VirtualAlloc是一个Windows API函数，该函数的功能是在调用进程的虚[地址空间](http://baike.so.com/doc/901224-952574.html),预定或者提交一部分页。

C概述

VirtualAlloc是一个Windows API函数，它包含在windows[系统文件](http://baike.so.com/doc/309368-327574.html)Kernel32.dll中，编程时直接使用就可以了，不需要再下载。

该函数的功能是在调用进程的虚[地址空间](http://baike.so.com/doc/901224-952574.html),预定或者提交一部分页

如果用于[内存分配](http://baike.so.com/doc/6467808-6681503.html)的话,并且分配类型未指定MEM\_RESET,则系统将自动设置为0;

[折叠](http://baike.so.com/doc/8677279-8998954.html)[编辑本段](http://baike.so.com/create/edit/?eid=8677279&sid=8998954&secid=2)VC声明

LPVOID VirtualAlloc{

LPVOID lpAddress, // 要分配的内存区域的地址

DWORD dwSize, // 分配的大小

DWORD flAllocationType, // 分配的类型

DWORD flProtect // 该内存的初始保护属性

};

[折叠](http://baike.so.com/doc/8677279-8998954.html)[编辑本段](http://baike.so.com/create/edit/?eid=8677279&sid=8998954&secid=3)说明

该函数的功能是在调用进程的虚地址空间,预定或者提交一部分页

如果用于内存分配的话,并且分配类型未指定MEM\_RESET,则该内存自动被初始化为0;

[折叠](http://baike.so.com/doc/8677279-8998954.html)[编辑本段](http://baike.so.com/create/edit/?eid=8677279&sid=8998954&secid=4)参数表说明

LPVOID lpAddress, 分配内存区域的地址。当你使用VirtualAlloc来提交一块以前保留的内存块的时候，lpAddress参数可以用来识别以前保留的内存块。如果这个参数是NULL，系统将会决定分配内存区域的位置，并且按64-KB向上取整(roundup)。

SIZE\_T dwSize, 要分配或者保留的区域的大小。这个参数以[字节](http://baike.so.com/doc/1114609-1179328.html)为单位，而不是页，系统会根据这个大小一直分配到下页的边界DWORD

flAllocationType, 分配类型 ,你可以指定或者合并以下标志:MEM\_COMMIT，MEM\_RESERVE和MEM\_TOP\_DOWN。

DWORD flProtect 指定了被分配区域的访问保护方式

分配类型 功能

| **可能的数值** | **含义** |
| --- | --- |
| **MEM\_COMMIT**0x1000 | 分配指定保留的内存页的物理存储在内存或磁盘上的分页文件。该函数的内存初始化为零。  保留并承诺在一个步骤的页面，**调用VirtualAlloc**的MEM\_COMMIT | MEM\_RESERVE 。  该函数将失败，如果你尝试提交一个还没有被保留页。由此产生的错误代码是**ERROR\_INVALID\_ADDRESS**。  尝试提交一个已经提交的页面不会导致函数失败。这意味着你可以承诺不首先确定当前承诺在每一页的状态页面。 |
| MEM\_RESERVE 0x2000 | 保留在没有任何实际的物理存储在内存或磁盘上的分页文件分配一个进程的虚拟地址空间的范围。你可以提交保留页的后续调用VirtualAlloc的功能。保留并承诺在一个步骤的页面，调用VirtualAlloc的MEM\_COMMIT|MEM\_RESERVE 。 |
| **MEM\_RESET**0x80000 | 表示，在所指定的内存范围内的数据*lpAddress*的dwSize不再是利益。页面不应该被读出或写入到分页文件。然而，将再次使用的内存块后，所以它不应该decommitted的 。这个值不能使用任何其他的价值 。  使用这个值并不保证范围与经营**MEM\_RESET**将包含零。如果你想的范围包含零，decommit内存，然后将它重新作出 。  当您指定**MEM\_RESET**，**VirtualAlloc**函数忽略值*flProtect*。但是，您仍然必须设置*flProtect*有效的值，比如，**PAGE\_NOACCESS**。  如果你使用**MEM\_RESET**的内存范围映射到一个文件，**VirtualAlloc**返回一个错误。如果它被映射到分页文件共享视图是唯一可以接受的的 。 |

