HOPE

Novell Netware程序员指南

用C编写Netware应用程序

袁 荣 编译

北京希望电脑公司

Novell Netware 程序员指南

——用 C 编写 NetWare 应用程序

袁　荣　编译

北京希望电脑公司

目　录

前　言

建议

在探究使用 NetWare 编程的复杂程度之前，我希望告诉你们一些我的设想，偏见和预想，这将使你建立起能从本书中得到什么的概念,同时帮助你确定这本书是否适合你学习。

对等通讯

首先，我假定你我作为点而相互谈话，我正巧知道不少 NetWare 的知识，我同样假设你对用 C 语言编程非常熟练，但不知你是否知道其它的编程的语言，我在一些地方写了一些带有注解的涉及 80x86 汇编语言的程序，但它们若引不起你的兴趣，你完全可以忽略它们。

可移植但并非需要发布

我假定你目前所开发的软件特别集中于 NetWare 的环境，你也可能同时为其它的环境开发软件，但你必须肯定利用了 NetWare 版本所提供的极强的特性，因此你也愿意牺牲可移植性。无论你正在开发专用文件服务器，LAN 维护应用程序或过去单用户软件的多用户版本，本书将帮助你编写对 NetWare 环境进行全面开发的程序。

程序范例应该完整

假定你买了一本书，将书上的程序输入计算机，而后你发现这些程序无法运行，因为程序的大部分仅适应于某些环境，你还必须付出额外的费用方可运行。此时，你会像我一样恼怒，因此，我的所有程序范例是完整的，也就是，你如果将这些程序全部输入计算机，待完成后，你将得到一个完整的，可执行的应用软件。若你不愿意输入，可花点钱，从 Addiso--Wesley 处购买一张软盘(假定你有 Novel 的用于 Dos 的 NetWare 接口)。

编程协定及假设

本书有许多 C 语言程序，故我愿意解释我的编程风格的一些要素，并给出使用这些要素的一些理由。

匈牙利标记法

我使用的一个协定就是变量命名协定，即匈牙利标记法，这些标记已成为 Ms--Windows 程序员的标准，简单地说，使用匈牙利标记变量名则隐含着变量类型，使你一看就知道这个是什么含意，头一个或头几个字母，采用小写形式，告诉你目标的数据类型，紧接着是一个大写字母，变量名的其余部分要求足以明确描述它的含义，大、小写字母混合的使用方法使变量名更易读。

我所使用的数据类型的缩略形式如下：

```
c     char
by    BYTE (unsigned char)
n     int
b     BOOL (int)
s     short
w     WORD (unsigned short)
l     long
dw    DWORD (unsigned long)
fn    function
sz    null-terminated string
ps    Pascal string
```

指针加上 P、NP 或 LP 在变量名前面，这样

```
p     pointer (e.g. psz = pointer to null-terminated string)
np    near pointer (void near *)
lp    long pointer (void far *)
```

结构的命名有时却是无规可循的，通常，我尽力提出一个包含有某种结构类型信息的名字，例如，将在本书后面出现的程序中，我定义了数据类型 APPLICATION-OBJECT，所创造的这种类型的一些变量名是：aoNewAPP、aoTempAPP、aoOLDAPP 和 aoRunAPP。

简短、含意模糊的变量名，在实际情况中并不多用，但我还是倾向于命名循环控制变量为 I,J,K,L,M,等等。

简单接口

在我给出的程序中，我仅仅满足于一个基本的电传接口，原因如下：为了提供一个完全菜单驱动，基于窗口的接口，要么我必须编写出许多额外的程序，要么我必须产生一个专用的库产品，迫使你购买，然后进行。

C 编译

从 1987 年起，我就一直在使用 Microsoft C，并不觉得有修改编译程序的需要，我所提供的 C 程序对你所喜欢的编译软件足可移植的，但是我的 MAKE 文件包含了许多适用一 Microsoft C 5.1 版的命令，你需对这些命令进行修改，以适应你自己的编译系统。

面向目标的编程

我并不认为在面向目标的编程方面我是专家，我曾在 MS-Windows 下做过许多工作，同时也偿试过 C++，故至少我熟悉面向目标设计的原理，NetWare 环境充满了目标：装订目标、目录、卷名、文件和其它许多东西，因此，在 NetWare 下的编程自然引导到一条面向目标的方式上去,即使你在面向目标偏移方面还十分陌生。

