Windows 程序员使用指南(一)

——DLL 和内存管理

〔美〕Mike Klein
赵人任　张燕　郑峰　　译
李春　审校

Windows Programmer's Guide to DLLs and Memory Management

清华大学出版社

Windows
程序员使用指南(一)
——DLL和内存管理

[美] Mike Klein 著
赵人佳 张 燕 郑 峰 译
李 春 审校

清华大学出版社

(京)新登字 158 号

Windows 程序员使用指南(一)——DLL 和内存管理
Windows Programmer's Guide to DLLs and Memory Management
Mike Klein
Authorized translation from the English language edition published by Sams.

图书在版编目(CIP)数据

Windows 程序员使用指南(一)—DLL 和内存管理/(美)克莱因(Klein, M.)著;赵人任等译 .—北京:清华大学出版社,1994
ISBN 7-302-01671-2

Ⅰ. W… Ⅱ. ①克…②赵 Ⅲ. 操作系统(软件)-基本知识 Ⅳ. TP316

中国版本图书馆 CIP 数据核字(94)第 13731 号

出版者:清华大学出版社(北京清华大学校内,邮编 100084)
印刷者:清华大学印刷厂
发行者:新华书店总店北京科技发行所
开　本:787×1092 1/16　印张:27.75　字数:656 千字
版　次:1995 年 3 月第 1 版　1995 年 3 月第 1 次印刷
书　号:ISBN 7-302-01671-2/TP·718
印　数:0001—5000

目 录

第二部分 附 录

引 言

是第三次魔术般的魅力？还是接连三次让你失望而去的打击？无论如何，Microsoft 凭借 Windows 3.1，终于能兑现它自 Windows 首次推出以来所作的承诺：提供一个实用的产品化的图形用户界面。PC 用户吵嚷了多年要求一个类似于 Macintosh 的界面，这个梦想终于得以实现。

Windows 最终并不仅仅是一个“巨大的”应用程序。实际上，用户运行 Windows 纯粹是为了 Windows 本身提供的种种好处，而不只是为了执行一个单一的应用程序，许多应用程序正慢慢地凝聚在这块“蛋糕”上。尽管 Windows 仍然建立在一个残缺的（但正不断改进的）平台——DOS 上，它正在迅速地改善其每一个版本。实际上，未来的 Windows 即 Windows NT 将不再需要 DOS。

Windows 的成功归功于下列几个特性：虚拟内存、保护模式、多任务、DDE、裁剪板、字体、全部模块化设计以及一个充满吸引力的可视界面。Windows 的成功还部分归功于增强版本的 DOS 所提供的支持。在 DOS 5.x 中有可能获得一个从未听说过的可用自由内存空间，大多数 80×86 系统会有多于 620K 的自由内存空间。由于 Microsoft 使用保护模式的 CPU 来作为扩展 DOS 未来版本的基础，它的健壮和成功的潜力是可以得到保证的。软硬件的向下兼容性必须在一定程度上为速度和功能作出牺牲，现在正是该这么做的时候！

Windows 3.1 大大地扩充了 Windows 3.0 中所提供的功能。3.1 版本中新增的特性及升级包括：

- TrueType 可调尺寸字体。现在在屏幕上见到的东西和在打印机上得到的是一样的。
- 改进的窗口速度和绘制消息处理。最令人注意的是程序管理器可以在屏幕上以比 3.0 版本快得多的速度“拍照”自己的窗口。
- 用于加速磁盘 I/O 的 FastDisk 驱动程序，允许这些 I/O 几乎完全在保护模式中执行。
- 一个增强的声音 API 和几个声音工具。更妙的是，它们几乎毫无缺陷地作用于除了 DOS 框之外的任何事情。
- 一系列附加的开发者工具、库和模板。也许最重要的是 ToolHelp.DLL，它提供了检查通常保留给 Windows 内部使用的数据列表的功能。
- 一个改进的 API，用来填补 3.0 SDK 的空白，改进了类型检查以及为对话框和控制增加了（并最终文档化）许多非常需要的消息。
- 一个改进的、速度更快的文件管理程序。
- 目标链接与嵌入（OLE）库，以及 DDEML.DLL、一个大大改善了的用于开发者的 DDE 接口的动态数据交换库。
- 健全的错误检查。当一个程序崩溃时，Windows 最终将很好地对它进行处理，允许

