# 学习大纲

|  |
| --- |
|  |
|  |
|  |
|  |

# API用法说明

## GlobalMemoryStatus函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [**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) |

### 这个函数不建议使用，会返回错误信息

## GlobalMemoryStatusEx函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关系统当前物理内存和虚拟内存使用情况的信息。 语法 C++复制  BOOL GlobalMemoryStatusEx(  [in, out] LPMEMORYSTATUSEX lpBuffer  ); 参数 [in, out] lpBuffer  指向 [MEMORYSTATUSEX](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-memorystatusex) 结构的指针，该结构接收有关当前内存可用性的信息。 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 可以使用 **GlobalMemoryStatusEx** 函数来确定应用程序可以分配多少内存，而不会严重影响其他应用程序。  **GlobalMemoryStatusEx** 函数返回的信息是可变的。 无法保证对此函数的两次顺序调用将返回相同的信息。  *lpBuffer* 中 [MEMORYSTATUSEX](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-memorystatusex) 结构的 **ullAvailPhys** 成员包括所有 NUMA 节点的内存。 示例 以下代码演示 **了 GlobalMemoryStatusEx** 函数的简单用法。  C++  // Sample output:  // There is 51 percent of memory in use.  // There are 2029968 total KB of physical memory.  // There are 987388 free KB of physical memory.  // There are 3884620 total KB of paging file.  // There are 2799776 free KB of paging file.  // There are 2097024 total KB of virtual memory.  // There are 2084876 free KB of virtual memory.  // There are 0 free KB of extended memory.  #include <windows.h>  #include <stdio.h>  #include <tchar.h>  // Use to convert bytes to KB  #define DIV 1024  // Specify the width of the field in which to print the numbers.  // The asterisk in the format specifier "%\*I64d" takes an integer  // argument and uses it to pad and right justify the number.  #define WIDTH 7  void \_tmain()  {  MEMORYSTATUSEX statex;  statex.dwLength = sizeof (statex);  GlobalMemoryStatusEx (&statex);  \_tprintf (TEXT("There is %\*ld percent of memory in use.\n"),  WIDTH, statex.dwMemoryLoad);  \_tprintf (TEXT("There are %\*I64d total KB of physical memory.\n"),  WIDTH, statex.ullTotalPhys/DIV);  \_tprintf (TEXT("There are %\*I64d free KB of physical memory.\n"),  WIDTH, statex.ullAvailPhys/DIV);  \_tprintf (TEXT("There are %\*I64d total KB of paging file.\n"),  WIDTH, statex.ullTotalPageFile/DIV);  \_tprintf (TEXT("There are %\*I64d free KB of paging file.\n"),  WIDTH, statex.ullAvailPageFile/DIV);  \_tprintf (TEXT("There are %\*I64d total KB of virtual memory.\n"),  WIDTH, statex.ullTotalVirtual/DIV);  \_tprintf (TEXT("There are %\*I64d free KB of virtual memory.\n"),  WIDTH, statex.ullAvailVirtual/DIV);  // Show the amount of extended memory available.  \_tprintf (TEXT("There are %\*I64d free KB of extended memory.\n"),  WIDTH, statex.ullAvailExtendedVirtual/DIV);  } 要求  |  |  | | --- | --- | | **标头** | sysinfoapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [MEMORYSTATUSEX](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-memorystatusex)  [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) |

