# 这一节我们来学习获取系统信息包括版本信息和硬件信息

# 学习大纲

|  |
| --- |
|  |

## GetVersionEx函数的语法

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C++复制  NOT\_BUILD\_WINDOWS\_DEPRECATE BOOL GetVersionExA(  [in, out] LPOSVERSIONINFOA lpVersionInformation  ); 参数 [in, out] lpVersionInformation  接收操作系统信息的 [OSVERSIONINFOA](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-osversioninfoa) 或 [OSVERSIONINFOEXA](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/ns-winnt-osversioninfoexa) 结构。  在调用 **GetVersionEx** 函数之前，请根据需要设置结构的 **dwOSVersionInfoSize** 成员，以指示要传递给此函数的数据结构。 返回值 如果函数成功，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。 如果为 **OSVERSIONINFOA 或 OSVERSIONINFOEXA 结构的 dwOSVersionInfoSize** 成员指定了无效值，则该函数将失败。 注解 识别当前操作系统通常不是确定是否存在特定操作系统功能的最佳方法。 这是因为操作系统可能已在可再发行 DLL 中添加了新功能。 测试功能本身是否存在，而不是使用 **GetVersionEx** 来确定操作系统平台或版本号。 有关详细信息，请参阅 [操作系统版本](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/operating-system-version)。  [GetSystemMetrics](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-getsystemmetrics) 函数提供有关当前操作系统的其他信息。  展开表   |  |  | | --- | --- | | **产品** | **设置** | | Windows XP Media Center Edition | SM\_MEDIACENTER | | Windows XP Starter Edition | SM\_STARTER | | Windows XP Tablet PC Edition | SM\_TABLETPC | | Windows Server 2003 R2 | SM\_SERVERR2 |     若要针对特定操作系统或操作系统功能检查，请使用 [IsOS](https://learn.microsoft.com/zh-cn/windows/desktop/api/shlwapi/nf-shlwapi-isos) 函数。 [GetProductInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getproductinfo) 函数检索产品类型。  若要检索远程计算机上操作系统的信息，请使用 [NetWkstaGetInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/lmwksta/nf-lmwksta-netwkstagetinfo) 函数、[Win32\_OperatingSystem](https://learn.microsoft.com/zh-cn/windows/desktop/CIMWin32Prov/win32-operatingsystem) WMI 类或 **IADsComputer** 接口的 [OperatingSystem](https://learn.microsoft.com/zh-cn/windows/desktop/ADSI/iadscomputer-property-methods) 属性。  若要将当前系统版本与所需版本进行比较，请使用 [VerifyVersionInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-verifyversioninfoa) 函数而不是 **GetVersionEx** 自行执行比较。  如果兼容模式有效， **则 GetVersionEx** 函数在标识自身时报告操作系统，该操作系统可能不是安装的操作系统。 例如，如果兼容模式有效， **GetVersionEx** 将报告为 [应用程序兼容性](https://learn.microsoft.com/zh-cn/previous-versions/bb757005(v=msdn.10))选择的操作系统。 示例 使用 **GetVersionEx** 函数确定应用程序是否在特定版本的操作系统上运行时，检查大于或等于所需版本号的版本号。 这可确保更高版本的操作系统的测试成功。 例如，如果应用程序需要 Windows XP 或更高版本，请使用以下测试。  #include <windows.h>  #include <stdio.h>  void main()  {  OSVERSIONINFO osvi;  BOOL bIsWindowsXPorLater;  ZeroMemory(&osvi, sizeof(OSVERSIONINFO));  osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  GetVersionEx(&osvi);  bIsWindowsXPorLater =  ( (osvi.dwMajorVersion > 5) ||  ( (osvi.dwMajorVersion == 5) && (osvi.dwMinorVersion >= 1) ));  if(bIsWindowsXPorLater)  printf("The system meets the requirements.\n");  else printf("The system does not meet the requirements.\n");  }  有关标识当前操作系统的示例，请参阅 [获取系统版本](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/getting-the-system-version)。  **备注**  sysinfoapi.h 标头将 GetVersionEx 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)。 |