分配类型 也可以是下列值

| **可能的数值** | **含义** |
| --- | --- |
| **MEM\_LARGE\_PAGES**0x20000000的 | 分配内存使用大页面支持。  大小和对齐必须是一个大页面的最低多个 。要获得这个值，使用**GetLargePageMinimum**。 |
| MEM\_PHYSICAL0x400000 | 储备的地址范围，可用于内存地址窗口扩展(AWE)的页面。此值必须使用MEM\_RESERVE，并没有其他值。 |
| **MEM\_TOP\_DOWN**0x100000 | 在尽可能高的地址分配内存。这可以比普通的分配速度较慢，尤其是当需要许多分配。 |
| **MEM\_WRITE\_WATCH**0x200000 | 导致系统来跟踪分配的地区，都写在页面 。如果指定此值，则还必须指定**MEM\_RESERVE**。  要检索的页面是否写入，因为该地区被分配或写跟踪状态被重置地址，调用**GetWriteWatch**功能。要重置写跟踪状态，调用**GetWriteWatch**或**ResetWriteWatch**。写跟踪功能仍然启用，直到该地区被释放。 |

访问类型

|  |  |
| --- | --- |
| PAGE\_READONLY 0x02 | 该区域为只读。如果应用程序试图访问区域中的页的时候，将会被拒绝访问。 |
| PAGE\_EXECUTE 0x10 | 区域包含可被执行的代码。试图读写该区域的操作将被拒绝。 |
| PAGE\_EXECUTE\_READ 0x20 | 区域包含可执行代码，应用程序只能读该区域。 |
| PAGE\_READWRITE 0x04 | 区域不可执行代码，应用程序可以读写该区域。 |
| PAGE\_EXECUTE\_READWRITE 0x40 | 区域可以执行代码，应用程序可以读写该区域。 |
| PAGE\_GUARD 0x100 | 区域第一次被访问时进入一个STATUS\_GUARD\_PAGE异常，这个标志要和其他保护标志合并使用，表明区域被第一次访问的权限 |
| PAGE\_NOACCESS 0x01 | 任何访问该区域的操作将被拒绝 |
| PAGE\_NOCACHE 0x200 | RAM中的页映射到该区域时将不会被微处理器缓存(cached) |

注:PAGE\_GUARD和PAGE\_NOCHACHE标志可以和其他标志合并使用以进一步指定页的特征。PAGE\_GUARD标志指定了一个防护页(guard page)，即当一个页被提交时会因第一次被访问而产生一个one-shot异常，接着取得指定的访问权限。PAGE\_NOCACHE防止当它映射到虚拟页的时候被微处理器缓存。这个标志方便[设备驱动](http://baike.so.com/doc/474579-502521.html)使用直接内存访问方式(DMA)来共享内存块。

[折叠](http://baike.so.com/doc/8677279-8998954.html)[编辑本段](http://baike.so.com/create/edit/?eid=8677279&sid=8998954&secid=5)返回值

如果调用成功,返回分配的首地址,

调用失败,返回NULL 你可以通过GetLastError函数来获取错误信息

[折叠](http://baike.so.com/doc/8677279-8998954.html)[编辑本段](http://baike.so.com/create/edit/?eid=8677279&sid=8998954&secid=6)笔记

VirtualAlloc可以通过并行多次调用提交一个区域的部分或全部来保留一个大的内存区域。多重调用提交同一块区域不会引起失败。这使得一个应用程序保留内存后可以随意提交将被写的页。当这种方式不在有效的时候，它会释放应用程序通过检测被保留页的状态看它是否在提交调用之前已经被提交