对 C 接口库的一个高级接口

在本书的许多章中，我开发了 C 数据类型和函数，这些对 NetWare 系统调用而言，形成了一个高级接口，就由 NetWare c 接口——DOC 所提供的调用相比较，在 NetWare c 接口库中的许多函数是相当低级的，仅高于它们所使用的汇编语言一个层次，许多函数相

当不适用，需要一个输入自变量的主程序，返回许多可能出错的编码，但这并不意味着是对 NetWare C 接口库的一个批评。NetWare 环境是相当丰富和复杂的，NetWare C 接口起着一个重要的作用，它站在高于基于中断的系统调用的层次上，与它们本身相比，更容易调用，我在本书中改进的函数属于高于 NetWare C 接口的层次，无论在功能上还是数据抽象上，均提高到了较高的水平。

从 NetWare 目标到 C 数据类型

我时常碰到的一个从 NetWare 目标到 C 数据类型的过程，就是首先将一个 NetWare 目标表示成关系数据库表，或表的集合，这些表经过筛选(通常用关系数据库的术语)，以更有效的方式存贮数据，这就给了我们一张从逻辑观点看意味着什么的映象图，而不像它们在 NETwARE 环境中所具有形式，同时，也使我们更加清楚地明白目标间的关系，然后，我将关系表结构转化为 c 数据类型，通常是某些种结构，最后，我改进了在这些结构上操作的功能函数，通过使用这些结构和函数，我即不必知道 NetWare C 接口如何实际表示目标，也不必知道 NetWare c 接口如何处理目标。在其它的一些地方，我省略了关系表的产生过程，但是，关系设计总是指导着我，尤其是在建立无冗余数据结构方面。

结束语

在本书中，我提供了许多程序范例，以演示 NetWare c 接口功能是如何工作的及如何使用这些功能，我也建议你们研究随产品的库的源程序，此库是本书的最终参考源。同样，在库中的函数均可互相调用，由此，产品本身就是一个如何使用产品的最好的例子。

一次又一次，我考虑 NetWare C 接口库的缺陷，建议修改源程序。我做这些并非出于我本身的利益，也并不想批评 Novell 而使其难堪，作为一开发者，你需要知道在使用库时注意什么，如何解决问题，另外，我肯定你会发现我程序中的缺陷，我已严格地进行了测试，但你的看法比我的清晰，因为你不必经历最初的设计时期，若你能告诉我你所发现的缺陷和错误，我将不胜感激。

我讨论的 NetWare C 接口在 1.10 版本中的缺陷，我已告诉了 Novell,在本书出版的时期，他们已经修改了缺陷，我写本书的目的的一部分就是让大家分享我使用 NetWare APIs 的经验，经验的一部分发现了什么方式工作是正确的，什么是错误的，当有些东西工作方式不正确时，由于提供了源程序，我们掌握了修改源程序的方法，最终使其工作方式正确，尽管我所讨论的特别的缺陷的改正需在几个月之后，但不需要有研究、修正和定制源程序的过程。

第一章　APIs 概述

简介

本书涉及 Novell 的 NetWare，NetWare 是一个丰富的、功能极强的、复杂的操作系统。有许多书是涉及该操作系统的，它比今天市场上所有的网络操作系统更具有优势，有许多理由可以证明它是杰出的操作系统，它本身在许多方面展现了杰出性，但主要有下面三点：

. 极专业和优化的文件服务器管理系统

. Novell 所追求的灵活的、无端点的、PC 为中心系统结构

. NetWare 提供给开发者的丰富的编程接口

术语 API 是应用程序接口的缩写，在近几年，已得到了广泛的流传。许多人使用该词来暗示许多不同的事情。由于本书是有关 API 的书，特指由 NetWare 提供的 API，故我希望一开始即说明我是如何弄懂该术语的意义的。

