Windows
DEVELOPER'S JOURNAL

Windows NT/2000 编程实践

[美] 保拉·汤姆林森等 著
周济 译

R&D BOOKS

中国电力出版社
www.infopower.com.cn

Windows NT /2000

编程实践

（美）保拉·汤姆林森等著　周济译

中国电力出版社

内 容 提 要

本书由多种著名杂志上的精品文章荟萃而成，是集多位专家多年经验、精心编著的基于 Windows NT/2000 平台编程的一本实用的指导书籍。内容包括内存管理、进程管理、事件日志、远程调用、多任务、Windows 版本检测、软件国际化、跨平台通信、调试、实例等。本书几乎涵盖了程序开发的各个层面，内容全面翔实，可读性较高。

本书适合程序员及大专院校师生阅读。

图书在版编目（CIP）数据

Windows NT/2000 编程实践/（美）汤姆林森等编著；周济译-北京：中国电力出版社，2001.3

ISBN 7-5083-0553-1

Ⅰ.W… Ⅱ.①汤…②周… Ⅲ.计算机网络-操作系统（软件），Windows NT/2000 Ⅳ.TP316.86

中国版本图书馆 CIP 数据核字（2001）第 10839 号

北京版权局著作权登记号 01-1999-3846

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

实验小学印刷厂印刷

各地新华书店经售

2001 年 3 月第一版 2001 年 3 月北京第一次印刷

787 毫米×1092 毫米 16 开本 29.75 印张 675 千字

定价 **45.00** 元

目　　录

第 1 章

编写可移植的 Windows 应用程序

David Van Camp

如果你是一个准备开始开发 Windows NT/2000 应用程序的程序员，或者你至少在考虑这么做，你最可能问的两个问题就是：1）移植到 Windows NT/2000 有多难？ 2）为了简化将目前的 Windows 3.x 应用程序移植到 NT/2000 的工作，我在开发过程中能够做哪些准备工作？

显然，将任何 Windows 应用程序移植到 NT/2000 的难易程度取决于程序员的水平，当然还取决于采用的开发技术。简单的应用程序，例如像 Windows 提供的应用程序，通常是很好移植的。（微软声称，在用户界面仍然运行的情况下，一周之内就完成了文件管理器的移植。）不过，对于大型的应用程序，移植过程是非常复杂和花费时间的。

为什么移植是困难的

尽管 NT/2000 支持的 32 位的 Windows API (Win32)与 16 位的 Windows 3.1 SDK (Win16)之间存在着细微差别，这两个平台是相同的。你可能遇到的最大困难就是你的应用程序必须同时在两个平台下开发或者维护。

不过，遵循下页中总结的准则，可以使你在编写 Win16 代码时能够跃过潜在的移植问题。这些准则的大多数都是基于我编写 NT/2000 应用程序的经验。而这些应用程序要求源代码唯一并且能够兼容 Win32（例如多磁带备份系统）。

实际上，如果你不知道问题出在何处或者是什么导致了问题，不管这个问题多么简单，你都不可能解决这个问题。因此，很难发现的移植性问题是最大的麻烦。而如果问题容易被发现，即使这些问题需要大量工作来改正，这些问题也是不太麻烦的。所以，你的一个基本策略就应该是使得这些不可能消除的问题尽可能地容易被发现和改正（尽管你还是想消除尽可能多的问题）。想要做到这一点的方法之一就是用 NTPORT 宏给代码作标记（见程序清单 1.1）。你可以用程序清单 1.1(a)中的 NTPORT 宏标记移植到 NT/2000 可能会产生

问题的 Win16 代码。这个宏应该定义在一个头文件中，而这个头文件又应该包含在所有的 C 语言文件中。如程序清单 1.1 (b)所示，你应该将宏放在靠近可疑的问题之处并且加上一段简短的说明。这样，当移植到 NT/2000 时，通过查找 C 语言文件中的 NTPORT，或者通过检查编译 NT/2000 代码时编译器的警告信息，你就可以很容易地发现问题。

