# 学习大纲

|  |
| --- |
|  |

# 这一节我们学习通过psapi.dll中的函数来进行枚举

## EnumProcesses函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索系统中每个进程对象的进程标识符。 语法 C++  BOOL EnumProcesses(  [out] DWORD \*lpidProcess,  [in] DWORD cb,  [out] LPDWORD lpcbNeeded  ); parameters [out] lpidProcess  指向接收进程标识符列表的数组的指针。  [in] cb  *pProcessIds* 数组的大小（以字节为单位）。  [out] lpcbNeeded  *pProcessIds* 数组中返回的字节数。 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 最好使用大型数组，因为很难预测在调用 **EnumProcesses** 时将存在多少个进程。  若要确定枚举的进程数，请将 l (DWORD) 的大小除以 *l (Needed* 值。 当缓冲区太小而无法存储所有进程标识符时，不会给出任何指示。 因此，如果 *lcbNeeded* 等于 *cb*，请考虑使用较大的数组重试调用。  若要获取刚刚获取其标识符的进程的进程句柄，请调用 [OpenProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openprocess) 函数。  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函数的名称以及程序必须加载的库。  如果 PSAPI\_VERSION 为 2 或更大，则此函数在 Psapi.h 中定义为 **K32EnumProcesses** ，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 **EnumProcesses** ，并在 Psapi.lib 中导出，Psapi.dll 为调用 **K32EnumProcesses 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 **EnumProcesses**。 为了确保符号的正确解析，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 [枚举所有进程](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/enumerating-all-processes) 或 [枚举进程的所有模块](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/enumerating-all-modules-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PSAPI\_VERSION=1) ，则为 Psapi.dll (;在 Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上 Psapi.dll |  请参阅 [CreateToolhelp32Snapshot](https://learn.microsoft.com/zh-cn/windows/desktop/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot)  [OpenProcess](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openprocess)  [PSAPI 函数](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/psapi-functions)  [进程信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/process-information) |

## EnumProcessModules函数的语法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索指定进程中每个模块的句柄。  若要控制 64 位应用程序是枚举 32 位模块、64 位模块还是同时枚举两种类型的模块，请使用 [EnumProcessModulesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-enumprocessmodulesex) 函数。 语法 C++复制  BOOL EnumProcessModules(  [in] HANDLE hProcess,  [out] HMODULE \*lphModule,// 接收模块句柄列表的数组。  [in] DWORD cb,  [out] LPDWORD lpcbNeeded  ); 参数 [in] hProcess  进程的句柄。  [out] lphModule  接收模块句柄列表的数组。  [in] cb  *lphModule* 数组的大小（以字节为单位）。  [out] lpcbNeeded  在 *lphModule* 数组中存储所有模块句柄所需的字节数。 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 **EnumProcessModules** 函数主要用于调试器和必须从另一个进程提取模块信息的类似应用程序。 如果目标进程中的模块列表已损坏或尚未初始化，或者模块列表在函数调用期间因加载或卸载 DLL 而更改， **则 EnumProcessModules** 可能会失败或返回不正确的信息。  最好指定大型 **HMODULE** 值数组，因为调用 **EnumProcessModules** 时很难预测进程中有多少个模块。 若要确定 *lphModule* 数组是否太小，无法容纳进程的所有模块句柄，请将 *lcbNeeded* 中返回的值与 *cb* 中指定的值进行比较。 如果 *lcbNeeded* 大于 *cb*，请增大数组的大小，然后再次调用 **EnumProcessModules** 。  若要确定通过调用 **EnumProcessModules** 枚举了多少个模块，请将 *lafterNeeded* 参数中的结果值除以 sizeof(HMODULE)。  **EnumProcessModules** 函数不会检索使用 **LOAD\_LIBRARY\_AS\_DATAFILE** 或类似标志加载的模块的句柄。 有关详细信息，请参阅 [LoadLibraryEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibraryexa)。  不要对此函数返回的任何句柄调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle) 。 信息来自快照，因此没有要释放的资源。  如果从 WOW64 上运行的 32 位应用程序调用此函数，则它只能枚举 32 位进程的模块。 如果进程是 64 位进程，则此函数将失败，最后一个错误代码 **ERROR\_PARTIAL\_COPY** (299) 。  若要获取指定进程以及这些进程使用的堆、模块和线程快照，请使用 [CreateToolhelp32Snapshot](https://learn.microsoft.com/zh-cn/windows/desktop/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot) 函数。  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 **K32EnumProcessModules** ，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 **EnumProcessModules** ，并在 Psapi.lib 中导出，Psapi.dll 为调用 **K32EnumProcessModules 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 **EnumProcessModules**。 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 [枚举所有进程](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/enumerating-all-processes) 或 [枚举进程的所有模块](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/enumerating-all-modules-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 [CreateToolhelp32Snapshot](https://learn.microsoft.com/zh-cn/windows/desktop/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot)  [EnumProcessModulesEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-enumprocessmodulesex)  [EnumProcesses](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-enumprocesses)  [模块信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/module-information)  [PSAPI 函数](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/psapi-functions) |

