# 学习大纲

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

## ­所谓的页面文件,就是指windows一次分配的内存大小,当我们需要内存的时候,我们需要项window申请分配内存,即使你只需要20个字节,但是,window不会分配给你那么小,它分配管理是最小内存单位是4k,也就是4096个字节.也就是说window的内存页面大小是4k.在windows中内存是按页分配的,如,你向系统申请5000字节的内存,系统会给你2个页,也就是8k,因为一个页放不下5000字节,多出的部分即使只有2个字节,也需要一个页,多了也没有办法.

## 注意:我们能够使用的内存地址其实不是从0开始,是从0x10000开始的,这一点需要记住,它前面的地址也就是0x00000000 - 0x0000ffff 被windows指定为空指针赋值分区

## 所谓的分配粒度,就是指windows内存分配的最小单位,他是64k,也就是我们向系统申请的内存地址的起始地址必须是64k的整数倍,否则系统分配内存失败,可以申请的内存大小是4k的整数倍.

## 32位windows能够管理的最大内存的4G,windows系统占用2G,剩下的需要出去两个64kb分区,剩余的我们才能够使用

|  |
| --- |
|  |

## CPU的活动掩码就是用来决定当前正在工作的cpu有几个

## 掩码是一个32位的二进制数字，非常长，从低位算起，凡是工作的cpu对应的位就是1，不工作的对应0

|  |
| --- |
|  |

### 0011转化为16进制就是3.我的电脑有2个cpu，他们都在工作，所以看到是3.（因为我们用16进制来显示信息）

# API用法

## GetSystemInfo函数用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索有关当前系统的信息。  若要检索 WOW64 上运行的应用程序的准确信息，请调用 [GetNativeSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo) 函数。 语法 C++  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 要求  |  |  | | --- | --- | | **标头** | sysinfoapi.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetNativeSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo)  [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) |

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GetNativeSystemInfo 函数 (sysinfoapi.h) 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) |

## SYSTEM\_INFO结构体

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 包含有关当前计算机系统的信息。 这包括处理器的体系结构和类型、系统中的处理器数、页面大小和其他此类信息。 语法 C++  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\_ARM64**  12 | ARM64 | | **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。 |  要求  |  |  | | --- | --- | | **标头** | sysinfoapi.h (包括 Windows.h) |  另请参阅 [GetNativeSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getnativesysteminfo)  [GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsysteminfo)  [MapViewOfFile](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffile)  [MapViewOfFileEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/memoryapi/nf-memoryapi-mapviewoffileex) |

## \_tcscpy\_s宏的用法,在tchar.h里面声明

|  |
| --- |
|  |

## \_countof宏

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 计算静态分配的数组中元素的数量。 语法 C  #define \_countof(array) (sizeof(array) / sizeof(array[0])) 参数 *array* 数组的名称。 返回值 数组中的元素数，表示为 **size\_t**。 注解 **\_countof** 作为类似于函数的预处理器宏实现。 C++ 版本具有额外的模板机制，用于在编译时检测传递的是指针还是静态声明的数组。  确保 *array* 实际上是数组，而不是指针。 在 C 中，如果 *array* 是指针，则 **\_countof** 将生成错误结果。 在 C++ 中，如果 *array* 是指针，则 **\_countof** 将无法编译。 作为参数传递给函数的数组会衰减为指针，这意味着在函数中，不能使用 **\_countof** 确定数组的范围。 要求 展开表   | **宏** | **必需的标头** | | --- | --- | | **\_countof** | <stdlib.h> |  示例 C++  // crt\_countof.cpp  #define \_UNICODE  #include <stdio.h>  #include <stdlib.h>  #include <tchar.h>  int main( void )  {  \_TCHAR arr[20], \*p;  printf( "sizeof(arr) = %zu bytes\n", sizeof(arr) );  printf( "\_countof(arr) = %zu elements\n", \_countof(arr) );  // In C++, the following line would generate a compile-time error:  // printf( "%zu\n", \_countof(p) ); // error C2784 (because p is a pointer)  \_tcscpy\_s( arr, \_countof(arr), \_T("a string") );  // unlike sizeof, \_countof works here for both narrow- and wide-character strings  }  Output  sizeof(arr) = 40 bytes  \_countof(arr) = 20 elements 另请参阅 [sizeof 运算符](https://learn.microsoft.com/zh-cn/cpp/cpp/sizeof-operator?view=msvc-170) |

