# 学习大纲

|  |
| --- |
|  |

## 注意：下面的API操作的环境变量不是系统环境变量或者用户环境变量，他们只是进程环境变量，对他们的修改不会响应系统环境变量和用户环境变量。他们只在程序运行的时候存在，一旦程序结束，他们就会被销毁。

## 修改系统环境变量需要使用注册表编程

在所有现代Windows操作系统中，系统的环境变量保存在这两个注册表键中的注册表值：  
系统级：HKEY\_LOCAL\_MACHINE\\System\\CurrentControlSet\\Control\\Session Manager\\Environment  
用户级：HKEY\_CURRENT\_USER\\Environment  
添加、修改注册表值，最方便的方法是使用RegSetKeyValue。删除注册表值可以用RegDeleteKeyValue。如果需要兼容Windows XP等老系统，需要使用RegOpenKeyEx-RegSetValueEx/RegDeleteValue-RegCloseKey的方式。  
  
修改了这两个地方，不会立即生效，还需要通知explorer.exe环境变量已经改变，需要重新读取注册表。方法是向所有窗口（HWND\_BROADCAST）发送WM\_SETTINGCHANGE消息，wParam为0，lParam为(LPARAM)TEXT("Environment")。向窗口发送消息可以用SendMessage函数，但为了避免受到挂起的窗口影响，这里一般是使用SendMessageTimeout或SendNotifyMessage函数代替，但不能使用PostMessage，这是因为这个消息需要直接调用WndProc，不能被放到消息队列中。

## GetEnvironmentStrings函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 检索当前进程的环境变量。 语法 C++复制  LPCH GetEnvironmentStrings(); 返回值 如果函数成功，则返回值是指向当前进程的环境块的指针。  如果函数失败，则返回值为 NULL。 注解 **GetEnvironmentStrings** 函数返回指向内存块的指针，该内存块包含调用进程的环境变量 (系统和用户环境变量) 。  每个环境块包含以下格式的环境变量：  Var1=Value1\0 Var2=Value2\0 Var3=Value3\0 ... VarN=ValueN\0\0  环境变量的名称不能包含等号 (=) 。  将此内存视为只读内存;不要直接修改它。 若要添加或更改环境变量，请使用 [GetEnvironmentVariable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getenvironmentvariable) 和 [SetEnvironmentVariable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setenvironmentvariable) 函数。  当不再需要 **GetEnvironmentStrings** 返回的块时，应通过调用 [FreeEnvironmentStrings](https://learn.microsoft.com/zh-cn/windows/desktop/api/processenv/nf-processenv-freeenvironmentstringsa) 函数来释放该块。  请注意，此函数的 ANSI 版本 **GetEnvironmentStringsA** 返回 OEM 字符。 示例 有关示例，请参阅 [更改环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/changing-environment-variables)。 要求  |  |  | | --- | --- | | **标头** | processenv.h (包括 Windows Server 2003 上的 Windows.h、Windows Vista、Windows 7、Windows Server 2008 Windows Server 2008 R2) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/environment-variables)  [FreeEnvironmentStrings](https://learn.microsoft.com/zh-cn/windows/desktop/api/processenv/nf-processenv-freeenvironmentstringsa)  [GetEnvironmentVariable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getenvironmentvariable)  [SetEnvironmentVariable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setenvironmentvariable) |

