Windows API 一日一练(74)GetSystemInfo 函数

有一次，我正在开发一个视频压缩程序，而压缩算法是需要非常高效的，也就是需要使用到 CPU 的多媒体指令。在 X86 的领域里，目前主要有两家 CPU，就是 INTEL 和 AMD。它们的多媒体指令是不一样的。为了区分这种不同的指令，就需要调用函数 GetSystemInfo 来获取 CPU 的信息，然后再调用不同的动态连接库来进行多媒体数据压缩。

函数 GetSystemInfo 声明如下：

```
WINBASEAPI
VOID
WINAPI
GetSystemInfo(
    __out LPSYSTEM_INFO lpSystemInfo
    );
```

lpSystemInfo 是返回硬件信息的结构。

调用函数的例子如下：

```
#001 //
#002   //获取当前系统的硬件信息。
#003   //蔡军生  2007/11/15 QQ:9073204  深圳
#004   void GetHardInfo(void)
#005   {
#006           //
#007           SYSTEM_INFO sysInfo;
#008
#009           //获取系统的信息。
#010           ::GetSystemInfo(&sysInfo);
#011
#012           //显示当前系统的信息。
#013           //
#014           const int nBufSize = 512;
#015           TCHAR chBuf[nBufSize];
#016           ZeroMemory(chBuf,nBufSize);
#017
#018           wsprintf(chBuf,_T("OEM ID: %u\n"),sysInfo.dwOemId);
#019           OutputDebugString(chBuf);
#020
#021           wsprintf(chBuf,_T("CPU 个数: %u\n"),sysInfo.dwNumberOfProcessors);
#022           OutputDebugString(chBuf);
#023
#024           wsprintf(chBuf,_T("内存分页大小: %u\n"),sysInfo.dwPageSize);
#025           OutputDebugString(chBuf);
#026
```

```
#027            wsprintf(chBuf,_T("CPU 类型: %u\n"),sysInfo.dwProcessorType);
#028            OutputDebugString(chBuf);
#029
#030            wsprintf(chBuf,_T("CPU 架构: %u\n"),sysInfo.wProcessorArchitecture);
#031            OutputDebugString(chBuf);
#032
#033            wsprintf(chBuf,_T("CPU 的级别: %u\n"),sysInfo.wProcessorLevel);
#034            OutputDebugString(chBuf);
#035
#036            wsprintf(chBuf,_T("CPU 的版本: %u\n"),sysInfo.wProcessorRevision);
#037            OutputDebugString(chBuf);
#038
#039    }
#040
```

Windows API 一日一练(75)SystemParametersInfo 函数

面对华丽的 Windows 桌面，工作的心情或许好很多，但是久了总会失去兴趣，总想定期地更新桌面的图片。软件开发人员又面对这样的需求了，需要怎么样去做呢？努力去找 API 函数吧。到目前为止，还有很多变桌面图片的软件，并且还能很挣钱的。其实设置桌面图片的需求，在目前数码相片处理软件也有现实的需求，比如当你去旅游回来后，想把照片当作桌面图片，就可以在处理图片时就设置为桌面图片。这样就需要使用函数 SystemParametersInfo 来完成这项工作了，当然这个函数还有很多其它功能，比如获取桌面工作区的大小。

函数 SystemParametersInfo 声明如下：

```
WINUSERAPI
BOOL
WINAPI
SystemParametersInfoA(
    __in UINT uiAction,
    __in UINT uiParam,
    __inout_opt PVOID pvParam,
    __in UINT fWinIni);
WINUSERAPI
BOOL
WINAPI
SystemParametersInfoW(
    __in UINT uiAction,
    __in UINT uiParam,
    __inout_opt PVOID pvParam,
    __in UINT fWinIni);
#ifdef UNICODE
#define SystemParametersInfo SystemParametersInfoW
#else
```

```
#define SystemParametersInfo SystemParametersInfoA
#endif // !UNICODE
```

uiAction 是作不同的操作参数。
uiParam 是设置的参数。
pvParam 是设置或返回的参数。
fWinIni 是设置的参数。

调用函数的例子如下：

```
#001 //
#002   //获取系统配置信息。
#003   //蔡军生 2007/11/16 QQ:9073204 深圳
#004   void GetSystemParam(void)
#005   {
#006           //获取桌面墙纸的路径。
#007           //SPI_GETDESKWALLPAPER
#008           TCHAR chPath[MAX_PATH];
#009          if (SystemParametersInfo(SPI_GETDESKWALLPAPER,MAX_PATH,chPath,0))
#010           {
#011                  //
#012                  OutputDebugString(chPath);
#013                  OutputDebugString(_T("\r\n"));
#014           }
#015
#016           //获取工作区的大小。
#017           //SPI_GETWORKAREA
#018           RECT rcWorkArea;
#019          if (SystemParametersInfo(SPI_GETWORKAREA,0,&rcWorkArea,0))
#020           {
#021                  //
#022                  const int nBufSize = 256;
#023                  TCHAR chBuf[nBufSize];
#024
#025                  wsprintf(chBuf,_T("%u,%u,%u,%u"),rcWorkArea.left,rcWorkArea.top,
#026                         rcWorkArea.right,rcWorkArea.bottom);
#027
#028                  OutputDebugString(chBuf);
#029                  OutputDebugString(_T("\r\n"));
#030           }
#031
#032   }
```

Windows API 一日一练(76)GlobalAlloc 函数
在 Windows 系统里，有一项功能非常实用，就是剪贴板功能，它能够从一个程序里与另一

个程序进行数据交换的功能，也就是说两个进程上是可以共享数据。要实现这样的功能，Windows 系统在底层上有相应的支持，就是高端地址的内存是系统内存，这样就可以不同的进程进行共享数据了。因此，调用函数 GlobalAlloc 来分配系统内存，让不同的进程实现共享数据，也就是剪贴板功能，可以在一个进程内分配内存，在另一个进程里访问数据后删除内存。

函数 GlobalAlloc 声明如下：

```
HGLOBAL
WINAPI
GlobalAlloc (
    __in UINT uFlags,
    __in SIZE_T dwBytes
    );
```

uFlags 是内存标志。
dwBytes 是分配内存的大小。

调用函数的例子如下：

```
#001 //
#002   //全局内存的分配。
#003   //蔡军生  2007/11/19 QQ:9073204  深圳
#004   void MemGlobal(void)
#005   {
#006            //分配全局内存。
#007           BYTE* pGlobal = (BYTE*)::GlobalAlloc(GMEM_FIXED,1024);
#008
#009            if (!pGlobal)
#010            {
#011                    return;
#012            }
#013            else
#014            {
#015                    //测试全局内存。
#016                    ZeroMemory(pGlobal,1024);
#017                    memcpy(pGlobal,_T("分配内存成功\r\n"),
#018                            sizeof(_T("分配内存成功\r\n")));
#019                    OutputDebugString((LPWSTR)pGlobal);
#020            }
#021
#022            //释放全局内存。
#023            ::GlobalFree((HGLOBAL)pGlobal);
#024   }
```

