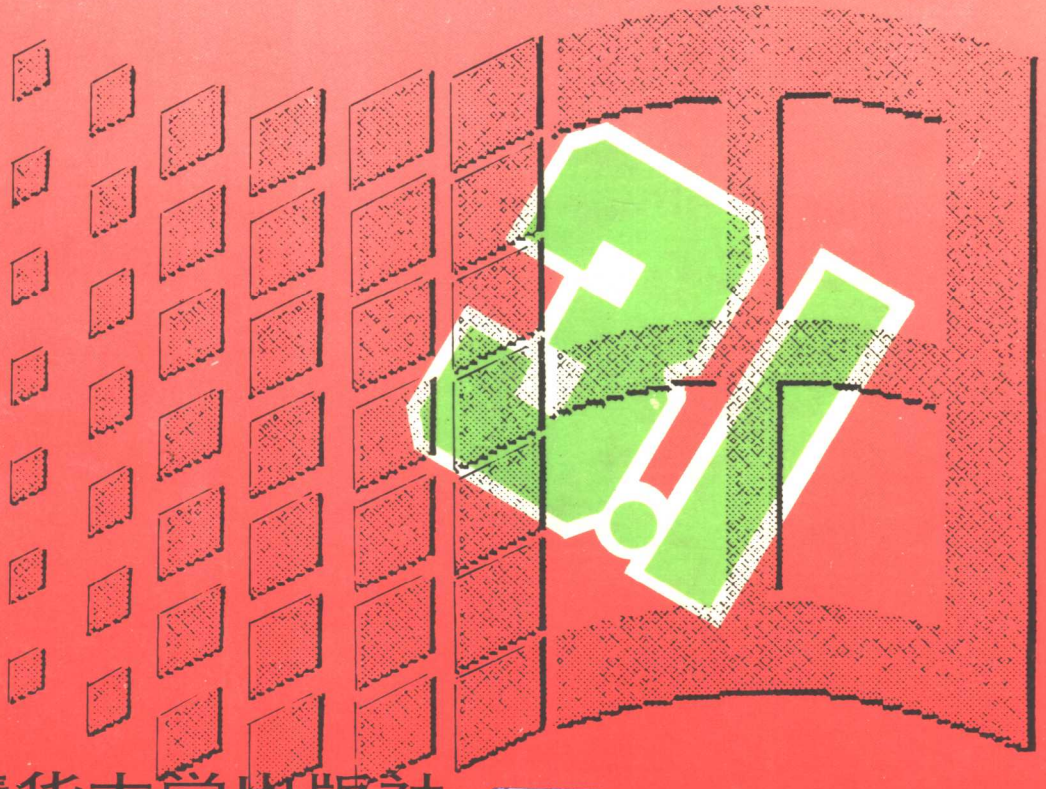
Microsoft Microsoft Microsoft

Microsoft Windows 3.1

程序员参考大全(一)

—综 述

沈金发 李莉 陈建伟 李逸 译



清华大学出版社

Microsoft

Windows 3.1

程序员参考大全(一)
—综述

[美] Microsoft 公司 著
沈金发 李莉 陈建伟 李逸 译

清华大学出版社

Microsoft Windows 3.1 Programmer's Reference
Volume 1—Overview

(京)新登字158号

Microsoft Windows 3.1
程序员参考大全(一)——综述
沈金发　李莉　陈建伟　李逸　译
☆
清华大学出版社出版
北京　清华园
清华大学印刷厂印刷
新华书店总店科技发行所发行
☆
开本：787×1092 1/16　印张：21　字数：490千字
1993年6月第1版　1993年6月第1次印刷
印数：0001—5000
ISBN 7-302-01222-9/TP·460
定价：36.00元

引　言

《Microsoft Windows 3.1程序员参考大全(一)——综述》介绍了各种接口函数和Microsoft Windows操作系统支持的扩展库；还包括描述Windows应用程序特性的应用程序注解。附录中列出了Windows函数的模块和库名。

第一部分"窗口管理、图形和系统服务"介绍了有关窗口管理、图形输出和系统服务的函数。窗口管理函数处理消息；创建、移动或切换窗口；产生系统输出。图形设备接口(GDI)函数完成与设备无关的图形操作，如在不同的输出设备上产生线、字符和位图输出。系统服务函数涉及到访问模块的数据和代码；分配和管理内存；翻译字符串以及创建、打开文件之类的操作。

第二部分"扩展库"描述了支持WindowsV3.1许多新特性的库。这些新特性包括公共对话框；简化动态数据交换(DDE)的管理函数、对象链接与嵌入(OLE)；注册数据库和拖动等Shell增强特性；可使Windows-hosted工具的创建得以简化的工具帮助函数；数据解压缩函数；人为消耗系统资源并用于调试应用程序的资源紧张测试实用程序；文件安装函数；利用80386和80486处理器32位内存寻址能力的函数；浮点仿真函数及Control Panel中的屏幕节约程序。

