VirtualQueryEx是一种函数功能，查询地址空间中内存地址的信息。

函数原型

DWORD VirtualQueryEx(

HANDLE hProcess,

LPCVOID lpAddress,

PMEMORY\_BASIC\_INFORMATION lpBuffer,

DWORD dwLength

);

参数:

hProcess 进程句柄。

lpAddress 查询内存的地址。

lpBuffer 指向MEMORY\_BASIC\_INFORMATION结构的指针，用于接收内存信息。

dwLength MEMORY\_BASIC\_INFORMATION结构的大小。

返回值:

函数写入lpBuffer的字节数，如果不等于sizeof(MEMORY\_BASIC\_INFORMATION)表示失败。

## 备注

MEMORY\_BASIC\_INFORMATION在WinNT.h中定义如下:

typedef struct \_MEMORY\_BASIC\_INFORMATION {

PVOID BaseAddress; // 区域基地址。

PVOID AllocationBase; // 分配基地址。

DWORD AllocationProtect; // 区域被初次保留时赋予的保护属性。

SIZE\_T RegionSize; // 区域大小(以[字节](http://baike.so.com/doc/1114609-1179328.html)为计量单位)。

DWORD State; // 状态(MEM\_FREE、MEM\_RESERVE或 MEM\_COMMIT)。

DWORD Protect; // 保护属性。

DWORD Type; // 类型。

} MEMORY\_BASIC\_INFORMATION, \*PMEMORY\_BASIC\_INFORMATION;

成员解释:

BaseAddress 与lpAddress参数的值相同，但是四舍五入为页面的边界值。

AllocationBase 指明用VirtualAlloc函数分配内存区域的基地址。lpAddress在该区域之内。

AllocationProtect 指明该[地址空间](http://baike.so.com/doc/901224-952574.html)区域被初次保留时赋予该区域的保护属性。

PAGE\_READONLY 只读属性，如果试图进行写操作，将引发访问违规。如果系统区分只读、执行两种属性，那么试图在该区域执行代码也将引发访问违规。

PAGE\_READWRITE 允许读写。

PAGE\_EXECUTE 只允许执行代码，对该区域试图进行读写操作将引发访问违规。

PAGE\_EXECUTE\_READ 允许执行和读取。

PAGE\_EXECUTE\_READWRITE 允许读写和执行代码。

PAGE\_EXECUTE\_WRITECOPY 对于该[地址空间](http://baike.so.com/doc/901224-952574.html)的区域，不管执行什么操作，都不会引发访问违规。如果试图在该页面上的内存中进行写入操作，就会将它自己的私有页面(受页文件的支持)拷贝赋予该进程。

PAGE\_GUARD 在页面上写入一个字节时使应用程序收到一个通知(通过一个异常条件)。该标志有一些非常巧妙的用法。Windows 2000在创建[线程](http://baike.so.com/doc/115404-121766.html)[堆栈](http://baike.so.com/doc/4915223-5133919.html)时使用该标志。

PAGE\_NOACCESS 禁止一切访问。

PAGE\_NOCACHE 停用已提交页面的高速缓存。一般情况下最好不要使用该标志，因为它主要是供需要处理内存缓冲区的硬件设备[驱动程序](http://baike.so.com/doc/104837-110638.html)的开发人员使用的。

RegionSize 用于指明内存块从基地址即BaseAddress开始的所有页面的大小(以[字节](http://baike.so.com/doc/1114609-1179328.html)为计量单位)这些页面与含有用LpAddress参数设定的地址的页面拥有相同的保护属性、状态和类型。

State 用于指明所有相邻页面的状态。

MEM\_COMMIT 指明已分配[物理内存](http://baike.so.com/doc/5677263-5889936.html)或者系统页文件。

MEM\_FREE 空闲状态。该区域的虚拟地址不受任何内存的支持。该[地址空间](http://baike.so.com/doc/901224-952574.html)没有被保留。该状态下AllocationBase、AllocationProtect、Protect和Type等成员均未定义。

MEM\_RESERVE 指明页面被保留，但是没有分配任何物理内存。该状态下Protect成员未定。

Protect 用于指明所有相邻页面(内存块)的保护属性。这些页面与含有拥有相同的保属性、状态和类型。意义同AllocationProtect。

Type 用于指明支持所有相邻页面的物理存储器的类型(MEM\_IMAGE，MEM\_MAPPED或MEM\_PRIVATE)。这些相邻页面拥有相同的保护属性、状态和类型。如果是Windows 98，那么这个成员将总是MEM\_PRIVATE 。