## GetModuleFileNameEx函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索包含指定模块的文件的完全限定路径。 语法 C++复制  DWORD GetModuleFileNameExW(  [in] HANDLE hProcess,  [in, optional] HMODULE hModule,  [out] LPWSTR lpFilename,  [in] DWORD nSize  ); 参数 [in] hProcess  包含模块的进程句柄。  句柄必须具有 **PROCESS\_QUERY\_INFORMATION** 和 **PROCESS\_VM\_READ** 访问权限。 有关详细信息，请参阅 [进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/process-security-and-access-rights)。  **Windows 10及更高版本，Windows Server 2016及更高版本**：如果 *hModule* 参数为 NULL，则句柄只需要**PROCESS\_QUERY\_LIMITED\_INFORMATION**访问权限。  **GetModuleFileNameEx** 函数不会检索使用 **LOAD\_LIBRARY\_AS\_DATAFILE** 标志加载的模块的路径。 有关详细信息，请参阅 [LoadLibraryEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibraryexw)。  [in, optional] hModule  模块的句柄。 如果此参数为 NULL， **则 GetModuleFileNameEx** 返回 *hProcess* 中指定的进程的可执行文件的路径。  [out] lpFilename  指向接收模块的完全限定路径的缓冲区的指针。 如果文件名的大小大于 *nSize* 参数的值，则函数会成功，但文件名将被截断并用 null 结尾。  [in] nSize  *lpFilename* 缓冲区的大小（以字符为单位）。 返回值 如果函数成功，则返回值指定复制到缓冲区的字符串的长度。  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 **GetModuleFileNameEx** 函数主要用于调试器和类似应用程序，这些应用程序必须从另一个进程中提取模块信息。 如果目标进程中的模块列表已损坏或尚未初始化，或者模块列表在函数调用期间由于加载或卸载 DLL 而发生更改， **则 GetModuleFileNameEx** 可能会失败或返回不正确的信息。  若要检索当前进程中模块的名称，请使用 [GetModuleFileName](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-getmodulefilenamew) 函数。 这比使用当前进程的句柄调用 **GetModuleFileNameEx** 更高效、更可靠。  若要检索远程进程的 main 可执行模块的名称，请使用 [GetProcessImageFileName](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getprocessimagefilenamew) 或 [QueryFullProcessImageName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-queryfullprocessimagenamew) 函数。 这比使用 NULL 模块句柄调用 **GetModuleFileNameEx** 函数更高效、更可靠。  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函数的名称以及程序必须加载的库。  如果 **PSAPI\_VERSION** 为 2 或更大，则此函数在 Psapi.h 中定义为 **K32GetModuleFileNameEx** ，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 **GetModuleFileNameEx** ，并在 Psapi.lib 中导出，Psapi.dll 为调用 **K32GetModuleFileNameEx** 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 **GetModuleFileNameEx**。 为了确保正确解析符号，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 [枚举进程的所有模块](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/enumerating-all-modules-for-a-process)。  **备注**  psapi.h 标头将 GetModuleFileNameEx 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名的使用与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 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PSAPI\_VERSION=1) ，则为 Psapi.dll (;在 Windows Server 2008、Windows Vista、Windows Server 2003 和 Windows XP 上 Psapi.dll |  另请参阅 [EnumProcesses](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-enumprocesses)  [GetModuleBaseName](https://learn.microsoft.com/zh-cn/windows/desktop/api/psapi/nf-psapi-getmodulebasenamew)  [GetModuleFileName](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-getmodulefilenamew)  [GetModuleHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-getmodulehandlew)  [LoadLibrary](https://learn.microsoft.com/zh-cn/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibrarya)  [模块信息](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/module-information)  [PSAPI 函数](https://learn.microsoft.com/zh-cn/windows/desktop/psapi/psapi-functions) |