### 64位的windows需要使用这个函数

## GetProcessMemoryInfo函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关指定进程的内存使用情况的信息。 语法 C++  BOOL GetProcessMemoryInfo(  [in] HANDLE Process,  [out] PPROCESS\_MEMORY\_COUNTERS ppsmemCounters,  [in] DWORD cb  ); 参数 [in] Process  进程的句柄。 句柄必须具有 **PROCESS\_QUERY\_INFORMATION** 或 **PROCESS\_QUERY\_LIMITED\_INFORMATION** 访问权限。 有关详细信息，请参阅 [进程安全和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  **Windows Server 2003 和 Windows XP：** 句柄必须具有 **PROCESS\_QUERY\_INFORMATION** 和 **PROCESS\_VM\_READ** 访问权限。  [out] ppsmemCounters  指向 [PROCESS\_MEMORY\_COUNTERS](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/ns-psapi-process_memory_counters)、 [PROCESS\_MEMORY\_COUNTERS\_EX](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/ns-psapi-process_memory_counters_ex)或 [PROCESS\_MEMORY\_COUNTERS\_EX2](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/ns-psapi-process_memory_counters_ex2) 结构的指针，该结构接收有关进程的内存使用情况的信息。  [in] cb  *ppsmemCounters* 结构的大小（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，则返回值为零。 要获得更多的错误信息，请调用 GetLastError。 注解 从 Windows 7 和 Windows Server 2008 R2 开始，Psapi.h 为 PSAPI 函数建立版本号。 PSAPI 版本号会影响用于调用程序必须加载的函数和库的名称。  如果 **PSAPI\_VERSION** 为 2 或更大，则此函数在 Psapi.h 中定义为 **K32GetProcessMemoryInfo** ，并在 Kernel32.lib 和 Kernel32.dll 中导出。 如果 **PSAPI\_VERSION** 为 1，则此函数在 Psapi.h 中定义为 **GetProcessMemoryInfo** ，并在 Psapi.lib 中导出，Psapi.dll 为调用 **K32GetProcessMemoryInfo** 的包装器。  必须在早期版本的 Windows 以及 Windows 7 及更高版本上运行的程序应始终将此函数称为 **GetProcessMemoryInfo**。 为确保符号正确解析，请将 Psapi.lib 添加到 **TARGETLIBS** 宏，并使用 **-DPSAPI\_VERSION=1** 编译程序。 若要使用运行时动态链接，请加载 Psapi.dll。 示例 有关示例，请参阅 [收集进程的内存使用情况信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/collecting-memory-usage-information-for-a-process)。 要求  |  |  | | --- | --- | | **标头** | psapi.h | | **Library** | Windows 7 和 Windows Server 2008 R2 上的 Kernel32.lib;如果 Windows 7 和 Windows Server 2008 R2 上的 PSAPI\_VERSION=1) ，则 Psapi.lib (;Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上的 Psapi.lib | | **DLL** | Windows 7 和 Windows Server 2008 R2 上的 Kernel32.dll;如果 Windows 7 和 Windows Server 2008 R2 上的 PSAPI\_VERSION=1) ，则为 Psapi.dll (;Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上的 Psapi.dll |  另请参阅 [EnumProcesses](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-enumprocesses)  [内存性能信息](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa965225(v=vs.85))  [PROCESS\_MEMORY\_COUNTERS](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/ns-psapi-process_memory_counters)  [PROCESS\_MEMORY\_COUNTERS\_EX](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/ns-psapi-process_memory_counters_ex)  [PSAPI 函数](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/psapi-functions)  [进程内存使用情况信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/process-memory-usage-information) |

