# 这一节，我们来学习获取时间消息

# 学习大纲

|  |
| --- |
|  |

## GetLocalTime函数用法

|  |
| --- |
| C++复制  void GetLocalTime(  [out] LPSYSTEMTIME lpSystemTime  ); 参数 [out] lpSystemTime  指向 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime) 结构的指针，用于接收当前本地日期和时间。 返回值 无 备注 若要设置当前本地日期和时间，请使用 [SetLocalTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-setlocaltime) 函数。 示例 有关示例，请参阅 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime)。 |

## SetLocalTime函数的语法

|  |
| --- |
| C++复制  BOOL SetLocalTime(  [in] const SYSTEMTIME \*lpSystemTime  ); 参数 [in] lpSystemTime  指向包含新本地日期和时间的 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime) 结构的指针。  将忽略 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime) 结构的 **wDayOfWeek** 成员。 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 调用进程必须具有SE\_SYSTEMTIME\_NAME特权。 默认情况下，此权限处于禁用状态。 **SetLocalTime** 函数在更改本地时间之前启用SE\_SYSTEMTIME\_NAME特权，并在返回之前禁用该特权。 有关详细信息，请参阅[使用特殊特权运行](https://learn.microsoft.com/zh-cn/windows/desktop/SecBP/running-with-special-privileges)。  系统在内部使用 UTC。 因此，调用 **SetLocalTime** 时，系统会使用当前时区信息来执行转换，包括夏令时设置。 请注意，系统使用当前时间的夏令时设置，而不是设置的新时间。 因此，为确保结果正确，请再次调用 **SetLocalTime** ，因为第一次调用已更新夏令时设置。 |

## SYSTEMTIME结构体

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指定日期和时间，使用月份、日、年、工作日、小时、分钟、秒和毫秒的单个成员。 时间采用协调世界时 (UTC) 或本地时间，具体取决于正在调用的函数。 语法 C++复制  typedef struct \_SYSTEMTIME {  WORD wYear;  WORD wMonth;  WORD wDayOfWeek;  WORD wDay;  WORD wHour;  WORD wMinute;  WORD wSecond;  WORD wMilliseconds;  } SYSTEMTIME, \*PSYSTEMTIME, \*LPSYSTEMTIME; 成员 wYear  年。 此成员的有效值为 1601 到 30827。  wMonth  月份。 此成员可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | 1 | 1 月 | | 2 | February | | 3 | 3 月 | | 4 | April | | 5 | 五月 | | 6 | June | | 7 | 7 月 | | 8 | 8 月 | | 9 | 9 月 | | 10 | 10 月 | | 11 | 11 月 | | 12 | 12 月 |   wDayOfWeek  星期几。 此成员可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | 0 | 星期日 | | 1 | 星期一 | | 2 | 星期二 | | 3 | 星期三 | | 4 | 星期四 | | 5 | 星期五 | | 6 | 星期六 |   wDay  每月的日期。 此成员的有效值为 1 到 31。  wHour  小时。 此成员的有效值为 0 到 23。  wMinute  分钟。 此成员的有效值为 0 到 59。  wSecond  秒钟。 此成员的有效值为 0 到 59。  wMilliseconds  毫秒。 此成员的有效值为 0 到 999。 注解 **备注**  **SYSTEMTIME** 不检查查看表示的日期是否为实际有效日期。 使用此 API 时，应确保其有效性，尤其是在闰年方案中。 有关详细信息 [**，请参阅闰日就绪情况**](https://techcommunity.microsoft.com/t5/azure-developer-community-blog/it-s-2020-is-your-code-ready-for-leap-day/ba-p/1157279) 。  建议不要从 **SYSTEMTIME** 结构中添加和减去值来获取相对时间。 相反，你应该   * 将 **SYSTEMTIME** 结构转换为 [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构。 * 将生成的 [FILETIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-filetime) 结构复制到 [ULARGE\_INTEGER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-ularge_integer-r1) 结构。 * 对 [ULARGE\_INTEGER](https://learn.microsoft.com/zh-cn/windows/win32/api/winnt/ns-winnt-ularge_integer-r1) 值使用正常的 64 位算术。   系统可以通过与时间源同步来定期刷新时间。 由于系统时间可以向前或向后调整，因此请勿比较系统时间读数以确定已用时间。 请改用 [Windows 时间](https://learn.microsoft.com/zh-cn/windows/desktop/SysInfo/windows-time)中所述的方法之一。 |