## FreeEnvironmentStrings函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 释放环境字符串块。 语法 C++复制  BOOL FreeEnvironmentStringsA(  LPCH penv  ); 参数 penv  指向环境字符串块的指针。 必须通过调用 [GetEnvironmentStrings](https://learn.microsoft.com/zh-cn/windows/desktop/api/rrascfg/nf-rrascfg-ieapproviderconfig-initialize) 函数来获取指向 块的指针。 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 若要获取扩展错误信息，请调用 [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror)。 注解 如果使用 [了 AnSI 版本的 GetEnvironmentStrings](https://learn.microsoft.com/zh-cn/windows/desktop/api/rrascfg/nf-rrascfg-ieapproviderconfig-initialize)，请务必使用 **AnSI 版本的 FreeEnvironmentStrings**。 同样，如果使用 **了 GetEnvironmentStrings** 的 Unicode 版本，请务必使用 **Unicode 版本的 FreeEnvironmentStrings**。 示例 有关示例，请参阅 [更改环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/changing-environment-variables)。  **备注**  processenv.h 标头将 FreeEnvironmentStrings 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  |  |  | | --- | --- | | **标头** | processenv.h (包括 Windows Server 2003、Windows Vista、Windows 7、Windows Server 2008 Windows Server 2008 R2) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/environment-variables)  [GetEnvironmentStrings](https://learn.microsoft.com/zh-cn/windows/desktop/api/rrascfg/nf-rrascfg-ieapproviderconfig-initialize) |

## GetEnvironmentVariable函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 从调用进程的环境块检索指定变量的内容。 语法 C++复制  DWORD GetEnvironmentVariable(  [in, optional] LPCTSTR lpName,  [out, optional] LPTSTR lpBuffer,  [in] DWORD nSize  ); 参数 [in, optional] lpName  环境变量的名称。  [out, optional] lpBuffer  指向缓冲区的指针，该缓冲区以 null 结尾的字符串的形式接收指定环境变量的内容。 环境变量的最大大小限制为 32,767 个字符，包括 null 终止字符。  [in] nSize  *lpBuffer* 参数指向的缓冲区的大小，包括 null 终止符，以字符为单位。 返回值 如果函数成功，则返回值是 *lpBuffer* 指向的缓冲区中存储的字符数，不包括终止 null 字符。  如果 *lpBuffer* 不够大来保存数据，则返回值是保存字符串及其终止 null 字符所需的缓冲区大小（以字符为单位）， *lpBuffer* 的内容未定义。  如果函数失败，则返回值为零。 如果在环境块中找不到指定的环境变量， [GetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/errhandlingapi/nf-errhandlingapi-getlasterror) 将返回ERROR\_ENVVAR\_NOT\_FOUND。 注解 此函数可以检索系统环境变量或用户环境变量。 示例 有关示例，请参阅 [更改环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/changing-environment-variables)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows Server 2003、Windows Vista、Windows 7、Windows Server 2008 Windows Server 2008 R2) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/environment-variables)  [GetEnvironmentStrings](https://learn.microsoft.com/zh-cn/windows/win32/api/processenv/nf-processenv-getenvironmentstrings)  [SetEnvironmentVariable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-setenvironmentvariable) |

### 这3个函数比较有用，下面这个个没有什么用

## SetEnvironmentVariable函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 设置当前进程的指定环境变量的内容。 语法 C++复制  BOOL SetEnvironmentVariable(  [in] LPCTSTR lpName,  [in, optional] LPCTSTR lpValue  ); 参数 [in] lpName  环境变量的名称。 如果环境变量不存在且 *lpValue* 不为 NULL，则操作系统将创建该环境变量。  [in, optional] lpValue  环境变量的内容。 用户定义的环境变量的最大大小为 32,767 个字符。 有关详细信息，请参阅 [环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/environment-variables)。  **Windows Server 2003 和 Windows XP：**进程的环境块的总大小不能超过 32,767 个字符。  如果此参数为 NULL，则会从当前进程的环境中删除变量。 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 此函数对系统环境变量或其他进程的环境变量没有影响。 示例 有关示例，请参阅 [更改环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/changing-environment-variables)。 要求  |  |  | | --- | --- | | **标头** | winbase.h (包括 Windows Server 2003 上的 Windows.h、Windows Vista、Windows 7、Windows Server 2008 Windows Server 2008 R2) | | **Library** | Kernel32.lib | | **DLL** | Kernel32.dll |  另请参阅 [环境变量](https://learn.microsoft.com/zh-cn/windows/desktop/ProcThread/environment-variables)  [GetEnvironmentVariable](https://learn.microsoft.com/zh-cn/windows/desktop/api/winbase/nf-winbase-getenvironmentvariable) |