## OSVERSIONINFO结构体

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| typedef struct \_OSVERSIONINFOA {  DWORD dwOSVersionInfoSize;  DWORD dwMajorVersion;  DWORD dwMinorVersion;  DWORD dwBuildNumber;  DWORD dwPlatformId;  CHAR szCSDVersion[128];  } OSVERSIONINFOA, \*POSVERSIONINFOA, \*LPOSVERSIONINFOA; 成员 dwOSVersionInfoSize  此数据结构的大小（以字节为单位）。 将此成员设置为 sizeof(OSVERSIONINFO)。  dwMajorVersion  操作系统的主版本号。 有关详细信息，请参阅“备注”。  dwMinorVersion  操作系统的次要版本号。 有关详细信息，请参阅“备注”。  dwBuildNumber  操作系统的内部版本号。  dwPlatformId  操作系统平台。 此成员可以是以下值。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **VER\_PLATFORM\_WIN32\_NT**  2 | 操作系统是 Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003、Windows XP 或 Windows 2000。 |   szCSDVersion[128]  一个以 null 结尾的字符串，例如“Service Pack 3”，指示系统上安装的最新 Service Pack。 如果未安装 Service Pack，则字符串为空。 注解 依赖版本信息不是测试功能的最佳方式。 有关相关功能，请参阅文档。 有关功能检测的常见技术的详细信息，请参阅 [操作系统版本](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/operating-system-version)。  如果必须需要特定的操作系统，请确保将其用作最低支持版本，而不是为一个操作系统设计测试。 这样，检测代码将继续在 Windows 的未来版本上运行。  下表汇总了受支持的 Windows 版本返回的值。 使用标记为“其他”的列中的信息来区分版本号相同的操作系统。  展开表   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **操作系统** | **版本号** | **dwMajorVersion** | **dwMinorVersion** | **其他** | | Windows 10 | 10.0\* | 10 | 0 | OSVERSIONINFOEX.wProductType == VER\_NT\_WORKSTATION | | Windows Server 2016 | 10.0\* | 10 | 0 | OSVERSIONINFOEX.wProductType ！= VER\_NT\_WORKSTATION | | Windows 8.1 | 6.3\* | 6 | 3 | OSVERSIONINFOEX.wProductType == VER\_NT\_WORKSTATION | | Windows Server 2012 R2 | 6.3\* | 6 | 3 | OSVERSIONINFOEX.wProductType ！= VER\_NT\_WORKSTATION | | Windows 8 | 6.2 | 6 | 2 | OSVERSIONINFOEX.wProductType == VER\_NT\_WORKSTATION | | Windows Server 2012 | 6.2 | 6 | 2 | OSVERSIONINFOEX.wProductType ！= VER\_NT\_WORKSTATION | | Windows 7 | 6.1 | 6 | 1 | OSVERSIONINFOEX.wProductType == VER\_NT\_WORKSTATION | | Windows Server 2008 R2 | 6.1 | 6 | 1 | OSVERSIONINFOEX.wProductType ！= VER\_NT\_WORKSTATION | | Windows Server 2008 | 6.0 | 6 | 0 | OSVERSIONINFOEX.wProductType ！= VER\_NT\_WORKSTATION | | Windows Vista | 6.0 | 6 | 0 | OSVERSIONINFOEX.wProductType == VER\_NT\_WORKSTATION | | Windows Server 2003 R2 | 5.2 | 5 | 2 | GetSystemMetrics (SM\_SERVERR2) ！= 0 | | Windows Server 2003 | 5.2 | 5 | 2 | GetSystemMetrics (SM\_SERVERR2) == 0 | | Windows XP | 5.1 | 5 | 1 | 不适用 | | Windows 2000 | 5.0 | 5 | 0 | 不适用 | | **\***对于已针对Windows 8.1或Windows 10进行清单的应用程序。 未针对Windows 8.1或Windows 10进行清单的应用程序将返回Windows 8 OS 版本值 (6.2) 。 若要针对Windows 8.1或Windows 10来清单应用程序，请参阅[面向 Windows 的应用程序](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/targeting-your-application-at-windows-8-1)。 | | | | |    示例 有关示例，请参阅 [获取系统版本](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/getting-the-system-version)。  **备注**  winnt.h 标头将 OSVERSIONINFO 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)。 |

## GetSystemInfo函数的语法

|  |
| --- |
| void GetSystemInfo(  [out] LPSYSTEM\_INFO lpSystemInfo  ); 参数 [out] lpSystemInfo  指向接收信息的 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的指针。 返回值 无 |