## MEMORYSTATUS结构体的用法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 包含有关物理内存和虚拟内存的当前状态的信息。 [GlobalMemoryStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalmemorystatus) 函数将信息存储在 **MEMORYSTATUS** 结构中。 语法 C++复制  typedef struct \_MEMORYSTATUS {  DWORD dwLength;  DWORD dwMemoryLoad;  SIZE\_T dwTotalPhys;  SIZE\_T dwAvailPhys;  SIZE\_T dwTotalPageFile;  SIZE\_T dwAvailPageFile;  SIZE\_T dwTotalVirtual;  SIZE\_T dwAvailVirtual;  } MEMORYSTATUS, \*LPMEMORYSTATUS; 成员 dwLength  **MEMORYSTATUS** 数据结构的大小（以字节为单位）。 在调用 [GlobalMemoryStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalmemorystatus) 函数之前，无需设置此成员;函数设置它。  dwMemoryLoad  一个介于 0 和 100 之间的数字，指定正在使用的物理内存的近似百分比 (0 表示不使用内存，100 表示) 已满内存使用。  dwTotalPhys  实际物理内存量（以字节为单位）。  dwAvailPhys  当前可用的物理内存量（以字节为单位）。 这是可以立即重复使用的物理内存量，而无需先将其内容写入磁盘。 它是备用列表、可用列表和零列表的大小之和。  dwTotalPageFile  已提交的内存限制的当前大小（以字节为单位）。 这是物理内存加上页面文件的大小，减去少量开销。  dwAvailPageFile  当前进程可以提交的最大内存量（以字节为单位）。 此值应小于系统范围的可用提交。 若要计算此值，请调用 [GetPerformanceInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getperformanceinfo) 并从 **CommitLimit** 中减去 **CommitTotal** 的值。  dwTotalVirtual  调用进程的虚拟地址空间的用户模式部分的大小（以字节为单位）。 此值取决于进程类型、处理器类型和操作系统的配置。 例如，对于 x86 处理器上的大多数 32 位进程，此值约为 2 GB，对于在启用了 4 GT RAM 优化的系统上运行的可识别大型地址的 32 位进程，此值约为 3 GB。  dwAvailVirtual  当前位于调用进程的虚拟地址空间的用户模式部分中的未保留和未提交的内存量（以字节为单位）。 注解 **MEMORYSTATUS** 反映调用时内存的状态。 它还反映当时分页文件的大小。 操作系统可以将分页文件放大到管理员设置的最大大小。  在内存超过 4 GB 的计算机上， **MEMORYSTATUS** 结构可能会返回错误的信息，报告值 –1 以指示溢出。 如果应用程序存在此行为的风险，请使用 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex) 函数而不是 [GlobalMemoryStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalmemorystatus) 函数。 示例 有关示例，请参阅 [GlobalMemoryStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalmemorystatus) 函数。 要求 展开表   | **要求** | **值** | | --- | --- | | **最低受支持的客户端** | Windows XP [仅限桌面应用] | | **最低受支持的服务器** | Windows Server 2003 [仅限桌面应用] | | **标头** | winbase.h (包括 Windows.h) |  另请参阅 [GlobalMemoryStatus](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-globalmemorystatus)  [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex)  [内存性能信息](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa965225(v=vs.85)) |

## MEMORYSTATUSEX结构体的用法

|  |  |  |
| --- | --- | --- |
| 包含有关物理内存和虚拟内存（包括扩展内存）的当前状态的信息。 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex) 函数在此结构中存储信息。 语法 C++复制  typedef struct \_MEMORYSTATUSEX {  DWORD dwLength;  DWORD dwMemoryLoad;  DWORDLONG ullTotalPhys;  DWORDLONG ullAvailPhys;  DWORDLONG ullTotalPageFile;  DWORDLONG ullAvailPageFile;  DWORDLONG ullTotalVirtual;  DWORDLONG ullAvailVirtual;  DWORDLONG ullAvailExtendedVirtual;  } MEMORYSTATUSEX, \*LPMEMORYSTATUSEX; 成员 dwLength  结构大小（以字节为单位）。 在调用之前，必须设置此成员 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex).  dwMemoryLoad  一个介于 0 和 100 之间的数字，指定正在使用的物理内存的近似百分比 (0 表示不使用内存，100 表示) 已满内存使用。  ullTotalPhys  实际物理内存量（以字节为单位）。  ullAvailPhys  当前可用的物理内存量（以字节为单位）。 这是可以立即重复使用的物理内存量，而无需先将其内容写入磁盘。 它是备用列表、可用列表和零列表的大小之和。  ullTotalPageFile  系统或当前进程的当前已提交内存限制，以字节为单位，以较小者为准。 若要获取系统范围的已提交内存限制，请调用 [GetPerformanceInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getperformanceinfo)。  ullAvailPageFile  当前进程可以提交的最大内存量（以字节为单位）。 此值等于或小于系统范围的可用提交值。 若要计算系统范围的可用提交值，请调用 [GetPerformanceInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getperformanceinfo)，并从 **CommitLimit** 的值中减去 **CommitTotal** 的值。  ullTotalVirtual  调用进程的虚拟地址空间的用户模式部分的大小（以字节为单位）。 此值取决于进程类型、处理器类型和操作系统的配置。 例如，对于 x86 处理器上的大多数 32 位进程，此值约为 2 GB，对于在启用了 [4 GB 优化](https://learn.microsoft.com/zh-cn/windows/desktop/Memory/4-gigabyte-tuning) 的系统上运行的大地址感知的 32 位进程，此值约为 3 GB。  ullAvailVirtual  当前位于调用进程的虚拟地址空间的用户模式部分中的未保留和未提交的内存量（以字节为单位）。  ullAvailExtendedVirtual  保留。 此值始终为 0。 注解 **MEMORYSTATUSEX** 反映调用时内存的状态。 它还反映当时分页文件的大小。 操作系统可以将分页文件放大到管理员设置的最大大小。  返回的物理内存大小包括来自所有节点的内存。 示例 有关示例，请参阅 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex) 函数。 要求  |  |  | | --- | --- | | **标头** | sysinfoapi.h (包括 Windows.h) |  另请参阅 [GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex)  [内存性能信息](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa965225(v=vs.85)) |