编写可移植的 Windows 应用程序的规则

1. 消除所有的编译器和链接器警告。
2. 用 NTPORT 宏标记所有可能出现的移植性问题。
3. 只在必要时使用 WORD 类型数据；不是必需的话，使用 INT 或 UINT。
4. 用 SetClassLong 或 SetWindowLong 存储加宽的类型。
5. 不要为短（16 位）数据类型指定句柄或指针。
6. 对句柄使用统一的类型（HWND、HPEN、HBRUSH 等等）。
7. 尽量避免使用过时的 API 过程。
8. 把所有的窗口和对话框过程声明为可移植的。
9. 用消息处理器处理窗口消息。
10. 不要把加宽的类型放进 lParam 字。
11. 不要使用与模块相关的或分段的地址。
12. 把使用全局共享的内存的操作隔离开。
13. 不要假设文件命名遵循 DOS 命名惯例。
14. 不要读或写系统文件。
15. 不要寄希望于定位在某个特定字节的数据元素。

避免移植性问题

尽管标记潜在的问题使得它们更加容易被发现，你仍然可以通过避免这些问题而节省更多的时间。对于任何多平台项目，如果你坚持结构化编程技术，确保你的代码编译过程中没有警告和错误，你就可以预防很多问题。编译你的 Windows 代码时一定要使用--W4（4 级警告）和--DSTRICT 命令行选项。你使用--W4 时，编译器会提醒你任何它认为可疑的操作。而--DSTRICT 选项使最严格的可能类型检查发挥作用，并且禁止许多使用不兼容数据类型的不可移植操作。

对于 NT/2000 来说，许多 Win16 的系统资源都改变了。不要直接读写系统文件或者包含可执行文件和资源，因为这些文件的二进制格式可能已经发生了变化。并且，许多系统对象的格式也改变了，你不应该直接访问它们。决不要试图从应用程序直接访问一个设备端口或者任何系统代码；通过设备驱动来实现这一点。一定要使用 Windows API 过程进行这些类型的操作，而不要使用未公开的调用和数据格式。

程序清单 1.1　用 NTPORT 宏标记不可移植的代码

(a)

```
#pragma NTPORT("warning, pointer stored in globally shared memory")
      gpszGlobalString = szLocalString;
```

(b)

```
#ifdef _WINNT_   /* if compiling for Windows NT/2000, generate a compiler
warning */
#define NTPORT(msg)  message(__FILE__  " NTPORT: " msg)
#else           /* we are compiling for Win16, so do nothing */
#define NTPORT(msg)
#endif
```

Windows API 中被删除的过程

(a) 下列过程已经被 32 位 Windows API 抛弃。它们无法被替代。因此，调用这些过程的代码不能被移植。

AccessResource	AllocDSToCSAlias	AllocResource
AllocSelector	ChangeSelector	GetCodeHandle
GetCodeInfo	GetCurrentPDB	GetEnvironment
GetInstanceData	GetKBCodePage	GetTempDrive
GlobalDosAlloc	GlobalDosFree	GlobalPageLock
GlobalPageUnlock	LimitEMSPages	LocalNotify
SetEnvironment	SetResourceHandler	SwitchStacKBack
SwitchStackTo	UngetCommChar	ValidateCodeSegment
ValidateFreeSpaces		

(b) 这些音频过程已经被抛弃；应该用多媒体音频支持 API。

CloseSound	CountVoiceNotes	GetThresholdEvent
GetThresholdStatus	OpenSound	SetSoundNoise
SetVoiceAccent	SetVoiceEnvelope	SetVoiceNote
SetVoiceQueueSize	SetVoiceSound	SetVoiceThreshold
StartSound	StopSound	SyncAllVoices
WaitSoundState		

Windows API 的变化

在 NT/2000 中，许多 Windows API 过程都扩展到 32 位。在大多数情况下，这意味着所有的指针都是 32 位，并且许多 WORD 类型的参数变成 UNIT 类型。总之，除非必要（例如一个过程的参数需要一个指向 WORD 类型的指针）应该尽量避免使用 WORD 数据类型。图形坐标应该用 UNIT 声明其类型；一般整数和数组下标应该用 INT。绝不要给一个 WORD 或者其他 16 位类型，特别是指针、整数或者任何类型的句柄，使用扩展的类型。不要把句柄类型混淆——HANDLE、HWND、HINSTANCE 和 HDC 都是单独的、不同的数据类型，不能相互转换。