## SYSTEM\_INFO

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C++复制  typedef struct \_SYSTEM\_INFO {  union {  DWORD dwOemId;  struct {  WORD wProcessorArchitecture;  WORD wReserved;  } DUMMYSTRUCTNAME;  } DUMMYUNIONNAME;  DWORD dwPageSize;  LPVOID lpMinimumApplicationAddress;  LPVOID lpMaximumApplicationAddress;  DWORD\_PTR dwActiveProcessorMask;  DWORD dwNumberOfProcessors;  DWORD dwProcessorType;  DWORD dwAllocationGranularity;  WORD wProcessorLevel;  WORD wProcessorRevision;  } SYSTEM\_INFO, \*LPSYSTEM\_INFO; 成员 DUMMYUNIONNAME  DUMMYUNIONNAME.dwOemId  为保持兼容性而保留的已过时成员。 应用程序应使用联合的 **wProcessorArchitecture** 分支。  DUMMYUNIONNAME.DUMMYSTRUCTNAME  DUMMYUNIONNAME.DUMMYSTRUCTNAME.wProcessorArchitecture  已安装操作系统的处理器体系结构。 此成员可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | **PROCESSOR\_ARCHITECTURE\_AMD64**  9 | x64 (AMD 或 Intel) | | **PROCESSOR\_ARCHITECTURE\_ARM**  5 | ARM | |  |  | | **PROCESSOR\_ARCHITECTURE\_IA64**  6 | 基于 Intel Itanium | | **PROCESSOR\_ARCHITECTURE\_INTEL**  0 | x86 | | **PROCESSOR\_ARCHITECTURE\_UNKNOWN**  0xffff | 未知的体系结构。 |   DUMMYUNIONNAME.DUMMYSTRUCTNAME.wReserved  此成员留待将来使用。  dwPageSize  页面保护和承诺的页面大小和粒度。 这是 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc) 函数使用的页大小。  lpMinimumApplicationAddress  指向应用程序和动态链接库可访问的最低内存地址的指针， (DLL) 。  lpMaximumApplicationAddress  指向应用程序和 DLL 可访问的最高内存地址的指针。  dwActiveProcessorMask  一个掩码，表示在系统中配置的处理器集。 位 0 是处理器 0;位 31 是处理器 31。  dwNumberOfProcessors  当前组中的逻辑处理器数。 若要检索当前处理器组，请使用 [GetLogicalProcessorInformation](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getlogicalprocessorinformation) 函数。  **注意** 有关逻辑处理器共享的物理处理器的信息，请调用 [**GetLogicalProcessorInformationEx**](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getlogicalprocessorinformationex) ，并将 *RelationshipType* 参数设置为 RelationProcessorPackage (3) 。    dwProcessorType  为保持兼容性而保留的已过时成员。 使用 **wProcessorArchitecture**、 **wProcessorLevel** 和 **wProcessorRevision** 成员来确定处理器的类型。 PROCESSOR\_INTEL\_386 (386)PROCESSOR\_INTEL\_486 (486)PROCESSOR\_INTEL\_PENTIUM (586)PROCESSOR\_INTEL\_IA64 (2200)PROCESSOR\_AMD\_X8664 (8664)PROCESSOR\_ARM (保留) dwAllocationGranularity  可以分配虚拟内存的起始地址的粒度。 有关详细信息，请参阅 [VirtualAlloc](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc)。  wProcessorLevel  依赖于体系结构的处理器级别。 它应仅用于显示目的。 若要确定处理器的功能集，请使用 [IsProcessorFeaturePresent](https://learn.microsoft.com/zh-cn/windows/desktop/api/processthreadsapi/nf-processthreadsapi-isprocessorfeaturepresent) 函数。  如果 **wProcessorArchitecture** PROCESSOR\_ARCHITECTURE\_INTEL， **则 wProcessorLevel** 由 CPU 供应商定义。  如果 **wProcessorArchitecture** 为PROCESSOR\_ARCHITECTURE\_IA64， **则 wProcessorLevel** 设置为 1。  wProcessorRevision  依赖于体系结构的处理器修订版。 下表显示了如何为每种类型的处理器体系结构组合修订值。  展开表   |  |  | | --- | --- | | **处理器** | **值** | | Intel Pentium、Cyrix 或 NextGen 586 | 高字节是模型，低字节是单步执行。 例如，如果值为 *xxyy，*则可以按如下所示显示型号和步进：  模型 *xx*，单步 *yy* | | Intel 80386 或 80486 | 形式的值 *xxyz*。  如果 *xx* 等于 0xFF， *则 y* - 0xA 是型号， *z* 是步进标识符。  如果 *xx* 不等于 0xFF， *则 xx* + “A” 是步进字母， *yz* 是次要单步执行。 | | ARM | 保留。 | |

