# 学习大纲

|  |
| --- |
|  |
|  |

# 三种内存分配机制

# 堆函数,是最常用的内存分配方法.他在内部其实是调用虚拟内存函数来操作内存的.微软有一个堆管理器.可以在我们调用堆函数时帮我们处理内存分页,页面对齐,页面属性和页面粘贴等等的问题.如果直接使用虚拟内存函数,这些都需要我们自己处理,是比较繁琐的.堆适合管理大量的小型数据的场合.

# 相关函数的用法

## HeapAlloc函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从堆中分配内存块。 分配的内存不可移动。 语法 C++  DECLSPEC\_ALLOCATOR LPVOID HeapAlloc(  [in] HANDLE hHeap,  [in] DWORD dwFlags,  [in] SIZE\_T dwBytes  ); 参数 [in] hHeap  要从中分配内存的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] dwFlags  堆分配选项。 指定这些值中的任何一个都将替代使用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 创建堆时指定的相应值。 此参数可使用以下一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HEAP\_GENERATE\_EXCEPTIONS**  0x00000004 | 系统将引发异常以指示函数失败（例如内存不足情况），而不是返回 **NULL**。  若要确保为此函数的所有调用生成异常，请在调用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 时指定**HEAP\_GENERATE\_EXCEPTIONS**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_GENERATE\_EXCEPTIONS** 。 | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 序列化访问将不用于此分配。  有关详细信息，请参阅“备注”。  若要确保禁用对此函数的所有调用的序列化访问，请在调用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 中指定**HEAP\_NO\_SERIALIZE**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_NO\_SERIALIZE** 。  访问进程的默认堆时，不应指定此值。 系统可能会在应用程序的进程中创建其他线程，例如同时访问进程的默认堆的 CTRL+C 处理程序。 | | **HEAP\_ZERO\_MEMORY**  0x00000008 | 分配的内存将初始化为零。 否则，内存不会初始化为零。 |   [in] dwBytes  要分配的字节数。  如果 *hHeap* 参数指定的堆是“不可增长的”堆， *则 dwBytes* 必须小于 0x7FFF8。 可以通过使用非零值调用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数来创建不可增长的堆。 返回值 如果函数成功，则返回值是指向已分配内存块的指针。  如果函数失败并且您尚未指定 **HEAP\_GENERATE\_EXCEPTIONS**，则返回值为 **NULL**。  如果函数失败并且已指定 **HEAP\_GENERATE\_EXCEPTIONS**，则函数可能会生成下表中列出的任一异常。 特定例外取决于堆损坏的性质。 有关详细信息，请参阅 [GetExceptionCode](https://learn.microsoft.com/zh-cn/windows/desktop/Debug/getexceptioncode)。  展开表   |  |  | | --- | --- | | **异常代码** | **说明** | | **STATUS\_NO\_MEMORY** | 由于缺少可用内存或堆损坏，分配尝试失败。 | | **STATUS\_ACCESS\_VIOLATION** | 由于堆损坏或函数参数不正确，分配尝试失败。 |   如果函数失败，则它不会调用 [SetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-setlasterror)。 应用程序无法调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 以获取扩展错误信息。 注解 如果 **HeapAlloc** 函数成功，它将分配至少请求的内存量。  若要从进程的默认堆分配内存，请将 **HeapAlloc** 与 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回的句柄一起使用。  若要释放 **HeapAlloc** 分配的内存块，请使用 [HeapFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapfree) 函数。  **HeapAlloc** 分配的内存不可移动。 **HeapAlloc** 返回的地址在释放或重新分配内存块之前有效;内存块不需要锁定。 由于系统无法压缩专用堆，因此它可能会碎片化。  **HeapAlloc** 返回的内存对齐方式在 WinNT.h 中**MEMORY\_ALLOCATION\_ALIGNMENT**：  C++复制  #if defined(\_WIN64) || defined(\_M\_ALPHA)  #define MEMORY\_ALLOCATION\_ALIGNMENT 16  #else  #define MEMORY\_ALLOCATION\_ALIGNMENT 8  #endif  以各种分配大小分配大量内存的应用程序可以使用 [低碎片堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/low-fragmentation-heap) 来减少堆碎片。  当两个或多个线程尝试从同一堆同时分配或释放块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一个堆分配和释放内存时，必须使用它。 设置 **HEAP\_NO\_SERIALIZE** 值可消除堆上的相互排斥。 如果不进行序列化，使用同一堆句柄的两个或多个线程可能会尝试同时分配或释放内存，这可能会导致堆损坏。 因此，只能在以下情况下安全地使用 **HEAP\_NO\_SERIALIZE** 值：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，应用程序提供自己的机制，用于对特定堆进行相互排斥。  示例 有关示例，请参阅 [AWE 示例](https://learn.microsoft.com/zh-cn/windows/win32/Memory/awe-example)。 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapFree](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapfree)  [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heaprealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## HeapFree函数的用法

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 释放由 [HeapAlloc 或 HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 函数从堆分配的内存块。 语法 C++  BOOL HeapFree(  [in] HANDLE hHeap,  [in] DWORD dwFlags,  [in] \_Frees\_ptr\_opt\_ LPVOID lpMem  ); 参数 [in] hHeap  要释放其内存块的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] dwFlags  堆免费选项。 使用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数创建堆时，指定以下值将替代 *flOptions* 参数中指定的相应值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 不会使用序列化访问。 有关详细信息，请参阅“备注”。  若要确保对此函数的所有调用禁用序列化访问，请在对 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 的调用中指定**HEAP\_NO\_SERIALIZE**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_NO\_SERIALIZE** 。  访问进程堆时不要指定此值。 系统可能会在应用程序的进程中创建其他线程，例如同时访问进程堆的 CTRL+C 处理程序。 |   [in] lpMem  指向要释放的内存的指针。 此指针由 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 或 [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaprealloc) 函数返回。 此指针可以为 **NULL**。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 应用程序可以调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 以获取扩展错误信息。 注解 不应以任何方式引用已由 **HeapFree** 释放的内存。 释放该内存后，可能已存在的任何信息将永远消失。 如果需要信息，请不要释放包含该信息的内存。 返回有关内存 (（例如 [HeapSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapsize)) ）信息的函数调用可能不会与释放的内存一起使用，因为它们可能会返回虚假数据。 使用同一指针调用 **HeapFree** 两次可能会导致堆损坏，导致对 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 的后续调用返回同一指针两次。  当两个或多个线程尝试从同一堆同时分配或释放块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一堆分配和释放内存时，必须使用它。 设置 **HEAP\_NO\_SERIALIZE** 值可消除堆上的相互排斥。 如果不进行序列化，使用同一堆句柄的两个或多个线程可能会尝试同时分配或释放内存，这可能会导致堆损坏。 因此，只能在以下情况下安全地使用 **HEAP\_NO\_SERIALIZE** 值：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，应用程序为特定堆提供自身的相互排斥机制。  示例 有关示例，请参阅 [获取进程堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/getting-process-heaps)。 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapalloc)  [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heaprealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## GetProcessHeap函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索调用进程的默认堆的句柄。 然后，可以在对堆函数的后续调用中使用此句柄。 语法 C++复制  HANDLE GetProcessHeap(); 返回值 如果函数成功，则返回值是调用进程的堆的句柄。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 **GetProcessHeap** 函数获取调用进程的默认堆的句柄。 进程可以使用此句柄从进程堆分配内存，而无需先使用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapcreate) 函数创建专用堆。  **Windows Server 2003 和 Windows XP：**若要为进程的默认堆启用低碎片堆，请使用 **GetProcessHeap** 返回的句柄调用 [HeapSetInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapsetinformation) 函数。 示例 有关示例，请参阅 [获取进程堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/getting-process-heaps)。 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapcreate)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## GetProcessHeaps函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 返回活动堆的数量，并检索调用进程的所有活动堆的句柄。 语法 C++复制  DWORD GetProcessHeaps(  [in] DWORD NumberOfHeaps,  [out] PHANDLE ProcessHeaps  ); 参数 [in] NumberOfHeaps  可以存储在 *ProcessHeaps* 指向的缓冲区中的堆句柄的最大数目。  [out] ProcessHeaps  指向接收堆句柄数组的缓冲区的指针。 返回值 返回值是调用进程处于活动状态的堆的句柄数。  如果返回值小于或等于 *NumberOfHeaps*，则函数已将堆句柄数存储在 *ProcessHeaps* 指向的缓冲区中。  如果返回值大于 *NumberOfHeaps*， *则 ProcessHeaps* 指向的缓冲区太小，无法容纳调用进程的所有堆句柄，并且函数将 *NumberOfHeaps 句柄* 存储在缓冲区中。 使用返回值分配一个足以接收所有句柄的缓冲区，然后再次调用函数。  如果返回值为零，则函数失败，因为每个进程至少有一个活动堆，即进程的默认堆。 要获得更多的错误信息，请调用 GetLastError。 注解 **GetProcessHeaps** 函数获取调用进程的默认堆的句柄，以及通过在进程中的任何线程上调用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数创建的任何其他专用堆的句柄。  **GetProcessHeaps** 函数主要用于调试，因为函数检索到的某些专用堆可能是由进程中运行的其他代码创建的，并且可能会在 **GetProcessHeaps** 返回后被销毁。 销毁堆会使堆的句柄失效，而继续使用此类句柄可能会导致应用程序中出现未定义的行为。 只应在调用进程的默认堆和进程创建和管理的专用堆上调用堆函数。  若要获取调用进程的进程堆的句柄，请使用 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-getprocessheap) 函数。 示例 有关示例，请参阅 [获取进程堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/getting-process-heaps)。 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-getprocessheap)  [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapcreate)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

### GetProcessHeaps函数有2个用途,1是获取一个进程一共使用了多少个堆(此时第一个参数是0,第二个参数是NULL),2是可以获取每一个堆的句柄,此时参数不能为0或者NULL

## SizeTMult函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 将 **size\_t** 类型的一个值乘以另一个值。 语法 C++复制  HRESULT SizeTMult(  [in] size\_t Multiplicand,  [in] size\_t Multiplier,  [out] size\_t \*pResult  ); 参数 [in] Multiplicand  类型： **size\_t**  要乘以 *cbMultiplier* 的值。  [in] Multiplier  类型： **size\_t**  要乘以 *cbMultiplicand* 的值。  [out] pResult  类型： **size\_t\***  指向结果的指针。 如果操作导致类型的容量溢出或不足的值，则函数将返回INTSAFE\_E\_ARITHMETIC\_OVERFLOW且此参数无效。 返回值 类型： **HRESULT**  如果此函数成功，则返回 **S\_OK**。 否则，将返回 HRESULT 错误代码。 注解 这是一组内联函数之一，旨在提供算术运算和执行有效性检查，同时尽量减少对性能的影响。 要求 展开表   | **要求** | **值** | | --- | --- | | **目标平台** | Windows | | **标头** | intsafe.h | |

## 实例

### 获取进程堆

此示例演示如何使用 [**GetProcessHeaps**](https://learn.microsoft.com/zh-cn/windows/desktop/api/HeapApi/nf-heapapi-getprocessheaps) 函数检索默认进程堆的句柄，以及当前进程处于活动状态的任何专用堆。

该示例调用 [**GetProcessHeaps**](https://learn.microsoft.com/zh-cn/windows/desktop/api/HeapApi/nf-heapapi-getprocessheaps) 两次，第一次是计算所需的缓冲区大小，然后是检索缓冲区中的句柄。 使用 [**GetProcessHeap**](https://learn.microsoft.com/zh-cn/windows/desktop/api/HeapApi/nf-heapapi-getprocessheap) 返回的句柄从默认进程堆分配缓冲区。 该示例将每个堆的起始地址打印到控制台。 然后，它使用 [**HeapFree**](https://learn.microsoft.com/zh-cn/windows/desktop/api/HeapApi/nf-heapapi-heapfree) 函数释放为缓冲区分配的内存。

进程中的堆数可能会有所不同。 进程始终至少有一个堆（默认进程堆），并且它可能有一个或多个由应用程序或加载到进程的地址空间中的 DLL 创建的专用堆。

请注意，应用程序应仅在其默认进程堆或应用程序创建的专用堆上调用堆函数;在另一个组件拥有的专用堆上调用堆函数可能会导致未定义的行为。

C++

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

#include <intsafe.h>

int \_\_cdecl \_tmain()

{

DWORD NumberOfHeaps;

DWORD HeapsIndex;

DWORD HeapsLength;

HANDLE hDefaultProcessHeap;

HRESULT Result;

PHANDLE aHeaps;

SIZE\_T BytesToAllocate;

//

// Retrieve the number of active heaps for the current process

// so we can calculate the buffer size needed for the heap handles.

//

NumberOfHeaps = GetProcessHeaps(0, NULL);

if (NumberOfHeaps == 0) {

\_tprintf(TEXT("Failed to retrieve the number of heaps with LastError %d.\n"),

GetLastError());

return 1;

}

//

// Calculate the buffer size.

//

Result = SIZETMult(NumberOfHeaps, sizeof(\*aHeaps), &BytesToAllocate);

if (Result != S\_OK) {

\_tprintf(TEXT("SIZETMult failed with HR %d.\n"), Result);

return 1;

}

//

// Get a handle to the default process heap.

//

hDefaultProcessHeap = GetProcessHeap();

if (hDefaultProcessHeap == NULL) {

\_tprintf(TEXT("Failed to retrieve the default process heap with LastError %d.\n"),

GetLastError());

return 1;

}

//

// Allocate the buffer from the default process heap.

//

aHeaps = (PHANDLE)HeapAlloc(hDefaultProcessHeap, 0, BytesToAllocate);

if (aHeaps == NULL) {

\_tprintf(TEXT("HeapAlloc failed to allocate %d bytes.\n"),

BytesToAllocate);

return 1;

}

//

// Save the original number of heaps because we are going to compare it

// to the return value of the next GetProcessHeaps call.

//

HeapsLength = NumberOfHeaps;

//

// Retrieve handles to the process heaps and print them to stdout.

// Note that heap functions should be called only on the default heap of the process

// or on private heaps that your component creates by calling HeapCreate.

//

NumberOfHeaps = GetProcessHeaps(HeapsLength, aHeaps);

if (NumberOfHeaps == 0) {

\_tprintf(TEXT("Failed to retrieve heaps with LastError %d.\n"),

GetLastError());

return 1;

}

else if (NumberOfHeaps > HeapsLength) {

//

// Compare the latest number of heaps with the original number of heaps.

// If the latest number is larger than the original number, another

// component has created a new heap and the buffer is too small.

//

\_tprintf(TEXT("Another component created a heap between calls. ") \

TEXT("Please try again.\n"));

return 1;

}

\_tprintf(TEXT("Process has %d heaps.\n"), HeapsLength);

for (HeapsIndex = 0; HeapsIndex < HeapsLength; ++HeapsIndex) {

\_tprintf(TEXT("Heap %d at address: %#p.\n"),

HeapsIndex,

aHeaps[HeapsIndex]);

}

//

// Release memory allocated from default process heap.

//

if (HeapFree(hDefaultProcessHeap, 0, aHeaps) == FALSE) {

\_tprintf(TEXT("Failed to free allocation from default process heap.\n"));

}

return 0;

}

## HeapCreate函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 创建可由调用进程使用的专用堆对象。 函数在进程的虚拟地址空间中保留空间，并为此块的指定初始部分分配物理存储。 语法 C++复制  HANDLE HeapCreate(  [in] DWORD flOptions,  [in] SIZE\_T dwInitialSize,  [in] SIZE\_T dwMaximumSize  ); 参数 [in] flOptions  堆分配选项。 这些选项通过调用堆函数影响对新堆的后续访问。 此参数可以是 0 或以下一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HEAP\_CREATE\_ENABLE\_EXECUTE**  0x00040000 | 如果硬件强制实施 [数据执行防护](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/data-execution-prevention)，则从此堆分配的所有内存块都允许代码执行。 在从堆运行代码的应用程序中使用此标志堆。 如果未指定 **HEAP\_CREATE\_ENABLE\_EXECUTE** ，并且应用程序尝试从受保护的页面运行代码，则应用程序将收到异常，状态代码 **STATUS\_ACCESS\_VIOLATION**。 | | **HEAP\_GENERATE\_EXCEPTIONS**  0x00000004 | 系统引发异常以指示失败 (例如，内存不足条件) 调用 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 和 [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaprealloc) 而不是返回 **NULL**。 | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 当堆函数访问此堆时，不使用序列化访问。 此选项适用于所有后续堆函数调用。 或者，可以在单个堆函数调用上指定此选项。  无法为使用此选项创建的堆启用低碎片化堆 (LFH) 。  不能锁定使用此选项创建的堆。  有关序列化访问的详细信息，请参阅本主题的“备注”部分。 |   [in] dwInitialSize  堆的初始大小（以字节为单位）。 此值确定为堆提交的初始内存量。 该值向上舍入为系统页面大小的倍数。 该值必须小于 *dwMaximumSize*。  如果此参数为 0，则函数将提交一页。 若要确定主计算机上的页面大小，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  [in] dwMaximumSize  堆的最大大小（以字节为单位）。 **HeapCreate** 函数将 *dwMaximumSize* 舍入到系统页大小的倍数，然后在堆的进程虚拟地址空间中保留该大小的块。 如果 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 或 [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaprealloc) 函数发出的分配请求超过 *dwInitialSize* 指定的大小，系统会为堆提交额外的内存页，最大大小为堆的最大大小。  如果 *dwMaximumSize* 不为零，则堆大小是固定的，并且不能增长到超过最大大小。 此外，对于 32 位进程，可从堆中分配的最大内存块略小于 512 KB，而对于 64 位进程，则略低于 1,024 KB。 即使堆的最大大小足以包含块，分配较大块的请求也会失败。  如果 *dwMaximumSize* 为 0，堆大小可能会增大。 堆的大小仅受可用内存的限制。 分配大于固定大小堆限制的内存块的请求不会自动失败;相反，系统会调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数来获取大型块所需的内存。 需要分配大量内存块的应用程序应将 *dwMaximumSize* 设置为 0。 返回值 如果函数成功，则返回值是新创建的堆的句柄。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 **HeapCreate** 函数创建一个专用堆对象，调用进程可以使用 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 函数从该对象分配内存块。 初始大小确定最初为堆分配的已提交页数。 最大大小确定保留页的总数。 这些页面在进程的虚拟地址空间中创建一个块，堆可以增长到其中。 如果 **HeapAlloc** 的请求超过已提交页面的当前大小，则如果物理存储可用，则会自动从此预留空间提交其他页面。  **Windows Server 2003 和 Windows XP：**默认情况下，新创建的专用堆是标准堆。 若要启用低碎片堆，请使用专用堆的句柄调用 [HeapSetInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapsetinformation) 函数。  专用堆对象的内存只能由创建它的进程访问。 如果动态链接库 (DLL) 创建专用堆，则会在调用 DLL 的进程地址空间中创建堆，并且只有该进程才能访问该堆。  系统使用专用堆中的内存来存储堆支持结构，因此并非所有指定的堆大小都可供进程使用。 例如，如果 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 函数从最大大小为 64K 的堆请求 64 KB (K) ，则请求可能会因系统开销而失败。  **如果未在**简单默认) (指定HEAP\_NO\_SERIALIZE，则堆会在调用过程中序列化访问。 当两个或多个线程同时尝试从同一堆分配或释放块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一个堆分配和释放内存时，必须使用它。 [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock) 和 [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapunlock) 函数可用于阻止和允许访问序列化堆。  设置 **HEAP\_NO\_SERIALIZE** 会消除堆上的相互排斥。 如果不进行序列化，使用同一堆句柄的两个或多个线程可能会尝试同时分配或释放内存，这可能会导致堆损坏。 因此， **只能在** 以下情况下安全地使用HEAP\_NO\_SERIALIZE：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，应用程序提供自己的机制，用于对特定堆进行相互排斥。   如果在使用 **HEAP\_NO\_SERIALIZE** 标志创建的堆上调用 [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock) 和 [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapunlock) 函数，则结果未定义。  若要获取进程的默认堆的句柄，请使用 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数。 若要获取对调用进程处于活动状态的默认堆和专用堆的句柄，请使用 [GetProcessHeaps](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheaps) 函数。 示例 [枚举堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/enumerating-a-heap) 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapalloc)  [HeapDestroy](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapdestroy)  [HeapValidate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapvalidate)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

### flOptions常量

|  |
| --- |
| #define HEAP\_NO\_SERIALIZE 0x00000001  #define HEAP\_GROWABLE 0x00000002  #define HEAP\_GENERATE\_EXCEPTIONS 0x00000004  #define HEAP\_ZERO\_MEMORY 0x00000008  #define HEAP\_REALLOC\_IN\_PLACE\_ONLY 0x00000010  #define HEAP\_TAIL\_CHECKING\_ENABLED 0x00000020  #define HEAP\_FREE\_CHECKING\_ENABLED 0x00000040  #define HEAP\_DISABLE\_COALESCE\_ON\_FREE 0x00000080  #define HEAP\_CREATE\_ALIGN\_16 0x00010000  #define HEAP\_CREATE\_ENABLE\_TRACING 0x00020000  #define HEAP\_CREATE\_ENABLE\_EXECUTE 0x00040000  #define HEAP\_MAXIMUM\_TAG 0x0FFF  #define HEAP\_PSEUDO\_TAG\_FLAG 0x8000  #define HEAP\_TAG\_SHIFT 18 |

### 注意创建堆只是创建了一共对象,还没有分配内存,创建后可以使用HeapAlloc或者HeapReAlloc函数来分配内存.此时只能使用win32Api而不能使用calloc和malloc函数

## HeapCompact 函数

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 返回指定堆中最大提交的可用块的大小。 如果已设置 [“在免费](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/debugger/disable-heap-coalesce-on-free) 全局标志上禁用堆合并”，则此函数还会合并堆中相邻的可用内存块。  **语法**  C++复制  SIZE\_T HeapCompact(  [in] HANDLE hHeap,  [in] DWORD dwFlags  );  **参数**  [in] hHeap  堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] dwFlags  堆访问选项。 此参数可以是以下值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 不会使用序列化访问。  有关详细信息，请参阅“备注”。  若要确保所有对此函数的调用都禁用序列化访问，请在调用 heapCreate中指定 HEAP\_NO\_SERIALIZE。 在这种情况下，不需要在此函数调用中额外指定 **HEAP\_NO\_SERIALIZE**。  访问进程堆时不要指定此值。 系统可能会在应用程序的进程（如 Ctrl+C 处理程序）中创建其他线程，以便同时访问进程堆。 |   **返回值**  如果函数成功，则返回值是堆中提交的最大可用块的大小（以字节为单位）。  如果函数失败，则返回值为零。 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)。  在堆中绝对没有可用空间的情况下，函数返回值为零，[GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回值NO\_ERROR。  **言论**  **HeapCompact** 函数主要用于调试。 通常，每当调用 [HeapFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapfree) 函数时，系统会压缩堆，**HeapCompact** 函数将返回堆中最大可用块的大小，但不进一步压缩堆。 如果在调试期间设置了免费 全局标志上的 禁用堆合并，则系统不会压缩堆，并且调用 HeapCompact 函数会压缩堆。 有关全局标志的详细信息，请参阅 [GFlags](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/debugger/gflags) 文档。  不能保证应用程序可以成功分配由 **HeapCompact**返回大小的内存块。 其他线程或提交阈值可能会阻止此类分配。  当两个或多个线程尝试同时分配或释放同一堆中的块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一堆中分配和释放内存时，都必须使用它。 设置 **HEAP\_NO\_SERIALIZE** 值可消除堆上的相互排斥。 如果没有序列化，使用同一堆句柄的两个或多个线程可能会同时尝试分配或释放内存，这可能会导致堆损坏。 因此，**HEAP\_NO\_SERIALIZE** 值只能在以下情况下安全地使用：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，并且应用程序提供自己的机制来相互排除到特定堆。   **要求**   |  |  | | --- | --- | | **标头** | heapapi.h （包括 Windows.h） | | **库** | Kernel32.lib | | **DLL** | Kernel32.dll |   **另请参阅**  [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapcreate)  [HeapValidate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapvalidate)  [内存管理功能](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) 中提供的 Vertdll API |

## HeapDestroy函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 销毁指定的堆对象。  **HeapDestroy** 会取消提交并释放私有堆对象的所有页面，并使堆的句柄失效。 语法 C++复制  BOOL HeapDestroy(  [in] HANDLE hHeap  ); 参数 [in] hHeap  要销毁的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数返回。 请勿使用 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回的进程堆的句柄。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 进程可以调用 **HeapDestroy** ，而无需先调用 [HeapFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapfree) 函数来释放从堆中分配的内存。 示例 [枚举堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/enumerating-a-heap) 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapcreate)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## HeapLock函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 尝试获取与指定堆关联的关键节对象或锁。 语法 C++复制  BOOL HeapLock(  [in] HANDLE hHeap  ); 参数 [in] hHeap  要锁定的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。 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 如果函数成功，则调用线程拥有堆锁。 只有调用线程才能从堆中分配或释放内存。 如果该线程尝试从堆分配或释放内存，则将阻止调用进程的任何其他线程的执行。 此类线程将一直被阻止，直到拥有堆锁的线程调用 [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapunlock) 函数。  **HeapLock** 函数主要用于防止其他线程在调用线程使用 [HeapWalk](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapwalk) 函数时分配和释放堆内存。  如果在使用 [HEAP\_NO\_SERIALIZE](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 标志创建的堆上调用 **HeapLock** 函数，则结果未定义。  每次成功调用 **HeapLock** 都必须与对 [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapunlock) 的相应调用匹配。 调用 **HeapUnlock** 失败将阻止尝试访问堆的调用进程的任何其他线程的执行。 示例 [枚举堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/enumerating-a-heap) 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapunlock)  [HeapWalk](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapwalk)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## HeapQueryInformation 函数的用法

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索有关指定堆的信息。 语法 C++复制  BOOL HeapQueryInformation(  [in, optional] HANDLE HeapHandle,  [in] HEAP\_INFORMATION\_CLASS HeapInformationClass,  [out] PVOID HeapInformation,  [in] SIZE\_T HeapInformationLength,  [out, optional] PSIZE\_T ReturnLength  ); 参数 [in, optional] HeapHandle  要检索其信息的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] HeapInformationClass  要检索的信息类。 此参数可以是 **HEAP\_INFORMATION\_CLASS** 枚举类型的以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HeapCompatibilityInformation**  0 | 指示已启用的堆功能。  *HeapInformation* 参数是指向 **ULONG** 变量的指针。  如果 *HeapInformation* 为 0，则堆是不支持旁观列表的标准堆。  如果 *HeapInformation* 为 1，则堆支持旁观列表。 有关详细信息，请参阅“备注”。  如果 *HeapInformation* 为 2，则为 [堆启用了低碎片堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/low-fragmentation-heap) (LFH) 。 启用 LFH 会禁用旁观列表。 |   [out] HeapInformation  指向接收堆信息的缓冲区的指针。 此数据的格式取决于 *HeapInformationClass* 参数的值。  [in] HeapInformationLength  正在查询的堆信息的大小（以字节为单位）。  [out, optional] ReturnLength  指向变量的指针，该变量接收写入到 *HeapInformation* 缓冲区的数据的长度。 如果缓冲区太小，函数将失败， *ReturnLength* 指定缓冲区所需的最小大小。  如果不想接收此信息，请指定 **NULL**。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 若要启用 LFH 或损坏时终止功能，请使用 [HeapSetInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapsetinformation) 函数。  **Windows XP 和 Windows Server 2003：**旁观列表是一种仅包含固定大小的块的快速内存分配机制。 默认情况下，为支持它们的堆启用旁观列表。 从 Windows Vista 开始，不使用旁观列表，并且默认启用 LFH。  旁观列表比大小不同的常规池分配更快，因为系统不会搜索适合该分配的可用内存。 此外，通常使用快速原子处理器交换指令（而不是互斥或自转锁）来同步对旁观列表的访问。 旁观列表可以由系统或驱动程序创建。 可以从分页池或非分页池中分配它们。 示例 以下示例使用 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 获取默认进程堆的句柄，并使用 **HeapQueryInformation** 检索有关堆的信息。  C++复制  #include <windows.h>  #include <tchar.h>  #include <stdio.h>  #define HEAP\_STANDARD 0  #define HEAP\_LAL 1  #define HEAP\_LFH 2  int \_\_cdecl \_tmain()  {  BOOL bResult;  HANDLE hHeap;  ULONG HeapInformation;  //  // Get a handle to the default process heap.  //  hHeap = GetProcessHeap();  if (hHeap == NULL) {  \_tprintf(TEXT("Failed to retrieve default process heap with LastError %d.\n"),  GetLastError());  return 1;  }  //  // Query heap features that are enabled.  //  bResult = HeapQueryInformation(hHeap,  HeapCompatibilityInformation,  &HeapInformation,  sizeof(HeapInformation),  NULL);  if (bResult == FALSE) {  \_tprintf(TEXT("Failed to retrieve heap features with LastError %d.\n"),  GetLastError());  return 1;  }  //  // Print results of the query.  //  \_tprintf(TEXT("HeapCompatibilityInformation is %d.\n"), HeapInformation);  switch(HeapInformation)  {  case HEAP\_STANDARD:  \_tprintf(TEXT("The default process heap is a standard heap.\n"));  break;  case HEAP\_LAL:  \_tprintf(TEXT("The default process heap supports look-aside lists.\n"));  break;  case HEAP\_LFH:  \_tprintf(TEXT("The default process heap has the low-fragmentation ") \  TEXT("heap enabled.\n"));  break;  default:  \_tprintf(TEXT("Unrecognized HeapInformation reported for the default ") \  TEXT("process heap.\n"));  break;  }  return 0;  } 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap)  [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate)  [HeapSetInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapsetinformation)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## HeapReAlloc函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从堆中重新分配内存块。 使用此函数可以调整内存块的大小并更改其他内存块属性。 分配的内存不可移动。  **语法**  C++复制  DECLSPEC\_ALLOCATOR LPVOID HeapReAlloc(  [in] HANDLE hHeap,  [in] DWORD dwFlags,  [in] \_Frees\_ptr\_opt\_ LPVOID lpMem,  [in] SIZE\_T dwBytes  );  **参数**  [in] hHeap  要从中重新分配内存的堆的句柄。 此句柄是由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回的 。  [in] dwFlags  堆重新分配选项。 使用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数创建堆时，指定值将替代 *flOptions* 参数中指定的相应值。 此参数可使用以下一个或多个值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HEAP\_GENERATE\_EXCEPTIONS**  0x00000004 | 操作系统引发异常以指示函数失败（例如内存不足的情况），而不是返回 **NULL**。  若要确保为此函数的所有调用生成异常，请在对 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 的调用中指定**HEAP\_GENERATE\_EXCEPTIONS**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_GENERATE\_EXCEPTIONS** 。 | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 不会使用序列化访问。 有关详细信息，请参阅“备注”。  若要确保对此函数的所有调用禁用序列化访问，请在对 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 的调用中指定**HEAP\_NO\_SERIALIZE**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_NO\_SERIALIZE** 。  访问进程堆时不应指定此值。 系统可能会在应用程序的进程中创建其他线程，例如同时访问进程堆的 CTRL+C 处理程序。 | | **HEAP\_REALLOC\_IN\_PLACE\_ONLY**  0x00000010 | 重新分配内存块时不能移动。 如果未指定此值，该函数可能会将块移动到新位置。 如果指定了此值，并且无法在不移动的情况下调整块的大小，则函数将失败，使原始内存块保持不变。 | | **HEAP\_ZERO\_MEMORY**  0x00000008 | 如果重新分配请求的大小更大，则超出原始大小的其他内存区域将初始化为零。 内存块的内容与其原始大小一起不受影响。 |   [in] lpMem  指向函数重新分配的内存块的指针。 此指针由先前对 [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 或 **HeapReAlloc** 函数的调用返回。  [in] dwBytes  内存块的新大小（以字节为单位）。 使用此函数可以增加或减小内存块的大小。  如果 *hHeap* 参数指定的堆是“不可增长”堆， *则 dwBytes* 必须小于 0x7FFF8。 可以通过调用具有非零值的 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数来创建不可增长的堆。  **返回值**  如果函数成功，则返回值是指向重新分配的内存块的指针。  如果函数失败，并且您尚未指定 **HEAP\_GENERATE\_EXCEPTIONS**，则返回值为 **NULL**。  如果函数失败，并且已指定 **HEAP\_GENERATE\_EXCEPTIONS**，该函数可能会生成下表中列出的任一异常。 有关详细信息，请参阅 [GetExceptionCode](https://learn.microsoft.com/zh-cn/windows/desktop/Debug/getexceptioncode)。  展开表   |  |  | | --- | --- | | **异常代码** | **说明** | | **STATUS\_NO\_MEMORY** | 由于缺少可用内存或堆损坏，分配尝试失败。 | | **STATUS\_ACCESS\_VIOLATION** | 由于堆损坏或函数参数不正确，分配尝试失败。 |   **HeapReAlloc** 返回的内存对齐方式在 WinNT.h **中MEMORY\_ALLOCATION\_ALIGNMENT**：  C++复制  #if defined(\_WIN64) || defined(\_M\_ALPHA)  #define MEMORY\_ALLOCATION\_ALIGNMENT 16  #else  #define MEMORY\_ALLOCATION\_ALIGNMENT 8  #endif  如果函数失败，则不调用 [SetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-setlasterror)。 应用程序无法调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 以获取扩展错误信息。  **注解**  如果 **HeapReAlloc** 成功，它将至少分配请求的内存量。  如果 **HeapReAlloc** 失败，则不会释放原始内存，并且原始句柄和指针仍然有效。  可以保证 **HeapReAlloc** 保留要重新分配的内存的内容，即使新内存是在不同的位置分配的。 保留内存内容的过程涉及可能非常耗时的内存复制操作。  若要释放 **HeapReAlloc** 分配的内存块，请使用 [HeapFree](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapfree) 函数。  当两个或多个线程尝试从同一堆同时分配或释放块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一堆分配和释放内存时，必须使用它。 设置 **HEAP\_NO\_SERIALIZE** 值可消除堆上的相互排斥。 如果不进行序列化，使用同一堆句柄的两个或多个线程可能会尝试同时分配或释放内存，这可能会导致堆损坏。 因此，只能在以下情况下安全地使用 **HEAP\_NO\_SERIALIZE** 值：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，应用程序为特定堆提供自身的相互排斥机制。   **要求**   |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |   **另请参阅**  [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapalloc)  [HeapFree](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapfree)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