第三部分"应用程序注解"介绍了应用程序应该用来实现一些Windows特性和增强功能的技术。手册的这一部分说明了如何创建Control Panel应用程序；如何创建和安装File Manager的扩展功能；如何使用ProgramManager的动态数据交换接口；如何编制与国家、语言无关的应用程序；如何编写网络应用程序；如何在Windows应用程序中使用Microsoft MS-DOS函数；如何编写产生Windows前置和后续代码的编译程序；如何初始化并启动Windows应用程序；如何改善Windows应用程序的视觉效果；如何编写自装载Windows应用程序；以及如何与可安装驱动器进行交互。

/ 附录中列出了每个Windows函数的模块名和库名。

符号约定

本手册中使用下列约定来定义语法：

约定	含　义
黑体	表示要照原样键入的词或字符，例如，资源定义语句或函数名——如**MENU**或**CreateWindows**、MOS-DOS命令或命令行选择项。
斜体	表示一个位置或变量，使用时必须以实际值代替。如，语句**SetCursorPos**(*x*,*y*)，就要求用实际值去替换*x*,*y*。
[]	可选参数。
|	分隔或选择项。

（续表）

约定	含　义
...	表示前面的项可以重复。
BEGIN . . . *END*	表示应用程序例子的省略部分。

另外，还有些约定用来帮助你更好地理解本手册：

约　定	含　义
小号大写字符	表示键名、顺序键或组合键。如，ALT+SPACEBAR。
正常大写字母	表示文件名和路径，大部分类型、结构名（同时也是黑体）和常数。
空格	分隔代码和表示语法间隔符。

目　录

第二部分 扩 展 库

第三部分 应用程序注释

第一部分

窗口管理、图形和系统服务

第 1 章

窗口管理

本章描述了 Microsoft Windows 操作系统中处理消息；创建、移动或更换窗口；或产生系统输出的一系列函数。这些函数组成了窗口管理程序接口。

1.1 消　息

消息是一个应用程序的输入。它们表明了需要应用程序做出响应的某种事件。消息由消息标识和消息参数构成，参数的内容随消息种类不同而不同。

1.1.1 消息的产生和处理

Windows 为每一个输入事件产生一个输入消息，如用户移动鼠标或按下一个按键都会产生输入消息。Windows 把输入消息收集在系统消息队列中，然后把这些消息，包括定时器和绘画消息，送到应用程序的消息队列中。应用程序消息队列遵循先进先出的原则，而定时器和绘画消息是例外的，这两类消息将保留在应用程序消息队列中直到应用程序处理完所有其它的消息。Windows 把属于特定应用程序的消息放置到相应的应用程序消息队列中。应用程序调用 **GetMessage** 函数读取消息，并用 **DispatchMessage** 函数把这些消息分发到相应的窗口过程。

Windows 可以把有些消息直接发送到相应应用程序的窗口过程，而无需先放置到应用程序消息队列中。这样的消息称之为非排队消息。一般情况下，非排队消息是只影响其窗口的消息。**SendMessage** 函数用于把消息直接发送至窗口过程。有关窗口过程的进一步情况，参见第 1.2.13 节“窗口过程”。

例如：**CreateWindow** 函数使 Windows 发送一个 WM_CREATE 消息到应用程序的窗口过程，并等待窗口过程处理完这条消息。也就是说，Windows 直接发送这条消息到窗口过程，并不把它放到应用程序消息队列中。

尽管消息大部分都由 Windows 产生，但应用程序本身也能产生自己的消息，并把它们放在自己或其他应用程序的消息队列中。

一般情况下，应用程序使用 **GetMessage** 函数从应用程序消息队列中移走消息，函数 **GetMessage** 在 WinMain 的一个循环语句中，这个循环叫做主消息环。**GetMessage** 函数在应用程序消息队列中搜索消息；如果有消息，就返回队列中最上面的消息；如果消息队列是空的，**GetMessage** 就处于等待状态。

处于等待状态的 **GetMessage** 把控制权交回 Windows，以便其它应用程序得到控制权来处理它们自己的消息。

一旦某个应用程序的 **WinMain** 函数从应用程序消息队列中获得了消息，就用函数

DispatchMessage 把消息发送到窗口过程。这个函数使 Windows 调用与消息有关窗口的窗口过程，并把消息的内容作为函数的参数；然后，窗口过程即可对消息进行处理，并对窗口实现所要求的改变。窗口过程返回后，Windows 即把控制权交给 **WinMain** 函数中的主消息环。这样主消息环即可获得队列中下一条消息。