一些 Win16 过程都被替换、修正或者删除了。对于被替换的函数，在移植到 NT/2000 时通过以下两种方法之一可以容易地找到并且修正调用：编写与新参数匹配的替代函数并调用新的 API 过程，或者调用替代的 Win32 过程。创建这些过程的 Win16 版本就可以与原来函数的参数匹配。采用上述两种方法的任何一种都不是什么问题。但是，对于那些功能发生变化的过程，问题就严重得多。最后，已经完全删除的过程就不应该再使用。

在修正过的 API 过程中，最重要的是回调函数，特别是窗口和对话过程。你在声明一个指向回调过程的指针时，一定要使用适当的类型，例如窗口过程使用 WNDPROC，对话过程使用 DLG-PROC，钩子过程使用 HOOK-PROC 等等。不要使用 FARPROC 或者 NEARPROC。并且，使用正确的函数原型声明这些函数。一定要这样声明窗口和对话过程：

```
RESULT CALLBACK ProcName (HWND hwnd, UINT wMsg, WPARAM wParam, LPARAM lParam)
```

这里，对于窗口过程来说，RESULT 是 LRESULT；对于对话过程来说，RESULT 是 BOOL。Win32 中 BOOL 和 wMsg 与 wParam（原来是 WORD 类型）都扩展到 32 位。Win16 或者 Win32 版本的 windows.h 中有完整的回调类型的列表。另外，要使用 CALLBACK 而不是 FAR PASCAL，因为 CALLBACK 是一个移植性更好的修饰符。

消息处理的修正

由于一些消息的参数的存储方式改变，窗口过程的声明原型也发生了变化。对于大多数消息这些改变是必需的。因为扩展了的值，通常是 HWND，在 16 位下是放在 LPARAM 的前 16 位和后 16 位。由于这些值被扩展了，它们一般变成了扩展了的 WPARAM 参数。另外，其他的参数被移动了。因此，你不可能编写可移植代码，从修正后的消息的参数中直接提取一个值。微软提供了一些统称为“消息破解器（message crackers）”的宏，这些宏可以提供不同的解决方法。

最早在 Windows 3.1 SDK 中引入的一对消息破解器被称为“处理器和转发器（handlers

and forwarders)"。我提倡在编写新的处理窗口消息的代码时使用这些宏。因为，这些宏不仅解决了 Win32 引起的参数压缩方式的改变，它们还提供了一种高度结构化的消息处理方法。这些在 windows.h 中定义的宏，允许一种基本上是面向对象的方法来处理消息，就像 Microsoft C++ Foundation Classes 采用的解决方法那样。程序清单 1.2 是一个 Win16 下的 WM_COMMAND 消息处理例子。程序清单 1.3 说明了相同的代码是如何使用消息处理器和转发器实现的。这些宏对消息处理器使用了如下的命名约定：

```
HANDLE_message (hwnd, wParam, lParam, function_name);
```

在这个约定中，message 是窗口消息标记符，function_name 是处理器函数的名称。这个宏将 lParam 和 wParam 中的参数解压，并调用你的处理器函数。一定要按照 windows.h 中消息处理器宏定义前的注释声明处理器函数。消息转发器的命名约定是：

```
FORWARD_message (paramlist, message_proc);
```

这里，paramlist 是特定消息所需要的参数列表，message_proc 是一个消息传递过程（SendMessage、PostMessage、CallWindowProc 等等）。这个宏将参数压入 lParam 和 wParam，并调用指定的过程。如果想知道更多的信息和这些宏的完整列表，请参阅微软 Windows SDK 文档和 windows.h。

另一对比处理器和转运器更简单的消息破解器通过一个可移植的宏将消息的参数压缩或者解压。程序清单 1.4 展示了 WM_COMMAND 如何使用压缩器和解压器进行处理的。解压器的命名约定是：

```
GET_message_item (wParam, lParam);
```

这里，message 是消息标记符，item 是你想从参数中解压的数据项。返回值类型取决于解压数据的类型。消息压缩器的命名约定如下：

```
GET_message_MPS (wParam, lParam, paramlist);
```

这里，paramlist 是特定消息所需要的参数的列表。

对于 Win32，微软只提供了处理那些参数压缩改变了的消息的宏。微软并没有为 Win16 提供这些宏的定义，所以我给出这些定义（程序清单 1.5）。因为只对变化了的消息定义了宏，这些代码可以用来快速查找变化了的消息。由于使用这些宏进行转化比使用处理器和转运器工作量要小，这些宏最适于将现有的代码转向 Windows NT/2000。处理器和转运器则适用于新开发，因为它们能适用于所有窗口消息并且提供了高度结构化的解决方法。