### 注意:

### 1重新分配的内存返回的指针可能和原来的指针一样,也可能不一样,如果重新分配内存时没有移动堆,这两个指针就是一样的,如果移动了堆,这两个指针就是不一样的.

### 2.如果重新分配的内存大小比堆的最大内存还要大,分配就会失败

## HeapSetInformation函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为指定的堆启用功能。 语法 C++复制  BOOL HeapSetInformation(  [in, optional] HANDLE HeapHandle,  [in] HEAP\_INFORMATION\_CLASS HeapInformationClass,  [in] PVOID HeapInformation,  [in] SIZE\_T HeapInformationLength  ); 参数 [in, optional] HeapHandle  要在其中设置信息的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] HeapInformationClass  要设置的信息类。 此参数可以是 **HEAP\_INFORMATION\_CLASS** 枚举类型的以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HeapCompatibilityInformation**  0 | 启用堆功能。 仅支持 [低碎片堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/low-fragmentation-heap) (LFH) 。 但是，应用程序不需要启用 LFH，因为系统根据需要使用 LFH 来为内存分配请求提供服务。  **Windows XP 和 Windows Server 2003：**默认情况下不启用 LFH。 若要为指定的堆启用 LFH，请将 *HeapInformation* 参数指向的变量设置为 2。 为堆启用 LFH 后，无法禁用它。  不能为使用 **HEAP\_NO\_SERIALIZE** 创建的堆或以固定大小创建的堆启用 LFH。 如果使用 Windows 或 [Microsoft 应用程序验证程序](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/devtest/application-verifier)调试[工具](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/debugger/)中的堆调试工具，则也无法启用 LFH。  当进程在任何调试器下运行时，将自动为进程中的所有堆启用某些堆调试选项。 这些堆调试选项阻止使用 LFH。 若要在调试器下运行时启用低碎片堆，请将 \_NO\_DEBUG\_HEAP 环境变量设置为 1。 | | **HeapEnableTerminationOnCorruption**  1 | 启用损坏时终止功能。 如果堆管理器在进程使用的任何堆中检测到错误，它将调用Windows 错误报告服务并终止进程。  进程启用此功能后，无法禁用此功能。  **Windows Server 2003 和 Windows XP：**在 Windows Vista 和 Windows XP 使用 SP3 之前，不支持此值。 函数成功，但 **HeapEnableTerminationOnCorruption** 值将被忽略。 | | **HeapOptimizeResources**  3 | 如果在将 *HeapHandle* 设置为 NULL 的情况下调用 HeapSetInformation，则进程中具有 [低碎片堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/low-fragmentation-heap) (LFH) 的所有堆都将优化其缓存，并且内存将尽可能取消提交。  如果在 *HeapHandle* 中提供了堆指针，则只会优化该堆。  请注意，必须正确初始化 *HeapInformation* 中传递的HEAP\_OPTIMIZE\_RESOURCES\_INFORMATION结构。  **注意**此值已添加到 Windows 8.1 中。 |   [in] HeapInformation  堆信息缓冲区。 此数据的格式取决于 *HeapInformationClass* 参数的值。  如果 *HeapInformationClass* 参数为 **HeapCompatibilityInformation**，则 *HeapInformation* 参数是指向 **ULONG** 变量的指针。  如果 *HeapInformationClass* 参数为 **HeapEnableTerminationOnCorruption**，*则 HeapInformation* 参数应为 *NULL，HeapInformationLength* 应为 0  [in] HeapInformationLength  *HeapInformation* 缓冲区的大小（以字节为单位）。 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 0（零）。 要获得更多的错误信息，请调用 GetLastError。 注解 若要检索堆的当前设置，请使用 [HeapQueryInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapqueryinformation) 函数。  强烈建议设置 **HeapEnableTerminateOnCorruption** 选项，因为它可以减少应用程序受到利用已损坏堆的安全攻击的风险。 示例 以下示例演示如何启用低碎片堆。  C++复制  #include <windows.h>  #include <tchar.h>  #include <stdio.h>  #define HEAP\_LFH 2  int \_\_cdecl \_tmain()  {  BOOL bResult;  HANDLE hHeap;  ULONG HeapInformation;  //  // Enable heap terminate-on-corruption.  // A correct application can continue to run even if this call fails,  // so it is safe to ignore the return value and call the function as follows:  // (void)HeapSetInformation(NULL, HeapEnableTerminationOnCorruption, NULL, 0);  // If the application requires heap terminate-on-corruption to be enabled,  // check the return value and exit on failure as shown in this example.  //  bResult = HeapSetInformation(NULL,  HeapEnableTerminationOnCorruption,  NULL,  0);  if (bResult != FALSE) {  \_tprintf(TEXT("Heap terminate-on-corruption has been enabled.\n"));  }  else {  \_tprintf(TEXT("Failed to enable heap terminate-on-corruption with LastError %d.\n"),  GetLastError());  return 1;  }  //  // Create a new heap with default parameters.  //  hHeap = HeapCreate(0, 0, 0);  if (hHeap == NULL) {  \_tprintf(TEXT("Failed to create a new heap with LastError %d.\n"),  GetLastError());  return 1;  }  //  // Enable the low-fragmentation heap (LFH). Starting with Windows Vista,  // the LFH is enabled by default but this call does not cause an error.  //  HeapInformation = HEAP\_LFH;  bResult = HeapSetInformation(hHeap,  HeapCompatibilityInformation,  &HeapInformation,  sizeof(HeapInformation));  if (bResult != FALSE) {  \_tprintf(TEXT("The low-fragmentation heap has been enabled.\n"));  }  else {  \_tprintf(TEXT("Failed to enable the low-fragmentation heap with LastError %d.\n"),  GetLastError());  return 1;  }  return 0;  } 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap)  [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate)  [HeapQueryInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapqueryinformation)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## HeapSize函数的用法

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检索由 [HeapAlloc 或 HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 函数从堆分配的内存块的大小。  **语法**  C++复制  SIZE\_T HeapSize(  [in] HANDLE hHeap,  [in] DWORD dwFlags,  [in] LPCVOID lpMem  );  **参数**  [in] hHeap  内存块所在的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] dwFlags  堆大小选项。 使用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 函数创建堆时，指定以下值将替代 *flOptions* 参数中指定的相应值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 不会使用序列化访问。 有关详细信息，请参阅“备注”。  若要确保禁用对此函数的所有调用的序列化访问，请在调用 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 中指定**HEAP\_NO\_SERIALIZE**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_NO\_SERIALIZE** 。  访问进程堆时，不应指定此值。 系统可能会在应用程序的进程中创建同时访问进程堆的其他线程，例如 CTRL+C 处理程序。 |   [in] lpMem  指向函数将获取其大小的内存块的指针。 这是 [由 HeapAlloc 或 HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapalloc) 函数返回 [的](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaprealloc) 指针。 内存块必须来自 *hHeap* 参数指定的堆。  **返回值**  如果函数成功，则返回值是请求的已分配内存块的大小（以字节为单位）。  如果该函数失败，则返回值为 (SIZE\_T)-1。 函数不调用 [SetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-setlasterror)。 应用程序无法调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 以获取扩展错误信息。  如果 *lpMem* 参数引用的堆分配不在 *hHeap* 参数指定的堆中，则 **HeapSize** 函数的行为未定义。  **注解**  当两个或多个线程尝试从同一堆同时分配或释放块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一个堆分配和释放内存时，必须使用它。 设置 **HEAP\_NO\_SERIALIZE** 值可消除堆上的相互排斥。 如果不进行序列化，使用同一堆句柄的两个或多个线程可能会尝试同时分配或释放内存，这可能会导致堆损坏。 因此，只能在以下情况下安全地使用 **HEAP\_NO\_SERIALIZE** 值：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，应用程序提供自己的机制，用于对特定堆进行相互排斥。   **要求**   |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |   **另请参阅**  [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapalloc)  [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heaprealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

### 注意:HeapSize函数表示获取堆的大小而是获取堆内存块的大小

## HeapUnlock 函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 释放与指定堆关联的关键节对象或锁的所有权。 它反转 [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock) 函数的操作。 语法 C++复制  BOOL HeapUnlock(  [in] HANDLE hHeap  ); 参数 [in] hHeap  要解锁的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。 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 [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock) 函数主要用于防止在调用线程使用 [HeapWalk](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapwalk) 函数时由其他线程分配和释放堆内存。 **HeapUnlock** 函数是 **HeapLock** 的反函数。  每次对 [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock) 的调用都必须与对 **HeapUnlock** 函数的相应调用相匹配。 调用 **HeapUnlock** 失败将阻止执行尝试访问堆的调用进程的任何其他线程。  如果在使用 [HEAP\_NO\_SERIALIZATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 标志创建的堆上调用 **HeapUnlock** 函数，则结果未定义。 示例 [枚举堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/enumerating-a-heap) 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/heap-functions)  [HeapLock](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heaplock)  [HeapWalk](https://learn.microsoft.com/zh-cn/windows/win32/api/heapapi/nf-heapapi-heapwalk)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## HeapValidate函数的用法

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 验证指定的堆。 函数扫描堆中的所有内存块，并验证堆管理器维护的堆控制结构是否处于一致状态。 还可以使用 **HeapValidate** 函数来验证指定堆中的单个内存块，而无需检查整个堆的有效性。  **语法**  C++复制  BOOL HeapValidate(  [in] HANDLE hHeap,  [in] DWORD dwFlags,  [in, optional] LPCVOID lpMem  );  **参数**  [in] hHeap  要验证的堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in] dwFlags  堆访问选项。 此参数可以是以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **HEAP\_NO\_SERIALIZE**  0x00000001 | 不会使用序列化访问。 有关详细信息，请参阅“备注”。  若要确保对此函数的所有调用禁用序列化访问，请在对 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 的调用中指定**HEAP\_NO\_SERIALIZE**。 在这种情况下，无需在此函数调用中额外指定 **HEAP\_NO\_SERIALIZE** 。  访问进程默认堆时，不应指定此值。 系统可能会在应用程序的进程中创建其他线程，例如同时访问进程默认堆的 CTRL+C 处理程序。 |   [in, optional] lpMem  指向指定堆中的内存块的指针。 此参数可以为 **NULL**。  如果此参数为 **NULL**，则该函数将尝试验证 *hHeap* 指定的整个堆。  如果此参数不为 **NULL**，则该函数将尝试验证 *lpMem* 指向的内存块。 它不会尝试验证堆的其余部分。  **返回值**  如果指定的堆或内存块有效，则返回值为非零值。  如果指定的堆或内存块无效，则返回值为零。 在设置为调试的系统上， **HeapValidate** 函数随后显示调试消息，这些消息描述堆或内存块的无效部分，并在硬编码断点处停止，以便你可以检查系统以确定无效性的来源。 **HeapValidate** 函数不设置线程的最后一个错误值。 此函数没有扩展的错误信息;不要调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。  **注解**  **HeapValidate** 函数主要用于调试，因为验证可能很耗时。 验证堆可能会阻止其他线程访问堆并降低性能，尤其是在对称多处理 (SMP) 计算机上。 这些副作用可能会持续到 **HeapValidate** 返回为止。  堆中的每个内存块以及整个堆都有堆控制结构。 使用 **HeapValidate** 函数验证完整的堆时，它会检查所有这些控制结构的一致性。  使用 **HeapValidate** 验证堆中的单个内存块时，它仅检查与该元素相关的控制结构。 **HeapValidate** 只能验证分配的内存块。 对释放的内存块调用 **HeapValidate** 将返回 **FALSE** ，因为没有要验证的控制结构。  如果要验证 [HeapWalk](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapwalk) 函数枚举的堆元素，则只应对在 PROCESS\_HEAP\_ENTRY 结构的 **wFlags** 成员中**具有PROCESS\_HEAP\_ENTRY\_BUSY**的元素调用 **HeapValidate**。 对于未设置此位的所有堆元素，**HeapValidate** 返回 **FALSE**。  当两个或多个线程尝试从同一堆同时分配或释放块时，序列化可确保相互排斥。 序列化的性能成本很小，但每当多个线程从同一堆分配和释放内存时，必须使用它。 设置 **HEAP\_NO\_SERIALIZE** 值可消除堆上的相互排斥。 如果不进行序列化，使用同一堆句柄的两个或多个线程可能会尝试同时分配或释放内存，这可能会导致堆损坏。 因此，只能在以下情况下安全地使用 **HEAP\_NO\_SERIALIZE** 值：   * 进程只有一个线程。 * 进程有多个线程，但只有一个线程调用特定堆的堆函数。 * 进程具有多个线程，应用程序为特定堆提供自身的相互排斥机制。   **要求**   |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |   **另请参阅**  [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate)  [HeapWalk](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapwalk)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry) |

## HeapWalk 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 枚举指定堆中的内存块。 语法 C++复制  BOOL HeapWalk(  [in] HANDLE hHeap,  [in, out] LPPROCESS\_HEAP\_ENTRY lpEntry  ); parameters [in] hHeap  堆的句柄。 此句柄由 [HeapCreate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapcreate) 或 [GetProcessHeap](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-getprocessheap) 函数返回。  [in, out] lpEntry  指向 [PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry) 结构的指针，该结构维护特定堆枚举的状态信息。  如果 **HeapWalk** 函数成功并返回值 **TRUE**，则此结构的成员包含有关堆中下一个内存块的信息。  若要启动堆枚举，请将 [PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry) 结构的 **lpData** 字段设置为 **NULL**。 若要继续特定的堆枚举，请重复调用 **HeapWalk** 函数，不更改 *hHeap*、 *lpEntry* 或 **PROCESS\_HEAP\_ENTRY** 结构的任何成员。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。  如果堆枚举通过到达堆的末尾成功终止，则函数返回 **FALSE**， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)**ERROR\_NO\_MORE\_ITEMS**返回错误代码。 注解 **HeapWalk** 函数主要用于调试，因为枚举堆可能非常耗时。 在枚举期间锁定堆会阻止其他线程访问堆，并可能降低性能，尤其是在对称多处理 (SMP) 计算机上。 副作用可能会持续到堆解锁。 使用 [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock) 和 [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapunlock) 函数在堆枚举期间控制堆锁定。  若要启动堆枚举，请将 *lpEntry* 指向的 [PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry) 结构的 **lpData** 字段设置为 **NULL**，调用 **HeapWalk**。  若要继续堆枚举，请使用相同的 *hHeap* 和 *lpEntry* 值调用 **HeapWalk**，并且[PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry)结构与前面对 **HeapWalk** 的调用不同。 重复此过程，直到无需进一步枚举，或直到函数返回 **FALSE** 且 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_NO\_MORE\_ITEMS**，指示已枚举所有堆的内存块。  无需特殊调用 **HeapWalk** 即可终止堆枚举，因为 [不会在PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry) 结构的内容之外维护枚举状态数据。  如果在堆枚举期间未锁定堆，**HeapWalk** 在多线程应用程序中可能会失败。 示例 [枚举堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/enumerating-a-heap) 要求  |  |  | | --- | --- | | **标头** | heapapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [HeapLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaplock)  [HeapReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heaprealloc)  [HeapUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapunlock)  [HeapValidate](https://learn.microsoft.com/zh-cn/windows/desktop/api/heapapi/nf-heapapi-heapvalidate)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [PROCESS\_HEAP\_ENTRY](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-process_heap_entry) |

# 虚拟内存函数

## 功能很强大,但是编程笔记复杂,很多细节都需要我们自己处理,适合有很多编程经验的人使用.适合处理大型对象数组或者大型结构数组.

## VirtualAlloc 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留、提交或更改调用进程的虚拟地址空间中页面区域的状态。 此函数分配的内存会自动初始化为零。  若要在另一进程的地址空间中分配内存，请使用 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex) 函数。 语法 C++复制  LPVOID VirtualAlloc(  [in, optional] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD flAllocationType,  [in] DWORD flProtect  ); 参数 [in, optional] lpAddress  要分配的区域的起始地址。 如果保留内存，则指定的地址向下舍入到分配粒度中最近的倍数。 如果内存已保留并正在提交，地址将向下舍入到下一页边界。 若要确定页面的大小和主机计算机上的分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。 如果此参数 **NULL**，系统将确定分配区域的位置。  如果此地址位于未通过调用 [InitializeEnclave](https://learn.microsoft.com/zh-cn/windows/win32/api/enclaveapi/nf-enclaveapi-initializeenclave)初始化的 enclave 中，**VirtualAlloc** 为该地址的 enclave 分配零页。 该页必须以前未提交，并且不会使用 Intel Software Guard Extensions 编程模型的 EEXTEND 指令进行测量。  如果地址位于初始化的 enclave 中，则分配操作将失败，并出现 **ERROR\_INVALID\_ADDRESS** 错误。 对于不支持动态内存管理的 enclave（即 SGX1），这是事实。 SGX2 enclave 将允许分配，并且该页必须在分配后由 enclave 接受。  [in] dwSize  区域的大小（以字节为单位）。 如果 lpAddress 参数 **NULL**，则此值向上舍入到下一页边界。 否则，分配的页面将包含范围中包含一个或多个字节的所有页面，从 lpAddress 到 lpAddress+dwSize。 这意味着跨页边界的 2 字节范围会导致这两个页面都包括在分配的区域。  [in] flAllocationType  内存分配的类型。 此参数必须包含以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_COMMIT**  0x00001000 | 为指定的保留内存页分配内存费用（从内存的总体大小和磁盘上的分页文件）。 该函数还保证当调用方稍后最初访问内存时，内容将为零。 除非实际访问虚拟地址/直到实际访问虚拟地址，否则不会分配实际物理页。  若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 **virtualAlloc**。  除非已保留整个范围，否则尝试通过指定不带 **MEM\_RESERVE** 的 **MEM\_COMMIT** 并**NULL***lpAddress* 来提交特定地址范围。 生成的错误代码 **ERROR\_INVALID\_ADDRESS**。  尝试提交已提交的页面不会导致函数失败。 这意味着可以提交页面，而无需首先确定每个页面的当前承诺状态。  如果 *lpAddress* 指定 enclave 中的地址，则必须 **MEM\_COMMIT***flAllocationType*。 | | **MEM\_RESERVE**  0x00002000 | 保留进程的虚拟地址空间范围，而无需在内存或磁盘上的分页文件中分配任何实际物理存储。  可以在对 **VirtualAlloc** 函数的后续调用中提交保留页。 若要在一个步骤中保留和提交页面，请使用 **MEM\_COMMIT** | **MEM\_RESERVE**调用 **VirtualAlloc**。  其他内存分配函数（如 **malloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)）在释放之前无法使用保留的内存范围。 | | **MEM\_RESET**  0x00080000 | 指示 *lpAddress* 指定的内存区域中的数据不再感兴趣，*dwSize* 不再感兴趣。 不应从分页文件读取或写入页面。 但是，内存块稍后将再次使用，因此不应将其取消提交。 此值不能用于任何其他值。  使用此值不能保证使用 **MEM\_RESET** 操作的范围将包含零。 如果希望该区域包含零，请取消提交内存，然后重新提交内存。  指定 **MEM\_RESET**时，**VirtualAlloc** 函数将忽略 *flProtect*的值。 但是，仍必须将 *flProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **VirtualAlloc** 如果使用 **MEM\_RESET** 并且内存范围映射到文件，则返回错误。 仅当共享视图映射到分页文件时，才可接受。 | | **MEM\_RESET\_UNDO**  0x1000000 | **MEM\_RESET\_UNDO** 只应在之前成功应用 **MEM\_RESET** 的地址范围上调用。 它指示由 *lpAddress 指定的指定内存范围中的数据* 和 *dwSize* 对调用方感兴趣，并尝试扭转 **MEM\_RESET**的影响。 如果函数成功，则表示指定地址范围中的所有数据都保持不变。 如果函数失败，则地址范围中的至少一些数据已替换为零。  此值不能用于任何其他值。 如果在之前未 **MEM\_RESET** 的地址范围上调用 **MEM\_RESET\_UNDO**，则行为是未定义的。 指定 **MEM\_RESET**时，**VirtualAlloc** 函数将忽略 *flProtect*的值。 但是，仍必须将 *flProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows 8 和 Windows Server 2012 之前不支持 **MEM\_RESET\_UNDO** 标志。 |   此参数还可以按指示指定以下值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_LARGE\_PAGES**  0x20000000 | 使用 [大型页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)分配内存。  大小和对齐方式必须是大页最小值的倍数。 若要获取此值，请使用 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数。  如果指定此值，还必须指定 **MEM\_RESERVE** 和 **MEM\_COMMIT**。 | | **MEM\_PHYSICAL**  0x00400000 | 保留可用于映射 [地址窗口扩展插件](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）页的地址范围。  此值必须与 **MEM\_RESERVE** 一起使用，并且不能用于其他值。 | | **MEM\_TOP\_DOWN**  0x00100000 | 以最高可能地址分配内存。 这比常规分配要慢，尤其是在分配很多时。 | | **MEM\_WRITE\_WATCH**  0x00200000 | 使系统跟踪写入到分配区域中的页面。 如果指定此值，还必须指定 **MEM\_RESERVE**。  若要检索自分配区域或重置写入跟踪状态以来已写入的页面的地址，请调用 [GetWriteWatch](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getwritewatch) 函数。 若要重置写入跟踪状态，请调用 **GetWriteWatch** 或 [ResetWriteWatch](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-resetwritewatch)。 写入跟踪功能将一直为内存区域启用，直到释放该区域。 |   [in] flProtect  要分配的页面区域的内存保护。 如果要提交页面，则可以指定内存保护常量之一。  如果 lpAddress 指定 enclave 中的地址，flProtect 不能是以下值之一：   * PAGE\_NOACCESS * PAGE\_GUARD * PAGE\_NOCACHE * PAGE\_WRITECOMBINE   为 enclave 分配动态内存时，flProtect 参数必须 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**。 返回值 如果函数成功，则返回值是页面分配区域的基址。  如果函数失败，则返回值 **NULL**。 若要获取扩展的错误信息，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror)。 言论 每个页面都有一个关联的 [页面状态](https://learn.microsoft.com/zh-cn/windows/win32/Memory/page-state)。 **VirtualAlloc** 函数可以执行以下操作：   * 提交保留页的区域 * 保留免费页面的区域 * 同时保留并提交一个免费页面区域   **VirtualAlloc** 无法保留保留页。 它可以提交已提交的页面。 这意味着你可以提交一系列页面，无论它们是否已提交，并且函数不会失败。  可以使用 **VirtualAlloc** 保留页面块，然后对 virtualAlloc 进行其他调用，以提交保留块中的单个页面。 这样一个进程就可以保留其虚拟地址空间的范围，而无需消耗物理存储，直到需要它。  如果 lpAddress 参数未 **NULL**，则该函数将使用 lpAddress，dwSize 参数来计算要分配的页面区域。 整个页面范围的当前状态必须与 flAllocationType 参数指定的分配类型兼容。 否则，该函数将失败，并且未分配任何页面。 如前所述，此兼容性要求不排除提交已提交的页面。  若要执行动态生成的代码，请使用 **VirtualAlloc** 来分配内存，并使用 [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect) 函数授予 **PAGE\_EXECUTE** 访问权限。  **VirtualAlloc** 函数可用于在指定进程的虚拟地址空间中保留内存 [地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/win32/Memory/address-windowing-extensions)（AWE）区域。 然后，可以使用此内存区域根据应用程序的要求将物理页映射到虚拟内存中和传出虚拟内存。 必须在 AllocationType 参数中设置 **MEM\_PHYSICAL** 和 **MEM\_RESERVE** 值。 不能设置 **MEM\_COMMIT** 值。 页面保护必须设置为 **PAGE\_READWRITE**。  [VirtualFree](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfree) 函数可以取消提交页面、释放页面的存储，也可以同时取消提交和释放已提交的页面。 它还可以释放保留页，使其成为免费页面。  创建可执行的区域时，调用程序负责确保在代码设置到位后通过适当的调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 缓存一致性。 否则，尝试从新可执行区域执行代码可能会产生不可预知的结果。 例子 有关示例，请参阅 [保留和提交内存](https://learn.microsoft.com/zh-cn/windows/win32/Memory/reserving-and-committing-memory)。 另请参阅 [内存管理功能](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/virtual-memory-functions)  [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex)  [VirtualFree](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfree)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtuallock)  [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect)  [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualquery)  [VBS enclave](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) 中提供的 Vertdll API |