什么是应用程序接口

一个应用程序接口当然是提供给应用程序的一个接口，但问题是：该接口提供什么？谁提供该接口，有何目的？

API 提供事先定义好的、连续一致的路径，通过这些路径，应用程序可以得到系统服务。使用 API 的程序，可以在较高级的、更抽象的水平上操作，要求由处理具体的专门问题的程序提供服务，通过 API 提供的服务是本质的、各种各样的，这些是许多具有多种功能的应用程序所需要的，例如，几乎没有什么应用程序是不需要文件服务的，不管上一个字处理软件、电子报表软件、语言翻译软件、CAD--CAM 程序或是一个数据库管理系统。

这不意味着 API 不能是特定的，例如，由 Novell 的 Btrieve 所提供的 API 完成特定的功能，然而，这些功能对各种高级程序却是有用的。

由于通过 API 所提供的服务是通用的，它们比由应用程序提供的服务更有生命力，它们可以向驻留内存的程序一样安装，当应用程序调用完成后，它们仍在内存中，也可驻留在动态连接库中(DLLs)，它们是公用的，随时可以调用。对所有使用机器的应用程序都应知道这些服务，从本质上说，这些服务是公共的。由于这些原因，我们可以说 API 是工具，通过这些工具，局部的、当前的进程从公共的、永久的服务中得到服务。

API 也可以是分层的，一个水平上的 API 可以调用在较低水平上的 API，例如，MS--DOS INT 21H 程序频繁地使用 ROM BIOS 的中断，在一个网络环境中，在网络一层的 API，例如，Novell 的 Netbios 仿真器工作在 OSI 会话上，就可以使用工作在网络层上的 IPX(将短短地讨论 OS2 结构)。

API 做什么？

API 提供一种相应的、文件式的方式以访问系统程序，同时，提供易懂的系统服务管理和一个所提供的服务的逻辑层。

相应的系统服务管理

API 减轻了应用程序对低层系统管理的压力，使其能够集中低层系统管理的特性，它们将保证对许多应用程序所需的服务以统一的结构、良好的方式进行管理。

这种统一管理的益处的一个有说服力的例子就是文件系统，这是由操作系统所提供的最重要的服务之一，同样，它也是最复杂的。如果一个应用程序要书写自己的文件系统，那其它的事就不能多做，同理，若每一个应用程序都有自己如何组织软盘的主意，那许多程序使用同一张软盘就是不可能的，每个程序就会有自己保留的存贮区域。

由于操作系统提供了一个标准的文件访问方式，任何程序均可将文件存放在软盘上，而且可以保证在下一次程序需要所存的文件时，它仍在软盘上，另外，程序本身并不需要考虑地址分配表、目录、节点、软盘扇区及其它包含在文件系统中的大量细节。

分层服务

分层是软件设计中最重要的准则之一，它提供了最低层到抽象的最高层的软件的分层结构。最低层的软件通常依赖于硬件而不可移植，越往上走，程序就越可移植。

分层设计的最重要的原则之一是所给层上的函数可以使用下一层的函数，并不受那些函数的实现方式的影响，这样，若第三层的函数变化，但提供给第四层的函数的没有变化，则第四层上的软件不需要写。

在独立环境中的分层

在独立环境中，API 将应用程序与特定的机器如何实现特定的功能分隔开，这样允许应用程序可以无选择地在各种类型的机器上运行，你进入的 API 链路越高，你的程序就越具有可移植性，在 PC 机上，链路的处理如下：

写入 I/O 端口：直接写入 I/O 端口的程序要求机器使用那些端口。

使用 ROM-BIOS 调用：使用 ROM-BIOS 调用程序可以在任何支持 BIOS 的机器上运行，例如，使用 IBM PC-AT 机上的 ROM-BIOS 写的程序可以在 COMPAQ 机上运行，但不能在 HP-150 机上运行。

使用操作系统的调用：使用操作系统调用而写的程序可以在任何具有该操作系统的机器上运行，用 MS-DOS 函数调用编写的程序，即可以在 IBM-PC/AT 上运行，也可以在 HP-150 机上运行；使用 OS/2 函数的程序，可以在 PS/2 80 型机上运行或 Compaq 386 机上运行；使用 NetWare 调用编写的程序可以在以 Compaq 386 为工作站的 Ethernet 局域网上运行，也可以在 PS/2 为工作站的 Token--Ring 局域网上运行。

