# 学习大纲

|  |
| --- |
|  |

|  |
| --- |
| 这里有3种内存分配方法 |

# 全局内存分配函数

## GlobalAlloc 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 从堆中分配指定的字节数。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  DECLSPEC\_ALLOCATOR HGLOBAL GlobalAlloc(  [in] UINT uFlags,  [in] SIZE\_T dwBytes  ); 参数 [in] uFlags  内存分配属性。 如果指定零，则默认值为 **GMEM\_FIXED**。 此参数可以是以下一个或多个值，但专门指出的不兼容组合除外。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GHND**  0x0042 | 将 **GMEM\_MOVEABLE** 和GMEM\_ZEROINIT组合在 **一**起。 | | **GMEM\_FIXED**  0x0000 | 分配固定内存。 返回值为指针。 | | **GMEM\_MOVEABLE**  0x0002 | 分配可移动内存。 内存块永远不会在物理内存中移动，但它们可以在默认堆中移动。  返回值是内存对象的句柄。 若要将句柄转换为指针，请使用 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数。  此值不能与 **GMEM\_FIXED** 组合使用。 | | **GMEM\_ZEROINIT**  0x0040 | 将内存内容初始化为零。 | | **GPTR**  0x0040 | 将 **GMEM\_FIXED** 和GMEM\_ZEROINIT组合在 **一**起。 |     以下值已过时，但提供这些值是为了兼容 16 位 Windows。 忽略它们。  **GMEM\_DDESHARE**  **GMEM\_DISCARDABLE**  **GMEM\_LOWER**  **GMEM\_NOCOMPACT**  **GMEM\_NODISCARD**  **GMEM\_NOT\_BANKED**  **GMEM\_NOTIFY**  **GMEM\_SHARE**  [in] dwBytes  要分配的字节数。 如果此参数为零，并且 *uFlags* 参数指定 **GMEM\_MOVEABLE**，则函数将返回一个标记为已放弃的内存对象的句柄。 返回值 如果函数成功，则返回值是新分配的内存对象的句柄。  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 Windows 内存管理不提供单独的本地堆和全局堆。 因此， **GlobalAlloc** 和 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 函数本质上是相同的。  可移动内存标志 **GHND** 和 **GMEM\_MOVABLE** 增加了不必要的开销，并且需要锁定才能安全使用。 除非文档明确指出应使用它们，否则应避免使用它们。  除非文档明确指出应使用全局函数，否则新应用程序应使用 [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 来分配和管理内存。 例如，全局函数仍与 Dynamic Data Exchange (DDE) 、剪贴板函数和 OLE 数据对象一起使用。  如果 **GlobalAlloc** 函数成功，则至少分配请求的内存量。 如果实际分配的金额大于请求的金额，则进程可以使用整个金额。 若要确定实际分配的字节数，请使用 [GlobalSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalsize) 函数。  如果堆中没有足够的可用空间来满足请求， **则 GlobalAlloc** 返回 **NULL**。 由于 **NULL** 用于指示错误，因此永远不会分配虚拟地址零。 因此，很容易检测 **NULL** 指针的使用。  使用此函数分配的内存保证在 8 字节边界上对齐。 若要执行动态生成的代码，请使用 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数分配内存，使用 [VirtualProtect](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualprotect) 函数授予 **PAGE\_EXECUTE** 访问权限。  若要释放内存，请使用 [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree) 函数。 使用 [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree) 释放 **GlobalAlloc** 分配的内存是不安全的。 示例 以下代码演示 **了 GlobalAlloc** 和 [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree) 的简单用法。  C++复制  #include <windows.h>  #include <stdio.h>  #include <tchar.h>  void \_cdecl main()  {  PSECURITY\_DESCRIPTOR pSD;  pSD = (PSECURITY\_DESCRIPTOR) GlobalAlloc(  GMEM\_FIXED,  sizeof(PSECURITY\_DESCRIPTOR));  // Handle error condition  if( pSD == NULL )  {  \_tprintf(TEXT("GlobalAlloc failed (%d)\n"), GetLastError());  return;  }  //see how much memory was allocated  \_tprintf(TEXT("GlobalAlloc allocated %d bytes\n"), GlobalSize(pSD));  // Use the memory allocated  // Free the memory when finished with it  GlobalFree(pSD);  } 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalDiscard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globaldiscard)  [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [GlobalSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalsize)  [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) | |