# 应用程序设置自己的环境变量实例

每个进程都有一个与之关联的环境块。 环境块由以 null 结尾的字符串块组成， (这意味着) 块末尾有两个 null 字节，其中每个字符串的格式为：

名字=价值

环境块中的所有字符串都必须按名称的字母顺序排序。 排序不区分大小写，Unicode 顺序不区分区域设置。 由于等号是分隔符，因此不得在环境变量的名称中使用。

## 示例 1

默认情况下，子进程继承父进程的环境块的副本。 以下示例演示如何使用 [**CreateProcess**](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa) 创建新的环境块以传递给子进程。

此示例使用示例 3 中的代码作为子进程，Ex3.exe。

C++

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

#include <strsafe.h>

#define BUFSIZE 4096

int \_tmain()

{

TCHAR chNewEnv[BUFSIZE];

LPTSTR lpszCurrentVariable;

DWORD dwFlags=0;

TCHAR szAppName[]=TEXT("ex3.exe");

STARTUPINFO si;

PROCESS\_INFORMATION pi;

BOOL fSuccess;

// Copy environment strings into an environment block.

lpszCurrentVariable = (LPTSTR) chNewEnv;

if (FAILED(StringCchCopy(lpszCurrentVariable, BUFSIZE, TEXT("MySetting=A"))))

{

printf("String copy failed\n");

return FALSE;

}

lpszCurrentVariable += lstrlen(lpszCurrentVariable) + 1;

if (FAILED(StringCchCopy(lpszCurrentVariable, BUFSIZE, TEXT("MyVersion=2"))))

{

printf("String copy failed\n");

return FALSE;

}

// Terminate the block with a NULL byte.

lpszCurrentVariable += lstrlen(lpszCurrentVariable) + 1;

\*lpszCurrentVariable = (TCHAR)0;

// Create the child process, specifying a new environment block.

SecureZeroMemory(&si, sizeof(STARTUPINFO));

si.cb = sizeof(STARTUPINFO);

#ifdef UNICODE

dwFlags = CREATE\_UNICODE\_ENVIRONMENT;

#endif

fSuccess = CreateProcess(szAppName, NULL, NULL, NULL, TRUE, dwFlags,

(LPVOID) chNewEnv, // new environment block

NULL, &si, &pi);

if (! fSuccess)

{

printf("CreateProcess failed (%d)\n", GetLastError());

return FALSE;

}

WaitForSingleObject(pi.hProcess, INFINITE);

return TRUE;

}

## 示例 2

在进程创建期间更改子进程的环境变量是一个进程直接更改另一个进程的环境变量的唯一方法。 进程永远无法直接更改不是该进程的子进程的另一个进程的环境变量。

如果希望子进程只需进行少量更改即可继承父进程的大多数环境，请使用 [**GetEnvironmentVariable**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-getenvironmentvariable) 检索当前值，保存这些值，为子进程创建要继承的更新块，创建子进程，然后使用 [**SetEnvironmentVariable**](https://learn.microsoft.com/zh-cn/windows/desktop/api/WinBase/nf-winbase-setenvironmentvariable) 还原保存的值，如以下示例所示。

此示例使用示例 3 中的代码作为子进程，Ex3.exe。

C++复制

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

#define BUFSIZE 4096

#define VARNAME TEXT("MyVariable")

int \_tmain()

{

DWORD dwRet, dwErr;

LPTSTR pszOldVal;

TCHAR szAppName[]=TEXT("ex3.exe");

DWORD dwFlags=0;

STARTUPINFO si;

PROCESS\_INFORMATION pi;

BOOL fExist, fSuccess;

// Retrieves the current value of the variable if it exists.