## GetTickCount函数用法

|  |
| --- |
| DWORD GetTickCount(); 返回值 返回值是自系统启动以来经过的毫秒数。 言论 **GetTickCount** 函数的分辨率限制为系统计时器的分辨率，通常范围为 10 毫秒到 16 毫秒。 **GetTickCount** 函数的解析不受 [GetSystemTimeAdjustment](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemtimeadjustment) 函数所做的调整的影响。  已用时间存储为 **DWORD** 值。 因此，如果系统连续运行 49.7 天，时间将环绕到零。 若要避免此问题，请使用 [GetTickCount64](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-gettickcount64) 函数。 否则，在比较时间时检查溢出条件。  如果需要更高分辨率的计时器，请使用 [多媒体计时器](https://learn.microsoft.com/zh-cn/windows/desktop/Multimedia/multimedia-timers) 或 [高分辨率计时器](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/about-timers)。  若要获取计算机启动后经过的时间，请检索注册表项中性能数据中的系统 Up Time 计数器 **HKEY\_PERFORMANCE\_DATA**。 返回的值是一个 8 字节值。 有关详细信息，请参阅 [性能计数器](https://learn.microsoft.com/zh-cn/windows/desktop/PerfCtrs/performance-counters-portal)。  若要获取系统自启动以来在工作状态下花费的时间，请使用 [QueryUnbiasedInterruptTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/realtimeapiset/nf-realtimeapiset-queryunbiasedinterrupttime) 函数。  **注释**[**QueryUnbiasedInterruptTime**](https://learn.microsoft.com/zh-cn/windows/desktop/api/realtimeapiset/nf-realtimeapiset-queryunbiasedinterrupttime) 函数在 Windows 的调试（“checked”）生成时生成不同的结果，因为中断时间计数和时钟周期计数在大约 49 天内进行高级。 这有助于识别在系统长时间运行之前可能不会发生的 bug。   例子 C++复制  // calculate a 't' value that will linearly interpolate from 0 to 1 and back every 20 seconds  DWORD currentTime = GetTickCount();  if ( m\_startTime == 0 )  {  m\_startTime = currentTime;  }  float t = 2 \* (( currentTime - m\_startTime) % 20000) / 20000.0f;  if (t > 1.0f)  {  t = 2 - t;  } |

## GetTickCount64函数用法

|  |
| --- |
| C++复制  ULONGLONG GetTickCount64(); 返回值 毫秒数。 注解 **GetTickCount64** 函数的分辨率限制为系统计时器的分辨率，通常介于 10 毫秒到 16 毫秒之间。 **GetTickCount64** 函数的分辨率不受 [GetSystemTimeAdjustment](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-getsystemtimeadjustment) 函数所做的调整的影响。  如果需要更高分辨率的计时器，请使用 [多媒体计时器](https://learn.microsoft.com/zh-cn/windows/desktop/Multimedia/multimedia-timers) 或 [高分辨率计时器](https://learn.microsoft.com/zh-cn/windows/desktop/winmsg/about-timers)。  若要获取系统自启动以来处于工作状态的时间，请使用 [QueryUnbiasedInterruptTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/realtimeapiset/nf-realtimeapiset-queryunbiasedinterrupttime) 函数。  **注意**[**QueryUnbiasedInterruptTime**](https://learn.microsoft.com/zh-cn/windows/desktop/api/realtimeapiset/nf-realtimeapiset-queryunbiasedinterrupttime) 函数在调试 (“checked”) Windows 内部版本上生成不同的结果，因为中断时间计数和时钟周期计数将提前大约 49 天。 这有助于识别在系统长时间运行之前可能不会发生的 bug。 选中的版本可通过 [**Microsoft 开发人员网络 (MSDN) 网站向 MSDN**](https://msdn.microsoft.com/default.aspx)订阅者提供。    若要编译使用此函数的应用程序，请将\_WIN32\_WINNT定义为 0x0600 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)。 |

# extra

## GetSystemTime 函数用法

|  |
| --- |
| void GetSystemTime(  [out] LPSYSTEMTIME lpSystemTime  ); 参数 [out] lpSystemTime  指向 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime) 结构的指针，用于接收当前系统日期和时间。 *lpSystemTime* 参数不得为 **NULL**。 使用 **NULL** 将导致访问冲突。 返回值 无 备注 若要设置当前系统日期和时间，请使用 [SetSystemTime](https://learn.microsoft.com/zh-cn/windows/desktop/api/sysinfoapi/nf-sysinfoapi-setsystemtime) 函数。 示例 C++复制  #include <windows.h>  #include <stdio.h>  void main()  {  SYSTEMTIME st, lt;    GetSystemTime(&st);  GetLocalTime(&lt);    printf("The system time is: %02d:%02d\n", st.wHour, st.wMinute);  printf(" The local time is: %02d:%02d\n", lt.wHour, lt.wMinute);  }  dos复制  // Sample output  The system time is: 19:34  The local time is: 12:34  有关此示例的详细信息，请参阅 [SYSTEMTIME](https://learn.microsoft.com/zh-cn/windows/desktop/api/minwinbase/ns-minwinbase-systemtime)。 |