## GlobalDiscard 宏

|  |  |  |
| --- | --- | --- |
| 放弃指定的全局内存块。 内存对象的锁计数必须为零。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  void GlobalDiscard(  [in] h  ); 参数 [in] h  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 无 备注 尽管 **GlobalDiscard** 会放弃对象的内存块，但对象的句柄仍然有效。 进程随后可以将句柄传递给 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数，以分配由同一句柄标识的另一个全局内存块。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalFlags 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关指定全局内存对象的信息。  **注意** 此函数仅用于与 16 位版本的 Windows 兼容。 新应用程序应使用 [**堆函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)。 有关详细信息，请参阅“备注”。   语法 C++复制  UINT GlobalFlags(  [in] HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 如果函数成功，则返回值指定内存对象的分配值和锁计数。  如果函数失败，则返回值 **GMEM\_INVALID\_HANDLE**，表示全局句柄无效。 要获得更多的错误信息，请调用 GetLastError。 注解 返回值的低序字的低序字节包含对象的锁计数。 若要从返回值检索锁计数，请将 **GMEM\_LOCKCOUNT** 掩码与按位 AND (&) 运算符一起使用。 使用 **GMEM\_FIXED** 分配的内存对象的锁计数始终为零。  返回值的低序字的高阶字节指示内存对象的分配值。 可以是零，也可以 **是GMEM\_DISCARDED**。  与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [全局函数和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalDiscard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globaldiscard)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalFree 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 释放指定的全局内存对象并使其句柄失效。  **注意** 与其他内存管理功能相比，全局函数的开销更大，提供的功能更少。 除非文档指出应使用全局函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  HGLOBAL GlobalFree(  [in] \_Frees\_ptr\_opt\_ HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 释放使用 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 分配的内存是不安全的。 返回值 如果函数成功，则返回值为 **NULL**。  如果函数失败，则返回值等于全局内存对象的句柄。 要获得更多的错误信息，请调用 GetLastError。 注解 如果进程在释放内存后检查或修改内存，则可能会发生堆损坏，或者可能会生成访问冲突异常 (EXCEPTION\_ACCESS\_VIOLATION) 。  **GlobalFree** 函数将释放锁定的内存对象。 锁定的内存对象的锁计数大于零。 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数锁定全局内存对象，并将锁计数递增 1。 [GlobalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalunlock) 函数将其解锁，并将锁计数递减 1。 若要获取全局内存对象的锁计数，请使用 [GlobalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalflags) 函数。  如果应用程序在系统的调试版本下运行， **GlobalFree** 将发出一条消息，告知你正在释放锁定的对象。 如果要调试应用程序， **GlobalFree** 将在释放锁定的对象之前输入断点。 这样，便可以验证预期行为，然后继续执行。 示例 有关示例，请参阅 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalflags)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [GlobalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalunlock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalHandle 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索与指向全局内存块的指定指针关联的句柄。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  HGLOBAL GlobalHandle(  [in] LPCVOID pMem  ); 参数 [in] pMem  指向全局内存块的第一个字节的指针。 此指针由 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数返回。 返回值 如果函数成功，则返回值是指定全局内存对象的句柄。  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 [当 GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 函数分配具有**GMEM\_MOVEABLE**的内存对象时，它将返回对象的句柄。 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数将此句柄转换为指向内存块的指针，**GlobalHandle** 将指针转换回句柄。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalLock 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 锁定全局内存对象并返回指向对象内存块的第一个字节的指针。  **注意** 与其他内存管理功能相比，全局函数的开销更大，提供的功能更少。 除非文档指出应使用全局函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  LPVOID GlobalLock(  [in] HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 如果函数成功，则返回值是指向内存块的第一个字节的指针。  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 每个内存对象的内部数据结构包括最初为零的锁计数。 对于可移动内存对象， **GlobalLock** 将计数递增 1， [GlobalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalunlock) 函数将计数递减 1。 进程对对象的 **GlobalLock** 进行的每个成功调用都必须通过对 **GlobalUnlock** 的相应调用进行匹配。 除非使用 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数重新分配内存对象，否则不会移动或丢弃锁定的内存。 锁定的内存对象的内存块将保持锁定状态，直到其锁计数减为零，此时可以移动或丢弃它。  使用 **GMEM\_FIXED** 分配的内存对象的锁计数始终为零。 对于这些对象，返回的指针的值等于指定句柄的值。  如果指定的内存块已被丢弃，或者内存块的大小为零字节，则此函数返回 **NULL**。  放弃的对象始终具有锁计数为零。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [GlobalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalunlock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalMemoryStatus 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [**GlobalMemoryStatus** 可以返回不正确的信息。 请改用 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex) 函数。]  检索有关系统当前使用物理内存和虚拟内存的信息。 语法 C++复制  void GlobalMemoryStatus(  [out] LPMEMORYSTATUS lpBuffer  ); 参数 [out] lpBuffer  指向 [MEMORYSTATUS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-memorystatus) 结构的指针。 **GlobalMemoryStatus** 函数将有关当前内存可用性的信息存储到此结构中。 返回值 无 备注 在内存超过 4 GB 的计算机上， **GlobalMemoryStatus** 函数可能会返回错误的信息，报告值 –1 以指示溢出。 因此，应用程序应改用 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex) 函数。  在内存超过 2 GB 且内存小于 4 GB 的 Intel x86 计算机上，**GlobalMemoryStatus** 函数将始终在 [MEMORYSTATUS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-memorystatus) 结构的 **dwTotalPhys** 成员中返回 2 GB。 同样，如果总可用内存在 2 到 4 GB 之间，则 **MEMORYSTATUS** 结构的 **dwAvailPhys** 成员将向下舍入为 2 GB。 如果使用 **/LARGEADDRESSAWARE** 链接器选项链接可执行文件，则 **GlobalMemoryStatus** 函数将在两个成员中返回正确的物理内存量。  **GlobalMemoryStatus** 函数返回的信息是可变的。 不能保证对此函数的两次连续调用将返回相同的信息。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex)  [MEMORYSTATUS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/ns-winbase-memorystatus)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions)  [内存性能信息](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa965225(v=vs.85))  [虚拟地址空间和物理存储](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/virtual-address-space-and-physical-storage) |

