**MinGW-W64使得printf、cout、wprintf、wcout显示出中文的种种**

2014年11月29日 14:33:05 [ligand](https://me.csdn.net/ligand) 阅读数：2457 标签： [c++](http://so.csdn.net/so/search/s.do?q=c++&t=blog)[VisualC++](http://so.csdn.net/so/search/s.do?q=VisualC++&t=blog)[gcc](http://so.csdn.net/so/search/s.do?q=gcc&t=blog)[mingw](http://so.csdn.net/so/search/s.do?q=mingw&t=blog)[iostream](http://so.csdn.net/so/search/s.do?q=iostream&t=blog)更多

个人分类： [C++](https://blog.csdn.net/ligand/article/category/2333999)

使用MinGW-W64-gcc 4.9.2 的64位 posix线程 seh异常机制的编译器。它内置了iconv支持，可以在UTF-8与gbk之间来回转码，这个很重要。

说明：printf、cout、wprintf、wcout这些如何输出中文，C/C++语言标准完全没提其实现细节。所以这些是针对具体的编译器环境而不同的。MinGW-W64、Linux GCC、Microsoft Visual C++都有所出入。

1  为使得printf显示出中文，仅需要

gcc 命令行选项-fexec-charset=gbk

这意味着编译器生成的可执行文件的字符串的内容都是按照gbk编码的。在Windows环境下调用相应的C的stdout，正确显示毫无问题。

2  为使得cout显示出中文，仅需要

gcc 命令行选项-fexec-charset=gbk

这意味着cout应该是完全基于C的stdout实现。

3    使wprintf能输出汉字，需要且仅需要这一行：

\_setmode(\_fileno(stdout), \_O\_WTEXT);            // 需要 #include <io.h> 和 <fcntl.h>

及gcc 命令行选项-fexec-charset=gbk

但是在Visual C++与Linux gcc中，都不支持上述\_setmode函数，虽然这两个编译器已经充分实现了wprintf、wcout输出中文。

但是，上述\_setmode语句会造成wcout输出中文与英文字符时一片乱码！

可见，wprintf是使用了C语言的标准输出stdout。仅此而已。至于stdout应该采取的默认编码字符集，由gcc 命令行选项-fexec-charset=gbk指定即可。

4  为使得wcout输出中文，仅需要下述2条语句：

setlocale(LC\_ALL, "");                          // 需要 #include <locale>

std::ios\_base::sync\_with\_stdio(false);       // 缺少的话，wcout输出的 wchar\_t字符串会漏掉中文字符

    /\*  ios\_base::sync\_with\_stdio 决定 C++ 标准 streams (cin,cout,cerr...) 是否  
    \*  与相应的 C 标准程序库文件 (stdin,stdout,stderr) 同步，  
    \*  也就是是否使用相同的 stream 缓冲区，缺省情况是同步的。  
    \*  但由于同步会带来某些不必要的负担，因此该函数作用就是我们自己可以取消同步  
    \*  std::ios::sync\_with\_stdio(false);  
    \*  注意:必须在任何 io 操作之前取消同步  
    \*/

完全不需要  std::wcout.imbue(std::locale(""));           // 使 wcout 使用操作系环境中缺省的 locale

由此可见，wcout是独立于C的标准输出stdout而独立实现的；且在把宽字符转换为操作系统默认字符编码时，使用了C语言标准的locale设置（即setlocale函数），没有setlocale函数的设置，仅靠gcc命令行选项gcc 命令行选项-fexec-charset=gbk是不行的。

仅此而已。

5  示例程序

1. #include <iostream>
2. #include <locale>
3. #include <cstdio>
4. #include <io.h>
5. #include <fcntl.h>
7. int main(void)
8. {
9. printf("printf的内容\n");
11. std::cout << "cout 的内容" << std::endl;

14. *//fflush(stdout); //must be done before \_setmode 实测不需要这行*
15. *//\_setmode(\_fileno(stdout), \_O\_WTEXT);            // 需要 #include <io.h> 和 <fcntl.h>*
16. wchar\_t t[] = L"pure english\n";
17. wprintf(L"wprintf: %ls", t);
18. wprintf(L"wprintf: C 宽字符字符串\n");

21. *//setlocale(LC\_ALL, "");                          // 需要 #include <locale>*
22. *//std::ios\_base::sync\_with\_stdio(false);       // 缺少的话，wcout wchar\_t 会漏掉中文*
23. */\*  ios\_base::sync\_with\_stdio 决定 C++ 标准 streams (cin,cout,cerr...) 是否*
24. *\*  与相应的 C 标准程序库文件 (stdin,stdout,stderr) 同步，*
25. *\*  也就是是否使用相同的 stream 缓冲区，缺省情况是同步的。*
26. *\*  但由于同步会带来某些不必要的负担，因此该函数作用就是我们自己可以取消同步*
27. *\*  std::ios::sync\_with\_stdio(false);*
28. *\*  注意:必须在任何 io 操作之前取消同步*
29. *\*/*
30. *//std::wcout.imbue(std::locale(""));           // 使 wcout 使用客户环境中缺省的 locale*
32. std::wcout << L"wcout: pure english" << std::endl;
33. std::wstring u = L"wcout: C++ 宽字符版 string";
34. std::wcout << L"wstring u 的内容是：" << u << std::endl;
35. std::wcout << L"wstring u 的字符数为：" << u.size() << std::endl;
37. return 0;
39. }

ps.

On Windows, the only way to achieve direct Unicode output is via WriteConsoleW(). The MS CRT (2008 and newer) provides a way to use C/C++ I/O facilities for direct Unicode output:

Code:

[?](http://cboard.cprogramming.com/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <fcntl.h>  #include <io.h>  #include <cstdio>  #include <cwchar>  #include <iostream>  using namespace std;  int main()  {      \_setmode(\_fileno(stdout), \_O\_U16TEXT);      fputws(L"\u00e7\n", stdout);      wcout << L"\u00e7" << endl;      return 0;  }//main |

This will send the UTF16LE string directly to WriteConsoleW(), unless output is redirected to a file, in which case the UTF16LE string is written as a stream of bytes via WriteFile().