// Sets the variable to a new value, creates a child process,

// then uses SetEnvironmentVariable to restore the original

// value or delete it if it did not exist previously.

pszOldVal = (LPTSTR) malloc(BUFSIZE\*sizeof(TCHAR));

if(NULL == pszOldVal)

{

printf("Out of memory\n");

return FALSE;

}

dwRet = GetEnvironmentVariable(VARNAME, pszOldVal, BUFSIZE);

if(0 == dwRet)

{

dwErr = GetLastError();

if( ERROR\_ENVVAR\_NOT\_FOUND == dwErr )

{

printf("Environment variable does not exist.\n");

fExist=FALSE;

}

}

else if(BUFSIZE < dwRet)

{

pszOldVal = (LPTSTR) realloc(pszOldVal, dwRet\*sizeof(TCHAR));

if(NULL == pszOldVal)

{

printf("Out of memory\n");

return FALSE;

}

dwRet = GetEnvironmentVariable(VARNAME, pszOldVal, dwRet);

if(!dwRet)

{

printf("GetEnvironmentVariable failed (%d)\n", GetLastError());

return FALSE;

}

else fExist=TRUE;

}

else fExist=TRUE;

// Set a value for the child process to inherit.

if (! SetEnvironmentVariable(VARNAME, TEXT("Test")))

{

printf("SetEnvironmentVariable failed (%d)\n", GetLastError());

return FALSE;

}

// Create a child process.

SecureZeroMemory(&si, sizeof(STARTUPINFO));

si.cb = sizeof(STARTUPINFO);

#ifdef UNICODE

dwFlags = CREATE\_UNICODE\_ENVIRONMENT;

#endif

fSuccess = CreateProcess(szAppName, NULL, NULL, NULL, TRUE, dwFlags,

NULL, // inherit parent's environment

NULL, &si, &pi);

if (! fSuccess)

{

printf("CreateProcess failed (%d)\n", GetLastError());

}

WaitForSingleObject(pi.hProcess, INFINITE);

// Restore the original environment variable.

if(fExist)

{

if (! SetEnvironmentVariable(VARNAME, pszOldVal))

{

printf("SetEnvironmentVariable failed (%d)\n", GetLastError());

return FALSE;

}

}

else SetEnvironmentVariable(VARNAME, NULL);

free(pszOldVal);

return fSuccess;

}

## 示例 3

以下示例使用 [**GetEnvironmentStrings**](https://learn.microsoft.com/zh-cn/windows/win32/api/processenv/nf-processenv-getenvironmentstrings) 检索进程的环境块，并将内容打印到控制台。

C++复制

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

int \_tmain()

{

LPTSTR lpszVariable;

LPTCH lpvEnv;

// Get a pointer to the environment block.

lpvEnv = GetEnvironmentStrings();

// If the returned pointer is NULL, exit.

if (lpvEnv == NULL)

{

printf("GetEnvironmentStrings failed (%d)\n", GetLastError());

return 0;

}

// Variable strings are separated by NULL byte, and the block is

// terminated by a NULL byte.

lpszVariable = (LPTSTR) lpvEnv;

while (\*lpszVariable)

{

\_tprintf(TEXT("%s\n"), lpszVariable);

lpszVariable += lstrlen(lpszVariable) + 1;

}

FreeEnvironmentStrings(lpvEnv);

return 1;