# 注意：使用EnumProcesses函数需要在源文件的包含指令下面添加一行

## #pragma comment(lib,"Psapi.lib") 否则会发生链接错误

## 当然也可以把Psapi.lib添加到项目属性-》连接器-》输入-》附加依赖库里面不过上面的写法比较简单

# 参考：提升进程权限的函数

|  |
| --- |
|  |

# 提示权限

|  |
| --- |
|  |

# 扩展函数

## OpenProcessToken函数的语法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **OpenProcessToken** 函数打开与进程关联的[访问令牌](https://learn.microsoft.com/zh-cn/windows/desktop/SecGloss/a-gly)。 语法 C++复制  BOOL OpenProcessToken(  [in] HANDLE ProcessHandle,  [in] DWORD DesiredAccess,  [out] PHANDLE TokenHandle  ); 参数 [in] ProcessHandle  打开访问令牌的进程句柄。 进程必须具有PROCESS\_QUERY\_LIMITED\_INFORMATION访问权限。 有关详细信息 [，请参阅进程安全性和访问权限](https://learn.microsoft.com/zh-cn/windows/win32/procthread/process-security-and-access-rights) 。  [in] DesiredAccess  指定 [访问掩码，该掩码](https://learn.microsoft.com/zh-cn/windows/desktop/SecGloss/a-gly) 指定访问令牌的请求访问类型。 这些请求的访问类型与令牌的 DACL) ([自由访问控制列表](https://learn.microsoft.com/zh-cn/windows/desktop/SecGloss/d-gly) 进行比较，以确定授予或拒绝了哪些访问。  有关访问令牌的访问权限列表，请参阅 [Access-Token 对象的访问权限](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/access-rights-for-access-token-objects)。  [out] TokenHandle  指向句柄的指针，该句柄标识函数返回时新打开的访问令牌。 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 若要从非提升进程内获取提升进程的句柄，必须从同一帐户启动这两个进程。  如果正在检查的进程是由其他帐户启动的，则检查过程需要启用SE\_DEBUG\_NAME特权。 有关详细信息 [，请参阅权限常量 (授权)](https://learn.microsoft.com/zh-cn/windows/win32/secauthz/privilege-constants)。  若要关闭通过 *TokenHandle* 参数返回的访问令牌句柄，请调用 [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)。 要求  |  |  | | --- | --- | | **标头** | processthreadsapi.h (包括 Windows.h) | | **Library** | Advapi32.lib | | **DLL** | Advapi32.dll |  另请参阅 [访问控制](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/access-control)  [基本访问控制函数](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-functions)  [AccessCheck](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-accesscheck)  [AdjustTokenGroups](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-adjusttokengroups)  [AdjustTokenPrivileges](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-adjusttokenprivileges)  [CloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/handleapi/nf-handleapi-closehandle)  [GetCurrentProcessToken](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getcurrentprocesstoken)  [GetCurrentThreadEffectiveToken](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getcurrentthreadeffectivetoken)  [GetCurrentThreadToken](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getcurrentthreadtoken)  [GetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-gettokeninformation)  [OpenThreadToken](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openthreadtoken)  [SetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-settokeninformation) |