Windows API 一日一练(77)VirtualAlloc 函数

上一次学习了全局内存的分配，在 Windows 里内存管理是分为两部份，全局内存是系统管理的内存，因而所有进程都可以访问的内存，而每一个进程又有自己的内存空间，这就是虚拟内存空间了，而虚拟内存的空间比较大，当物理内存不足时，系统会把虚拟内存的数据保存到硬盘里，这样只要硬盘的空间足够大，每个进程就可以使用 3G 的内存。虚拟内存分配可以作为程序里分配内存的主要方式，比如大量的数据缓冲区，动态分配内存的空间。使用 VirtualAlloc 函数来分配内存的速度要比全局内存要快。

函数 VirtualAlloc 声明如下：

```
WINBASEAPI
__bcount(dwSize)
LPVOID
WINAPI
VirtualAlloc(
    __in_opt LPVOID lpAddress,
    __in     SIZE_T dwSize,
    __in     DWORD flAllocationType,
    __in     DWORD flProtect
    );
```

lpAddress 是指定内存开始的地址。
dwSize 是分配内存的大小。
flAllocationType 是分配内存的类型。
flProtect 是访问这块分配内存的权限。

调用函数的例子如下：

```
#001 //
#002   //分配虚拟内存的分配。
#003   //蔡军生  2007/11/20 QQ:9073204  深圳
#004   void MemVirtual(void)
#005   {
#006           //
#007           //分配新内存大小。
#008           UINT nNewSize = (UINT) ceil(1500 / 1024.0) * 1024;
#009                                                   PBYTE     pNewBuffer     =     (PBYTE)
VirtualAlloc(NULL,nNewSize,MEM_COMMIT,PAGE_READWRITE);
#010           if (pNewBuffer)
#011           {
#012                   //测试虚拟内存。
#013                   ZeroMemory(pNewBuffer,1500);
#014                   memcpy(pNewBuffer,_T("分配虚拟内存成功\r\n"),
#015                           sizeof(_T("分配虚拟内存成功\r\n")));
#016                   OutputDebugString((LPWSTR)pNewBuffer);
#017
```

```
#018                    //删除分配的内存。
#019                    VirtualFree(pNewBuffer,0,MEM_RELEASE);
#020            }
#021
#022    }
```

Windows API 一日一练(78)HeapAlloc 函数

前面已经介绍两个分配内存的函数，一个全局的内存分配，一个是私有的内存分配。在进程私有的内存里分配里，又有两种分配情况，一种上基于栈式的内存分配，另一种是基于堆内存的分配。在 c++里使用堆内存分配是使用 HeapAlloc 函数来实现的，也就是实现 new 操作符分配内存时会调这个函数。

函数 HeapAlloc 声明如下：

```
WINBASEAPI
__bcount(dwBytes)
LPVOID
WINAPI
HeapAlloc(
    __in HANDLE hHeap,
    __in DWORD dwFlags,
    __in SIZE_T dwBytes
    );
```

hHeap 是进程堆内存开始位置。

dwFlags 是分配堆内存的标志。

dwBytes 是分配堆内存的大小。

调用函数的例子如下：

```
#001    //
#002    //分配堆内存。
#003    //蔡军生  2007/11/26 QQ:9073204  深圳
#004    void MemHeap(void)
#005    {
#006            //
#007            const int nHeapSize = 1024;
#008            PBYTE pNewHeap = (PBYTE) ::HeapAlloc(GetProcessHeap(), 0, nHeapSize);
#009
#010            if (pNewHeap)
#011            {
#012                    //测试分配堆内存。
#013                    ZeroMemory(pNewHeap,nHeapSize);
#014                    memcpy(pNewHeap,_T("分配堆内存成功\r\n"),
#015                            sizeof(_T("分配堆内存成功\r\n")));
#016                    OutputDebugString((LPWSTR)pNewHeap);
#017
```

```
#018                    //释放内存
#019                    BOOL bRes = ::HeapFree(GetProcessHeap(), 0, pNewHeap);
#020                    if (bRes != TRUE)
#021                    {
#022                            OutputDebugString(_T("释放内存出错\r\n"));
#023                    }
#024            }
#025
#026    }
```

Windows API 一日一练(79)GlobalMemoryStatusEx 函数

在开发软件的过程中，经常会碰到不同用户的 PC 系统配置不一样。比如有些用户的系统内存配置比较差，这样处理大量数据时，就不能把大量的数据读取到内存里处理了。而又有一些用户的内存比较多，或者是机器比较新，那么就可以加载大量的数据到内存里处理，这样可以随着系统的更新，软件的处理能力能大幅地提高性能。这样就需要了解系统的配置信息了，最重要的资源之一内存，就是最需要了解的，需要调用函数 GlobalMemoryStatusEx 来了解内存的分配情况。

函数 GlobalMemoryStatusEx 声明如下：

```
WINBASEAPI
BOOL
WINAPI
GlobalMemoryStatusEx(
    __out LPMEMORYSTATUSEX lpBuffer
    );
```

lpBuffer 是接收内存信息的结构。

调用函数的例子如下：

```
#001    //
#002    //当前系统内存信息。
#003    //蔡军生  2007/11/27 QQ:9073204  深圳
#004    void MemInfo(void)
#005    {
#006            //获取内存信息。
#007            MEMORYSTATUSEX memStatex;
#008              memStatex.dwLength = sizeof(memStatex);
#009
#010           if (GlobalMemoryStatusEx(&memStatex))
#011            {
#012                    //
#013                    const int nBufSize = 512;
```

```
#014                    TCHAR chBuf[nBufSize];
#015                    ZeroMemory(chBuf,nBufSize);
#016
#017                    //内存使用率。
#018                    wsprintf(chBuf,_T("内存使用率: %u%%\n"),memStatex.dwMemoryLoad);
#019                    OutputDebugString(chBuf);
#020
#021                    //总共物理内存。
#022                    wsprintf(chBuf,_T("总共物理内存: %u\n"),memStatex.ullTotalPhys );
#023                    OutputDebugString(chBuf);
#024
#025                    //可用物理内存。
#026                    wsprintf(chBuf,_T("可用物理内存: %u\n"),memStatex.ullAvailPhys );
#027                    OutputDebugString(chBuf);
#028
#029                    //全部内存。
#030                    wsprintf(chBuf,_T("全部内存: %u\n"),memStatex.ullTotalPageFile );
#031                    OutputDebugString(chBuf);
#032
#033                    //全部可用的内存。
#034                                                wsprintf(chBuf,_T("全部可用的内
存: %u\n"),memStatex.ullAvailPageFile);
#035                    OutputDebugString(chBuf);
#036
#037                    //全部的虚拟内存。
#038                    wsprintf(chBuf,_T("全部的内存: %u\n"),memStatex.ullTotalVirtual);
#039                    OutputDebugString(chBuf);
#040
#041            }
#042    }
```