## PROCESS\_MEMORY\_COUNTERS\_EX结构体的用法

|  |  |  |
| --- | --- | --- |
| 包含进程的扩展内存统计信息。 语法 C++复制  typedef struct \_PROCESS\_MEMORY\_COUNTERS\_EX {  DWORD cb;  DWORD PageFaultCount;  SIZE\_T PeakWorkingSetSize;  SIZE\_T WorkingSetSize;  SIZE\_T QuotaPeakPagedPoolUsage;  SIZE\_T QuotaPagedPoolUsage;  SIZE\_T QuotaPeakNonPagedPoolUsage;  SIZE\_T QuotaNonPagedPoolUsage;  SIZE\_T PagefileUsage;  SIZE\_T PeakPagefileUsage;  SIZE\_T PrivateUsage;  } PROCESS\_MEMORY\_COUNTERS\_EX; 成员 cb  结构大小（以字节为单位）。  PageFaultCount  页错误数。  PeakWorkingSetSize  峰值工作集大小（以字节为单位）。  WorkingSetSize  当前工作集大小（以字节为单位）。  QuotaPeakPagedPoolUsage  分页池使用量峰值（以字节为单位）。  QuotaPagedPoolUsage  当前分页池使用情况（以字节为单位）。  QuotaPeakNonPagedPoolUsage  非分页池使用量的峰值（以字节为单位）。  QuotaNonPagedPoolUsage  当前非分页池使用情况（以字节为单位）。  PagefileUsage  此过程的“提交费用”值（以字节为单位）。 提交费用是内存管理器为正在运行的进程提交的专用内存总量。  **Windows 7 和 Windows Server 2008 R2 及更早版本： PagefileUsage** 始终为零。 请改为选中 **PrivateUsage** 。  PeakPagefileUsage  此过程生存期内提交费用的峰值（以字节为单位）。  PrivateUsage  与 **PagefileUsage** 相同。 此过程的“提交费用”值（以字节为单位）。 提交费用是内存管理器为正在运行的进程提交的专用内存总量。 要求  |  |  | | --- | --- | | **标头** | psapi.h |  另请参阅 [GetProcessMemoryInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getprocessmemoryinfo)  [内存性能信息](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/aa965225(v=vs.85))  [进程内存使用情况信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/process-memory-usage-information)  [工作集信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/working-set-information) |

# 演练、

## 1新建一个常规空项目取名Lesson81-virtual-mem-status

|  |
| --- |
|  |

## 2.在资源文件夹里面添加一个对话框资源，使得修改一些标题和按钮

|  |
| --- |
|  |

## 3.我们调整对话框的布局如下

|  |
| --- |
|  |

## 4.然后我们新建一个cpp源文件，取名v-mem-status.cpp,先添加一些基本代码

|  |
| --- |
| #include<Windows.h>  #include<Psapi.h>  #include"resource.h"  #pragma comment(lib,"Psapi.lib")  INT\_PTR CALLBACK DlgProc(HWND,UINT,WPARAM,LPARAM);  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmd, int nshow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,DlgProc);  return 0;  }  INT\_PTR CALLBACK DlgProc(HWND hdlg,UINT msg,WPARAM wParam,LPARAM lParam)  {  BOOL ret = TRUE;  switch(msg)  {  case WM\_CLOSE:  EndDialog(hdlg,FALSE);  break;  case WM\_COMMAND:  switch(LOWORD(wParam))  {  case IDOK:  if(IDOK == MessageBox(hdlg,TEXT("退出程序"),TEXT("确认"),MB\_OKCANCEL))  {  EndDialog(hdlg,FALSE);  }  break;  }  break;  case WM\_INITDIALOG:  break;  default:  ret =FALSE;  }  return ret;  } |