MEM\_IMAGE 指明该区域的[虚拟地址](http://baike.so.com/doc/889976-940779.html)原先受内存映射的[映像文件](http://baike.so.com/doc/6234543-6447898.html)(如.exe或DLL文件)的支持，但也许不再受映像文件的支持。例如，当写入模块映像中的[全局变量](http://baike.so.com/doc/2493130-2634693.html)时，"写入时拷贝"的机制将由页文件来支持特定的页面，而不是受原始映像文件的支持。

MEM\_MAPPED 该区域的虚拟地址原先是受内存映射的数据文件的支持，但也许不再受数据文件的支持。例如，数据文件可以使用"写入时拷贝"的保护属性来映射。对文件的任何写入操作都将导致页文件而不是[原始数据](http://baike.so.com/doc/7862473-8136568.html)支持特定的页面。

MEM\_PRIVATE 指明该内存区域是私有的。不被其他进程共享。

示例代码:

#include "stdafx.h"

#include <windows.h>

#include <TCHAR.H>

BOOL ShowProcMemInfo(DWORD dwPID);

int \_tmain(int argc, char\* argv[])

{

ShowProcMemInfo( GetCurrentProcessId() );

return 0;

}

// 显示一个进程的内存状态 dwPID为进程ID

BOOL ShowProcMemInfo(DWORD dwPID)

{

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION,

FALSE,

dwPID);

if(hProcess == NULL)

return FALSE;

MEMORY\_BASIC\_INFORMATION mbi;

PBYTE pAddress = NULL;

TCHAR szInfo[200] = \_T("BaseAddr Size Type State Protect \n");

\_tprintf(szInfo);

while(TRUE)

{

if(VirtualQueryEx(hProcess, pAddress, &mbi, sizeof(mbi)) != sizeof(mbi))

{

break;

}

if((mbi.AllocationBase != mbi.BaseAddress) && (mbi.State != MEM\_FREE))

{

\_stprintf(szInfo, \_T(" %08X %8dK "),

mbi.BaseAddress,

mbi.RegionSize>>10);

}

else

{

\_stprintf(szInfo, \_T("%08X %8dK "),

mbi.BaseAddress,

mbi.RegionSize>>10);

}

LPCTSTR pStr = \_T("");

switch(mbi.Type)

{

case MEM\_IMAGE: pStr = \_T("MEM\_IMAGE "); break;

case MEM\_MAPPED: pStr = \_T("MEM\_MAPPED "); break;

case MEM\_PRIVATE: pStr = \_T("MEM\_PRIVATE"); break;

[default](http://baike.so.com/doc/5367471-5603233.html): pStr = \_T("-----------"); break;

}

\_tcscat(szInfo, pStr);

\_tcscat(szInfo, \_T(" "));

switch(mbi.State)

{

case MEM\_COMMIT: pStr = \_T("MEM\_COMMIT "); break;

case MEM\_RESERVE: pStr = \_T("MEM\_RESERVE"); break;

case MEM\_FREE: pStr = \_T("MEM\_FREE "); break;

[default](http://baike.so.com/doc/5367471-5603233.html): pStr = \_T("-----------"); break;

}

\_tcscat(szInfo, pStr);

\_tcscat(szInfo, \_T(" "));

switch(mbi.AllocationProtect)

{

case PAGE\_READONLY: pStr = \_T("PAGE\_READONLY "); break;

case PAGE\_READWRITE: pStr = \_T("PAGE\_READWRITE "); break;

case PAGE\_WRITECOPY: pStr = \_T("PAGE\_WRITECOPY "); break;

case PAGE\_EXECUTE: pStr = \_T("PAGE\_EXECUTE "); break;

case PAGE\_EXECUTE\_READ: pStr = \_T("PAGE\_EXECUTE\_READ "); break;

case PAGE\_EXECUTE\_READWRITE: pStr = \_T("PAGE\_EXECUTE\_READWRITE"); break;

case PAGE\_EXECUTE\_WRITECOPY: pStr = \_T("PAGE\_EXECUTE\_WRITECOPY"); break;

case PAGE\_GUARD: pStr = \_T("PAGE\_GUARD "); break;

case PAGE\_NOACCESS: pStr = \_T("PAGE\_NOACCESS "); break;

case PAGE\_NOCACHE: pStr = \_T("PAGE\_NOCACHE "); break;

[default](http://baike.so.com/doc/5367471-5603233.html): pStr = \_T("----------------------"); break;

}

\_tcscat(szInfo, pStr);

\_tcscat(szInfo, \_T("\n"));

\_tprintf(szInfo);

pAddress = ((PBYTE)mbi.BaseAddress + mbi.RegionSize);

}

CloseHandle(hProcess);

return TRUE;

}