**[原创图文]  
Windows API学习 三种版本 字符转换 遍历卷 磁盘属性 磁盘容量**

# 1.三种版本： ASCII版               宽字符版            通用版本(实际上是一个宏，会根据我们项目字符集，变成A版，或者W版)    CHAR                  WCHAR            TCHAR PCHAR                PWCHAR          PTCHAR ...                         ...                        ... MessageBoxA   MessageBoxW   MessageBox ...                       ...                        ...

# 2.字符转换

int main()

{

    setlocale(LC\_ALL, "");//影响所有，使用当前系统默认地域设置

    DWORD dword;

    char c[] = "多字节字符串转换成宽字符串";

    dword = MultiByteToWideChar(CP\_ACP, 0, c, -1, NULL, 0);//先求出需要多大的空间

    wchar\_t\* wc = new wchar\_t[dword];//就用多大的空间

    MultiByteToWideChar(CP\_ACP, 0, c, -1, wc, dword);//在次进行转换

    wprintf(L"%s\n", wc);

    //-------------------------------------------------

    wchar\_t c1[] = L"宽字符串转换成多字节字符串";

    dword = WideCharToMultiByte(CP\_OEMCP, 0, c1, -1, NULL, 0, NULL, NULL);//先求出需要多大的空间

    char\* wc1 = new char[dword];//就用多大的空间

    WideCharToMultiByte(CP\_ACP, 0, c1, -1, wc1, dword, NULL, NULL);//在次进行转换

    printf("%s", wc1);

}

**效果**

|  |
| --- |
|  |

**MultiByteToWideChar**:多字节字符串转换成宽字符串

int MultiByteToWideChar(

    UINT                              CodePage,

    DWORD                             dwFlags,

    \_In\_NLS\_string\_(cbMultiByte)LPCCH lpMultiByteStr,

    int                               cbMultiByte,

    LPWSTR                            lpWideCharStr,

    int                               cchWideChar

);

CodePage--指定执行转换的字符集（要转换的是多字节的用CP\_ACP，要转换的是宽字符的用CP\_OEMCP）

dwFlags--为选项，一般用 0 就可以了  
lpMultiByteStr--指向将被转换的字符串  
cbMultiByte--指定由参数lpMultiByteStr指向的字符串中[字节](https://www.xuenixiang.com/study_linkkiller-link.html?url=https%3A%2F%2Fbaike.baidu.com%2Fitem%2F%25E5%25AD%2597%25E8%258A%2582)的个数。如果lpMultiByteStr指定的字符串以空字符终止，可以设置为-1（如果字符串不是以空字符中止，设置为-1可能失败，可能成功，-1表示全部），此参数设置为0函数将失败。  
lpWideCharStr--指向接收被转换字符串的[缓冲区](https://www.xuenixiang.com/study_linkkiller-link.html?url=https%3A%2F%2Fbaike.baidu.com%2Fitem%2F%25E7%25BC%2593%25E5%2586%25B2%25E5%258C%25BA)  
cchWideChar--指定由参数lpWideCharStr指向的缓冲区的宽字符个数。若此值为零，函数返回缓冲区所必需的[宽字符](https://www.xuenixiang.com/study_linkkiller-link.html?url=https%3A%2F%2Fbaike.baidu.com%2Fitem%2F%25E5%25AE%25BD%25E5%25AD%2597%25E7%25AC%25A6)数，在这种情况下，lpWideCharStr中的缓冲区不被使用。

**WideCharToMultiByte**:宽字符串转换成多字节字符串转

    int WideCharToMultiByte(

        UINT                               CodePage,

        DWORD                              dwFlags,

        \_In\_NLS\_string\_(cchWideChar)LPCWCH lpWideCharStr,

        int                                cchWideChar,

        LPSTR                              lpMultiByteStr,

        int                                cbMultiByte,

        LPCCH                              lpDefaultChar,

        LPBOOL                             lpUsedDefaultChar

    );

CodePage--指定执行转换的字符集（要转换的是多字节的用CP\_ACP，要转换的是宽字符的用CP\_OEMCP）  
dwFlags--为选项，一般用 0 就可以了  
lpWideCharStr--指向将被转换的字符串  
cchWideChar--要转换的长度，-1表示全部都转换  
lpMultiByteStr--指向接收转换字符串的缓冲区的指针。  
cbMultiByte--由指示的缓冲区的大小（以字节为单位）lpMultiByteStr.如果此参数设置为0，则函数将返回所需的缓冲区大小lpMultiByteStr并且不使用输出参数本身。  
lpDefaultChar--指向在指定代码页中无法表示字符时要使用的字符的指针。应用程序将此参数设置为**无效的**如果函数是使用系统默认值。要获取系统默认字符，应用程序可以调用[color=var(--theme-visited)][GetCPInfo](https://www.xuenixiang.com/study_linkkiller-link.html?url=https%3A%2F%2Fdocs.microsoft.com%2Fen-us%2Fwindows%2Fdesktop%2Fapi%2Fwinnls%2Fnf-winnls-getcpinfo)或[color=var(--theme-visited)][GetCPInfoEx](https://www.xuenixiang.com/study_linkkiller-link.html?url=https%3A%2F%2Fdocs.microsoft.com%2Fen-us%2Fwindows%2Fdesktop%2Fapi%2Fwinnls%2Fnf-winnls-getcpinfoexa)功能对于的CP\_UTF7和CP\_UTF8设置*CodePage*，此参数必须设置为**无效的**. 否则，函数将失败，错误为“无效的”参数。  
lpUsedDefaultChar--指向指示函数是否在转换中使用了默认字符的标志的指针。标志设置为**真的**如果源字符串中的一个或多个字符无法在指定的代码页中表示。否则，标志设置为**错误的**. 此参数可以设置为**无效的** .对于的CP\_UTF7和CP\_UTF8设置*CodePage*，此参数必须设置为**无效的**. 否则，函数将失败，错误为“无效的”参数。