# 演练

## 1.新建一个c++常规空项目，取名：Lesson25-get-set-time，然后新建一个cpp源文件，取名：get-set-time.cpp,然后添加骨架代码

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  int main()  {  system("pause");  return 0;  } |

## 2.然后我们可以添加如下代码，获取系统时间本地时间和从开机到现在持续时间，还可以使用GetTickCount函数的结果来作为随机数种子生成一个随机数数组，这里我们还学习到了一个qsort的c语言排序函数，它的最后一个参数是一个回调函数，指定排序的规则，代码如下：

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #define ArrSize 10  int cmp(const void \*a,const void \*b) //c 语言的笔记回调函数  {  return \*(int \*)a - \*(int \*)b;  }  int main()  {  //获取当前系统时间和当前本地时间，他们的小时数是不一样的  SYSTEMTIME st,localSt;  GetSystemTime(&st);  printf("系统时间：%d年-%#02d月-%#02d日 %#02d-%#02d-%#02d\n",  st.wYear,st.wMonth,st.wDay,st.wHour,st.wMinute,st.wSecond);  GetLocalTime(&localSt);  printf("本地时间：%d年-%#02d月-%#02d日 %#02d-%#02d-%#02d\n",  localSt.wYear,localSt.wMonth,localSt.wDay,localSt.wHour,localSt.wMinute,localSt.wSecond);  //使用2个函数获取从开机到现在持续的时间  DWORD c1 = GetTickCount();//电脑持续开机超过49天后，这个函数就得不到正确结果  ULONGLONG l1 = GetTickCount64();  printf("GetTickCount：%d\n",c1);  printf("GetTickCount64：%d\n",l1);  //可以使用GetTickCount函数来获取随机数  //如我们这里输出10个随机数  int randNum[ArrSize];  printf("Before Sort\n");  srand(GetTickCount());//设置随机数种子  for (int i=0;i<ArrSize;i++)  {  randNum[i]=rand()%100;//注意这里%100表示对100取余，也就是取100以内的随机数  printf("%d: %d\n",i,randNum[i]);  }  //利用qsort函数来对数组元素排序  printf("After Sort\n");  qsort(randNum,ArrSize,sizeof(randNum[0]),cmp);  for (int i=0;i<ArrSize;i++)  {  printf("%d: %d\n",i,randNum[i]);  }  system("pause");  return 0;  } |

### 效果：

|  |
| --- |
|  |

## 3.我们还可以用GetTickCount来测试一些执行qsort函数对数组进行排序需要多少时间，我们把代码修改下

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #define ArrSize 100000  int cmp(const void \*a,const void \*b) //c 语言的笔记回调函数  {  return \*(int \*)a - \*(int \*)b;  }  int main()  {  //获取当前系统时间和当前本地时间，他们的小时数是不一样的  SYSTEMTIME st,localSt;  GetSystemTime(&st);  printf("系统时间：%d年-%#02d月-%#02d日 %#02d-%#02d-%#02d\n",  st.wYear,st.wMonth,st.wDay,st.wHour,st.wMinute,st.wSecond);  GetLocalTime(&localSt);  printf("本地时间：%d年-%#02d月-%#02d日 %#02d-%#02d-%#02d\n",  localSt.wYear,localSt.wMonth,localSt.wDay,localSt.wHour,localSt.wMinute,localSt.wSecond);  //使用2个函数获取从开机到现在持续的时间  DWORD c1 = GetTickCount();//电脑持续开机超过49天后，这个函数就得不到正确结果  ULONGLONG l1 = GetTickCount64();  printf("GetTickCount：%d\n",c1);  printf("GetTickCount64：%d\n",l1);  //可以使用GetTickCount函数来获取随机数  //如我们这里输出10个随机数  int randNum[ArrSize];  //printf("Before Sort\n");  srand(GetTickCount());//设置随机数种子  for (int i=0;i<ArrSize;i++)  {  randNum[i]=rand()%1000;//注意这里%100表示对100取余，也就是取100以内的随机数  //printf("%d: %d\n",i,randNum[i]);  }  //利用qsort函数来对数组元素排序  //printf("After Sort\n");  DWORD start,end;  start = GetTickCount();  qsort(randNum,ArrSize,sizeof(randNum[0]),cmp);  /\*for (int i=0;i<ArrSize;i++)  {  printf("%d: %d\n",i,randNum[i]);  }\*/  end = GetTickCount();  printf("给%d个数组元素排序需要时间：%d毫秒\n",ArrSize,end - start);  system("pause");  return 0;  } |

## 效果：

|  |
| --- |
|  |

# 这一节学习到此为止。上面就是完整代码