### 运行程序，效果如下

|  |
| --- |
|  |

### 点击确定，退出程序,说明程序正常工作

## 5,我们可用设置定时器,让程序每每隔一个间隔就获取一下虚拟内存信息并且显示到编辑框上.我们可用在WM\_INITDIALOG消息处理代码里面设置定时器,然后再添加对WM\_TIMER消息的响应代码,需要先定义一个timer的ID

|  |
| --- |
|  |

## 然后在初始化对话框的信息处理代码里面设置定时器

|  |
| --- |
|  |

## 然后添加对WM\_TIMER消息的响应代码,头7给指标比较简单,我们可以直接设置,注意格式,格式不对有时候会显示负数.

|  |
| --- |
| #include<Windows.h>  #include<Psapi.h>  #include<strsafe.h>  #include"resource.h"  #pragma comment(lib,"Psapi.lib")  //定义timer 的ID  #define TIMER\_ID 1  INT\_PTR CALLBACK DlgProc(HWND,UINT,WPARAM,LPARAM);  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmd, int nshow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,DlgProc);  return 0;  }  INT\_PTR CALLBACK DlgProc(HWND hdlg,UINT msg,WPARAM wParam,LPARAM lParam)  {  char szBuf[64];  MEMORYSTATUSEX mse;  int factor = 1024\*1024\*1024;  BOOL ret = TRUE;  switch(msg)  {  case WM\_CLOSE:  EndDialog(hdlg,FALSE);  break;  case WM\_COMMAND:  switch(LOWORD(wParam))  {  case IDOK:  if(IDOK == MessageBox(hdlg,TEXT("退出程序"),TEXT("确认"),MB\_OKCANCEL))  {  EndDialog(hdlg,FALSE);  }  break;  }  break;  case WM\_INITDIALOG:  //设置定时器  SetTimer(hdlg,TIMER\_ID,1000,NULL);  break;  case WM\_TIMER:  //初始化变量  memset(szBuf,0,64);  mse.dwLength = sizeof(mse);//一定要初始化长度  GlobalMemoryStatusEx(&mse);  StringCchPrintf(szBuf,\_countof(szBuf),"%d %s",mse.dwMemoryLoad,"%");  SetDlgItemText(hdlg,IDC\_EDIT\_LOAD,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,\_countof(szBuf),"%d GB",mse.ullTotalPhys/factor);  SetDlgItemText(hdlg,IDC\_EDIT\_TOTAL\_PHIS,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,\_countof(szBuf),"%I64d KB",(\_\_int64)mse.ullAvailPhys/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_MEM\_AVAIL,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,\_countof(szBuf),"%d GB",(\_\_int64)mse.ullTotalPageFile/factor);  SetDlgItemText(hdlg,IDC\_EDIT\_TOT\_PAGE,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,\_countof(szBuf),"%I64d KB",(\_\_int64)mse.ullAvailPageFile/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_AVAI\_PAGE,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,\_countof(szBuf),"%I64d KB",(\_\_int64)mse.ullTotalVirtual/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_TOT\_V,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,\_countof(szBuf),"%I64d KB",(\_\_int64)mse.ullAvailVirtual/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_MEM\_V\_AVAIL,szBuf);  break;  default:  ret =FALSE;  }  return ret;  } |

### 运行程序,效果如下

|  |
| --- |
|  |

## 6.然后我们来处理剩下的2个指标,这两个指标需要使用GetProcessMemoryInfo函数来获取,为了方便编程,我们把定时器响应代码封装到一个叫做DisplayMemInfo函数里面,这个函数代码如下

|  |
| --- |
| void DisplayMemInfo(HWND hdlg,char\* szBuf,int len)  {  MEMORYSTATUSEX mse;  PROCESS\_MEMORY\_COUNTERS\_EX pmce;  memset(szBuf,0,64);  mse.dwLength = sizeof(mse);//一定要初始化长度  GlobalMemoryStatusEx(&mse);  StringCchPrintf(szBuf,len,"%d %s",mse.dwMemoryLoad,"%");  SetDlgItemText(hdlg,IDC\_EDIT\_LOAD,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%d GB",mse.ullTotalPhys/factor);  SetDlgItemText(hdlg,IDC\_EDIT\_TOTAL\_PHIS,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullAvailPhys/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_MEM\_AVAIL,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%d GB",(\_\_int64)mse.ullTotalPageFile/factor);  SetDlgItemText(hdlg,IDC\_EDIT\_TOT\_PAGE,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullAvailPageFile/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_AVAI\_PAGE,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullTotalVirtual/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_TOT\_V,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullAvailVirtual/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_MEM\_V\_AVAIL,szBuf);  //获取WorkingSet和PrivateBytes  GetProcessMemoryInfo(GetCurrentProcess(),(PROCESS\_MEMORY\_COUNTERS\*)&pmce,sizeof(pmce));  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)pmce.WorkingSetSize/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_WKSET,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)pmce.PrivateUsage/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_PRIVATE\_BYTE,szBuf);  } |