# 3.遍历卷

int main()

{

CHAR str[128] = {0};

    //获取遍历卷

    GetLogicalDriveStrings(sizeof(str), str);

    PCHAR pStr = str;

    do

    {

        cout << pStr<<endl;

        pStr += (lstrlen(pStr) + 1);

    } while (\*pStr);

    //获取磁盘驱动器的设备名

    HANDLE handle =FindFirstVolume(str, sizeof(str));

    do

    {

        cout << str<<endl;

    } while (FindNextVolume(handle,str,sizeof(str)));

    FindVolumeClose(handle);//关闭指定的卷搜索句柄

}

## 效果：

|  |
| --- |
|  |

**GetLogicalDriveStrings：**获取系统中有效驱动器的字符串填充缓冲区。

DWORD GetLogicalDriveStringsW(

    DWORD  nBufferLength,//缓冲区的大小

    LPWSTR lpBuffer//指定接收字符串的缓冲区

);

//返回值:成功返回接收到字符串的长度，失败返回0

**FindFirstVolume：**检索计算机上卷的名称。FindFirstVolume用于开始扫描计算机的卷。

HANDLE FindFirstVolumeW(

        LPWSTR lpszVolumeName,//指向接收字符串的缓冲区的指针

        DWORD  cchBufferLength//缓冲区的大小

    );

//返回值:成功，则返回值是在后续调用中使用的搜索句柄,失败则返回INVALID\_HANDLE\_VALUE

**FindNextVolume：**继续搜索下个

BOOL FindNextVolumeW(

    HANDLE hFindVolume,//上一次调用返回的卷搜索句柄

    LPWSTR lpszVolumeName,//指向接收字符串的缓冲区的指针

    DWORD  cchBufferLength//缓冲区的大小

);

//返回值：成功返回非零，失败返回零

**FindVolumeClose**:关闭

BOOL FindVolumeClose(

    HANDLE hFindVolume//句柄

);

//返回值：成功返回非零，失败返回零

# 4.获取磁盘属性

void Attribute(LPCSTR Directory)

{

    int n=GetDriveType(Directory);

    cout << Directory << "是:";

    switch (n)

    {

    case DRIVE\_UNKNOWN:

        cout << "无法确定驱动器类型" << endl;

    case DRIVE\_FIXED:

        cout << "硬盘驱动器" << endl;

        break;

    }

    CHAR VolumeName[MAX\_PATH];//卷名称

    CHAR FileSystemName[10];//文件系统名称

    DWORD Serial;//序列号

    DWORD  Filenamelength;//文件名称长度

    DWORD Sign;//二进制位标志

    GetVolumeInformation(Directory, VolumeName, MAX\_PATH,

&Serial, &Filenamelength, &Sign, FileSystemName, sizeof(FileSystemName));

    cout << "卷名称:"<<VolumeName << endl;

    cout << "序列号:" << Serial << endl;

    cout << "允许文件名长度" << Filenamelength << endl;

    cout << "文件系统:" << FileSystemName << endl;

    //二进制位判断需要用位与运算(&)

    if (Sign & FILE\_VOLUME\_QUOTAS)

        cout << "这个磁盘允许配额"<<endl;

    if (Sign & FILE\_SUPPORTS\_ENCRYPTION)

        cout << "这个磁盘支持加密"<<endl;

}

int main()

{

    Attribute("c:\\");

    cout << "\n" << endl;

    Attribute("d:\\");

}

## 效果

|  |
| --- |
|  |

**GetDriveType**:确定磁盘驱动器是可移动、固定、CD-ROM、RAM磁盘还是网络驱动器

UINT GetDriveTypeA(

    LPCSTR lpRootPathName//驱动器的根目录,后面的反斜杠是必需的。如果此参数为无效的，函数使用当前目录的根目录

);