对该程序进行编辑、重新编译及调试——所有这些都不要求用户重新启动机器。

作为一个字符方式的 DOS 下的长期用户，我对 Microsoft 提供的新的 GUI 非常感兴趣。在 Windows 上工作越多，就越想知道并且越来越发现它的潜力。Windows 的积木块设计和几乎面向对象的结构给予了一种难以置信的无穷无尽的感觉。

我对于这个新环境的问题不在于需要掌握超过 800 个的 API 调用函数，也不在于这些事件驱动结构提出的新概念。我的问题是缺乏好的文档资料。Microsoft 提供的手册乍看似乎是一个很好的技术助手；但是，我在 Windows 中钻得越深，就越想爬出那个我觉察到自己所处的深洞。这些手册涉及的主题相当广泛齐全，但没有把足够的精力放在每一个主题上，特别是那些关键的问题上，如 DLL 设计、内存管理、用户定制控制和消息的产生与处理。这并不是说在这样一种不同的环境中工作不会有问题，因为问题确实存在。但是，我想读者将会发现使用事件和消息要比试图在 DOS 的过程驱动环境中生成一个相同的应用程序容易得多。

经过多年在 Windows 密林中劈荆斩棘(带着一把巨大而锋利的砍刀)之后，现在我终于感到足够地从容，并已经积累了足够的题材来写一本关于 Windows 中一些比较重要的主题的书，这些主题恰巧是 DLL 设计、内存管理、用户定制控制和消息的产生与处理。我保证在为未来的应用程序开发打基础时这些主题中的每一个都是同样关键的。

本书的对象

本书根本不打算作为 SDK 的替代品，也不是为那些希望得到有关 SDK 的入门指导的人写的。本书的读者对象是中高级 Windows 程序员。有时候，对于那些关心 Windows 和 C 语言的人来说，本书的某些概念也许会显得太基本，层次太低，其实不然。读者将会发现，了解这种低层的与系统相关的细节，将会大大地改善你的调试技能。有时候具有解决问题所需要的背景知识比获得对某个特定问题的简单而直接了当的答案更好。读者将发现，靠着对这样低层内部的深入了解，你，而不是别人，将能回答你的大部分问题。可以想象一下，如果读者理解了 Windows 的内存管理和消息系统是如何工作的，那么阅读并理解 DDE 就会成为一件再容易不过的事情，因为读者已经理解了它的代码的支持基础。读者必须很好地理解隐藏在模块设计、消息产生与处理以及内存管理后面的基本概念，以便迅速熟练地获得新的 Windows 技能。

系统配置

我用于本书的编程(和写作)环境由下列软件和硬件组成：Microsoft Windows 3.1 和 3.0(在保护模式下)、Microsoft C 6.0 和 SDK；Nu-Mega Technology 的 Soft-ICE/W 调试器；Solution System 的 Brief 编程编辑器；Gimpel Software 的 PC-lint；一个具有 64K 高速缓存的 80386/33 处理器；180M 的 IDE 硬盘；Headland Technology 的 Video 7 VRAM 视频适配器(运行在 640 × 480 × 256 模式)；还有一个 Zenith 平面直角 VGA 彩色显示器。本书中的所有例子都是为 Windows 3.1 编译的。由于显而易见的原因，本书只针对运行在保护模式下的 Windows。我的这个决定基于以下事实：其一是 Windows 3.1 不再支持实模式，此外，实模式并不代表大规模应用程序开发所需要的处理器环境类型。但是读者

将在本书中到处可以找到大量对实模式的描述和引用。

本书是如何组织的

本书第 1 章“Windows 基础”深入讨论了 Windows 的建立基础，包括模块布局和消息的产生与处理。第 2 章“处理器和内存基础”讨论了处理器的本质，包括与 CPU 有关的重要主题如地址空间、段、分页和虚拟内存。第 3 章“程序和模块基础”讨论了程序/模块的结构。第 4 章“内存管理”包括内存和资源管理。第 5 章“库设计概念”讨论与库设计有关的各个方面。第 6 章“对话框和用户定制控制设计”包括许多围绕着对话框和窗口设计的主题。附录中包含了本书讨论的几个用户定制控制的源代码。附录中还讨论了对于每个控制和库可以做的改进。在源代码中给出的所有控制和函数都是可以立即实现的。为了证实这点，我实现了一个使用本书所列控制的示范应用程序。充分使用 DLL 是复用的最好形式。做出决定把你的 SpecialLittleSomething()函数包装到一个单一的、可重复使用(并且总是被测试过)的 DLL 文件中吧。你用你节省下来的时间所做的一切是令人惊讶的，因为你已经不再重复错误。