}

# 演练

## 1.新建应该c++常规空项目，取名：Lesson59-enironment-variable,然后新建应该cpp源文件取名：enironment-variable.cpp,然后添加一些基本代码

|  |
| --- |
|  |

## 2.我们先来获取整个环境变量字符串，代码如下

|  |
| --- |
|  |

### 运行程序，此时只能看到第一个环境变量，它是应该特殊的标记

|  |
| --- |
|  |

## 3.如果想看所有的环境变量，必须使用循环，代码如下，注意写法有点特别需要适当记忆一下

|  |
| --- |
|  |

## 4.这里我们需要使用命令行参数，所以我们修改一下代码，我们把上面的代码封装到应该函数中

|  |
| --- |
|  |

## 5.然后我们来判断一下argv是什么字符串，根据它来做适当的操作，注意因为这个环境变量只有在程序运行的时候才有，我们需要在添加或者修改了变量后马上去获取或者删除，否则已经没有操作的必要了。因为程序退出，它就会消失。完整代码如下

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  DWORD SetProcVariable(char \*name,char \*value);  void DeleteOneVariable(char \*key);  void showAllEnvVariables()  {  //获取应该完整的环境变量字符串包含所有的环境变量  PVOID data = GetEnvironmentStrings();  LPTSTR szEnv;  for(szEnv=(LPTSTR)data;\*szEnv;)  {  printf("环境变量字符串：%s\n",szEnv);  while(\*szEnv++);  }  //显示完毕后，需要释放环境变量字符串  FreeEnvironmentStrings(LPCH(data));  }  void GetOneVariable(char\* key,char\* val ,int size)  {    if(0 == GetEnvironmentVariable(key,val,size))  {  printf("获取变量失败\n");  return ;  }  else  {  printf("%s:%s\n",key,val);  //使用了一次getchar()后，在使用第二次之前，需要先调用一次getchar让他先读取车，然后再次调用getchar函数并且接收结果  //这里输入没有调用，但是外部已经调用了一次，所以这一次需要清理  getchar();  DeleteOneVariable(key);  }    }  void DeleteOneVariable(char \*key)  {  printf("需要删除这个环境变量吗？(y/n): ");  if('y'==getchar())  {  printf("deleting\n");  SetProcVariable(key,NULL); //设置为NULL就是删除  printf("环境变量已经被删除\n");  showAllEnvVariables();  }  else  {  printf("\n\n");  showAllEnvVariables();  }  }    DWORD SetProcVariable(char \*name,char \*value)  {  if(!SetEnvironmentVariable(name,value))  {  printf("设置环境变量失败：%d\n",GetLastError());  return -1;  }  return 0;  }  int main(int argc,char \*argv[])  {  if(0==lstrcmp("all",argv[1])&& argc==2)//这里all表示显示所有  {  showAllEnvVariables();  }  else if(0==lstrcmp("add",argv[1])&& argc==4)//这里add表示新增，所以需要  {  SetProcVariable(argv[2],argv[3]);  //因为整个进程环境变量在程序结束时就会消失，  //所以我们需要在创建它以后马上调用查看所以进程环境变量的函数来查看  showAllEnvVariables();  printf("需要查找这个环境变量吗？(y/n): ");  if('y'==getchar())  {  printf("getting\n");  char val[256];  memset(val,0,256);  GetOneVariable(argv[2],val,256);//获取后可以在里面调用删除函数  }  }  else if(0==lstrcmp("set",argv[1])&& argc==4)//这里set表示修改，所以需要  {  SetProcVariable(argv[2],argv[3]);  //因为整个进程环境变量在程序结束时就会消失，  //所以我们需要在创建它以后马上调用查看所以进程环境变量的函数来查看  showAllEnvVariables();  printf("需要查找这个环境变量吗？(y/n): ");  if('y'==getchar())  {  printf("getting\n");  char val[256];  memset(val,0,256);  GetOneVariable(argv[2],val,256);//获取后可以在里面调用删除函数  }  }  system("pause");  return 0;  } |

