode* 中未给出，则默认转换模式由全局变量 [\_fmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/fmode?view=msvc-170) 定义。 如果 **t** 或 **b** 是该参数的前缀，则函数将失败并返回 NULL。  有关在 Unicode 和多字节流 I/O 中使用文本和二进制模式的详细信息，请参阅[文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170) 和[文本和二进制模式下的 Unicode 流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/unicode-stream-i-o-in-text-and-binary-modes?view=msvc-170)。  展开表   | ***mode* 修饰符** | **行为** | | --- | --- | | **c** | 启用关联 *filename* 的提交标志，以便在调用 **fflush** 或 **\_flushall** 时将文件缓冲区的内容直接写入磁盘。 | | **n** | 将关联的 *filename* 的提交标志重置为“no-commit”。此标志为默认值。 如果将程序显式链接到 *COMMODE.OBJ*，它还将重写全局提交标志。 除非将程序显式链接到 *COMMODE.OBJ*，否则全局提交标志默认为 "no-commit"（请参阅[链接选项](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/link-options?view=msvc-170)）。 | | **N** | 指定文件不由子进程继承。 | | **S** | 指定缓存针对（但不限于）从磁盘的顺序访问进行优化。 | | **R** | 指定缓存针对（但不限于）从磁盘的随机访问进行优化。 | | **T** | 指定一个文件，除非内存压力需要它，否则不会写入磁盘。 | | **D** | 指定在关闭最后一个指向该文件的指针时删除的临时文件。 | | **ccs=UNICODE** | 指定 UNICODE 作为要用于此文件的编码字符集。 如果需要 ANSI 编码，请不要指定此字符集。 | | **ccs=UTF-8** | 指定 UTF-8 作为要用于此文件的编码字符集。 如果需要 ANSI 编码，请不要指定此字符集。 | | **ccs=UTF-16LE** | 指定 UTF-16LE 作为要用于此文件的编码字符集。 如果需要 ANSI 编码，请不要指定此字符集。 |   在 **fopen\_s** 和 [\_fdopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fdopen-wfdopen?view=msvc-170) 中使用的 *mode* 字符串的有效字符对应于在 [\_open](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) 和 [\_sopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sopen-wsopen?view=msvc-170) 中使用的 *oflag* 参数，如下所示。  展开表   | **字符串中的 *mode* 字符** | **\_open/\_sopen 的等效 *oflag* 值** | | --- | --- | | **a** | \_O\_WRONLY | \_O\_APPEND （通常为 \_O\_WRONLY | \_O\_CREAT | \_O\_APPEND ） | | **a+** | \_O\_RDWR | \_O\_APPEND （通常为 \_O\_RDWR | \_O\_APPEND | \_O\_CREAT ） | | **R** | \_O\_RDONLY | | **r+** | \_O\_RDWR | | **w** | \_O\_WRONLY （通常为 \_O\_WRONLY | \_O\_CREAT | \_O\_TRUNC ） | | **w+** | \_O\_RDWR（通常为 \*\*\_O\_RDWR | \_O\_CREAT | \_O\_TRUNC） | | **b** | \_O\_BINARY | | **t** | \_O\_TEXT（翻译） | | **c** | 无 | | **n** | 无 | | **D** | \_O\_TEMPORARY | | **R** | \_O\_RANDOM | | **S** | \_O\_SEQUENTIAL | | **T** | \_O\_SHORTLIVED | | **ccs=UNICODE** | \_O\_WTEXT | | **ccs=UTF-8** | \_O\_UTF8 | | **ccs=UTF-16LE** | \_O\_UTF16 |   **c**、、**nR**、**S**、**t**和 **T***mode* **D**选项Microsoft扩展fopen\_s\_wfopen\_s，并且不应在需要 ANSI 可移植性时使用。  如果使用 **rb** 模式，并且无需移植代码、预计将读取大量文件或者不关心网络性能，则内存映射的 Win32 文件可能也是一个选项。  关于 T 和 D：   * 只要内存压力不需要，T 就避免将文件写入磁盘。 有关详细信息，请参阅[文件属性常量](https://learn.microsoft.com/zh-cn/windows/win32/fileio/file-attribute-constants)中的 FILE\_ATTRIBUTE\_TEMPORARY，以及此博客文章[它只是暂时性的](https://learn.microsoft.com/archive/blogs/larryosterman/its-only-temporary)。 * D 指定写入磁盘的常规文件。 区别在于它在关闭时会自动删除。 可以组合 TD 来获取这两种语义。  要求 展开表   | **函数** | **必需的标头** | **C++ 标头** | | --- | --- | --- | | **fopen\_s** | <stdio.h> | <cstdio> | | **\_wfopen\_s** | <stdio.h> 或 <wchar.h> | <cstdio> |   有关 C 运行时库中的标准符合性和命名约定的详细信息，请参阅[兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 一般文本例程映射 展开表   | **<tchar.h> 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_tfopen\_s** | **fopen\_s** | **fopen\_s** | **\_wfopen\_s** |  库 [C 运行时库](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/crt-library-features?view=msvc-170)的所有版本。 示例 C复制  // crt\_fopen\_s.c  // This program opens two files. It uses  // fclose to close the first file and  // \_fcloseall to close all remaining files.  #include <stdio.h>  FILE \*stream, \*stream2;  int main( void )  {  errno\_t err;  // Open for read (will fail if file "crt\_fopen\_s.c" doesn't exist)  err = fopen\_s( &stream, "crt\_fopen\_s.c", "r" );  if( err == 0 )  {  printf( "The file 'crt\_fopen\_s.c' was opened\n" );  }  else  {  printf( "The file 'crt\_fopen\_s.c' was not opened\n" );  }  // Open for write  err = fopen\_s( &stream2, "data2", "w+, ccs=UTF-8" );  if( err == 0 )  {  printf( "The file 'data2' was opened\n" );  }  else  {  printf( "The file 'data2' was not opened\n" );  }  // Close stream if it isn't NULL  if( stream )  {  err = fclose( stream );  if ( err == 0 )  {  printf( "The file 'crt\_fopen\_s.c' was closed\n" );  }  else  {  printf( "The file 'crt\_fopen\_s.c' was not closed\n" );  }  }  // All other files are closed:  int numclosed = \_fcloseall( );  printf( "Number of files closed by \_fcloseall: %u\n", numclosed );  }  Output复制  The file 'crt\_fopen\_s.c' was opened  The file 'data2' was opened  Number of files closed by \_fcloseall: 1 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [fclose, \_fcloseall](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fclose-fcloseall?view=msvc-170) [\_fdopen, \_wfdopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fdopen-wfdopen?view=msvc-170) [ferror](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ferror?view=msvc-170) [\_fileno](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fileno?view=msvc-170) [freopen, \_wfreopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/freopen-wfreopen?view=msvc-170) [\_open, \_wopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) [\_setmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/setmode?view=msvc-170) |

## \_sopen\_s, \_wsopen\_s

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开文件以供共享。 这些版本的 [\_sopen 和 \_wsopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sopen-wsopen?view=msvc-170) 具有安全增强功能，如 [CRT 中的安全功能](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/security-features-in-the-crt?view=msvc-170)中所述。 语法 C复制  errno\_t \_sopen\_s(  int\* pfh,  const char \*filename,  int oflag,  int shflag,  int pmode  );  errno\_t \_wsopen\_s(  int\* pfh,  const wchar\_t \*filename,  int oflag,  int shflag,  int pmode,  ); 参数 *pfh* 文件句柄或 -1（如果出现错误）。  *filename* 文件名。  *oflag* 允许的操作类型。  *shflag* 允许的共享类型。  *pmode* 权限设置。 返回值 非零返回值指示错误；在这种情况下，请将 errno 设置为以下值之一。  展开表   | **errno 值** | **条件** | | --- | --- | | EACCES | 给定路径是目录，或者文件是只读的，但是已尝试打开以供写入操作。 | | EEXIST | 已指定 \_O\_CREAT 和 \_O\_EXCL 标志，但 *filename* 已经存在。 | | EINVAL | *oflag*、*shflag* 或 *pmode* 参数无效，或者 *pfh* 或 *filename* 是空指针。 | | EMFILE | 没有更多可用的文件描述符。 | | ENOENT | 未找到文件或路径。 |   如果将无效参数传递到该函数，则调用无效参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许执行继续，会将 errno 设置为 EINVAL 并返回 EINVAL。  有关这些和其他的返回代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。  如果出现错误，则将通过 *pfh* 返回 -1（除非 *pfh* 是空指针）。 备注 **\_sopen\_s** 函数将打开由 *filename* 指定的文件并使该文件做好共享的读写准备，如 *oflag* 和 *shflag* 所定义。 **\_wsopen\_s** 是 **\_sopen\_s**的宽字符版本； *filename* 的 **\_wsopen\_s** 参数是宽字符字符串。 除此以外，**\_wsopen\_s** 和 **\_sopen\_s** 的行为完全相同。  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此状态，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 一般文本例程映射 展开表   | **Tchar.h 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_tsopen\_s** | **\_sopen\_s** | **\_sopen\_s** | **\_wsopen\_s** |   整数表达式 *oflag* 是通过合并在 <fcntl.h> 中定义的一个或多个清单常量而形成的。 当两个或多个常量构成 *oflag* 参数时，这些常量将与按位 OR 运算符合并 ( **|** )。  展开表   | ***oflag* 常数** | **行为** | | --- | --- | | \_O\_APPEND | 在执行每个写入操作之前，将文件指针移动到文件末尾。 | | \_O\_BINARY | 在二进制（未转换）模式下打开该文件。 （有关二进制模式的说明，请参阅 [fopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170)。） | | \_O\_CREAT | 创建文件并打开它以供写入。 如果由 *filename* 指定的文件存在，则不会产生任何影响。 指定 *pmode* 时，需要使用 \_O\_CREAT 参数。 | | **\_O\_CREAT | \_O\_SHORT\_LIVED** | 创建一个文件作为临时文件，如果可能，请不要将它刷新到磁盘中。 指定 *pmode* 时，需要使用 \_O\_CREAT 参数。 | | **\_O\_CREAT | \_O\_TEMPORARY** | 创建一个文件作为临时文件；在关闭最后一个文件描述符时，删除该文件。 指定 *pmode* 时，需要使用 \_O\_CREAT 参数。 为了保留旧行为以保持应用兼容性，不会阻止其他进程删除此文件。 | | **\_O\_CREAT | \_O\_EXCL** | 如果由 *filename* 指定的文件存在，则返回一个错误值。 仅在与 \_O\_CREAT 一起使用时应用。 | | \_O\_NOINHERIT | 阻止创建共享文件描述符。 | | \_O\_RANDOM | 指定缓存针对（但不限于）从磁盘的随机访问进行优化。 | | \_O\_RDONLY | 打开文件以供只读。 不能使用 \_O\_RDWR 或 \_O\_WRONLY 指定。 | | \_O\_RDWR | 打开文件以供读取和写入。 不能使用 \_O\_RDONLY 或 \_O\_WRONLY 指定。 | | \_O\_SEQUENTIAL | 指定缓存针对（但不限于）从磁盘的顺序访问进行优化。 | | \_O\_TEXT | 在 ANSI 文本（已翻译）模式下打开文件。 （有关详细信息，请参阅[文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170) 和 [fopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170)。） | | \_O\_TRUNC | 打开文件并将其长度截断为零；该文件必须具有写入权限。 不能使用 \_O\_RDONLY 指定。 结合使用 \_O\_TRUNC 和 \_O\_CREAT 来打开现有文件或创建文件。 注意：\_O\_TRUNC 标志会损坏指定文件的内容。 | | \_O\_WRONLY | 打开文件以供只写。 不能使用 \_O\_RDONLY 或 \_O\_RDWR 指定。 | | \_O\_U16TEXT | 在 Unicode UTF-16 模式下打开文件。 | | \_O\_U8TEXT | 在 Unicode UTF-8 模式下打开文件。 | | \_O\_WTEXT | 在 Unicode 模式下打开文件。 |   若要指定文件访问模式，你必须指定 \_O\_RDONLY、\_O\_RDWR 或 \_O\_WRONLY。 对于访问模式，不存在默认值。  使用 \_O\_WTEXT、\_O\_U8TEXT 或 \_O\_U16TEXT 在 Unicode 模式下打开文件时，输入函数会将从该文件中读取的数据转换为存储为 **wchar\_t** 类型的 UTF-16 数据。 写入到在 Unicode 模式下打开的文件的函数需要包含存储为 **wchar\_t**类型的 UTF-16 数据的缓冲区。 如果将文件编码为 UTF-8，则在写入它时，UTF-16 数据会转换为 UTF-8。 在读取它时，该文件的 UTF-8 编码的内容会转换为 UTF-16。 尝试在 Unicode 模式下读取或写入奇数个字节会导致 参数验证 错误。 若要读取或写入在你的程序中存储为 UTF-8 的数据，请使用文本或二进制文件模式，而不是 Unicode 模式。 你应负责所有必需的编码转换。  如果使用 **\_O\_WRONLY | \_O\_APPEND**（追加模式）和 \_O\_WTEXT、\_O\_U16TEXT 或 \_O\_U8TEXT 调用 **\_sopen\_s**，则它首先会尝试打开该文件以供读取和写入、读取 BOM，然后重新打开它以供只写。 如果无法打开该文件以供读取和写入，则它将打开该文件以供只写，并使用 Unicode 模式设置的默认值。  *shflag* 参数是常量表达式，它包含在 <share.h> 中定义的以下清单常量之一。  展开表   | ***shflag* 常数** | **行为** | | --- | --- | | \_SH\_DENYRW | 拒绝对文件的读取和写入访问。 | | \_SH\_DENYWR | 拒绝对文件的写入访问。 | | \_SH\_DENYRD | 拒绝对文件的读取访问。 | | \_SH\_DENYNO | 允许读取和写入访问。 |   不同于 *pmode*，始终需要 **\_sopen** 参数。 指定 \_O\_CREAT 时，如果该文件不存在，则 *pmode* 将指定在首次关闭新文件时设置的该文件的权限设置。 否则，将忽略 *pmode*。 *pmode* 是一个整数表达式，它包含在 <sys\stat.h> 中定义的以下 \_S\_IWRITE 和 \_S\_IREAD 常量之一或两个。 当给定这两个常量时，将使用按位“或”运算符合并它们。 *pmode* 的含义如下。  展开表   | ***pmode*** | **含义** | | --- | --- | | \_S\_IREAD | 只允许读取。 | | \_S\_IWRITE | 允许写入。 （实际上，允许读取和写入。） | | **\_S\_IREAD | \_S\_IWRITE** | 允许读取和写入。 |   如果未授予写入权限，则该文件为只读。 在 Windows 操作系统中，所有文件均可读；不可能提供只写权限。 因此，模式 \_S\_IWRITE 和 **\_S\_IREAD | \_S\_IWRITE** 是等效的。  在设置这些权限之前，**\_sopen\_s** 会将当前文件权限掩码应用到 *pmode*。 （请参阅[\_umask](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/umask?view=msvc-170).) 要求 展开表   | **函数** | **必需的标头** | **可选标头** | | --- | --- | --- | | **\_sopen\_s** | <io.h> | <fcntl.h>、、<sys\types.h><sys\stat.h>、、<share.h> | | **\_wsopen\_s** | <io.h> 或 <wchar.h> | <fcntl.h>、、<sys/types.h><sys/stat.h>、、<share.h> |   **\_sopen\_s** 和 **\_wsopen\_s** 是 Microsoft 扩展。 有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 请参阅 [\_locking](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/locking?view=msvc-170) 的示例。 另请参阅 [低级别 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/low-level-i-o?view=msvc-170) [\_close](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/close?view=msvc-170) [\_creat, \_wcreat](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/creat-wcreat?view=msvc-170) [fopen, \_wfopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) [\_fsopen, \_wfsopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fsopen-wfsopen?view=msvc-170) [\_open, \_wopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/open-wopen?view=msvc-170) |