## 64位的系统需要使用GetNativeSystemInfo，参数是一样的

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GetNativeSystemInfo 函数 (sysinfoapi.h)  * 项目 * 2024/03/02   反馈 本文内容  1. [语法](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo#syntax) 2. [参数](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo#parameters) 3. [返回值](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo#return-value) 4. [备注](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo#remarks)   显示另外 2 个  将有关当前系统的信息检索到在 [WOW64](https://learn.microsoft.com/zh-cn/windows/desktop/WinProg64/running-32-bit-applications) 下运行的应用程序。 如果从 64 位应用程序调用函数，则它等效于 [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo) 函数。 如果函数是从在没有 Intel64 或 x64 处理器 (（如 ARM64) ）的 64 位系统上运行的 x86 或 x64 应用程序调用的，则仅当支持 x86 仿真 (或 x64（如果 x64 仿真也支持 x64) ）时，它才会返回信息，就像系统是 x86 一样。 语法 C++复制  void GetNativeSystemInfo(  [out] LPSYSTEM\_INFO lpSystemInfo  ); 参数 [out] lpSystemInfo  指向接收信息的 [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info) 结构的指针。 返回值 无 备注 若要确定基于 Win32 的应用程序是否在 WOW64 (下运行，或者 64 位系统是否没有 Intel64 或 x64 处理器) ，请调用 [IsWow64Process2](https://learn.microsoft.com/zh-cn/windows/desktop/api/wow64apiset/nf-wow64apiset-iswow64process2) 函数。  若要编译使用此函数的应用程序，请将\_WIN32\_WINNT定义为 0x0501 或更高版本。 有关详细信息，请参阅 [使用 Windows 标头](https://learn.microsoft.com/zh-cn/windows/desktop/WinProg/using-the-windows-headers)。 示例 有关示例，请参阅 [获取系统版本](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/getting-the-system-version)。 要求  |  |  | | --- | --- | | **标头** | sysinfoapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [IsWow64Process](https://learn.microsoft.com/zh-cn/windows/desktop/api/wow64apiset/nf-wow64apiset-iswow64process)  [SYSTEM\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/ns-sysinfoapi-system_info)  [系统信息函数](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/system-information-functions) |

# 以下函数可用于确定当前操作系统版本，或者确定它是 Windows 或 Windows Server 版本。 这些函数提供简单的测试，这些测试使用 [VerifyVersionInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/Winbase/nf-winbase-verifyversioninfoa) 函数以及建议的大于或等于比较，这些被证明是确定操作系统版本的可靠方法。

| **函数** | **说明** |
| --- | --- |
| [**IsWindowsXPOrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsxporgreater) | 指示当前操作系统版本是与 Windows XP 版本匹配还是高于此版本。 |
| [**IsWindowsXPSP1OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsxpsp1orgreater) | 指示当前操作系统版本是与 Windows XP Service Pack 1 (SP1) 版本匹配还是高于此版本。 |
| [**IsWindowsXPSP2OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsxpsp2orgreater) | 指示当前操作系统版本是与 Windows XP Service Pack 2 (SP2) 版本匹配还是高于此版本。 |
| [**IsWindowsXPSP3OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsxpsp3orgreater) | 指示当前操作系统版本是与 Windows XP Service Pack 3 (SP3) 版本匹配还是高于此版本。 |
| [**IsWindowsVistaOrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsvistaorgreater) | 指示当前操作系统版本是与 Windows Vista 版本匹配还是高于此版本。 |
| [**IsWindowsVistaSP1OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsvistasp1orgreater) | 指示当前操作系统版本是与 Windows Vista Service Pack 1 (SP1) 版本匹配还是高于此版本。 |
| [**IsWindowsVistaSP2OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsvistasp2orgreater) | 指示当前操作系统版本是与 Windows Vista Service Pack 2 (SP2) 版本匹配还是高于此版本。 |
| [**IsWindows7OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows7orgreater) | 指示当前操作系统版本是与 Windows 7 版本匹配还是高于此版本。 |
| [**IsWindows7SP1OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows7sp1orgreater) | 指示当前操作系统版本是与 Windows 7 Service Pack 1 (SP1) 版本匹配还是高于此版本。 |
| [**IsWindows8OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows8orgreater) | 指示当前操作系统版本是与 Windows 8 版本匹配还是高于此版本。 |
| [**IsWindows8Point1OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows8point1orgreater) | 指示当前操作系统版本是与 Windows 8.1 版本匹配还是高于此版本。  对于 Windows 10，[IsWindows8Point1OrGreater](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows8point1orgreater) 会返回 false，除非应用程序包含的清单包括一个兼容性部分，并且该兼容性部分包含用于指定 Windows 8.1 和/或 Windows 10 的 GUID。 |
| [**IsWindows10OrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows10orgreater) | 指示当前操作系统版本是与 Windows 10 版本匹配还是高于此版本。  对于 Windows 10，[IsWindows10OrGreater](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindows10orgreater) 会返回 false，除非应用程序包含的清单包括一个兼容性部分，并且该兼容性部分包含用于指定 Windows 10 的 GUID。 |
| [**IsWindowsServer**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsserver) | 指示当前操作系统是否为 Windows Server 版本。 需要区分 Windows 的服务器版本和客户端版本的应用程序应调用此函数。 |
| [**IsWindowsVersionOrGreater**](https://learn.microsoft.com/zh-cn/windows/win32/api/VersionHelpers/nf-versionhelpers-iswindowsversionorgreater) | 仅当其他提供的版本帮助程序函数不适合您的场景时，您才应该使用此函数。  指示当前操作系统版本是与提供的版本信息匹配还是高于此版本。 此函数可用于确认不与客户端版本共享版本号的 Windows Server 版本。 |

## 示例

利用 **VersionHelpers.h** 头文件中定义的内联函数，你可以在测试 Windows 版本时返回**布尔**值来验证操作系统版本。

例如，如果应用程序需要 Windows 10 或更高版本，请使用以下测试。

C++

#include <windows.h>

#include <VersionHelpers.h>

if (!IsWindows10OrGreater())