## 7.然后我们可以在对话框初始化的时候调用一次这个函数,然后在定时器消息响应函数里面也调用这个函数

|  |
| --- |
|  |
|  |

## 8.也可以在对话框初始化代码里设置定时器完成后马上发送定时器消息

|  |
| --- |
|  |

# 这一节的学习到此为止,完整代码如下

## v-mem-status.cpp

|  |
| --- |
| #include<Windows.h>  #include<Psapi.h>  #include<strsafe.h>  #include"resource.h"  #pragma comment(lib,"Psapi.lib")  //定义timer 的ID  #define TIMER\_ID 1  int factor = 1024\*1024\*1024;  void DisplayMemInfo(HWND hdlg,char\* szBuf,int len)  {  MEMORYSTATUSEX mse;  PROCESS\_MEMORY\_COUNTERS\_EX pmce;  memset(szBuf,0,64);  mse.dwLength = sizeof(mse);//一定要初始化长度  GlobalMemoryStatusEx(&mse);  StringCchPrintf(szBuf,len,"%d %s",mse.dwMemoryLoad,"%");  SetDlgItemText(hdlg,IDC\_EDIT\_LOAD,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%d GB",mse.ullTotalPhys/factor);  SetDlgItemText(hdlg,IDC\_EDIT\_TOTAL\_PHIS,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullAvailPhys/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_MEM\_AVAIL,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%d GB",(\_\_int64)mse.ullTotalPageFile/factor);  SetDlgItemText(hdlg,IDC\_EDIT\_TOT\_PAGE,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullAvailPageFile/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_AVAI\_PAGE,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullTotalVirtual/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_TOT\_V,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)mse.ullAvailVirtual/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_MEM\_V\_AVAIL,szBuf);  //获取WorkingSet和PrivateBytes  GetProcessMemoryInfo(GetCurrentProcess(),(PROCESS\_MEMORY\_COUNTERS\*)&pmce,sizeof(pmce));  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)pmce.WorkingSetSize/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_WKSET,szBuf);  memset(szBuf,0,64);  StringCchPrintf(szBuf,len,"%I64d KB",(\_\_int64)pmce.PrivateUsage/1024);  SetDlgItemText(hdlg,IDC\_EDIT\_PRIVATE\_BYTE,szBuf);  }  INT\_PTR CALLBACK DlgProc(HWND,UINT,WPARAM,LPARAM);  int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE prev,LPTSTR lpCmd, int nshow)  {  DialogBox(hInstance,MAKEINTRESOURCE(IDD\_DIALOG1),NULL,DlgProc);  return 0;  }  INT\_PTR CALLBACK DlgProc(HWND hdlg,UINT msg,WPARAM wParam,LPARAM lParam)  {  //变量定义  char szBuf[64];  BOOL ret = TRUE;  switch(msg)  {  case WM\_CLOSE:  EndDialog(hdlg,FALSE);  break;  case WM\_COMMAND:  switch(LOWORD(wParam))  {  case IDOK:  if(IDOK == MessageBox(hdlg,TEXT("退出程序"),TEXT("确认"),MB\_OKCANCEL))  {  EndDialog(hdlg,FALSE);  }  break;  }  break;  case WM\_INITDIALOG:  //设置定时器  //DisplayMemInfo(hdlg,szBuf,\_countof(szBuf));//先调用这个函数显示一次信息,编码一开始什么都没有显示  SetTimer(hdlg,TIMER\_ID,1000,NULL);//设置定时器  SendMessage(hdlg,WM\_TIMER,TIMER\_ID,0);  break;  case WM\_TIMER:  DisplayMemInfo(hdlg,szBuf,\_countof(szBuf));  break;  default:  ret =FALSE;  }  return ret;  } |