## LookupPrivilegeValue函数的语法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **LookupPrivilegeValue** 函数检索[本地唯一标识符](https://learn.microsoft.com/zh-cn/windows/desktop/SecGloss/l-gly) (LUID) 指定系统上用于本地表示指定特权名称。 语法 C++复制  BOOL LookupPrivilegeValueA(  [in, optional] LPCSTR lpSystemName,  [in] LPCSTR lpName,  [out] PLUID lpLuid  ); 参数 [in, optional] lpSystemName  指向以 null 结尾的字符串的指针，该字符串指定检索特权名称的系统的名称。 如果指定了空字符串，则该函数将尝试在本地系统上查找特权名称。  [in] lpName  指向以 null 结尾的字符串的指针，该字符串指定权限的名称，如 Winnt.h 头文件中定义的那样。 例如，此参数可以指定常量、SE\_SECURITY\_NAME或其对应的字符串“SeSecurityPrivilege”。  [out] lpLuid  指向一个变量的指针，该变量接收 *LUID，lpSystemName* 参数指定的系统上的权限是已知的。 返回值 如果函数成功，则函数返回非零值。  如果函数失败，则返回零。 要获得更多的错误信息，请调用 GetLastError。 注解 **LookupPrivilegeValue** 函数仅支持 Winnt.h 的 Defined Privileges 节中指定的特权。 有关值的列表，请参阅 [特权常量](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/privilege-constants)。 示例 有关使用此函数的示例，请参阅 [启用和禁用特权](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/enabling-and-disabling-privileges-in-c--)。  **备注**  winbase.h 标头将 LookupPrivilegeValue 定义为别名，该别名根据 UNICODE 预处理器常量的定义自动选择此函数的 ANSI 或 Unicode 版本。 将非特定编码别名与非非特定编码的代码混合使用可能会导致不匹配，从而导致编译或运行时错误。 有关详细信息，请参阅 [**函数原型的约定**](https://learn.microsoft.com/zh-cn/windows/win32/intl/conventions-for-function-prototypes)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows.h) | | **Library** | Advapi32.lib | | **DLL** | Advapi32.dll |  另请参阅 [访问控制](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/access-control)  [基本访问控制函数](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-functions)  [LookupPrivilegeDisplayName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-lookupprivilegedisplaynamea)  [LookupPrivilegeName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-lookupprivilegenamea) |

