# 枚举进程方法1，参考：[用系统快照枚举所有进程](Windows%20Api学习指南学习笔记062-枚举进程1.docx)

# 枚举进程方法2，参考：[用psapi函数枚举所有进程](Windows%20Api学习指南学习笔记063-枚举进程2.docx)

# 枚举进程方法3，参考：[用wtsapi函数来枚举所有进程](Windows%20Api学习指南学习笔记064-枚举进程3.docx)

# 这一节我们来学习枚举进程方法4

|  |
| --- |
|  |

# API语法说明

## ZwQuerySystemInformation函数的语法，win8之后不可用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [**ZwQuerySystemInformation** 自Windows 8起不再可用。 请改用本主题中列出的备用函数。]  检索指定的系统信息。 语法 C++复制  NTSTATUS WINAPI ZwQuerySystemInformation(  \_In\_ SYSTEM\_INFORMATION\_CLASS SystemInformationClass,  \_Inout\_ PVOID SystemInformation,  \_In\_ ULONG SystemInformationLength,  \_Out\_opt\_ PULONG ReturnLength  ); 参数 SystemInformationClass [in]  要检索的系统信息的类型。 此参数可以是 **SYSTEM\_INFORMATION\_CLASS** 枚举类型的下列值之一。  **SystemBasicInformation**  **系统中SYSTEM\_BASIC\_INFORMATION**结构中的处理器数。 请改用 [**GetSystemInfo**](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。  **SystemPerformanceInformation**  一种不透明的 **SYSTEM\_PERFORMANCE\_INFORMATION** 结构，可用于为随机数生成器生成不可预知的种子。 请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数。  **SystemTimeOfDayInformation**  一种不透明的 **SYSTEM\_TIMEOFDAY\_INFORMATION** 结构，可用于为随机数生成器生成不可预知的种子。 请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数。  **SystemProcessInformation**  **一个SYSTEM\_PROCESS\_INFORMATION**结构的数组，每个结构对应系统中运行的每个进程。  这些结构包含有关每个进程的资源使用情况的信息，包括进程使用的句柄数、页面文件的峰值使用情况以及进程已分配的内存页数。  **SystemProcessorPerformanceInformation**  **SYSTEM\_PROCESSOR\_PERFORMANCE\_INFORMATION**结构的数组，系统中安装的每个处理器对应一个。  **SystemInterruptInformation**  一种不透明的 **SYSTEM\_INTERRUPT\_INFORMATION** 结构，可用于为随机数生成器生成不可预知的种子。 请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数。  **SystemExceptionInformation**  一种不透明的 **SYSTEM\_EXCEPTION\_INFORMATION** 结构，可用于为随机数生成器生成不可预知的种子。 请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数。  **SystemRegistryQuotaInformation**  **SYSTEM\_REGISTRY\_QUOTA\_INFORMATION** 结构。  **SystemLookasideInformation**  一种不透明的 **SYSTEM\_LOOKASIDE\_INFORMATION** 结构，可用于为随机数生成器生成不可预知的种子。 请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数。  SystemInformation [in， out]  指向接收请求信息的缓冲区的指针。 此信息的大小和结构因 SystemInformationClass 参数的值而异，如下表所示。  **SYSTEM\_BASIC\_INFORMATION**  当 SystemInformationClass 参数为 **SystemBasicInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳具有以下布局 **的单个SYSTEM\_BASIC\_INFORMATION** 结构：  syntax复制  typedef struct \_SYSTEM\_BASIC\_INFORMATION {  BYTE Reserved1[24];  PVOID Reserved2[4];  CCHAR NumberOfProcessors;  } SYSTEM\_BASIC\_INFORMATION;  **NumberOfProcessors** 成员包含系统中存在的处理器数。 请改用 [**GetSystemInfo**](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 来检索此信息。  结构的其他成员保留供操作系统内部使用。  **SYSTEM\_PERFORMANCE\_INFORMATION**  当 SystemInformationClass 参数为 **SystemPerformanceInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳不透明的 **SYSTEM\_PERFORMANCE\_INFORMATION** 结构，以便在为随机数生成器生成不可预知的种子时使用。 为此，结构具有以下布局：  syntax复制  typedef struct \_SYSTEM\_PERFORMANCE\_INFORMATION {  BYTE Reserved1[312];  } SYSTEM\_PERFORMANCE\_INFORMATION;  结构的各个成员保留供操作系统内部使用。  