第一部分

Windows 开发的基本概念

第 1 章

Windows 基础

有人说过编写好的 Windows 应用程序要求不同的思想方法——至少对于那些久经考验的优秀 DOS 程序员来说是这样的。这个说法纯属错误。一个优秀的 DOS 开发者，付出同样的努力就可以成为一个优秀的 Windows 程序员。这里所需要的技能完全相同，只不过要重新应用这些技能而已。当然也并非没有任何区别。不仅仅是你使用的环境完全不一样了，而且当你在开发过程中被难住的时候，几乎没有多少工具可以帮助你。

本章讨论：

- 应用程序和库模块
- 基于库的 Windows 结构
- 消息
- 面向对象的概念

Windows 环境为应用程序提供了许多服务程序，每个服务程序都对编程周期具有影响(好的和坏的)。在 Windows 环境下进行开发时初始的最大的差别和障碍是：

- 事件驱动结构/消息
- 不好的文档资料
- Windows 本身

在 DOS 中，程序员实际上可完全控制程序的执行。在 Windows 下则有所不同，因为用于程序流的是事件驱动模型而不是语法或过程驱动模型。关于你的环境的许多事情都不像在 DOS 中那样可以设定。如果你试图认为某事是当然的，那么你的应用程序就很可能被终止！事件驱动的结构会成为最大的障碍。开发人员不能再通过 CodeView 来进行一步步调试和跟踪，也不再保证他或她能观察当前的执行线程。从模块 A 调用一个 Windows 函数(很可能不为你所知)会使在另一个模块中的代码开始执行。为了在 Windows 下有效地进行调试必须获得一定级别的“外部显示(peripheral vision)”。

Windows 的消息系统，例如何时及如何发送一定的消息，是开发人员遇到的另一容易混淆的地方。这个问题的出现，Windows 本身是最大的“罪犯”。正如在 DOS 中一样，Windows 使用了一些未公开的函数和消息，而这些消息和函数限制开发者使用。Windows 还经常违反自己的消息规则，特别是在多文本接口(MDI)的设置方面和用户定制控制方面。

在 Windows 下进行成功的开发要求程序员完全彻底地理解操作系统，因为应用程序的个人特有的风格将不再像在 DOS 中那样可以容忍。必须从底层开始清清楚楚地编写源代码。这样的要求需要对 Windows 的透彻的理解。本章致力于使读者更好地领会有关 Windows 设计和消息的基本概念。本章最后一节“Windows 下的面向对象概念”简要地讨论了一些面向对象的原理及其与 Windows 操作系统的联系。

1.1 模块:应用程序和库

Windows 环境具有两个基本的程序单元:应用程序(.exe 可执行文件)和动态连接库(.dll, .drv, .exe 和 .fon)。有时候也把它们称为可装载的或可执行的模块(参见图 1.1)。从这以后,"程序(program)"一词专指一个应用程序或库,"库(library)"一词与 DLL 则可互换使用。

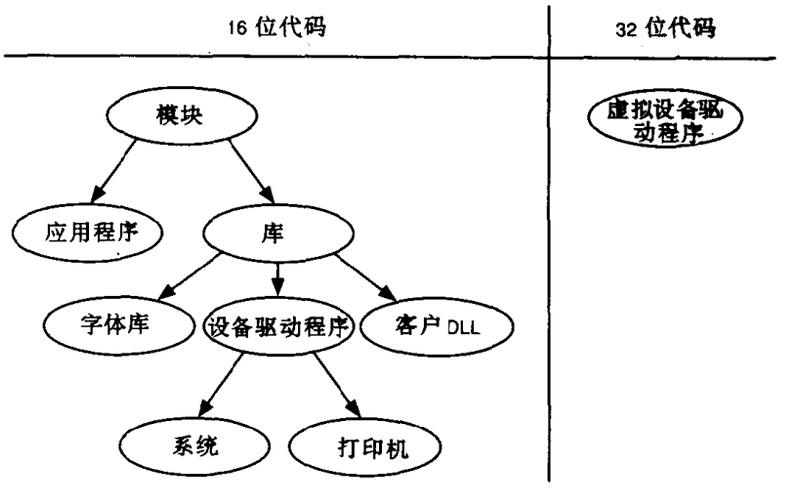

图 1.1 典型 Windows 应用程序中的基本程序单元

应用程序区别于动态交换库 DLL 的地方在于可以运行一个应用程序的备份拷贝或实例。应用程序的每个拷贝都由一个实例句柄来引用。DLL 被装载进内存一次,只有一个实例句柄。实例句柄实际上被 Windows 用作指向程序缺省数据段的句柄。原因很简单,因为 Windows 共享代码和资源段,数据段是唯一可以用来区分一个模块中各个实例的段。