## VirtualAlloc2 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留、提交或更改指定进程的虚拟地址空间中内存区域的状态（分配的内存初始化为零）。 语法 C++复制  PVOID VirtualAlloc2(  [in, optional] HANDLE Process,  [in, optional] PVOID BaseAddress,  [in] SIZE\_T Size,  [in] ULONG AllocationType,  [in] ULONG PageProtection,  [in, out, optional] MEM\_EXTENDED\_PARAMETER \*ExtendedParameters,  [in] ULONG ParameterCount  ); 参数 [in, optional] Process  进程的句柄。 该函数在进程的虚拟地址空间中分配内存。  句柄必须具有 **PROCESS\_VM\_OPERATION** 访问权限。 有关详细信息，请参阅 [进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  如果 *进程***NULL**，则该函数将为调用进程分配内存。  [in, optional] BaseAddress  为要分配的页面区域指定所需起始地址的指针。  如果 *BaseAddress***NULL**，则该函数将确定分配区域的位置。  如果 *BaseAddress* 未 **NULL**，则提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零，并且基址必须是系统分配粒度的倍数。 若要确定分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  如果此地址位于未通过调用 [InitializeEnclave](https://learn.microsoft.com/zh-cn/windows/desktop/api/enclaveapi/nf-enclaveapi-initializeenclave)初始化的 enclave 中，**VirtualAlloc2** 为该地址的 enclave 分配零页。 该页必须以前未提交，并且不会使用 Intel Software Guard Extensions 编程模型的 EEXTEND 指令进行测量。  如果初始化的 enclave 中的地址，则分配操作将失败，并出现 **ERROR\_INVALID\_ADDRESS** 错误。 对于不支持动态内存管理的 enclave（即 SGX1），这是事实。 SGX2 enclave 将允许分配，并且该页必须在分配后由 enclave 接受。  [in] Size  要分配的内存区域的大小（以字节为单位）。  大小必须始终为页面大小的倍数。  如果 *BaseAddress* 未 **NULL**，则该函数会将包含一个或多个字节的所有页面分配在从 *BaseAddress* 到 *BaseAddress*+*size*。 例如，这意味着跨页边界的 2 字节范围会导致函数分配这两个页面。  [in] AllocationType  内存分配的类型。 此参数必须包含以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_COMMIT**  0x00001000 | 为指定的保留内存页分配内存费用（从内存的总体大小和磁盘上的分页文件）。 该函数还保证当调用方稍后最初访问内存时，内容将为零。 除非实际访问虚拟地址/直到实际访问虚拟地址，否则不会分配实际物理页。  若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 **VirtualAlloc2**。  除非已保留整个范围，否则尝试通过指定不带 **MEM\_RESERVE** 的 **MEM\_COMMIT** 并**NULL***BaseAddress* 来尝试提交特定地址范围。 生成的错误代码 **ERROR\_INVALID\_ADDRESS**。  尝试提交已提交的页面不会导致函数失败。 这意味着可以提交页面，而无需首先确定每个页面的当前承诺状态。  如果 baseAddress 指定 enclave 中的地址，则必须 AllocationType MEM\_COMMIT。 | | **MEM\_RESERVE**  0x00002000 | 保留进程的虚拟地址空间范围，而无需在内存或磁盘上的分页文件中分配任何实际物理存储。  通过使用 **MEM\_COMMIT**再次调用 **VirtualAlloc2** 来提交保留页。 若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 **VirtualAlloc2**。  其他内存分配函数（如 **malloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)）在释放之前无法使用保留内存。 | | **MEM\_REPLACE\_PLACEHOLDER**  0x00004000 | 将占位符替换为正常的专用分配。 仅支持数据/pf 支持的分区视图（无图像、物理内存等）。 替换占位符时，*BaseAddress* 和 *Size* 必须与占位符的占位符完全匹配，并且提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零。  将占位符替换为专用分配后，若要将该分配释放回占位符，请参阅 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 的 *dwFreeType* 参数，并 [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)。  占位符是保留内存区域的类型。 | | **MEM\_RESERVE\_PLACEHOLDER**  0x00040000 | 若要创建占位符，请调用 **VirtualAlloc2**，并将 MEM\_RESERVE | MEM\_RESERVE\_PLACEHOLDER 和 *PageProtection* 设置为 **PAGE\_NOACCESS**。 若要释放/拆分/合并占位符，请参阅 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 的 *dwFreeType* 参数，[VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)。  占位符是保留内存区域的类型。 | | **MEM\_RESET**  0x00080000 | 指示 *BaseAddress* 和 *大小* 指定的内存范围中的数据不再感兴趣。 不应从分页文件读取或写入页面。 但是，内存块稍后将再次使用，因此不应将其取消提交。 此值不能用于任何其他值。  使用此值不能保证使用 **MEM\_RESET** 操作的范围将包含零。 如果希望该区域包含零，请取消提交内存，然后重新提交内存。  使用 **MEM\_RESET**时，**VirtualAlloc2** 函数将忽略 *fProtect*的值。 但是，仍必须将 *fProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **VirtualAlloc2** 如果使用 **MEM\_RESET** 并且内存范围映射到文件，则返回错误。 仅当共享视图映射到分页文件时，才可接受。 | | **MEM\_RESET\_UNDO**  0x1000000 | **MEM\_RESET\_UNDO** 只应在之前成功应用 **MEM\_RESET** 的地址范围上调用。 它表示，*BaseAddress* 指定的指定内存范围中的数据和 *大小* 对调用方感兴趣，并尝试扭转 **MEM\_RESET**的影响。 如果函数成功，则表示指定地址范围中的所有数据都保持不变。 如果函数失败，则地址范围中的至少一些数据已替换为零。  此值不能用于任何其他值。 如果在之前未 **MEM\_RESET** 的地址范围上调用 **MEM\_RESET\_UNDO**，则行为是未定义的。 指定 MEM\_RESET时，VirtualAlloc2 函数将忽略 pageProtection的值。 但是，仍必须将 pageProtection 设置为有效的保护值，例如 PAGE\_NOACCESS。  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows 8 和 Windows Server 2012 之前不支持 **MEM\_RESET\_UNDO** 标志。 |     此参数还可以按指示指定以下值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_LARGE\_PAGES**  0x20000000 | 使用 [大型页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)分配内存。  大小和对齐方式必须是大页最小值的倍数。 若要获取此值，请使用 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数。  如果指定此值，还必须指定 **MEM\_RESERVE** 和 **MEM\_COMMIT**。 | | **MEM\_64K\_PAGES**  0x20400000 | 如果可能，向操作系统提示使用 64K 页映射内存。  64K 页是一个内存区域，大小为 64K，几乎和物理连续，几乎和物理对齐在 64K 边界上。  默认情况下，使用MEM\_64K\_PAGES分配的内存是可分页的，支持内存的物理页是按需分配的（访问时）。 如果物理内存过于碎片，无法组装物理连续的 64K 页，则MEM\_64K\_PAGES分配的全部或部分都可以改用非连续的小页进行映射。  如果MEM\_64K\_PAGES与 [MEM\_EXTENDED\_PARAMETER\_NONPAGED](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter) 属性相结合，将使用非分页的 64K 页映射分配。 在这种情况下，如果无法获取连续的 64K 页，则分配将失败。  如果指定了MEM\_64K\_PAGES，则 Size 和 BaseAddress 参数必须为 64K 的倍数（BaseAddress 可能为 NULL）。 | | **MEM\_PHYSICAL**  0x00400000 | 保留可用于映射 [地址窗口扩展插件](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）页的地址范围。  此值必须与 **MEM\_RESERVE** 一起使用，并且不能用于其他值。 | | **MEM\_TOP\_DOWN**  0x00100000 | 以最高可能地址分配内存。 这比常规分配要慢，尤其是在分配很多时。 |   [in] PageProtection  要分配的页面区域的内存保护。 如果要提交页面，则可以指定内存保护常量之一。  如果 baseAddress 指定 enclave 中的地址，则 PageProtection 不能为以下任何值：   * PAGE\_NOACCESS * PAGE\_GUARD * PAGE\_NOCACHE * PAGE\_WRITECOMBINE   为 enclave 分配动态内存时，PageProtection 参数必须 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**。  [in, out, optional] ExtendedParameters  指向 [MEM\_EXTENDED\_PARAMETER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter)类型的一个或多个扩展参数的可选指针。 每个扩展参数值本身都可以具有 *Type* 字段，**MemExtendedParameterAddressRequirements** 或 **MemExtendedParameterNumaNode**。 如果未提供 **MemExtendedParameterNumaNode** 扩展参数，则行为与 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)/[MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 函数（即，物理页的首选 NUMA 节点）是根据首次访问内存的线程的理想处理器确定的。  [in] ParameterCount  *ExtendedParameters*指向的扩展参数数。 返回值 如果函数成功，则返回值是页面分配区域的基址。  如果函数失败，则返回值 **NULL**。 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 使用此函数可以指定：   * 新分配的一系列虚拟地址空间和 2 元对齐限制 * 任意数量的扩展参数 * 作为扩展参数的物理内存的首选 NUMA 节点（请参阅 *ExtendedParameters* 参数） * 占位符操作（具体而言，替换）。   此 API 提供用于管理虚拟内存以支持高性能游戏和服务器应用程序的专用技术。 例如，占位符允许显式分区、覆盖和重新映射保留内存范围;这可用于实现任意扩展区域或虚拟内存环缓冲区。 **VirtualAlloc2** 还允许分配具有特定内存对齐方式的内存。  每个页面都有一个关联的 [页面状态](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/page-state)。 virtualAlloc2 函数 可以执行以下操作：   * 提交保留页的区域 * 保留免费页面的区域 * 同时保留并提交一个免费页面区域   **VirtualAlloc2** 可以提交已提交但无法保留已保留的页面。 这意味着你可以提交一系列页面，无论它们是否已提交，并且函数不会失败。 但是，一般情况下，应只指定大多数未提交的页面的最小范围，因为提交已提交的大量页面可能会导致 **VirtualAlloc2** 调用花费更长的时间。  可以使用 **VirtualAlloc2** 保留页块，然后对 **virtualAlloc2** 进行其他调用，以提交保留块中的单个页面。 这样一个进程就可以保留其虚拟地址空间的范围，而无需消耗物理存储，直到需要它。  如果 *lpAddress* 参数未 **NULL**，则该函数将使用 *lpAddress*，*dwSize* 参数来计算要分配的页面区域。 整个页面范围的当前状态必须与 *flAllocationType* 参数指定的分配类型兼容。 否则，该函数将失败，并且未分配任何页面。 此兼容性要求不排除提交已提交的页面;请参阅前面的列表。  若要执行动态生成的代码，请使用 **VirtualAlloc2** 分配内存，并使用 [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectex) 函数授予 **PAGE\_EXECUTE** 访问权限。  **VirtualAlloc2** 函数可用于在指定进程的虚拟地址空间中保留内存的 [地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）区域。 然后，可以使用此内存区域根据应用程序的要求将物理页映射到虚拟内存中和传出虚拟内存。 必须在 *AllocationType* 参数中设置 **MEM\_PHYSICAL** 和 **MEM\_RESERVE** 值。 不能设置 **MEM\_COMMIT** 值。 页面保护必须设置为 **PAGE\_READWRITE**。  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex) 函数可以取消提交页面、释放页面的存储，也可以同时取消提交和释放已提交的页面。 它还可以释放保留页，使其成为免费页面。  创建可执行的区域时，调用程序负责确保在代码设置到位后通过适当的调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 缓存一致性。 否则，尝试从新可执行区域执行代码可能会产生不可预知的结果。 例子 方案 1. 通过映射同一共享内存部分的两个相邻视图来创建循环缓冲区。  C++复制  #include <windows.h>  #include <stdio.h>  #include <stdlib.h>  //  // This function creates a ring buffer by allocating a pagefile-backed section  // and mapping two views of that section next to each other. This way if the  // last record in the buffer wraps it can still be accessed in a linear fashion  // using its base VA.  //  void\*  CreateRingBuffer (  unsigned int bufferSize,  \_Outptr\_ void\*\* secondaryView  )  {  BOOL result;  HANDLE section = nullptr;  SYSTEM\_INFO sysInfo;  void\* ringBuffer = nullptr;  void\* placeholder1 = nullptr;  void\* placeholder2 = nullptr;  void\* view1 = nullptr;  void\* view2 = nullptr;  GetSystemInfo (&sysInfo);  if ((bufferSize % sysInfo.dwAllocationGranularity) != 0) {  return nullptr;  }  //  // Reserve a placeholder region where the buffer will be mapped.  //  placeholder1 = (PCHAR) VirtualAlloc2 (  nullptr,  nullptr,  2 \* bufferSize,  MEM\_RESERVE | MEM\_RESERVE\_PLACEHOLDER,  PAGE\_NOACCESS,  nullptr, 0  );  if (placeholder1 == nullptr) {  printf ("VirtualAlloc2 failed, error %#x\n", GetLastError());  goto Exit;  }  //  // Split the placeholder region into two regions of equal size.  //  result = VirtualFree (  placeholder1,  bufferSize,  MEM\_RELEASE | MEM\_PRESERVE\_PLACEHOLDER  );  if (result == FALSE) {  printf ("VirtualFreeEx failed, error %#x\n", GetLastError());  goto Exit;  }  placeholder2 = (void\*) ((ULONG\_PTR) placeholder1 + bufferSize);  //  // Create a pagefile-backed section for the buffer.  //  section = CreateFileMapping (  INVALID\_HANDLE\_VALUE,  nullptr,  PAGE\_READWRITE,  0,  bufferSize, nullptr  );  if (section == nullptr) {  printf ("CreateFileMapping failed, error %#x\n", GetLastError());  goto Exit;  }  //  // Map the section into the first placeholder region.  //  view1 = MapViewOfFile3 (  section,  nullptr,  placeholder1,  0,  bufferSize,  MEM\_REPLACE\_PLACEHOLDER,  PAGE\_READWRITE,  nullptr, 0  );  if (view1 == nullptr) {  printf ("MapViewOfFile3 failed, error %#x\n", GetLastError());  goto Exit;  }  //  // Ownership transferred, don't free this now.  //  placeholder1 = nullptr;  //  // Map the section into the second placeholder region.  //  view2 = MapViewOfFile3 (  section,  nullptr,  placeholder2,  0,  bufferSize,  MEM\_REPLACE\_PLACEHOLDER,  PAGE\_READWRITE,  nullptr, 0  );  if (view2 == nullptr) {  printf ("MapViewOfFile3 failed, error %#x\n", GetLastError());  goto Exit;  }  //  // Success, return both mapped views to the caller.  //  ringBuffer = view1;  \*secondaryView = view2;  placeholder2 = nullptr;  view1 = nullptr;  view2 = nullptr;  Exit:  if (section != nullptr) {  CloseHandle (section);  }  if (placeholder1 != nullptr) {  VirtualFree (placeholder1, 0, MEM\_RELEASE);  }  if (placeholder2 != nullptr) {  VirtualFree (placeholder2, 0, MEM\_RELEASE);  }  if (view1 != nullptr) {  UnmapViewOfFileEx (view1, 0);  }  if (view2 != nullptr) {  UnmapViewOfFileEx (view2, 0);  }  return ringBuffer;  }  int \_\_cdecl wmain()  {  char\* ringBuffer;  void\* secondaryView;  unsigned int bufferSize = 0x10000;  ringBuffer = (char\*) CreateRingBuffer (bufferSize, &secondaryView);  if (ringBuffer == nullptr) {  printf ("CreateRingBuffer failed\n");  return 0;  }  //  // Make sure the buffer wraps properly.  //  ringBuffer[0] = 'a';  if (ringBuffer[bufferSize] == 'a') {  printf ("The buffer wraps as expected\n");  }  UnmapViewOfFile (ringBuffer);  UnmapViewOfFile (secondaryView);  }  方案 2. 分配内存时指定首选 NUMA 节点。  C++复制  void\*  AllocateWithPreferredNode (size\_t size, unsigned int numaNode)  {  MEM\_EXTENDED\_PARAMETER param = {0};  param.Type = MemExtendedParameterNumaNode;  param.ULong = numaNode;  return VirtualAlloc2 (  nullptr, nullptr,  size,  MEM\_RESERVE | MEM\_COMMIT,  PAGE\_READWRITE,  &param, 1);  }  方案 3. 在特定虚拟地址范围（此示例中低于 4GB）和特定对齐方式分配内存。  C++复制  void\*  AllocateAlignedBelow2GB (size\_t size, size\_t alignment)  {  MEM\_ADDRESS\_REQUIREMENTS addressReqs = {0};  MEM\_EXTENDED\_PARAMETER param = {0};  addressReqs.Alignment = alignment;  addressReqs.HighestEndingAddress = (PVOID)(ULONG\_PTR) 0x7fffffff;  param.Type = MemExtendedParameterAddressRequirements;  param.Pointer = &addressReqs;  return VirtualAlloc2 (  nullptr, nullptr,  size,  MEM\_RESERVE | MEM\_COMMIT,  PAGE\_READWRITE,  &param, 1);  } 要求  |  |  | | --- | --- | | **标头** | memoryapi.h （包括 Windows.h） | | **库** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [ReadProcessMemory](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-readprocessmemory)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma)  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock)  [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotect)  [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery)  [WriteProcessMemory](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-writeprocessmemory) |

## VirtualAlloc2FromApp 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留、提交或更改调用进程的虚拟地址空间中页面区域的状态。 此函数分配的内存会自动初始化为零。  使用此函数，可以：对于新分配，请指定一系列虚拟地址空间和 2 元对齐限制;指定任意数量的扩展参数;将物理内存的首选 NUMA 节点指定为扩展参数;并指定占位符操作（具体而言，替换）。  若要指定 NUMA 节点，请参阅 *ExtendedParameters* 参数。 语法 C++复制  PVOID VirtualAlloc2FromApp(  [in, optional] HANDLE Process,  [in, optional] PVOID BaseAddress,  [in] SIZE\_T Size,  [in] ULONG AllocationType,  [in] ULONG PageProtection,  [in, out, optional] MEM\_EXTENDED\_PARAMETER \*ExtendedParameters,  [in] ULONG ParameterCount  ); 参数 [in, optional] Process  进程的句柄。 该函数在进程的虚拟地址空间中分配内存。  句柄必须具有 **PROCESS\_VM\_OPERATION** 访问权限。 有关详细信息，请参阅 [进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  [in, optional] BaseAddress  为要分配的页面区域指定所需起始地址的指针。  如果 *BaseAddress***NULL**，则该函数将确定分配区域的位置。  如果 *BaseAddress* 未 **NULL**，则提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零，并且基址必须是系统分配粒度的倍数。 若要确定分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  [in] Size  要分配的内存区域的大小（以字节为单位）。  大小必须始终为页面大小的倍数。  如果 *BaseAddress* 未 **NULL**，则该函数会将包含一个或多个字节的所有页面分配在从 *BaseAddress* 到 *BaseAddress*+*size*。 例如，这意味着跨页边界的 2 字节范围会导致函数分配这两个页面。  [in] AllocationType  内存分配的类型。 此参数必须包含以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_COMMIT**  0x00001000 | 为指定的保留内存页分配内存费用（从内存的总体大小和磁盘上的分页文件）。 该函数还保证当调用方稍后最初访问内存时，内容将为零。 除非实际访问虚拟地址/直到实际访问虚拟地址，否则不会分配实际物理页。  若要在一个步骤中保留和提交页面，请使用 **MEM\_COMMIT | MEM\_RESERVE调用 virtual2AllocFromApp**。  除非已保留整个范围，否则尝试通过指定不带 **MEM\_RESERVE** 的 **MEM\_COMMIT** 并**NULL***BaseAddress* 来尝试提交特定地址范围。 生成的错误代码 **ERROR\_INVALID\_ADDRESS**。  尝试提交已提交的页面不会导致函数失败。 这意味着可以提交页面，而无需首先确定每个页面的当前承诺状态。 | | **MEM\_RESERVE**  0x00002000 | 保留进程的虚拟地址空间范围，而无需在内存或磁盘上的分页文件中分配任何实际物理存储。  可以在后续调用 **Virtual2AllocFromApp** 函数时提交保留页。 若要在一个步骤中保留和提交页面，请使用 **MEM\_COMMIT** | **MEM\_RESERVE**调用 **Virtual2AllocFromApp**。  其他内存分配函数（如 **malloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)）在释放之前无法使用保留的内存范围。 | | **MEM\_REPLACE\_PLACEHOLDER**  0x00004000 | 将占位符替换为正常的专用分配。 仅支持数据/pf 支持的分区视图（无图像、物理内存等）。 替换占位符时，*BaseAddress* 和 *Size* 必须与占位符的占位符完全匹配，并且提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零。  将占位符替换为专用分配后，若要将该分配释放回占位符，请参阅 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 的 *dwFreeType* 参数，并 [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)。  占位符是保留内存区域的类型。 | | **MEM\_RESERVE\_PLACEHOLDER**  0x00040000 | 若要创建占位符，请调用 [VirtualAlloc2](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc2)，并将 MEM\_RESERVE | MEM\_RESERVE\_PLACEHOLDER 和 *PageProtection* 设置为 **PAGE\_NOACCESS**。 若要释放/拆分/合并占位符，请参阅 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 的 *dwFreeType* 参数，[VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)。  占位符是保留内存区域的类型。 | | **MEM\_RESET**  0x00080000 | 指示 *BaseAddress* 和 *大小* 指定的内存范围中的数据不再感兴趣。 不应从分页文件读取或写入页面。 但是，内存块稍后将再次使用，因此不应将其取消提交。 此值不能用于任何其他值。  使用此值不能保证使用 **MEM\_RESET** 操作的范围将包含零。 如果希望该区域包含零，请取消提交内存，然后重新提交内存。  指定 **MEM\_RESET**时，**Virtual2AllocFromApp** 函数将忽略 *保护*的值。 但是，仍必须将保护 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **Virtual2AllocFromApp** 如果使用 **MEM\_RESET** 并且内存范围映射到文件，则返回错误。 仅当共享视图映射到分页文件时，才可接受。 | | **MEM\_RESET\_UNDO**  0x1000000 | **MEM\_RESET\_UNDO** 只应在之前成功应用 **MEM\_RESET** 的地址范围上调用。 它表示，*BaseAddress* 指定的指定内存范围中的数据和 *大小* 对调用方感兴趣，并尝试扭转 **MEM\_RESET**的影响。 如果函数成功，则表示指定地址范围中的所有数据都保持不变。 如果函数失败，则地址范围中的至少一些数据已替换为零。  此值不能用于任何其他值。 如果在之前未 **MEM\_RESET** 的地址范围上调用 **MEM\_RESET\_UNDO**，则行为是未定义的。 指定 **MEM\_RESET**时，**Virtual2AllocFromApp** 函数将忽略 *保护*的值。 但是，仍必须将保护 设置为有效的保护值，例如 **PAGE\_NOACCESS**。 |     此参数还可以按指示指定以下值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_LARGE\_PAGES**  0x20000000 | 使用 [大型页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)分配内存。  大小和对齐方式必须是大页最小值的倍数。 若要获取此值，请使用 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数。  如果指定此值，还必须指定 **MEM\_RESERVE** 和 **MEM\_COMMIT**。 | | **MEM\_PHYSICAL**  0x00400000 | 保留可用于映射 [地址窗口扩展插件](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）页的地址范围。  此值必须与 **MEM\_RESERVE** 一起使用，并且不能用于其他值。 | | **MEM\_TOP\_DOWN**  0x00100000 | 以最高可能地址分配内存。 这比常规分配要慢，尤其是在分配很多时。 | | **MEM\_WRITE\_WATCH**  0x00200000 | 使系统跟踪写入到分配区域中的页面。 如果指定此值，还必须指定 **MEM\_RESERVE**。  若要检索自分配区域或重置写入跟踪状态以来已写入的页面的地址，请调用 [GetWriteWatch](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getwritewatch) 函数。 若要重置写入跟踪状态，请调用 **GetWriteWatch** 或 [ResetWriteWatch](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-resetwritewatch)。 写入跟踪功能将一直为内存区域启用，直到释放该区域。 |   [in] PageProtection  要分配的页面区域的内存保护。 如果要提交页面，则可以指定内存保护常量之一。 以下常量生成错误：   * **PAGE\_EXECUTE** * **PAGE\_EXECUTE\_READ** * **PAGE\_EXECUTE\_READWRITE** * **PAGE\_EXECUTE\_WRITECOPY**   [in, out, optional] ExtendedParameters  指向 [MEM\_EXTENDED\_PARAMETER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter)类型的一个或多个扩展参数的可选指针。 每个扩展参数值本身都可以具有 *Type* 字段，**MemExtendedParameterAddressRequirements** 或 **MemExtendedParameterNumaNode**。 如果未提供 **MemExtendedParameterNumaNode** 扩展参数，则行为与 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)/[MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 函数（即，物理页的首选 NUMA 节点）是根据首次访问内存的线程的理想处理器确定的。  [in] ParameterCount  *ExtendedParameters*指向的扩展参数数。 返回值 如果函数成功，则返回值是页面分配区域的基址。  如果函数失败，则返回值 **NULL**。 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 此 API 有助于支持高性能游戏和服务器应用程序，这些应用程序在管理虚拟地址空间方面有特定要求。 例如，将内存映射在以前保留的区域之上;这可用于实现自动包装环缓冲区。 分配具有特定对齐方式的内存;例如，若要使应用程序能够按需提交大型/巨大的页面映射区域。  可以从具有实时 （JIT） 功能的 Windows 应用商店应用中调用 **Virtual2AllocFromApp**，以使用 JIT 功能。 应用必须在应用清单文件中包括 **codeGeneration** 功能才能使用 JIT 功能。  每个页面都有一个关联的 [页面状态](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/page-state)。 **Virtual2AllocFromApp** 函数可以执行以下操作：   * 提交保留页的区域 * 保留免费页面的区域 * 同时保留并提交一个免费页面区域   **Virtual2AllocFromApp** 无法保留保留页。 它可以提交已提交的页面。 这意味着你可以提交一系列页面，无论它们是否已提交，并且函数不会失败。  可以使用 **Virtual2AllocFromApp** 来保留页面块，然后对 **virtual2AllocFromApp** 进行其他调用，以提交保留块中的单个页面。 这样一个进程就可以保留其虚拟地址空间的范围，而无需消耗物理存储，直到需要它。  如果 *BaseAddress* 参数未 **NULL**，则该函数使用 *BaseAddress*，*大小* 参数来计算要分配的页面区域。 整个页面范围的当前状态必须与 *AllocationType* 参数指定的分配类型兼容。 否则，该函数将失败，并且未分配任何页面。 如前所述，此兼容性要求不排除提交已提交的页面。  **Virtual2AllocFromApp** 不允许创建可执行页面。  **Virtual2AllocFromApp** 函数可用于在指定进程的虚拟地址空间中保留内存的 [地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）区域。 然后，可以使用此内存区域根据应用程序的要求将物理页映射到虚拟内存中和传出虚拟内存。 必须在 *AllocationType* 参数中设置 **MEM\_PHYSICAL** 和 **MEM\_RESERVE** 值。 不能设置 **MEM\_COMMIT** 值。 页面保护必须设置为 **PAGE\_READWRITE**。  [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 函数可以取消提交页面、释放页面的存储，也可以同时取消提交和释放已提交的页面。 它还可以释放保留页，使其成为免费页面。  创建可执行的区域时，调用程序负责确保在代码设置到位后通过适当的调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 缓存一致性。 否则，尝试从新可执行区域执行代码可能会产生不可预知的结果。 例子 有关代码示例，请参阅 virtual2Alloc 。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h （包括 Windows.h） | | **库** | WindowsApp.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex)  [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock)  [VirtualProtectFromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectfromapp)  [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery) |

## VirtualAllocEx 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留、提交或更改指定进程的虚拟地址空间中内存区域的状态。 该函数初始化它分配给零的内存。  若要为物理内存指定 NUMA 节点，请参阅 virtualAllocExNuma。 语法 C++复制  LPVOID VirtualAllocEx(  [in] HANDLE hProcess,  [in, optional] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD flAllocationType,  [in] DWORD flProtect  ); 参数 [in] hProcess  进程的句柄。 该函数在进程的虚拟地址空间中分配内存。  句柄必须具有 **PROCESS\_VM\_OPERATION** 访问权限。 有关详细信息，请参阅 [进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  [in, optional] lpAddress  为要分配的页面区域指定所需起始地址的指针。  如果要保留内存，该函数会将此地址舍入到分配粒度中最近的倍数。  如果要提交已保留的内存，函数会将此地址向下舍入到最近的页面边界。 若要确定页面的大小和主机计算机上的分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  如果 *lpAddress***NULL**，则该函数将确定分配区域的位置。  如果此地址位于未通过调用 [InitializeEnclave](https://learn.microsoft.com/zh-cn/windows/desktop/api/enclaveapi/nf-enclaveapi-initializeenclave)初始化的 enclave 中，**VirtualAllocEx** 为该地址的 enclave 分配零页。 该页必须以前未提交，并且不会使用 Intel Software Guard Extensions 编程模型的 EEXTEND 指令进行测量。  如果初始化的 enclave 中的地址，则分配操作将失败，并出现 **ERROR\_INVALID\_ADDRESS** 错误。 对于不支持动态内存管理的 enclave（即 SGX1），这是事实。 SGX2 enclave 将允许分配，并且该页必须在分配后由 enclave 接受。  [in] dwSize  要分配的内存区域的大小（以字节为单位）。  如果 *lpAddress***NULL**，则该函数将 *dwSize* 向上舍入到下一页边界。  如果 *lpAddress* 未 **NULL**，则该函数会将包含一个或多个字节的所有页面分配给从 *lpAddress* 到 *lpAddress*+*dwSize*。 例如，这意味着跨页边界的 2 字节范围会导致函数分配这两个页面。  [in] flAllocationType  内存分配的类型。 此参数必须包含以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_COMMIT**  0x00001000 | 为指定的保留内存页分配内存费用（从内存的总体大小和磁盘上的分页文件）。 该函数还保证当调用方稍后最初访问内存时，内容将为零。 除非实际访问虚拟地址/直到实际访问虚拟地址，否则不会分配实际物理页。  若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 **virtualAllocEx**。  除非已保留整个范围，否则尝试通过指定不带 **MEM\_RESERVE** 的 **MEM\_COMMIT** 并**NULL***lpAddress* 来提交特定地址范围。 生成的错误代码 **ERROR\_INVALID\_ADDRESS**。  尝试提交已提交的页面不会导致函数失败。 这意味着可以提交页面，而无需首先确定每个页面的当前承诺状态。  如果 *lpAddress* 指定 enclave 中的地址，则必须 **MEM\_COMMIT***flAllocationType*。 | | **MEM\_RESERVE**  0x00002000 | 保留进程的虚拟地址空间范围，而无需在内存或磁盘上的分页文件中分配任何实际物理存储。  使用 **MEM\_COMMIT再次调用 virtualAllocEx**，提交保留页。 若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 **virtualAllocEx**。  其他内存分配函数（如 **malloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)）在释放之前无法使用保留内存。 | | **MEM\_RESET**  0x00080000 | 指示 *lpAddress* 指定的内存区域中的数据不再感兴趣，*dwSize* 不再感兴趣。 不应从分页文件读取或写入页面。 但是，内存块稍后将再次使用，因此不应将其取消提交。 此值不能用于任何其他值。  使用此值不能保证使用 **MEM\_RESET** 操作的范围将包含零。 如果希望该区域包含零，请取消提交内存，然后重新提交内存。  使用 **MEM\_RESET**时，**VirtualAllocEx** 函数将忽略 *fProtect*的值。 但是，仍必须将 *fProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **VirtualAllocEx** 如果使用 **MEM\_RESET** 并且内存范围映射到文件，则返回错误。 仅当共享视图映射到分页文件时，才可接受。 | | **MEM\_RESET\_UNDO**  0x1000000 | **MEM\_RESET\_UNDO** 只应在之前成功应用 **MEM\_RESET** 的地址范围上调用。 它指示由 *lpAddress 指定的指定内存范围中的数据* 和 *dwSize* 对调用方感兴趣，并尝试扭转 **MEM\_RESET**的影响。 如果函数成功，则表示指定地址范围中的所有数据都保持不变。 如果函数失败，则地址范围中的至少一些数据已替换为零。  此值不能用于任何其他值。 如果在之前未 **MEM\_RESET** 的地址范围上调用 **MEM\_RESET\_UNDO**，则行为是未定义的。 指定 **MEM\_RESET**时，**VirtualAllocEx** 函数将忽略 *flProtect*的值。 但是，仍必须将 *flProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows 8 和 Windows Server 2012 之前不支持 **MEM\_RESET\_UNDO** 标志。 |     此参数还可以按指示指定以下值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_LARGE\_PAGES**  0x20000000 | 使用 [大型页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)分配内存。  大小和对齐方式必须是大页最小值的倍数。 若要获取此值，请使用 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数。  如果指定此值，还必须指定 **MEM\_RESERVE** 和 **MEM\_COMMIT**。 | | **MEM\_PHYSICAL**  0x00400000 | 保留可用于映射 [地址窗口扩展插件](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）页的地址范围。  此值必须与 **MEM\_RESERVE** 一起使用，并且不能用于其他值。 | | **MEM\_TOP\_DOWN**  0x00100000 | 以最高可能地址分配内存。 这比常规分配要慢，尤其是在分配很多时。 |   [in] flProtect  要分配的页面区域的内存保护。 如果要提交页面，则可以指定内存保护常量之一。  如果 lpAddress 指定 enclave 中的地址，flProtect 不能是以下值之一：   * PAGE\_NOACCESS * PAGE\_GUARD * PAGE\_NOCACHE * PAGE\_WRITECOMBINE   为 enclave 分配动态内存时，flProtect 参数必须 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**。 返回值 如果函数成功，则返回值是页面分配区域的基址。  如果函数失败，则返回值 **NULL**。 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 每个页面都有一个关联的 [页面状态](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/page-state)。 virtualAllocEx 函数 可以执行以下操作：   * 提交保留页的区域 * 保留免费页面的区域 * 同时保留并提交一个免费页面区域   **VirtualAllocEx** 无法保留保留页。 它可以提交已提交的页面。 这意味着你可以提交一系列页面，无论它们是否已提交，并且函数不会失败。  可以使用 **VirtualAllocEx** 来保留页面块，然后对 **virtualAllocEx** 进行其他调用，以提交保留块中的单个页面。 这样一个进程就可以保留其虚拟地址空间的范围，而无需消耗物理存储，直到需要它。  如果 *lpAddress* 参数未 **NULL**，则该函数将使用 *lpAddress*，*dwSize* 参数来计算要分配的页面区域。 整个页面范围的当前状态必须与 *flAllocationType* 参数指定的分配类型兼容。 否则，该函数将失败，并且未分配任何页面。 此兼容性要求不排除提交已提交的页面;请参阅前面的列表。  若要执行动态生成的代码，请使用 **VirtualAllocEx** 来分配内存和 [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectex) 函数来授予 **PAGE\_EXECUTE** 访问权限。  **VirtualAllocEx** 函数可用于在指定进程的虚拟地址空间中保留内存 [地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）区域。 然后，可以使用此内存区域根据应用程序的要求将物理页映射到虚拟内存中和传出虚拟内存。 必须在 *AllocationType* 参数中设置 **MEM\_PHYSICAL** 和 **MEM\_RESERVE** 值。 不能设置 **MEM\_COMMIT** 值。 页面保护必须设置为 **PAGE\_READWRITE**。  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex) 函数可以取消提交页面、释放页面的存储，也可以同时取消提交和释放已提交的页面。 它还可以释放保留页，使其成为免费页面。  创建可执行的区域时，调用程序负责确保在代码设置到位后通过适当的调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 缓存一致性。 否则，尝试从新可执行区域执行代码可能会产生不可预知的结果。 要求 展开表   | **要求** | **价值** | | --- | --- | | **最低支持的客户端** | Windows XP [桌面应用 |UWP 应用] | | **支持的最低服务器** | Windows Server 2003 [桌面应用 |UWP 应用] | | **目标平台** | 窗户 | | **标头** | memoryapi.h （包括 Windows.h、Memoryapi.h） | | **库** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [ReadProcessMemory](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-readprocessmemory)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma)  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock)  [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotect)  [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery)  [WriteProcessMemory](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-writeprocessmemory) |