{

MessageBox(NULL, "You need at least Windows 10", "Version Not Supported", MB\_OK);

}

# 演练

## 1.新建一个c++常规空项目，取名Lesson23-24-sys-ver-and-info,然后添加一个c++源文件，取名：sys-ver-and-info.cpp,并且添加最基本的骨架代码，如下

|  |
| --- |
|  |

## 2.我们先来做第一个功能，定义一个函数，叫做：ShowVersionInfo，然后添加代码如下

|  |
| --- |
| void ShowVersionInfo()  {  OSVERSIONINFO osinfo;  //有坑，这个结构体需要初始化它的大小成员属性  osinfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  if(!GetVersionEx(&osinfo))//有坑，这个ex函数使用没有ex的结构体！！！  {  printf("Get OS Version Failed:%d\n",GetLastError());  return;  }  printf("OS Build Number:%d\n",osinfo.dwBuildNumber);  printf("OS Major Version:%d\n",osinfo.dwMajorVersion);  printf("OS Minor Version:%d\n",osinfo.dwMinorVersion);  printf("OS Platform ID:%d\n",osinfo.dwPlatformId);  printf("Service Pack:%s\n",osinfo.szCSDVersion);  } |

### 注意：这里有2个坑，

#### 1）.这个GetVersionEx函数使用OSVERSIONINFO结构体而不是OSVERSIONINFOEX

#### 2）.在定义函数之前，需要把oSVERSIONINFO结构体对象的dwOSVersionInfoSize成员进行初始化赋值，否则函数调用失败

## 3.这个函数写的不太好，我又写了一个ShowVersionInfoString()函数，代码如下

|  |
| --- |
| void ShowVersionInfoString()  {  TCHAR szVersionInfo[1024];  OSVERSIONINFO osinfo;  \*szVersionInfo = NULL;  //坑2，这个结构体需要初始化他的一个大小成员  osinfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  if(!GetVersionEx(&osinfo))//注意，坑1，这个ex函数使用的不是ex结构体！！！  {  printf("Get OS Version Failed:%d\n",GetLastError());  return;  }  if(osinfo.dwMajorVersion == 5)  {  if(osinfo.dwMinorVersion == 0) lstrcat(szVersionInfo,TEXT("Window 2000"));  else if(osinfo.dwMinorVersion == 1) lstrcat(szVersionInfo,TEXT("Window xp"));  else if(osinfo.dwMinorVersion == 2) lstrcat(szVersionInfo,TEXT("Window Server 2003"));  }  else if(osinfo.dwMajorVersion == 6)  {  if(osinfo.dwMinorVersion == 0) lstrcat(szVersionInfo,TEXT("Window Vista"));  else if(osinfo.dwMinorVersion == 1) lstrcat(szVersionInfo,TEXT("Window7"));  else if(osinfo.dwMinorVersion == 2) lstrcat(szVersionInfo,TEXT("Window8"));  else if(osinfo.dwMinorVersion == 3) lstrcat(szVersionInfo,TEXT("Window8.1"));    }  else if(osinfo.dwMajorVersion == 10)  {  lstrcat(szVersionInfo,TEXT("Window10"));  }  printf("当前操作系统是:%s\n",szVersionInfo);  printf("Windows %d.%d Build %d\n",osinfo.dwMajorVersion,  osinfo.dwMinorVersion,osinfo.dwBuildNumber);    } |

## 4.编写另外一个函数：ShowSystemInfo()用来显示系统硬件信息

|  |
| --- |
| void ShowSystemInfo()  {  SYSTEM\_INFO sysinfo;  GetNativeSystemInfo(&sysinfo);  printf("OEM ID:%d\n",sysinfo.dwOemId);  printf("Number Of Processors:%d\n",sysinfo.dwNumberOfProcessors);  printf("Page Size:%d\n",sysinfo.dwPageSize);  printf("Maximum Application Address:%d\n",sysinfo.lpMaximumApplicationAddress);  printf("Minimum Application Address:%d\n",sysinfo.lpMinimumApplicationAddress);  printf("Processor Level:%d\n",sysinfo.wProcessorLevel);  printf("Processor Revision:%d\n",sysinfo.wProcessorRevision);  printf("Processor Type:");  switch(sysinfo.dwProcessorType)  {  case PROCESSOR\_INTEL\_386:  printf("Intel 80386 Processor\n");  break;  case PROCESSOR\_INTEL\_486:  printf("Intel 80486 Processor\n");  break;  case PROCESSOR\_INTEL\_PENTIUM:  printf("Intel Pentium 80586 Processor\n");  break;  case PROCESSOR\_INTEL\_IA64:  printf("Intel IA64 Processor\n");  break;  case PROCESSOR\_AMD\_X8664:  printf("AMD x8664 Processor\n");  break;  default:  printf("Other Processor\n");    break;  }  printf("Processor Architecture:");  switch(sysinfo.wProcessorArchitecture)  {  case PROCESSOR\_ARCHITECTURE\_INTEL:  printf("因特尔芯片架构\n");  break;  case PROCESSOR\_ARCHITECTURE\_IA64:  printf("64位因特尔芯片架构\n");  break;  case PROCESSOR\_ARCHITECTURE\_AMD64:  printf("64位AMD芯片架构\n");  break;  case PROCESSOR\_ARCHITECTURE\_ARM:  printf("ARM芯片架构\n");  break;  default:  printf("未知芯片架构\n");  break;  }  } |