在 Windows 3.1 版本中,所有应用程序和库都在第 3 个环中运行,这是对非系统代码的规范(参见图 1.2)。这与 Windows 3.0 版本形成对照,在 3.0 版本中,所有用户程序都

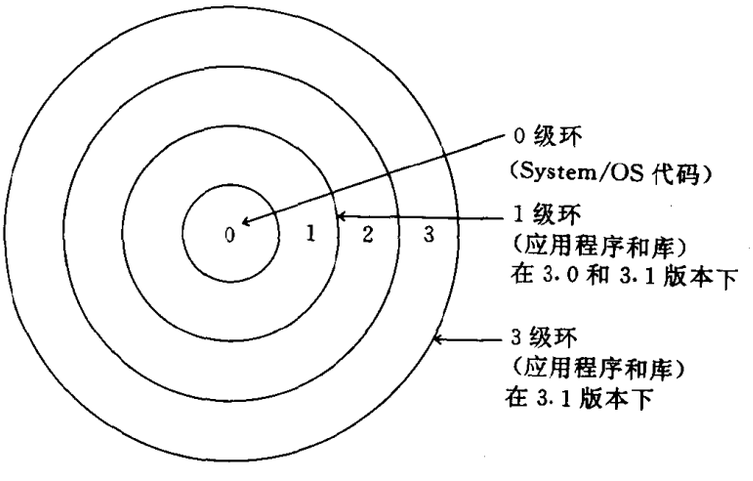

图 1.2 保护级环图

在第 1 环中运行。在这两个版本中，第 0 环全部为 Windows 的虚拟内存管理程序和虚拟设备驱动器留用。在第 0 环中运行的程序可以看到所有的线性内存并且可以毫不受阻地访问所有系统资源。

模块/实例对与类/窗口对非常相似，因为可以有一个模块（只能是应用程序）的多个实例或者有一个特定类的多个窗口。在这种情况下，类和模块这两个词都指的是信息集合（数据库），这个集合要用来生成特定类型的活跃的对象。

1.1.1 实例/模块 = = 窗口/类

当应用程序或库第一次被装载到内存中时，Windows 生成一个被称为模块数据库的数据结构。模块数据库像所有其它程序段（代码、数据和资源）一样被存储在全局堆中。模块数据库中包含的信息可以在应用程序或 DLL 的新风格的 .exe 文件首部中找到。这个相当公开化的首部包含了大量的信息，这些信息涉及该程序的所有调出函数、资源及其它唯一的、不可更改的信息。在 Windows 中，通过模块句柄来保持与模块数据库的联系。Windows 还与指定的应用程序的每一个实例一起与一个任务相关联，这个任务通过一个任务句柄来引用。该任务句柄指向应用程序的任务数据库（TDB）。TDB 中包含诸如 DOS 文件句柄表、当前 DOS 路径和指向一个应用程序消息的指针这样一些实例唯一的信息。它的功能与 DOS 的程序段前缀（PSP）相似。实际上 TDB 确实包含一个 PSP。

DLL 没有 TDB，这也就是为什么要用当前激活的任务（通常必须是一个应用程序）来连接所有由库创建的窗口的原因之一。这也是为什么那么多的 API 调用都集中于应用程序的原因。一个任务是一种线程，既可以通过应用程序来运行，也可以通过依附在该应用程序的任何库来运行。虽然在 Windows 未来的版本，特别是 NT 中，将允许多线程的应用模块，但库将仍然由一个调用进程（应用程序）来驱动并且不能具有自己的“生命”。