## AdjustTokenPrivileges函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AdjustTokenPrivileges** 函数启用或禁用指定[访问令牌](https://learn.microsoft.com/zh-cn/windows/desktop/SecGloss/a-gly)中的特权。 启用或禁用访问令牌中的特权需要TOKEN\_ADJUST\_PRIVILEGES访问权限。 语法 C++复制  BOOL AdjustTokenPrivileges(  [in] HANDLE TokenHandle,  [in] BOOL DisableAllPrivileges,  [in, optional] PTOKEN\_PRIVILEGES NewState,  [in] DWORD BufferLength,  [out, optional] PTOKEN\_PRIVILEGES PreviousState,  [out, optional] PDWORD ReturnLength  ); 参数 [in] TokenHandle  包含要修改的权限的访问令牌的句柄。 句柄必须具有TOKEN\_ADJUST\_PRIVILEGES令牌的访问权限。 如果 *PreviousState* 参数不为 **NULL**，则句柄还必须具有TOKEN\_QUERY访问权限。  [in] DisableAllPrivileges  指定函数是否禁用令牌的所有特权。 如果此值为 **TRUE**，则函数将禁用所有特权并忽略 *NewState* 参数。 如果为 **FALSE**，则函数根据 *NewState* 参数指向的信息修改权限。  [in, optional] NewState  指向 [TOKEN\_PRIVILEGES](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_privileges) 结构的指针，该结构指定特权及其属性的数组。 如果 *DisableAllPrivileges* 参数为 **FALSE**， **则 AdjustTokenPrivileges** 函数将启用、禁用或删除令牌的这些特权。 下表描述了 **AdjustTokenPrivileges** 函数基于特权属性执行的操作。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SE\_PRIVILEGE\_ENABLED** | 函数启用 特权。 | | **SE\_PRIVILEGE\_REMOVED** | 特权将从令牌中的特权列表中删除。 列表中的其他特权将重新排序，以保持连续。  SE\_PRIVILEGE\_REMOVED取代SE\_PRIVILEGE\_ENABLED。  由于特权已从令牌中删除，因此尝试重新启用该特权会导致警告ERROR\_NOT\_ALL\_ASSIGNED好像该特权从未存在过一样。  尝试删除令牌中不存在的特权会导致返回ERROR\_NOT\_ALL\_ASSIGNED。  对已删除特权的特权检查会导致STATUS\_PRIVILEGE\_NOT\_HELD。 审核检查失败的权限正常发生。  删除特权是不可逆的，因此在调用 **AdjustTokenPrivileges** 后，删除的特权的名称不包含在 *PreviousState* 参数中。  **Windows XP SP1：**函数无法删除特权。 不支持此值。 | | 无 | 函数禁用特权。 |     如果 *DisableAllPrivileges* 为 **TRUE**，则函数将忽略此参数。  [in] BufferLength  指定 *PreviousState* 参数指向的缓冲区的大小（以字节为单位）。 如果 *PreviousState* 参数为 NULL，则此参数可以为 **零**。  [out, optional] PreviousState  指向缓冲区的指针，该缓冲区由 [函数填充TOKEN\_PRIVILEGES](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_privileges) 结构，该结构包含函数修改的任何特权的先前状态。 也就是说，如果此函数修改了权限，则权限及其以前的状态包含在 *PreviousState* 引用的**TOKEN\_PRIVILEGES**结构中。 如果 **TOKEN\_PRIVILEGES** 的 **PrivilegeCount** 成员为零，则此函数未更改任何特权。 此参数可以为 NULL。  如果指定的缓冲区太小，无法接收修改的权限的完整列表，则函数将失败，并且不会调整任何特权。 在这种情况下， 函数将 *ReturnLength* 参数指向的变量设置为保存修改的权限的完整列表所需的字节数。  [out, optional] ReturnLength  指向变量的指针，该变量接收 *PreviousState* 参数指向的缓冲区的所需大小（以字节为单位）。 如果 *PreviousState* 为 **NULL**，此参数可以为 **NULL**。 返回值 如果该函数成功，则返回值为非零值。 若要确定函数是否调整了所有指定的特权，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)，该函数在函数成功时返回以下值之一：  展开表   |  |  | | --- | --- | | **返回代码** | **说明** | | **ERROR\_SUCCESS** | 函数调整了所有指定的特权。 | | **ERROR\_NOT\_ALL\_ASSIGNED** | 令牌没有 *NewState* 参数中指定的一个或多个特权。 即使未调整任何特权，函数也可能成功并显示此错误值。 *PreviousState* 参数指示调整的权限。 |     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 **AdjustTokenPrivileges** 函数无法向访问令牌添加新特权。 它只能启用或禁用令牌的现有权限。 若要确定令牌的权限，请调用 [GetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-gettokeninformation) 函数。  *NewState* 参数可以指定令牌不具有的权限，而不会导致函数失败。 在这种情况下， 函数调整令牌确实拥有的特权，并忽略其他特权，以便函数成功。 调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 函数以确定该函数是否调整了所有指定的特权。 *PreviousState* 参数指示调整的权限。  *PreviousState* 参数检索包含调整的权限的原始状态的[TOKEN\_PRIVILEGES](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_privileges)结构。 若要还原原始状态，请在后续调用 **AdjustTokenPrivileges** 函数时将 *PreviousState* 指针作为 *NewState* 参数传递。 示例 有关使用此函数的示例，请参阅 [启用和禁用特权](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/enabling-and-disabling-privileges-in-c--)。 要求  |  |  | | --- | --- | | **标头** | securitybaseapi.h (包括 Windows.h) | | **Library** | Advapi32.lib | | **DLL** | Advapi32.dll |  另请参阅 [访问控制概述](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/access-control)  [AdjustTokenGroups](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-adjusttokengroups)  [基本访问控制函数](https://learn.microsoft.com/zh-cn/windows/desktop/SecAuthZ/authorization-functions)  [GetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-gettokeninformation)  [OpenProcessToken](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openprocesstoken)  [OpenThreadToken](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openthreadtoken)  [SetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-settokeninformation)  [TOKEN\_PRIVILEGES](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_privileges) |