## GlobalReAlloc 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 更改指定全局内存对象的大小或属性。 大小可以增加或减少。  **注意** 与其他内存管理功能相比，全局函数的开销更大，提供的功能更少。 除非文档指出应使用全局函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  DECLSPEC\_ALLOCATOR HGLOBAL GlobalReAlloc(  [in] \_Frees\_ptr\_ HGLOBAL hMem,  [in] SIZE\_T dwBytes,  [in] UINT uFlags  ); 参数 [in] hMem  要重新分配的全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 **GlobalReAlloc** 函数返回。  [in] dwBytes  内存块的新大小（以字节为单位）。 如果 *uFlags* 指定 **GMEM\_MODIFY**，则忽略此参数。  [in] uFlags  重新分配选项。 如果指定 **了GMEM\_MODIFY** ，则函数仅在忽略 *dwBytes* 参数 (修改内存对象的属性。) 否则，函数将重新分配内存对象。  可以选择将 **GMEM\_MODIFY** 与以下值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GMEM\_MOVEABLE**  0x0002 | 分配可移动内存。  如果内存是锁定 **GMEM\_MOVEABLE** 内存块或 **GMEM\_FIXED** 内存块，并且未指定此标志，则只能就地重新分配内存。 |     如果此参数未指定 **GMEM\_MODIFY**，则可以使用以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **GMEM\_ZEROINIT**  0x0040 | 如果内存对象的大小增大，则会导致其他内存内容初始化为零。 |  返回值 如果函数成功，则返回值是重新分配的内存对象的句柄。  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 如果 **GlobalReAlloc** 重新分配可移动对象，则返回值是内存对象的句柄。 若要将句柄转换为指针，请使用 [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数。  如果 **GlobalReAlloc** 重新分配固定对象，则返回的句柄值是内存块的第一个字节的地址。 若要访问内存，进程只需将返回值强制转换为指针。  如果 **GlobalReAlloc** 失败，则不会释放原始内存，并且原始句柄和指针仍然有效。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalDiscard](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globaldiscard)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalSize 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索指定全局内存对象的当前大小（以字节为单位）。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  SIZE\_T GlobalSize(  [in] HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 如果函数成功，则返回值是指定全局内存对象的大小（以字节为单位）。  如果指定的句柄无效或对象已被丢弃，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 内存块的大小可能大于分配内存时请求的大小。  若要验证指定对象的内存块未被丢弃，请在调用 **GlobalSize** 之前使用 [GlobalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalflags) 函数。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalflags)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## GlobalUnlock 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 递减与使用 **GMEM\_MOVEABLE** 分配的内存对象关联的锁计数。 此函数对使用 **GMEM\_FIXED** 分配的内存对象没有影响。  **注意** 与其他内存管理功能相比，全局函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用全局函数，否则新的应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++  BOOL GlobalUnlock(  [in] HGLOBAL hMem  ); 参数 [in] hMem  全局内存对象的句柄。 此句柄由 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 或 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数返回。 返回值 如果内存对象在减少锁计数后仍被锁定，则返回值为非零值。 如果内存对象在减少锁计数后解锁，则函数返回零， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)**返回NO\_ERROR**。  如果函数失败，则返回值为零， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **NO\_ERROR**以外的值。 注解 每个内存对象的内部数据结构包括最初为零的锁计数。 对于可移动内存对象， [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock) 函数将计数递增 1， **GlobalUnlock** 将计数递减 1。 对于进程对 **对象的 GlobalLock** 进行的每个调用，它最终必须调用 **GlobalUnlock**。 除非使用 [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc) 函数重新分配内存对象，否则不会移动或丢弃锁定的内存。 锁定的内存对象的内存块将保持锁定状态，直到其锁定计数减为零，此时可以移动或放弃该内存块。  使用 **GMEM\_FIXED** 分配的内存对象始终具有零锁计数。 如果指定的内存块是固定内存，则此函数返回 **TRUE**。  如果内存对象已解锁，**GlobalUnlock** 将返回 **FALSE**，**ERROR\_NOT\_LOCKED**[GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 报告。  进程不应依赖返回值来确定它随后必须为内存对象调用 **GlobalUnlock** 的次数。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc)  [GlobalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globallock)  [GlobalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