## StringCchPrintf函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将格式化的数据写入指定的字符串。 目标缓冲区的大小提供给函数，以确保它不会写入此缓冲区的末尾。  **StringCchPrintf** 取代了以下函数：   * [sprintf、swprintf、\_stprintf](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/sprintf-sprintf-l-swprintf-swprintf-l-swprintf-l) * [wsprintf](https://learn.microsoft.com/zh-cn/windows/desktop/api/winuser/nf-winuser-wsprintfa) * [wnsprintf](https://learn.microsoft.com/zh-cn/windows/desktop/api/shlwapi/nf-shlwapi-wnsprintfa) * [\_snprintf、\_snwprintf、\_sntprintf](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/reference/snprintf-snprintf-snprintf-l-snwprintf-snwprintf-l)  语法 C++  STRSAFEAPI StringCchPrintfA(  [out] STRSAFE\_LPSTR pszDest,  [in] size\_t cchDest,  [in] STRSAFE\_LPCSTR pszFormat,  ...  ); 参数 [out] pszDest  类型： **LPTSTR**  目标缓冲区，接收从 *pszFormat* 及其参数创建的格式化的以 null 结尾的字符串。  [in] cchDest  类型： **size\_t**  目标缓冲区的大小（以字符为单位）。 此值必须足够大，才能容纳最终格式化字符串加 1，以解释终止 null 字符。 允许的最大字符数是 **STRSAFE\_MAX\_CCH**。  [in] pszFormat  类型： **LPCTSTR**  格式字符串。 此字符串必须以 null 结尾。 有关详细信息，请参阅 [格式规范语法](https://learn.microsoft.com/zh-cn/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions)。  ...  要插入 *到 pszFormat* 字符串中的参数。 返回值 类型： **HRESULT**  此函数可以返回以下值之一。 强烈建议使用 [SUCCEEDED](https://learn.microsoft.com/zh-cn/windows/desktop/api/winerror/nf-winerror-succeeded) 和 [FAILED](https://learn.microsoft.com/zh-cn/windows/desktop/api/winerror/nf-winerror-failed) 宏来测试此函数的返回值。  展开表   |  |  | | --- | --- | | **返回代码** | **说明** | | **S\_OK** | 有足够的空间将结果复制到 *pszDest* 而不截断，并且缓冲区以 null 结尾。 | | **STRSAFE\_E\_INVALID\_PARAMETER** | *cchDest* 中的值为 0 或大于 **STRSAFE\_MAX\_CCH**。 | | **STRSAFE\_E\_INSUFFICIENT\_BUFFER** | 由于缓冲区空间不足，复制操作失败。 目标缓冲区包含预期结果的截断、以 null 结尾的版本。 在截断是可接受的情况下，这不一定被视为失败条件。 |     请注意，此函数返回 **HRESULT** 值，这与它所替换的函数不同。 注解 与它替换的函数相比， **StringCchPrintf** 为代码中的正确缓冲区处理提供了额外的处理。 不良的缓冲区处理与许多涉及缓冲区溢出的安全问题有关。 **StringCchPrintf** 始终为 null 终止非零长度的目标缓冲区。  如果 *pszDest*、 *pszFormat* 或任何参数字符串指向的字符串重叠，则行为未定义。  *pszFormat* 和 *pszDest* 都不应为 **NULL**。 如果需要处理 null 字符串指针值，请参阅 [StringCchPrintfEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/strsafe/nf-strsafe-stringcchprintfexa) 。  **StringCchPrintf** 可以在其泛型形式或更具体的形式中使用。 字符串的数据类型决定了应使用的此函数的形式。  展开表   |  |  |  | | --- | --- | --- | | **String 数据类型** | **字符串** | **函数** | | **char** | “字符串” | **StringCchPrintfA** | | **TCHAR** | TEXT (“string”) | **StringCchPrintf** | | **WCHAR** | L“string” | **StringCchPrintfW** |    示例 以下示例演示了使用四个参数的 **StringCchPrintf** 的简单用法。  C++复制  TCHAR pszDest[30];  size\_t cchDest = 30;  LPCTSTR pszFormat = TEXT("%s %d + %d = %d.");  TCHAR\* pszTxt = TEXT("The answer is");  HRESULT hr = StringCchPrintf(pszDest, cchDest, pszFormat, pszTxt, 1, 2, 3);  // The resultant string at pszDest is "The answer is 1 + 2 = 3."  **备注**  strsafe.h 标头将 StringCchPrintf 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** | strsafe.h |  另请参阅 **引用**  [StringCbPrintf](https://learn.microsoft.com/zh-cn/windows/desktop/api/strsafe/nf-strsafe-stringcbprintfa)  [StringCchPrintfEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/strsafe/nf-strsafe-stringcchprintfexa)  [StringCchVPrintf](https://learn.microsoft.com/zh-cn/windows/desktop/api/strsafe/nf-strsafe-stringcchvprintfa) |