注意：除非特别说明，Windows 可以按任意次序发送消息；应用程序接收消息也无特定的次序。

用户每按一次键，Windows 即产生一个消息，这个消息含有一个虚键值，表明哪一个键被按过，但并不一定等于该键的字符值。而要想得到字符值，需要在 **WinMain** 函数的主消息环中调用 **TranslateMessage** 函数对虚键值消息进行翻译。此函数把另一条含有相应字符值的消息送到应用程序消息队列中，然后，这条消息即可被分发到窗口过程。

1.1.2 翻译消息

通常，WinMain 函数应使用 **TranslateMessage** 函数对每一条消息进行翻译，而并不仅仅是虚键消息。尽管 **TranslateMessage** 函数对其他类型的消息没有作用，但这样做能保证准确地对键盘输入进行翻译。

下面的例子是一个典型的主消息环，**WinMain** 函数用它从应用程序消息队列中获得消息，并分发到应用程序的窗口过程。

```
int PASCAL WinMain(hinst, hPrevInst, lpCmdLine, ShowCmd)
HINSTANCE hinst;
HINSTANCE hPrevInst;
LPSTR lpCmdLine;
int ShowCmd;
{
    MSG msg;
     .
     .
     .
 while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
 }
```

使用加速键的应用程序必须首先用 **LoadAccelerators** 函数装入资源定义文件中的加速键表，然后用 **TranslateAccelerator** 函数把键盘消息翻译成加速键消息。有关加速键的内容，参见《Microsoft Windows 3.1 程序员参考大全(六)——编程指南》。

一个使用加速键的应用程序，其主消息环应具有下列格式：

```
while (GetMessage(&msg, NULL, 0, 0)) {
    if (TranslateAccelerator(hwnd, haccel, &msg) == 0) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
return msg.wParam;
```

TranslateAccelerator 函数应该放在标准的 **TranslateMessage** 和 **DispatchMessage** 函数的前面。而且，因为 **TranslateAccelerator** 自动地发送加速键消息到相应的窗口过程，所以，**TranslateAccelerator** 返回一个非零值时，就无需再调用 **TranslateMessage** 和 **DispatchMessage** 函数。

1.1.3 检查消息

应用程序可以通过 **PeekMessage** 函数检查其消息队列中的指定消息，而又不把它从队列中删除。如果队列中存在所要检索的消息，则返回一个非零值，这样就可以不必通过主消息环获取并处理这条消息。

一般情况下，当应用程序正实现一个长时操作时，如处理输入、输出时，应用程序用 **PeekMessage** 对消息进行定时检查。例如：这个函数可以用来检测操作的结束。如果消息队列中没有消息的话，**PeekMessage** 也给了应用程序一个放弃控制的机会，因为消息队列中没有消息时，函数 **PeekMessage** 可以放弃控制。

1.1.4 发送消息

应用程序用 **SendMessage** 和 **PostMessage** 函数发送消息至自己的窗口或其它应用程序的窗口。**PostAppMessage** 函数与 **PostMessage** 的不同之处是通过应用程序的模块句柄而非窗口句柄来发送消息。

PostMessage 函数使 Windows 投递消息，也就是把消息放到应用程序消息队列中。**PostMessage** 函数立即返回控制给调用应用程序；而在消息未被读出队列之前，由消息指定的任何操作都不会发生。

SendMessage 函数使 Windows 把消息直接发送到给定的窗口过程，而不通过应用程序的消息队列。窗口过程接收并处理完该消息或者调用 **ReplyMessage** 函数返回控制之前，Windows 不会将控制返回调用应用程序。

如果应用程序需要消息的返回值，那么发送该消息时，就必须用 **SendMessage** 函数；**SendMessage** 的返回值与处理消息的窗口过程的返回值相同。函数 **PostMessage** 发送消息之后就立即返回，它的返回值仅是一个布尔数，表明消息是否被成功地放进到队列中，并不表明消息被如何处理。

1.1.5 避免消息死锁

应用程序在处理由 **SendMessage** 函数从其他应用程序（或是通过 Windows）发来的消息时如放弃控制，那么就可能在 Windows 中产生一个死锁。

一般情况下，一个调用 **SendMessage** 发送消息给另一个任务的任务在接收消息的窗口过程返回之前是不能继续运行的，如果接收消息的任务放弃了控制，而发送消息的任务又不能继续运行和处理消息（因为它正在等待函数 **SendMessage** 返回），那么就出现了消息死锁。

处理消息的应用程序不能直接放弃控制，以免导致这类问题。调用下列函数之一会导致应用程序失去控制：