使用高级语言库

采用高级语言且仅用操作系统的标准运行时库程序编写的程序，可以在任何具有该源语言的编译程序的机器上运行，例如，仅使用标准的 UNIX 函数调用的 C 程序可以在 C 编译程序支持 UNIX 标准库的任何机器上运行。

网络中的分层

在网络环境中，基于网络分层的概念，API 也是分层的，无论是否涉及网络结构，较

低层提供基本的服务，再一次说明，较高层的 API 可以使用较低层的 API。

由于这是一本有关网络编程的书，花费一点时间讨论国际标准组织(ISO)的开放系统互联(OSI)结构是有必要的，该结构除了是广泛公共的结构外，还是 NetWare 设计所基于的网络结构。

ISO-OSI 的较完整的讨论在 Xerox 的出版物《Xreox 网络系统资料大全》，Xerox 说明了 OSI 模型及任何目标源的设计概念与分工，故在此引用一些。

Xerox 网络系统概念及设备

大多数网络结构(包括 XNS)的基本思想是分层结构，这意味着由结构所支持的各种功能可分为一系列的层次，依据贯例，最基本的工作位于较低的层次，而较高层通常保留用于较复杂的工作。

ISO 开放系统互联标准模型

这种分层结构有用的一个模型是由国际标准组织(ISO)所开发的，该模型称为“开放系统互联(OSI)标准模型”，尽管作为标准，还缺乏足够的说明，但 ISO 模型提供了讨论复杂目标的基本思想。

ISO 模型具有各种网络的结构，不论如何复杂或非标准，均可划分为七层。从其功能的相对简单和复杂性上说，每一层均高于下一层，而低于上一层。这种分层的意义是 ISO 模型的一个基本的特点。一个共同可接受的透明性就是某一层的功能可以使用下一层的资源以完成指定的工作，由于这个模型的分层特性，较低一层主要处理数据通信事宜，较高一层的处理信息的控制、管理和应用的广域问题。

ISO 模型的分层结构及在网络结构中的作用描述如下：

第一层：物理层　所有的网络提供数据传输，在本层中，基本的工作与数据传输发生有关，如位流控制，拨号(用于开关网络)，调制解调器控制等等。

第二层：数据链层　本层负责通过可靠的数据链获取信息，信息的组织一为传送，二为处理(输入)；检测传输错误(有时能修正)，数据传输率可通过链路控制。

第三层：　网络层　本层负责获取通过网络的信息单元，它提供数据组织功能，用于将信息送出或送入传输介质，它同样包括高级的出错恢复结构及多数网络寻址、路径和开关的决定的控制。

第四层：　传输层　在本层，决定的形成基于适当的传输服务，提供多种数据的组织和重组织功能，进行一些网络的管理工作。

第五层：　会话层　本层负责产生两方面的通信关系，使双方在必要的时间里完成双方的交易(称为对话)，相应的任务如缓冲，排队等也同样在这层进行。

第六层：　表达层　数据转换和类似的工作在这层实现，其目标是将由用户应用程序识别的数据形式转换成在较低层所用的形式(或远程用户处理)，反之亦然，这包括从编码及格式转换这种简单工作到语法转换(如在两种不同编程语言的移位操作这种极端情况)的复杂事宜。

第七层　应用层　特定的应用在此实现，例如：在一个面向文件的环境中，本层

可能包括：文件的形式、打印、邮送等。

作为一个说明 NetWare 如何遵循 OSI 模型规则的例子，它利用 XNS 协议。网间数据协议(IPP)和顺序报文包协议(SPP)于它的网络和传输层服务(IPX 和 SPX)，然而，XEROX 文件仅定义了报文包格式和使用方法，由 NetWare 提供、使用，作为事件控制块(ECB)数据结构的实际的 API 是 Novell 自己实现的，因此使用这种 API 的程序只能在 NetWare 局域上送行。

在 OSI 模型的第二层即会话层上，NetBios 是一个全面支持在 LAN.APPC(LU6.2)上的对等通信的标准，同样，它提供会话层服务，它的范围是广泛区域的对等通信，另外，APPC 通常提供属于表达层的服务，如 ASCII 和 EBCDIC 数据之间的转换。NetWare 和 APPC 两种均可移植到非 NetWare 环境中。