## NUMBERFMT结构体

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 包含定义数字字符串格式的信息。 [GetNumberFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-getnumberformata) 函数使用此信息来自定义指定区域设置的数字字符串。 语法 C++复制  typedef struct \_numberfmtA {  UINT NumDigits;  UINT LeadingZero;  UINT Grouping;  LPSTR lpDecimalSep;  LPSTR lpThousandSep;  UINT NegativeOrder;  } NUMBERFMTA, \*LPNUMBERFMTA; 成员 NumDigits  小数位数。 此值等效于值 [LOCALE\_IDIGITS](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-idigits) 指定的区域设置信息。  LeadingZero  一个值，该值指示是否应在十进制字段中使用前导零。 此值等效于值 [LOCALE\_ILZERO](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-ilzero) 指定的区域设置信息。  Grouping  **lpDecimalSep** 指定的小数分隔符左侧每组数字中的位数。 0 到 9 和 32 范围内的值有效。 最有效分组数字表示紧接在小数分隔符左侧的最小有效组中的位数。 每个后续分组数字都表示上一组左侧的下一个重要数字组。 如果提供的最后一个值不为 0，则其余组将重复最后一个组。 此成员设置的典型示例如下：0 将数字分组为 123456789.00;3 将数字分组为 123，456，789.00;和 32 将数字分组为 12，34，56，789.00。  **注意**可以使用典型设置以外的设置，但它们不会显示在控制面板的区域和语言选项部分中。 此类设置极不常见，可能会产生意外结果。    lpDecimalSep  指向以 null 结尾的十进制分隔符字符串的指针。  lpThousandSep  指向以 null 结尾的千分隔符字符串的指针。  NegativeOrder  负数模式。 此模式等效于值 [LOCALE\_INEGNUMBER](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-ineg-constants) 指定的区域设置信息。 注解 **备注**  winnls.h 标头将 NUMBERFMT 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 展开表   |  |  | | --- | --- | | **最低受支持的客户端** | Windows 2000 专业版 [桌面应用 |UWP 应用] | | **最低受支持的服务器** | Windows 2000 Server [桌面应用 |UWP 应用] | | **标头** | winnls.h (包括 Windows.h) |  另请参阅 [GetNumberFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-getnumberformata)  [国家语言支持结构](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/national-language-support-structures) |