//返回值:返回一个整数,用来判断是什么驱动器

DRIVE\_UNKNOWN:无法确定驱动器类型  
DRIVE\_NO\_ROOT\_DIR: 根路径无效；例如，指定路径上没有装入卷。  
DRIVE\_REMOVABLE: 驱动器具有可移动介质；例如，软盘驱动器、拇指驱动器或闪存卡读取器。  
DRIVE\_FIXED: 驱动器具有固定介质；例如，硬盘驱动器或闪存驱动器。  
DRIVE\_REMOTE: 该驱动器是远程（网络）驱动器。  
DRIVE\_CDROM:该驱动器是一个CD - ROM驱动器。  
DRIVE\_RAMDISK: 驱动器是RAM磁盘

**GetVolumeInformation**：获取指定磁盘的属性

BOOL GetVolumeInformationA(

    LPCSTR  lpRootPathName,//驱动器的根目录

    LPSTR   lpVolumeNameBuffer,指向接收指定卷名的缓冲区的指针

    DWORD   nVolumeNameSize,//卷名缓冲区的长度，最大缓冲区大小为MAX\_PATH（260）

    LPDWORD lpVolumeSerialNumber,//指向接收卷序列号的变量的指针。

    LPDWORD lpMaximumComponentLength,//指向接收文件名最大长度的变量的指针

    LPDWORD lpFileSystemFlags,//指向接收与指定文件系统关联的二进制位的标志的变量的指针

    LPSTR   lpFileSystemNameBuffer,//指向接收文件系统名称的缓冲区的指针，例如FAT文件系统或NTFS文件系统

    DWORD   nFileSystemNameSize//文件系统名称缓冲区的长度

);

lpFileSystemFlags:  
FS\_CASE\_IS\_PRESERVED:文件名的大小写记录于文件系统  
FS\_CASE\_SENSITIVE:文件名要区分大小写  
FS\_UNICODE\_STORED\_ON\_DISK:文件名保存为Unicode格式  
FS\_PERSISTANT\_ACLS:文件系统支持文件的访问控制列表（ACL）安全机制  
FS\_FILE\_COMPRESSION:文件系统支持逐文件的进行文件压缩  
FS\_VOL\_IS\_COMPRESSED:整个磁盘卷都是压缩的  
FILE\_VOLUME\_QUOTAS：配额  
FILE\_SUPPORTS\_ENCRYPTION：支持加密

# 5.获取磁盘容量

**int main()  
{**

    DWORD Asector;//扇区

    DWORD Byte;//字节

    DWORD Cluster;//总簇

    DWORD Freecluster;//空闲簇

    GetDiskFreeSpace("c:", &Asector, &Byte, &Freecluster, &Cluster);

    cout << "扇区:" << Asector << endl;;

    cout << "字节" << Byte << endl;

    cout << "总簇" << Cluster << endl;

    cout << "空闲簇" << Freecluster << endl;

    cout << "总空间" << (DWORD64)Cluster \* (DWORD64)Asector \* (DWORD64)Byte << endl;

    cout << "可用空间" << (DWORD64)Freecluster \* (DWORD64)Asector \* (DWORD64)Byte << endl;

    cout << "已用空间" << ((DWORD64)Cluster \* (DWORD64)Asector \* (DWORD64)Byte)-

                          ((DWORD64)Freecluster \* (DWORD64)Asector \* (DWORD64)Byte) << endl;

    DWORD64  Total;//总空间

    DWORD64  Usable;//可用空间

    DWORD64  Free;//空闲用空间

    GetDiskFreeSpaceEx("c:", (PULARGE\_INTEGER)&Free, (PULARGE\_INTEGER)&Total, (PULARGE\_INTEGER)&Usable);

    cout << "总空间" << Total << endl;

    cout << "可用空间" << Usable << endl;

    cout << "空闲空间" << Free << endl;

}

## 效果：

|  |
| --- |
|  |

**GetDiskFreeSpace:获取磁盘容量需要自己计算**  
总空间：总簇\*每簇扇区数量\*每扇区字节数  
空闲空间：空闲簇\*每簇扇区数量\*每扇区字节数  
已用空间**:**总空间-空闲空间

BOOL GetDiskFreeSpaceA(

    LPCSTR  lpRootPathName,//磁盘根目录，不加\

    LPDWORD lpSectorsPerCluster,//指向变量的指针,接收每个簇的扇区数

    LPDWORD lpBytesPerSector,//指向变量的指针,接收每个扇区的字节数

    LPDWORD lpNumberOfFreeClusters,//指向变量的指针,接收总簇数

    LPDWORD lpTotalNumberOfClusters//指向变量的指针,接收空闲簇数

);

**GetDiskFreeSpaceEx**：获取磁盘容量

BOOL GetDiskFreeSpaceExA(

    LPCSTR          lpDirectoryName,//磁盘根目录，不加\

    PULARGE\_INTEGER lpFreeBytesAvailableToCaller,//接收获取到的空闲的空间

    PULARGE\_INTEGER lpTotalNumberOfBytes,//接收获取到的总空间

    PULARGE\_INTEGER lpTotalNumberOfFreeBytes//接收获取到的可用的空间

);