## VirtualAllocExNuma 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留、提交或更改指定进程的虚拟地址空间中的内存区域的状态，并为物理内存指定 NUMA 节点。 语法 C++复制  LPVOID VirtualAllocExNuma(  [in] HANDLE hProcess,  [in, optional] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD flAllocationType,  [in] DWORD flProtect,  [in] DWORD nndPreferred  ); parameters [in] hProcess  进程的句柄。 函数在此进程的虚拟地址空间内分配内存。  句柄必须具有 **PROCESS\_VM\_OPERATION** 访问权限。 有关详细信息，请参阅 [进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  [in, optional] lpAddress  为要分配的页面区域指定所需起始地址的指针。  如果要保留内存，函数会将此地址向下舍入到分配粒度的最近倍数。  如果提交已保留的内存，函数会将此地址向下舍入到最近的页边界。 若要确定页面的大小和主计算机上的分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  如果 *lpAddress* 为 **NULL**，则该函数确定分配区域的位置。  [in] dwSize  要分配的内存区域的大小（以字节为单位）。  如果 *lpAddress* 为 **NULL**，则该函数会将 *dwSize* 向上舍入到下一页边界。  如果 *lpAddress* 不为 **NULL**，则该函数会将包含一个或多个字节的所有页面分配在 *从 lpAddress* 到 (lpAddress+dwSize)的范围内。 例如，这意味着跨页边界的 2 字节范围会导致函数分配这两个页面。  [in] flAllocationType  内存分配的类型。 此参数必须包含以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_COMMIT**  0x00001000 | 从指定保留内存页的磁盘) 的总内存大小和分页文件 (分配内存费用。 函数还保证当调用方稍后最初访问内存时，内容将为零。 除非实际访问虚拟地址，否则不会分配实际物理页面。  若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 函数。  尝试通过指定 **MEM\_COMMIT** 而不指定 **MEM\_RESERVE** 和非 **NULL***lpAddress* 来提交特定地址范围，除非已保留整个范围。 生成的**错误代码ERROR\_INVALID\_ADDRESS。**  尝试提交已提交的页面不会导致函数失败。 这意味着，无需先确定每个页面的当前承诺状态即可提交页面。 | | **MEM\_RESERVE**  0x00002000 | 保留进程的虚拟地址空间范围，而无需在内存或磁盘上的分页文件中分配任何实际物理存储。  通过使用 **MEM\_COMMIT** 再次调用 函数来提交保留页。 若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 函数。  其他内存分配函数（如 **malloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)）在释放内存之前不能使用预留内存。 | | **MEM\_RESET**  0x00080000 | 指示 *lpAddress* 和 *dwSize* 指定的内存范围中的数据不再感兴趣。 不应从分页文件读取或写入页面。 但是，内存块稍后将再次使用，因此不应取消提交。 此值不能与任何其他值一起使用。  使用此值并不能保证使用 **MEM\_RESET** 操作的范围将包含零。 如果希望范围包含零，请取消提交内存，然后重新提交。  使用 **MEM\_RESET** 时，函数会忽略 *fProtect* 的值。 但是，仍必须将 *fProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  如果使用 **MEM\_RESET** 并且内存范围映射到文件，则函数将返回错误。 仅当共享视图映射到分页文件时，才可接受该视图。 | | **MEM\_RESET\_UNDO**  0x1000000 | **应** 仅对之前成功应用MEM\_RESET的地址范围调用 **MEM\_RESET\_UNDO** 。 它指示调用方对 *lpAddress* 和 *dwSize* 指定的指定内存范围中的数据感兴趣，并尝试反转 **MEM\_RESET**的影响。 如果该函数成功，则表示指定地址范围中的所有数据都保持不变。 如果函数失败，则至少将地址范围中的某些数据替换为零。  此值不能与任何其他值一起使用。 如果在之前未**MEM\_RESET**的地址范围上调用**MEM\_RESET\_UNDO**，则行为未定义。 指定 **MEM\_RESET**时， **VirtualAllocExNuma** 函数将忽略 *flProtect* 的值。 但是，仍必须将 *flProtect* 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **Windows Server 2008 R2、Windows 7、Windows Server 2008 和 Windows Vista：**在 Windows 8 和 Windows Server 2012 之前，不支持 **MEM\_RESET\_UNDO** 标志。 |     此参数还可以按指示指定以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_LARGE\_PAGES**  0x20000000 | 使用 [大页支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)分配内存。  大小和对齐方式必须是大页最小值的倍数。 若要获取此值，请使用 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数。  如果指定此值，还必须指定 **MEM\_RESERVE** 和 **MEM\_COMMIT**。 | | **MEM\_PHYSICAL**  0x00400000 | 保留可用于将 [地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions) (AWE) 页映射的地址范围。  此值必须与 **MEM\_RESERVE** 一起使用，不能与其他值一起使用。 | | **MEM\_TOP\_DOWN**  0x00100000 | 在可能的最高地址分配内存。 |   [in] flProtect  要分配的页区域的内存保护。 如果正在提交页面，则可以指定任何一个 [内存保护常量](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-protection-constants)。  保护页面时指定的保护属性不能与分配页面时指定的属性冲突。  [in] nndPreferred  物理内存应驻留的 NUMA 节点。  仅在分配新的 VA 区域 (已提交或保留) 时才使用。 否则，当 API 用于提交已存在的区域中的页面时，将忽略此参数 返回值 如果函数成功，则返回值是已分配页区域的基址。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 每个页面都有一个关联的 [页状态](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/page-state)。 **VirtualAllocExNuma** 函数可以执行以下操作：   * 提交保留页的区域 * 保留可用页面区域 * 同时保留和提交可用页面区域   **VirtualAllocExNuma** 无法保留保留页。 它可以提交已提交的页面。 这意味着，无论页面是否已提交，都可以提交一系列页面，并且函数不会失败。  可以使用 **VirtualAllocExNuma** 保留页面块，然后对 **VirtualAllocExNuma** 进行其他调用，以提交保留块中的单个页面。 这使进程能够保留其虚拟地址空间的范围，而无需使用物理存储，直到需要为止。  如果 *lpAddress* 参数不为 **NULL**，则该函数使用 *lpAddress* 和 *dwSize* 参数来计算要分配的页面区域。 整个页面范围的当前状态必须与 *flAllocationType* 参数指定的分配类型兼容。 否则，函数将失败，并且不会分配任何页面。 此兼容性要求不排除提交已提交的页面;请参阅前面的列表。  由于 **VirtualAllocExNuma** 不分配任何物理页面，因此无论页面在该节点上或系统中的其他位置上是否可用，它都会成功。 物理页面是按需分配的。 如果首选节点的页数不足，内存管理器将使用其他节点中的页。 如果内存已分页，则返回内存时会使用相同的进程。  若要执行动态生成的代码，请使用 **VirtualAllocExNuma** 分配内存，并使用 [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectex) 函数授予 **PAGE\_EXECUTE** 访问权限。  **VirtualAllocExNuma** 函数可用于在指定进程的虚拟地址空间中保留[地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions) (AWE) 内存区域。 然后，可以使用此内存区域将物理页映射到虚拟内存中，以及根据应用程序的要求映射出虚拟内存。 必须在 *AllocationType* 参数中设置**MEM\_PHYSICAL**和**MEM\_RESERVE**值。 不得设置 **MEM\_COMMIT** 值。 页面保护必须设置为 **PAGE\_READWRITE**。  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex) 函数可以取消提交已提交的页面、释放页面的存储，也可以同时取消提交和释放已提交的页面。 它还可以释放保留页，使其成为免费页面。  若要编译使用此函数的应用程序， **请将\_WIN32\_WINNT** 定义为 0x0600 或更高版本。 示例 有关示例，请参阅 [从 NUMA 节点分配内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/allocating-memory-from-a-numa-node)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [NUMA 支持](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/numa-support)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex)  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfreeex)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock)  [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotect)  [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery) |

## VirtualAllocFromApp 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留、提交或更改调用进程的虚拟地址空间中页面区域的状态。 此函数分配的内存会自动初始化为零。 语法 C++复制  PVOID VirtualAllocFromApp(  [in, optional] PVOID BaseAddress,  [in] SIZE\_T Size,  [in] ULONG AllocationType,  [in] ULONG Protection  ); 参数 [in, optional] BaseAddress  要分配的区域的起始地址。 如果保留内存，则指定的地址向下舍入到分配粒度中最近的倍数。 如果内存已保留并正在提交，地址将向下舍入到下一页边界。 若要确定页面的大小和主机计算机上的分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。 如果此参数 **NULL**，系统将确定分配区域的位置。  [in] Size  区域的大小（以字节为单位）。 如果 *BaseAddress* 参数 **NULL**，则此值向上舍入到下一页边界。 否则，分配的页面将包含从 *BaseAddress* 到 *BaseAddress*+*Size*范围内包含一个或多个字节的所有页。 这意味着跨页边界的 2 字节范围会导致这两个页面都包括在分配的区域。  [in] AllocationType  内存分配的类型。 此参数必须包含以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_COMMIT**  0x00001000 | 为指定的保留内存页分配内存费用（从内存的总体大小和磁盘上的分页文件）。 该函数还保证当调用方稍后最初访问内存时，内容将为零。 除非实际访问虚拟地址/直到实际访问虚拟地址，否则不会分配实际物理页。  若要在一个步骤中保留和提交页面，请使用 MEM\_COMMIT | MEM\_RESERVE调用 **virtualAllocFromApp**。  除非已保留整个范围，否则尝试通过指定不带 **MEM\_RESERVE** 的 **MEM\_COMMIT** 并**NULL***BaseAddress* 来尝试提交特定地址范围。 生成的错误代码 **ERROR\_INVALID\_ADDRESS**。  尝试提交已提交的页面不会导致函数失败。 这意味着可以提交页面，而无需首先确定每个页面的当前承诺状态。 | | **MEM\_RESERVE**  0x00002000 | 保留进程的虚拟地址空间范围，而无需在内存或磁盘上的分页文件中分配任何实际物理存储。  可以在后续调用 **VirtualAllocFromApp** 函数中提交保留页。 若要在一个步骤中保留和提交页面，请使用 **MEM\_COMMIT** | **MEM\_RESERVE**调用 **VirtualAllocFromApp**。  其他内存分配函数（如 **malloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)）在释放之前无法使用保留的内存范围。 | | **MEM\_RESET**  0x00080000 | 指示 *BaseAddress* 和 *大小* 指定的内存范围中的数据不再感兴趣。 不应从分页文件读取或写入页面。 但是，内存块稍后将再次使用，因此不应将其取消提交。 此值不能用于任何其他值。  使用此值不能保证使用 **MEM\_RESET** 操作的范围将包含零。 如果希望该区域包含零，请取消提交内存，然后重新提交内存。  指定 **MEM\_RESET**时，**VirtualAllocFromApp** 函数将忽略 *Protection*的值。 但是，仍必须将保护 设置为有效的保护值，例如 **PAGE\_NOACCESS**。  **VirtualAllocFromApp** 如果使用 **MEM\_RESET** 并且内存范围映射到文件，则返回错误。 仅当共享视图映射到分页文件时，才可接受。 | | **MEM\_RESET\_UNDO**  0x1000000 | **MEM\_RESET\_UNDO** 只应在之前成功应用 **MEM\_RESET** 的地址范围上调用。 它表示，*BaseAddress* 指定的指定内存范围中的数据和 *大小* 对调用方感兴趣，并尝试扭转 **MEM\_RESET**的影响。 如果函数成功，则表示指定地址范围中的所有数据都保持不变。 如果函数失败，则地址范围中的至少一些数据已替换为零。  此值不能用于任何其他值。 如果在之前未 **MEM\_RESET** 的地址范围上调用 **MEM\_RESET\_UNDO**，则行为是未定义的。 指定 **MEM\_RESET**时，**VirtualAllocFromApp** 函数将忽略 *Protection*的值。 但是，仍必须将保护 设置为有效的保护值，例如 **PAGE\_NOACCESS**。 |     此参数还可以按指示指定以下值。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_LARGE\_PAGES**  0x20000000 | 使用 [大型页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)分配内存。  大小和对齐方式必须是大页最小值的倍数。 若要获取此值，请使用 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数。  如果指定此值，还必须指定 **MEM\_RESERVE** 和 **MEM\_COMMIT**。 | | **MEM\_PHYSICAL**  0x00400000 | 保留可用于映射 [地址窗口扩展插件](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）页的地址范围。  此值必须与 **MEM\_RESERVE** 一起使用，并且不能用于其他值。 | | **MEM\_TOP\_DOWN**  0x00100000 | 以最高可能地址分配内存。 这比常规分配要慢，尤其是在分配很多时。 | | **MEM\_WRITE\_WATCH**  0x00200000 | 使系统跟踪写入到分配区域中的页面。 如果指定此值，还必须指定 **MEM\_RESERVE**。  若要检索自分配区域或重置写入跟踪状态以来已写入的页面的地址，请调用 [GetWriteWatch](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getwritewatch) 函数。 若要重置写入跟踪状态，请调用 **GetWriteWatch** 或 [ResetWriteWatch](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-resetwritewatch)。 写入跟踪功能将一直为内存区域启用，直到释放该区域。 |   [in] Protection  要分配的页面区域的内存保护。 如果要提交页面，则可以指定内存保护常量之一。 以下常量生成错误：   * **PAGE\_EXECUTE** * **PAGE\_EXECUTE\_READ** * **PAGE\_EXECUTE\_READWRITE** * **PAGE\_EXECUTE\_WRITECOPY**  返回值 如果函数成功，则返回值是页面分配区域的基址。  如果函数失败，则返回值 **NULL**。 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 可以从具有实时 （JIT） 功能的 Windows 应用商店应用中调用 **VirtualAllocFromApp**，以使用 JIT 功能。 应用必须在应用清单文件中包括 **codeGeneration** 功能才能使用 JIT 功能。  每个页面都有一个关联的 [页面状态](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/page-state)。 **VirtualAllocFromApp** 函数可以执行以下操作：   * 提交保留页的区域 * 保留免费页面的区域 * 同时保留并提交一个免费页面区域   **VirtualAllocFromApp** 无法保留保留页。 它可以提交已提交的页面。 这意味着你可以提交一系列页面，无论它们是否已提交，并且函数不会失败。  可以使用 **VirtualAllocFromApp** 保留页面块，然后对 **virtualAllocFromApp** 进行其他调用，以提交保留块中的单个页面。 这样一个进程就可以保留其虚拟地址空间的范围，而无需消耗物理存储，直到需要它。  如果 *BaseAddress* 参数未 **NULL**，则该函数使用 *BaseAddress*，*大小* 参数来计算要分配的页面区域。 整个页面范围的当前状态必须与 *AllocationType* 参数指定的分配类型兼容。 否则，该函数将失败，并且未分配任何页面。 如前所述，此兼容性要求不排除提交已提交的页面。  **VirtualAllocFromApp** 不允许创建可执行页面。  **VirtualAllocFromApp** 函数可用于在指定进程的虚拟地址空间中保留内存的 [地址窗口扩展](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/address-windowing-extensions)（AWE）区域。 然后，可以使用此内存区域根据应用程序的要求将物理页映射到虚拟内存中和传出虚拟内存。 必须在 *AllocationType* 参数中设置 **MEM\_PHYSICAL** 和 **MEM\_RESERVE** 值。 不能设置 **MEM\_COMMIT** 值。 页面保护必须设置为 **PAGE\_READWRITE**。  [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 函数可以取消提交页面、释放页面的存储，也可以同时取消提交和释放已提交的页面。 它还可以释放保留页，使其成为免费页面。  创建可执行的区域时，调用程序负责确保在代码设置到位后通过适当的调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 缓存一致性。 否则，尝试从新可执行区域执行代码可能会产生不可预知的结果。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h （包括 Windows.h） | | **库** | WindowsApp.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理功能](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex)  [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock)  [VirtualProtectFromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectfromapp)  [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery) |

## VirtualFree 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 释放、取消提交或释放和取消提交调用进程的虚拟地址空间中的页面区域。  若要释放 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex) 函数在另一个进程中分配的内存，请使用 [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfreeex) 函数。 语法 C++复制  BOOL VirtualFree(  [in] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD dwFreeType  ); 参数 [in] lpAddress  指向要释放的页区域的基址的指针。  如果 dwFreeType 参数 **MEM\_RELEASE**，则此参数必须是保留页面区域时 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 函数返回的基址。  [in] dwSize  要释放的内存区域的大小（以字节为单位）。  如果 dwFreeType 参数 **MEM\_RELEASE**，则此参数必须为 0 (零) 。 函数释放在 [对 VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 的初始分配调用中保留的整个区域。  如果 dwFreeType 参数 **MEM\_DECOMMIT**，则函数将取消提交包含 lpAddress 参数 (lpAddress+dwSize)到 范围内的一个或多个字节的所有内存页。 例如，这意味着跨越页边界的 2 字节内存区域会导致两个页面都解除提交。 如果 lpAddress 是 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 返回的基址，而 dwSize 为 0 (零) ，则函数将取消提交 **VirtualAlloc** 分配的整个区域。 之后，整个区域将处于保留状态。  [in] dwFreeType  自由操作的类型。 此参数须为下列值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_DECOMMIT**  0x00004000 | 取消提交已提交页面的指定区域。 操作后，页面将处于保留状态。  如果尝试取消提交未提交的页面，函数不会失败。 这意味着，无需先确定当前承诺状态，即可取消提交一系列页面。  当 lpAddress 参数提供 enclave 的基址时，不支持**MEM\_DECOMMIT**值。 对于不支持动态内存管理 (（即 SGX1) ）的 enclave 也是如此。 SGX2 enclave 允许 **MEM\_DECOMMIT** enclave 中的任何位置。 | | **MEM\_RELEASE**  0x00008000 | 释放指定的页面区域或占位符 (占位符，释放地址空间并可用于) 的其他分配。 在执行该操作之后，这些页面将处于可用状态。  如果指定此值， dwSize 必须为 0 (零) ，并且 lpAddress 必须指向保留区域时 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 函数返回的基址。 如果未满足上述任一条件，该函数将失败。  如果当前已提交区域中的任何页面，该函数将首先取消提交，然后释放它们。  如果尝试释放处于不同状态（一些保留和一些已提交）的页面，函数不会失败。 这意味着，无需首先确定当前承诺状态即可发布一系列页面。 |   使用 **MEM\_RELEASE** 时，此参数还可以指定以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_COALESCE\_PLACEHOLDERS**  0x00000001 | 若要合并两个相邻占位符，请指定 MEM\_RELEASE | MEM\_COALESCE\_PLACEHOLDERS。 合并占位符时， *lpAddress* 和 *dwSize* 必须与要合并的占位符的总体范围完全匹配。 | | **MEM\_PRESERVE\_PLACEHOLDER**  0x00000002 | 使用 [VirtualAlloc2 或](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc2)[Virtual2AllocFromApp](https://msdn.microsoft.com/en-us/library/Mt832850(v=VS.85).aspx)) 将占位符替换为专用分配后，将分配释放回占位符 (。  若要将占位符拆分为两个占位符，请指定 MEM\_RELEASE | MEM\_PRESERVE\_PLACEHOLDER。 |  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 0（零）。 要获得更多的错误信息，请调用 GetLastError。 注解 进程虚拟地址空间中的每个内存页都有一个 [页状态](https://learn.microsoft.com/zh-cn/windows/win32/Memory/page-state)。 **VirtualFree** 函数可以取消提交处于不同状态（一些已提交和一些未提交）的一系列页面。 这意味着，无需首先确定每个页面的当前承诺状态，即可取消提交一系列页面。 取消提交页面会释放其物理存储，无论是在内存中还是在磁盘上的分页文件中。  如果已取消提交但未释放页面，则其状态将更改为保留。 随后，可以调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 来提交它，或 **调用 VirtualFree** 来释放它。 尝试读取或写入保留页会导致访问冲突异常。  **VirtualFree** 函数可以释放处于不同状态（一些保留和一些已提交）的一系列页面。 这意味着，无需首先确定每个页面的当前承诺状态，即可发布一系列页面。 必须同时释放 [最初由 VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 函数保留的整个页面范围。  如果释放页面，则其状态将更改为释放，并且可用于后续分配操作。 释放或取消提交内存后，永远无法再次引用内存。 该内存中可能已存在的任何信息将永远消失。 尝试读取或写入免费页面会导致访问冲突异常。 如果需要保留信息，请不要取消提交或释放包含该信息的内存。  **VirtualFree** 函数可用于内存的 AWE 区域，并在释放地址空间时使该区域中的任何物理页映射失效。 但是，不会删除物理页，应用程序可以使用它们。 应用程序必须显式调用 [FreeUserPhysicalPages](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-freeuserphysicalpages) 以释放物理页面。 进程终止后，将自动清理所有资源。  **Windows 10版本 1709 及更高版本以及Windows 11：**若要在使用完 enclave 后将其删除，请调用 [DeleteEnclave](https://learn.microsoft.com/zh-cn/windows/win32/api/enclaveapi/nf-enclaveapi-deleteenclave)。 无法通过调用 **VirtualFree 或 VirtualFreeEx** 函数删除 VBS enclave。 仍可通过调用 **VirtualFree 或 VirtualFreeEx** 删除 SGX enclave。  **Windows 10版本 1507、Windows 10、版本 1511、Windows 10、版本 1607 和 Windows 10 版本 1703：**若要在使用完 enclave 后删除它，请调用 **VirtualFree** 或 [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfreeex) 函数并指定以下值：   * lpAddress 参数的 enclave 基址。 * 0 表示 dwSize 参数。 * dwFreeType 参数的**MEM\_RELEASE**。  示例 有关示例，请参阅 [保留和提交内存](https://learn.microsoft.com/zh-cn/windows/win32/Memory/reserving-and-committing-memory)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/virtual-memory-functions)  [VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfreeex)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## VirtualFreeEx 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 释放、取消提交或释放和取消提交指定进程的虚拟地址空间中的内存区域。 语法 C++复制  BOOL VirtualFreeEx(  [in] HANDLE hProcess,  [in] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD dwFreeType  ); 参数 [in] hProcess  进程的句柄。 函数释放进程的虚拟地址空间中的内存。  句柄必须具有 **PROCESS\_VM\_OPERATION** 访问权限。 有关详细信息，请参阅 [进程安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  [in] lpAddress  指向要释放的内存区域的起始地址的指针。  如果 *dwFreeType* 参数 **MEM\_RELEASE**，则 *lpAddress* 必须是保留区域时 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex) 函数返回的基址。  [in] dwSize  要释放的内存区域的大小（以字节为单位）。  如果 *dwFreeType* 参数 **MEM\_RELEASE**，则 *dwSize* 必须为 0 (零) 。 函数释放在 [对 VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex) 的初始分配调用中保留的整个区域。  如果 *dwFreeType***MEM\_DECOMMIT**，则函数将取消提交包含从 *lpAddress* 参数到 (lpAddress+dwSize)范围内的一个或多个字节的所有内存页。 例如，这意味着跨越页边界的 2 字节内存区域会导致两个页面都解除提交。 如果 *lpAddress* 是 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex) 返回的基址，而 *dwSize* 为 0 (零) ，则函数将取消提交 **VirtualAllocEx** 分配的整个区域。 之后，整个区域将处于保留状态。  [in] dwFreeType  自由操作的类型。 此参数须为下列值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_DECOMMIT**  0x00004000 | 取消提交已提交页面的指定区域。 操作后，页面将处于保留状态。  如果尝试取消提交未提交的页面，函数不会失败。 这意味着，无需先确定当前承诺状态，即可取消提交一系列页面。  当 *lpAddress* 参数提供 enclave 的基址时，不支持**MEM\_DECOMMIT**值。 对于不支持动态内存管理 (（即 SGX1) ）的 enclave 也是如此。 SGX2 enclave 允许MEM\_DECOMMIT enclave 中的任意位置。 | | **MEM\_RELEASE**  0x00008000 | 释放指定的页面区域或占位符 (占位符，释放地址空间并可用于) 的其他分配。 在执行该操作之后，这些页面将处于可用状态。  如果指定此值， *dwSize* 必须为 0 (零) ，并且 *lpAddress* 必须指向保留区域时 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数返回的基址。 如果未满足上述任一条件，该函数将失败。  如果当前已提交区域中的任何页面，该函数将首先取消提交，然后释放它们。  如果尝试释放处于不同状态（一些保留和一些已提交）的页面，函数不会失败。 这意味着，无需首先确定当前承诺状态即可发布一系列页面。 |   使用 **MEM\_RELEASE** 时，此参数还可以指定以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_COALESCE\_PLACEHOLDERS**  0x00000001 | 若要合并两个相邻占位符，请指定 MEM\_RELEASE | MEM\_COALESCE\_PLACEHOLDERS。 合并占位符时， *lpAddress* 和 *dwSize* 必须与要合并的占位符的总体范围完全匹配。 | | **MEM\_PRESERVE\_PLACEHOLDER**  0x00000002 | 使用 [VirtualAlloc2 或](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc2)[Virtual2AllocFromApp](https://msdn.microsoft.com/en-us/library/Mt832850(v=VS.85).aspx)) 将占位符替换为专用分配后，将分配释放回占位符 (。  若要将占位符拆分为两个占位符，请指定 MEM\_RELEASE | MEM\_PRESERVE\_PLACEHOLDER。 |  返回值 如果函数成功，则返回值为非零值。  如果函数失败，则返回值为 0（零）。 要获得更多的错误信息，请调用 GetLastError。 注解 进程虚拟地址空间中的每个内存页都有一个 [页状态](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/page-state)。 **VirtualFreeEx** 函数可以取消提交处于不同状态（一些已提交和一些未提交）的页面范围。 这意味着，无需首先确定每个页面的当前承诺状态，即可取消提交一系列页面。 取消提交页面会释放其物理存储，无论是在内存中还是在磁盘上的分页文件中。  如果已取消提交但未释放页面，则其状态将更改为保留。 随后，可以调用 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex) 来提交它，或 **调用 VirtualFreeEx** 来释放它。 尝试读取或写入保留页会导致访问冲突异常。  **VirtualFreeEx** 函数可以释放处于不同状态（一些保留和一些已提交）的一系列页面。 这意味着，无需首先确定每个页面的当前承诺状态，即可发布一系列页面。 必须同时释放 [最初由 VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex) 保留的整个页面范围。  如果释放页面，则其状态将更改为释放，并且可用于后续分配操作。 释放或取消提交内存后，永远无法再次引用内存。 该内存中可能已存在的任何信息将永远消失。 尝试读取或写入免费页面会导致访问冲突异常。 如果需要保留信息，请不要取消提交或释放包含该信息的内存。  **VirtualFreeEx** 函数可用于内存的 AWE 区域，并在释放地址空间时使该区域中的任何物理页映射失效。 但是，不会删除物理页，应用程序可以使用它们。 应用程序必须显式调用 [FreeUserPhysicalPages](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-freeuserphysicalpages) 以释放物理页面。 进程终止后，会自动清理所有资源。  **Windows 10版本 1709 及更高版本以及Windows 11：**若要在使用完 enclave 后将其删除，请调用 [DeleteEnclave](https://learn.microsoft.com/zh-cn/windows/win32/api/enclaveapi/nf-enclaveapi-deleteenclave)。 无法通过调用 [VirtualFree 或 VirtualFreeEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfree) 函数删除 VBS enclave。 仍可通过调用 **VirtualFree 或 VirtualFreeEx** 删除 SGX enclave。  **Windows 10版本 1507、Windows 10、版本 1511、Windows 10、版本 1607 和 Windows 10 版本 1703：**若要在使用完 enclave 后删除它，请调用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfree) 或 **VirtualFreeEx** 函数并指定以下值：   * lpAddress 参数的 enclave 基址。 * 0 表示 dwSize 参数。 * dwFreeType 参数的**MEM\_RELEASE**。  要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/virtual-memory-functions)  [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex) |

## VirtualLock 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 将进程的虚拟地址空间的指定区域锁定到物理内存中，确保后续访问该区域不会产生页面错误。 语法 C++复制  BOOL VirtualLock(  [in] LPVOID lpAddress,  [in] SIZE\_T dwSize  ); parameters [in] lpAddress  指向要锁定的页区域的基址的指针。  [in] dwSize  要锁定的区域的大小（以字节为单位）。 受影响页面的区域包括包含从 *lpAddress* 参数到 (lpAddress+dwSize)范围内的一个或多个字节的所有页面。 这意味着跨页边界的 2 字节范围会导致两个页面被锁定。 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 必须提交指定区域中的所有页面。 无法锁定受 **PAGE\_NOACCESS** 保护的内存。  将页面锁定到内存中可能会降低系统的性能，方法是减少可用 RAM，并强制系统将其他关键页交换到分页文件。 每个版本的 Windows 对进程可锁定的最大页数都有限制。 此限制是特意较小的，以避免严重的性能下降。 需要锁定更多页的应用程序必须首先调用 [SetProcessWorkingSetSize](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-setprocessworkingsetsize) 函数，以增加其最小和最大工作集大小。 进程可锁定的最大页数等于其最小工作集中的页数减去少量开销。  进程锁定的页面将保留在物理内存中，直到进程解锁或终止。 保证这些页面在锁定时不会写入页面文件。  若要解锁锁定页面的区域，请使用 [VirtualUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualunlock) 函数。 当进程终止时，锁定的页面会自动解锁。  此函数与 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 或 [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock) 函数不一样，因为它不会递增锁计数并将句柄转换为指针。 虚拟页面没有锁计数，因此无需多次调用 [VirtualUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualunlock) 函数即可解锁页面区域。 示例 有关示例，请参阅 [创建保护页](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-guard-pages)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [SetProcessWorkingSetSize](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-setprocessworkingsetsize)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualunlock) |