在程序清单 1.5 中，WM_CTLCOLOR 消息宏是上述命名约定的一个例外。这个 Win16 的消息提出了一个问题：它包含了两个扩展到 32 位的参数和一个 16 位的参数。因此，由于空间的紧张，这个消息必须被分解成一系列消息。当声明你自己的消息时，不要把两个以上扩展了的类型或者一个扩展的、一个 16 位的值压缩进消息参数。

Win32 动态数据交换（DDE）消息发生了相当大的变化，因此，实际上是不可能编写

出处理这些消息的可移植代码来的。所以，一定要使用高层 DDEML 过程实现 DDE 函数。消息处理的其他方面也发生了变化。在 Win32 中是不可能将属于另一个进程的窗口作为子类的。另外，只有在系统初始化就装载的 DLL 中才能注册全局类，而绝不能在应用程序中这么做。要尽量避免使用不可移植的技术。

程序清单 1.2　Win16 中的 WM_COMMAND 消息处理

```
#include <windows.h>          /* normal include for all windows applications*/

/*The following window procedure declaration is NOT portable to WindowsNT/2000! */

LONG FAR PASCAL MyWinProc (HWND hwnd, WORD wMsg, WORD wParam, WORD wParam)
{
   switch ( wMsg )
      {
          case WM_COMMAND:
          /* Nonportable reference to control ID in message params */
          switch ( wParam )
          {
          /* processing for WM_COMMAND based on control ID goes here...*/
          }
          case WM_SOMEMESSAGE:
          /* non-portable method to send WM_COMMAND message to parent */
          SendMessage ( GetParent (hwnd), WM_COMMAND, wMyID,
                    MAKELONG (hwnd, wNotifyCode) );
      }
   }
```

程序清单 1.3　使用处理器和转运器的 WM_COMMAND 消息处理

```
#include <windows.h>      /* normal include for all windows applications */
#include <windowsx.h>     /* include macro definitions for Win16 or NT  */

/* Declare portable WM_COMMAND message handler function...  */

void MyWinProc_OnCommand (HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
   /* Portable reference to control ID in message params    */
   switch ( id )
      {
         /* processing for WM_COMMAND based on control ID goes here... */
      }
```

```
    }

/* The following window procedure declaration IS portable to Windows NT/2000! */

    LRESULT CALLBACK MyWinProc (HWND hwnd, UINT wMsg, WPARAM wParam, LPARAM wParam)
    {
       switch ( wMsg )
          {
             case WM_COMMAND:
             /* Portable WM_COMMAND processing using macro... */
             return HANDLE_WM_COMMAND (hwnd, wParam, lParam, MyWinProc_OnCommand );
             case WM_SOMEMESSAGE:
             /* portable method to send WM_COMMAND message to parent */
             FORWARD_WM_COMMAND ( GetParent (hwnd), wMyID, hwnd, wNotifyCode,
                                  SendMessage);
          }
    }
```

程序清单 1.4　使用压缩器和解压器的 WM_COMMAND 消息处理

```
#include <windows.h>   /* normal include for all windows applications     */
#include <windowsx.h>  /* include macro definitions for NT/2000 only      */
#include <move2nt.h>   /* include macro definitions for Win16             */

/* The following window procedure declaration IS portable to Windows NT/2000!  */

LRESULT CALLBACK MyWinProc (HWND hwnd, UINT wMsg, WPARAM wParam, LPARAM wParam)
{
     switch ( wMsg )
     {
        case WM_COMMAND:
        /* Portable reference to control ID in message params      */
        switch ( GET_WM_COMMAND_ID (wParam, lParam) )
        {
        /* processing for WM_COMMAND based on control ID goes here */
        /* using GET_WM_COMMAND_xxx macros for portability...      */
        }
        case WM_SOMEMESSAGE:
        /*  Portable method to send WM_COMMAND message to parent    */
           SendMessage ( GetParent (hwnd), WM_COMMAND,
                         GET_WM_COMMAND_MPS (wMyID,  hwnd, wNotifyCode));
     }
}
```