# 局部内存分配函数

## LocalAlloc 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 从堆中分配指定的字节数。  **注意** 与其他内存管理功能相比，本地函数的开销更大，提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  DECLSPEC\_ALLOCATOR HLOCAL LocalAlloc(  [in] UINT uFlags,  [in] SIZE\_T uBytes  ); 参数 [in] uFlags  内存分配属性。 默认值为 **LMEM\_FIXED** 值。 此参数可以是以下一个或多个值，但特别指出的不兼容组合除外。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **LHND**  0x0042 | 合并 **LMEM\_MOVEABLE** 和 **LMEM\_ZEROINIT**。 | | **LMEM\_FIXED**  0x0000 | 分配固定内存。 返回值是指向内存对象的指针。 | | **LMEM\_MOVEABLE**  0x0002 | 分配可移动内存。 内存块永远不会在物理内存中移动，但它们可以在默认堆中移动。  返回值是内存对象的句柄。 若要将句柄转换为指针，请使用 [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock) 函数。  此值不能与 **LMEM\_FIXED**组合使用。 | | **LMEM\_ZEROINIT**  0x0040 | 将内存内容初始化为零。 | | **LPTR**  0x0040 | 合并 **LMEM\_FIXED** 和 **LMEM\_ZEROINIT**。 | | **NONZEROLHND** | 与 **LMEM\_MOVEABLE** 相同。 | | **NONZEROLPTR** | 与 **LMEM\_FIXED** 相同。 |     以下值已过时，但提供这些值是为了与 16 位 Windows 兼容。 它们将被忽略。  **LMEM\_DISCARDABLE**  **LMEM\_NOCOMPACT**  **LMEM\_NODISCARD**  [in] uBytes  要分配的字节数。 如果此参数为零且 *uFlags* 参数指定 **LMEM\_MOVEABLE**，则函数将返回标记为已丢弃的内存对象的句柄。 返回值 如果函数成功，则返回值是新分配的内存对象的句柄。  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 Windows 内存管理不提供单独的本地堆和全局堆。 因此， **LocalAlloc** 和 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 函数实质上是相同的。  可移动内存标志 **LHND**、 **LMEM\_MOVABLE** 和 **NONZEROLHND** 会增加不必要的开销，并要求安全使用锁定。 除非文档明确指出应使用它们，否则应避免使用它们。  除非文档特别指出应使用本地函数，否则新应用程序应使用 [堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 例如，某些 Windows 函数分配必须使用 [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree) 释放的内存。  如果堆不包含满足请求的足够可用空间， **则 LocalAlloc** 返回 **NULL**。 由于 **NULL** 用于指示错误，因此永远不会分配虚拟地址 0。 因此，很容易检测到 **NULL** 指针的使用。  如果 **LocalAlloc** 函数成功，它将至少分配请求的量。 如果分配的金额大于请求的金额，则进程可以使用整个金额。 若要确定实际分配的字节数，请使用 [LocalSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localsize) 函数。  若要释放内存，请使用 [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree) 函数。 使用 [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree) 释放**通过 LocalAlloc** 分配的内存是不安全的。 示例 以下代码演示 **了 LocalAlloc** 和 [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree) 的简单用法。  C++复制  #include <windows.h>  #include <stdio.h>  #include <tchar.h>  void \_cdecl \_tmain()  {  LPTSTR pszBuf=NULL;  pszBuf = (LPTSTR)LocalAlloc(  LPTR,  MAX\_PATH\*sizeof(TCHAR));  // Handle error condition  if( pszBuf == NULL )  {  \_tprintf(TEXT("LocalAlloc failed (%d)\n"), GetLastError());  return;  }  //see how much memory was allocated  \_tprintf(TEXT("LocalAlloc allocated %d bytes\n"), LocalSize(pszBuf));  // Use the memory allocated  // Free the memory when finished with it  LocalFree(pszBuf);  } 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [堆函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)  [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree)  [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock)  [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc)  [LocalSize](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localsize)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalFlags 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关指定本地内存对象的信息。  **注意** 提供此函数只是为了与 16 位版本的 Windows 兼容。 新应用程序应使用 [**堆函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions)。 有关详细信息，请参阅“备注”。   语法 C++复制  UINT LocalFlags(  [in] HLOCAL hMem  ); 参数 [in] hMem  本地内存对象的句柄。 