## \_fsopen、\_wfsopen

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开具有文件共享的流。 语法 C复制  FILE \*\_fsopen(  const char \*filename,  const char \*mode,  int shflag  );  FILE \*\_wfsopen(  const wchar\_t \*filename,  const wchar\_t \*mode,  int shflag  ); 参数 *filename* 要打开的文件的名称。  *mode* 允许的访问类型。  *shflag* 允许的共享类型。 返回值 这些函数均返回指向流的指针。 一个 null 指针值指示错误。 如果 *filename* 或 *mode* 是 NULL 指针或空字符串，这些函数将调用无效参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许执行继续，则这些函数将返回 NULL 并将 errno 设置为 EINVAL。  有关这些和其他错误代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 备注 **\_fsopen** 函数以流的形式打开由 *filename* 指定的文件并使该文件做好准备以进行后续的共享读写，如模式和 *shflag* 参数所定义。 **\_wfsopen** 是 **\_fsopen** 的宽字符版本；**\_wfsopen** 的 *filename* 和 *mode* 参数是宽字符串。 除此以外，**\_wfsopen** 和 **\_fsopen** 的行为完全相同。  字符串 *mode* 指定为文件请求的访问类型，如下表所示。  展开表   | **术语** | **定义** | | --- | --- | | :" | 打开以便读取。 如果文件不存在或找不到，**\_fsopen** 调用将失败。 | | :" | 打开用于写入的空文件。 如果给定文件存在，则其内容会被销毁。 | | :" | 打开以便在文件末尾进行写入（追加）；如果文件不存在，则先创建文件。 | | :" | 打开以便读取和写入。 （该文件必须存在。） | | :" | 打开用于读取和写入的空文件。 如果给定文件存在，则其内容会被销毁。 | | :" | 打开以便进行读取和追加；如果文件不存在，则先创建。 |   使用 **“w”** 和 **“w+”** 类型时要小心，因为它们可能会破坏现有文件。  使用 **“a”** 或 **“a+”** 访问类型打开文件时，所有写入操作均将在文件末尾进行。 使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 或 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170) 可重新定位文件指针，但在执行任何写入操作前，文件指针将始终移回文件末尾。因此，无法覆盖现有数据。 指定 **“r+”**、**“w+”** 或 **“a+”** 访问类型时，允许读取和写入（文件将处于打开状态以进行更新）。 但是，在读取与写入之间切换时，必须有中间 [fsetpos](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fsetpos?view=msvc-170)、[fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 或 [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170) 操作。 如果需要的话，可以为 [fsetpos](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fsetpos?view=msvc-170) 或 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 操作指定当前位置。 除了以上值之外，可以在 *mode* 中包含以下字符之一以指定换行符和文件管理的转换模式。  展开表   | **术语** | **定义** | | --- | --- | | **t** | 在文本（转换）模式下打开文件。 在这种模式下，输入时，回车换行 (CR-LF) 组合将转换为单一的换行 (LF)；输出时，LF 字符将转换为 CR-LF 组合。 CTRL+Z 也将在输入时解释为文件尾字符。 在打开以进行读取或读取/写入的文件中，**\_fsopen** 将检查文件末尾的 Ctrl+Z 并在可能的情况下将其移除。 它已被删除，因为使用 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 和 [ftell](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 在以 CTRL+Z 结尾的文件中移动时，可能会导致 [fseek](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170) 在文件末尾附近运行不当。 | | **b** | 在二进制（未转换）模式下打开文件；禁止上述转换。 | | **D** | 指定在关闭最后一个指向该文件的指针时删除的临时文件。 | | **R** | 指定缓存针对（但不限于）从磁盘的随机访问进行优化。 | | **S** | 指定缓存针对（但不限于）从磁盘的顺序访问进行优化。 | | **T** | 指定一个文件，除非内存压力需要它，否则不会写入磁盘。 |   如果 **t** 或 **b** 在 *mode* 中未给出，则转换模式由默认模式变量 **\_fmode** 定义。 如果 **t** 或 **b** 是该参数的前缀，则函数将失败并返回 NULL。 有关文本模式和二进制模式的讨论，请参阅[文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170)。  关于 T 和 D：   * 只要内存压力不需要，T 就避免将文件写入磁盘。 有关详细信息，请参阅[文件属性常量](https://learn.microsoft.com/zh-cn/windows/win32/fileio/file-attribute-constants)中的 FILE\_ATTRIBUTE\_TEMPORARY，以及此博客文章[它只是暂时性的](https://learn.microsoft.com/archive/blogs/larryosterman/its-only-temporary)。 * D 指定写入磁盘的常规文件。 区别在于它在关闭时会自动删除。 可以组合 TD 来获取这两种语义。   \_fsopen 和 \_wfsopen 是 Microsoft 特定的 [fopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) 变体。 它们不属于 ANSI 标准。 对于移植性和安全性更强的函数，如果你不需要文件共享，请考虑 [\_wfopen\_s 或 fopen\_s](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-s-wfopen-s?view=msvc-170)。  *shflag* 参数是常量表达式，它包含 Share.h 中定义的以下清单常量之一。  展开表   | **术语** | **定义** | | --- | --- | | \_SH\_DENYNO | 允许读取和写入访问。 | | \_SH\_DENYRD | 拒绝对文件的读取访问。 | | \_SH\_DENYRW | 拒绝对文件的读取和写入访问。 | | \_SH\_DENYWR | 拒绝对文件的写入访问。 |   默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此行为，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 一般文本例程映射 展开表   | **Tchar.h 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_tfsopen** | **\_fsopen** | **\_fsopen** | **\_wfsopen** |  要求 展开表   | **函数** | **必需的标头** | **可选标头** | | --- | --- | --- | | **\_fsopen** | <stdio.h> | <share.h>  用于 *shflag* 参数的清单常量。 | | **\_wfsopen** | <stdio.h> 或 <wchar.h> | <share.h>  用于 *shflag* 参数的清单常量。 |  示例 C复制  // crt\_fsopen.c  #include <stdio.h>  #include <stdlib.h>  #include <share.h>  int main( void )  {  FILE \*stream;  // Open output file for writing. Using \_fsopen allows us to  // ensure that no one else writes to the file while we are  // writing to it.  //  if( (stream = \_fsopen( "outfile", "wt", \_SH\_DENYWR )) != NULL )  {  fprintf( stream, "No one else in the network can write "  "to this file until we are done.\n" );  fclose( stream );  }  // Now others can write to the file while we read it.  system( "type outfile" );  }  Output复制  No one else in the network can write to this file until we are done. 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [ferror](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ferror?view=msvc-170) [\_fileno](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fileno?view=msvc-170) [\_setmode](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/setmode?view=msvc-170) |

## fgets、fgetws

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从流获取字符串。 语法 C复制  char \*fgets(  char \*str,  int numChars,  FILE \*stream  );  wchar\_t \*fgetws(  wchar\_t \*str,  int numChars,  FILE \*stream  ); 参数 *str* 数据的存储位置。  *numChars* 要读取的最大字符数。  *stream* 指向 FILE 结构的指针。 返回值 其中每个函数都会返回 *str*。 将返回 NULL 指示错误或文件尾条件。 使用 **feof** 或 **ferror** 确定是否出错。 如果 *str* 或 *stream* 是空指针，或者 *numChars* 小于或等于零，此函数会调用无效的参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许继续执行，则将 errno 设置为 EINVAL 并且该函数返回 NULL中所述。  有关返回代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 备注 **fgets** 函数将读取输入 *stream* 参数中的一个字符串，并将其存储到 *str* 中。 **fgets** 会将字符从当前流位置读取到流的结尾（包含第一个换行符），或直到读取的字符数量等于 *numChars* - 1，以先到者为准。 将向存储在 *str* 中的结果追加一个 null 字符。 换行符（如果读取）将包括在字符串中。  **fgetws** 是 **fgets** 的宽字符版本。  当 *stream* 分别以文本模式或二进制模式打开时，**fgetws** 会将宽字符自变量 *str* 作为多字节字符串或宽字符串读取。 若要详细了解如何在 Unicode 和多字节流 I/O 中使用文本和二进制模式，请参阅[文本和二进制模式文件 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/text-and-binary-mode-file-i-o?view=msvc-170) 和[文本和二进制模式下的 Unicode 流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/unicode-stream-i-o-in-text-and-binary-modes?view=msvc-170)。  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此行为，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 一般文本例程映射 展开表   | **TCHAR.H 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_fgetts** | **fgets** | **fgets** | **fgetws** |  要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fgets** | <stdio.h> | | **fgetws** | <stdio.h> 或 <wchar.h> |   有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fgets.c  // This program uses fgets to display  // the first line from a file.  #include <stdio.h>  int main( void )  {  FILE \*stream;  char line[100];  if( fopen\_s( &stream, "crt\_fgets.txt", "r" ) == 0 )  {  if( fgets( line, 100, stream ) == NULL)  printf( "fgets error\numChars" );  else  printf( "%s", line);  fclose( stream );  }  } 输入：crt\_fgets.txt Input复制  Line one.  Line two. 输出 Output复制  Line one. 另请参阅 |

## fputs、fputws

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将字符串写入流。 语法 C复制  int fputs(  const char \*str,  FILE \*stream  );  int fputws(  const wchar\_t \*str,  FILE \*stream  ); 参数 *str* 输出字符串。  *stream* 指向 FILE 结构的指针。 返回值 如果成功，其中每个函数都将返回一个非负值。 发生错误时，**fputs** 和 **fputws** 将返回 EOF。 如果 *str* 或 *stream* 为空指针，这些函数会调用无效的参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许执行继续，则这些功能将 errno 设置为 EINVAL，然后返回 EOF。  有关错误代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 备注 其中每个函数将 *str* 复制到当前位置的输出 *stream*。 当 *stream* 分别以文本模式或二进制模式打开时，**fputws** 会将宽字符自变量 *str* 作为多字节字符串或宽字符串复制到 *stream*。 函数不会复制终止的 null 字符。  如果在 ANSI 模式下打开流，则这两个函数行为相同。 **fputs** 当前不支持到 UNICODE 流中的输出。  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此状态，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 一般文本例程映射 展开表   | **TCHAR.H 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_fputts** | **fputs** | **fputs** | **fputws** |  要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fputs** | <stdio.h> | | **fputws** | <stdio.h> 或 <wchar.h> |   通用 Windows 平台 (UWP) 应用中不支持控制台。 与控制台（**stdin**、**stdout** 和 **stderr**）关联的标准流句柄必须重定向，然后 C 运行时函数才能在 UWP 应用中使用它们。 有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fputs.c  // This program uses fputs to write  // a single line to the stdout stream.  #include <stdio.h>  int main( void )  {  fputs( "Hello world from fputs.\n", stdout );  }  Output复制  Hello world from fputs. |