## GetNumberFormat函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 将数字字符串的格式设置为为标识符指定的区域设置自定义的数字字符串。  **注意** 出于互操作性原因，应用程序应优先使用 [**GetNumberFormatEx**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-getnumberformatex) 函数而不是 **GetNumberFormat** ，因为 Microsoft 正在迁移到使用区域设置名称而不是新区域设置的区域设置标识符。 仅在 Windows Vista 及更高版本上运行的任何应用程序都应使用 **GetNumberFormatEx**。   语法 C++复制  int GetNumberFormatA(  [in] LCID Locale,  [in] DWORD dwFlags,  [in] LPCSTR lpValue,  [in, optional] const NUMBERFMTA \*lpFormat,  [out, optional] LPSTR lpNumberStr,  [in] int cchNumber  ); 参数 [in] Locale  指定区域设置的区域设置[标识符](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-identifiers)。 可以使用 [MAKELCID](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnt/nf-winnt-makelcid) 宏创建区域设置标识符或使用以下预定义值之一。   * [LOCALE\_CUSTOM\_DEFAULT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-custom-constants) * [LOCALE\_CUSTOM\_UI\_DEFAULT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-custom-constants) * [LOCALE\_CUSTOM\_UNSPECIFIED](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-custom-constants) * [LOCALE\_INVARIANT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-invariant) * [LOCALE\_SYSTEM\_DEFAULT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-system-default) * [LOCALE\_USER\_DEFAULT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-user-default)   [in] dwFlags  控制函数操作的标志。 如果 *lpFormat* 未设置为 **NULL**，则应用程序必须将此参数设置为 0。 在这种情况下， 函数使用用户替代将字符串的格式设置为区域设置的默认数字格式。 如果 *lpFormat* 设置为 **NULL**，则应用程序可以指定 [LOCALE\_NOUSEROVERRIDE](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-nouseroverride) ，以使用指定区域设置的系统默认数字格式设置字符串的格式。  **谨慎** 强烈建议不要使用 LOCALE\_NOUSEROVERRIDE，因为它会禁用用户首选项。    [in] lpValue  指向以 null 结尾的字符串的指针，该字符串包含要设置格式的数字字符串。 此字符串只能包含以下字符。 所有其他字符都无效。 如果 *lpValue* 指示的字符串偏离这些规则，则函数将返回错误。   * 字符“0”到“9”。 * 如果数字是浮点值，则一个小数点 (点) 。 * 如果数字为负值，则为第一个字符位置的减号。   [in, optional] lpFormat  指向 [NUMBERFMT](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/ns-winnls-numberfmta) 结构的指针，该结构包含数字格式设置信息，所有成员都设置为适当的值。 如果此参数未设置为 **NULL**，则该函数仅使用区域设置来设置结构中未指定的格式信息，例如，负号的区域设置特定字符串值。  [out, optional] lpNumberStr  指向缓冲区的指针，此函数在其中检索带格式的数字字符串。  [in] cchNumber  大小，以 TCHAR 值表示，表示 *由 lpNumberStr* 指示的数字字符串缓冲区。 或者，应用程序可以将此参数设置为 0。 在这种情况下，函数返回数字字符串缓冲区所需的大小，并且不使用 *lpNumberStr* 参数。 返回值 如果成功，则返回 *在 lpNumberStr* 指示的缓冲区中检索的 TCHAR 值的数目。 如果 *cchNumber* 参数设置为 0，则该函数将返回保存格式化数字字符串所需的字符数，包括终止 null 字符。  如果函数不成功，则返回 0。 若要获取扩展的错误信息，应用程序可以调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)，这会返回以下错误代码之一：   * ERROR\_INSUFFICIENT\_BUFFER。 提供的缓冲区大小不够大，或者错误地设置为 **NULL**。 * ERROR\_INVALID\_FLAGS。 为标志提供的值无效。 * ERROR\_INVALID\_PARAMETER。 任何参数值都无效。 * ERROR\_OUTOFMEMORY。 没有足够的存储空间来完成此操作。  注解 此函数可以从 [自定义区域设置](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/custom-locales)检索数据。 不保证数据在计算机之间或应用程序运行之间的数据相同。 如果应用程序必须保留或传输数据，请参阅 [使用持久区域设置数据](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/using-persistent-locale-data)。  当此函数的 ANSI 版本与仅限 Unicode 的区域设置标识符一起使用时，该函数可以成功，因为操作系统使用系统代码页。 但是，系统代码页中未定义的字符在字符串中显示为问号 (？) 。  **备注**  winnls.h 标头将 GetNumberFormat 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 展开表   | **要求** | **值** | | --- | --- | | **最低受支持的客户端** | Windows 2000 Professional [仅限桌面应用] | | **最低受支持的服务器** | Windows 2000 Server [仅限桌面应用] | | **目标平台** | Windows | | **标头** | winnls.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetNumberFormatEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-getnumberformatex)  [NUMBERFMT](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/ns-winnls-numberfmta)  [国家/地区语言支持](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/national-language-support)  [国家/地区语言支持函数](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/national-language-support-functions) |