Windows API 一日一练(81)FormatMessage 函数

在开发软件的过程里，经常要做的工作就是调试程序，许多问题的出现，不但是逻辑的问题，还有可能是对 API 的不熟悉，或者某种条件下调用 API 会出错的。那么这些出错的原因是什么呢？通常只获取到错误码，也就是通过函数 GetLastError 得到。当然可以根据这个错误码去查找 MSDN 就可以知道出错的原因，但有时在客户那里并没有 MSDN，那么就需要把调用 API 函数出错的信息显示出来，或者写到 LOG 里去。这时就需要调用函数 FormatMessage 把出错码详细原因显示出来。

函数 FormatMessage 声明如下：

```
WINBASEAPI
DWORD
```

```
WINAPI
FormatMessageA(
    DWORD dwFlags,
    LPCVOID lpSource,
    DWORD dwMessageId,
    DWORD dwLanguageId,
    LPSTR lpBuffer,
    DWORD nSize,
    va_list *Arguments
    );
WINBASEAPI
DWORD
WINAPI
FormatMessageW(
    DWORD dwFlags,
    LPCVOID lpSource,
    DWORD dwMessageId,
    DWORD dwLanguageId,
    LPWSTR lpBuffer,
    DWORD nSize,
    va_list *Arguments
    );
#ifdef UNICODE
#define FormatMessage FormatMessageW
#else
#define FormatMessage FormatMessageA
#endif // !UNICODE
```

调用函数的例子如下：

```
#001 //系统错误信息提示。
#002   //蔡军生  2007/11/28 QQ:9073204  深圳
#003   void TestErrorInfo(void)
#004   {
#005           //进行出错。
#006           if (!CreateDirectory(_T("c:\\"),0))
#007           {
#008                 TCHAR szBuf[128];
#009                 LPVOID lpMsgBuf;
#010                 DWORD dw = GetLastError();
#011
#012                FormatMessage(
#013                       FORMAT_MESSAGE_ALLOCATE_BUFFER |
#014                       FORMAT_MESSAGE_FROM_SYSTEM,
#015                       NULL,
```

```
#016                        dw,
#017                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
#018                        (LPTSTR) &lpMsgBuf,
#019                        0, NULL );
#020
#021                    wsprintf(szBuf,
#022                        _T("%s 出错信息 (出错码＝%d): %s"),
#023                        _T("CreateDirectory"), dw, lpMsgBuf);
#024
#025                    LocalFree(lpMsgBuf);
#026
#027                    //输出提示。
#028                    OutputDebugString(szBuf);
#029            }
#030
#031    }
```

调用后输出下面的提示信息：
CreateDirectory 出错信息 (出错码＝5): 拒绝访问。

Windows API 一日一练(82)LoadLibrary 函数
随着软件规模的扩大，要求的功能也是越来越多，开发人员的参与也是越来越多。因此软件的功能划分，就成为了现代软件工程的重大任务，还有软件开发的并行性也越来越重要。为了解决这些问题，大家都会看到 IT 硬件发展非常迅速，功能也越来越复杂，但硬件中发展明显提高在于采用 IC 的方式来实现复杂的功能，也就是把大部份功能集成到一起，只要给出一些引脚就可以实现产品了。而软件中有没有相同于硬件中的 IC 呢？我想是应有的，就是动态连接库了。在 Windows 这座大厦里，很多基石就是动态连接库构成的。一个动态连接库就封装了特别复杂的功能，使用者不必关心它是怎么样实现的。当然，这样也可以让不同的开发者同时开发产品，提高软件开发的速度。要使用动态连接库里的函数，就需要使用 LoadLibrary 函数来加载动态连接库，使用函数 GetProcAddress 来获取功能函数的地址。

函数 LoadLibrary 声明如下：

```
WINBASEAPI
__out
HMODULE
WINAPI
LoadLibraryA(
    __in LPCSTR lpLibFileName
    );
WINBASEAPI
__out
HMODULE
WINAPI
```

```
LoadLibraryW(
    __in LPCWSTR lpLibFileName
    );
#ifdef UNICODE
#define LoadLibrary LoadLibraryW
#else
#define LoadLibrary LoadLibraryA
#endif // !UNICODE
```

lpLibFileName 是动态连接库的名称。

调用函数的例子如下：

```
#001 //加载动态连接库。
#002   //蔡军生  2007/12/03 QQ:9073204  深圳
#003   void TestLoadDLL(void)
#004   {
#005            //加载动态连接库。
#006           HMODULE hDllLib = LoadLibrary(_T("Kernel32.dll"));
#007            if (hDllLib)
#008            {
#009                    //获取动态连接库里的函数地址。
#010                    FARPROC fpFun = GetProcAddress(hDllLib,"GetVersion");
#011
#012                    //调用函数运行。
#013                    DWORD dwVersion = (*fpFun)();
#014
#015                    //获取 WINDOWS 的版本。
#016                                                    DWORD    dwWindowsMajorVersion    =
(DWORD)(LOBYTE(LOWORD(dwVersion)));
#017                                                    DWORD    dwWindowsMinorVersion    =
(DWORD)(HIBYTE(LOWORD(dwVersion)));
#018
#019                    //显示。
#020                    const int nBufSize = 512;
#021                    TCHAR chBuf[nBufSize];
#022                    ZeroMemory(chBuf,nBufSize);
#023
#024                    wsprintf(chBuf,_T("显示版本：%d,%d\r\n"),
#025                            dwWindowsMajorVersion,dwWindowsMinorVersion);
#026                    OutputDebugString(chBuf);
#027
#028                    //释放动态连接库。
#029                    FreeLibrary(hDllLib);
#030            }
```

```
#031
#032    }
```

Windows API 一日一练(83)GetModuleFileName 函数

在开发软件的过程里，经常需要把数据保存到当前执行文件路径下面，或者读取当前执行文件路径下的一些配置信息。这时就需要从当前模块里获取所在的目录路径，以便进行固定的位置操作文件。要解决这个需求，就需要调用 API 函数 GetModuleFileName 来获取模块所在的路径。