## fprintf\_s、\_fprintf\_s\_l 、 fwprintf\_s、\_fwprintf\_s\_l

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将格式化数据输出到流。 这些函数的版本是 [fprintf、\_fprintf\_l、fwprintf、\_fwprintf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fprintf-fprintf-l-fwprintf-fwprintf-l?view=msvc-170)，具有安全性增强功能，如 [CRT 中的安全功能](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/security-features-in-the-crt?view=msvc-170)中所述。  有关 \_ftprintf\_s 和 \_ftprintf\_s\_l，请参阅 [泛型文本函数映射](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fprintf-s-fprintf-s-l-fwprintf-s-fwprintf-s-l?view=msvc-170#generic-text-function-mappings)。 语法 C复制  int fprintf\_s(  FILE \*stream,  const char \*format [,  argument\_list ]  );  int \_fprintf\_s\_l(  FILE \*stream,  const char \*format,  \_locale\_t locale [,  argument\_list ]  );  int fwprintf\_s(  FILE \*stream,  const wchar\_t \*format [,  argument\_list ]  );  int \_fwprintf\_s\_l(  FILE \*stream,  const wchar\_t \*format,  \_locale\_t locale [,  argument\_list ]  ); 参数 *stream* 指向 FILE 结构的指针。  *format* 窗体控件字符串。  *argument\_list* 格式字符串的可选参数。  *locale* 要使用的区域设置。 返回值 **fprintf\_s** 返回已写入的字节数。 **fwprintf\_s** 返回已写入的宽字符数。 其中每个函数在出现输出错误时返回一个负值。 备注 **fprintf\_s** 格式化一系列字符和值并将其输出到输出 *stream*。 *argument\_list* 中的每个参数（如果有）根据 *format* 中相应的格式规范进行转换和输出。 *format* 参数使用[printf 和 wprintf 函数的格式规范语法](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions?view=msvc-170)。  **fwprintf\_s** 是 **fprintf\_s** 的宽字符版本；在 **fwprintf\_s** 中，*format* 是宽字符字符串。 如果在 ANSI 模式下打开流，则这些函数行为相同。 **fprintf\_s** 当前不支持到 UNICODE 流中的输出。  这些带有 **\_l** 后缀的函数的版本相同，只不过它们使用传递的区域设置参数而不是当前区域设置。  **重要**  确保 *format* 不是用户定义的字符串。  从 Windows 10 版本 2004（内部版本 19041）开始，printf 系列函数根据 IEEE 754 的舍入规则输出可精确表示的浮点数。 在早期的 Windows 版本中，以“5”结尾并且可精确表示的浮点数总是向上取整。 IEEE 754 规定它们必须舍入到最接近的偶数（也称为“四舍六入五成双”）。 例如，printf("%1.0f", 1.5) 和 printf("%1.0f", 2.5) 都应舍入为 2。 之前，1.5 舍入为 2，2.5 舍入为 3。 此更改仅影响可精确表示的数字。 例如，2.35（用于内存表示时更接近于 2.35000000000000008）仍然向上取整为 2.4。 这些函数完成的舍入现在也遵循 [**fesetround**](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fegetround-fesetround2?view=msvc-170) 设置的浮点舍入模式。 以前，舍入始终选择 FE\_TONEAREST 行为。 此更改仅影响使用 Visual Studio 2019 版本 16.2 及更高版本生成的程序。 若要使用旧的浮点舍入行为，请链接到 [**'legacy\_stdio\_float\_rounding.obj`**](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/link-options?view=msvc-170)。  与不安全版本一样（请参见 [fprintf、\_fprintf\_l、fwprintf、\_fwprintf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fprintf-fprintf-l-fwprintf-fwprintf-l?view=msvc-170)），如果 *stream* 或 *format* 为 NULL 指针，则这些函数将验证其参数并调用无效参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 格式字符串本身也会得到验证。 如果有任何未知或格式错误的格式化说明符，则这些函数将生成无效参数异常。 在所有情况下，如果允许执行继续，则这些函数将返回 -1 并将 errno 设置为 EINVAL。 有关返回代码的详细信息，请参阅 [errno、\_doserrno、\_sys\_errlist 和 \_sys\_nerr](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/errno-doserrno-sys-errlist-and-sys-nerr?view=msvc-170)。 泛型文本函数映射 列中的 tchar.h 函数映射到其他列中的函数，具体取决于编译时定义的字符集。  展开表   | **tchar.h 函数** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | \_ftprintf\_s | fprintf\_s | fprintf\_s | fwprintf\_s | | \_ftprintf\_s\_l | \_fprintf\_s\_l | \_fprintf\_s\_l | \_fwprintf\_s\_l |   有关详细信息，请参阅[格式规范语法](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions?view=msvc-170)。 要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fprintf\_s**, **\_fprintf\_s\_l** | <stdio.h> | | **fwprintf\_s**, **\_fwprintf\_s\_l** | <stdio.h> 或 <wchar.h> |   有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fprintf\_s.c  // This program uses fprintf\_s to format various  // data and print it to the file named FPRINTF\_S.OUT. It  // then displays FPRINTF\_S.OUT on the screen using the system  // function to invoke the operating-system TYPE command.  #include <stdio.h>  #include <process.h>  FILE \*stream;  int main( void )  {  int i = 10;  double fp = 1.5;  char s[] = "this is a string";  char c = '\n';  fopen\_s( &stream, "fprintf\_s.out", "w" );  fprintf\_s( stream, "%s%c", s, c );  fprintf\_s( stream, "%d\n", i );  fprintf\_s( stream, "%f\n", fp );  fclose( stream );  system( "type fprintf\_s.out" );  }  Output复制  this is a string  10  1.500000 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [\_cprintf、、\_cprintf\_l\_cwprintf、、\_cwprintf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/cprintf-cprintf-l-cwprintf-cwprintf-l?view=msvc-170) [fscanf、、\_fscanf\_lfwscanf、、\_fwscanf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fscanf-fscanf-l-fwscanf-fwscanf-l?view=msvc-170) [sprintf、、\_sprintf\_lswprintf、\_swprintf\_l、\_\_swprintf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sprintf-sprintf-l-swprintf-swprintf-l-swprintf-l?view=msvc-170) |

## fscanf\_s、\_fscanf\_s\_l、fwscanf\_s、\_fwscanf\_s\_l

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从流中读取格式化数据。 这些版本的 [fscanf、\_fscanf\_l、fwscanf、\_fwscanf\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fscanf-fscanf-l-fwscanf-fwscanf-l?view=msvc-170) 具有安全增强功能，如 [CRT 中的安全功能](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/security-features-in-the-crt?view=msvc-170)中所述。 语法 C复制  int fscanf\_s(  FILE \*stream,  const char \*format [,  argument ]...  );  int \_fscanf\_s\_l(  FILE \*stream,  const char \*format,  \_locale\_t locale [,  argument ]...  );  int fwscanf\_s(  FILE \*stream,  const wchar\_t \*format [,  argument ]...  );  int \_fwscanf\_s\_l(  FILE \*stream,  const wchar\_t \*format,  \_locale\_t locale [,  argument ]...  ); 参数 *stream* 指向 FILE 结构的指针。  *format* 窗体控件字符串。  *argument* 可选参数。  *locale* 要使用的区域设置。 返回值 其中每个函数都将返回成功转换和分配的字段数。 返回值不包括已读取但未分配的字段。 返回值为 0 表示没有分配任何字段。 如果出现错误或在首次转换前达到文件流的结尾，则 **fscanf\_s** 和 **fwscanf\_s** 的返回值为 EOF。  这些函数验证其参数。 如果 *stream* 是无效文件指针，或 *format* 是空指针，这些函数将调用无效参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许执行继续，则这些函数将返回 EOF 并将 errno 设置为 EINVAL。 备注 **fscanf\_s** 函数将从 *stream* 的当前位置将数据读取到 *argument*（如果有）指定的位置。 每个 *argument* 必须为指向类型的变量的指针，该类型与 *format* 中的类型说明符对应。 *format* 控制对输入域的解释，其形式和函数与 **scanf\_s** 的 *format* 参数相同；有关 *format* 的说明，请参阅[格式规范域：scanf 和 wscanf 函数](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/format-specification-fields-scanf-and-wscanf-functions?view=msvc-170)。 **fwscanf\_s** 是 **fscanf\_s** 的宽字符版本；**fwscanf\_s** 格式参数是宽字符字符串。 如果在 ANSI 模式下打开流，则这些函数行为相同。 **fscanf\_s** 当前不支持 UNICODE 流的输入。  更安全的函数（带有 **\_s** 后缀）与其他版本之间的主要区别在于，更安全的函数需要每个 **c**、**C**、**s**、**S** 和 **[** 类型字段的以字符为单位的大小作为紧跟变量的参数进行传递。 有关详细信息，请参阅 [scanf\_s、\_scanf\_s\_l、wscanf\_s、\_wscanf\_s\_l](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/scanf-s-scanf-s-l-wscanf-s-wscanf-s-l?view=msvc-170) 和 [scanf宽度规范](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/scanf-width-specification?view=msvc-170)。  **备注**  大小参数的类型具有 **unsigned**，而不具有 **size\_t**。  这些带有 **\_l** 后缀的函数的版本相同，只不过它们使用传递的区域设置参数而不是当前线程区域设置。 一般文本例程映射 展开表   | **TCHAR.H 例程** | **\_UNICODE 和 \_MBCS 未定义** | **\_MBCS 已定义** | **\_UNICODE 已定义** | | --- | --- | --- | --- | | **\_ftscanf\_s** | **fscanf\_s** | **fscanf\_s** | **fwscanf\_s** | | **\_ftscanf\_s\_l** | **\_fscanf\_s\_l** | **\_fscanf\_s\_l** | **\_fwscanf\_s\_l** |  要求 展开表   | **函数** | **必需的标头** | | --- | --- | | %> | <stdio.h> | | %> | <stdio.h> 或 <wchar.h> |   有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fscanf\_s.c  // This program writes formatted  // data to a file. It then uses fscanf to  // read the various data back from the file.  #include <stdio.h>  #include <stdlib.h>  FILE \*stream;  int main( void )  {  long l;  float fp;  char s[81];  char c;  errno\_t err = fopen\_s( &stream, "fscanf.out", "w+" );  if( err )  printf\_s( "The file fscanf.out was not opened\n" );  else  {  fprintf\_s( stream, "%s %ld %f%c", "a-string",  65000, 3.14159, 'x' );  // Set pointer to beginning of file:  fseek( stream, 0L, SEEK\_SET );  // Read data back from file:  fscanf\_s( stream, "%s", s, \_countof(s) );  fscanf\_s( stream, "%ld", &l );  fscanf\_s( stream, "%f", &fp );  fscanf\_s( stream, "%c", &c, 1 );  // Output data read:  printf( "%s\n", s );  printf( "%ld\n", l );  printf( "%f\n", fp );  printf( "%c\n", c );  fclose( stream );  }  }  Output复制  a-string  65000  3.141590  x 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) |