# 扩展问题，GetEnvironmentVariable函数能否获取手动设置的环境变量？

## 1、我们来测试一下，先手动设置一个环境变量Kenny，值是Boss

|  |
| --- |
|  |

### 注意：需要按确定按钮保存

## 2.然后我们来修改代码

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  DWORD SetProcVariable(char \*name,char \*value);  void DeleteOneVariable(char \*key);  void showAllEnvVariables()  {  //获取应该完整的环境变量字符串包含所有的环境变量  PVOID data = GetEnvironmentStrings();  LPTSTR szEnv;  for(szEnv=(LPTSTR)data;\*szEnv;)  {  printf("环境变量字符串：%s\n",szEnv);  while(\*szEnv++);  }  //显示完毕后，需要释放环境变量字符串  FreeEnvironmentStrings(LPCH(data));  }  void GetOneVariableAndShow(char\* key,char\* val ,int size)  {    if(0 == GetEnvironmentVariable(key,val,size))  {  printf("获取变量失败\n");  return ;  }  else  {  printf("%s:%s\n",key,val);    }    }  void GetOneVariableDeleteOrNot(char\* key,char\* val ,int size)  {    if(0 == GetEnvironmentVariable(key,val,size))  {  printf("获取变量失败\n");  return ;  }  else  {  printf("%s:%s\n",key,val);  //使用了一次getchar()后，在使用第二次之前，需要先调用一次getchar让他先读取车，然后再次调用getchar函数并且接收结果  //这里输入没有调用，但是外部已经调用了一次，所以这一次需要清理  getchar();  DeleteOneVariable(key);  }    }  void DeleteOneVariable(char \*key)  {  printf("需要删除这个环境变量吗？(y/n): ");  if('y'==getchar())  {  printf("deleting\n");  SetProcVariable(key,NULL); //设置为NULL就是删除  printf("环境变量已经被删除\n");  showAllEnvVariables();  }  else  {  printf("\n\n");  showAllEnvVariables();  }  }    DWORD SetProcVariable(char \*name,char \*value)  {  if(!SetEnvironmentVariable(name,value))  {  printf("设置环境变量失败：%d\n",GetLastError());  return -1;  }  return 0;  }  int main(int argc,char \*argv[])  {  if(0==lstrcmp("all",argv[1])&& argc==2)//这里all表示显示所有  {  showAllEnvVariables();  }  else if(0==lstrcmp("add",argv[1])&& argc==4)//这里add表示新增，所以需要  {  SetProcVariable(argv[2],argv[3]);  //因为整个进程环境变量在程序结束时就会消失，  //所以我们需要在创建它以后马上调用查看所以进程环境变量的函数来查看  showAllEnvVariables();  printf("需要查找这个环境变量吗？(y/n): ");  if('y'==getchar())  {  printf("getting\n");  char val[256];  memset(val,0,256);  GetOneVariableDeleteOrNot(argv[2],val,256);//获取后可以在里面调用删除函数  }  }  else if(0==lstrcmp("set",argv[1])&& argc==4)//这里set表示修改，所以需要  {  SetProcVariable(argv[2],argv[3]);  //因为整个进程环境变量在程序结束时就会消失，  //所以我们需要在创建它以后马上调用查看所以进程环境变量的函数来查看  showAllEnvVariables();  printf("需要查找这个环境变量吗？(y/n): ");  if('y'==getchar())  {  printf("getting\n");  char val[256];  memset(val,0,256);  GetOneVariableDeleteOrNot(argv[2],val,256);//获取后可以在里面调用删除函数  }  }  else if(0==lstrcmp("get",argv[1])&& argc==3)//这里set表示修改，所以需要  {  printf("getting\n");  char val[256];  memset(val,0,256);  GetOneVariableAndShow(argv[2],val,256);//获取后可以在里面调用删除函数  }  system("pause");  return 0;  } |

### 运行程序，其实是获取得到的

|  |
| --- |
|  |

### 用SetEnvironmentVariable函数设置已经存在的环境变量是没有用的

## 也就是说虽然无法修改系统里面的环境变量，但是可以获取所以系统环境变量到一个字符串，也可以获取特定的环境变量，这在很多情形下面还是很有用的，还有获取环境变量时是不区分大小写的。

### 需要注意的是：有时候我们修改环境变量，不是把他原来的值直接覆盖掉，而是在原来的基础上添加新内容，新内容和原来的内容用分号隔开

# 扩展，这个项目可能用MFC对话框程序来做比较好