## TOKEN\_PRIVILEGES 结构

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TOKEN\_PRIVILEGES** 结构包含有关[访问令牌](https://learn.microsoft.com/zh-cn/windows/desktop/SecGloss/a-gly)的一组特权的信息。 语法 C++复制  typedef struct \_TOKEN\_PRIVILEGES {  DWORD PrivilegeCount;  LUID\_AND\_ATTRIBUTES Privileges[ANYSIZE\_ARRAY];  } TOKEN\_PRIVILEGES, \*PTOKEN\_PRIVILEGES; 成员 PrivilegeCount  必须将其设置为 **Privileges** 数组中的条目数。  Privileges[ANYSIZE\_ARRAY]  指定 [LUID\_AND\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-luid_and_attributes) 结构的数组。 每个结构都包含特权的 [LUID](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-luid) 和属性。 若要获取与 **LUID** 关联的特权的名称，请调用 [LookupPrivilegeName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-lookupprivilegenamea) 函数，并将 **LUID** 的地址作为 *lpLuid* 参数的值传递。  **重要** 公共标头 Winnt.h 中的常**量ANYSIZE\_ARRAY** 定义为 1。 若要创建具有多个元素的数组，必须为结构分配足够的内存，以考虑其他元素。    特权的属性可以是以下值的组合。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **SE\_PRIVILEGE\_ENABLED** | 权限已启用。 | | **SE\_PRIVILEGE\_ENABLED\_BY\_DEFAULT** | 默认情况下，权限处于启用状态。 | | **SE\_PRIVILEGE\_REMOVED** | 用于删除特权。 有关详细信息，请参阅 [AdjustTokenPrivileges](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-adjusttokenprivileges)。 | | **SE\_PRIVILEGE\_USED\_FOR\_ACCESS** | 该特权用于获取对对象或服务的访问权限。 此标志用于标识客户端应用程序传递的集中的相关特权，这些权限可能包含不必要的特权。 |  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** | winnt.h (包括 Windows.h) |  另请参阅 [AdjustTokenPrivileges](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-adjusttokenprivileges)  [GetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-gettokeninformation)  [LUID](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-luid)  [LUID\_AND\_ATTRIBUTES](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-luid_and_attributes)  [LookupPrivilegeName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-lookupprivilegenamea)  [PRIVILEGE\_SET](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-privilege_set)  [PrivilegeCheck](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-privilegecheck)  [PrivilegedServiceAuditAlarm](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-privilegedserviceauditalarma)  [SetTokenInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/securitybaseapi/nf-securitybaseapi-settokeninformation)  [TOKEN\_CONTROL](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_control)  [TOKEN\_DEFAULT\_DACL](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_default_dacl)  [TOKEN\_GROUPS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_groups)  [TOKEN\_INFORMATION\_CLASS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ne-winnt-token_information_class)  [TOKEN\_OWNER](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_owner)  [TOKEN\_PRIMARY\_GROUP](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_primary_group)  [TOKEN\_SOURCE](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_source)  [TOKEN\_STATISTICS](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_statistics)  [TOKEN\_TYPE](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ne-winnt-token_type)  [TOKEN\_USER](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-token_user) |