## fseek、\_fseeki64

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将文件指针移到指定位置。 语法 C复制  int fseek(  FILE \*stream,  long offset,  int origin  );  int \_fseeki64(  FILE \*stream,  \_\_int64 offset,  int origin  ); 参数 *stream* 指向 FILE 结构的指针。  *offset* *origin* 中的字节数。  *origin* 初始位置。 返回值 如果成功，则 **fseek** 和 **\_fseeki64** 返回 0。 否则，返回一个非零值。 在无法查找的设备上，返回值是未定义的。 如果 *stream* 为空指针，或者 *origin* 不是下述允许的值之一，**fseek** 和 **\_fseeki64** 会调用无效的参数处理程序，如[参数验证](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/parameter-validation?view=msvc-170)中所述。 如果允许执行继续，则这些功能将 errno 设置为 EINVAL 并返回 -1。 注解 **fseek** 和 **\_fseeki64** 函数将与 *stream* 关联的文件指针（如有）一起移动到新位置，该位置在 *origin* 中为 *offset* 个字节。 该流上的下一步操作发生在新位置。 在准备更新的流上，下一个操作可以是读取或写入。 参数 *origin* 必须是 STDIO.H 中定义的以下常量之一：  展开表   | **origin 值** | **含义** | | --- | --- | | SEEK\_CUR | 文件指针的当前位置。 | | SEEK\_END | 文件结尾。 | | SEEK\_SET | 文件开头。 |   可以使用 **fseek** 和 **\_fseeki64** 在文件的任何位置重新定位指针。 此外还可以在文件结尾外放置指针。 **fseek** 和 **\_fseeki64** 清除文件结尾指示器，并使先前针对 *stream* 的任何 [ungetc](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ungetc-ungetwc?view=msvc-170) 调用失效。  当文件打开以追加数据时，当前文件位置由最后的 I/O 操作确定，而不是由发生下一个写入的位置确定。 如果在为追加而打开的文件中尚未发生 I/O 操作，则文件位置是文件开头。  对于在文本模式下打开的流，**fseek** 和 **\_fseeki64** 的用途相当有限，因为回车-换行转换可能导致 **fseek** 和 **\_fseeki64** 产生意外结果。 唯一保证作用于在文本模式下打开的流的 **fseek** 和 **\_fseeki64** 操作是：   * 使用相对于任何原始值的偏移 0 进行查找。 * 使用 **fseek** 或 [\_ftelli64](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 或使用 **\_fseeki64** 时，从文件开头通过调用 [ftell](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 所返回的偏移值进行查找。   此外，在文本模式中，CTRL+Z 将在输入时解释为文件结尾字符。 在打开以进行读取/写入的文件中，[fopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) 和所有相关例程将检查文件末尾的 Ctrl+Z 并在可能的情况下将其移除。 会移除它，因为将 **fseek** 和 [ftell](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 或者将 **\_fseeki64** 和 [\_ftelli64](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 结合使用在以 CTRL+Z 结尾的文件中移动，可能导致 **fseek** 或 **\_fseeki64** 在文件结尾附近无法正常工作。  当 CRT 打开以字节顺序标记 (BOM) 开头的文件时，文件指针位于 BOM 后面。 （即，位于文件实际内容的开头）。 如果需要 **fseek** 到文件开头，请使用 [ftell](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/ftell-ftelli64?view=msvc-170) 来获取初始位置，然后 **fseek** 到该位置而不是位置 0。  此函数在执行期间将锁定其他线程，因此是线程安全的。 有关非锁定版本，请参阅 [\_fseek\_nolock、\_fseeki64\_nolock](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-nolock-fseeki64-nolock?view=msvc-170)。  默认情况下，此函数的全局状态范围限定为应用程序。 若要更改此行为，请参阅 [CRT 中的全局状态](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/global-state?view=msvc-170)。 要求 展开表   | **函数** | **必需的标头** | | --- | --- | | **fseek** | <stdio.h> | | **\_fseeki64** | <stdio.h> |   有关兼容性的详细信息，请参阅 [兼容性](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/compatibility?view=msvc-170)。 示例 C复制  // crt\_fseek.c  // This program opens the file FSEEK.OUT and  // moves the pointer to the file's beginning.  #include <stdio.h>  int main( void )  {  FILE \*stream;  char line[81];  int result;  if ( fopen\_s( &stream, "fseek.out", "w+" ) != 0 )  {  printf( "The file fseek.out was not opened\n" );  return -1;  }  fprintf( stream, "The fseek begins here: "  "This is the file 'fseek.out'.\n" );  result = fseek( stream, 23L, SEEK\_SET);  if( result )  perror( "Fseek failed" );  else  {  printf( "File pointer is set to middle of first line.\n" );  fgets( line, 80, stream );  printf( "%s", line );  }  fclose( stream );  }  Output复制  File pointer is set to middle of first line.  This is the file 'fseek.out'. 另请参阅 [流 I/O](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/stream-i-o?view=msvc-170) [rewind](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/rewind?view=msvc-170) |

# 2.C++文件输入输出流fstream的基本用法

## 一、文件流

C++的IO类中定义了三个文件读写流fstream、ifstream以及ofstream，它们都继承于相同的父类istream，通过不同的实现以实现不同的文件流操作。

三者的区别为：

* ifstream：从文件读取数据
* ofstream：从文件写入数据
* fstream：既可以读数据、又可以写数据

### 1.1 IO接口和读写模式

三个文件流实现了以下几个函数接口：

| **函数名** | **用途** |
| --- | --- |
| open(s, mode) | 以mode模式打开文件s |
| close() | 关闭文件流 |
| is\_open() | 返回文件是否已经打开 |
| read(buff, size) | 读入最多size字节数据到buff中 |
| write(buff, size) | 写入size字节数据到文件中 |

在使用open的时候，可以只传入文件s，不指定打开模式。如果不指定模式，系统会自动根据文件类型选择默认的打开模式。同时，除了open()的方式打开文件以外，还可以在对象构造的时候打开文件：

|  |  |
| --- | --- |
| 1 | ofstream output("/tmp/test.txt", ios::out); |

ios::out表示已只读方式打开文件，对应unix c中的O\_WRONLY模式。在C++中，有以下读写模式可以选择：

| **模式** | **说明** |
| --- | --- |
| ios::in | 以读方式打开 |
| ios::out | 以写方式打开 |
| ios::app | 以追加写方式打开 |
| ios::trunc | 以截断方式打开文件 |
| ios:binary | 以二进制方式打开文件 |
| ios::ate | 打开文件后指针定位到文件尾 |

这些模式可以单独使用，也可以组合使用，如果需要组合使用，使用逻辑操作符|或起来即可。这里要特别注意的是ios::out模式默认会截断文件，也就是说，ios::out和ios::out | ios::trunc效果是一样的，都会将文件截断。如果不希望以截断方式打开文件时，则需要设置读写模式为ios::out | ios::app，以这种模式打开文件后，数据会以追加的方式写入到文件。

### 1.2 读写文件示例

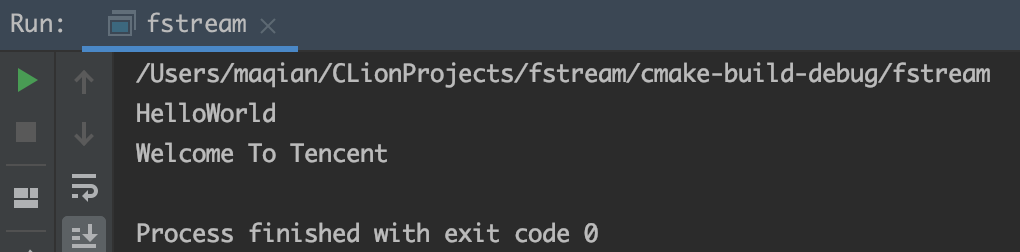
**写文件**

|  |  |
| --- | --- |
|  | // 写文件  void write\_file() {    ofstream output;    // 待写入数据    string output\_data = "HelloWorld\nWelcome To Tencent\n";      // 打开文件    output.open("test.txt", ios::out);      // 写入数据    output.write(output\_data.c\_str(), output\_data.size());      // 关闭文件    output.close();  } |

**读文件**

|  |  |
| --- | --- |
|  | void read\_file() {    ifstream input;    char input\_data[1024];      // 打开文件    input.open("test.txt", ios::in);      // 读取数据    input.read(input\_data, 1024);      // 打印读取到的数据    cout << input\_data;      // 关闭文件    input.close();  } |

打印结果：



### 1.3 以IO操作符读写数据

三个IO操作类都继承于istream，可以直接使用IO操作符（<<和>>）来进行文件读写。

|  |
| --- |
| ofstream output("test.txt");  ifstream input("test.txt");  std::string input\_data;    output << "HelloWorld\n";  output.close();    input >> input\_data;  cout << input\_data;  input.close();    return 0; |

注意：**在使用IO操作符从文件读取数据的时候，数据输入的对象可以是字符串，也可以是对应的数据类型（如int）。如果输入到字符串，默认是读到空白字符的时候就停止了，需要通过循环控制读取后面的数据。**

## 二、流状态

流在执行IO操作的时候，会根据不同的情况产生不同的状态码，如：

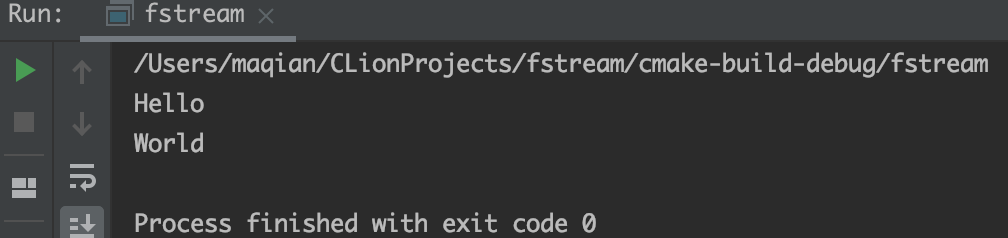
| **状态位** | **说明** |
| --- | --- |
| strm::badbit | 流已崩溃 |
| strm::failgit | IO已崩溃 |
| strm::eofbit | 已经读到文件尾 |
| strm::goodbit | 一切正常，没有异常 |

我们可以使用已经封装好的函数bad()/fail()/eof()/good()来判断当前IO是否已经达到某种状态，如判断文件是否已经读到结束。同时，我们还可以使用clear()函数来清除当前IO对象的状态位，当流已崩溃（如将流中的字符串对象读到一个int对象上）时，可以手动来清除状态位继续往下读取。

示例，使用eof标志位判断文件是否读完成

|  |
| --- |
| ofstream output("test.txt");  ifstream input("test.txt");  std::string input\_data;    output << "Hello World\n";  output.close();    while (!input.eof()) {  // 没有读到文件尾就一直读取    input\_data.clear();      input >> input\_data;  // 读到空白符就停止    // 过滤掉空行    if (input\_data == "") continue;      cout << input\_data << endl;  }  input.close(); |

因为使用IO操作符从流中读取数据默认遇到空行就停止，所以Hello World\n字符串需要读三次才能读完，第一次是Hello，第二次是World，第三次是换行后的空行，所以，最后的输出结果是：



## 三、getline一次读一行

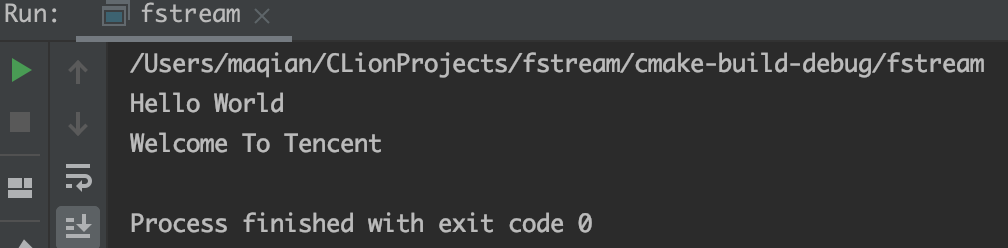
流对象中内置了getline()成员方法可以一次读一行数据到字符串数组中，需要传入一个字符数组和最大长度size：

|  |
| --- |
| ifstream input("test.txt", ios::in);  char buff[1024];  while (!input.eof()) {    input.getline(buff, 1024)    cout << buff << endl;  } |

这个成员方法可以实现一次读一行的操作，但是只支持C风格的字符串传入，无法支持C++中的string类型。这里可以使用系统库的getline()函数来实现，系统库中有一个getline()函数可以直接从IO流中读取一行数据到string中：

|  |  |
| --- | --- |
|  | ifstream input("test.txt", ios::in);  std::string input\_data;  while (!input.eof()) {    getline(input, input\_data);    cout << input\_data << endl;  } |

上面两个实现都可以正常按行输出文件内容：



## 四、fstream和文件指针

当使用fstream流时，对象既可以写入文件，也可以读取文件。此时读写文件指针是公共的，写入文件会导致指针后移，再读就会从当前位置重新读取（读到垃圾数据）。为了避免这个问题，可以使用seekp和seekg来移动文件指针：

|  |  |
| --- | --- |
|  | fstream file("test.txt", ios::in | ios::out);  string input\_data;    file << "HelloWorld\n" << unitbuf;    // 移动读指针到文件开始处  file.seekg(0, ios::beg);  file >> input\_data;  cout << input\_data; |

## 五、错误处理

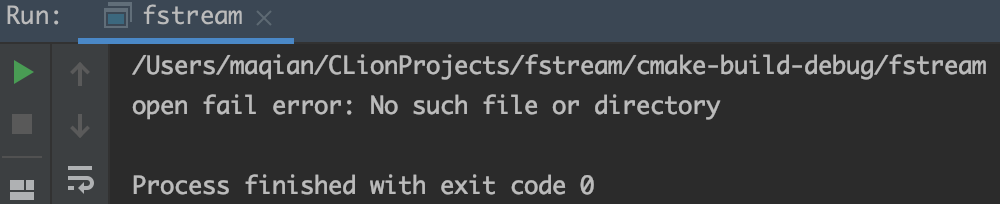
### 5.1 读取文件失败了，应该如何判断

执行完open操作后，直接判断对象是否为真即可确定打开文件是否失败。出现错误后，可以使用errno输出错误原因。

例如，打开一个不存在的文件：

|  |  |
| --- | --- |
|  | ifstream input("aabbcc.txt", ios::in);  if (!input) {    cout << "open fail error: " << strerror(errno) << endl;  } |