函数 GetModuleFileName 声明如下：

```
WINBASEAPI
DWORD
WINAPI
GetModuleFileNameA(
    __in_opt HMODULE hModule,
    __out_ecount_part(nSize, return + 1) LPCH lpFilename,
    __in        DWORD nSize
    );
WINBASEAPI
DWORD
WINAPI
GetModuleFileNameW(
    __in_opt HMODULE hModule,
    __out_ecount_part(nSize, return + 1) LPWCH lpFilename,
    __in        DWORD nSize
    );
#ifdef UNICODE
#define GetModuleFileName GetModuleFileNameW
#else
#define GetModuleFileName GetModuleFileNameA
#endif // !UNICODE
```

hModule 是模块的句柄，或者设置为 NULL 表示当前模块。
lpFilename 是保存路径的缓冲区。
nSize 是缓冲区的大小。

调用函数的例子如下：

```
#001 //获取当前程序所在路径。
#002    //蔡军生  2007/12/05 QQ:9073204  深圳
#003    void TestGetExePath(void)
#004    {
#005            //
#006            const int nBufSize = 512;
#007            TCHAR chBuf[nBufSize];
```

```
#008            ZeroMemory(chBuf,nBufSize);
#009
#010            //获取当前执行文件的路径。
#011            if (GetModuleFileName(NULL,chBuf,nBufSize))
#012            {
#013                    //输出带文件名称路径。
#014                    OutputDebugString(chBuf);
#015                    OutputDebugString(_T("\r\n"));
#016
#017                    //获取文件路径。
#018                    TCHAR* lpStrPath = chBuf;
#019                    PathRemoveFileSpec(lpStrPath);
#020                    OutputDebugString(lpStrPath);
#021                    OutputDebugString(_T("\r\n"));
#022            }
#023
#024    }
```

输出的结果如下：

```
g:\work\windows_api\wincpp2\debug\WinCpp.exe
g:\work\windows_api\wincpp2\debug
```

Windows API 一日一练(84)FlushInstructionCache 函数

一般的程序都是在运行前已经编译好的，因此修改指令的机会比较少，但在软件的防确解里，倒是使用很多。当修改指令之后，怎么样才能让 CPU 去执行新的指令呢？这样就需要使用函数 FlushInstructionCache 来把缓存里的数据重写回主内存里去，让 CPU 重新加载新的指令，才能执行新的指令。下面就来学习一下使用这个函数来实现跳到一个静态函数里执行，而不是直接地调用这个函数。

函数 FlushInstructionCache 声明如下：

```
WINBASEAPI
BOOL
WINAPI
FlushInstructionCache(
    __in HANDLE hProcess,
    __in_bcount_opt(dwSize) LPCVOID lpBaseAddress,
    __in SIZE_T dwSize
    );
```

hProcess 是进程句柄。

lpBaseAddress 是要同步内存的开始地址。

dwSize 是要同步内存的大小。

调用函数的例子如下：

```
#001 //声明函数类型。
#002 typedef void (*TESTFUN)(void);
#003
#004 //定义修改代码的结构。
#005 #pragma pack(push,1)
#006 struct ThunkCode
#007 {
#008    BYTE        m_jmp;          // jmp TESTFUN，跳转指令。
#009    DWORD       m_relproc;      // relative jmp，相对跳转的位置。
#010 };
#011 #pragma pack(pop)
#012
#013 //测试动态修改内存里的指令数据。
#014 //蔡军生  2007/12/06 QQ:9073204  深圳
#015 class CFlush
#016 {
#017 public:
#018    //保存动态修改代码的内存。
#019    ThunkCode m_Thunk;
#020
#021    //初始化跳转代码。
#022    void Init(TESTFUN pFun, void* pThis)
#023    {
#024                //设置跳转指针。
#025                  m_Thunk.m_jmp = 0xe9;
#026
#027                //设置跳转的相对地址。
#028                  m_Thunk.m_relproc = (int)pFun - ((int)this+sizeof(m_Thunk));
#029
#030                //把 CPU 里的缓冲数据写到主内存。
#031               FlushInstructionCache(GetCurrentProcess(),
#032                        &m_Thunk, sizeof(m_Thunk));
#033    }
#034
#035    //真实运行的函数。
#036    static void TestFun(void)
#037    {
#038                OutputDebugString(_T("CFlush  动态修改代码运行\r\n"));
#039    }
#040
#041 };
#042
```

如下调用这个类：

```
#001  //测试运行。
#002  CFlush flushTest;
#003
#004  flushTest.Init(flushTest.TestFun,&flushTest);
#005  TESTFUN pTestFun = (TESTFUN)&(flushTest.m_Thunk);
#006  pTestFun();
```

Windows API 一日一练(85)OpenClipboard 函数

多个软件之间进行数据共享是非常重要的，难以想像编辑软件没有 CTRL+C，CTRL+V 的功能，是多么的不方便。很多的操作，就是拷贝的动作，就是为了数据共享。剪贴板共享是 Windows 里比较重要的功能，比如很多采集数据的软件为了方便导数据到 Excel 里面，就可以使用剪贴板的功能。还有即见即所得的界面导到 Word 里面，也可以使用剪贴板的功能。

函数 OpenClipboard 声明如下：

```
WINUSERAPI
BOOL
WINAPI
OpenClipboard(
    __in_opt HWND hWndNewOwner);
```

hWndNewOwner 是前贴板所属于的窗口。

调用函数的例子如下：

```
#001 //拷贝数据到剪贴板。
#002  //蔡军生  2007/12/09 QQ:9073204  深圳
#003  void TestClipBoard(void)
#004  {
#005          //打开剪贴板并清空。
#006          if (OpenClipboard(m_hWnd) &&
#007                  EmptyClipboard())
#008          {
#009                  //
#010                  HGLOBAL hMem;
#011                  std::wstring strText(_T("拷贝数据到剪贴板"));
#012
#013                  //分配全局内存。
#014                  hMem = GlobalAlloc(GMEM_MOVEABLE,
#015                          (strText.length() + 1) * sizeof(TCHAR));
#016                  if (hMem == NULL)
#017                  {
#018                          CloseClipboard();
#019                          return;
#020                  }
#021
```

```
#022                    //拷贝数据到剪贴板内存。
#023                    LPTSTR lpStr = (LPTSTR)GlobalLock(hMem);
#024                    memcpy(lpStr, strText.c_str(),
#025                            strText.length() * sizeof(TCHAR));
#026                    lpStr[strText.length()] = (TCHAR) 0;
#027                    GlobalUnlock(hMem);
#028
#029                    //设置数据到剪贴板
#030                    SetClipboardData(CF_UNICODETEXT, hMem);
#031
#032                    //关闭剪贴板。
#033                    CloseClipboard();
#034            }
#035    }
#036
```

Windows API 一日一练(86)GetClipboardData 函数

前面介绍怎么样把数据放到剪贴板里面，那么又是怎么样从剪贴板里面获取数据出来呢？当然还是需要使用其它的 API 函数来获取剪贴板里的数据。获取剪贴板里的数据时，是不知道当前剪贴板里是否有数据的，也不知道剪贴板里的数据格式是什么。那么下面就来解决这两个问题，先使用函数 IsClipboardFormatAvailable 来获取剪贴板里的格式是否可以处理，接着使用函数 OpenClipboard 打开剪贴板，然后使用函数 GetClipboardData 来获取剪贴板数据。