## VirtualProtect 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 更改调用进程的虚拟地址空间中已提交页面区域的保护。  若要更改任何进程的访问保护，请使用 [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualprotectex) 函数。 语法 C++复制  BOOL VirtualProtect(  [in] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD flNewProtect,  [out] PDWORD lpflOldProtect  ); 参数 [in] lpAddress  要更改其访问保护属性的页区域的起始页的地址。  指定区域中的所有页面都必须在使用 MEM\_RESERVE 调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 或 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex) 函数时分配 **的**同一保留区域。 这些页面不能跨通过单独调用 **VirtualAlloc 或 VirtualAllocEx**（使用 **MEM\_RESERVE**）分配的相邻保留区域。  [in] dwSize  要更改其访问保护属性的区域的大小（以字节为单位）。 受影响页面的区域包括从 ***lpAddress*** 参数到 (lpAddress+dwSize)的范围中包含一个或多个字节的所有页面。 这意味着跨页边界的 2 字节范围会导致两个页面的保护属性发生更改。  [in] flNewProtect  内存保护选项。 此参数可以是 [内存保护常量](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-protection-constants)之一。  对于映射视图，此值必须与映射视图时指定的访问保护兼容 (请参阅 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile)、 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffileex) 和 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-mapviewoffileexnuma)) 。  [out] lpflOldProtect  指向变量的指针，该变量接收指定页区域中第一页的先前访问保护值。 如果此参数为 **NULL** 或未指向有效的变量，则该函数将失败。 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 只能在提交的页面上设置访问保护值。 如果未提交指定区域中任何页面的状态，则函数将失败并返回 ，而不会修改指定区域中任何页面的访问保护。  **PAGE\_GUARD**保护修饰符建立保护页。 保护页充当一次性访问警报。 有关更多信息，请参见[创建保护页](https://learn.microsoft.com/zh-cn/windows/win32/Memory/creating-guard-pages)。  最好避免使用 **VirtualProtect** 更改[由 GlobalAlloc、HeapAlloc 或 LocalAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-globalalloc) 分配的内存块上的页面保护，因为单个页上可以存在多个内存块。 堆管理器假定堆中的所有页面至少授予读取和写入访问权限。  保护可执行的区域时，调用程序负责在代码设置到位后通过适当调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 来确保缓存一致性。 否则，尝试在新可执行区域之外执行代码可能会产生不可预知的结果。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [内存保护常量](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-protection-constants)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/virtual-memory-functions)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc)  [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualprotectex)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## VirtualProtectEx 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 更改指定进程的虚拟地址空间中已提交页面区域的保护。 语法 C++复制  BOOL VirtualProtectEx(  [in] HANDLE hProcess,  [in] LPVOID lpAddress,  [in] SIZE\_T dwSize,  [in] DWORD flNewProtect,  [out] PDWORD lpflOldProtect  ); 参数 [in] hProcess  要更改其内存保护的进程句柄。 句柄必须具有 **PROCESS\_VM\_OPERATION** 访问权限。 有关详细信息，请参阅 [进程安全和访问权限](https://learn.microsoft.com/zh-cn/windows/win32/ProcThread/process-security-and-access-rights)。  [in] lpAddress  指向要更改其访问保护属性的页区域的基址的指针。  指定区域中的所有页面都必须在使用 MEM\_RESERVE 调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 或 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex) 函数时分配 **的**同一保留区域。 这些页面不能跨通过单独调用 **VirtualAlloc 或 VirtualAllocEx**（使用 **MEM\_RESERVE**）分配的相邻保留区域。  [in] dwSize  访问保护属性已更改的区域的大小（以字节为单位）。 受影响页面的区域包括从 lpAddress 参数到 (lpAddress+dwSize)的范围中包含一个或多个字节的所有页面。 这意味着跨页边界的 2 字节范围会导致两个页面的保护属性发生更改。  [in] flNewProtect  内存保护选项。 此参数可以是 [内存保护常量](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-protection-constants)之一。  对于映射视图，此值必须与映射视图时指定的访问保护兼容 (请参阅 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile)、 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffileex) 和 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-mapviewoffileexnuma)) 。  [out] lpflOldProtect  指向变量的指针，该变量接收指定页区域中第一页的先前访问保护。 如果此参数为 **NULL** 或未指向有效的变量，则该函数将失败。 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 只能在提交的页面上设置访问保护值。 如果未提交指定区域中任何页面的状态，则函数将失败并返回 ，而不会修改指定区域中任何页面的访问保护。  **PAGE\_GUARD**保护修饰符建立保护页。 保护页充当一次性访问警报。 有关更多信息，请参见[创建保护页](https://learn.microsoft.com/zh-cn/windows/win32/Memory/creating-guard-pages)。  最好避免使用 **VirtualProtectEx** 更改 [GlobalAlloc、HeapAlloc 或 LocalAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-globalalloc) 分配的内存块上的页面保护，因为单个页面上可能有多个内存块。 堆管理器假定堆中的所有页面至少授予读取和写入访问权限。  保护可执行的区域时，调用程序负责在代码设置到位后通过适当调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 来确保缓存一致性。 否则，尝试在新可执行区域之外执行代码可能会产生不可预知的结果。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [内存保护常量](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-protection-constants)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/virtual-memory-functions)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc)  [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect)  [VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualqueryex) |

## VirtualProtectFromApp 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 更改调用进程的虚拟地址空间中已提交页面区域的保护。 语法 C++复制  BOOL VirtualProtectFromApp(  [in] PVOID Address,  [in] SIZE\_T Size,  [in] ULONG NewProtection,  [out] PULONG OldProtection  ); 参数 [in] Address  一个指针，该地址描述要更改其访问保护属性的页面区域的起始页。  指定区域中的所有页面都必须在使用 MEM\_RESERVE 调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)、 [VirtualAllocFromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocfromapp) 或 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex) 函数时分配 **的**同一保留区域。 页面不能跨通过单独调用 **VirtualAlloc、VirtualAllocFromApp** 或 **VirtualAllocEx**（使用 MEM\_RESERVE）分配**的**相邻保留区域。  [in] Size  要更改其访问保护属性的区域的大小（以字节为单位）。 受影响页面的区域包括从 *Address* 参数到 (Address+Size)的范围中包含一个或多个字节的所有页面。 这意味着跨页边界的 2 字节范围会导致两个页面的保护属性发生更改。  [in] NewProtection  内存保护选项。 此参数可以是 [内存保护常量](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-protection-constants)之一。  对于映射视图，此值必须与映射视图时指定的访问保护兼容 (请参阅 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)、 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 和 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma)) 。  以下常量生成错误：   * **PAGE\_EXECUTE\_READWRITE** * **PAGE\_EXECUTE\_WRITECOPY**   只有具有 **codeGeneration** 功能的应用才允许使用以下常量：   * **PAGE\_EXECUTE** * **PAGE\_EXECUTE\_READ**   [out] OldProtection  指向变量的指针，该变量接收指定页区域中第一页的先前访问保护值。 如果此参数为 **NULL** 或未指向有效的变量，则该函数将失败。 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 你可以从具有实时 (JIT) 功能的 Windows 应用商店应用调用 **VirtualProtectFromApp** ，以使用 JIT 功能。 应用必须在应用清单文件中包含 **codeGeneration** 功能才能使用 JIT 功能。  只能在提交的页面上设置访问保护值。 如果未提交指定区域中任何页面的状态，则函数将失败并返回 ，而不会修改指定区域中任何页面的访问保护。  **PAGE\_GUARD**保护修饰符建立保护页。 保护页充当一次性访问警报。 有关更多信息，请参见[创建保护页](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-guard-pages)。  最好避免使用 **VirtualProtectFromApp** 更改 [GlobalAlloc、HeapAlloc 或 LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 分配的内存块上的页面保护，因为单个页面上可能存在多个内存块。 堆管理器假定堆中的所有页面至少授予读取和写入访问权限。  **VirtualProtectFromApp** 允许将页面标记为可执行文件，但不允许同时设置写入和执行权限。  保护可执行的区域时，调用程序负责在代码设置到位后通过适当调用 [FlushInstructionCache](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-flushinstructioncache) 来确保缓存一致性。 否则，尝试在新可执行区域之外执行代码可能会产生不可预知的结果。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h) | | **Library** | WindowsApp.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAllocFromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocfromapp)  [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotect)  [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectex) |

## VirtualQuery 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关调用进程的虚拟地址空间中的页范围的信息。  若要检索有关另一个进程的地址空间中的页面范围的信息，请使用 [VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualqueryex) 函数。 语法 C++复制  SIZE\_T VirtualQuery(  [in, optional] LPCVOID lpAddress,  [out] PMEMORY\_BASIC\_INFORMATION lpBuffer,  [in] SIZE\_T dwLength  ); 参数 [in, optional] lpAddress  指向要查询的页区域的基址的指针。 此值向下舍入到下一页边界。 若要确定主计算机上的页面大小，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  如果 lpAddress 指定的地址高于进程可访问的最高内存地址，则函数将失败 **并ERROR\_INVALID\_PARAMETER**。  [out] lpBuffer  指向 [MEMORY\_BASIC\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-memory_basic_information) 结构的指针，在该结构中返回有关指定页范围的信息。  [in] dwLength  lpBuffer 参数指向的缓冲区的大小（以字节为单位）。 返回值 返回值是在信息缓冲区中返回的实际字节数。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 可能的错误值包括 **ERROR\_INVALID\_PARAMETER**。 注解 **VirtualQuery** 提供有关从共享以下属性的指定地址开始的连续页区域的信息：   * 所有页面的状态 (**MEM\_COMMIT**、 **MEM\_RESERVE**、 **MEM\_FREE**、 **MEM\_PRIVATE**、 **MEM\_MAPPED**或 **MEM\_IMAGE**) 相同。 * 如果初始页面不可用，则区域中的所有页面都是通过单个调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc)、 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile) 或以下函数扩展版本之一创建的页的相同初始分配的一部分： [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex)、 [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualallocexnuma)、 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffileex)、 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-mapviewoffileexnuma)。 * 授予对所有页面的访问权限 (**PAGE\_READONLY、PAGE\_READWRITE**、**PAGE\_NOACCESS**、**PAGE\_WRITECOPY**、**PAGE\_EXECUTE**、**PAGE\_EXECUTE\_READ**、**PAGE\_EXECUTE\_READWRITE**、**PAGE\_EXECUTE\_WRITECOPY**、**PAGE\_GUARD**或**PAGE\_NOCACHE**) 。   函数确定区域中第一页的属性，然后扫描后续页面，直到扫描整个页面范围或遇到具有不匹配属性集的页面。 函数返回具有匹配属性的页区域的属性和大小（以字节为单位）。 例如，如果有一个 40 兆字节 (MB) 可用内存区域，并且对区域中 10 MB 的页面上调用 **VirtualQuery** ，则函数将获取 **MEM\_FREE** 状态和大小为 30 MB 的状态。  如果修改了共享的写入复制页，则它将成为修改该页面的进程专用的。 但是， **VirtualQuery** 函数将继续报告数据视图 **)**MEM\_MAPPED (等 **页面，或者** 针对可执行图像视图) 而不是MEM\_PRIVATE MEM\_IMAGE **(**。 若要检测特定页面是否发生了写入时复制，请使用 [VirtualLock](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtuallock) 函数访问页面或锁定该页，以确保页面驻留在内存中，然后使用 [QueryWorkingSetEx](https://learn.microsoft.com/zh-cn/windows/win32/api/psapi/nf-psapi-queryworkingsetex) 函数检查页面扩展工作集信息中的**共享**位。 如果 **“共享”** 位是清除的，则页面是私有的。  此函数报告调用进程内存中的页面区域， [VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualqueryex) 函数报告指定进程内存中的页面区域。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MEMORY\_BASIC\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-memory_basic_information)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/win32/Memory/virtual-memory-functions)  [VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualqueryex)  [VBS enclave 中可用的 Vertdll API](https://learn.microsoft.com/zh-cn/windows/win32/trusted-execution/enclaves-available-in-vertdll) |

## VirtualQueryEx 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关指定进程的虚拟地址空间中的页范围的信息。 语法 C++复制  SIZE\_T VirtualQueryEx(  [in] HANDLE hProcess,  [in, optional] LPCVOID lpAddress,  [out] PMEMORY\_BASIC\_INFORMATION lpBuffer,  [in] SIZE\_T dwLength  ); parameters [in] hProcess  查询其内存信息的进程句柄。 句柄必须已使用 **PROCESS\_QUERY\_INFORMATION** 访问权限打开，从而允许使用句柄从进程对象中读取信息。 有关详细信息，请参阅 [进程安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  [in, optional] lpAddress  指向要查询的页区域的基址的指针。 此值向下舍入到下一页边界。 若要确定主计算机上的页面大小，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  如果 *lpAddress* 指定的地址高于进程可访问的最高内存地址，则函数将失败 **并ERROR\_INVALID\_PARAMETER**。  [out] lpBuffer  指向 [MEMORY\_BASIC\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-memory_basic_information) 结构的指针，在该结构中返回有关指定页范围的信息。  [in] dwLength  *lpBuffer* 参数指向的缓冲区的大小（以字节为单位）。 返回值 返回值是在信息缓冲区中返回的实际字节数。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 可能的错误值包括 **ERROR\_INVALID\_PARAMETER**。 注解 **VirtualQueryEx** 提供有关从共享以下属性的指定地址开始的连续页区域的信息：   * 所有页面的状态 (**MEM\_COMMIT**、 **MEM\_RESERVE**、 **MEM\_FREE**、 **MEM\_PRIVATE**、 **MEM\_MAPPED**或 **MEM\_IMAGE**) 相同。 * 如果初始页面不可用，则区域中的所有页面都是通过单个调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)、 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 或以下函数扩展版本之一创建的页的相同初始分配的一部分： [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex)、 [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma)、 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)、 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma)。 * 授予对所有页面的访问权限 (**PAGE\_READONLY、PAGE\_READWRITE**、**PAGE\_NOACCESS**、**PAGE\_WRITECOPY**、**PAGE\_EXECUTE**、**PAGE\_EXECUTE\_READ**、**PAGE\_EXECUTE\_READWRITE**、**PAGE\_EXECUTE\_WRITECOPY**、**PAGE\_GUARD**或**PAGE\_NOCACHE**) 。   **VirtualQueryEx** 函数确定区域中第一页的属性，然后扫描后续页面，直到扫描整个页面范围，或遇到具有非匹配属性集的页面。 函数返回具有匹配属性的页区域的属性和大小（以字节为单位）。 例如，如果有一个 40 兆字节 (MB) 可用内存区域，并在区域中 10 MB 的页面上调用 **VirtualQueryEx** ，则函数将获取 **MEM\_FREE** 状态，大小为 30 MB。  如果修改了共享的写入复制页，则它将成为修改该页面的进程专用的。 但是， **VirtualQueryEx** 函数将继续报告数据视图 **)**MEM\_MAPPED (等页面，或者) 而不是 **MEM\_PRIVATE** 可执行图像视图的MEM\_IMAGE **(**。 若要检测特定页面是否发生了写入时复制，请使用 [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock) 函数访问该页或锁定该页，以确保该页驻留在内存中，然后使用 [QueryWorkingSet](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-queryworkingset) 或 [QueryWorkingSetEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-queryworkingsetex) 函数检查页面扩展工作集信息中的**共享**位。 如果 **“共享”** 位是清除的，则页面是私有的。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MEMORY\_BASIC\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-memory_basic_information)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex)  [VirtualProtectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotectex) |

## VirtualUnlock 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 解锁进程的虚拟地址空间中的指定页范围，使系统能够在必要时将页面交换到分页文件。 语法 C++复制  BOOL VirtualUnlock(  [in] LPVOID lpAddress,  [in] SIZE\_T dwSize  ); parameters [in] lpAddress  指向要解锁的页区域的基址的指针。  [in] dwSize  要解锁的区域的大小（以字节为单位）。 受影响页面的区域包括从 *lpAddress* 参数到 (lpAddress+dwSize)的范围中包含一个或多个字节的所有页面。 这意味着跨页边界的 2 字节范围会导致两个页面都解锁。 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 要使函数成功，指定的范围不需要与传递给之前对 [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock) 函数的调用的范围匹配，但必须锁定该范围中的所有页面。 如果未锁定指定范围中的任何页， **VirtualUnlock** 将从工作集中删除此类页面，将最后一个错误设置为 **ERROR\_NOT\_LOCKED**，并返回 **FALSE**。  对未锁定的内存范围调用 **VirtualUnlock** 会从进程的工作集中释放页面。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [虚拟内存函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-memory-functions)  [VirtualLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtuallock) |

## QueryVirtualMemoryInformation 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **QueryVirtualMemoryInformation** 函数返回有关指定进程的虚拟地址空间中的一页或一组页面的信息。 语法 C++复制  BOOL QueryVirtualMemoryInformation(  [in] HANDLE Process,  [in] const VOID \*VirtualAddress,  [in] WIN32\_MEMORY\_INFORMATION\_CLASS MemoryInformationClass,  [out] PVOID MemoryInformation,  [in] SIZE\_T MemoryInformationSize,  [out, optional] PSIZE\_T ReturnSize  ); 参数 [in] Process  要查询的页面所在的上下文的进程句柄。  [in] VirtualAddress  要查询的页区域的地址。 此值向下舍入到下一个主机页地址边界。  [in] MemoryInformationClass  要检索其相关信息的内存信息类。 唯一支持的值是 **MemoryRegionInfo**。  [out] MemoryInformation  指向接收指定信息的缓冲区的指针。  如果 *MemoryInformationClass* 参数的值为 **MemoryRegionInfo**，则此参数必须指向 [WIN32\_MEMORY\_REGION\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/ns-memoryapi-win32_memory_region_information) 结构。  [in] MemoryInformationSize  指定内存信息缓冲区的长度（以字节为单位）。  [out, optional] ReturnSize  一个可选指针，如果指定，则接收放置在内存信息缓冲区中的字节数。 返回值 成功时返回 **TRUE** 。 如果失败，则返回 **FALSE** 。 要获得更多的错误信息，请调用 GetLastError。 注解 如果 *MemoryInformationClass* 参数的值为 **MemoryRegionInfo**， *则 MemoryInformation* 参数必须指向 [WIN32\_MEMORY\_REGION\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/ns-memoryapi-win32_memory_region_information) 结构。 *VirtualAddress* 参数必须指向有效内存分配中的地址。 如果 *VirtualAddress* 参数指向未分配的内存区域，则函数将失败。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h) | | **Library** | Onecore.lib | | **DLL** | Api-ms-win-core-memory-l1-1-4.dll |  另请参阅 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)  [MEMORY\_BASIC\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-memory_basic_information)  [WIN32\_MEMORY\_REGION\_INFORMATION](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/ns-memoryapi-win32_memory_region_information) |

# 内存映射文件.

## 是一种把磁盘上的一个文件直接映射到内存中的方法,适合进程之间传递比较大的文件的场景.他比读写文件的IO操作效率更高.可以用这种方法来实现一些比较高效的数据结构算法.这种方式不需要读写文件,也不需要处理文件内容缓冲区.也是一种进程间共享数据的方法.适合大型数据文件在进程之间共享的场景