输出结果会打印出文件不存在的错误：



 https://www.dyxmq.cn/program/code/c-cpp/fstream.html

# 3.win32文件函数

## CreateFileA 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) |

## 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) |

## 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++复制  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) |

## CopyFile 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将现有文件复制到新文件。  [CopyFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-copyfileexa) 函数提供两个附加功能。 每次完成复制操作的一部分时，**CopyFileEx** 都可以调用指定的回调函数，并且可以在复制操作期间取消 **CopyFileEx**。  若要以事务处理操作的形式执行此操作，请使用 [CopyFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-copyfiletransacteda) 函数。 语法 C++复制  BOOL CopyFile(  [in] LPCTSTR lpExistingFileName,  [in] LPCTSTR lpNewFileName,  [in] BOOL bFailIfExists  ); 参数 [in] lpExistingFileName  现有文件的名称。  默认情况下，名称限制为MAX\_PATH个字符。 若要将此限制扩展到 32,767 个宽字符，请在路径前面添加“\？\”。 有关详细信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，可以选择删除MAX\_PATH限制，而无需在前面添加“\\？\”。 有关详细信息，请参阅 [**命名文件、路径和命名空间**](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file) 的“最大路径长度限制”部分。  如果 *lpExistingFileName* 不存在， **则 CopyFile** 将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_FILE\_NOT\_FOUND**。  [in] lpNewFileName  新文件的名称。  默认情况下，名称限制为MAX\_PATH个字符。 若要将此限制扩展到 32,767 个宽字符，请在路径前面添加“\？\”。 有关详细信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，可以选择删除MAX\_PATH限制，而无需在前面添加“\\？\”。 有关详细信息，请参阅 [**命名文件、路径和命名空间**](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file) 的“最大路径长度限制”部分。  [in] bFailIfExists  如果此参数为 **TRUE** 并且 *lpNewFileName* 指定的新文件已存在，则函数将失败。 如果此参数为 **FALSE** 且新文件已存在，则函数将覆盖现有文件并成功。 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 现有文件 (**ATTRIBUTE\_SECURITY\_INFORMATION**) 的安全资源属性将复制到新文件。  **Windows 7、Windows Server 2008 R2、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在Windows 8和Windows Server 2012之前，现有文件的安全资源属性不会复制到新文件。  现有文件的文件属性将复制到新文件。 例如，如果现有文件具有 **FILE\_ATTRIBUTE\_READONLY** 文件属性，则通过调用 **CopyFile** 创建的副本也将具有 **FILE\_ATTRIBUTE\_READONLY** 文件属性。 有关详细信息，请参阅 [检索和更改文件属性](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/retrieving-and-changing-file-attributes)。  如果目标文件已存在并且设置了**FILE\_ATTRIBUTE\_HIDDEN或FILE\_ATTRIBUTE\_READONLY**属性，则此函数将失败**并ERROR\_ACCESS\_DENIED**。  当 **CopyFile** 用于复制加密的文件时，它会尝试使用源文件加密中使用的密钥来加密目标文件。 如果无法执行此操作，此函数将尝试使用默认密钥加密目标文件。 如果这两种方法都无法完成， **则 CopyFile** 将失败并 **显示ERROR\_ENCRYPTION\_FAILED** 错误代码。  符号链接行为 - 如果源文件是符号链接，则复制的实际文件是符号链接的目标。  如果目标文件已存在并且是符号链接，则符号链接的目标将被源文件覆盖。  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 [检索和更改文件属性](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/retrieving-and-changing-file-attributes)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CopyFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-copyfileexa)  [CopyFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-copyfiletransacteda)  [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)  [文件特性常量](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-attribute-constants)  [文件管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [MoveFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefile)  [符号链接](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/symbolic-links) |

## DeleteFile 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 删除现有的文件。  若要以事务处理操作的形式执行此操作，请使用 [DeleteFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-deletefiletransacteda) 函数。 语法 C++复制  BOOL DeleteFile(  [in] LPCTSTR lpFileName  ); 参数 [in] lpFileName  要删除的文件的名称。  默认情况下，名称限制为MAX\_PATH个字符。 若要将此限制扩展到 32,767 个宽字符，请在路径前面添加“\\？\”。 有关详细信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，可以选择删除MAX\_PATH限制，而无需在前面添加“\\？\”。 有关详细信息，请参阅 [**命名文件、路径和命名空间**](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file) 的“最大路径长度限制”部分。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零 (0)。 要获得更多的错误信息，请调用 GetLastError。 注解 如果应用程序尝试删除不存在的文件， **DeleteFile** 函数将失败并 **ERROR\_FILE\_NOT\_FOUND**。 如果文件是只读文件，则函数将失败并 **ERROR\_ACCESS\_DENIED**。  以下列表标识了删除、删除或关闭文件的一些提示：   * 若要删除只读文件，首先必须删除只读属性。 * 若要删除或重命名文件，必须对文件具有删除权限，或者具有父目录中的删除子权限。 * 若要以递归方式删除目录中的文件，请使用 [SHFileOperation](https://learn.microsoft.com/zh-cn/windows/desktop/api/shellapi/nf-shellapi-shfileoperationa) 函数。 * 若要删除空目录，请使用 [RemoveDirectory](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-removedirectorya) 函数。 * 若要关闭打开的文件，请使用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数。   如果设置了除 delete 和 delete child 以外的所有访问权限的目录，并且访问控制列表 (ACL) 继承了新文件，则可以创建一个文件，但无法将其删除。 但是，你可以创建一个文件，然后获取在创建文件时返回给你的句柄上请求的所有访问权限。  如果在创建文件时请求删除权限，则可以使用该句柄删除或重命名文件，但不能使用任何其他句柄。 有关详细信息，请参阅 [文件安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)。  如果应用程序尝试删除具有打开正常 I/O 的其他句柄的文件或作为内存映射文件的文件 (**FILE\_SHARE\_DELETE**必须在打开其他句柄) 时指定，**DeleteFile** 函数将失败。  **DeleteFile** 函数在关闭时标记要删除的文件。 因此，在关闭文件的最后一个句柄之前，不会删除文件。 后续调用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 以打开文件失败， **并显示ERROR\_ACCESS\_DENIED**。  符号链接行为 —  如果路径指向符号链接，则会删除符号链接，而不是目标。 若要删除目标，必须调用 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea) 并指定 **FILE\_FLAG\_DELETE\_ON\_CLOSE**。  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 [锁定和解锁文件中的字节范围](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/locking-and-unlocking-byte-ranges-in-files)。 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)  [DeleteFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-deletefiletransacteda)  [文件管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [符号链接](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/symbolic-links) |

## MoveFile 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 移动现有文件或目录，包括其子级。  若要指定如何移动文件，请使用 [MoveFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefileexa) 或 [MoveFileWithProgress](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefilewithprogressa) 函数。  若要以事务处理操作的形式执行此操作，请使用 [MoveFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefiletransacteda) 函数。 语法 C++  BOOL MoveFile(  [in] LPCTSTR lpExistingFileName,  [in] LPCTSTR lpNewFileName  ); 参数 [in] lpExistingFileName  本地计算机上文件或目录的当前名称。  默认情况下，名称限制为MAX\_PATH个字符。 若要将此限制扩展到 32，767 个宽字符，请在路径前面添加“\\？\”。 有关详细信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，可以选择删除MAX\_PATH限制，而无需在前面添加“\\？\”。 有关详细信息，请参阅 [**命名文件、路径和命名空间**](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file) 的“最大路径长度限制”部分。  [in] lpNewFileName  文件或目录的新名称。 新名称不得已存在。 新文件可能位于不同的文件系统或驱动器上。 新目录必须位于同一驱动器上。  默认情况下，名称限制为MAX\_PATH个字符。 若要将此限制扩展到 32，767 个宽字符，请在路径前面添加“\\？\”。 有关详细信息，请参阅[命名文件、路径和命名空间](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file)。  **提示**  从 Windows 10 版本 1607 开始，可以选择删除MAX\_PATH限制，而无需在前面添加“\\？\”。 有关详细信息，请参阅 [**命名文件、路径和命名空间**](https://learn.microsoft.com/zh-cn/windows/win32/fileio/naming-a-file) 的“最大路径长度限制”部分。 返回值 如果该函数成功，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 **MoveFile** 函数将移动 (重命名) 文件或目录 (包括其子级) 在同一目录中或跨目录。 需要注意的是，当目标位于其他卷上时，移动目录时 **，MoveFile** 函数将失败。  如果文件跨卷移动， **MoveFile** 不会随文件一起移动安全描述符。 将为文件分配目标目录中的默认安全描述符。  **MoveFile** 函数将其操作与链接跟踪服务协调，因此可以在移动链接源时对其进行跟踪。  在 Windows 8 和 Windows Server 2012 中，以下技术支持此函数。  展开表   |  |  | | --- | --- | | **技术** | **支持** | | (SMB) 3.0 协议的服务器消息块 | 是 | | SMB 3.0 透明故障转移 (TFO) | 查看注释 | | 具有横向扩展文件共享的 SMB 3.0 (SO) | 查看注释 | | 群集共享卷文件系统 (CsvFS) | 是 | | 弹性文件系统 (ReFS) | 是 |     SMB 3.0 不支持重命名具有连续可用性功能的文件共享上的备用数据流。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CopyFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-copyfile)  [文件管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-management-functions)  [MoveFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefileexa)  [MoveFileTransacted](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefiletransacteda)  [MoveFileWithProgress](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-movefilewithprogressa) |

# 4.MFC文件函数: CStdioFile 类

表示由运行时函数 [fopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) 打开的 C 运行时流文件。

## 语法

class CStdioFile : public CFile

## 成员

### 公共构造函数

展开表

| **名称** | **描述** |
| --- | --- |
| [CStdioFile::CStdioFile](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#cstdiofile) | 从路径或文件指针构造 CStdioFile 对象。 |

### 公共方法

展开表

| **名称** | **描述** |
| --- | --- |
| [CStdioFile::Open](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#open) | 已重载。 Open 旨在与默认 CStdioFile 构造函数结合使用（替代 [CFile::Open](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#open)）。 |
| [CStdioFile::ReadString](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#readstring) | 读取单行文本。 |
| [CStdioFile::Seek](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#seek) | 定位当前文件指针。 |
| [CStdioFile::WriteString](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#writestring) | 写入单行文本。 |

### 公共数据成员

展开表

| **“属性”** | **描述** |
| --- | --- |
| [CStdioFile::m\_pStream](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#m_pstream) | 包含指向已打开文件的指针。 |

## 注解

流文件会进行缓冲，可以在文本模式（默认）或二进制模式下打开。

文本模式为回车-换行符对提供特殊处理。 将换行符 (0x0A) 写入文本模式 CStdioFile 对象时，字节对 (0x0D, 0x0A) 会发送到文件。 进行读取时，字节对 (0x0D, 0x0A) 会转换为单个 0x0A 字节。

CStdioFile 不支持 [CFile](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170) 函数 [Duplicate](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#duplicate)、[LockRange](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#lockrange) 和 [UnlockRange](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#unlockrange)。

如果对 CStdioFile 调用这些函数，则会获得 [CNotSupportedException](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cnotsupportedexception-class?view=msvc-170)。

有关如何使用 CStdioFile 的详细信息，请参阅运行时库参考中的文章 [MFC 中的文件](https://learn.microsoft.com/zh-cn/cpp/mfc/files-in-mfc?view=msvc-170)和[文件处理](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/file-handling?view=msvc-170)。

## 继承层次结构

[CObject](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cobject-class?view=msvc-170)

[CFile](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170)

CStdioFile

## 要求

**标头**：afx.h

## CStdioFile::CStdioFile

构造并初始化一个 CStdioFile 对象。

复制

CStdioFile();

CStdioFile(CAtlTransactionManager\* pTM);

CStdioFile(FILE\* pOpenStream);

CStdioFile(

LPCTSTR lpszFileName,

UINT nOpenFlags);

CStdioFile(

LPCTSTR lpszFileName,

UINT nOpenFlags,

CAtlTransactionManager\* pTM);

### 参数

*pOpenStream*  
指定调用 C 运行时函数 [fopen](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fopen-wfopen?view=msvc-170) 时返回的文件指针。

*lpszFileName*  
指定一个字符串，它是所需文件的路径。 路径可以是相对路径或绝对路径。

*nOpenFlags*  
指定用于文件创建、文件共享和文件访问模式的选项。 可以使用按位“或”(**|**) 运算符指定多个选项。

一个文件访问模式选项是必需的；其他模式是可选的。 请参阅 [CFile::CFile](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#cfile) 以了解模式选项和其他标志的列表。 在 MFC 版本 3.0 及更高版本中，允许使用共享标志。

*pTM*  
指向 CAtlTransactionManager 对象的指针。

### 注解

默认构造函数不会将文件附加到 CStdioFile 对象。 使用此构造函数时，必须使用 CStdioFile::Open 方法打开文件并将其附加到 CStdioFile 对象。

单参数构造函数会将打开的文件流附加到 CStdioFile 对象。 允许的指针值包括预定义输入/输出文件指针 *stdin*、*stdout* 或 *stderr*。