在网络的最高层即应用层上，仅有一个标准的 API：MS-DOS 3.1 函数调用以及由 MS-DOS 3.3 附加函数的调用。包括 MS-DOS 3.3 在内，MS-DOS 提供 10 种函数调用(见表 1-1 所列的 MS-DOS 函数)。相反，超过 200 种的函数调用通过 NetWare 扩展到 INT 21H 接口。Novell 提供了一整套用于文件和目录管理、锁定、网络通信和打印服务的函数，只有少数可命名，但是 Novell 的 API 不是一个标准，它是一个专利。

与由 NetWare 提供的功能极强的环境相比，MS--DOS 的 API 就显得那样弱小，迫使开发者做出某些困难的决定，要么牺牲功能，要么牺牲可移植性，若要求 MS-DOS 的可移植性，就要放弃大量的功能，若要求 NetWare 的功能，就要放弃 MS--DOS 的可移植性。

表1-1: MS--DOS网络函数

主要函数号	子函数号	函数描述
5CH	0	锁定字节范围
5CH	1	开锁字节范围
5EH	0	获取机器名字
5EH	2	设置打印机准备串
5EH	3	获取打印机准备串
5FH	2	获取改向列表入口
5FH	3	改向设备
5FH	4	取消改向
67H	无	设置句柄计数(仅用于MS-DOS)
68H	无	认可(仅用于MS-DOS)

正如前言中所述，本书的一个假设即是你已经不用 MS--DOS，而全身心地使用所提供的 NetWare 的特性，当然，这需要付出可移植性这个代价。

API 的结构

有两种基本的访问 API 的方式：通过中断处理程序和函数调用，有两种参数传递的公用方式：机器寄存器方式和堆栈方式。

通过中断访问

在 MS-DOS 环境中，API 是由一个装载程序安装的，该程序设置一个中断向量，指向处理中断的程序，然后结束并驻留内存。为使 API 有效，由 API 使用的中断号以及如何传递参数和返回参数，应是公开的。例如，MS--DOS 和许多 NetWare 函数是通过 INT 21H 访问，NetBios 是通过 INT 5CH。而 APPC/PC 和 NetWare LU 6.2 两种是通过 INT 68H。由于它们通过 INT 指令访问，作为中断处理程序实现的 API 必须从汇编语言调用。

安装每一个 API 作为中断处理程序需永久性地占用一定的低区内存，这样，就使应用程序的可用空间减少，例如，MS--DOS(驻留的 COMMAND.COM)部分需 4K 内存，NetWare 工作站的外壳约占 55K，NetBios 约 20K，APPC/PC 占用大于 300K，明显地，如果使用一个 API，不可能无节制地占用内存，或者，它对应用程序提出若干约束条件，例如，APPC/PC 的内存要求总认为是超大的。

通过函数调用访问

其它的操作系统和环境为访问 API，使用系统库的函数调用，这里包括 OS/2、NetWare 的 OS/2 Requestor、Macintosh、MS-Windows、VAX/VMS 和 UNIX。在这种情况下，系统程序不必永久常驻内存，也不必将其连接成可执行的映像，取而代之，在运行时需要访问它们时，它们驻留在中间位置。OS/2 和 MS-WWindows 的系统程序存储在动态连接库中。UNIX 系统程序是操作系统核心的一部分。

采用寄存器传递参数

按贯例，大多中断驱动的 API 需要寄存器传递参数和存放返回的参数，例如，为进行 MS--DOS 打开文件调用，参数存放在 AX、DX 和 DS 寄存器中，产生 INT 21H 调用，这里是一个打开文件 MYFILE.TXT，以便读写访问的指令序列，该指令序列允许文件公用(拒绝非公用方式)：

```
FILENAME DB 'MYFILE.TXT', 0
.
.                          .
.
MOV  AH, 3DH          ; Open File
MOV  AL, 2            ; Read-write access
OR   AL, 4 SHL 4      ; Deny none
MOV  DX, OFFSET FILENAME
INT  21H
```

该功能的返回值放入 AX 中，若进位标志未置，则这是 DOS 指定文件的文件句柄，否则这是一个错误代码，指示为什么文件不能打开的原因。

越复杂的 API 需传递越多的信息，返回越多的信息给调用程序。对许多此类的 API，8086 系统处理器无法提供更多的寄存器以存放所有的参数，类似于 NetBios