请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数生成加密随机数据。  **SYSTEM\_TIMEOFDAY\_INFORMATION**  当 SystemInformationClass 参数为 **SystemTimeOfDayInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳不透明的 **SYSTEM\_TIMEOFDAY\_INFORMATION** 结构，以便在为随机数生成器生成不可预知的种子时使用。 为此，结构具有以下布局：  syntax复制  typedef struct \_SYSTEM\_TIMEOFDAY\_INFORMATION {  BYTE Reserved1[48];  } SYSTEM\_TIMEOFDAY\_INFORMATION;  结构的各个成员保留供操作系统内部使用。  请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数生成加密随机数据。  **SYSTEM\_PROCESS\_INFORMATION**  当 SystemInformationClass 参数为 **SystemProcessInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳一个数组，该数组包含与系统中运行的进程数一样多 **的SYSTEM\_PROCESS\_INFORMATION** 结构。 每个结构都具有以下布局：  syntax复制  typedef struct \_SYSTEM\_PROCESS\_INFORMATION {  ULONG NextEntryOffset;  ULONG NumberOfThreads;  BYTE Reserved1[48];  PVOID Reserved2[3];  HANDLE UniqueProcessId;  PVOID Reserved3;  ULONG HandleCount;  BYTE Reserved4[4];  PVOID Reserved5[11];  SIZE\_T PeakPagefileUsage;  SIZE\_T PrivatePageCount;  LARGE\_INTEGER Reserved6[6];  } SYSTEM\_PROCESS\_INFORMATION;  **NumberOfThreads** 成员包含进程中当前正在运行的线程总数。  **HandleCount** 成员包含有问题的进程正在使用的句柄总数;改用 [**GetProcessHandleCount**](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getprocesshandlecount) 检索此信息。  **PeakPagefileUsage** 成员包含进程使用的页面文件存储的最大字节数，**PrivatePageCount** 成员包含为使用此进程分配的内存页数。  还可以使用 [**GetProcessMemoryInfo**](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getprocessmemoryinfo) 函数或 [**Win32\_Process**](https://learn.microsoft.com/zh-cn/windows/desktop/CIMWin32Prov/win32-process) 类检索此信息。  结构的其他成员保留供操作系统内部使用。  **SYSTEM\_PROCESSOR\_PERFORMANCE\_INFORMATION**  当 SystemInformationClass 参数为 **SystemProcessorPerformanceInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳一个数组，该数组包含与系统中安装的处理器 (CPU) 一 **样多的** SYSTEM\_PROCESS\_INFORMATION结构。 每个结构都具有以下布局：  syntax复制  typedef struct  \_SYSTEM\_PROCESSOR\_PERFORMANCE\_INFORMATION {  LARGE\_INTEGER IdleTime;  LARGE\_INTEGER KernelTime;  LARGE\_INTEGER UserTime;  LARGE\_INTEGER Reserved1[2];  ULONG Reserved2;  } SYSTEM\_PROCESSOR\_PERFORMANCE\_INFORMATION;  **IdleTime** 成员包含系统空闲的时间量，以纳秒的 1/100 秒为单位。  **KernelTime** 成员包含系统在内核模式下执行所用的时间， (包括所有进程中的所有线程、) 的所有处理器上的所有线程，以纳秒的 1/100 秒为单位。  **UserTime** 成员包含系统在用户模式下执行所花费的时间量， (包括所有进程中的所有线程，) 所有处理器上的所有线程，以纳秒的 1/100 秒为单位。  请改用 [**GetSystemTimes**](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-getsystemtimes) 来检索此信息。  **SYSTEM\_INTERRUPT\_INFORMATION**  当 SystemInformationClass 参数为 **SystemInterruptInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳一个数组，该数组包含与系统上安装的处理器 (CPU) 一样多的不透明 **SYSTEM\_INTERRUPT\_INFORMATION** 结构。 每个结构或整个数组都可用于为随机数生成器生成不可预知的种子。 为此，结构具有以下布局：  syntax复制  typedef struct \_SYSTEM\_INTERRUPT\_INFORMATION {  BYTE Reserved1[24];  } SYSTEM\_INTERRUPT\_INFORMATION;  结构的各个成员保留供操作系统内部使用。  请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数生成加密随机数据。  **SYSTEM\_EXCEPTION\_INFORMATION**  当 SystemInformationClass 参数为 **SystemExceptionInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳不透明的 **SYSTEM\_EXCEPTION\_INFORMATION** 结构，以便在为随机数生成器生成不可预知的种子时使用。 为此，结构具有以下布局：  syntax复制  typedef struct \_SYSTEM\_EXCEPTION\_INFORMATION {  BYTE Reserved1[16];  } SYSTEM\_EXCEPTION\_INFORMATION;  结构的各个成员保留供操作系统内部使用。  请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数生成加密随机数据。  **SYSTEM\_REGISTRY\_QUOTA\_INFORMATION**  当 SystemInformationClass 参数为 **SystemRegistryQuotaInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳具有以下布局 **的单个SYSTEM\_REGISTRY\_QUOTA\_INFORMATION** 结构：  syntax复制  typedef struct \_SYSTEM\_REGISTRY\_QUOTA\_INFORMATION {  ULONG RegistryQuotaAllowed;  ULONG RegistryQuotaUsed;  PVOID Reserved1;  } SYSTEM\_REGISTRY\_QUOTA\_INFORMATION;  **RegistryQuotaAllowed** 成员包含注册表在此系统上可实现的最大大小（以字节为单位）。  **RegistryQuotaUsed** 成员包含注册表的当前大小（以字节为单位）。  请改用 [**GetSystemRegistryQuota**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winbase/nf-winbase-getsystemregistryquota) 来检索此信息。  结构的其他成员保留供操作系统内部使用。  **SYSTEM\_LOOKASIDE\_INFORMATION**  当 SystemInformationClass 参数为 **SystemLookasideInformation** 时， SystemInformation 参数指向的缓冲区应足够大，以容纳不透明的 **SYSTEM\_LOOKASIDE\_INFORMATION** 结构，以便在为随机数生成器生成不可预知的种子时使用。 为此，结构具有以下布局：  syntax复制  typedef struct \_SYSTEM\_LOOKASIDE\_INFORMATION {  BYTE Reserved1[32];  } SYSTEM\_LOOKASIDE\_INFORMATION;  结构的各个成员保留供操作系统内部使用。  请改用 [**CryptGenRandom**](https://learn.microsoft.com/zh-cn/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom) 函数生成加密随机数据。  SystemInformationLength [in]  SystemInformation 参数指向的缓冲区的大小（以字节为单位）。  ReturnLength [out， optional]  指向函数写入所请求信息的实际大小的位置的可选指针。 如果该大小小于或等于 SystemInformationLength 参数，该函数会将信息复制到 SystemInformation 缓冲区中;否则，它将返回 NTSTATUS 错误代码，并在 ReturnLength 中返回接收请求的信息所需的缓冲区大小。 返回值 返回 NTSTATUS 成功或错误代码。  NTSTATUS 错误代码的形式和重要性在 DDK 中提供的 Ntstatus.h 头文件中列出，并在 DDK 文档中进行介绍。 备注 **ZwQuerySystemInformation** 函数及其返回的结构在操作系统内部，可能会从一个版本的 Windows 更改为另一个版本。 为了保持应用程序的兼容性，最好改用前面提到的备用函数。  如果使用 **ZwQuerySystemInformation**，请通过 [运行时动态链接](https://learn.microsoft.com/zh-cn/windows/desktop/Dlls/using-run-time-dynamic-linking)访问函数。 这样，当函数已更改或从操作系统中删除时，代码就有机会正常响应。 但是，可能无法检测到签名更改。  此函数没有关联的导入库。 必须使用 [**LoadLibrary**](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibrarya) 和 [**GetProcAddress**](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-getprocaddress) 函数动态链接到Ntdll.dll。 要求 展开表   | **要求** | **值** | | --- | --- | | DLL | Ntdll.dll |  另请参阅 [**GetSystemInfo**](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [**GetProcessHandleCount**](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getprocesshandlecount)  [**GetProcessMemoryInfo**](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getprocessmemoryinfo)  [**GetSystemTimes**](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-getsystemtimes)  [**GetSystemRegistryQuota**](https://learn.microsoft.com/zh-cn/windows/desktop/api/Winbase/nf-winbase-getsystemregistryquota) |