### 效果：

|  |
| --- |
|  |

# 完整代码如下

## sys-ver-and-info.cpp

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  void ShowVersionInfo()  {  OSVERSIONINFO osinfo;  //坑2，这个结构体需要初始化他的一个大小成员  osinfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  if(!GetVersionEx(&osinfo))//注意，坑1，这个ex函数使用的表示ex结构体！！！  {  printf("Get OS Version Failed:%d\n",GetLastError());  return;  }  printf("OS Build Number:%d\n",osinfo.dwBuildNumber);  printf("OS Major Version:%d\n",osinfo.dwMajorVersion);  printf("OS Minor Version:%d\n",osinfo.dwMinorVersion);  printf("OS Platform ID:%d\n",osinfo.dwPlatformId);  printf("Service Pack:%s\n",osinfo.szCSDVersion);  }  void ShowVersionInfoString()  {  TCHAR szVersionInfo[1024];  OSVERSIONINFO osinfo;  \*szVersionInfo = NULL;  //坑2，这个结构体需要初始化他的一个大小成员  osinfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  if(!GetVersionEx(&osinfo))//注意，坑1，这个ex函数使用的表示ex结构体！！！  {  printf("Get OS Version Failed:%d\n",GetLastError());  return;  }  if(osinfo.dwMajorVersion == 5)  {  if(osinfo.dwMinorVersion == 0) lstrcat(szVersionInfo,TEXT("Window 2000"));  else if(osinfo.dwMinorVersion == 1) lstrcat(szVersionInfo,TEXT("Window xp"));  else if(osinfo.dwMinorVersion == 2) lstrcat(szVersionInfo,TEXT("Window Server 2003"));  }  else if(osinfo.dwMajorVersion == 6)  {  if(osinfo.dwMinorVersion == 0) lstrcat(szVersionInfo,TEXT("Window Vista"));  else if(osinfo.dwMinorVersion == 1) lstrcat(szVersionInfo,TEXT("Window7"));  else if(osinfo.dwMinorVersion == 2) lstrcat(szVersionInfo,TEXT("Window8"));  else if(osinfo.dwMinorVersion == 3) lstrcat(szVersionInfo,TEXT("Window8.1"));    }  else if(osinfo.dwMajorVersion == 10)  {  lstrcat(szVersionInfo,TEXT("Window10"));  }  printf("当前操作系统是:%s\n",szVersionInfo);  printf("Windows %d.%d Build %d\n",osinfo.dwMajorVersion,  osinfo.dwMinorVersion,osinfo.dwBuildNumber);    }  void ShowSystemInfo()  {  SYSTEM\_INFO sysinfo;  GetNativeSystemInfo(&sysinfo);  printf("OEM ID:%d\n",sysinfo.dwOemId);  printf("Number Of Processors:%d\n",sysinfo.dwNumberOfProcessors);  printf("Page Size:%d\n",sysinfo.dwPageSize);  printf("Maximum Application Address:%p\n",sysinfo.lpMaximumApplicationAddress);//这个属性是16进制的要用%p  printf("Minimum Application Address:%p\n",sysinfo.lpMinimumApplicationAddress); //这个属性是16进制的要用%p  printf("Processor Level:%d\n",sysinfo.wProcessorLevel);  printf("Processor Revision:%d\n",sysinfo.wProcessorRevision);  printf("Processor Type:");  switch(sysinfo.dwProcessorType)  {  case PROCESSOR\_INTEL\_386:  printf("Intel 80386 Processor\n");  break;  case PROCESSOR\_INTEL\_486:  printf("Intel 80486 Processor\n");  break;  case PROCESSOR\_INTEL\_PENTIUM:  printf("Intel Pentium 80586 Processor\n");  break;  case PROCESSOR\_INTEL\_IA64:  printf("Intel IA64 Processor\n");  break;  case PROCESSOR\_AMD\_X8664:  printf("AMD x8664 Processor\n");  break;  default:  printf("Other Processor\n");    break;  }  printf("Processor Architecture:");  switch(sysinfo.wProcessorArchitecture)  {  case PROCESSOR\_ARCHITECTURE\_INTEL:  printf("因特尔芯片架构\n");  break;  case PROCESSOR\_ARCHITECTURE\_IA64:  printf("64位因特尔芯片架构\n");  break;  case PROCESSOR\_ARCHITECTURE\_AMD64:  printf("64位AMD芯片架构\n");  break;  case PROCESSOR\_ARCHITECTURE\_ARM:  printf("ARM芯片架构\n");  break;  default:  printf("未知芯片架构\n");  break;  }  }  int main()  {  //ShowVersionInfo();  //ShowVersionInfoString();  ShowSystemInfo();  system("pause");  return 0;  } |