## CreateFileMappingA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为指定文件创建或打开命名或未命名的文件映射对象。  若要指定物理内存的 NUMA 节点，请参阅 [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa)。 语法 C++复制  HANDLE CreateFileMappingA(  [in] HANDLE hFile,  [in, optional] LPSECURITY\_ATTRIBUTES lpFileMappingAttributes,  [in] DWORD flProtect,  [in] DWORD dwMaximumSizeHigh,  [in] DWORD dwMaximumSizeLow,  [in, optional] LPCSTR lpName  ); 参数 [in] hFile  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 有关详细信息，请参阅 [文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)。  如果 *hFile***是INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMapping** 会创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  [in, optional] lpFileMappingAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定是否可由子进程继承返回的句柄。 **SECURITY\_ATTRIBUTES** 结构的 **lpSecurityDescriptor** 成员为新的文件映射对象指定安全描述符。  如果 *lpFileMappingAttributes* 为 **NULL**，则无法继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制列出了来自创建者的主要令牌或模拟令牌的文件映射对象的默认安全描述符中 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] flProtect  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_EXECUTE\_READ**  0x20 | 允许映射视图，以便进行只读、写入时复制或执行访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_READWRITE**  0x40 | 允许映射视图，以便进行只读、写入时复制、读/写或执行访问。  必须使用**GENERIC\_READ**、**GENERIC\_WRITE**和**GENERIC\_EXECUTE**访问权限创建 *hFile* 参数指定的文件句柄。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_WRITECOPY**  0x80 | 允许映射视图，以便进行只读、写入时复制或执行访问。 此值等效于 **PAGE\_EXECUTE\_READ**。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Vista：**此值在 Windows Vista SP1 中之前不可用。  **Windows Server 2003 和 Windows XP：**不支持此值。 | | **PAGE\_READONLY**  0x02 | 允许映射视图，以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图，以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |     应用程序可以通过将以下一个或多个属性与前面的页面保护值组合来为文件映射对象指定以下一个或多个属性。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定在将文件的视图映射到进程地址空间时，将提交而不是保留整个页面范围。 系统必须有足够的可提交页面来保存整个映射。 否则， **CreateFileMapping** 将失败。  对于由可执行图像文件或数据文件支持的文件映射对象， (*hfile* 参数是文件) 句柄，则此属性不起作用。  **SEC\_COMMIT** 不能与 **SEC\_RESERVE**结合使用。  如果未指定任何属性，则假定 **SEC\_COMMIT** 。 | | **SEC\_IMAGE**  0x1000000 | 指定 *hFile* 参数指定的文件是可执行映像文件。  **SEC\_IMAGE** 属性必须与页面保护值（如**PAGE\_READONLY**）结合使用。 但是，此页面保护值对可执行映像文件的视图没有影响。 对可执行映像文件视图的页面保护由可执行文件本身确定。  没有其他属性对 **SEC\_IMAGE**有效。 | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。**  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在Windows Server 2012和Windows 8之前，不支持此值。 | | **SEC\_LARGE\_PAGES**  0x80000000 | 使大型页面可用于操作系统分页文件支持的文件映射对象， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) 。 由可执行图像文件或数据文件支持的文件映射对象不支持此属性， (*hFile* 参数是可执行映像或数据文件的句柄) 。  文件映射对象的最大大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数返回的大页最小大小的倍数。 否则， **CreateFileMapping** 将失败。 映射使用 **SEC\_LARGE\_PAGES** 创建的文件映射对象的视图时，基址和视图大小也必须是最小大页面大小的倍数。  **SEC\_LARGE\_PAGES** 要求在调用方令牌中启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。  如果指定 **了SEC\_LARGE\_PAGES** ，还必须指定 **SEC\_COMMIT** 。  **Windows Server 2003：**直到 Windows Server 2003 SP1 才支持此值。  **Windowsxp：**不支持此值。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可缓存。  应用程序不应使用此属性，除非设备显式需要。 将互锁 **函数与SEC\_NOCACHE** 映射的内存配合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 | | **SEC\_RESERVE**  0x4000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定当文件的视图映射到进程地址空间时，将保留整个页面范围供进程以后使用，而不是提交。  可以在后续调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数时提交保留页。 提交页面后，无法使用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 函数释放或取消提交页面。  此属性对由可执行映像文件或数据文件支持的文件映射对象无效， (*hfile* 参数是文件) 的句柄。  **SEC\_RESERVE** 不能与 **SEC\_COMMIT** 结合使用。 | | **SEC\_WRITECOMBINE**  0x40000000 | 将所有页面设置为写合并。  除非设备明确需要，否则应用程序不应使用此属性。 将互锁函数 **与SEC\_WRITECOMBINE映射** 的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 要求设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。  **Windows Server 2003 和 Windows XP：**在 Windows Vista 之前，不支持此标志。 |   [in] dwMaximumSizeHigh  文件映射对象最大大小的高序 **DWORD** 。  [in] dwMaximumSizeLow  文件映射对象最大大小的低序 **DWORD** 。  如果此参数和 *dwMaximumSizeHigh* 为 0 (零) ，则文件映射对象的最大大小等于 *hFile* 标识的文件的当前大小。  尝试映射长度为 0 (零的文件) 失败，错误代码 **为 ERROR\_FILE\_INVALID**。 应用程序应测试长度为 0 (零的文件) 并拒绝这些文件。  [in, optional] lpName  文件映射对象的名称。  如果此参数与现有映射对象的名称匹配，则函数会请求访问具有 *flProtect* 指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况是因为这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/win32/secauthz/privilege-constants#constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录的用户使用会话 1 (一个) ，依此而过。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。 返回值 如果函数成功，则返回值是新创建的文件映射对象的句柄。  如果对象在函数调用之前存在，则函数将返回一个句柄，该句柄指向现有对象 (其当前大小，而不是指定大小 [) ，GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 创建文件映射对象后，文件大小不得超过文件映射对象的大小;如果存在，并非所有文件内容都可用于共享。  如果应用程序为文件映射对象指定的大小大于磁盘上实际命名文件的大小，并且页面保护允许写入访问 (即 *flProtect* 参数指定 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**) ，则磁盘上的文件将增加以匹配文件映射对象的指定大小。 如果文件已扩展，则不保证文件旧端与文件新端之间的文件内容为零;行为由文件系统定义。 如果磁盘上的文件无法增加， **CreateFileMapping** 将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_DISK\_FULL**。  操作系统分页文件支持的文件映射对象中页面的初始内容为 0 (零) 。  **CreateFileMapping** 返回的句柄对新的文件映射对象具有完全访问权限，并且可以与需要文件映射对象的句柄的任何函数一起使用。  多个进程可以通过使用单个共享文件映射对象或创建由同一文件支持的单独文件映射对象来共享同一文件的视图。 通过继承进程创建时的句柄、复制句柄或按名称打开文件映射对象，多个进程可以共享单个文件映射对象。 有关详细信息，请参阅 [CreateProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa)、 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数。  创建文件映射对象实际上不会将视图映射到进程地址空间。 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 和 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 函数将文件的视图映射到进程地址空间中。  有一个重要例外，从同一文件支持的任何文件映射对象派生的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **CreateFileMapping** 适用于远程文件，但它不会使它们保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只能看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  映射文件和使用输入和输出访问的文件 (I/O) 函数 ([ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 和 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)) 不一定是一致的。  文件映射对象的映射视图保留对该对象的内部引用，并且文件映射对象在释放对其的所有引用之前不会关闭。 因此，若要完全关闭文件映射对象，应用程序必须通过调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 取消映射文件映射对象的所有映射视图，并通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 关闭文件映射对象句柄。 可以按任意顺序调用这些函数。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 请注意，此特权检查仅限于创建文件映射对象，不适用于打开现有对象。 例如，如果服务或系统在全局命名空间中创建文件映射对象，则在任何会话中运行的任何进程都可以访问该文件映射对象，前提是调用方具有所需的访问权限。  **Windowsxp：**上一段中所述的要求是在 Windows Server 2003 和 Windows XP SP2 中引入的  使用结构化异常处理来保护写入文件视图或从文件视图中读取的任何代码。 有关详细信息，请参阅 [从文件视图中读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  若要具有具有可执行权限的映射，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 **CreateFileMapping**，然后使用 或 FILE\_MAP\_EXECUTE | FILE\_MAP\_READ调用 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)FILE\_MAP\_EXECUTE | FILE\_MAP\_WRITE。  在 Windows Server 2012 中，以下技术支持此函数。  展开表   |  |  | | --- | --- | | **技术** | **支持** | | 服务器消息块 (SMB) 3.0 协议 | 是 | | SMB 3.0 透明故障转移 (TFO) | 是 | | 具有横向扩展文件共享的 SMB 3.0 (SO) | 是 | | 群集共享卷文件系统 (CSV) | 是 | | 弹性文件系统 (ReFS) | 是 |    示例 有关示例，请参阅 [创建命名共享内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-named-shared-memory) 或使用 [大型页面创建文件映射](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-mapping-using-large-pages)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h、Memoryapi.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)  [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa)  [创建文件映射对象](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-mapping-object)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  文件映射函数  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## CreateFileMappingW 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为指定文件创建或打开命名或未命名的文件映射对象。  若要指定物理内存的 NUMA 节点，请参阅 [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa)。 语法 C++复制  HANDLE CreateFileMappingW(  [in] HANDLE hFile,  [in, optional] LPSECURITY\_ATTRIBUTES lpFileMappingAttributes,  [in] DWORD flProtect,  [in] DWORD dwMaximumSizeHigh,  [in] DWORD dwMaximumSizeLow,  [in, optional] LPCWSTR lpName  ); parameters [in] hFile  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 有关详细信息，请参阅 [文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/FileIO/file-security-and-access-rights)。  如果 *hFile***是INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMapping** 会创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  [in, optional] lpFileMappingAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定是否可由子进程继承返回的句柄。 **SECURITY\_ATTRIBUTES** 结构的 **lpSecurityDescriptor** 成员为新的文件映射对象指定安全描述符。  如果 *lpAttributes* 为 **NULL**，则无法继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制列出了来自创建者的主要令牌或模拟令牌的文件映射对象的默认安全描述符中 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] flProtect  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_EXECUTE\_READ**  0x20 | 允许映射视图，以便进行只读、写入时复制或执行访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_READWRITE**  0x40 | 允许映射视图，以便进行只读、写入时复制、读/写或执行访问。  必须使用**GENERIC\_READ**、**GENERIC\_WRITE**和**GENERIC\_EXECUTE**访问权限创建 *hFile* 参数指定的文件句柄。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_WRITECOPY**  0x80 | 允许映射视图，以便进行只读、写入时复制或执行访问。 此值等效于 **PAGE\_EXECUTE\_READ**。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Vista：**此值在 Windows Vista SP1 中之前不可用。  **Windows Server 2003 和 Windows XP：**不支持此值。 | | **PAGE\_READONLY**  0x02 | 允许映射视图，以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图，以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |     应用程序可以通过将以下一个或多个属性与前面的页面保护值组合来为文件映射对象指定以下一个或多个属性。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定在将文件的视图映射到进程地址空间时，将提交而不是保留整个页面范围。 系统必须有足够的可提交页面来保存整个映射。 否则， **CreateFileMapping** 将失败。  对于由可执行图像文件或数据文件支持的文件映射对象， (*hfile* 参数是文件) 句柄，则此属性不起作用。  **SEC\_COMMIT** 不能与 **SEC\_RESERVE**结合使用。  如果未指定任何属性，则假定 **SEC\_COMMIT** 。 但是， **在将SEC\_COMMIT** 与另一个需要它的 **SEC\_** 属性组合时，必须显式指定它。 | | **SEC\_IMAGE**  0x1000000 | 指定 *hFile* 参数指定的文件是可执行映像文件。  **SEC\_IMAGE** 属性必须与页面保护值（如**PAGE\_READONLY**）结合使用。 但是，此页面保护值对可执行映像文件的视图没有影响。 对可执行映像文件视图的页面保护由可执行文件本身确定。  没有其他属性对 **SEC\_IMAGE**有效。 | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。**  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows Server 2012 和 Windows 8 之前不支持此值。 | | **SEC\_LARGE\_PAGES**  0x80000000 | 使大型页面可用于操作系统分页文件支持的文件映射对象， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) 。 由可执行图像文件或数据文件支持的文件映射对象不支持此属性， (*hFile* 参数是可执行映像或数据文件的句柄) 。  文件映射对象的最大大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数返回的大页最小大小的倍数。 否则， **CreateFileMapping** 将失败。 映射使用 **SEC\_LARGE\_PAGES** 创建的文件映射对象的视图时，基址和视图大小也必须是最小大页面大小的倍数。  **SEC\_LARGE\_PAGES** 要求在调用方令牌中启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。  如果指定 **了SEC\_LARGE\_PAGES** ，还必须指定 **SEC\_COMMIT** 。  **Windows Server 2003：**直到 Windows Server 2003 SP1 才支持此值。  **Windowsxp：**不支持此值。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可缓存。  应用程序不应使用此属性，除非设备显式需要。 将互锁 **函数与SEC\_NOCACHE** 映射的内存配合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 | | **SEC\_RESERVE**  0x4000000 | 如果文件映射对象由操作系统分页文件支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定当文件视图映射到进程地址空间时，将保留整个页面范围供进程以后使用，而不是提交。  可以在后续调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数时提交保留页。 提交页面后，无法使用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 函数释放或取消提交页面。  对于由可执行图像文件或数据文件支持的文件映射对象， (*hfile* 参数是文件) 句柄，则此属性不起作用。  **SEC\_RESERVE** 不能与 **SEC\_COMMIT** 结合使用。 | | **SEC\_WRITECOMBINE**  0x40000000 | 设置要写入组合的所有页面。  应用程序不应使用此属性，除非设备显式需要。 将互锁 **函数与SEC\_WRITECOMBINE** 映射的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。  **Windows Server 2003 和 Windows XP：**此标志在 Windows Vista 之前不受支持。 |   [in] dwMaximumSizeHigh  文件映射对象最大大小的高阶 **DWORD** 。  [in] dwMaximumSizeLow  文件映射对象最大大小的低序 **DWORD** 。  如果此参数和 *dwMaximumSizeHigh* 为 0 (零) ，则文件映射对象的最大大小等于 *hFile* 标识的文件的当前大小。  尝试映射长度为 0 (零的文件) 失败，错误代码为 **ERROR\_FILE\_INVALID**。 应用程序应测试长度为 0 (零) 的文件，并拒绝这些文件。  [in, optional] lpName  文件映射对象的名称。  如果此参数与现有映射对象的名称匹配，则函数会请求访问具有 *flProtect* 指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况的原因是这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录用户使用会话 1 (一个) ，依此依此。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。 返回值 如果函数成功，则返回值是新创建的文件映射对象的句柄。  如果对象在函数调用之前存在，则函数将 (当前大小（而不是指定大小) ）返回现有对象的句柄， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 创建文件映射对象后，文件大小不得超过文件映射对象的大小;如果存在，并非所有文件内容都可用于共享。  如果应用程序为文件映射对象指定的大小大于磁盘上实际命名文件的大小，并且页面保护允许写入访问 (即 *flProtect* 参数指定 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**) ，则磁盘上的文件将增加以匹配文件映射对象的指定大小。 如果扩展了文件，则文件旧端与新文件端之间的文件内容不保证为零;行为由文件系统定义。 如果磁盘上的文件无法增加， **CreateFileMapping** 将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_DISK\_FULL**。  操作系统分页文件支持的文件映射对象中页面的初始内容为 0 (零) 。  **CreateFileMapping** 返回的句柄具有对新文件映射对象的完全访问权限，并且可以与需要文件映射对象句柄的任何函数一起使用。  多个进程可以通过使用单个共享文件映射对象或创建由同一文件支持的单独文件映射对象来共享同一文件的视图。 单个文件映射对象可由多个进程共享，方法是在创建进程时继承句柄、复制句柄或按名称打开文件映射对象。 有关详细信息，请参阅 [CreateProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa)、 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数。  创建文件映射对象实际上不会将视图映射到进程地址空间。 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 和 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 函数将文件视图映射到进程地址空间。  有一个重要例外，派生自由同一文件支持的任何文件映射对象的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **CreateFileMapping** 适用于远程文件，但无法使其保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  映射文件和使用输入和输出访问的文件 (I/O) 函数 ([ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 和 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)) 不一定一致。  文件映射对象的映射视图保持对 对象的内部引用，并且文件映射对象在释放对其的所有引用之前不会关闭。 因此，若要完全关闭文件映射对象，应用程序必须通过调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 取消映射文件映射对象的所有映射视图，并通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 关闭文件映射对象句柄。 可以按任何顺序调用这些函数。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 请注意，此权限检查仅限于创建文件映射对象，不适用于打开现有对象。 例如，如果服务或系统在全局命名空间中创建文件映射对象，则任何会话中运行的任何进程都可以访问该文件映射对象，前提是调用方具有所需的访问权限。  **Windowsxp：**上一段中所述的要求是在 Windows Server 2003 和 Windows XP 和 SP2 中引入的  使用结构化异常处理来保护写入文件视图或从文件视图读取的任何代码。 有关详细信息，请参阅 [从文件视图读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  若要具有具有可执行权限的映射，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 **CreateFileMapping**，然后使用 或 FILE\_MAP\_EXECUTE | FILE\_MAP\_READ调用 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)FILE\_MAP\_EXECUTE | FILE\_MAP\_WRITE。  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 |    示例 有关示例，请参阅 [创建命名共享内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-named-shared-memory) 或使用 [大页创建文件映射](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-mapping-using-large-pages)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)  [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa)  [创建文件映射对象](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-mapping-object)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  文件映射函数  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## CreateFileMappingNumaA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为指定文件创建或打开命名或未命名的文件映射对象，并为物理内存指定 NUMA 节点。 语法 C++复制  HANDLE CreateFileMappingNumaA(  [in] HANDLE hFile,  [in, optional] LPSECURITY\_ATTRIBUTES lpFileMappingAttributes,  [in] DWORD flProtect,  [in] DWORD dwMaximumSizeHigh,  [in] DWORD dwMaximumSizeLow,  [in, optional] LPCSTR lpName,  [in] DWORD nndPreferred  ); 参数 [in] hFile  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 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 *hFile***INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMappingNuma** 创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  [in, optional] lpFileMappingAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定返回的句柄是否可以由子进程继承。 的 **lpSecurityDescriptor** 成员 **SECURITY\_ATTRIBUTES** 结构指定新文件映射对象的安全描述符。  如果 *lpFileMappingAttributes* 为 **NULL**，则不能继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制在文件映射对象来自创建者的主令牌或模拟令牌的默认安全描述符中列出 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] flProtect  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_EXECUTE\_READ**  0x20 | 允许映射视图，以便进行只读、写入时复制或执行访问。  *必须使用 GENERIC\_READ 创建 hFile* 参数指定的文件句柄**，**GENERIC\_EXECUTE访问权限。 | | **PAGE\_EXECUTE\_READWRITE**  0x40 | 允许映射视图，以便进行只读、写入时复制、读/写或执行访问。  必须使用**GENERIC\_READ**、**GENERIC\_WRITE**和**GENERIC\_EXECUTE**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_EXECUTE\_WRITECOPY**  0x80 | 允许映射视图，以便进行只读、写入时复制或执行访问。 此值等效于 **PAGE\_EXECUTE\_READ**。  *必须使用 GENERIC\_READ 创建 hFile* 参数指定的文件句柄**，**GENERIC\_EXECUTE访问权限。  **Windows Vista：**此值在 Windows Vista SP1 之前不可用。 | | **PAGE\_READONLY**  0x02 | 允许映射视图以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄**，GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |     应用程序可以通过将以下属性与前面的页面保护值之一组合，为文件映射对象指定以下一个或多个属性。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 为所有页面分配内存中的物理存储或分页文件。  这是默认设置。 | | **SEC\_IMAGE**  0x1000000 | 将指定为可执行映像文件的文件。  **SEC\_IMAGE** 属性必须与页面保护值（如 **PAGE\_READONLY**）结合使用。 但是，此页面保护值对可执行文件的视图没有影响。 可执行文件的视图的页面保护由可执行文件本身决定。  SEC\_IMAGE中没有其他属性**有效。** | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。**  **Windows Server 2008 R2、Windows 7、Windows Server 2008 和 Windows Vista：**在 Windows Server 2012 和 Windows 8 之前不支持此值。 | | **SEC\_LARGE\_PAGES**  0x80000000 | 允许在映射图像或从页面文件进行支持时使用大页面，但在映射常规文件的数据时不使用。 请务必将文件映射对象的最大大小指定为 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面的最小大小，并启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可访问。  除非设备明确需要，否则应用程序不应使用此标志。 将互锁函数与 **SEC\_NOCACHE** 映射的内存配合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 。 | | **SEC\_RESERVE**  0x4000000 | 保留所有页而不分配物理存储。  在释放页范围之前，任何其他分配操作都无法使用保留的页范围。  可以在后续调用 [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma) 函数时标识保留页。 仅当 *hFile* 参数 **INVALID\_HANDLE\_VALUE** (即) 系统分页文件支持的文件映射对象时，此属性才有效。 | | **SEC\_WRITECOMBINE**  0x40000000 | 将所有页面设置为写合并。  除非设备明确需要，否则应用程序不应使用此属性。 将互锁函数 **与SEC\_WRITECOMBINE映射** 的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 要求设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 |   [in] dwMaximumSizeHigh  文件映射对象最大大小的高序 **DWORD** 。  [in] dwMaximumSizeLow  文件映射对象最大大小的低序 **DWORD** 。  如果此参数和 *dwMaximumSizeHigh* 参数为 0 (零) ，则文件映射对象的最大大小等于 *hFile* 参数标识的文件的当前大小。  尝试映射长度为 0 (零的文件) 失败，错误代码 **为 ERROR\_FILE\_INVALID**。 应用程序应测试长度为 0 (零) 的文件，并拒绝这些文件。  [in, optional] lpName  文件映射对象的名称。  如果此参数与现有文件映射对象的名称匹配，则函数请求访问具有 *flProtect* 参数指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 参数与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况的原因是这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录用户使用会话 1 (一个) ，依此依此。 内核对象名称必须遵循准则，以便应用程序可以支持多个用户。  [in] nndPreferred  物理内存应驻留的 NUMA 节点。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **NUMA\_NO\_PREFERRED\_NODE**  0xffffffff | 不首选 NUMA 节点。 这与调用 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 函数相同。 |  返回值 如果函数成功，则返回值是文件映射对象的句柄。  如果对象在函数调用之前存在，则函数返回具有当前大小的现有对象的句柄 (，而不是指定大小) ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 若要获得更多的错误信息，请调用 GetLastError 函数。 注解 创建文件映射对象后，文件大小不得超过文件映射对象的大小;如果存在，并非所有文件内容都可用于共享。  文件映射对象可以按重复、继承或按名称共享。 由页面文件支持的文件映射对象中页面的初始内容为 0 (零) 。  如果应用程序为文件映射对象指定的大小大于磁盘上实际命名文件的大小，并且页面保护允许写入访问 (即 *flProtect* 参数指定 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**) ，则磁盘上的文件将增加以匹配文件映射对象的指定大小。 如果扩展了文件，则文件旧端与新文件端之间的文件内容不保证为零;行为由文件系统定义。  如果无法增加文件，则结果是创建文件映射对象失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_DISK\_FULL**。  **CreateFileMappingNuma** 函数返回的句柄对新的文件映射对象具有完全访问权限，并且可以与需要文件映射对象的句柄的任何函数一起使用。 可以通过进程创建、处理重复或按名称共享文件映射对象。 有关详细信息，请参阅 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数。  创建文件映射对象会创建映射文件视图的可能性，但不映射视图。 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma) 函数将文件视图映射到进程地址空间。  有一个重要例外，派生自单个文件映射对象的文件视图在特定时间是一致的或相同的。 如果多个进程具有同一个文件映射对象的句柄，则它们会在映射文件的视图时看到一致的数据视图。  异常与远程文件相关。 尽管 **CreateFileMappingNuma** 函数适用于远程文件，但它不能使它们保持一致。 例如，如果两台计算机将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，页面不会合并。  映射文件和使用输入和输出访问的文件 (I/O) 函数 ([ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 和 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)) 不一定一致。  若要完全关闭文件映射对象，应用程序必须通过调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 函数取消映射文件映射对象的所有映射视图，然后通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数关闭文件映射对象句柄。  可以按任何顺序调用这些函数。 必须调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 函数，因为文件映射对象的映射视图保持对象的内部打开句柄，并且文件映射对象在关闭所有打开的句柄之前不会关闭。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  从会话 0 以外的会话创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 请注意，此权限检查仅限于创建文件映射对象，不适用于打开现有对象。 例如，如果服务或系统创建文件映射对象，则任何会话中运行的任何进程都可以访问该文件映射对象，前提是调用方具有所需的访问权限。  使用结构化异常处理来保护写入内存映射视图或从内存映射视图读取的任何代码。 有关详细信息，请参阅 [从文件视图读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  若要具有具有可执行权限的映射，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 **CreateFileMappingNuma** 函数，然后使用 或 FILE\_MAP\_EXECUTE | FILE\_MAP\_READ调用 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma) 函数FILE\_MAP\_EXECUTE | FILE\_MAP\_WRITE。  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 |  要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h、Memoryapi.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)  [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma)  [NUMA 支持](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/numa-support)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## CreateFileMapping2 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为指定文件创建或打开命名或未命名的文件映射对象。 可以将物理内存的首选 NUMA 节点指定为扩展参数;请参阅 ExtendedParameters 参数。 语法 C++复制  HANDLE CreateFileMapping2(  HANDLE File,  SECURITY\_ATTRIBUTES \*SecurityAttributes,  ULONG DesiredAccess,  ULONG PageProtection,  ULONG AllocationAttributes,  ULONG64 MaximumSize,  PCWSTR Name,  MEM\_EXTENDED\_PARAMETER \*ExtendedParameters,  ULONG ParameterCount  ); parameters File  类型：\_In\_ [**HANDLE**](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types)  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 有关详细信息，请参阅 [文件安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/win32/FileIO/file-security-and-access-rights)。  如果 *hFile***是INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMapping** 会创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  SecurityAttributes  类型：\_In\_opt\_ [**SECURITY\_ATTRIBUTES**](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))**\***  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定是否可由子进程继承返回的句柄。 **SECURITY\_ATTRIBUTES** 结构的 **lpSecurityDescriptor** 成员为新的文件映射对象指定安全描述符。  如果 *lpAttributes* 为 **NULL**，则无法继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制列出了来自创建者的主要令牌或模拟令牌的文件映射对象的默认安全描述符中 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/win32/Memory/file-mapping-security-and-access-rights)。  DesiredAccess  类型：\_In\_ [**ULONG**](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types)  返回的文件映射句柄的所需访问掩码。 有关访问权限的列表，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/win32/memory/file-mapping-security-and-access-rights)。  PageProtection  类型：\_In\_ [**ULONG**](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types)  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_EXECUTE\_READ**  0x20 | 允许映射视图，以便进行只读、写入时复制或执行访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_READWRITE**  0x40 | 允许映射视图，以便进行只读、写入时复制、读/写或执行访问。  必须使用**GENERIC\_READ**、**GENERIC\_WRITE**和**GENERIC\_EXECUTE**访问权限创建 *hFile* 参数指定的文件句柄。  **Windows Server 2003 和 Windows XP：**此值在 Windows XP SP2 和 Windows Server 2003 SP1 之前不可用。 | | **PAGE\_EXECUTE\_WRITECOPY**  0x80 | 允许映射视图，以便进行只读、写入时复制或执行访问。 此值等效于 **PAGE\_EXECUTE\_READ**。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**并GENERIC\_EXECUTE**访问权限。  **Windows Vista：**此值在 Windows Vista SP1 中之前不可用。  **Windows Server 2003 和 Windows XP：**不支持此值。 | | **PAGE\_READONLY**  0x02 | 允许映射视图，以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用 **GENERIC\_READ** 创建 *hFile* 参数指定的文件句柄，**GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图，以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |   AllocationAttributes  类型：\_In\_ [**ULONG**](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types)  可以为文件映射对象指定以下一个或多个属性。 另请参阅 PageProtection 参数。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定在将文件的视图映射到进程地址空间时，将提交而不是保留整个页面范围。 系统必须有足够的可提交页面来保存整个映射。 否则， **CreateFileMapping** 将失败。  对于由可执行图像文件或数据文件支持的文件映射对象， (*hfile* 参数是文件) 句柄，则此属性不起作用。  **SEC\_COMMIT** 不能与 **SEC\_RESERVE**结合使用。  如果未指定任何属性，则假定 **SEC\_COMMIT** 。 | | **SEC\_IMAGE**  0x1000000 | 指定 *hFile* 参数指定的文件是可执行映像文件。  **SEC\_IMAGE** 属性必须与页面保护值（如**PAGE\_READONLY**）结合使用。 但是，此页面保护值对可执行映像文件的视图没有影响。 对可执行映像文件视图的页面保护由可执行文件本身确定。  没有其他属性对 **SEC\_IMAGE**有效。 | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。**  **Windows Server 2008 R2、Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP：**在 Windows Server 2012 和 Windows 8 之前不支持此值。 | | **SEC\_LARGE\_PAGES**  0x80000000 | 使大型页面可用于操作系统分页文件支持的文件映射对象， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) 。 由可执行图像文件或数据文件支持的文件映射对象不支持此属性， (*hFile* 参数是可执行映像或数据文件的句柄) 。  文件映射对象的最大大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数返回的大页最小大小的倍数。 否则， **CreateFileMapping** 将失败。 映射使用 **SEC\_LARGE\_PAGES** 创建的文件映射对象的视图时，基址和视图大小也必须是最小大页面大小的倍数。  **SEC\_LARGE\_PAGES** 要求在调用方令牌中启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/win32/SecAuthZ/authorization-constants) 特权。  如果指定 **了SEC\_LARGE\_PAGES** ，还必须指定 **SEC\_COMMIT** 。  **Windows Server 2003：**直到 Windows Server 2003 SP1 才支持此值。  **Windowsxp：**不支持此值。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可缓存。  应用程序不应使用此属性，除非设备显式需要。 将互锁 **函数与SEC\_NOCACHE** 映射的内存配合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 | | **SEC\_RESERVE**  0x4000000 | 如果文件映射对象由操作系统分页文件支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定当文件视图映射到进程地址空间时，将保留整个页面范围供进程以后使用，而不是提交。  可以在后续调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc) 函数时提交保留页。 提交页面后，无法使用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualfree) 函数释放或取消提交页面。  对于由可执行图像文件或数据文件支持的文件映射对象， (*hfile* 参数是文件) 句柄，则此属性不起作用。  **SEC\_RESERVE** 不能与 **SEC\_COMMIT** 结合使用。 | | **SEC\_WRITECOMBINE**  0x40000000 | 设置要写入组合的所有页面。  应用程序不应使用此属性，除非设备显式需要。 将互锁 **函数与SEC\_WRITECOMBINE** 映射的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。  **Windows Server 2003 和 Windows XP：**此标志在 Windows Vista 之前不受支持。 |   MaximumSize  类型：\_In\_ [**ULONG64**](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types)  文件映射对象的最大大小。  如果此参数为 0 (零) ，则文件映射对象的最大大小等于 *hFile* 标识的文件的当前大小。  尝试映射长度为 0 (零的文件) 失败，错误代码为 **ERROR\_FILE\_INVALID**。 应测试长度为 0 (零) 的文件，并拒绝这些文件。  Name  类型：\_In\_opt\_ [**PCWSTR**](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types)  文件映射对象的名称。  如果此参数与现有映射对象的名称匹配，则该函数将请求访问具有 *flProtect* 指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况的原因是这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/win32/SecAuthZ/authorization-constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/win32/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录用户使用会话 1 (一个) ，依此依此。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。  ExtendedParameters  类型：\_Inout\_updates\_opt\_ (ParameterCount) [**MEM\_EXTENDED\_PARAMETER**](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter)**\***  指向 [MEM\_EXTENDED\_PARAMETER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter)类型的一个或多个扩展参数的可选指针。 每个扩展参数值本身都可以具有 **MemExtendedParameterAddressRequirements** 或 **MemExtendedParameterNumaNode***的 Type* 字段。 如果未提供 **MemExtendedParameterNumaNode** 扩展参数，则行为与 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc)/[MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile) 函数 (相同，也就是说，物理页的首选 NUMA 节点是根据首先访问内存) 线程的理想处理器确定的。  ParameterCount  In ULONG 参数计数  ExtendedParameters 指向的扩展参数数。 返回值 如果函数成功，则返回值是新创建的文件映射对象的句柄。  如果对象在函数调用之前存在，则函数将 (当前大小（而不是指定大小) ）返回现有对象的句柄， [GetLastError](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-createfilemappingw#remarks) 的**备注**。 示例 有关示例，请参阅 [创建命名共享内存](https://learn.microsoft.com/zh-cn/windows/win32/memory/creating-named-shared-memory)或使用 [大页创建文件映射](https://learn.microsoft.com/zh-cn/windows/win32/Memory/creating-a-file-mapping-using-large-pages)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/win32/api/handleapi/nf-handleapi-closehandle)  [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-createfilemappingnumaa)  [创建文件映射对象](https://learn.microsoft.com/zh-cn/windows/win32/Memory/creating-a-file-mapping-object)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/win32/api/handleapi/nf-handleapi-duplicatehandle)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理功能](https://learn.microsoft.com/zh-cn/windows/win32/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/win32/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-readfile)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/win32/api/fileapi/nf-fileapi-writefile) |