# 这一节的内容自己作为选学，因为在win10下面没有作用，实例参考：

|  |
| --- |
| #include <ntddk.h>    #define SystemProcessesAndThreadsInformation 5    typedef struct \_SYSTEM\_PROCESSES  {      ULONG NextEntryDelta;      ULONG ThreadCount;      ULONG Reserved[6];      LARGE\_INTEGER CreateTime;      LARGE\_INTEGER UserTime;      LARGE\_INTEGER KernelTime;      UNICODE\_STRING ProcessName;      KPRIORITY BasePriority;      ULONG ProcessId;      ULONG InheritedFromProcessId;      ULONG HandleCount;      ULONG Reserved2[2];      VM\_COUNTERS VmCounters;      IO\_COUNTERS IoCounters;  } \_SYSTEM\_PROCESSES, \*PSYSTEM\_PROCESSES;    NTSTATUS EnumSystemProcess( );      NTSYSAPI  NTSTATUS  NTAPI ZwQuerySystemInformation(  IN ULONG SystemInformationClass,  IN OUT PVOID SystemInformation,  IN ULONG SystemInformationLength,  OUT PULONG ReturnLength  );    NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath);    NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)  {      UNREFERENCED\_PARAMETER(DriverObject);      UNREFERENCED\_PARAMETER(RegistryPath);      NTSTATUS status = STATUS\_SUCCESS;      status  = EnumSystemProcess( );      return status;  }    NTSTATUS EnumSystemProcess( )  {      NTSTATUS status = STATUS\_UNSUCCESSFUL;      \*pRet = FALSE;        PSYSTEM\_PROCESSES pProcessInfo = NULL;      PSYSTEM\_PROCESSES pTemp = NULL;//这个留作以后释放指针的时候用。      ULONG ulNeededSize;      ULONG ulNextOffset;        if (NULL == pProcess)      {          return status;      }          //第一次使用肯定是缓冲区不够，不过本人在极少数的情况下第二次也会出现不够，所以用while循环      status = ZwQuerySystemInformation(SystemProcessesAndThreadsInformation , pProcessInfo, 0, &ulNeededSize);      while (STATUS\_INFO\_LENGTH\_MISMATCH == status)      {          pProcessInfo = ExAllocatePoolWithTag(NonPagedPool, ulNeededSize, '1aes');          pTemp = pProcessInfo;          if (NULL == pProcessInfo)          {              KdPrint(("[allocatePoolWithTag] failed"));              return status;          }          status = ZwQuerySystemInformation(SystemProcessesAndThreadsInformation , pProcessInfo, ulNeededSize, &ulNeededSize);      }      if (NT\_SUCCESS(status))      {          KdPrint(("[ZwQuerySystemInformation]success bufferSize:%x", ulNeededSize));      }          else          {                 KdPrint(("[error]:++++%d", status));                 return status;          }        do      {          KdPrint(("[imageName Buffer]:%08x", pProcessInfo->ProcessName.Buffer));            if (MmIsAddressValid(pProcessInfo->ProcessName.Buffer) && NULL != pProcessInfo)          {              KdPrint(("[ProcessID]:%d , [imageName]:%ws", pProcessInfo->ProcessId, pProcessInfo->ProcessName.Buffer));          }            ulNextOffset = pProcessInfo->NextEntryDelta;          pProcessInfo = (PSYSTEM\_PROCESSES)((PUCHAR)pProcessInfo + pProcessInfo->NextEntryDelta);        } while (ulNextOffset != 0);        ExFreePoolWithTag(pTemp, '1aes');        return status;  } |

## 效果

|  |
| --- |
| https://img2018.cnblogs.com/blog/933913/201810/933913-20181009094701545-1912756788.png |

## 案例网址：[内核下枚举进程 （二）ZwQuerySystemInformation - priarieNew - 博客园 (cnblogs.com)](https://www.cnblogs.com/priarieNew/p/9758621.html)