### 注意:GetVersonEx函数微软官方已经声明位弃用,因为它能够检测到的最高版本是windows8.

# 新的检测操作系统版本的代码如下

|  |
| --- |
| #include <windows.h>  #include <stdio.h>  #include <VersionHelpers.h>  int \_\_cdecl wmain(  \_\_in int argc,  \_\_in\_ecount(argc) PCWSTR argv[]  )  {  UNREFERENCED\_PARAMETER(argc);  UNREFERENCED\_PARAMETER(argv);  if (IsWindows10OrGreater()) //>=win10  {  printf("Windows10OrGreater\n");  }  else if (IsWindows8Point1OrGreater())//>=win8.1  {  printf("Windows8Point1OrGreater\n");  }  else if (IsWindows8OrGreater()) //>=win8  {  printf("Windows8OrGreater\n");  }  else if (IsWindows7SP1OrGreater()) //>=win7.1  {  printf("Windows7SP1OrGreater\n");  }  else if (IsWindows7OrGreater())//>=win7.1  {  printf("Windows7OrGreater\n");  }  else if (IsWindowsVistaSP2OrGreater())//>=vista2.0  {  printf("VistaSP2OrGreater\n");  }  else if (IsWindowsVistaSP1OrGreater())//>=vista1.0  {  printf("VistaSP1OrGreater\n");  }  else if (IsWindowsVistaOrGreater())//>=vista  {  printf("VistaOrGreater\n");  }  else if (IsWindowsXPSP3OrGreater())//>=win xp3  {  printf("XPSP3OrGreater\n");  }  else if (IsWindowsXPSP2OrGreater())//>=win xp2  {  printf("XPSP2OrGreater\n");  }  else if (IsWindowsXPSP1OrGreater())//>=win xp1  {  printf("XPSP1OrGreater\n");  }  else if (IsWindowsXPOrGreater()) //>=win xp0  {  printf("XPOrGreater\n");  }    if (IsWindowsServer())  {  printf("Server\n");  }  else  {  printf("Client\n");  }  } |