函数 GetClipboardData 声明如下：

```
WINUSERAPI
HANDLE
WINAPI
GetClipboardData(
    __in UINT uFormat);
```

uFormat 是剪贴板的格式。

调用函数的例子如下：

```
#001 //获取剪贴板的数据。
#002    //蔡军生  2007/12/10 QQ:9073204  深圳
#003    void TestGetClipBoard(void)
#004    {
#005            //判断剪贴板的数据格式是否可以处理。
#006            if (!IsClipboardFormatAvailable(CF_UNICODETEXT))
#007            {
#008                    return;
#009            }
#010
#011            //打开剪贴板。
```

```
#012            if (!OpenClipboard(m_hWnd))
#013            {
#014                    return;
#015            }
#016
#017            //获取 UNICODE 的数据。
#018            HGLOBAL hMem = GetClipboardData(CF_UNICODETEXT);
#019            if (hMem != NULL)
#020            {
#021                    //获取 UNICODE 的字符串。
#022                    LPTSTR lpStr = (LPTSTR)GlobalLock(hMem);
#023                    if (lpStr != NULL)
#024                    {
#025                            //显示输出。
#026                            OutputDebugString(lpStr);
#027
#028                            //释放锁内存。
#029                            GlobalUnlock(hMem);
#030                    }
#031            }
#032
#033            //关闭剪贴板。
#034            CloseClipboard();
#035    }
```

Windows API 一日一练(87)CreateProcess 函数

人们需要处理的信息越来越复杂，往往在一个应用程序里是处理不完的，因此，就出现多个应用程序协同处理同一件事情。当然多个应用程序分开处理，也是比较容易开发，并且让应用程序复杂难度迅速降低。比如在开发一个银行的交易系统，有一个报表生成的主程序，然后还有很多小的，不同的报表生成程序。这样就需要从主程序里创建小报表程序进行运行。创建进程运行，需要使用函数 CreateProcess 来实现。

函数 CreateProcess 声明如下：

```
WINBASEAPI
BOOL
WINAPI
CreateProcessA(
    __in_opt       LPCSTR lpApplicationName,
    __inout_opt LPSTR lpCommandLine,
    __in_opt       LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt       LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in            BOOL bInheritHandles,
```

```
    __in            DWORD dwCreationFlags,
    __in_opt        LPVOID lpEnvironment,
    __in_opt        LPCSTR lpCurrentDirectory,
    __in            LPSTARTUPINFOA lpStartupInfo,
    __out           LPPROCESS_INFORMATION lpProcessInformation
    );
WINBASEAPI
BOOL
WINAPI
CreateProcessW(
    __in_opt        LPCWSTR lpApplicationName,
    __inout_opt LPWSTR lpCommandLine,
    __in_opt        LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt        LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in            BOOL bInheritHandles,
    __in            DWORD dwCreationFlags,
    __in_opt        LPVOID lpEnvironment,
    __in_opt        LPCWSTR lpCurrentDirectory,
    __in            LPSTARTUPINFOW lpStartupInfo,
    __out           LPPROCESS_INFORMATION lpProcessInformation
    );
#ifdef UNICODE
#define CreateProcess CreateProcessW
#else
#define CreateProcess CreateProcessA
#endif // !UNICODE
```

lpApplicationName 是应用程序的名称。
lpCommandLine 是命令行参数。
lpProcessAttributes 是进程的属性。
lpThreadAttributes 是线程的属性。
bInheritHandles 是否继承父进程的属性。
dwCreationFlags 是创建标志。
lpEnvironment 是环境变量。
lpCurrentDirectory 是当前目录。
lpStartupInfo 是传给新进程的信息。
lpProcessInformation 是进程返回的信息。

调用函数的例子如下：

```
#001 //创建进程。
#002    //蔡军生  2007/12/11 QQ:9073204  深圳
#003    void TestCreateProcess(void)
#004    {
#005                //清空结构。
```

```
#006            STARTUPINFO sInfo;
#007            PROCESS_INFORMATION pInfo;
#008
#009            ZeroMemory( &sInfo, sizeof(sInfo) );
#010            sInfo.cb = sizeof(sInfo);
#011            sInfo.dwFlags = STARTF_USESHOWWINDOW;
#012            sInfo.wShowWindow = SW_SHOWNORMAL;
#013
#014            ZeroMemory( &pInfo, sizeof(pInfo) );
#015
#016            //创建一个进程。
#017           if( !::CreateProcess( _T("WinCpp.exe"),
#018                    NULL,
#019                    NULL,
#020                    NULL,
#021                    FALSE,
#022                    0,
#023                    NULL,
#024                    NULL,
#025                    &sInfo,
#026                    &pInfo )
#027                     )
#028            {
#029                    //输出出错信息。
#030                    const int nBufSize = 512;
#031                    TCHAR chBuf[nBufSize];
#032                    ZeroMemory(chBuf,nBufSize);
#033
#034                    wsprintf(chBuf,_T("CreateProcess failed (%d).\n"), GetLastError() );
#035                    OutputDebugString(chBuf);
#036                    return;
#037            }
#038
#039
#040            //等进程关闭。
#041            WaitForSingleObject( pInfo.hProcess, INFINITE );
#042
#043            //关闭进程和线程的句柄。
#044            CloseHandle( pInfo.hProcess );
#045            CloseHandle( pInfo.hThread );
#046
#047    }
```

Windows API 一日一练(88)EnumProcesses 函数

当你开发的软件在用户那里运行出错了，想怎么办呢？当然是希望把出错时候的运行环境信息生成报表，然后再 Email 回来查看了。这里就介绍一个函数可以把当时运行环境的进程全部找到，然后可以输出每个进程的信息。当然，这个函数也可以使用到杀病毒软件里，用来查看可疑的进程信息。

函数 EnumProcesses 声明如下：

```
BOOL
WINAPI
EnumProcesses (
    DWORD * lpidProcess,
    DWORD    cb,
    DWORD * cbNeeded
    );
```

lpidProcess 是保存进程 ID 的数组。
cb 是进程组数的大小。
cbNeeded 是返回进程数组的大小。

调用函数的例子如下：

```
#001 //获取系统所有进程。
#002   //蔡军生  2007/12/12 QQ:9073204  深圳
#003   void TestEnumProcesses(void)
#004   {
#005           //
#006           const int nBufSize = 512;
#007           TCHAR chBuf[nBufSize];
#008           ZeroMemory(chBuf,nBufSize);
#009
#010           //
#011           DWORD dwProcs[1024*2];
#012           DWORD dwNeeded;
#013
#014           //枚举所有进程 ID。
#015          if ( !EnumProcesses( dwProcs, sizeof(dwProcs), &dwNeeded ) )
#016           {
#017                 //输出出错信息。
#018                 wsprintf(chBuf,_T("EnumProcesses failed (%d).\n"), GetLastError() );
#019                 OutputDebugString(chBuf);
#020
#021                 return;
#022           }
#023
#024           // 计算有多少个进程 ID。
```

```
#025            DWORD dwProcCount = dwNeeded / sizeof(DWORD);
#026
#027            wsprintf(chBuf,_T("EnumProcesses Count(%d).\n"), dwProcCount );
#028            OutputDebugString(chBuf);
#029
#030    }
#031
```