此句柄由 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 或 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 函数返回。 返回值 如果函数成功，则返回值指定内存对象的分配值和锁计数。  如果函数失败，则返回值 **LMEM\_INVALID\_HANDLE**，指示本地句柄无效。 要获得更多的错误信息，请调用 GetLastError。 注解 返回值的低序字的低序字节包含对象的锁计数。 若要从返回值检索锁计数，请将 **LMEM\_LOCKCOUNT** 掩码与按位 AND (&) 运算符一起使用。 使用 **LMEM\_FIXED** 分配的内存对象的锁计数始终为零。  返回值的低序字的高位字节指示内存对象的分配值。 可以是零或 **LMEM\_DISCARDABLE**。  与其他内存管理功能相比，本地函数的开销更大，提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [堆](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalflags)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalDiscard](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/nf-minwinbase-localdiscard)  [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock)  [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc)  [LocalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localunlock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalFree 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 释放指定的本地内存对象并使其句柄失效。  **注意** 与其他内存管理功能相比，本地函数的开销更大，提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++  HLOCAL LocalFree(  [in] \_Frees\_ptr\_opt\_ HLOCAL hMem  ); 参数 [in] hMem  本地内存对象的句柄。 此句柄由 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 或 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 函数返回。 释放使用 [GlobalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalalloc) 分配的内存是不安全的。 返回值 如果函数成功，则返回值为 **NULL**。  如果函数失败，则返回值等于本地内存对象的句柄。 要获得更多的错误信息，请调用 GetLastError。 注解 如果进程在释放内存后尝试检查或修改内存，则可能会发生堆损坏，或者可能会生成访问冲突异常 (EXCEPTION\_ACCESS\_VIOLATION) 。  如果 *hMem* 参数为 **NULL**， **则 LocalFree** 将忽略该参数并返回 **NULL**。  **LocalFree** 函数将释放锁定的内存对象。 锁定的内存对象的锁计数大于零。 [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock) 函数锁定本地内存对象，并将锁计数递增 1。 [LocalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localunlock) 函数将其解锁，并将锁计数递减 1。 若要获取本地内存对象的锁计数，请使用 [LocalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localflags) 函数。  如果应用程序在系统的调试版本下运行， **LocalFree** 将发出一条消息，告知你正在释放锁定的对象。 如果要调试应用程序， **LocalFree** 将在释放锁定的对象之前输入断点。 这样，便可以验证预期行为，然后继续执行。 示例 有关示例，请参阅 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [GlobalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalfree)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localflags)  [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock)  [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc)  [LocalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localunlock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalHandle 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索与指向本地内存对象的指定指针关联的句柄。  **注意** 与其他内存管理功能相比，本地函数的开销更大，提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  HLOCAL LocalHandle(  [in] LPCVOID pMem  ); 参数 [in] pMem  指向本地内存对象的第一个字节的指针。 此指针由 [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock) 函数返回。 返回值 如果函数成功，则返回值是指定本地内存对象的句柄。  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 当 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 函数分配具有 **LMEM\_MOVEABLE**的本地内存对象时，它将返回该对象的句柄。 [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock) 函数将此句柄转换为指向对象内存块的指针，**LocalHandle** 将指针转换回句柄。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalLock 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 锁定本地内存对象并返回指向对象内存块的第一个字节的指针。  **注意** 与其他内存管理功能相比，本地函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  LPVOID LocalLock(  [in] HLOCAL hMem  ); 参数 [in] hMem  本地内存对象的句柄。 此句柄由 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 或 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 函数返回。 返回值 如果函数成功，则返回值是指向内存块的第一个字节的指针。  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 每个内存对象的内部数据结构包括最初为零的锁计数。 对于可移动内存对象， **LocalLock** 将计数递增 [1，LocalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localunlock) 函数将计数递减 1。 进程对对象的 **LocalLock** 进行的每个成功调用都必须通过对 **LocalUnlock** 的相应调用进行匹配。 除非使用 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 函数重新分配内存对象，否则不会移动或丢弃锁定的内存。 锁定的内存对象的内存块在内存中保持锁定状态，直到其锁计数减少到零，此时可以移动或放弃它。  使用 **LMEM\_FIXED** 分配的内存对象始终具有零锁计数。 对于这些对象，返回的指针的值等于指定句柄的值。  如果指定的内存块已被丢弃，或者内存块具有零字节大小，则此函数返回 **NULL**。  丢弃的对象始终具有零的锁计数。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localflags)  [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc)  [LocalUnlock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localunlock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalReAlloc 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 更改指定本地内存对象的大小或属性。 大小可以增加或减少。  **注意** 与其他内存管理功能相比，本地函数的开销更大，提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++  DECLSPEC\_ALLOCATOR HLOCAL LocalReAlloc(  [in] \_Frees\_ptr\_opt\_ HLOCAL hMem,  [in] SIZE\_T uBytes,  [in] UINT uFlags  ); 参数 [in] hMem  要重新分配的本地内存对象的句柄。 此句柄由 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 或 **LocalReAlloc** 函数返回。  [in] uBytes  内存块的新大小（以字节为单位）。 如果 *uFlags* 指定 **LMEM\_MODIFY**，则忽略此参数。  [in] uFlags  重新分配选项。 如果指定 **了LMEM\_MODIFY** ，则函数仅在忽略 *uBytes* 参数 (修改内存对象的属性。) 否则，函数将重新分配内存对象。  可以选择将 **LMEM\_MODIFY** 与以下值组合在一起。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **LMEM\_MOVEABLE**  0x0002 | 分配固定或可移动内存。  如果内存是锁定 **LMEM\_MOVEABLE** 内存块或 **LMEM\_FIXED** 内存块，并且未指定此标志，则只能就地重新分配内存。 |     如果此参数未指定 **LMEM\_MODIFY**，则可以使用以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **LMEM\_ZEROINIT**  0x0040 | 如果内存对象的大小增大，则会导致其他内存内容初始化为零。 |  返回值 如果函数成功，则返回值是重新分配的内存对象的句柄。  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 如果 **LocalReAlloc** 失败，则不会释放原始内存，并且原始句柄和指针仍然有效。  如果 **LocalReAlloc** 重新分配固定对象，则返回的句柄值是内存块的第一个字节的地址。 若要访问内存，进程只需将返回值强制转换为指针。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalFree](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localfree)  [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalSize 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索指定本地内存对象的当前大小（以字节为单位）。  **注意** 与其他内存管理功能相比，本地函数具有更大的开销，并且提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局函数和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++复制  SIZE\_T LocalSize(  [in] HLOCAL hMem  ); 参数 [in] hMem  本地内存对象的句柄。 此句柄由 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)、 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 或 [LocalHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localhandle) 函数返回。 返回值 如果函数成功，则返回值是指定本地内存对象的大小（以字节为单位）。 如果指定的句柄无效或对象已被丢弃，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 内存块的大小可能大于分配内存时请求的大小。  若要验证指定对象的内存块未被丢弃，请在调用 **LocalSize** 之前调用 [LocalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localflags) 函数。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localflags)  [LocalHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localhandle)  [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