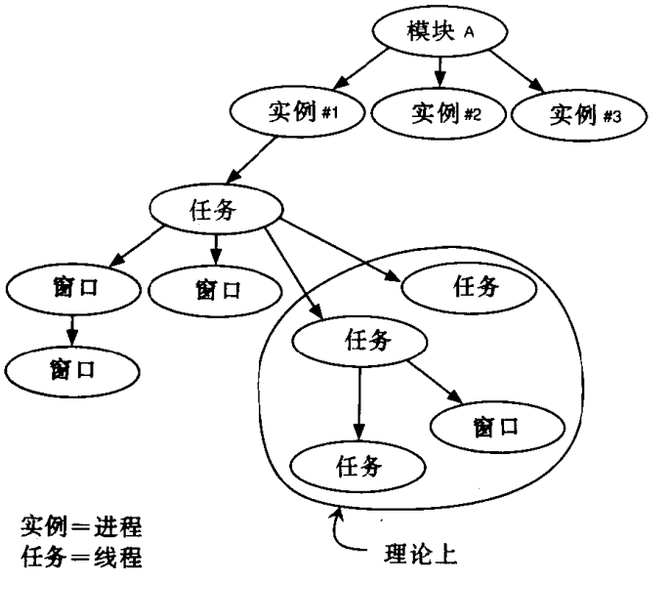

图 1.3 任务、实例、模块和窗口句柄之间的联系

然而库将能够以应用程序的名义来产生一些单独的任务。在 Windows 3.x 版本中，“进

程"和"任务"这两个词是可以互换的。

任务、实例、模块和窗口句柄可以放到一起来形成一个简单的树。把树中扇形张开的低层向上移动,可以确定隶属关系及其它依赖关系(参见图 1.3)。利用任何类型的句柄,你在编程方面只能做这么多,所以通过关联性可以使开发人员获得树中的其它句柄,提供对所需信息的访问。例如,一个应用程序正在扫描桌面窗口并想知道创建这些窗口的应用程序,它将不能仅仅利用一个窗口句柄来找到结果。窗口句柄一定与一个实例句柄相联系,而实例句柄可以用来确定阶梯中的上一层——模块的全部路径名。

1.1.2 应用程序

应用程序被称为任务化的可执行模块。这是因为在 Windows 中每一个应用程序都由一个任务驱动并响应一系列的外部事件(消息)。目前每个应用程序实例只有一个任务,但 Windows 最终将会变成多线索的系统,允许应用程序中的初始化任务或进程(也许只有这个任务或进程)生成多条线索。

从某种意义上说,Windows 应用程序可被认为是由代码起动然后被消息驱动的。所有的应用程序都通过具有一些职能的 WinMain()函数开始获得控制。由于可以运行你的应用程序的多个实例,在 WinMain()中要做的第一件事情就是调用任何模块初始化代码。这是作用于你的应用程序的所有实例上的严格的一次使用的代码。这些代码基本上是用来注册应用程序窗口类以及初始化全局数据。WinMain()的第二项工作是调用实例指定的创建代码,通常这意味着创建窗口。Windows 只通过 4 个参数来调用 WinMain():应用程序实例的句柄,应用程序另一个实例的句柄(如果存在的话),一个指向应用程序启动时的命令行的长指针,以及一个整数值,该值指定应用程序第一次执行窗口显示的方式(即显示的窗口是最小化的还是标准的或是最大的)。

一旦你的程序创建了至少一个窗口,控制通常就被传递到 WinMain()中,这样,应用程序就可以进入其中央消息处理与分发循环之中。如果,由于某些原因(由于错误或其它原因)最终没有一个窗口被创建而仍然进入了 WinMain()循环,那么,应用程序可能永远不会让出控制。消息循环将永远不会中断或把控制返回到应用程序中,因为桌面上不存在应用程序创建的窗口来把消息送入循环中。最为关键的是,循环将永远不会处理 WM_QUIT 消息,这个消息是用来结束消息循环并最终结束应用程序的。对这个规则的唯一例外是在使用 PostAppMessage 发送消息到应用程序中的时候,因为是任务句柄而不是窗口句柄作为消息的接收对象。这个技术将在本章后面讨论。

WinMain()消息循环是每一个应用程序的心脏,因为它的任务是把消息输送到前面几行代码前所创建的所有窗口之中。应用程序可以不受限制地创建、维护和销毁任意多的窗口。但是对整个系统来说似乎存在一个大约 500—700 个窗口左右的限制(读者的情况可能会不同)。一旦某个窗口调用 PostQuitMessage(),Windows 就传递一个 WM_QUIT 消息到应用程序中,然后控制将中断消息循环。此时,清除代码可能被调用,应用程序终止。图 1.4 说明了这个消息流并用图形表示了一个基本 Windows 应用程序的分段概观。