Windows API 一日一练(89)OpenProcess 函数

这一年来流氓软件特别多，面对这种非常恶心的软件，让大家非常痛苦。正是在这种环境之下，众多客户需要强大查杀这种流氓软件的工具。如果让你来开发一个查杀这种病毒的软件，你会怎么做呢？当然是先把电脑里所有进程遍历出来，然后把每个进程的详细信息显示给用户，让用户决定自己那些进程可以运行，那些不可以运行。或者根据当前进程的信息，再跟根据病毒库里的特征码进行比较，就可以标识那些是可疑的病毒了。下面就来演示用函数 OpenProcess 怎么打开进程并获取进程的名称。

函数 OpenProcess 声明如下：

```
WINBASEAPI
__out
HANDLE
WINAPI
OpenProcess(
    __in DWORD dwDesiredAccess,
    __in BOOL bInheritHandle,
    __in DWORD dwProcessId
    );
```

dwDesiredAccess 是访问进程的权限。

bInheritHandle 是句柄是否继承进程属性。

dwProcessId 是进程ＩＤ。

调用函数的例子如下：

```
#001 //获取进程的信息。
#002    //蔡军生  2007/12/13 QQ:9073204  深圳
#003    void TestOpenProcesses(void)
#004    {
#005            //
#006            const int nBufSize = 512;
#007            TCHAR chBuf[nBufSize];
#008            ZeroMemory(chBuf,nBufSize);
#009
#010            //
#011            DWORD dwProcs[1024];
#012            DWORD dwNeeded;
```

```
#013
#014            //枚举所有进程 ID。
#015            if ( !EnumProcesses( dwProcs, sizeof(dwProcs), &dwNeeded ) )
#016            {
#017                    //输出出错信息。
#018                    wsprintf(chBuf,_T("EnumProcesses failed (%d).\n"), GetLastError() );
#019                    OutputDebugString(chBuf);
#020
#021                    return;
#022            }
#023
#024            // 计算有多少个进程 ID。
#025            DWORD dwProcCount = dwNeeded / sizeof(DWORD);
#026
#027            wsprintf(chBuf,_T("EnumProcesses Count(%d).\n"), dwProcCount );
#028            OutputDebugString(chBuf);
#029
#030            //遍历所有进程 ID，打开进程。
#031            for (DWORD i = 0; i < dwProcCount; i++)
#032            {
#033                    wsprintf(chBuf,_T("EnumProcesses (%d).\r\n"), dwProcs[i] );
#034                OutputDebugString(chBuf);
#035
#036                    //根据进程 ID 打开进程。
#037                 HANDLE hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
#038                        PROCESS_VM_READ,
#039                        FALSE, dwProcs[i] );
#040
#041                    if (hProcess)
#042                    {
#043                            HMODULE hMod;
#044                            DWORD cbNeeded;
#045
#046                            //获取进程第一个模块的句柄。
#047                            if ( EnumProcessModules( hProcess, &hMod, sizeof(hMod),
#048                                    &cbNeeded) )
#049                            {
#050                                    //
#051                                    ZeroMemory(chBuf,nBufSize);
#052
#053                                    //获取进程第一个模块的名称。
#054                                if (::GetModuleBaseName(hProcess,hMod,chBuf,nBufSize))
#055                                    {
#056                                            //
```

```
#057                                    OutputDebugString(chBuf);
#058                                    OutputDebugString(_T("\r\n"));
#059                                }
#060                            }
#061                        }
#062                }
#063
#064    }
```

Windows API 一日一练(90)GetGlyphOutline 函数

中西文化的差异，导致在电子信息里处理也大不相同，在英文里只需要 26 个字母就可以显示所有文章了，而在中文里需要最基本的字符就有 2000 多个。对于一些在嵌入式软件里要显示的字符，那么就得手动去构造所有图形，这是一个比较大的工作量，如果让每个厂家都去完成这个任务，显然是不可能的。面对着大量嵌入式用户的需求，那么就需要解决中文字模的图形问题。毕竟大家经常使用 Windows，最先想到的，肯定是怎么样把里面的字符提取图形出来，生成自己需要的几个字库。下面就来介绍怎么样用函数 GetGlyphOutline 获取显示字符的图形数据。

函数 GetGlyphOutline 声明如下：

```
WINGDIAPI DWORD WINAPI GetGlyphOutlineA(      __in HDC hdc,
                                              __in UINT uChar,
                                              __in UINT fuFormat,
                                              __out LPGLYPHMETRICS lpgm,
                                              __in DWORD cjBuffer,
                                              __out_bcount_opt(cjBuffer)          LPVOID
pvBuffer,
                                              __in CONST MAT2 *lpmat2
                                          );
WINGDIAPI DWORD WINAPI GetGlyphOutlineW(      __in HDC hdc,
                                              __in UINT uChar,
                                              __in UINT fuFormat,
                                              __out LPGLYPHMETRICS lpgm,
                                              __in DWORD cjBuffer,
                                              __out_bcount_opt(cjBuffer)          LPVOID
pvBuffer,
                                              __in CONST MAT2 *lpmat2
                                          );
#ifdef UNICODE
#define GetGlyphOutline GetGlyphOutlineW
#else
#define GetGlyphOutline GetGlyphOutlineA
#endif // !UNICODE
```

hdc 是设备句柄。
uChar 是需要获取图形数据的字符。
fuFormat 是获取数据的格式。
lpgm 是获取字符的相关信息。
cjBuffer 是保存字符数据的缓冲区大小。
pvBuffer 是保存字符数据的缓冲区。
lpmat2 是 3*3 的变换矩阵。