## LocalUnlock 函数

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 递减与分配有 **LMEM\_MOVEABLE**的内存对象关联的锁计数。 此函数对使用 **LMEM\_FIXED** 分配的内存对象没有影响。  **注意** 与其他内存管理功能相比，本地函数的开销更大，提供的功能更少。 除非文档指出应使用本地函数，否则新应用程序应使用 [**堆**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/heap-functions) 函数。 有关详细信息，请参阅 [**全局和本地函数**](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)。   语法 C++  BOOL LocalUnlock(  [in] HLOCAL hMem  ); 参数 [in] hMem  本地内存对象的句柄。 此句柄由 [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc) 或 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 函数返回。 返回值 如果在递减锁计数后内存对象仍被锁定，则返回值为非零值。 如果在递减锁计数后解锁内存对象，则函数返回零， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **NO\_ERROR**。  如果函数失败，则返回值为零， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 返回 **除 NO\_ERROR** 以外的值。 注解 每个内存对象的内部数据结构包括最初为零的锁计数。 对于可移动内存对象， [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock) 函数将计数递增 1， **LocalUnlock** 将计数递减 1。 对于进程对对象的 **LocalLock** 进行的每个调用，它最终必须调用 **LocalUnlock**。 除非使用 [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc) 函数重新分配内存对象，否则不会移动或丢弃锁定的内存。 锁定的内存对象的内存块将保持锁定状态，直到其锁计数减为零，此时可以移动或丢弃它。  如果内存对象已解锁， **则 LocalUnlock** 将返回 **FALSE** ， [并且 GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 报告 **ERROR\_NOT\_LOCKED**。 使用 **LMEM\_FIXED** 分配的内存对象的锁计数始终为零，并导致 **ERROR\_NOT\_LOCKED** 错误。  进程不应依赖返回值来确定它随后必须为内存块调用 **LocalUnlock** 的次数。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [全局和本地函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/global-and-local-functions)  [LocalAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localalloc)  [LocalFlags](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localflags)  [LocalLock](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-locallock)  [LocalReAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-localrealloc)  [内存管理函数](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/memory-management-functions) |

# 堆函数

# 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) |

# 注意,

## 1.这三种内存分配方式中,global和local函数的开销都很大,不太建议使用,我们更加建议使用Heapxxx函数,也就是第三种分配方式.效率更高.前面2种是早期windows使用的,现在还是可以使用,学习他们是因为1.他们的参数比较简单,2.使得我们可以看得懂别人写的代码.所以如果了愿意,还是可以使用的.