## 权限常量

|  |
| --- |
| ////////////////////////////////////////////////////////////////////////  // //  // NT Defined Privileges //  // //  ////////////////////////////////////////////////////////////////////////  #define SE\_CREATE\_TOKEN\_NAME TEXT("SeCreateTokenPrivilege")  #define SE\_ASSIGNPRIMARYTOKEN\_NAME TEXT("SeAssignPrimaryTokenPrivilege")  #define SE\_LOCK\_MEMORY\_NAME TEXT("SeLockMemoryPrivilege")  #define SE\_INCREASE\_QUOTA\_NAME TEXT("SeIncreaseQuotaPrivilege")  #define SE\_UNSOLICITED\_INPUT\_NAME TEXT("SeUnsolicitedInputPrivilege")  #define SE\_MACHINE\_ACCOUNT\_NAME TEXT("SeMachineAccountPrivilege")  #define SE\_TCB\_NAME TEXT("SeTcbPrivilege")  #define SE\_SECURITY\_NAME TEXT("SeSecurityPrivilege")  #define SE\_TAKE\_OWNERSHIP\_NAME TEXT("SeTakeOwnershipPrivilege")  #define SE\_LOAD\_DRIVER\_NAME TEXT("SeLoadDriverPrivilege")  #define SE\_SYSTEM\_PROFILE\_NAME TEXT("SeSystemProfilePrivilege")  #define SE\_SYSTEMTIME\_NAME TEXT("SeSystemtimePrivilege")  #define SE\_PROF\_SINGLE\_PROCESS\_NAME TEXT("SeProfileSingleProcessPrivilege")  #define SE\_INC\_BASE\_PRIORITY\_NAME TEXT("SeIncreaseBasePriorityPrivilege")  #define SE\_CREATE\_PAGEFILE\_NAME TEXT("SeCreatePagefilePrivilege")  #define SE\_CREATE\_PERMANENT\_NAME TEXT("SeCreatePermanentPrivilege")  #define SE\_BACKUP\_NAME TEXT("SeBackupPrivilege")  #define SE\_RESTORE\_NAME TEXT("SeRestorePrivilege")  #define SE\_SHUTDOWN\_NAME TEXT("SeShutdownPrivilege")  #define SE\_DEBUG\_NAME TEXT("SeDebugPrivilege")  #define SE\_AUDIT\_NAME TEXT("SeAuditPrivilege")  #define SE\_SYSTEM\_ENVIRONMENT\_NAME TEXT("SeSystemEnvironmentPrivilege")  #define SE\_CHANGE\_NOTIFY\_NAME TEXT("SeChangeNotifyPrivilege")  #define SE\_REMOTE\_SHUTDOWN\_NAME TEXT("SeRemoteShutdownPrivilege")  #define SE\_UNDOCK\_NAME TEXT("SeUndockPrivilege")  #define SE\_SYNC\_AGENT\_NAME TEXT("SeSyncAgentPrivilege")  #define SE\_ENABLE\_DELEGATION\_NAME TEXT("SeEnableDelegationPrivilege")  #define SE\_MANAGE\_VOLUME\_NAME TEXT("SeManageVolumePrivilege")  #define SE\_IMPERSONATE\_NAME TEXT("SeImpersonatePrivilege")  #define SE\_CREATE\_GLOBAL\_NAME TEXT("SeCreateGlobalPrivilege")  #define SE\_TRUSTED\_CREDMAN\_ACCESS\_NAME TEXT("SeTrustedCredManAccessPrivilege")  #define SE\_RELABEL\_NAME TEXT("SeRelabelPrivilege")  #define SE\_INC\_WORKING\_SET\_NAME TEXT("SeIncreaseWorkingSetPrivilege")  #define SE\_TIME\_ZONE\_NAME TEXT("SeTimeZonePrivilege")  #define SE\_CREATE\_SYMBOLIC\_LINK\_NAME TEXT("SeCreateSymbolicLinkPrivilege") |