双参数构造函数会创建 CStdioFile 对象，并使用给定路径打开对应文件。

如果为 *pOpenStream* 或 *lpszFileName* 传递 NULL，则构造函数会引发 CInvalidArgException\*。

如果无法打开或创建文件，则构造函数会引发 CFileException\*。

### 示例

C++复制

TCHAR\* pFileName = \_T("CStdio\_File.dat");

CStdioFile f1;

if(!f1.Open(pFileName, CFile::modeCreate | CFile::modeWrite

| CFile::typeText))

{

TRACE(\_T("Unable to open file\n"));

}

CStdioFile f2(stdout);

try

{

CStdioFile f3( pFileName,

CFile::modeCreate | CFile::modeWrite | CFile::typeText );

}

catch(CFileException\* pe)

{

TRACE(\_T("File could not be opened, cause = %d\n"),

pe->m\_cause);

pe->Delete();

}

## CStdioFile::m\_pStream

m\_pStream 数据成员是指向 C 运行时函数 fopen 返回的已打开文件的指针。

复制

FILE\* m\_pStream;

### 备注

如果文件从未打开或已关闭，则为 NULL。

## CStdioFile::Open

已重载。 Open 旨在与默认 CStdioFile 构造函数结合使用。

复制

virtual BOOL Open(

LPCTSTR lpszFileName,

UINT nOpenFlags,

CFileException\* pError = NULL);

virtual BOOL Open(

LPCTSTR lpszFileName,

UINT nOpenFlags,

CAtlTransactionManager\* pTM,

CFileException\* pError = NULL);

### 参数

*lpszFileName*  
一个字符串，它是所需文件的路径。 路径可以是相对路径或绝对路径。

*nOpenFlags*  
共享和访问模式。 指定打开文件时要执行的操作。 可以使用按位“或”(**|**) 运算符来组合选项。 一个访问权限和一个共享选项是必需的；modeCreate 和 modeNoInherit 模式是可选的。

*pError*  
指向接收失败操作状态的现有文件异常对象的指针。

*pTM*  
指向 CAtlTransactionManager 对象的指针。

### 返回值

如果成功，则返回 TRUE；否则返回 FALSE。

### 注解

## CStdioFile::ReadString

将文本数据从与 CStdioFile 对象关联的文件读取到缓冲区中（最多限制为 *nMax*-1 个字符）。

复制

virtual LPTSTR ReadString(

LPTSTR lpsz,

UINT nMax);

virtual BOOL ReadString(CString& rString);

### 参数

*lpsz*  
指定一个指向用户提供的缓冲区的指针，该缓冲区将接收以 null 结尾的文本字符串。

*nMax*  
指定要写入到 *lpsz* 缓冲区的最大字符数，包括结尾的 null。

*rString*  
对 CString 对象的引用，该对象将在函数返回时包含字符串。

### 返回值

指向包含文本数据的缓冲区的指针。 如果在没有读取任何数据的情况下到达文件尾，则为 NULL；或者，如果是布尔值，在没有读取任何数据的情况下到达文件尾，则为 FALSE。

### 备注

读取由第一个换行符停止。 在这种情况下，如果读取的字符少于 *nMax*-1 个，则换行符会存储在缓冲区中。 在任一情况下都会追加一个空字符 ('\0')。

[CFile::Read](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#read) 也可用于文本模式输入，但它不会以回车换行符对终止。

**备注**

此函数的 CString 版本会在 '\n' 存在时移除它；LPTSTR 版本则不会。

### 示例

C++

CStdioFile f(stdin);

TCHAR buf[100];

f.ReadString(buf, 99);

## CStdioFile::Seek

将指针重新定位在以前打开的文件中。

复制

virtual ULONGLONG Seek(

LONGLONG lOff,

UINT nFrom);

### 参数

*lOff*  
要将指针移动的字节数。

*nFrom*  
指针移动模式。 必须是以下值之一：

* CFile::begin：将文件指针从文件开头向前移动 *lOff* 个字节。
* CFile::current：将文件指针从文件中的当前位置移动 *lOff* 个字节。
* CFile::end：将文件指针从文件末尾移动 *lOff* 个字节。 请注意，*lOff* 必须为负值才能搜索现有文件；正值将在文件末尾后面搜索。

### 返回值

如果请求的位置合法，则 Seek 返回新字节相对于文件开头的偏移量。 否则，返回值未定义，会引发 CFileException 对象。

### 备注

Seek 函数允许通过将指针绝对或相对移动指定的量来随机访问文件的内容。 在搜索期间，实际上不会读取任何数据。 如果请求的位置大于文件大小，则文件长度会延长到该位置，不会引发异常。

打开文件时，文件指针位于偏移量 0 处，即文件的开头。

Seek 的此实现基于运行时库 (CRT) 函数 fseek。 对文本模式下打开的流使用 Seek 时有几个限制。 有关详细信息，请参阅 [fseek、 \_fseeki64](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/fseek-fseeki64?view=msvc-170)。

### 示例

下面的示例演示如何使用 Seek 将指针从 cfile 文件开头移动 1000 个字节。 请注意，Seek 不读取数据，因此必须随后调用 [CStdioFile::ReadString](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cstdiofile-class?view=msvc-170#readstring) 以读取数据。

C++

CStdioFile cfile(\_T("Stdio\_Seek\_File.dat"), CFile::modeWrite |

CFile::modeCreate);

LONGLONG lOff = 1000;

ULONGLONG lActual = cfile.Seek(lOff, CFile::begin);

## CStdioFile::WriteString

将缓冲区中的数据写入与 CStdioFile 对象关联的文件。

virtual void WriteString(LPCTSTR lpsz);

### 参数

*lpsz*  
指定指向包含以 null 结尾的字符串的缓冲区的指针。

### 备注

终止空字符 (\0) 不会写入文件。 此方法会将 *lpsz* 中的换行符作为回车-换行符对写入文件中。

如果要将不是以 null 结尾的数据写入文件，请使用 CStdioFile::Write 或 [CFile::Write](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#write)。

如果为 *lpsz* 参数指定 NULL，此方法会引发 CInvalidArgException\*。

此方法会引发 CFileException\* 以响应文件系统错误。

### 示例

C++复制

CStdioFile f(stdout);

TCHAR buf[] = \_T("test string");

f.WriteString(buf);

## 另请参阅

[CFile 类](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170)  
[层次结构图](https://learn.microsoft.com/zh-cn/cpp/mfc/hierarchy-chart?view=msvc-170)  
[CFile 类](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170)  
[CFile::Duplicate](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#duplicate)  
[CFile::LockRange](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#lockrange)  
[CFile::UnlockRange](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cfile-class?view=msvc-170#unlockrange)  
[CNotSupportedException 类](https://learn.microsoft.com/zh-cn/cpp/mfc/reference/cnotsupportedexception-class?view=msvc-170)

# 演练

## 1.创建一个MFC应用程序项目，取名: Lesson15-4way-file-op选择基于对话框的应用程序，点击完成

|  |
| --- |
|  |

## 2.设计界面，把原来的对话框上面的控件删除，然后添加一些按钮和一个文本框，把文本框拉大，在属性面板中，把Multiline属性设置为true，把Want Return设置为true，把Vertical scroll也设置为true

|  |
| --- |
|  |

## 3.然后修改按钮的ID和按钮文字

|  |
| --- |
|  |

## 4.为每一个按钮添加点击事件处理代码

|  |
| --- |
|  |

## 5.我们先来做c语言的，先读取一个叫做harry.txt的文本文件，在MFC的程序里面，你可以使用c或者c++的函数，也可以使用win32的函数，也可以都使用。没有问题不过你必须知道这么使用。c语言读取文件的代码如下：

|  |
| --- |
| void CLesson154wayfileopDlg::OnBnClickedBtnCWay()  {  // TODO: 在此添加控件通知处理程序代码  //0.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  SetDlgItemText(IDC\_EDIT1,\_T(""));  FILE \*pFile;  int i, numread;  char line[256];  //1.打开文件  pFile = fopen("harry.txt","r");  CString content(\_T("\*\*\*\*\*\*\*\*\*C\*\*\*\*\*\*\*\*\*\r\n"));    //2.读取文件内容，需要使用循环  while(fgets(line,256,pFile)!=NULL)  {  //3.将读取到的每一行添加到content中  content +=line;  content +=\_T("\r\n");  }  //4.读取完毕需要关闭文件句柄  fclose(pFile);  //5.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  } |

## 6.然后我们来做c++的方式，注意需要包含<fstream>

|  |
| --- |
| void CLesson154wayfileopDlg::OnBnClickedBtnCppWay()  {  // TODO: 在此添加控件通知处理程序代码  char line[256];  //1.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  SetDlgItemText(IDC\_EDIT1,\_T(""));  CString content(\_T("\*\*\*\*\*\*\*\*\*CPP Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //2.创建一个文件输入流对象  ifstream ifs("harry.txt");  //3.读取文件，需要利用循环，把读取到的每一行添加到content中  while(ifs.getline(line,256)!=NULL)  {  content += line;  content +=\_T("\r\n");  }  //4.关闭我流  ifs.close();  //5.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  } |

## 7.Win32方式的代码如下，注意需要先清空字符数组使用ZeroMemory函数，然后在显示内容之前需要把“\n“替换为”\r\n”,还有，ReadFile函数会把文件内容一次全部读完，需要定义大一点的字符数组。

|  |
| --- |
| void CLesson154wayfileopDlg::OnBnClickedBtnWin32Way()  {  // TODO: 在此添加控件通知处理程序代码  //win32方式可以一次把整个文件读完。  HANDLE hFile;  CHAR szBuffer[4096];  DWORD dwRead;  ZeroMemory(szBuffer,4096);  //1.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  SetDlgItemText(IDC\_EDIT1,\_T(""));  CString content(\_T("\*\*\*\*\*\*\*\*\*Win32 Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //2.打开文件，使用CreateFile函数  hFile = CreateFile(\_T("harry.txt"),GENERIC\_READ,FILE\_SHARE\_READ,NULL,  OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(\_T("Error Creating File"));  }  //3.读取文件，使用ReadFile函数  ReadFile(hFile,szBuffer,4096,&dwRead,NULL);  content +=szBuffer;  content +=\_T("\r\n");  //content.Replace(\_T("\n"),\_T("\r\n"));  //4.关闭文件句柄  CloseHandle(hFile);  //5.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  } |

## 8.下面我们来使用MFC的文件相关类来读取文件，这里使用CStdioFile类

|  |
| --- |
| void CLesson154wayfileopDlg::OnBnClickedBtnMfcWay()  {  // TODO: 在此添加控件通知处理程序代码  CString line; //保存每一次读取的一行文本的变量  //1.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  //SetDlgItemText(IDC\_EDIT1,\_T(""));  CString content(\_T("\*\*\*\*\*\*\*\*\*MFC Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //2.读取文件，使用CStdioFile类，需要循环  CStdioFile file(\_T("harry.txt"),CFile::modeRead);  while(file.ReadString(line))  {  content +=line;  content +=\_T("\r\n");  }  //3.关闭文件句柄  file.Close();    //4.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  } |