调用函数的例子如下：

```
#001 //浮点数据转换为固定浮点数。
#002    FIXED FixedFromDouble(double d)
#003    {
#004            long l;
#005             l = (long) (d * 65536L);
#006            return *(FIXED *)&l;
#007    }
#008
#009    //设置字体图形变换矩阵。
#010    void SetMat(LPMAT2 lpMat)
#011    {
#012            lpMat->eM11 = FixedFromDouble(2);
#013            lpMat->eM12 = FixedFromDouble(0);
#014            lpMat->eM21 = FixedFromDouble(0);
#015            lpMat->eM22 = FixedFromDouble(2);
#016    }
#017
#018    //
#019    //获取字模信息。
#020    //蔡军生  2007/12/16 QQ:9073204  深圳
#021    void TestFontGlyph(void)
#022    {
#023            //创建字体。
#024            HFONT hFont = GetFont();
#025
#026            //设置字体到当前设备。
#027            HDC hDC = ::GetDC(m_hWnd);
#028            HFONT hOldFont = (HFONT)SelectObject(hDC,hFont);
#029
#030            //设置字体图形变换矩阵
#031            MAT2 mat2;
#032            SetMat(&mat2);
#033
#034
```

```
#035            GLYPHMETRICS gm;
#036
#037            //设置要显示的字符。
#038            TCHAR chText = L'蔡';
#039
#040            //获取这个字符图形需要的字节的大小。
#041                                                    DWORD       dwNeedSize      =
GetGlyphOutline(hDC,chText,GGO_BITMAP,&gm,0,NULL,&mat2);
#042            if (dwNeedSize > 0 && dwNeedSize < 0xFFFF)
#043            {
#044                    //按需要分配内存。
#045                                                                LPBYTE      lpBuf       =
(LPBYTE)HeapAlloc(GetProcessHeap(),HEAP_ZERO_MEMORY,dwNeedSize);
#046                    if (lpBuf)
#047                    {
#048                            //获取字符图形的数据到缓冲区。
#049
GetGlyphOutline(hDC,chText,GGO_BITMAP,&gm,dwNeedSize,lpBuf,&mat2);
#050
#051                            //计算图形每行占用的字节数。
#052                            int nByteCount = ((gm.gmBlackBoxX +31) >> 5) << 2;
#053
#054                            //显示每行图形的数据。
#055                            for (int i = 0; i < gm.gmBlackBoxY; i++)
#056                            {
#057                                    //
#058                                    for (int j = 0; j < nByteCount; j++)
#059                                    {
#060
#061                                            BYTE btCode = lpBuf[i* nByteCount + j];
#062
#063                                            //按字节输出每点的数据。
#064                                            for (int k = 0; k < 8; k++)
#065                                            {
#066
#067                                                    if (btCode & (0x80>>k))
#068                                                    {
#069
#070                                                            OutputDebugString(_T("1"));
#071                                                    }
#072                                                    else
#073                                                    {
#074                                                            OutputDebugString(_T("0"));
#075                                                    }
```

```
#076
#077                                    }
#078
#079                                }
#080
#081                                //
#082                                OutputDebugString(_T("\r\n"));
#083                            }
#084
#085                            //
#086                            HeapFree(GetProcessHeap(),0,lpBuf);
#087                        }
#088                }
#089
#090                //
#091                SelectObject(hDC,hOldFont);
#092                DeleteObject(hFont);
#093
#094                //
#095                ReleaseDC(m_hWnd,hDC);
#096    }
#097
```

输出的结果如下：

```
00000000000000010000000000000000
00000000110000011000000000000000
00000000100000011000000000000000
00000000100000011000011000000000
11111111111111111111111100000000
01000000100000011000000000000000
00000000100000011000000000000000
00000100100000011000000000000000
00000110100000010000000000000000
00000100000000000000000000000000
00001100000001000000100000000000
00001111111101111111110000000000
00001000001111000000110000000000
00011000001000100001100000000000
00010100001000100001000000000000
00010010011000100011000000000000
00100011010000010010000000000000
00110010110000011010000000000000
01011000110000001100000000000000
10001000100000001100000000000000
```

00001001100000010110000000000000
00001011011111111011000000000000
00000010000000000001110000000000
00000110000000000000111100000000
00001100000000000110011100000000
00011111111111111111001000000000
00110000000010000000000000000000
01000000000010000000000000000000
00000001000010000000000000000000
00000011100010001100000000000000
00000011000010000010000000000000
00000110000010000011100000000000
00001100000010000001100000000000
00011000100010000000110000000000
00110000011110000000110000000000
01000000001110000000010000000000
00000000000010000000000000000000

Windows API 一日一练(91)GetProcessMemoryInfo 函数
当大家打开 Windows 任务管理器时，就会看到每个进程使用内存的分布情况，往往会发现有一些进程占用大量的内存，在这种情况也是一种异常情况，可以作为是否恶意软件的标志之一。下面就来使用 API 函数 GetProcessMemoryInfo 来获取内存的使用情况。

函数 GetProcessMemoryInfo 声明如下：

```
BOOL
WINAPI
GetProcessMemoryInfo(
    HANDLE Process,
    PPROCESS_MEMORY_COUNTERS ppsmemCounters,
    DWORD cb
    );
```