## CreateFileMappingFromApp 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为 Windows 应用商店应用中的指定文件创建或打开命名或未命名的文件映射对象。 语法 C++复制  HANDLE CreateFileMappingFromApp(  [in] HANDLE hFile,  [in, optional] PSECURITY\_ATTRIBUTES SecurityAttributes,  [in] ULONG PageProtection,  [in] ULONG64 MaximumSize,  [in, optional] PCWSTR Name  ); parameters [in] hFile  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 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 *hFile***INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMappingFromApp** 创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  [in, optional] SecurityAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定返回的句柄是否可以由子进程继承。 **SECURITY\_ATTRIBUTES** 结构的 **lpSecurityDescriptor** 成员为新的文件映射对象指定安全描述符。  如果 *SecurityAttributes* 为 **NULL**，则无法继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制在文件映射对象来自创建者的主令牌或模拟令牌的默认安全描述符中列出 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] PageProtection  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_READONLY**  0x02 | 允许映射视图以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄**，GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |     应用程序可以通过将以下属性与前面的页面保护值之一组合，为文件映射对象指定以下一个或多个属性。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定当文件的视图映射到进程地址空间时，将提交而不是保留整个页面范围。 系统必须有足够的可提交页来保存整个映射。 否则， **CreateFileMappingFromApp** 将失败。  此属性对由可执行映像文件或数据文件支持的文件映射对象无效， (*hfile* 参数是文件) 的句柄。  **SEC\_COMMIT** 不能与 **SEC\_RESERVE** 结合使用。  如果未指定任何属性，则假定 **SEC\_COMMIT** 。 | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。** | | **SEC\_LARGE\_PAGES**  0x80000000 | 允许将大型页面用于操作系统分页文件支持的文件映射对象， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) 。 由可执行映像文件或数据文件支持的文件映射对象不支持此属性， (*hFile* 参数是可执行映像或数据文件的句柄) 。  文件映射对象的最大大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数返回的大页面最小大小的倍数。 否则， **CreateFileMappingFromApp** 将失败。 映射使用 **SEC\_LARGE\_PAGES** 创建的文件映射对象的视图时，基址和视图大小也必须是最小大页面大小的倍数。  **SEC\_LARGE\_PAGES** 需要在调用方令牌中启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 权限。  如果指定 **了SEC\_LARGE\_PAGES** ，还必须指定 **SEC\_COMMIT** 。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可缓存。  除非设备明确需要，否则应用程序不应使用此属性。 将互锁函数 **与SEC\_NOCACHE映射** 的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 要求设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 | | **SEC\_RESERVE**  0x4000000 | 如果文件映射对象由操作系统分页文件提供支持， (*hfile* 参数 **INVALID\_HANDLE\_VALUE**) ，则指定当文件的视图映射到进程地址空间时，将保留整个页面范围供进程以后使用，而不是提交。  可以在后续调用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数时提交保留页。 提交页面后，无法使用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 函数释放或取消提交页面。  此属性对由可执行映像文件或数据文件支持的文件映射对象无效， (*hfile* 参数是文件) 的句柄。  **SEC\_RESERVE** 不能与 **SEC\_COMMIT** 结合使用。 | | **SEC\_WRITECOMBINE**  0x40000000 | 将所有页面设置为写合并。  除非设备明确需要，否则应用程序不应使用此属性。 将互锁函数 **与SEC\_WRITECOMBINE映射** 的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 要求设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 |   [in] MaximumSize  文件映射对象的最大大小。  尝试映射长度为 0 (零的文件) 失败，错误代码 **为 ERROR\_FILE\_INVALID**。 应用程序应测试长度为 0 (零的文件) 并拒绝这些文件。  [in, optional] Name  文件映射对象的名称。  如果此参数与现有映射对象的名称匹配，则函数会请求访问具有 *flProtect* 指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况是因为这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 从会话 0 以外的会话在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录的用户使用会话 1 (一个) ，依此而过。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。 返回值 如果函数成功，则返回值是新创建的文件映射对象的句柄。  如果对象在函数调用之前存在，则函数将返回一个句柄，该句柄指向现有对象 (其当前大小，而不是指定大小) ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 创建文件映射对象后，文件大小不得超过文件映射对象的大小;如果存在，并非所有文件内容都可用于共享。  如果应用程序为文件映射对象指定的大小大于磁盘上实际命名文件的大小，并且页面保护允许写入访问 (即 *flProtect* 参数指定 **PAGE\_READWRITE**) ，则磁盘上的文件将增加以匹配文件映射对象的指定大小。 如果文件已扩展，则不保证文件旧端与文件新端之间的文件内容为零;行为由文件系统定义。 如果无法增加磁盘上的文件， **CreateFileMappingFromApp** 将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回 **ERROR\_DISK\_FULL**。  操作系统分页文件支持的文件映射对象中页面的初始内容为 0 (零) 。  **CreateFileMappingFromApp** 返回的句柄对新的文件映射对象具有完全访问权限，并且可以与需要文件映射对象句柄的任何函数一起使用。  多个进程可以通过使用单个共享文件映射对象或创建由同一文件支持的单独文件映射对象来共享同一文件的视图。 通过继承进程创建时的句柄、复制句柄或按名称打开文件映射对象，多个进程可以共享单个文件映射对象。 有关详细信息，请参阅 [CreateProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa)、 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数。  创建文件映射对象实际上不会将视图映射到进程地址空间。 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 函数将文件视图映射到进程地址空间。  有一个重要例外，从同一文件支持的任何文件映射对象派生的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **CreateFileMappingFromApp** 适用于远程文件，但它不会使它们保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只能看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  映射文件和使用输入和输出访问的文件 (I/O) 函数 ([ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 和 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)) 不一定是一致的。  文件映射对象的映射视图保留对该对象的内部引用，并且文件映射对象在释放对其的所有引用之前不会关闭。 因此，若要完全关闭文件映射对象，应用程序必须通过调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 取消映射文件映射对象的所有映射视图，并通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 关闭文件映射对象句柄。 可以按任意顺序调用这些函数。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  使用结构化异常处理来保护写入文件视图或从文件视图中读取的任何代码。 有关详细信息，请参阅 [从文件视图中读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  仅当应用具有 **codeGeneration** 功能时，才能成功请求可执行保护。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)  [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [创建文件映射对象](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-mapping-object)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  文件映射函数  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## CreateFileMappingNumaA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 为指定文件创建或打开命名或未命名的文件映射对象，并为物理内存指定 NUMA 节点。 语法 C++复制  HANDLE CreateFileMappingNumaA(  [in] HANDLE hFile,  [in, optional] LPSECURITY\_ATTRIBUTES lpFileMappingAttributes,  [in] DWORD flProtect,  [in] DWORD dwMaximumSizeHigh,  [in] DWORD dwMaximumSizeLow,  [in, optional] LPCSTR lpName,  [in] DWORD nndPreferred  ); 参数 [in] hFile  要从中创建文件映射对象的文件的句柄。  必须使用与 *flProtect* 参数指定的保护标志兼容的访问权限打开文件。 这不是必需的，但建议打开要映射的文件进行独占访问。 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 *hFile***INVALID\_HANDLE\_VALUE**，则调用进程还必须在 *dwMaximumSizeHigh* 和 *dwMaximumSizeLow* 参数中指定文件映射对象的大小。 在此方案中， **CreateFileMappingNuma** 创建一个指定大小的文件映射对象，该对象由系统分页文件而不是文件系统中的文件提供支持。  [in, optional] lpFileMappingAttributes  指向 [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85)) 结构的指针，该结构确定返回的句柄是否可以由子进程继承。 的 **lpSecurityDescriptor** 成员 **SECURITY\_ATTRIBUTES** 结构指定新文件映射对象的安全描述符。  如果 *lpFileMappingAttributes* 为 **NULL**，则不能继承句柄，并且文件映射对象将获取默认的安全描述符。 访问控制在文件映射对象来自创建者的主令牌或模拟令牌的默认安全描述符中列出 (ACL) 。 有关详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] flProtect  指定文件映射对象的页保护。 对象的所有映射视图都必须与此保护兼容。  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PAGE\_EXECUTE\_READ**  0x20 | 允许映射视图，以便进行只读、写入时复制或执行访问。  *必须使用 GENERIC\_READ 创建 hFile* 参数指定的文件句柄**，**GENERIC\_EXECUTE访问权限。 | | **PAGE\_EXECUTE\_READWRITE**  0x40 | 允许映射视图，以便进行只读、写入时复制、读/写或执行访问。  必须使用**GENERIC\_READ**、**GENERIC\_WRITE**和**GENERIC\_EXECUTE**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_EXECUTE\_WRITECOPY**  0x80 | 允许映射视图，以便进行只读、写入时复制或执行访问。 此值等效于 **PAGE\_EXECUTE\_READ**。  *必须使用 GENERIC\_READ 创建 hFile* 参数指定的文件句柄**，**GENERIC\_EXECUTE访问权限。  **Windows Vista：**此值在 Windows Vista SP1 之前不可用。 | | **PAGE\_READONLY**  0x02 | 允许映射视图以便进行只读或写入时复制访问。 尝试写入特定区域会导致访问冲突。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 | | **PAGE\_READWRITE**  0x04 | 允许映射视图，以便进行只读、写入时复制或读/写访问。  必须使用**GENERIC\_READ**创建 *hFile* 参数指定的文件句柄**，GENERIC\_WRITE**访问权限。 | | **PAGE\_WRITECOPY**  0x08 | 允许映射视图以便进行只读或写入时复制访问。 此值等效于 **PAGE\_READONLY**。  必须使用**GENERIC\_READ**访问权限创建 *hFile* 参数指定的文件句柄。 |     应用程序可以通过将以下属性与前面的页面保护值之一组合，为文件映射对象指定以下一个或多个属性。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SEC\_COMMIT**  0x8000000 | 为所有页面分配内存中的物理存储或分页文件。  这是默认设置。 | | **SEC\_IMAGE**  0x1000000 | 将指定为可执行映像文件的文件。  **SEC\_IMAGE** 属性必须与页面保护值（如 **PAGE\_READONLY**）结合使用。 但是，此页面保护值对可执行文件的视图没有影响。 可执行文件的视图的页面保护由可执行文件本身决定。  SEC\_IMAGE中没有其他属性**有效。** | | **SEC\_IMAGE\_NO\_EXECUTE**  0x11000000 | 指定 *hFile* 参数指定的文件是不会执行的可执行映像文件，并且加载的映像文件不会运行强制完整性检查。 此外，映射使用 **SEC\_IMAGE\_NO\_EXECUTE** 属性创建的文件映射对象的视图不会调用使用 [PsSetLoadImageNotifyRoutine](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine) 内核 API 注册的驱动程序回调。  **SEC\_IMAGE\_NO\_EXECUTE** 属性必须与**PAGE\_READONLY**页保护值结合使用。 SEC\_IMAGE\_NO\_EXECUTE中没有其他属性**有效。**  **Windows Server 2008 R2、Windows 7、Windows Server 2008 和 Windows Vista：**在 Windows Server 2012 和 Windows 8 之前不支持此值。 | | **SEC\_LARGE\_PAGES**  0x80000000 | 允许在映射图像或从页面文件进行支持时使用大页面，但在映射常规文件的数据时不使用。 请务必将文件映射对象的最大大小指定为 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面的最小大小，并启用 [SeLockMemoryPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 | | **SEC\_NOCACHE**  0x10000000 | 将所有页面设置为不可访问。  除非设备明确需要，否则应用程序不应使用此标志。 将互锁函数与 **SEC\_NOCACHE** 映射的内存配合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_NOCACHE** 需要设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 。 | | **SEC\_RESERVE**  0x4000000 | 保留所有页而不分配物理存储。  在释放页范围之前，任何其他分配操作都无法使用保留的页范围。  可以在后续调用 [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma) 函数时标识保留页。 仅当 *hFile* 参数 **INVALID\_HANDLE\_VALUE** (即) 系统分页文件支持的文件映射对象时，此属性才有效。 | | **SEC\_WRITECOMBINE**  0x40000000 | 将所有页面设置为写合并。  除非设备明确需要，否则应用程序不应使用此属性。 将互锁函数 **与SEC\_WRITECOMBINE映射** 的内存结合使用可能会导致 **EXCEPTION\_ILLEGAL\_INSTRUCTION** 异常。  **SEC\_WRITECOMBINE** 要求设置 **SEC\_RESERVE** 或 **SEC\_COMMIT** 属性。 |   [in] dwMaximumSizeHigh  文件映射对象最大大小的高序 **DWORD** 。  [in] dwMaximumSizeLow  文件映射对象最大大小的低序 **DWORD** 。  如果此参数和 *dwMaximumSizeHigh* 参数为 0 (零) ，则文件映射对象的最大大小等于 *hFile* 参数标识的文件的当前大小。  尝试映射长度为 0 (零的文件) 失败，错误代码 **为 ERROR\_FILE\_INVALID**。 应用程序应测试长度为 0 (零) 的文件，并拒绝这些文件。  [in, optional] lpName  文件映射对象的名称。  如果此参数与现有文件映射对象的名称匹配，则函数请求访问具有 *flProtect* 参数指定的保护的对象。  如果此参数为 **NULL**，则创建不带名称的文件映射对象。  如果 *lpName* 参数与现有事件、信号灯、互斥体、可等待计时器或作业对象的名称匹配，则函数将失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数返回 **ERROR\_INVALID\_HANDLE**。 发生这种情况的原因是这些对象共享相同的命名空间。  名称可以具有“Global”或“Local”前缀，以在全局命名空间或会话命名空间中显式创建对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 在全局命名空间中创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。  使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0 (零) ，下一个登录用户使用会话 1 (一个) ，依此依此。 内核对象名称必须遵循准则，以便应用程序可以支持多个用户。  [in] nndPreferred  物理内存应驻留的 NUMA 节点。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **NUMA\_NO\_PREFERRED\_NODE**  0xffffffff | 不首选 NUMA 节点。 这与调用 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 函数相同。 |  返回值 如果函数成功，则返回值是文件映射对象的句柄。  如果对象在函数调用之前存在，则函数返回具有当前大小的现有对象的句柄 (，而不是指定大小) ， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数返回 **ERROR\_ALREADY\_EXISTS**。  如果函数失败，则返回值为 NULL。 若要获得更多的错误信息，请调用 GetLastError 函数。 注解 创建文件映射对象后，文件大小不得超过文件映射对象的大小;如果存在，并非所有文件内容都可用于共享。  文件映射对象可以按重复、继承或按名称共享。 由页面文件支持的文件映射对象中页面的初始内容为 0 (零) 。  如果应用程序为文件映射对象指定的大小大于磁盘上实际命名文件的大小，并且页面保护允许写入访问 (即 *flProtect* 参数指定 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE**) ，则磁盘上的文件将增加以匹配文件映射对象的指定大小。 如果扩展了文件，则文件旧端与新文件端之间的文件内容不保证为零;行为由文件系统定义。  如果无法增加文件，则结果是创建文件映射对象失败， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数将返回 **ERROR\_DISK\_FULL**。  **CreateFileMappingNuma** 函数返回的句柄对新的文件映射对象具有完全访问权限，并且可以与需要文件映射对象的句柄的任何函数一起使用。 可以通过进程创建、处理重复或按名称共享文件映射对象。 有关详细信息，请参阅 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数。  创建文件映射对象会创建映射文件视图的可能性，但不映射视图。 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma) 函数将文件视图映射到进程地址空间。  有一个重要例外，派生自单个文件映射对象的文件视图在特定时间是一致的或相同的。 如果多个进程具有同一个文件映射对象的句柄，则它们会在映射文件的视图时看到一致的数据视图。  异常与远程文件相关。 尽管 **CreateFileMappingNuma** 函数适用于远程文件，但它不能使它们保持一致。 例如，如果两台计算机将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，页面不会合并。  映射文件和使用输入和输出访问的文件 (I/O) 函数 ([ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 和 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile)) 不一定一致。  若要完全关闭文件映射对象，应用程序必须通过调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 函数取消映射文件映射对象的所有映射视图，然后通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数关闭文件映射对象句柄。  可以按任何顺序调用这些函数。 必须调用 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 函数，因为文件映射对象的映射视图保持对象的内部打开句柄，并且文件映射对象在关闭所有打开的句柄之前不会关闭。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  从会话 0 以外的会话创建文件映射对象需要 [SeCreateGlobalPrivilege](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-constants) 特权。 请注意，此权限检查仅限于创建文件映射对象，不适用于打开现有对象。 例如，如果服务或系统创建文件映射对象，则任何会话中运行的任何进程都可以访问该文件映射对象，前提是调用方具有所需的访问权限。  使用结构化异常处理来保护写入内存映射视图或从内存映射视图读取的任何代码。 有关详细信息，请参阅 [从文件视图读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  若要具有具有可执行权限的映射，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 **CreateFileMappingNuma** 函数，然后使用 或 FILE\_MAP\_EXECUTE | FILE\_MAP\_READ调用 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma) 函数FILE\_MAP\_EXECUTE | FILE\_MAP\_WRITE。  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 |  要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h、Memoryapi.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)  [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma)  [NUMA 支持](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/numa-support)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SECURITY\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa379560(v=vs.85))  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## FlushViewOfFile 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将文件映射视图中的字节范围写入磁盘。 语法 C++复制  BOOL FlushViewOfFile(  [in] LPCVOID lpBaseAddress,  [in] SIZE\_T dwNumberOfBytesToFlush  ); 参数 [in] lpBaseAddress  指向要刷新到映射文件的磁盘表示形式的字节范围的基址的指针。  [in] dwNumberOfBytesToFlush  要刷新的字节数。 如果 *dwNumberOfBytesToFlush* 为零，则将文件从基址刷新到映射的末尾。 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 刷新映射视图的范围将启动该范围内的脏页写入磁盘。 脏页是自映射文件视图以来内容发生更改的页面。 **FlushViewOfFile** 函数不会刷新文件元数据，并且不会等待返回，直到从基础硬件磁盘缓存刷新更改并物理写入磁盘。 若要刷新所有脏页以及文件的元数据并确保它们以物理方式写入磁盘，请调用 **FlushViewOfFile**，然后调用 [FlushFileBuffers](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-flushfilebuffers) 函数。  通过网络刷新内存映射文件时， **FlushViewOfFile** 保证数据已从本地计算机写入，但不保证数据驻留在远程计算机上。 服务器可以在远程端缓存数据。 因此， **FlushViewOfFile** 可以在数据以物理方式写入磁盘之前返回。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  在 Windows Server 2012 中，以下技术支持此函数。  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 |     暂停 CsvFs 时，此调用可能会失败，并显示一个错误，指示存在锁冲突。 示例 有关示例，请参阅 [从文件视图中读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-createfilea)  [创建文件视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-view)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) |

## GetMappedFileNameA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 检查指定的地址是否位于指定进程的地址空间中的内存映射文件中。 如果是这样，函数将返回内存映射文件的名称。 语法 C++复制  DWORD GetMappedFileNameA(  [in] HANDLE hProcess,  [in] LPVOID lpv,  [out] LPSTR lpFilename,  [in] DWORD nSize  ); 参数 [in] hProcess  进程的句柄。 句柄必须具有 **PROCESS\_QUERY\_INFORMATION** 访问权限。 有关详细信息，请参阅 [进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  [in] lpv  要验证的地址。  [out] lpFilename  指向缓冲区的指针，该缓冲区接收 *由 lpv* 指定的地址所属的内存映射文件的名称。  [in] nSize  *lpFilename* 缓冲区的大小（以字符为单位）。 返回值 如果函数成功，则返回值指定复制到缓冲区的字符串的长度（以字符为单位）。  如果函数失败，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 从 Windows 7 和 Windows Server 2008 R2 开始，Psapi.h 为 PSAPI 函数建立版本号。 PSAPI 版本号会影响用于调用函数的名称以及程序必须加载的库。  如果 **PSAPI\_VERSION** 为 2 或更大，则此函数在 Psapi.h 中定义为 **K32GetMappedFileName** ，并在 Kernel32.lib 和 Kernel32.dll 中导出。 如果 **PSAPI\_VERSION** 为 1，则此函数在 Psapi.h 中定义为 **GetMappedFileName** ，并在 Psapi.lib 中导出，Psapi.dll 为调用 **K32GetMappedFileName** 的包装器。  必须在早期版本的 Windows 以及 Windows 7 及更高版本上运行的程序应始终将此函数称为 **GetMappedFileName**。 为了确保正确解析符号，请将 Psapi.lib 添加到 **TARGETLIBS** 宏，并使用 **-DPSAPI\_VERSION=1** 编译程序。 若要使用运行时动态链接，请加载 Psapi.dll。  在 Windows Server 2012 中，以下技术支持此函数。  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 [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)。  **备注**  psapi.h 标头将 GetMappedFileName 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | psapi.h | | **Library** | Windows 7 和 Windows Server 2008 R2 上的 Kernel32.lib;如果 Windows 7 和 Windows Server 2008 R2 上的 PSAPI\_VERSION=1) ，则 Psapi.lib (;Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上的 Psapi.lib | | **DLL** | Windows 7 和 Windows Server 2008 R2 上的 Kernel32.dll;如果 Windows 7 和 Windows Server 2008 R2 上PSAPI\_VERSION=1) ，则为 Psapi.dll (;在 Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上 Psapi.dll |  另请参阅 [EnumProcesses](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-enumprocesses)  [内存映射文件信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/memory-mapped-file-information)  [PSAPI 函数](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/psapi-functions) |

## MapViewOfFile 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将文件映射的视图映射到调用进程的地址空间中。  若要为视图指定建议的基址，请使用 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 函数。 但是，不建议这样做。 语法 C++复制  LPVOID MapViewOfFile(  [in] HANDLE hFileMappingObject,  [in] DWORD dwDesiredAccess,  [in] DWORD dwFileOffsetHigh,  [in] DWORD dwFileOffsetLow,  [in] SIZE\_T dwNumberOfBytesToMap  ); parameters [in] hFileMappingObject  文件映射对象的句柄。 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数返回此句柄。  [in] dwDesiredAccess  文件映射对象的访问类型，该对象确定页面的页面保护。 此参数可以是以下值之一，也可以是多个值的按位 OR 组合（如果适用）。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_ALL\_ACCESS** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFile** 函数一起使用时， **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 | | **FILE\_MAP\_READ** | 映射文件的只读视图。 尝试写入文件视图会导致访问冲突。  创建文件映射对象时必须具有 **PAGE\_READONLY**、 **PAGE\_READWRITE**、 **PAGE\_EXECUTE\_READ**或 **PAGE\_EXECUTE\_READWRITE** 保护。 | | **FILE\_MAP\_WRITE** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFile** 一起使用时， (**FILE\_MAP\_WRITE** | **FILE\_MAP\_READ**) 和 **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 |     使用按位 OR，可以将上述值与这些值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_COPY** | 映射文件的写入时复制视图。 文件映射对象必须已**创建PAGE\_READONLY、PAGE\_READ\_EXECUTE**、**PAGE\_WRITECOPY**、**PAGE\_EXECUTE\_WRITECOPY**、**PAGE\_READWRITE**或**PAGE\_EXECUTE\_READWRITE**保护。  当进程写入写入时复制页时，系统会将原始页面复制到进程专用的新页。 新页面由分页文件提供支持。 对新页面的保护从写入时复制更改为读/写。  指定写入时复制访问权限时，系统将和进程提交费用用于整个视图，因为调用进程可能会写入视图中的每个页面，从而使所有页面都成为私有页面。 新页面的内容永远不会写回到原始文件，并且当视图未映射时会丢失。 | | **FILE\_MAP\_EXECUTE** | 映射文件的可执行视图 (映射的内存可以作为代码) 运行。 文件映射对象必须已创建 **PAGE\_EXECUTE\_READ**、 **PAGE\_EXECUTE\_WRITECOPY**或 **PAGE\_EXECUTE\_READWRITE** 保护。  **Windows Server 2003 和 Windows XP：**此值从具有 SP2 的 Windows XP 和 SP1 的 Windows Server 2003 开始可用。 | | **FILE\_MAP\_LARGE\_PAGES** | 从 Windows 10 版本 1703 开始，此标志指定应使用[大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且文件映射对象必须已使用 **SEC\_LARGE\_PAGES** 选项创建。 如果为 *lpBaseAddress* 提供非 null 值，则该值必须是 **GetLargePageMinimum** 的倍数。  **注意：**在 Windows 10 版本 1703 之前的 OS 版本上，**FILE\_MAP\_LARGE\_PAGES**标志不起作用。 在这些版本中，如果分区是使用 **SEC\_LARGE\_PAGES标志创建的** ，则使用大页面自动映射视图。 | | **FILE\_MAP\_TARGETS\_INVALID** | 将映射文件中的所有位置设置为控制流防护 (CFG) 的无效目标。 此标志类似于 **PAGE\_TARGETS\_INVALID**。 将此标志与执行访问权限 **FILE\_MAP\_EXECUTE**结合使用。 对这些页面中位置的任何间接调用都将失败 CFG 检查，并且进程将终止。 分配的可执行页面的默认行为是标记为 CFG 的有效调用目标。 |     对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *dwDesiredAccess* 参数不起作用，应设置为任何有效值，例如 **FILE\_MAP\_READ**。  有关访问文件映射对象的详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] dwFileOffsetHigh  视图开始位置的文件偏移量的高阶 **DWORD** 。  [in] dwFileOffsetLow  视图开始位置的文件偏移量的低序 **DWORD** 。 高偏移量和低偏移量的组合必须在文件映射中指定偏移量。 它们还必须匹配系统的内存分配粒度。 也就是说，偏移量必须是分配粒度的倍数。 若要获取系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数，该函数填充 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的成员。  [in] dwNumberOfBytesToMap  要映射到视图的文件映射的字节数。 所有字节都必须在 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 指定的最大大小内。 如果此参数为 0 (零) ，则映射将从指定的偏移量扩展到文件映射的末尾。 返回值 如果函数成功，则返回值是映射视图的起始地址。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 映射文件会使文件的指定部分在调用进程的地址空间中可见。  对于大于地址空间的文件，一次只能映射文件数据的一小部分。 第一个视图完成后，可以取消映射它并映射新视图。  若要获取视图的大小，请使用 [VirtualQuery](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualquery) 函数。  如果文件 (或文件映射对象的多个视图及其映射文件) 在指定时间包含相同的数据，则它们是 *一致的* 。 如果文件视图派生自由同一文件支持的任何文件映射对象，则会发生此情况。 进程可以使用 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 函数将文件映射对象句柄复制到另一个进程，或者另一个进程可以使用 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数按名称打开文件映射对象。  有一个重要例外，派生自由同一文件支持的任何文件映射对象的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **MapViewOfFile** 适用于远程文件，但无法使其保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  不保证文件的映射视图与 [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 或 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 函数访问的文件一致。  不要将指针存储在内存映射文件中;存储与文件映射基的偏移量，以便可以在任何地址使用映射。  若要防范 **EXCEPTION\_IN\_PAGE\_ERROR** 异常，请使用结构化异常处理来保护任何写入或读取页面文件以外的文件的内存映射视图的代码。 有关详细信息，请参阅 [从文件视图读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  如果文件映射对象由分页文件提供支持， ([调用 CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 并将 *hFile* 参数设置为 **INVALID\_HANDLE\_VALUE**) ，则分页文件必须足够大才能保存整个映射。 否则， **MapViewOfFile** 将失败。 由分页文件支持的文件映射对象中页面的初始内容为 0 (零) 。  创建由分页文件支持的文件映射对象时，调用方可以指定 **是 mapViewOfFile** 应同时保留和提交页面 (**SEC\_COMMIT**) ，还是仅保留页面 (**SEC\_RESERVE**) 。 映射文件会使整个映射的虚拟地址范围无法用于进程中的其他分配。 提交保留范围内的页面后，无法通过调用 [VirtualFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualfree) 来释放或取消提交该页。 取消映射视图且文件映射对象关闭时，将释放保留页和提交的页。 有关详细信息，请参阅 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 和 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 函数。  若要拥有具有可执行权限的文件，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)，然后使用 **FILE\_MAP\_EXECUTE FILE\_MAP\_WRITE** | 或 **FILE\_MAP\_EXECUTE** | **FILE\_MAP\_READ** 调用 **MapViewOfFile**。  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 |     暂停 CsvFs 时，此调用可能会失败，并显示一个错误，指示存在锁冲突。 示例 有关示例，请参阅 [创建命名共享内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-named-shared-memory)。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [创建文件视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-view)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) |

## MapViewOfFile2 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 将文件视图或页面文件支持的节映射到指定进程的地址空间中。 语法 C++复制  PVOID MapViewOfFile2(  [in] HANDLE FileMappingHandle,  [in] HANDLE ProcessHandle,  [in] ULONG64 Offset,  [in, optional] PVOID BaseAddress,  [in] SIZE\_T ViewSize,  [in] ULONG AllocationType,  [in] ULONG PageProtection  ); parameters [in] FileMappingHandle  要映射到指定进程的地址空间的节的 **HANDLE** 。  [in] ProcessHandle  节将映射到的进程句 **柄** 。 句柄必须具有 **PROCESS\_VM\_OPERATION** 访问掩码。  [in] Offset  部分开头的偏移量。 这必须是 64k 对齐的。  [in, optional] BaseAddress  视图的所需基址。 地址向下舍入到最近的 64k 边界。 如果此参数为 **NULL**，则系统选取基址。  [in] ViewSize  要映射的字节数。 值为 0 (0) 指定要映射整个部分。  [in] AllocationType  分配的类型。 此参数可以是零 (0) 或以下常量值之一：   * **MEM\_RESERVE** - 映射保留视图。 * **MEM\_LARGE\_PAGES** - 映射大型页面视图。 此标志指定应使用 [大页支持](https://learn.microsoft.com/zh-cn/windows/win32/memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且文件映射对象必须已使用 **SEC\_LARGE\_PAGES** 选项创建。 如果为 *BaseAddress* 参数提供非 null 值，则该值必须是 **GetLargePageMinimum** 的倍数。   [in] PageProtection  所需的页面保护。  对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *PageProtection* 参数不起作用，应设置为任何有效值，例如 **PAGE\_READONLY**。 返回值 如果成功，则返回映射视图的基址。 否则，返回 **NULL** ，并且可以使用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror) 扩展错误状态。 注解 此函数在标头中作为内联函数实现，不能在任何导出库或 DLL 中找到。 它与调用 [MapViewOfFileNuma2](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffilenuma2) （最后一个参数设置为 NUMA\_NO\_PREFERRED\_NODE）相同。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileNuma2](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffilenuma2) |

## MapViewOfFile3 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将文件或页文件支持的分区视图映射到指定进程的地址空间。 语法 C++复制  PVOID MapViewOfFile3(  [in] HANDLE FileMapping,  [in] HANDLE Process,  [in, optional] PVOID BaseAddress,  [in] ULONG64 Offset,  [in] SIZE\_T ViewSize,  [in] ULONG AllocationType,  [in] ULONG PageProtection,  [in, out, optional] MEM\_EXTENDED\_PARAMETER \*ExtendedParameters,  [in] ULONG ParameterCount  ); 参数 [in] FileMapping  要映射到指定进程的地址空间的节 **HANDLE**。  [in] Process  **HANDLE** 要在其中映射节的进程。  [in, optional] BaseAddress  视图的所需基址（地址向下舍入到最近的 64k 边界）。  如果此参数 **NULL**，则系统会选取基址。  如果 *BaseAddress* 未 **NULL**，则提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 都必须包含所有零。  [in] Offset  节开头的偏移量。  偏移量必须对齐 64k。  [in] ViewSize  要映射的字节数。 值为零（0）指定要映射整个节。  大小必须始终为页面大小的倍数。  [in] AllocationType  内存分配的类型。 此参数可以是零（0）或以下值之一。  展开表   |  |  | | --- | --- | | **价值** | **意义** | | **MEM\_RESERVE**  0x00002000 | 映射保留视图。 | | **MEM\_REPLACE\_PLACEHOLDER**  0x00004000 | 将占位符替换为映射视图。 仅支持数据/pf 支持的分区视图（无图像、物理内存等）。 替换占位符时，*BaseAddress* 和 *ViewSize* 必须与占位符的占位符完全匹配，并且提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零。  将占位符替换为映射视图后，若要将映射视图释放回占位符，请参阅 [UnmapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffileex) 的 *UnmapFlags* 参数，并 [UnmapViewOfFile2](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile2)。  占位符是保留内存区域的类型。  指定此标志时，*Offset* 和 *BaseAddress* 的 64k 对齐要求不适用。 | | **MEM\_LARGE\_PAGES**  0x20000000 | 映射大型页面视图。 此标志指定应使用 [大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且必须使用 **SEC\_LARGE\_PAGES** 选项创建文件映射对象。 如果为 *BaseAddress* 参数提供非 null 值，则该值必须是 **GetLargePageMinimum**的倍数。 |   [in] PageProtection  所需的页面保护。  对于使用 **SEC\_IMAGE** 属性创建的文件映射对象，*PageProtection* 参数无效，应设置为任何有效值，如 **PAGE\_READONLY**。  [in, out, optional] ExtendedParameters  指向 [MEM\_EXTENDED\_PARAMETER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter)类型的一个或多个扩展参数的可选指针。 每个扩展参数值本身都可以具有 *Type* 字段，**MemExtendedParameterAddressRequirements** 或 **MemExtendedParameterNumaNode**。 如果未提供 **MemExtendedParameterNumaNode** 扩展参数，则行为与 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)/[MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 函数（即，物理页的首选 NUMA 节点）是根据首次访问内存的线程的理想处理器确定的。  [in] ParameterCount  *ExtendedParameters*指向的扩展参数数。 返回值 如果成功，则返回映射视图的基址。 否则，使用 getLastError返回 NULL 和扩展错误状态 。 言论 此 API 有助于支持高性能游戏和服务器应用程序，这些应用程序在管理虚拟地址空间方面有特定要求。 例如，将内存映射在以前保留的区域之上;这可用于实现自动包装环缓冲区。 分配具有特定对齐方式的内存;例如，若要使应用程序能够按需提交大型/巨大的页面映射区域。  将此函数用于新分配，可以：   * 指定虚拟地址空间的范围和 2 的对齐限制 * 指定任意数量的扩展参数 * 将物理内存的首选 NUMA 节点指定为扩展参数 * 指定占位符操作（具体而言，替换）。   若要指定 NUMA 节点，请参阅 *ExtendedParameters* 参数。 例子 有关代码示例，请参阅 virtualAlloc2中的方案 1。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h （包括 Windows.h） | | **库** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [VirtualAlloc2](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc2)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFile2](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile2)  [MapViewOfFileNuma2](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffilenuma2) |

## MapViewOfFile3FromApp 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将文件映射的视图映射到调用的 Windows 应用商店应用的地址空间中。  使用此函数，可以：对于新分配，指定虚拟地址空间范围和 2 次幂对齐限制;指定任意数量的扩展参数;将物理内存的首选 NUMA 节点指定为扩展参数;并指定占位符操作，具体 (替换) 。  若要指定 NUMA 节点，请参阅 *ExtendedParameters* 参数。 语法 C++复制  PVOID MapViewOfFile3FromApp(  [in] HANDLE FileMapping,  [in] HANDLE Process,  [in, optional] PVOID BaseAddress,  [in] ULONG64 Offset,  [in] SIZE\_T ViewSize,  [in] ULONG AllocationType,  [in] ULONG PageProtection,  [in, out, optional] MEM\_EXTENDED\_PARAMETER \*ExtendedParameters,  [in] ULONG ParameterCount  ); 参数 [in] FileMapping  要映射到指定进程的地址空间的节的 **HANDLE** 。  [in] Process  节将映射到的进程句 **柄** 。  [in, optional] BaseAddress  视图的所需基址。 地址向下舍入到最近的 64k 边界。  复制  If this parameter is <b>NULL</b>, the system picks the base  address.  如果 *BaseAddress* 不是 **NULL**，则提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零。  [in] Offset  部分开头的偏移量。 这必须是 64k 对齐的。  [in] ViewSize  要映射的字节数。 值为 0 (0) 指定要映射整个部分。  大小必须始终是页面大小的倍数。  [in] AllocationType  内存分配的类型。 此参数可以是零 (0) 或以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_RESERVE**  0x00002000 | 映射保留视图。 | | **MEM\_REPLACE\_PLACEHOLDER**  0x00004000 | 将占位符替换为映射视图。 仅支持数据/pf 支持的分区视图， (无图像、物理内存等) 。 替换占位符时， *BaseAddress* 和 *ViewSize* 必须与占位符完全匹配，并且提供的任何 [MEM\_ADDRESS\_REQUIREMENTS](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_address_requirements) 结构都必须包含所有零。  将占位符替换为映射视图后，若要将该映射视图释放回占位符，请参阅 [UnmapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffileex) 和 [UnmapViewOfFile2](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile2) 的 *UnmapFlags* 参数。  占位符是保留内存区域的一种类型。 | | **MEM\_LARGE\_PAGES**  0x20000000 | 映射大型页面视图。 请参阅 [大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)。 |   [in] PageProtection  所需的页面保护。  对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *PageProtection* 参数不起作用，应设置为任何有效值，例如 **PAGE\_READONLY**。  [in, out, optional] ExtendedParameters  指向 [MEM\_EXTENDED\_PARAMETER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-mem_extended_parameter)类型的一个或多个扩展参数的可选指针。 每个扩展参数值本身都可以具有 **MemExtendedParameterAddressRequirements** 或 **MemExtendedParameterNumaNode***的 Type* 字段。 如果未提供 **MemExtendedParameterNumaNode** 扩展参数，则行为与 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)/[MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 函数 (相同，也就是说，物理页的首选 NUMA 节点是根据首先访问内存) 线程的理想处理器确定的。  [in] ParameterCount  *ExtendedParameters 指向的扩展参数数*。 返回值 如果成功，则返回映射视图的基址。 否则，返回 **NULL** ，并且可以使用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 扩展错误状态。 注解 此 API 有助于支持高性能游戏和服务器应用程序，这些应用程序在管理虚拟地址空间方面有特定要求。 例如，在以前保留的区域上映射内存;这对于实现自动环绕环缓冲区很有用。 并分配具有特定对齐方式的内存;例如，使应用程序能够按需提交大型/大型页面映射区域。  有一个重要例外，派生自由同一文件支持的任何文件映射对象的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **MapViewOfFile3FromApp** 适用于远程文件，但无法使其保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  仅当应用具有 **codeGeneration** 功能时，才能成功请求可执行保护。 示例 有关代码示例，请参阅 [Virtual2Alloc](https://msdn.microsoft.com/en-us/library/Mt832849(v=VS.85).aspx) 中的方案 1。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h) | | **Library** | WindowsApp.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [创建文件视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-view)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MapViewOfFile3](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile3)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) |