APPC 和许多 NetWare 扩展函数调用一样，API 传送指针到参数区。

NetBios 参数区被认为是网络控制块(NCB)，一个 NetBios 调用是通过将 NCB 的地址放入 ES:BX 和执行 INT 5CH 来实现的，APPC 调用它的参数区动词记录(Verb record)，调用程序将指示动词记录格式的编码放入 AH 中，将记录的地址放入 DS:SI 中，NetWare 扩充到 INR 21H 需要两个指针：一个在 DS:SI(请求包)中，一个在 ES:DI(回答包)中。NetBios

和 APPC 从参数块的一些区域得到参数，返回输出信息到其它的区域，NetWare 调用从请求包中获得获入信息，在回答包中放置返回信息。

使用堆栈传递参数

以函数调用为基础的 API，通常采用堆栈传递参数，在这种方式下，用 PUSH 指令将参数放入系统堆栈中，然后，相应的程序通过 CALL 指令调用，然而，在系统程序中，通过使用 BP 寄存器，参数通常是由高级语言的人口程序进行恢复。

```
PUSH BP
MOV  BP, SP
; At this point, in a large-model program,
; BP + 6 points to parameter 1
MOV  AX, [BP+6]
```

传递参数即可用值，也可用相对值，如果一个参数仅用于输人，则有必要将其值压人栈中，如果一个参数用于输出或包括许多信息，以致于 16 位的堆栈区不够用(80386 为 32 位)，则可传递其地址。

比较

函数调用比中断更具有优越性，最重要的就是用堆栈方式传递参数，由于这个过程完全兼容高级语言调用约定，采用函数调用的 API 可以直接由这些高级语言访问，无汇编语言调用之必要。

从高级语言直接调用 API 的能力意味着如下一些优越性：

. 助记函数名：当从汇编语言调用时，可用描述名调用函数，这比十六进码充分显示了描述名的目的特性。

. 简单的调用语法：高级语言的调用语法比汇编语言的调用语法简单得多。

. 可移植性：用 C、Pascal、FORTRAN、ADA、COBOL 或其它任何高级语言编写的程序，可以用于任何环境，在此环境中，支持所用的 API。

使用函数调用的 API 不需要中断向量，由于越来越多的 API 在 MS--DOS 环境中有效，矢量变成热门，例如，在 NetWare 2.0 版中，通过 INT 7AH 访问 IPX。当 Novell 得到 CXI (专业生产微型机到大型机通信产品的公司)时就改变了这个接口，因为 Novell 发现所提供的两个 API 均使用了 INT 7AH，而 IBM 的低层应用程序的 CXI 的实现及 LLAPI 也同样使用了 INT 7AH。

NetWare API

在 MS-DOS 环境中，NetWare API 是作为中断处理程序而安装的，API 的大部分是 MS-DOS INT 21H 接口的扩展，这是 Novell 部件上的一个智能设计。我们已习惯了 MS-DOS 提供的 INT 21H 的调用，使 Novell 的操作系统调用与 INT 21H 的调用有联系，只强调了 Novell 扩展和增强了 MS-DOS，但绝不取代 MS-DOS 这样一个事实，它还允许在单用户 MS-DOS 环境下开发的软件不加修改地在 NetWare 下运行。

NetWare 的 API 功能在核心中保留了汇编语言接口，但是，Novell 明确地承认：汇编语言正低落，而 C 正逐渐占优势，该公司的 NetWare C 接口(这为本书的焦点)是 INT 21H

再加一个人为接口，由于它提供了99%的汇编语言接口函数，也就没有必要要求用汇编语言写 NetWare 程序，在我们给的为 C 前端函数所写程序中，在极少情况下，汇编语言还是需要的。

第二章　多方面服务

概述

在一章中用“Miscellaneous”这个词作为标题占据第二章这样一个醒目的位置，我个人觉得不妥。多方面服务是一组不太吸引人的函数，这些函数有讨论的必要。为何如此迅速地讨论？其原因是：它们对其他的函数群而言，是简单的、方便的，它们有可能在本书的许多地方出现。我们将使用多方面服务函数中的一个在这些地方。现在讨论它们，我能在以后涉及这些函数时，放宽你们知道我在谈论什么的假设。