# 演练

## 老师的代码

|  |
| --- |
|  |
|  |

## 1.新建一个常规的c++空项目，取名Lesson63-enum-procs-psapi，然后新建有cpp文件，取名enum-procs-psapi

|  |
| --- |
|  |

## 2使用psapi枚举进程的代码如下，注意需要使用x64位的编译方式，否则只能找到一些32位的进程

|  |
| --- |
| #include<Windows.h>  #include<Psapi.h>  #include<stdio.h>  #pragma comment(lib,"Psapi.lib")  BOOL UpdateProcessPrivilege(HANDLE hProcess,LPCTSTR lpPrivilegeName = SE\_DEBUG\_NAME);//提升进程权限的函数  void OutputProcessInfo(DWORD pid);  int main()  {  //提示权限  UpdateProcessPrivilege(GetCurrentProcess());  DWORD processes[1024],cbNeeded,processCount;//processes数组里面保存的是进程的id号  if(!EnumProcesses(processes,sizeof(processes),&cbNeeded))  {  printf("枚举进程失败\n");  return 1;  }  //计算一共有多少个进程  processCount = cbNeeded / sizeof(DWORD);  printf("total process:%d\n",processCount);  for(DWORD i=0;i<processCount;i++)  {  OutputProcessInfo(processes[i]);  }  system("pause");  return 0;  }  void OutputProcessInfo(DWORD pid)  {  TCHAR procName[MAX\_PATH] = TEXT("<unknown>");  HMODULE hModule;//这么写只是枚举进程，不枚举进程模块  DWORD dwNeeded;  HANDLE hProcess;  //根据pid获取进程句柄  hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION|PROCESS\_VM\_READ,FALSE,pid);  if(INVALID\_HANDLE\_VALUE == hProcess)  {  printf("获取进程信息错误\n");  return;  }  if(EnumProcessModules(hProcess,&hModule,sizeof(hModule),&dwNeeded))//只显示获取到的  {  GetModuleBaseName(hProcess,hModule,procName,sizeof(procName)/sizeof(TCHAR));  printf("%s (pid: %u)\n",procName,pid);  }    CloseHandle(hProcess);  }  BOOL UpdateProcessPrivilege(HANDLE hProcess,LPCTSTR lpPrivilegeName)  {  HANDLE hToken;  int iResult;  TOKEN\_PRIVILEGES tokenPrivileges;  if(OpenProcessToken(hProcess,TOKEN\_ALL\_ACCESS,&hToken))  {  LUID destLuid;  if(LookupPrivilegeValue(NULL,lpPrivilegeName,&destLuid))  {  tokenPrivileges.PrivilegeCount =1;  tokenPrivileges.Privileges[0].Attributes = SE\_PRIVILEGE\_ENABLED;  tokenPrivileges.Privileges[0].Luid = destLuid;  if(AdjustTokenPrivileges(hToken,FALSE,&tokenPrivileges,0,NULL,NULL))  {  return TRUE;  }  }  }  return FALSE;  } |

## 注意：上面的程序只能够枚举一个进程，不能够枚举这个进程使用了哪些模块，枚举进行和模块的写法参考下面这个例子

# 微软网站提供了一个枚举每一个进程的所有模块的例子

|  |
| --- |
|  |

## enumallmods.cpp的代码如下

|  |
| --- |
| #include <windows.h>  #include <tchar.h>  #include <stdio.h>  #include <psapi.h>  #pragma comment(lib,"Psapi.lib")  // To ensure correct resolution of symbols, add Psapi.lib to TARGETLIBS  // and compile with -DPSAPI\_VERSION=1  int PrintModules( DWORD processID )  {  HMODULE hMods[1024];  HANDLE hProcess;  DWORD cbNeeded;  unsigned int i;  // Print the process identifier.  printf( "\nProcess ID: %u\n", processID );  // Get a handle to the process.  hProcess = OpenProcess( PROCESS\_QUERY\_INFORMATION |  PROCESS\_VM\_READ,  FALSE, processID );  if (NULL == hProcess)  return 1;  // Get a list of all the modules in this process.  if( EnumProcessModules(hProcess, hMods, sizeof(hMods), &cbNeeded))  {  for ( i = 0; i < (cbNeeded / sizeof(HMODULE)); i++ )  {  TCHAR szModName[MAX\_PATH];  // Get the full path to the module's file.  if ( GetModuleFileNameEx( hProcess, hMods[i], szModName,  sizeof(szModName) / sizeof(TCHAR)))  {  // Print the module name and handle value.  \_tprintf( TEXT("\t%s (0x%08X)\n"), szModName, hMods[i] );  }  }  }    // Release the handle to the process.  CloseHandle( hProcess );  return 0;  }  int main( void )  {  DWORD aProcesses[1024];  DWORD cbNeeded;  DWORD cProcesses;  unsigned int i;  // Get the list of process identifiers.  if ( !EnumProcesses( aProcesses, sizeof(aProcesses), &cbNeeded ) )  return 1;  // Calculate how many process identifiers were returned.  cProcesses = cbNeeded / sizeof(DWORD);  // Print the names of the modules for each process.  for ( i = 0; i < cProcesses; i++ )  {  PrintModules( aProcesses[i] );  }  return 0;  } |

## 只有两个项目其实只有一个地方不一样，就是定义HMODLE变量的时候是定义一个单个变量还是定义一个数组，如果是单个变量，就只能够枚举进程，如果是一个数组，就可以枚举进程和它所用到的所有模块，当然，使用模块的时候需要使用循环来遍历。

# 扩展：可以把这个程序用MFC对话框应用程序来实现以便