## 宽字符版本

|  |
| --- |
|  |
| #include<Windows.h>  #include<iostream>  #include<VersionHelpers.h>  #include<locale.h>  #pragma warning(disable:4996)  void OutputOsVersion()  {  TCHAR szVersionInfo[1024];  ZeroMemory(szVersionInfo, 1024);  OSVERSIONINFO osinfo;  osinfo.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  if (!GetVersionEx(&osinfo))  {  wprintf\_s(L"Get Os Version information failed\n");  return;  }  switch (osinfo.dwMajorVersion)  {  case 5:  switch (osinfo.dwMinorVersion)  {  case 0:  lstrcat(szVersionInfo, TEXT("Window 2000"));  break;  case 1:  lstrcat(szVersionInfo, TEXT("Window xp"));  break;  case 2:  lstrcat(szVersionInfo, TEXT("Window Server 2003"));  break;  }  break;  case 6:  switch (osinfo.dwMinorVersion)  {  case 0:  lstrcat(szVersionInfo, TEXT("Window vista"));  break;  case 1:  lstrcat(szVersionInfo, TEXT("Window 7"));  break;  case 2:  lstrcat(szVersionInfo, TEXT("Window 8"));  break;  case 3:  lstrcat(szVersionInfo, TEXT("Window 8.1"));  break;  }  break;  case 10:  lstrcat(szVersionInfo, TEXT("Window10"));  break;  }  wprintf\_s(L"Os:%s\n %d.%d Build%d\n", szVersionInfo,osinfo.dwMinorVersion,osinfo.dwMinorVersion,osinfo.dwBuildNumber);  }  void NewMethodToDetectOsVersion()  {  if (IsWindows10OrGreater())  {  printf("Windows10OrGreater\n");  }  else if (IsWindows8Point1OrGreater())  {  printf("Windows8Point1OrGreater\n");  }  else if (IsWindows8OrGreater())  {  printf("Windows8OrGreater\n");  }  else if (IsWindows7SP1OrGreater())  {  printf("Windows7SP1OrGreater\n");  }  else if (IsWindows7OrGreater())  {  printf("Windows7OrGreater\n");  }  else if (IsWindowsVistaSP2OrGreater())  {  printf("VistaSP2OrGreater\n");  }  else if (IsWindowsVistaSP1OrGreater())  {  printf("VistaSP1OrGreater\n");  }  else if (IsWindowsVistaOrGreater())  {  printf("VistaOrGreater\n");  }  else if (IsWindowsXPSP3OrGreater())  {  printf("XPSP3OrGreater\n");  }  else if (IsWindowsXPSP2OrGreater())  {  printf("XPSP2OrGreater\n");  }  else if (IsWindowsXPSP1OrGreater())  {  printf("XPSP1OrGreater\n");  }  else if (IsWindowsXPOrGreater())  {  printf("XPOrGreater\n");  }    if (IsWindowsServer())  {  printf("Server\n");  }  else  {  printf("Client\n");  }  }  void GetProcessorTypeStr(DWORD type, WCHAR\*\* infoStr)  {  //这里需要在堆分配内存,因为不能把局部变量的地址返回  WCHAR\* szInfo = (WCHAR\*)HeapAlloc(GetProcessHeap(), HEAP\_ZERO\_MEMORY, 100);    switch (type)  {  case PROCESSOR\_INTEL\_386:  lstrcpy(szInfo, L"Intel 80386 Processor");  break;  case PROCESSOR\_INTEL\_486:  lstrcpy(szInfo, L"Intel 80486 Processor");  break;  case PROCESSOR\_INTEL\_PENTIUM:  lstrcpy(szInfo, L"Intel Pentium 80586 Processor");  break;  case PROCESSOR\_INTEL\_IA64:  lstrcpy(szInfo, L"Intel IA64 Processor");  break;  case PROCESSOR\_AMD\_X8664:  lstrcpy(szInfo, L"AMD x8664 Processor");  break;  default:  lstrcpy(szInfo, L"Other Processor");  break;  }  \*infoStr = szInfo;//在这个函数里面没有释放内存,需要在调用的地方使用完后释放内存  }  void GetProcessorArchitectureStr(DWORD arch, WCHAR\*\* archStr)  {  //这里需要在堆分配内存,因为不能把局部变量的地址返回  WCHAR\* szInfo = (WCHAR\*)HeapAlloc(GetProcessHeap(), HEAP\_ZERO\_MEMORY, 100);  switch (arch)  {  case PROCESSOR\_ARCHITECTURE\_INTEL:  lstrcpy(szInfo, L"因特尔芯片架构");  break;  case PROCESSOR\_ARCHITECTURE\_IA64:  lstrcpy(szInfo, L"64位因特尔芯片架构");  break;  case PROCESSOR\_ARCHITECTURE\_AMD64:  lstrcpy(szInfo, L"64位AMD芯片架构");  break;  case PROCESSOR\_ARCHITECTURE\_ARM:  lstrcpy(szInfo, L"ARM芯片架构");  break;  default:  lstrcpy(szInfo, L"未知芯片架构");  break;  }  \*archStr = szInfo;//在这个函数里面没有释放内存,需要在调用的地方使用完后释放内存  }  void DisplayNativeSysInto()  {  WCHAR \*szType;  WCHAR \*szarchType;  SYSTEM\_INFO si;  GetNativeSystemInfo(&si);  wprintf\_s(L"OEM ID:%d\n", si.dwOemId);  wprintf\_s(L"Page Size:%d BYTES\n", si.dwPageSize);  wprintf\_s(L"Number Of Processors:%d\n", si.dwNumberOfProcessors);  wprintf\_s(L"Allocation Granularity:%d\n", si.dwAllocationGranularity);  wprintf\_s(L"Active Processor Mask:%lld\n", si.dwActiveProcessorMask);  wprintf\_s(L"Maximum Application Address:0x%Ix\n", (INT\_PTR)si.lpMaximumApplicationAddress);  wprintf\_s(L"Minimum Application Address:0x%Ix\n", (INT\_PTR)si.lpMinimumApplicationAddress);  wprintf\_s(L"Processor Level:%d\n", si.wProcessorLevel);  wprintf\_s(L"Processor Revision:%d\n", si.wProcessorRevision);  GetProcessorTypeStr(si.dwProcessorType, &szType);  GetProcessorArchitectureStr(si.wProcessorArchitecture, &szarchType);  wprintf\_s(L"Porcess Type:%s\n", szType);  wprintf\_s(L"Processor Architecture:%s\n", szarchType);  HeapFree(GetProcessHeap(), 0, szType);// 调用完函数需要释放它在堆中分配的内存  HeapFree(GetProcessHeap(), 0, szarchType);// 调用完函数需要释放它在堆中分配的内存  }  int wmain()  {  setlocale(LC\_ALL, "");  //OutputOsVersion();  NewMethodToDetectOsVersion();  DisplayNativeSysInto();  system("pause");  return 0;  } |

### 效果

|  |
| --- |
|  |

# 这一节的学习到此为止