我在此讨论不想涉及太深，这些函数的多方面特性避免了那些必然性，就大部分而言，这些函数是容易使用和理解的。

顺便说一下，在文本中，对这些函数的若干存在一个错误，在程序范例中，写成：

```
#include <nat,h>
```

应该是：

```
#include <nit.h>
```

有一个 NAT.H，但它用于 AFP 服务功能，将在第十一章讨论。

函数分组

多方面函数进行四个基本的工作

. 数值数据(短或长的整数)的转换。主要用于 80x86 机器格式(在低地址上的低字节)和 NETWare 格式(低地址上的高字节，通常称为 High--low)之间的转换。[IntSwap (),LongSwap()]。

. 字符串转换。主要用于空结尾字符串(在 C 语言编程中经常使用)和在前面加上一定的长字节的字符数组(多用于 pascal)之间的转换。[ASCIITOLenStr(), LenToASCIIStr ()]。

. Pascal 型字符串的处理。[LenStrCat(), LenStrCMP(), LenStrCpy()].

. 表示网络文件名的字符串的语法分析(CovertNameToFullPath(), CovertNameToVolumePath(), GetFullpath(), Parsepath(), StripFileServerFrompath ()]。

表 2-1 列出了多方面服务函数目录

数值数据的转换——Intswap()和 Longswap()

Intswap()和 Longswap()转换了整数和长整数数据的字节顺序，它们将转换的变元看成是输入变量，返回开关顺序号，这些功能的标准样式是：

```
extern int intswap (int ~int variable);
extern long longswap (long ~longvariable);
```

有一个小错误需说明：intswap()标准样式即规定了它的输入量又规定了它的返回值是

整数型的，从技术上说，这是不正确的。整数数据类型在C语言中是一个机器字，这样，如果存在这样的编绎程序，该程序可以识别在386机上一个整数是长整数；intswap()将产生错误。例如，假如有值为1的整数变量，它的十六进制字节值为：

01 00 00 00

当你调用intswap()，编绎程序传递给它整个四字节数。同时，该功能正寻找一个双字节的整数，它将这数从栈中弹出 放入AX中，交换AH和AL，然而将该值返回。但是，编绎程序将四个字节压入栈中，要返回一个整数值，则将四个字节从EAX中弹出。

表2-1 多方面函数调用

函数名字	源程序模块
ASCIIZToLenStr	\MISC\AZLENSTR.ASM
ConvertNameToFullPath	\MISC\GETPATH.C
ConvertNameToVolumePath	\MISC\GETPATH.C
GetFullPath	\MISC\FULLPATH.C
IntSwap	\MISC\INTSWAP.ASM
LenStrCat	\MISC\LSTRCAT.ASM
LenStrCmp	\MISC\LSTRCMP.ASM
LenStrCpy	\MISC\LSTRCPY.ASM
LenToASCIIZStr	\MISC\LENAZSTR.ASM
LongSwap	\MISC\LONGSWAP.ASM
ParsePath	\MISC\GETPATH.C
StripFileServerFromPath	\MISC\SETPATH.C

包括文件：NWMISC.H

假如EAX包含如下的数据：

0C 3D 88 55

在你调用intswap()之前，intswap()将从栈中复制变量到AX中，然后交换字节，给出如下结果：

0C 3D 01 00

这个结果在EAX，当INTSWAP()返回时，开关顺序变量包含下列值：

00 01 3D 0C

Netware c接口程序在许多地方使用int来表示短int。检查和转换所有的短int是一项艰巨的任务。除非在32位机上，编绎程序识别一个int等于一个长int，否则你不必顾虑这个问题。

串转换——ASSIIZTOlenstr()和lenToASSIItstr()

有两个函数用于C和PASCAL表达式之间的串转换，ASCIIZTOlenstr()和LenToASCIIZstr()，尽管C格式对大多数在MS-DOS工作站上的Netware C编程是足够的，但在AFP服务群(将在第八章讨论)中的许多功能要求用pascal格式串。

两个函数均有相同的调用语法，这可以从它们的标准样式上看出：

```
extern int ASCIIZToLenStr(char *pszDestination,
   char *pszSource);
```