# 这一节的学习到此为止，Lesson15-4way-file-opDlg.cpp的完整代码如下

|  |
| --- |
| // Lesson15-4way-file-opDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "Lesson15-4way-file-op.h"  #include "Lesson15-4way-file-opDlg.h"  #include<fstream>  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  using namespace std;  // 用于应用程序“关于”菜单项的CAboutDlg 对话框  class CAboutDlg : public CDialog  {  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() : CDialog(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialog::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialog)  END\_MESSAGE\_MAP()  // CLesson154wayfileopDlg 对话框  CLesson154wayfileopDlg::CLesson154wayfileopDlg(CWnd\* pParent /\*=NULL\*/)  : CDialog(CLesson154wayfileopDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  }  void CLesson154wayfileopDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialog::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_EDIT1, m\_edit);  }  BEGIN\_MESSAGE\_MAP(CLesson154wayfileopDlg, CDialog)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  //}}AFX\_MSG\_MAP  ON\_BN\_CLICKED(IDC\_BTN\_C\_WAY, &CLesson154wayfileopDlg::OnBnClickedBtnCWay)  ON\_BN\_CLICKED(IDC\_BTN\_CPP\_WAY, &CLesson154wayfileopDlg::OnBnClickedBtnCppWay)  ON\_BN\_CLICKED(IDC\_BTN\_WIN32\_WAY, &CLesson154wayfileopDlg::OnBnClickedBtnWin32Way)  ON\_BN\_CLICKED(IDC\_BTN\_MFC\_WAY, &CLesson154wayfileopDlg::OnBnClickedBtnMfcWay)  END\_MESSAGE\_MAP()  // CLesson154wayfileopDlg 消息处理程序  BOOL CLesson154wayfileopDlg::OnInitDialog()  {  CDialog::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)  {  CString strAboutMenu;  strAboutMenu.LoadString(IDS\_ABOUTBOX);  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 CLesson154wayfileopDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialog::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 CLesson154wayfileopDlg::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  {  CDialog::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 CLesson154wayfileopDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  void CLesson154wayfileopDlg::OnBnClickedBtnCWay()  {  // TODO: 在此添加控件通知处理程序代码  //0.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  //SetDlgItemText(IDC\_EDIT1,\_T(""));  FILE \*pFile;  char line[256];  //1.打开文件  pFile = fopen("harry.txt","r");  CString content(\_T("\*\*\*\*\*\*\*\*\*C\*\*\*\*\*\*\*\*\*\r\n"));    //2.读取文件内容，需要使用循环  while(fgets(line,256,pFile)!=NULL)  {  //3.将读取到的每一行添加到content中  content +=line;  content +=\_T("\r\n");  }  //4.读取完毕需要关闭文件句柄  fclose(pFile);  //5.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  }  void CLesson154wayfileopDlg::OnBnClickedBtnCppWay()  {  // TODO: 在此添加控件通知处理程序代码  char line[256];  //1.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  //SetDlgItemText(IDC\_EDIT1,\_T(""));  CString content(\_T("\*\*\*\*\*\*\*\*\*CPP Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //2.创建一个文件输入流对象  ifstream ifs("harry.txt");  //3.读取文件，需要利用循环，把读取到的每一行添加到content中  while(ifs.getline(line,256)!=NULL)  {  content += line;  content +=\_T("\r\n");  }  //4.关闭我流  ifs.close();  //5.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  }  void CLesson154wayfileopDlg::OnBnClickedBtnWin32Way()  {  // TODO: 在此添加控件通知处理程序代码  //win32方式可以一次把整个文件读完。  HANDLE hFile;  CHAR szBuffer[4096];  DWORD dwRead;  ZeroMemory(szBuffer,4096);  //1.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  //SetDlgItemText(IDC\_EDIT1,\_T(""));  CString content(\_T("\*\*\*\*\*\*\*\*\*Win32 Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //2.打开文件，使用CreateFile函数  hFile = CreateFile(\_T("harry.txt"),GENERIC\_READ,FILE\_SHARE\_READ,NULL,  OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(\_T("Error Creating File"));  }  //3.读取文件，使用ReadFile函数  ReadFile(hFile,szBuffer,4096,&dwRead,NULL);  content +=szBuffer;  content +=\_T("\r\n");  //content.Replace(\_T("\n"),\_T("\r\n"));  //4.关闭文件句柄  CloseHandle(hFile);  //5.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  }  void CLesson154wayfileopDlg::OnBnClickedBtnMfcWay()  {  // TODO: 在此添加控件通知处理程序代码  CString line; //保存每一次读取的一行文本的变量  //1.先清空编辑框内容  //m\_edit.SetWindowTextW(\_T(""));  //SetDlgItemText(IDC\_EDIT1,\_T(""));  CString content(\_T("\*\*\*\*\*\*\*\*\*MFC Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //2.读取文件，使用CStdioFile类，需要循环  CStdioFile file(\_T("harry.txt"),CFile::modeRead);  while(file.ReadString(line))  {  content +=line;  content +=\_T("\r\n");  }  //3.关闭文件句柄  file.Close();    //4.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1,content); //也可以这么写  } |

# 扩展:windows平台项目读取中文文档的4种方式

## 1.新建一个MFC对话框应用程序,把默认控件删除,然后修改对话框的布局如下

|  |
| --- |
|  |
| #define IDC\_C 1000  #define IDC\_CPP 1001  #define IDC\_WIN32 1002  #define IDC\_C4 1003  #define IDC\_MFC 1003  #define IDC\_EDIT1 1004 |

## 2.双击c语言方式按钮,进入点击事件处理函数,代码如下

|  |
| --- |
| void CLesson15fourWayfileopDlg::OnBnClickedC()  {  // TODO: 在此添加控件通知处理程序代码  //setlocale(LC\_ALL, "");  WCHAR szContent[MAX\_PATH];  CString line(\_T("\*\*\*\*\*\*\*\*\*C Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  FILE\* pf;  memset(szContent, 0, MAX\_PATH);  //\_wfopen\_s(&pf, L"testc.txt", L"r,ccs=utf-8");  \_wfopen\_s(&pf, L"test.txt", L"r,ccs=utf-8");  while (fgetws(szContent, MAX\_PATH, pf))  {  line += szContent;  memset(szContent, 0, MAX\_PATH);  }  SetDlgItemText(IDC\_EDIT1, line);  fclose(pf);  } |

### 注意,c语言方式,需要包含头文件<locale.h>,而且需要在构造函数里面设置地域编码

|  |
| --- |
|  |
|  |

## 3.下面是cpp方式的读中文

|  |
| --- |
| void CLesson15fourWayfileopDlg::OnBnClickedCpp()  {  // TODO: 在此添加控件通知处理程序代码  WCHAR line[256];    CString content(\_T("\*\*\*\*\*\*\*\*\*CPP Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //.创建一个文件输入流对象  wifstream ifs("testcpp.txt");  ifs.imbue(locale(".UTF-8"));//C++设置支持中文//需要include<fstream> 和using namespace std  //.读取文件，需要利用循环，把读取到的每一行添加到content中  while (ifs.getline(line, 256))  {  content += line;  content += \_T("\r\n");  }  //.关闭我流  ifs.close();  //.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1, content); //也可以这么写  } |

### 注意:cpp的设置中文编码的方式: ifs.imbue(locale(".UTF-8"))

## 4.下面是win32读取中文的方式

|  |
| --- |
| void CLesson15fourWayfileopDlg::OnBnClickedWin32()  {  // TODO: 在此添加控件通知处理程序代码  setlocale(LC\_ALL, "utf-8");  HANDLE hFile;  CHAR szBuf[bufsize];  WCHAR\* szLine;  DWORD len;  DWORD dwRead;  CString cnt("------------------------Win32 Way Reading-------------------------\r\n");  memset(szBuf, 0, bufsize);  hFile = CreateFile(L"testwin32.txt", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  //hFile = CreateFile(L"testwin32-2.txt", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  //hFile = CreateFile(L"testmfc.txt", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE)  {  MessageBox(L"创建文件失败");  CloseHandle(hFile);  return;  }  while (ReadFile(hFile, szBuf, bufsize, &dwRead, NULL) && dwRead >0)  {  len = MultiByteToWideChar(CP\_UTF8, 0, szBuf, -1, 0, NULL);  szLine = new WCHAR[len];  MultiByteToWideChar(CP\_UTF8, 0, szBuf, -1, szLine, len);  cnt += szLine;  memset(szBuf, 0, bufsize);    }  SetDlgItemText(IDC\_EDIT1, cnt);  CloseHandle(hFile);  } |

## 注意:win32需要调用2次MultiByteToWideChar函数才能够完成字符转换.

## 5.下面是MFC读取中文的代码,注意这里需要使用CFile,因为CStdioFile对中文支持很不好.

|  |
| --- |
| void CLesson15fourWayfileopDlg::OnBnClickedMfc()  {  // TODO: 在此添加控件通知处理程序代码  CString line; //保存每一次读取的一行文本的变量  CHAR szRead[bufsize];//使用ansi的方式读取  memset(szRead, 0, bufsize);  CString content(\_T("\*\*\*\*\*\*\*\*\*MFC Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //读取文件，使用CStdioFile类，需要循环  CFile file;  file.Open(L"test.txt", CFile::modeRead);  //CStdioFile file(\_T("testwin32.txt"), CFile::modeRead);  while (file.Read(szRead,bufsize)>0)  {  line = CA2W(szRead,CP\_UTF8);//把ansi转化为UTF-8编码  content += line;  content += \_T("\r\n");  }  //3.关闭文件句柄  file.Close();  //4.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1, content); //也可以这么写  } |

### MFC倒比较简单,只需要一个CA2W函数,就可以完成转换

# 扩展2,写入中文:添加另外一个对话框,实现4种方式的写入

## 1.把文件读取对话框资源复制一份,ID改为IDD\_DIALOG\_WRITE,然后把布局改动一下如图

|  |
| --- |
|  |

## 2.在这个对话框的上面点击右键->添加类,弹出添加类对话框

|  |
| --- |
|  |
|  |

## 3.检查一下对话框的ID是否是新对话框的id,如果是,添加类名,基类需要的对话框类,然后点击确定,这样子就生成了一个和这个对话框相关联的类

|  |
| --- |
|  |

## 4.点击项目-类向导,下载CDialogWrite类,重写虚函数PreTranslateMessage,屏蔽回车键

|  |
| --- |
|  |

## 5.别忘了在读取对话框里面添加一个按钮来弹出写对话框

|  |
| --- |
|  |

## 6.给这个按钮添加点击事件代码,在这里弹出写对话框,注意需要在读取对话框的头文件里面包含写对话框的头文件

|  |
| --- |
|  |
|  |

## 7.回到写对话框类,先给他添加一个成员变量m\_str

|  |
| --- |
|  |

## 8.然后在对话框的cpp文件里面定义一个判空函数用来判断编辑框的内容是否为空需要,需要定义为成员函数

|  |
| --- |
|  |

## 9. 然后在写对话框的cpp文件里面实现他的代码

|  |
| --- |
| BOOL CDialogWrite::IsEditCtrlEmpty()  {  GetDlgItemText(IDC\_EDIT1, m\_str);  if (m\_str.GetLength() == 0) {  return TRUE;  }  return FALSE;  } |

### 给CDialogWrite的编辑框控件绑定一个CEdit类的成员变量m\_edit

|  |
| --- |
|  |

## 10.然后我们这个c语言方式按钮的代码如下

|  |
| --- |
| void CDialogWrite::OnBnClickedC()  {  // TODO: 在此添加控件通知处理程序代码  FILE\* fp;  if (IsEditCtrlEmpty())  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  //MessageBox(m\_str);//如果不报错,就获取到数据了  m\_str += L"\n";  //\_wfopen\_s(&fp, L"cwrite.txt", L"w+,ccs=utf-8");//覆盖内容  //\_wfopen\_s(&fp, L"cwrite.txt", L"a+,ccs=utf-8");  \_wfopen\_s(&fp, L"飞鸟和蝉.txt", L"a+,ccs=utf-8");  fputws(m\_str, fp);  fflush(fp);  fclose(fp);  MessageBox(L"文件写入成功");  } |

## 测试一下.运行程序,弹出写入对话框,然后在编辑框里面添加内容

|  |
| --- |
|  |

### 点击c语言方式按钮,效果如下

|  |
| --- |
|  |

### 然后我们发现生成了飞鸟和蝉.txt,内容如下

|  |
| --- |
|  |

## 11.下面来看cpp按钮的代码,需要先包含fstream头文件

|  |
| --- |
|  |

## 12.接着来写cpp按钮代码

|  |
| --- |
| void CDialogWrite::OnBnClickedCpp()  {  // TODO: 在此添加控件通知处理程序代码  if (IsEditCtrlEmpty())//这个我们我们的成员函数用来判断编辑框是否是空的  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  ////如果不报错,就获取到数据了  wofstream wfs;  wfs.imbue(locale(".UTF-8"));  wfs.open(L"吻别cpp.txt",ios::out);  wfs.seekp(0,ios::beg);  wfs.write(m\_str, m\_str.GetLength() + 1);  wfs.flush();  wfs.close();  MessageBox(L"文件写入成功");  } |

### 测试一下,运行程序,弹出写对话框,输入吻别歌词,点击cpp方式按钮,效果如下

|  |
| --- |
|  |

### 然后到cpp源文件所在目录下面,发现生成了一个吻别cpp.txt文件,里面是歌词就是我们输入的内容

|  |
| --- |
|  |

## 13,接着是win32方式

|  |  |
| --- | --- |
| void CDialogWrite::OnBnClickedWin32()  {  // TODO: 在此添加控件通知处理程序代码  HANDLE hFile;  DWORD dwWrite;  PCHAR szContent;  DWORD dwSize;  if (IsEditCtrlEmpty())//这个我们我们的成员函数用来判断编辑框是否是空的  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  //代码运行到这里,说明已经拿到数据  hFile = CreateFile(L"不想长大win32.txt", GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(L"创建文件失败");  CloseHandle(hFile);  return;  }  dwSize = WideCharToMultiByte(CP\_OEMCP, 0, m\_str, -1,NULL ,0, NULL,NULL);  szContent = new CHAR[dwSize];  WideCharToMultiByte(CP\_OEMCP, 0, m\_str, -1, szContent, dwSize, NULL, NULL);  if (WriteFile(hFile, szContent, dwSize, &dwWrite, NULL) && dwSize ==dwWrite)  {  MessageBox(L"写入文件成功!");  }  else  {  MessageBox(L"写入文件失败!");  }  CloseHandle(hFile);  delete[] szContent;  } |  |