## MapViewOfFileEx 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将映射的文件视图映射到调用进程的地址空间中。 调用方可以选择为视图指定建议的基内存地址。  若要指定物理内存的 NUMA 节点，请参阅 [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma)。 语法 C++复制  LPVOID MapViewOfFileEx(  [in] HANDLE hFileMappingObject,  [in] DWORD dwDesiredAccess,  [in] DWORD dwFileOffsetHigh,  [in] DWORD dwFileOffsetLow,  [in] SIZE\_T dwNumberOfBytesToMap,  [in, optional] LPVOID lpBaseAddress  ); 参数 [in] hFileMappingObject  文件映射对象的句柄。 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数返回此句柄。  [in] dwDesiredAccess  对文件映射对象的访问类型，该对象确定页面的页面保护。 此参数可以是以下值之一，也可以是多个值的按位 OR 组合（如果适用）。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_ALL\_ACCESS** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFileEx** 函数一起使用时， **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 | | **FILE\_MAP\_READ** | 映射文件的只读视图。 尝试写入文件视图会导致访问冲突。  创建文件映射对象时必须具有 **PAGE\_READONLY**、 **PAGE\_READWRITE**、 **PAGE\_EXECUTE\_READ**或 **PAGE\_EXECUTE\_READWRITE** 保护。 | | **FILE\_MAP\_WRITE** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFileEx** 一起使用时， (**FILE\_MAP\_WRITE** | **FILE\_MAP\_READ**) 和 **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 |     使用按位 OR，可以将上述值与这些值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_COPY** | 映射文件的写入时复制视图。 文件映射对象必须已使用 **PAGE\_READONLY**、 **PAGE\_READ\_EXECUTE**、 **PAGE\_WRITECOPY**、 **PAGE\_EXECUTE\_WRITECOPY**、 **PAGE\_READWRITE**或 **PAGE\_EXECUTE\_READWRITE** 保护创建。  当进程写入写入时复制页时，系统会将原始页面复制到进程专用的新页。 新页面由分页文件提供支持。 对新页面的保护从写入时复制更改为读/写。  指定写入时复制访问权限时，系统将和进程提交费用用于整个视图，因为调用进程可能会写入视图中的每一页，从而使所有页面都成为私有页面。 新页面的内容永远不会写回到原始文件，并在取消映射视图时丢失。 | | **FILE\_MAP\_LARGE\_PAGES** | 从 Windows 10 版本 1703 开始，此标志指定应使用[大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且文件映射对象必须已使用 **SEC\_LARGE\_PAGES** 选项创建。 如果为 *lpBaseAddress* 提供非 null 值，则该值必须是 **GetLargePageMinimum** 的倍数。 | | **FILE\_MAP\_EXECUTE** | 映射文件的可执行视图 (映射的内存可以作为代码) 运行。 创建文件映射对象时必须具有 **PAGE\_EXECUTE\_READ**、 **PAGE\_EXECUTE\_WRITECOPY**或 **PAGE\_EXECUTE\_READWRITE** 保护。  **Windows Server 2003 和 Windows XP：**此值从 Windows XP SP2 和 Windows Server 2003 SP1 开始可用。 | | **FILE\_MAP\_TARGETS\_INVALID** | 将映射文件中的所有位置设置为控制流防护 (CFG) 的无效目标。 此标志类似于 **PAGE\_TARGETS\_INVALID**。 将此标志与执行访问权限 **FILE\_MAP\_EXECUTE**结合使用。 对这些页面中的位置的任何间接调用都将失败 CFG 检查，并且进程将终止。 分配的可执行页面的默认行为是标记为 CFG 的有效调用目标。 |     对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *dwDesiredAccess* 参数不起作用，应设置为任何有效值，例如 **FILE\_MAP\_READ**。  有关访问文件映射对象的详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] dwFileOffsetHigh  视图开始位置的文件偏移量的高阶 **DWORD** 。  [in] dwFileOffsetLow  视图开始位置的文件偏移量的低序 **DWORD** 。 高偏移量和低偏移量的组合必须在文件映射中指定偏移量。 它们还必须与系统的内存分配粒度匹配。 也就是说，偏移量必须是分配粒度的倍数。 若要获取系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数，该函数填充 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的成员。  [in] dwNumberOfBytesToMap  要映射到视图的文件映射的字节数。 所有字节都必须在 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 指定的最大大小内。 如果此参数为 0 (零) ，则映射将从指定的偏移量扩展到文件映射的末尾。  [in, optional] lpBaseAddress  指向调用进程地址空间中映射开始位置的内存地址的指针。 这必须是系统内存分配粒度的倍数，否则函数将失败。 若要确定系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。 如果指定地址上没有足够的地址空间，则函数将失败。  如果 *lpBaseAddress* 为 **NULL**，则操作系统会选择映射地址。 在此方案中， 函数等效于 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 函数。  虽然可以指定现在 (操作系统) 不使用的安全地址，但不能保证该地址随时间推移保持安全。 因此，最好让操作系统选择地址。 在这种情况下，不会将指针存储在内存映射文件中，而应存储与文件映射基的偏移量，以便可以在任何地址使用映射。 返回值 如果函数成功，则返回值为映射视图的起始地址。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 映射文件会使文件的指定部分在调用进程的地址空间中可见。  对于大于地址空间的文件，一次只能映射一小部分文件数据。 第一个视图完成后，取消映射该视图并映射新视图。  若要获取视图的大小，请使用 [VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualqueryex) 函数。  页面文件支持的文件映射对象中页面的初始内容为 0 (零) 。  通常，建议的地址用于指定文件应映射到多个进程中的同一地址。 这要求地址空间区域在所有涉及的进程中都可用。 不能在用于映射的区域进行其他内存分配，包括使用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 或 [VirtualAllocEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex) 函数来保留内存。  如果 *lpBaseAddress* 参数指定基偏移量，则当调用进程尚未使用指定的内存区域时，函数将成功。 系统不确保同一内存区域可用于其他 32 位进程中的内存映射文件。  文件 (或文件映射对象的多个视图及其映射文件) 如果它们在指定时间包含相同的数据，则它们是 *一致的* 。 如果文件视图派生自同一个文件映射对象，则会发生此情况。 进程可以使用 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 函数将文件映射对象句柄复制到另一个进程中，或者另一个进程可以使用 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数按名称打开文件映射对象。  有一个重要例外，从同一文件支持的任何文件映射对象派生的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **MapViewOfFileEx** 适用于远程文件，但它不会使它们保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只能看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  不保证文件的映射视图与 [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 或 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 函数访问的文件一致。  若要防止 **EXCEPTION\_IN\_PAGE\_ERROR** 异常，请使用结构化异常处理来保护写入或读取除页面文件以外的文件的内存映射视图的任何代码。 有关详细信息，请参阅 [从文件视图中读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  若要拥有具有可执行权限的文件，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)，然后使用 **FILE\_MAP\_EXECUTE FILE\_MAP\_WRITE** | 或 **FILE\_MAP\_EXECUTE** | **FILE\_MAP\_READ** 调用 **MapViewOfFileEx**。  在 Windows Server 2012 中，以下技术支持此函数。  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 |  要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [创建文件视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-view)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MapViewOfFileExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-mapviewoffileexnuma)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## MapViewOfFileExNuma 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将映射的文件视图映射到调用进程的地址空间中，并指定物理内存的 NUMA 节点。 语法 C++复制  LPVOID MapViewOfFileExNuma(  [in] HANDLE hFileMappingObject,  [in] DWORD dwDesiredAccess,  [in] DWORD dwFileOffsetHigh,  [in] DWORD dwFileOffsetLow,  [in] SIZE\_T dwNumberOfBytesToMap,  [in, optional] LPVOID lpBaseAddress,  [in] DWORD nndPreferred  ); 参数 [in] hFileMappingObject  文件映射对象的句柄。 [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa) 和 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数返回此句柄。  [in] dwDesiredAccess  对文件映射对象的访问类型，该对象确定页面的页面保护。 此参数可以是以下值之一，也可以是多个值的按位 OR 组合（如果适用）。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_ALL\_ACCESS** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFileExNuma 一**起使用时， **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 | | **FILE\_MAP\_READ** | 映射文件的只读视图。 尝试写入文件视图会导致访问冲突。  创建文件映射对象时必须具有 **PAGE\_READONLY**、 **PAGE\_READWRITE**、 **PAGE\_EXECUTE\_READ**或 **PAGE\_EXECUTE\_READWRITE** 保护。 | | **FILE\_MAP\_WRITE** | 映射文件的读/写视图。 文件映射对象必须已创建 **PAGE\_READWRITE** 或 **PAGE\_EXECUTE\_READWRITE** 保护。  与 **MapViewOfFileExNuma 一**起使用时， (FILE\_MAP\_WRITE | FILE\_MAP\_READ) 等效于 **FILE\_MAP\_WRITE**。 |     使用按位 OR，可以将上述值与这些值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_COPY** | 映射文件的写入时复制视图。 文件映射对象必须已使用 **PAGE\_READONLY**、 **PAGE\_READ\_EXECUTE**、 **PAGE\_WRITECOPY**、 **PAGE\_EXECUTE\_WRITECOPY**、 **PAGE\_READWRITE**或 **PAGE\_EXECUTE\_READWRITE** 保护创建。  当进程写入写入时复制页时，系统会将原始页面复制到进程专用的新页。 新页面由分页文件提供支持。 对新页面的保护从写入时复制更改为读/写。  指定写入时复制访问权限时，系统将和进程提交费用用于整个视图，因为调用进程可能会写入视图中的每一页，从而使所有页面都成为私有页面。 新页面的内容永远不会写回到原始文件，并在取消映射视图时丢失。 | | **FILE\_MAP\_EXECUTE** | 映射文件的可执行视图 (映射的内存可以作为代码) 运行。 创建文件映射对象时必须具有 **PAGE\_EXECUTE\_READ**、 **PAGE\_EXECUTE\_WRITECOPY**或 **PAGE\_EXECUTE\_READWRITE** 保护。 | | **FILE\_MAP\_LARGE\_PAGES** | 从 Windows 10 版本 1703 开始，此标志指定应使用[大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且文件映射对象必须已使用 **SEC\_LARGE\_PAGES** 选项创建。 如果为 *lpBaseAddress* 提供非 null 值，则该值必须是 **GetLargePageMinimum** 的倍数。 | | **FILE\_MAP\_TARGETS\_INVALID** | 将映射文件中的所有位置设置为控制流防护 (CFG) 的无效目标。 此标志类似于 **PAGE\_TARGETS\_INVALID**。 将此标志与执行访问权限 **FILE\_MAP\_EXECUTE**结合使用。 对这些页面中的位置的任何间接调用都将失败 CFG 检查，并且进程将终止。 分配的可执行页面的默认行为是标记为 CFG 的有效调用目标。 |     对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *dwDesiredAccess* 参数不起作用，应设置为任何有效值，例如 **FILE\_MAP\_READ**。  有关访问文件映射对象的详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] dwFileOffsetHigh  视图开始位置的文件偏移量的高阶 **DWORD** 。  [in] dwFileOffsetLow  视图开始位置的文件偏移量的低序 **DWORD** 。 高偏移量和低偏移量的组合必须在文件映射中指定偏移量。 它们还必须与系统的内存分配粒度匹配。 也就是说，偏移量必须是分配粒度的倍数。 若要获取系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数，该函数填充 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的成员。  [in] dwNumberOfBytesToMap  要映射到视图的文件映射的字节数。 所有字节都必须在 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga) 指定的最大大小内。 如果此参数为 0 (零) ，则映射将从指定的偏移量扩展到文件映射的末尾。  [in, optional] lpBaseAddress  指向调用进程地址空间中映射开始位置的内存地址的指针。 这必须是系统内存分配粒度的倍数，否则函数将失败。 若要确定系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。 如果指定地址上没有足够的地址空间，则函数将失败。  如果 *lpBaseAddress* 参数为 **NULL**，则操作系统选择映射地址。  虽然可以指定现在 (操作系统) 不使用的安全地址，但不能保证该地址随时间推移保持安全。 因此，最好让操作系统选择地址。 在这种情况下，不会将指针存储在内存映射文件中;将存储与文件映射基的偏移量，以便可以在任何地址使用映射。  [in] nndPreferred  物理内存应驻留的 NUMA 节点。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **NUMA\_NO\_PREFERRED\_NODE**  0xffffffff | 无 NUMA 节点为首选节点。 这与调用 [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) 函数相同。 |  返回值 如果函数成功，则返回值为映射视图的起始地址。  如果函数失败，则返回值为 NULL。 若要获得更多的错误信息，请调用 GetLastError 函数。 注解 映射文件会使文件的指定部分在调用进程的地址空间中可见。  对于大于地址空间的文件，一次只能映射一小部分文件数据。 第一个视图完成后，取消映射该视图并映射新视图。  若要获取视图的大小，请使用 [VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualqueryex) 函数。  页面文件支持的文件映射对象中页面的初始内容为 0 (零) 。  如果提供了建议的映射地址，则文件将映射到指定的地址 (向下舍入到最近的 64 KB 边界) 如果指定地址处有足够的地址空间。 如果地址空间不足，函数将失败。  通常，建议的地址用于指定文件应映射到多个进程中的同一地址。 这要求地址空间区域在所有涉及的进程中都可用。 不能在用于映射的区域进行其他内存分配，包括使用 [VirtualAllocExNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocexnuma) 函数来保留内存。  如果 *lpBaseAddress* 参数指定基偏移量，则当调用进程尚未使用指定的内存区域时，函数将成功。 系统不确保同一内存区域可用于其他 32 位进程中的内存映射文件。  文件 (或文件映射对象的多个视图及其映射文件) 如果它们在指定时间包含相同的数据，则它们是 *一致的* 。 如果文件视图派生自同一个文件映射对象，则会发生此情况。 进程可以使用 [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle) 函数将文件映射对象句柄复制到另一个进程中，或者另一个进程可以使用 [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga) 函数按名称打开文件映射对象。  有一个重要例外，从同一文件支持的任何文件映射对象派生的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **MapViewOfFileExNuma** 适用于远程文件，但它不会使它们保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只能看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  不保证文件的映射视图与 [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile) 或 [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) 函数访问的文件一致。  若要防止 **EXCEPTION\_IN\_PAGE\_ERROR** 异常，请使用结构化异常处理来保护写入或读取除页面文件以外的文件的内存映射视图的任何代码。 有关详细信息，请参阅 [从文件视图中读取和写入](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/reading-and-writing-from-a-file-view)。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  若要使文件具有可执行权限，应用程序必须使用 **PAGE\_EXECUTE\_READWRITE** 或 **PAGE\_EXECUTE\_READ** 调用 [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa) 函数，然后使用 **FILE\_MAP\_EXECUTE** | **FILE\_MAP\_WRITE** 或 **FILE\_MAP\_EXECUTE** | **FILE\_MAP\_READ**调用 **MapViewOfFileExNuma** 函数。  在 Windows Server 2012 中，以下技术支持此函数。  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 |  要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMappingNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappingnumaa)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [NUMA 支持](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/numa-support)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [ReadFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-readfile)  [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)  [WriteFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-writefile) |

## MapViewOfFileFromApp 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将文件映射的视图映射到调用的 Windows 应用商店应用的地址空间中。 语法 C++复制  PVOID MapViewOfFileFromApp(  [in] HANDLE hFileMappingObject,  [in] ULONG DesiredAccess,  [in] ULONG64 FileOffset,  [in] SIZE\_T NumberOfBytesToMap  ); 参数 [in] hFileMappingObject  文件映射对象的句柄。 [CreateFileMappingFromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-createfilemappingfromapp) 函数返回此句柄。  [in] DesiredAccess  文件映射对象的访问类型，该对象确定页面的页面保护。 此参数可以是以下值之一，也可以是多个值的按位 OR 组合（如果适用）。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_ALL\_ACCESS** | 映射文件的读/写视图。 创建文件映射对象时必须具有 **PAGE\_READWRITE** 保护。  与 **MapViewOfFileFromApp** 一起使用时， **FILE\_MAP\_ALL\_ACCESS** 等效于 **FILE\_MAP\_WRITE**。 | | **FILE\_MAP\_READ** | 映射文件的只读视图。 尝试写入文件视图会导致访问冲突。  创建文件映射对象时必须具有 **PAGE\_READONLY**、 **PAGE\_READWRITE**、 **PAGE\_EXECUTE\_READ**或 **PAGE\_EXECUTE\_READWRITE** 保护。 | | **FILE\_MAP\_WRITE** | 映射文件的读/写视图。 创建文件映射对象时必须具有 **PAGE\_READWRITE** 保护。  与 **MapViewOfFileFromApp** 一起使用时， (FILE\_MAP\_WRITE | FILE\_MAP\_READ) 等效于 **FILE\_MAP\_WRITE**。 |     使用按位 OR，可以将上述值与这些值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **FILE\_MAP\_COPY** | 映射文件的写入时复制视图。 文件映射对象必须已**创建PAGE\_READONLY、PAGE\_READ\_EXECUTE**、**PAGE\_WRITECOPY**或**PAGE\_READWRITE**保护。  当进程写入写入时复制页时，系统会将原始页面复制到进程专用的新页。 新页面由分页文件提供支持。 对新页面的保护从写入时复制更改为读/写。  指定写入时复制访问权限时，系统将和进程提交费用用于整个视图，因为调用进程可能会写入视图中的每个页面，从而使所有页面都成为私有页面。 新页面的内容永远不会写回到原始文件，并且当视图未映射时会丢失。 | | **FILE\_MAP\_LARGE\_PAGES** | 从 Windows 10 版本 1703 开始，此标志指定应使用[大页面支持](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/large-page-support)映射视图。 视图的大小必须是 [GetLargePageMinimum](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-getlargepageminimum) 函数报告的大型页面大小的倍数，并且文件映射对象必须已使用 **SEC\_LARGE\_PAGES** 选项创建。 如果为 *lpBaseAddress* 提供非 null 值，则该值必须是 **GetLargePageMinimum** 的倍数。 | | **FILE\_MAP\_TARGETS\_INVALID** | 将映射文件中的所有位置设置为控制流防护 (CFG) 的无效目标。 此标志类似于 **PAGE\_TARGETS\_INVALID**。 将此标志与执行访问权限 **FILE\_MAP\_EXECUTE**结合使用。 对这些页面中位置的任何间接调用都将失败 CFG 检查，并且进程将终止。 分配的可执行页面的默认行为是标记为 CFG 的有效调用目标。 |     对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *dwDesiredAccess* 参数不起作用，应设置为任何有效值，例如 **FILE\_MAP\_READ**。  有关访问文件映射对象的详细信息，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] FileOffset  视图要开始的文件偏移量。 偏移量必须在文件映射中指定偏移量。 它们还必须匹配系统的内存分配粒度。 也就是说，偏移量必须是分配粒度的倍数。 若要获取系统的内存分配粒度，请使用 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数，该函数填充 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的成员。  [in] NumberOfBytesToMap  要映射到视图的文件映射的字节数。 所有字节都必须在 [CreateFileMappingFromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-createfilemappingfromapp) 指定的最大大小内。 如果此参数为 0 (零) ，则映射将从指定的偏移量扩展到文件映射的末尾。 返回值 如果函数成功，则返回值是映射视图的起始地址。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 有一个重要例外，派生自由同一文件支持的任何文件映射对象的文件视图在特定时间是一致的或相同的。 对于进程内的视图以及由不同进程映射的视图，可以保证一致性。  异常与远程文件相关。 尽管 **MapViewOfFileFromApp** 适用于远程文件，但它并不能使它们保持一致。 例如，如果两台计算机都将一个文件映射为可写文件，并且都更改了同一页，则每台计算机只看到自己对页面的写入。 在磁盘上更新数据时，不会合并数据。  仅当应用具有 **codeGeneration** 功能时，才能成功请求可执行保护。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [创建文件视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-file-view)  [DuplicateHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-duplicatehandle)  [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) |

## MapViewOfFileNuma2 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 将文件视图或页面文件支持的节映射到指定进程的地址空间中。 语法 C++复制  PVOID MapViewOfFileNuma2(  [in] HANDLE FileMappingHandle,  [in] HANDLE ProcessHandle,  [in] ULONG64 Offset,  [in, optional] PVOID BaseAddress,  [in] SIZE\_T ViewSize,  [in] ULONG AllocationType,  [in] ULONG PageProtection,  [in] ULONG PreferredNode  ); 参数 [in] FileMappingHandle  要映射到指定进程的地址空间的节的 **HANDLE** 。  [in] ProcessHandle  节将映射到的进程句 **柄** 。  [in] Offset  部分开头的偏移量。 这必须是 64k 对齐的。  [in, optional] BaseAddress  视图的所需基址。 地址向下舍入到最近的 64k 边界。 如果此参数为 **NULL**，则系统选取基址。  [in] ViewSize  要映射的字节数。 值为零 (0) 指定映射整个部分。  [in] AllocationType  分配的类型。 此参数可以是零 (0) 或以下常量值之一：   * **MEM\_RESERVE** - 映射保留视图 * **MEM\_LARGE\_PAGES** - 映射大型页面视图   [in] PageProtection  所需的页面保护。  对于使用 **SEC\_IMAGE** 属性创建的文件映射对象， *PageProtection* 参数不起作用，应设置为任何有效值，例如 **PAGE\_READONLY**。  [in] PreferredNode  此内存的首选 NUMA 节点。 返回值 如果成功，则返回映射视图的基址。 否则，返回 **NULL** ，并使用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 提供扩展错误状态。 要求  |  |  | | --- | --- | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | Onecore.lib;Onecoreuap.lib | | **DLL** | Api-ms-win-core-memory-l1-1-5.dll |  另请参阅 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileNuma](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffilenuma2) |

## OpenFileMappingA 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开命名文件映射对象。 语法 C++复制  HANDLE OpenFileMappingA(  [in] DWORD dwDesiredAccess,  [in] BOOL bInheritHandle,  [in] LPCSTR lpName  ); 参数 [in] dwDesiredAccess  对文件映射对象的访问。 针对目标文件映射对象上的任何安全描述符检查此访问。 有关值列表，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。  [in] bInheritHandle  如果此参数为 **TRUE**，则 [CreateProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa) 函数创建的进程可以继承句柄;否则，无法继承句柄。  [in] lpName  要打开的文件映射对象的名称。 如果文件映射对象具有按此名称打开的句柄，并且映射对象上的安全描述符与 *dwDesiredAccess* 参数不冲突，则打开操作会成功。 名称可以具有“Global\”或“Local\”前缀，以显式打开全局命名空间或会话命名空间中的对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。 使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0，下一个登录用户使用会话 1，依此。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。 返回值 如果函数成功，则返回值是指定文件映射对象的打开句柄。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 **OpenFileMapping** 返回的句柄可以与需要文件映射对象的句柄的任何函数一起使用。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  当不再需要时，调用方应调用 **OpenFileMapping** 返回的句柄释放，并调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)。  在 Windows Server 2012 中，以下技术支持此函数。  展开表   |  |  | | --- | --- | | **技术** | **支持** | | 服务器消息块 (SMB) 3.0 协议 | 是 | | SMB 3.0 透明故障转移 (TFO) | 是 | | 具有横向扩展文件共享的 SMB 3.0 (SO) | 是 | | 群集共享卷文件系统 (CSV) | 是 | | 弹性文件系统 (ReFS) | 是 |    示例 有关示例，请参阅 [创建命名共享内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-named-shared-memory)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h、Memoryapi.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  内存管理函数  [共享文件和内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/sharing-files-and-memory) |

## OpenFileMappingFromApp 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 打开命名文件映射对象。 语法 C++复制  HANDLE OpenFileMappingFromApp(  [in] ULONG DesiredAccess,  [in] BOOL InheritHandle,  [in] PCWSTR Name  ); parameters [in] DesiredAccess  对文件映射对象的访问。 针对目标文件映射对象上的任何安全描述符检查此访问。 有关值列表，请参阅 [文件映射安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/file-mapping-security-and-access-rights)。 仅当应用具有 **codeGeneration** 功能时，才能打开文件映射对象**以FILE\_MAP\_EXECUTE**访问。  [in] InheritHandle  如果此参数为 **TRUE**，则 [CreateProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createprocessa) 函数创建的进程可以继承句柄;否则，无法继承句柄。  [in] Name  要打开的文件映射对象的名称。 如果文件映射对象具有此名称的打开句柄，并且映射对象上的安全描述符与 *DesiredAccess* 参数不冲突，则打开操作会成功。 名称可以具有“Global”或“Local”前缀，以显式打开全局命名空间或会话命名空间中的对象。 名称的其余部分可以包含除反斜杠字符 (\) 以外的任何字符。 有关详细信息，请参阅 [内核对象命名空间](https://learn.microsoft.com/zh-cn/windows/desktop/TermServ/kernel-object-namespaces)。 使用终端服务会话实现快速用户切换。 第一个登录用户使用会话 0，下一个登录用户使用会话 1，依此。 内核对象名称必须遵循终端服务概述的准则，以便应用程序可以支持多个用户。 返回值 如果函数成功，则返回值是指定文件映射对象的打开句柄。  如果函数失败，则返回值为 NULL。 要获得更多的错误信息，请调用 GetLastError。 注解 可以通过实时 (JIT) 功能从 Windows 应用商店应用调用 **OpenFileMappingFromApp** 以使用 JIT 功能。 应用必须在应用清单文件中包含 **codeGeneration** 功能才能使用 JIT 功能。 **OpenFileMappingFromApp** 允许 Windows 应用商店应用在.NET Framework中使用 [MemoryMappedFile](https://learn.microsoft.com/zh-cn/dotnet/api/system.io.memorymappedfiles.memorymappedfile) 类。  **OpenFileMappingFromApp** 返回的句柄可以与需要文件映射对象句柄的任何函数一起使用。  通过映射视图修改文件时，上次修改时间戳可能不会自动更新。 如果需要，调用方应使用 [SetFileTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/fileapi/nf-fileapi-setfiletime) 设置时间戳。  当不再需要时，调用方应通过调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 调用 **OpenFileMappingFromApp** 返回的句柄释放。 要求 展开表   |  |  | | --- | --- | | **最低受支持的客户端** | Windows 10 [桌面应用 |UWP 应用] | | **最低受支持的服务器** | Windows Server 2016 [桌面应用 |UWP 应用] | | **目标平台** | Windows | | **标头** | memoryapi.h (包括 Windows.h) | | **Library** | WindowsApp.lib | | **DLL** | Kernel32.dll |  另请参阅 [CreateFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-createfilemappinga)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  内存管理函数  [OpenFileMapping](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-openfilemappinga)  [共享文件和内存](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/sharing-files-and-memory) |

## UnmapViewOfFile 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从调用进程的地址空间中取消映射文件的映射视图。 语法 C++复制  BOOL UnmapViewOfFile(  [in] LPCVOID lpBaseAddress  ); 参数 [in] lpBaseAddress  指向要取消映射的文件的映射视图基址的指针。 此值必须与之前调用 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 系列中的函数之一返回的值相同。 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 取消映射文件的映射视图会使视图在进程的地址空间中占用的范围失效，并使该范围可用于其他分配。 它删除属于进程工作集的每个未映射虚拟页面的工作集条目，并减小进程的工作集大小。 它还会递减相应物理页的共享计数。  未映射视图中修改的页面在共享计数达到零之前不会写入磁盘，或者换而言之，直到从共享页面的所有进程的工作集中取消映射或剪裁这些页面。 即便如此，修改后的页面也会“懒洋洋地”写入磁盘：也就是说，修改可以在内存中缓存，并在以后写入磁盘。 为了在电源故障或系统崩溃时最大程度地降低数据丢失的风险，应用程序应使用 [FlushViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-flushviewoffile) 函数显式刷新修改后的页面。  尽管应用程序可能会关闭用于创建文件映射对象的文件句柄，但系统会保持打开相应的文件，直到文件的最后一个视图未映射。 最后一个视图尚未取消映射的文件保持打开状态，没有共享限制。  在 Windows Server 2012 中，以下技术支持此函数。  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 [在文件中创建视图](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/creating-a-view-within-a-file)。 要求 展开表   |  |  | | --- | --- | | **最低受支持的客户端** | Windows XP [桌面应用 | UWP 应用] | | **最低受支持的服务器** | Windows Server 2003 [桌面应用 | UWP 应用] | | **目标平台** | Windows | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll |  另请参阅 [关闭文件映射对象](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/closing-a-file-mapping-object)  [文件映射函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex)  内存管理函数 |

## UnmapViewOfFile2 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 取消映射文件或页面文件支持的节的以前映射视图。 语法 C++复制  BOOL UnmapViewOfFile2(  [in] HANDLE Process,  [in] PVOID BaseAddress,  [in] ULONG UnmapFlags  ); 参数 [in] Process  要从中取消映射节的进程句 **柄** 。  [in] BaseAddress  要取消映射的先前映射视图的基址。 此值必须与上一次调用 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 系列中的某个函数返回的值相同。  [in] UnmapFlags  此参数可以是零 (0) 或以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_UNMAP\_WITH\_TRANSIENT\_BOOST**  0x00000001 | 指定应暂时提升正在取消映射的页面的优先级， (自动进行短期衰减) ，因为调用方期望很快从另一个线程再次访问这些页面。 有关内存优先级的详细信息，请参阅 [SetThreadInformation (ThreadMemoryPriority)](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-setthreadinformation)函数。 | | **MEM\_PRESERVE\_PLACEHOLDER**  0x00000002 | 在使用 [MapViewOfFile3 或 MapViewOfFile3FromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile3)) 将占位符替换为映射视图后，将映射视图取消映射到占位符 (。 |  返回值 如果成功，则返回 **TRUE** 。 否则，返回 **FALSE** ，并使用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 提供扩展错误状态。 要求 展开表   | **要求** | **值** | | --- | --- | | **最低受支持的客户端** | Windows 10版本 1703 [仅限桌面应用] | | **最低受支持的服务器** | Windows Server 2016 [仅限桌面应用] | | **目标平台** | Windows | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | WindowsApp.lib | | **DLL** | Kernelbase.dll |  另请参阅 [MapViewOfFile2](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile2)  [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile)  [UnmapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffileex) |

## UnmapViewOfFileEx 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 这是 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 的扩展版本，它采用其他 flags 参数。 语法 C++复制  BOOL UnmapViewOfFileEx(  [in] PVOID BaseAddress,  [in] ULONG UnmapFlags  ); 参数 [in] BaseAddress  指向要取消映射的文件的映射视图基址的指针。 此值必须与上一次调用 [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile) 系列中的某个函数返回的值相同。  [in] UnmapFlags  此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | **MEM\_UNMAP\_WITH\_TRANSIENT\_BOOST**  0x00000001 | 指定应暂时提升正在取消映射的页面的优先级， (自动进行短期衰减) ，因为调用方期望很快从另一个线程再次访问这些页面。 有关内存优先级的详细信息，请参阅 [SetThreadInformation (ThreadMemoryPriority)](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-setthreadinformation)函数。 | | **MEM\_PRESERVE\_PLACEHOLDER**  0x00000002 | 在使用 [MapViewOfFile3 或 MapViewOfFile3FromApp](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile3)) 将占位符替换为映射视图后，将映射视图取消映射到占位符 (。 |  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 有关此函数的行为的详细信息，请参阅 [UnmapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile) 函数。 要求 展开表   | **要求** | **值** | | --- | --- | | **最低受支持的客户端** | Windows 8 [桌面应用 |UWP 应用] | | **最低受支持的服务器** | Windows Server 2012 [桌面应用 |UWP 应用] | | **目标平台** | Windows | | **标头** | memoryapi.h (包括 Windows.h、Memoryapi.h) | | **Library** | onecore.lib | | **DLL** | Kernel32.dll | |

# 这一节只是学习理论,没有代码