## GetNumberFormatEx 函数用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 将数字字符串的格式设置为为名称指定的区域设置自定义的数字字符串。  **注意** 如果设计为仅在 Windows Vista 及更高版本上运行，则应用程序应优先调用 [**GetNumberFormat**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-getnumberformata) 此函数。    **注意** 此函数可以设置不同版本之间更改的数据的格式，例如，由于自定义区域设置。 如果应用程序必须保留或传输数据，请参阅 [**使用永久性区域设置数据**](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/using-persistent-locale-data)。   语法 C++复制  int GetNumberFormatEx(  [in, optional] LPCWSTR lpLocaleName,  [in] DWORD dwFlags,  [in] LPCWSTR lpValue,  [in, optional] const NUMBERFMTW \*lpFormat,  [out, optional] LPWSTR lpNumberStr,  [in] int cchNumber  ); 参数 [in, optional] lpLocaleName  指向 [区域设置名称](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-names)或以下预定义值之一的指针。   * [LOCALE\_NAME\_INVARIANT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-name-constants) * [LOCALE\_NAME\_SYSTEM\_DEFAULT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-name-constants) * [LOCALE\_NAME\_USER\_DEFAULT](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-name-constants)   [in] dwFlags  控制函数操作的标志。 如果 *lpFormat* 未设置为 **NULL**，则应用程序必须将此参数设置为 0。 在这种情况下，函数使用用户设置字符串的格式将替代为区域设置的默认数字格式。 如果 *lpFormat* 设置为 **NULL**，则应用程序可以指定 [LOCALE\_NOUSEROVERRIDE](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/locale-nouseroverride) ，以使用指定区域设置的系统默认数字格式设置字符串的格式。  **谨慎** 强烈建议不要使用LOCALE\_NOUSEROVERRIDE，因为它会禁用用户首选项。    [in] lpValue  指向以 null 结尾的字符串的指针，其中包含要设置格式的数字字符串。 此字符串只能包含以下字符。 所有其他字符都无效。 如果 *lpValue* 指示的字符串偏离这些规则，则函数将返回错误。   * 字符“0”到“9”。 * 如果数字是浮点值，则一个小数点 (点) 。 * 如果数字为负值，则为第一个字符位置的减号。   [in, optional] lpFormat  指向 [NUMBERFMT](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/ns-winnls-numberfmta) 结构的指针，该结构包含数字格式设置信息，所有成员都设置为适当的值。 如果应用程序未将此参数设置为 **NULL**，则该函数仅将区域设置用于设置结构中未指定的信息的格式，例如，负号的区域设置字符串值。  [out, optional] lpNumberStr  指向缓冲区的指针，此函数在其中检索格式化的数字字符串。 或者，如果 *cchNumber* 设置为 0，则此参数包含 **NULL**。 在这种情况下，函数返回数字字符串缓冲区所需的大小。  [in] cchNumber  *lpNumberStr* 指示的数字字符串缓冲区的大小（以字符为单位）。 或者，应用程序可以将此参数设置为 0。 在这种情况下，函数返回数字字符串缓冲区所需的大小，并且不使用 *lpNumberStr* 参数。 返回值 如果成功，则返回 *在 lpNumberStr* 指示的缓冲区中检索到的字符数。 如果 *cchNumber* 参数设置为 0，则该函数将返回保存格式化数字字符串所需的字符数，包括终止 null 字符。  如果函数不成功，则返回 0。 若要获取扩展错误信息，应用程序可以调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)，这会返回以下错误代码之一：   * ERROR\_INSUFFICIENT\_BUFFER。 提供的缓冲区大小不够大，或者错误地设置为 **NULL**。 * ERROR\_INVALID\_FLAGS。 为标志提供的值无效。 * ERROR\_INVALID\_PARAMETER。 任何参数值都无效。 * ERROR\_OUTOFMEMORY。 没有足够的存储空间可用于完成此操作。  注解 **从Windows 8开始：**如果你的应用将语言标记从 [Windows.Globalization](https://learn.microsoft.com/zh-cn/uwp/api/Windows.Globalization) 命名空间传递到此函数，它必须首先通过调用 [ResolveLocaleName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-resolvelocalename) 来转换标记。 要求  |  |  | | --- | --- | | **标头** | winnls.h (包括 Windows.h) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [GetNumberFormat](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/nf-winnls-getnumberformata)  [NUMBERFMT](https://learn.microsoft.com/zh-cn/windows/desktop/api/winnls/ns-winnls-numberfmta)  [国家语言支持](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/national-language-support)  [国家语言支持函数](https://learn.microsoft.com/zh-cn/windows/desktop/Intl/national-language-support-functions) |