### 测试,运行程序,弹出写入对话框,输入歌词,点击win32方式按钮,效果如下

|  |
| --- |
|  |

### 点击确定,然后发现生成了一个:不想长大win32.txt,打开一看,内容就是我们的歌词,不过他是ansi字符编码.其实 也没有关系.

|  |
| --- |
|  |

## 14,最后,我们来实现一下MFC写中文

|  |
| --- |
|  |

### 测试:运行程序,弹出写入对话框,输入潇洒走一回歌词,点击MFC方式按钮,效果如下

|  |
| --- |
|  |

### 点击确定,然后仓库运行cpp源代码目录,发现生成了一个潇洒走一回mfc.txt文件,打开一看,就是我们的歌词

|  |
| --- |
|  |

## 对话框类的完整代码如下

### Lesson15fourWayfileopDlg.h

|  |
| --- |
| // Lesson15fourWayfileopDlg.h: 头文件  //  #pragma once  #include"CDialogWrite.h" //包含写入对话框的头文件  // CLesson15fourWayfileopDlg 对话框  class CLesson15fourWayfileopDlg : public CDialogEx  {  // 构造  public:  CLesson15fourWayfileopDlg(CWnd\* pParent = nullptr); // 标准构造函数  // 对话框数据  #ifdef AFX\_DESIGN\_TIME  enum { IDD = IDD\_LESSON15FOURWAYFILEOP\_DIALOG };  #endif  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);  afx\_msg void OnBnClickedC();  afx\_msg void OnBnClickedCpp();  afx\_msg void OnBnClickedWin32();  afx\_msg void OnBnClickedMfc();  afx\_msg void OnBnClickedBtnWriteDlg();  }; |

### Lesson15fourWayfileopDlg.cpp

|  |
| --- |
| // Lesson15fourWayfileopDlg.cpp: 实现文件  //  #include "pch.h"  #include "framework.h"  #include "Lesson15fourWayfileop.h"  #include "Lesson15fourWayfileopDlg.h"  #include "afxdialogex.h"  #include<locale.h> //c语言显示中文  #include<fstream>  using namespace std;  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  #define bufsize 8704 //不能太小,太小又乱码  // 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  #ifdef AFX\_DESIGN\_TIME  enum { IDD = IDD\_ABOUTBOX };  #endif  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(IDD\_ABOUTBOX)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CLesson15fourWayfileopDlg 对话框  CLesson15fourWayfileopDlg::CLesson15fourWayfileopDlg(CWnd\* pParent /\*=nullptr\*/)  : CDialogEx(IDD\_LESSON15FOURWAYFILEOP\_DIALOG, pParent)  {  setlocale(LC\_ALL, "");  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);    }  void CLesson15fourWayfileopDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CLesson15fourWayfileopDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()  ON\_BN\_CLICKED(IDC\_C, &CLesson15fourWayfileopDlg::OnBnClickedC)  ON\_BN\_CLICKED(IDC\_CPP, &CLesson15fourWayfileopDlg::OnBnClickedCpp)  ON\_BN\_CLICKED(IDC\_WIN32, &CLesson15fourWayfileopDlg::OnBnClickedWin32)  ON\_BN\_CLICKED(IDC\_MFC, &CLesson15fourWayfileopDlg::OnBnClickedMfc)  ON\_BN\_CLICKED(IDC\_BTN\_WRITE\_DLG, &CLesson15fourWayfileopDlg::OnBnClickedBtnWriteDlg)  END\_MESSAGE\_MAP()  // CLesson15fourWayfileopDlg 消息处理程序  BOOL CLesson15fourWayfileopDlg::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 != nullptr)  {  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 CLesson15fourWayfileopDlg::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 CLesson15fourWayfileopDlg::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 CLesson15fourWayfileopDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  BOOL CLesson15fourWayfileopDlg::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:  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  void CLesson15fourWayfileopDlg::OnBnClickedC()  {  // TODO: 在此添加控件通知处理程序代码  //setlocale(LC\_ALL, "");  WCHAR szContent[MAX\_PATH];  CString line(\_T("\*\*\*\*\*\*\*\*\*C Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  FILE\* pf;  memset(szContent, 0, MAX\_PATH);  //\_wfopen\_s(&pf, L"testc.txt", L"r,ccs=utf-8");  \_wfopen\_s(&pf, L"test.txt", L"r,ccs=utf-8");  while (fgetws(szContent, MAX\_PATH, pf))  {  line += szContent;  memset(szContent, 0, MAX\_PATH);  }  SetDlgItemText(IDC\_EDIT1, line);  fclose(pf);  }  void CLesson15fourWayfileopDlg::OnBnClickedCpp()  {  // TODO: 在此添加控件通知处理程序代码  WCHAR line[256];  CString content(\_T("\*\*\*\*\*\*\*\*\*CPP Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //创建一个文件输入流对象  wifstream ifs("testcpp.txt");  ifs.imbue(locale(".UTF-8"));//C++设置支持中文//需要include<fstream> 和using namespace std  //读取文件，需要利用循环，把读取到的每一行添加到content中  while (ifs.getline(line, 256))  {  content += line;  content += \_T("\r\n");  }  //关闭我流  ifs.close();  //显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1, content); //也可以这么写  }  void CLesson15fourWayfileopDlg::OnBnClickedWin32()  {  // TODO: 在此添加控件通知处理程序代码  setlocale(LC\_ALL, "utf-8");  HANDLE hFile;  CHAR szBuf[bufsize];  WCHAR\* szLine;  DWORD len;  DWORD dwRead;  CString cnt("------------------------Win32 Way Reading-------------------------\r\n");  memset(szBuf, 0, bufsize);  hFile = CreateFile(L"testwin32.txt", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  //hFile = CreateFile(L"testwin32-2.txt", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  //hFile = CreateFile(L"testmfc.txt", GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (hFile == INVALID\_HANDLE\_VALUE)  {  MessageBox(L"创建文件失败");  CloseHandle(hFile);  return;  }  while (ReadFile(hFile, szBuf, bufsize, &dwRead, NULL) && dwRead >0)  {  len = MultiByteToWideChar(CP\_UTF8, 0, szBuf, -1, 0, NULL);  szLine = new WCHAR[len];  MultiByteToWideChar(CP\_UTF8, 0, szBuf, -1, szLine, len);  cnt += szLine;  memset(szBuf, 0, bufsize);    }  SetDlgItemText(IDC\_EDIT1, cnt);  CloseHandle(hFile);  }  void CLesson15fourWayfileopDlg::OnBnClickedMfc()  {  // TODO: 在此添加控件通知处理程序代码  CString line; //保存每一次读取的一行文本的变量  CHAR szRead[bufsize];//使用ansi的方式读取  memset(szRead, 0, bufsize);  CString content(\_T("\*\*\*\*\*\*\*\*\*MFC Way Of Read File\*\*\*\*\*\*\*\*\*\r\n"));  //读取文件，使用CStdioFile类，需要循环  CFile file;  file.Open(L"test.txt", CFile::modeRead);  //CStdioFile file(\_T("testwin32.txt"), CFile::modeRead);  while (file.Read(szRead,bufsize)>0)  {  line = CA2W(szRead,CP\_UTF8);//把ansi转化为UTF-8编码  content += line;  content += \_T("\r\n");  }  //3.关闭文件句柄  file.Close();  //4.显示内容到编辑框  //m\_edit.SetWindowTextW(content);//可以这么写  SetDlgItemText(IDC\_EDIT1, content); //也可以这么写  }  void CLesson15fourWayfileopDlg::OnBnClickedBtnWriteDlg()  {  // TODO: 在此添加控件通知处理程序代码  CDialogWrite dlg;  dlg.DoModal();//弹出写对话框  } |

### CDialogWrite.h

|  |
| --- |
| #pragma once  // CDialogWrite 对话框  class CDialogWrite : public CDialogEx  {  DECLARE\_DYNAMIC(CDialogWrite)  public:  CDialogWrite(CWnd\* pParent = nullptr); // 标准构造函数  virtual ~CDialogWrite();  BOOL IsEditCtrlEmpty();//判断编辑框是否为空的成员函数  // 对话框数据  #ifdef AFX\_DESIGN\_TIME  enum { IDD = IDD\_DIALOG\_WRITE };  #endif  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  DECLARE\_MESSAGE\_MAP()  public:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  afx\_msg void OnBnClickedC();  private:  CString m\_str;  public:  // //绑定编辑框控件的成员变量  CEdit m\_edit;  afx\_msg void OnBnClickedCpp();  afx\_msg void OnBnClickedWin32();  afx\_msg void OnBnClickedMfc();  }; |

### CDialogWrite.cpp

|  |
| --- |
| // CDialogWrite.cpp: 实现文件  //  #include "pch.h"  #include "Lesson15fourWayfileop.h"  #include "CDialogWrite.h"  #include "afxdialogex.h"  #include<fstream>  using namespace std;  // CDialogWrite 对话框  IMPLEMENT\_DYNAMIC(CDialogWrite, CDialogEx)  CDialogWrite::CDialogWrite(CWnd\* pParent /\*=nullptr\*/)  : CDialogEx(IDD\_DIALOG\_WRITE, pParent)  {  m\_str = \_T("");  }  CDialogWrite::~CDialogWrite()  {  }  void CDialogWrite::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  DDX\_Control(pDX, IDC\_EDIT1, m\_edit);  }  BEGIN\_MESSAGE\_MAP(CDialogWrite, CDialogEx)  ON\_BN\_CLICKED(IDC\_C, &CDialogWrite::OnBnClickedC)  ON\_BN\_CLICKED(IDC\_CPP, &CDialogWrite::OnBnClickedCpp)  ON\_BN\_CLICKED(IDC\_WIN32, &CDialogWrite::OnBnClickedWin32)  ON\_BN\_CLICKED(IDC\_MFC, &CDialogWrite::OnBnClickedMfc)  END\_MESSAGE\_MAP()  // CDialogWrite 消息处理程序  BOOL CDialogWrite::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:  case VK\_ESCAPE:  return TRUE;  }  }  return CDialogEx::PreTranslateMessage(pMsg);  }  BOOL CDialogWrite::IsEditCtrlEmpty()  {  GetDlgItemText(IDC\_EDIT1, m\_str);  if (m\_str.GetLength() == 0) {  return TRUE;  }  return FALSE;  }  void CDialogWrite::OnBnClickedC()  {  // TODO: 在此添加控件通知处理程序代码  FILE\* fp;  if (IsEditCtrlEmpty())  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  //MessageBox(m\_str);//如果不报错,就获取到数据了  m\_str += L"\n";  //\_wfopen\_s(&fp, L"cwrite.txt", L"w+,ccs=utf-8");//覆盖内容  //\_wfopen\_s(&fp, L"cwrite.txt", L"a+,ccs=utf-8");  \_wfopen\_s(&fp, L"飞鸟和蝉.txt", L"a+,ccs=utf-8");  fputws(m\_str, fp);  fflush(fp);  fclose(fp);  MessageBox(L"文件写入成功");  }  void CDialogWrite::OnBnClickedCpp()  {  // TODO: 在此添加控件通知处理程序代码  if (IsEditCtrlEmpty())//这个我们我们的成员函数用来判断编辑框是否是空的  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  ////如果不报错,就获取到数据了  wofstream wfs;  wfs.imbue(locale(".UTF-8"));  wfs.open(L"吻别cpp.txt",ios::out);  wfs.seekp(0,ios::beg);  wfs.write(m\_str, m\_str.GetLength() + 1);  wfs.flush();  wfs.close();  MessageBox(L"文件写入成功");  }  void CDialogWrite::OnBnClickedWin32()  {  // TODO: 在此添加控件通知处理程序代码  HANDLE hFile;  DWORD dwWrite;  PCHAR szContent;  DWORD dwSize;  if (IsEditCtrlEmpty())//这个我们我们的成员函数用来判断编辑框是否是空的  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  //代码运行到这里,说明已经拿到数据  hFile = CreateFile(L"不想长大win32.txt", GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);  if (INVALID\_HANDLE\_VALUE == hFile)  {  MessageBox(L"创建文件失败");  CloseHandle(hFile);  return;  }  dwSize = WideCharToMultiByte(CP\_OEMCP, 0, m\_str, -1,NULL ,0, NULL,NULL);  szContent = new CHAR[dwSize];  WideCharToMultiByte(CP\_OEMCP, 0, m\_str, -1, szContent, dwSize, NULL, NULL);  if (WriteFile(hFile, szContent, dwSize, &dwWrite, NULL) && dwSize ==dwWrite)  {  MessageBox(L"写入文件成功!");  }  else  {  MessageBox(L"写入文件失败!");  }  CloseHandle(hFile);  delete[] szContent;  }  void CDialogWrite::OnBnClickedMfc()  {  // TODO: 在此添加控件通知处理程序代码  PCHAR szContent;  DWORD dwSize;  if (IsEditCtrlEmpty())//这个我们我们的成员函数用来判断编辑框是否是空的  {  MessageBox(L"请输入内容");  m\_edit.SetFocus();  return;  }  ////代码运行到这里,说明已经拿到数据  CFile file;  file.Open(L"潇洒走一回mfc.txt", CFile::modeCreate | CFile::modeWrite);  dwSize = WideCharToMultiByte(CP\_OEMCP, 0, m\_str, -1, NULL, 0, NULL, NULL);  szContent = new CHAR[dwSize];  WideCharToMultiByte(CP\_OEMCP, 0, m\_str, -1, szContent, dwSize, NULL, NULL);  file.Write(szContent, dwSize);  file.Close();  MessageBox(L"写入文件成功!");  } |

# 这一节的学习到此为止,主要扩充了读写中文的代码,以后复习的时候如果有新的经验或者想法,会继续扩充笔记