## 2.前面2种分配方法只能够在进程默认堆里面分配内存,如果你需要创建多个堆,并且在这些堆里面分配内存,了就只能够使用HeapXxx函数了

# 演练

## 1.新建一个win32控制台应用程序,取名Lesson89-mem-alloc,在向导里面选择控制台,然后点击确定

|  |
| --- |
|  |
|  |

## 2.默认是没有包含windows头文件的,我们需要把它包含进来.

|  |
| --- |
|  |

## 3.然后我们编写下面的代码

### Lesson89-mem-alloc.cpp

|  |
| --- |
| // Lesson89-mem-alloc.cpp : 定义控制台应用程序的入口点。  #include "stdafx.h"  //需要将#include<Windows.h>和#include<locale.h>写在stdafx.h头文件里面  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL,"");//设置地域编码,不设置的话没有显示  LPVOID lpMem;  HGLOBAL hMem;  //内存分配需要注意:如果了分配的是可移动的内存,那么它返回一个句柄,如果了分配的是固定大小的内存,它返回一个指针  hMem = GlobalAlloc(GMEM\_MOVEABLE | GMEM\_ZEROINIT,1024);//分配方法内存是按页计算的不管了写多少  if(hMem==NULL)  {  \_tprintf(\_T("分配内存失败,错误码:%d"),GetLastError());  return -1;  }  //因为这里分配的是可移动的内存,所以在操作它的时候需要先锁定,操作完成后在解锁  lpMem = GlobalLock(hMem);  lstrcpy((LPTSTR)lpMem,\_T("Hello,Global Memory!!"));  GlobalUnlock(hMem);  \_tprintf(TEXT("内存地址为:0x%.8x\n"),(LPTSTR)lpMem);  \_tprintf(TEXT("内存大小为:%d\n"),GlobalSize(hMem));  \_tprintf(TEXT("内存内容为:%s\n"),(LPTSTR)lpMem);  \_tprintf(TEXT("================================================\n"));  //MessageBox(NULL,(LPTSTR)lpMem,(LPTSTR)lpMem,0);//如果不设置地域编码,这个内存的内容用消息框可以显示用\_tprintf显示不了  //如果需要更大内存可以用GlobalReAlloc来程序分配内存  hMem = GlobalReAlloc(hMem,2048,GHND);  lpMem = GlobalLock(hMem);  lstrcpy((LPTSTR)lpMem,\_T("Hello,Global Memory!!We've Got a Bigger Memory"));  GlobalUnlock(hMem);  \_tprintf(TEXT("内存地址为:0x%.8x\n"),(LPTSTR)lpMem);  \_tprintf(TEXT("内存大小为:%d\n"),GlobalSize(hMem));  \_tprintf(TEXT("内存内容为:%s\n"),(LPTSTR)lpMem);  GlobalFree(hMem);  //下面的代码OK  //LPVOID lpMem;  //HLOCAL hMem;  ////内存分配需要注意:如果了分配的是可移动的内存,那么它返回一个句柄,如果了分配的是固定大小的内存,它返回一个指针  //hMem = LocalAlloc(LHND,1024);//分配方法内存是按页计算的不管了写多少  //if(hMem==NULL)  //{  // \_tprintf(\_T("分配内存失败,错误码:%d"),GetLastError());  // return -1;  //}  ////因为这里分配的是可移动的内存,所以在操作它的时候需要先锁定,操作完成后在解锁  //lpMem = LocalLock(hMem);  //lstrcpy((LPTSTR)lpMem,\_T("Hello,Local Memory!!"));  ////\_tprintf(\_T("内存地址为:0x%.8x\n"),lpMem);  //LocalUnlock(hMem);  //MessageBox(NULL,(LPTSTR)lpMem,(LPTSTR)lpMem,0);  //LocalFree(hMem);  //下面的代码OK  //LPVOID lpMem;  //HANDLE hHeap;  //hHeap = GetProcessHeap();  //lpMem = HeapAlloc(hHeap,HEAP\_ZERO\_MEMORY,1024);  //if(lpMem == NULL)  //{  // \_tprintf(\_T("分配内存失败,错误码:%d"),GetLastError());  // return -1;  //}  //lstrcpy((LPTSTR)lpMem,\_T("Hello,Memory!!"));  //\_tprintf(\_T("内存地址为:0x%.8x\n"),lpMem);  //MessageBox(NULL,(LPTSTR)lpMem,(LPTSTR)lpMem,0);//用消息框是可以的.  // HeapFree(hHeap,0,lpMem);  system("pause");      return 0;  } |

### 注意:使用UNICODE编程的时候,一定要包含locale.h,并且在main函数一开始的地方设置地域编码,否则程序什么都不显示

### stdafx.h

|  |
| --- |
| // stdafx.h : 标准系统包含文件的包含文件，  // 或是经常使用但不常更改的  // 特定于项目的包含文件  //  #pragma once  #include "targetver.h"  #include <stdio.h>  #include <tchar.h>  // TODO: 在此处引用程序需要的其他头文件  #include<Windows.h>  #include<locale.h> |

### stdafx.cpp

|  |
| --- |
| // stdafx.cpp : 只包括标准包含文件的源文件  // Lesson89-mem-alloc,.pch 将作为预编译头  // stdafx.obj 将包含预编译类型信息  #include "stdafx.h"  // TODO: 在 STDAFX.H 中  // 引用任何所需的附加头文件，而不是在此文件中引用 |

### 效果

|  |
| --- |
|  |

## 老师的例子

|  |
| --- |
|  |

### 此时是显示不了的,解决办法是在stdafx.h里面下面包含windows.h头文件和locale.h头文件还需要设置地域

|  |
| --- |
|  |
|  |

# 这一节的学习到此为止