# 专用名称

## allocation granularity 分配粒度

# 演练这一节做cpu信息

## 1.新建一个文件夹,取名:Lesson80-cpu-info,然后在这里新建一个常规空项目,取名:cpu-info,然后在资源文件夹里面新建一个对话框资源,然后添加一些显示cpu信息的文件文本方式编辑框,布局如下

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

## 2.给项目添加一个cpp源文件，取名cpuinfo.cpp,代码如下，注意cpu信息有2个是很简单的，我们可用直接设置到控件上面，但是又一个信息的关于cpu的型号的比较复杂，我们需要写一个函数来显示

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<tchar.h>  #include<strsafe.h> //这个头文件需要在tchar.h后面写,否则保存  #include"resource.h"  PTSTR BigNumToStr(long lNum,LPSTR szBuf,DWORD bufSize);  INT\_PTR CALLBACK DlgProc(HWND hdlg,UINT msg,WPARAM wParam ,LPARAM lParam);  void ShowCpuTypeInfo(HWND hdlg,WORD wArch,WORD wLevel,WORD wRevi);  int WINAPI 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\_INITDIALOG:  //MessageBox(hdlg,TEXT("hello"),TEXT("TEST"),0);  //获取内存信息并且设置到编辑框上面  SYSTEM\_INFO sinf;  GetNativeSystemInfo(&sinf);//64位系统需要使用这个函数    TCHAR szmask[256];//  //为了更好的显示信息我们把他们转化为字符串并且设置地区显示格式,注意这里我们使用微软推荐的安全字符串函数  //StringCchPrintf取代了sprinft,wsprintf,swprintf,stprintf,wnsprintf,snprintf等待函数  StringCchPrintf(szmask,\_countof(szmask),TEXT("0x%016I64X"),(\_\_int64)sinf.dwActiveProcessorMask);//掩码用16进制来表示  SetDlgItemText(hdlg,IDC\_EDIT\_MASK,szmask);  SetDlgItemInt(hdlg,IDC\_EDIT\_CPU\_NUM,sinf.dwNumberOfProcessors,TRUE);  //显示cpu型号信息的自定义函数  ShowCpuTypeInfo(hdlg,sinf.wProcessorArchitecture,sinf.wProcessorLevel,sinf.wProcessorRevision);  break;  case WM\_CLOSE:  EndDialog(hdlg,FALSE);  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;  default:  ret = FALSE;//这里需要设置位FALSE让默认对话框过程处理我们不感兴趣的信息  break;  }  return ret;  }  //使得数字显示格式位1,000,000.00的格式  PTSTR BigNumToStr(long lNum,LPSTR szBuf,DWORD bufSize)  {  TCHAR szNum[100];  StringCchPrintf(szNum,sizeof(szNum),TEXT("%d"),lNum);  NUMBERFMT nf;//数字格式结构体,使用前需要初始化  nf.NumDigits = 0;  nf.LeadingZero = FALSE;  nf.Grouping =3;  nf.lpDecimalSep = TEXT(".");//设置小数点分割符  nf.lpThousandSep = TEXT(",");//设置千分位分隔符  nf.NegativeOrder = 0;  GetNumberFormat(LOCALE\_USER\_DEFAULT,0,szNum,&nf,szBuf,bufSize);//设置格式为1,000,000.00  return szBuf;  }  //显示CPU型号信息的自定义函数  void ShowCpuTypeInfo(HWND hdlg,WORD wArch,WORD wLevel,WORD wRevi)  {  TCHAR szarch[64] = TEXT("unknown");//processor architecture  TCHAR szlevel[64] = TEXT("unknown");//processor level  TCHAR szrevi[64] = TEXT("unknown");//processor revision  //处理信息  switch(wArch)  {  case PROCESSOR\_ARCHITECTURE\_INTEL:  \_tcscpy\_s(szarch,\_countof(szarch),TEXT("Intel"));//tchar.h  switch(wLevel)  {  case 3:  case 4:  StringCchPrintf(szlevel,\_countof(szlevel),TEXT("80%c86"),wLevel+'0');  StringCchPrintf(szrevi,\_countof(szrevi),TEXT("%c%d"),HIBYTE(wRevi)+TEXT('A'),LOBYTE(wRevi));  break;  case 5:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Pentium"));  StringCchPrintf(szrevi,\_countof(szrevi),TEXT("Model %d Stepping %d"),HIBYTE(wRevi),LOBYTE(wRevi));  break;  case 6:  switch(HIBYTE(wRevi))  {  case 1:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Pentium pro"));  break;  case 3:  case 5:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Pentium II"));  break;  case 6:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Celeron"));  break;  case 7:  case 8:  case 11:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Pentium III"));  break;  case 9:  case 13:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Pentium M"));  break;  case 10:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Pentium Xeon"));  break;  case 15:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Core 2 Duo"));  break;  default:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Core 2 Duo"));  break;  }  StringCchPrintf(szrevi,\_countof(szrevi),TEXT("Model %d Stepping %d"),HIBYTE(wRevi),LOBYTE(wRevi));  break;  case 15:  \_tcscpy\_s(szlevel,\_countof(szlevel),TEXT("Unknown Pentium"));  StringCchPrintf(szrevi,\_countof(szrevi),TEXT("Model %d Stepping %d"),HIBYTE(wRevi),LOBYTE(wRevi));  break;  }  break;  case PROCESSOR\_ARCHITECTURE\_AMD64:  \_tcscpy\_s(szarch,\_countof(szarch),TEXT("AMD64"));  StringCchPrintf(szlevel,\_countof(szlevel),TEXT("%d"),wLevel);  StringCchPrintf(szrevi,\_countof(szrevi),TEXT("Model %d Stepping %d"),HIBYTE(wRevi)+TEXT('A'),LOBYTE(wRevi));  break;  case PROCESSOR\_ARCHITECTURE\_IA64:  \_tcscpy\_s(szarch,\_countof(szarch),TEXT("IA64"));  StringCchPrintf(szlevel,\_countof(szlevel),TEXT("%d"),wLevel);  StringCchPrintf(szrevi,\_countof(szrevi),TEXT("Model %d Stepping %d"),HIBYTE(wRevi)+TEXT('A'),LOBYTE(wRevi));  break;  case PROCESSOR\_ARCHITECTURE\_UNKNOWN:  default:  \_tcscpy\_s(szarch,\_countof(szarch),TEXT("未知型号CPU"));  break;  }    //输出信息  SetDlgItemText(hdlg,IDC\_EDIT\_ARCH,szarch);  SetDlgItemText(hdlg,IDC\_EDIT\_LEVEL,szlevel);  SetDlgItemText(hdlg,IDC\_EDIT\_REVI,szrevi);  } |

## 上面就是完整代码

### 效果

|  |
| --- |
|  |

# 这一节学习到此为止。完整代码在上面