Process 是获取内存使用情况的进程句柄。
ppsmemCounters 是返回内存使用情况的结构。
cb 是结构的大小。

调用函数的例子如下：

```
#001 //获取某一个进程的内存信息。
#002    //蔡军生  2007/12/18 QQ:9073204  深圳
#003    void TestGetProcessMemoryInfo(void)
#004    {
#005            //
#006            const int nBufSize = 512;
#007            TCHAR chBuf[nBufSize];
```

```
#008        ZeroMemory(chBuf,nBufSize);
#009
#010        //
#011        DWORD dwProcs[1024];
#012        DWORD dwNeeded;
#013
#014        //枚举所有进程 ID。
#015        if ( !EnumProcesses( dwProcs, sizeof(dwProcs), &dwNeeded ) )
#016        {
#017            //输出出错信息。
#018            wsprintf(chBuf,_T("EnumProcesses failed (%d).\n"), GetLastError() );
#019            OutputDebugString(chBuf);
#020
#021            return;
#022        }
#023
#024        // 计算有多少个进程 ID。
#025        DWORD dwProcCount = dwNeeded / sizeof(DWORD);
#026
#027        wsprintf(chBuf,_T("EnumProcesses Count(%d).\n"), dwProcCount );
#028        OutputDebugString(chBuf);
#029
#030        //遍历所有进程 ID，打开进程。
#031        for (DWORD i = 0; i < dwProcCount; i++)
#032        {
#033            wsprintf(chBuf,_T("EnumProcesses (%d).\r\n"), dwProcs[i] );
#034            OutputDebugString(chBuf);
#035
#036            //根据进程 ID 打开进程。
#037            HANDLE hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
#038                PROCESS_VM_READ,
#039                FALSE, dwProcs[i] );
#040
#041            if (hProcess)
#042            {
#043                //
#044                PROCESS_MEMORY_COUNTERS pmc;
#045                pmc.cb = sizeof(PROCESS_MEMORY_COUNTERS);
#046
#047                //获取这个进程的内存使用情况。
#048               if ( ::GetProcessMemoryInfo( hProcess, &pmc, sizeof(pmc)) )
#049                {
#050                    ZeroMemory(chBuf,nBufSize);
#051
```

```
#052                                    wsprintf(chBuf,_T("\t 缺页中断次数: 0x%08X\n"),
pmc.PageFaultCount );
#053                                OutputDebugString(chBuf);
#054
#055                                wsprintf(chBuf,_T("\t 使用内存高峰: 0x%08X\n"),
#056                                    pmc.PeakWorkingSetSize );
#057                                OutputDebugString(chBuf);
#058
#059                                    wsprintf(chBuf,_T("\t 当前使用的内存: 0x%08X\n"),
pmc.WorkingSetSize );
#060                                OutputDebugString(chBuf);
#061
#062                                wsprintf(chBuf,_T("\t 使用页面缓存池高峰: 0x%08X\n"),
#063                                    pmc.QuotaPeakPagedPoolUsage );
#064                                OutputDebugString(chBuf);
#065
#066                                wsprintf(chBuf,_T("\t 使用页面缓存池: 0x%08X\n"),
#067                                    pmc.QuotaPagedPoolUsage );
#068                                OutputDebugString(chBuf);
#069
#070                                wsprintf(chBuf,_T("\t 使用非分页缓存池高峰: 0x%08X\n"),
#071                                    pmc.QuotaPeakNonPagedPoolUsage );
#072                                OutputDebugString(chBuf);
#073
#074                                wsprintf(chBuf,_T("\t 使用非分页缓存池: 0x%08X\n"),
#075                                    pmc.QuotaNonPagedPoolUsage );
#076                                OutputDebugString(chBuf);
#077
#078                                    wsprintf(chBuf,_T("\t 使用分页文件: 0x%08X\n"),
pmc.PagefileUsage );
#079                                OutputDebugString(chBuf);
#080
#081                                wsprintf(chBuf,_T("\t 使用分页文件的高峰: 0x%08X\n"),
#082                                    pmc.PeakPagefileUsage );
#083                                OutputDebugString(chBuf);
#084                            }
#085
#086                            //
#087                            CloseHandle(hProcess);
#088                        }
#089                }
#090
#091    }
```

Windows API 一日一练(92)GetOpenFileName 函数

当用户想选择打开以前保存的文件时，就需要使用到选择文件对话框。其实在 Windows 的 API 里已经有这样完美的对话，只需要简单的调用，就可以使用了，而不需要写一大堆其它的代码。下面就来演示一下怎么使用文件选择对话框。

函数 GetOpenFileName 声明如下：

```
WINCOMMDLGAPI BOOL APIENTRY GetOpenFileNameA(LPOPENFILENAMEA);
WINCOMMDLGAPI BOOL APIENTRY GetOpenFileNameW(LPOPENFILENAMEW);
#ifdef UNICODE
#define GetOpenFileName GetOpenFileNameW
#else
#define GetOpenFileName GetOpenFileNameA
#endif // !UNICODE
```

LPOPENFILENAMEA 是指向文件选择对话框的结构。

调用函数的例子如下：

```
#001 //获取用户选择的文件名称。
#002   //蔡军生  2007/12/21 QQ:9073204  深圳
#003   void TestGetOpenFileName(void)
#004   {
#005           //
#006           OPENFILENAME ofn;           // 公共对话框结构。
#007           TCHAR szFile[MAX_PATH]; // 保存获取文件名称的缓冲区。
#008
#009           // 初始化选择文件对话框。
#010           ZeroMemory(&ofn, sizeof(ofn));
#011           ofn.lStructSize = sizeof(ofn);
#012           ofn.hwndOwner = m_hWnd;
#013           ofn.lpstrFile = szFile;
#014           //
#015           //
#016           ofn.lpstrFile[0] = _T('\0');
#017           ofn.nMaxFile = sizeof(szFile);
#018           ofn.lpstrFilter = _T("All\0*.*\0Text\0*.TXT\0");
#019           ofn.nFilterIndex = 1;
#020           ofn.lpstrFileTitle = NULL;
#021           ofn.nMaxFileTitle = 0;
#022           ofn.lpstrInitialDir = NULL;
#023           ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
#024
#025           // 显示打开选择文件对话框。
```

```
#026            if ( GetOpenFileName(&ofn) )
#027            {
#028                    //显示选择的文件。
#029                    OutputDebugString(szFile);
#030                    OutputDebugString(_T("\r\n"));
#031
#032            }
#033    }
```

Windows API 一日一练(93)GetSaveFileName 函数

前面介绍了怎么样打开选择文件读取的对话框，当你设计的软件需要让用户选择保存时，就需要让用户去选择自己合适的文件名称来保存。那么就需要使用到保存文件对话框，也就是 API 函数 GetSaveFileName。下面就来演示怎么样使用这个 API 函数。

函数 GetSaveFileName 声明如下：

```
WINCOMMDLGAPI BOOL APIENTRY GetSaveFileNameA(LPOPENFILENAMEA);
WINCOMMDLGAPI BOOL APIENTRY GetSaveFileNameW(LPOPENFILENAMEW);
#ifdef UNICODE
#define GetSaveFileName GetSaveFileNameW
#else
#define GetSaveFileName GetSaveFileNameA
#endif // !UNICODE
```

调用函数的例子如下：

```
#001 //获取用户选择保存的文件名称。
#002    //蔡军生  2007/12/25 QQ:9073204  深圳
#003    void TestGetSaveFileName(void)
#004    {
#005            //
#006            OPENFILENAME ofn;           // 公共对话框结构。
#007            TCHAR szFile[MAX_PATH]; // 保存获取文件名称的缓冲区。
#008
#009            // 初始化选择文件对话框。
#010            ZeroMemory(&ofn, sizeof(ofn));
#011           ofn.lStructSize = sizeof(ofn);
#012            ofn.hwndOwner = m_hWnd;
#013            ofn.lpstrFile = szFile;
#014
#015            //
#016            ofn.lpstrFile[0] = _T('\0');
#017            ofn.nMaxFile = sizeof(szFile);
#018            ofn.lpstrFilter = _T("All\0*.*\0Text\0*.TXT\0");
#019            ofn.nFilterIndex = 1;
```

```
#020            ofn.lpstrFileTitle = NULL;
#021            ofn.nMaxFileTitle = 0;
#022            ofn.lpstrInitialDir = NULL;
#023            ofn.Flags = OFN_SHOWHELP | OFN_OVERWRITEPROMPT;
#024
#025            // 显示打开选择文件对话框。
#026           if ( GetSaveFileName(&ofn) )
#027            {
#028                    //显示选择的文件。
#029                    OutputDebugString(szFile);
#030                    OutputDebugString(_T("\r\n"));
#031            }
#032    }
```