第 3 章　Linux 文件系统

文件系统是操作系统用于确定磁盘或分区上的文件的方法和数据结构，即文件在磁盘上的组织方法。文件系统由 3 部分组成：与文件管理有关的软件、被管理的文件以及实施文件管理所需的数据结构。从系统角度来看，文件系统是对文件存储空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。

3.1　Ubuntu 的文件系统

3.1.1　文件系统简介

计算机文件是数据元素的有序序列，根据不同的实现方式，这些数据元素可以是机器字、字符或二进制位。程序或用户只需通过文件系统就可以创建、修改、删除文件。每个文件都有一个符号化的名称，即文件名。用户可以通过指定文件名以及数据元素在文件中的线性索引，从而引用文件中的数据元素。

目录是文件系统维护所需的特殊文件，它包含了一个项目列表。一个计算机系统中有成千上万的文件，为了便于对文件进行存取和管理，计算机系统要建立文件的索引，即文件名和文件物理位置之间的映射关系，这种文件的索引称为文件目录。文件目录为每个文件设立一个表目。文件目录的表目中至少要包含文件名、物理地址、文件逻辑结构、文件物理结构和存取控制信息等，以建立起文件名与物理地址的对应关系，方便用户对文件的查找和修改等操作，实现按名存取文件。

文件系统是操作系统用于确定磁盘或分区上的文件的方法和数据结构，即文件在磁盘上的组织方法。文件系统也指用于存储文件的磁盘或分区。操作系统中负责管理和存储文件信息的部分称为文件管理系统，简称文件系统。从系统角度来看，文件系统是对文件存储空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立、保存、读出、修改、转储文件，控制文件的存取，当用户不再使用时删除文件等。

从另一个角度来看，文件系统还是操作系统在计算机的硬盘上存储和检索数据的逻辑方法，这些硬盘可以是本地驱动器，也可以是在网络上使用的卷或存储区域网络（Storage Area Network，SAN）上的导出共享等。一般来说，一个操作系统对文件的操作包括：创建和删除文件，打开文件以进行读写操作，在文件中搜索，关闭文件，创建目录以存储一系列文件，列出目录内容，从目录中删除文件，等等。

磁盘或分区和它所包括的文件系统的种类有很大的关系。少数程序直接对磁盘或分区的原始扇区进行操作，这可能会破坏文件系统。大部分程序基于特定文件系统进行操作，在其他类型的文件系统中不能工作。一个分区或磁盘在作为文件系统使用前，需要进行初始化，并将记录数据结构写到磁盘上。这个过程就称为文件系统的建立。

下面介绍几种常见的文件系统类型。

1. FAT 文件系统

FAT 的英文全称是 File Allocation Table，即文件分配表，最早于 1982 年开始应用于 MS-DOS 中。FAT 文件系统主要的优点是允许多种操作系统访问，如 MS-DOS、Windows 3.x、Windows 9x、Windows NT 和 OS/2 等。这一文件系统在使用时遵循 8.3 命名规则（即文件名最多为 8 个字符，扩展名为 3 个字符）。MS-DOS、Windows 3.x 和 Windows 95 都使用 FAT16 文件系统。

FAT 文件系统也是一种最初用于小型磁盘和简单文件夹结构的简单文件系统，用文件分配表映射和管理磁盘空间。它是专门为单用户操作系统开发的，不保存文件的权限信息，除了隐藏、只读等公共属性以外，不具备高级别安全防护措施。FAT 文件系统在磁盘的第一个扇区保存其目录信息。当文件改变时，FAT 必须随之更新，当复制多个小文件时，这种开销就会变得很大。FAT16 和 FAT32 能够管理的磁盘空间的大小不同。FAT16 最大只能支持 2GB，而 FAT32 可达到 2047GB。

FAT32 文件系统主要应用于 Windows 98 系统，它可以增强磁盘性能并增加可用磁盘空间。FAT32 的一个簇要比 FAT16 小很多，所以可以节省磁盘空间。而且它支持 2GB 以上的分区大小。默认情况下，Windows 98 也可以使用 FAT16。

2. NTFS 文件系统

NTFS（New Technology File System，新技术文件系统）最早专用于 Windows NT/2000 操作系统的文件系统。它支持文件系统故障恢复，尤其是大存储量的媒体和长文件名。

NTFS 是一个高度可靠、可恢复的文件系统，提供了 FAT 文件系统所没有的安全性、可靠性和兼容性。它提供了文件加密功能，能够大大提高信息的安全性，可以赋予单个文件和文件夹访问权限。它支持在大容量的硬盘上快速执行读、写和搜索等文件操作，支持文件系统恢复等高级操作。NTFS 为本地用户及网络上的远程用户都提供了文件和文件夹的安全特性。在 NTFS 分区上，可以为共享资源、文件夹以及文件设置访问权限。

与 FAT16 和 FAT32 相比，NTFS 的主要优点是：它通过使用相同卷大小条件下较小的簇，从而更有效地利用磁盘空间。只要简单地将文件系统从 FAT 转换为 NTFS，就可以释放出几百兆字节（MB）的磁盘空间。

NTFS 的主要弱点是只能被 Windows NT/2000 及以后的操作系统所识别。虽然它可以读取 FAT 文件系统和 HPFS（High Performance File System，高性能文件系统）文件系统的文件，但其文件却不能被 FAT 文件系统和 HPFS 文件系统所存取，因此兼容性较差。Windows NT 支持 FAT16、NTFS 文件系统，Windows 2000 支持 FAT16、FAT32 和 NTFS 文件系统。

3.1.2 Linux 文件系统架构

Linux 操作系统的核心是内核，而文件系统则是操作系统与用户进行交互的主要工具。不同的操作系统，其文件系统也不尽相同。例如，MS-DOS、Windows 95 等操作系统采用的是 FAT16 文件系统，Windows 2000/NT 等操作系统采用的是 NTFS 文件系统，Sun Solaris 操作系统采用的是 ZFS、UFS 文件系统。Linux 操作系统采用的文件系统一般为 Ext2、

Ext3 和 Ext4 等。Ext 是 Extended File System 的缩写，意为扩展的文件系统。

Linux 作为开源操作系统，其最大的优势就是支持多种文件系统。现代 Linux 内核几乎支持计算机系统中的所有文件系统。Linux 操作系统从一开始就追求让用户能够使用多个文件系统。事实上，用户在硬盘上使用一个或多个非 Linux 分区（如 FAT32 或 NTFS）的情况并不少见，如果用户想在 Linux 操作系统中使用非 Linux 文件系统，就必须在操作系统中挂载这些文件系统到 Linux 内核中。

Linux 文件系统采用了分层结构，如图 3-1 所示。

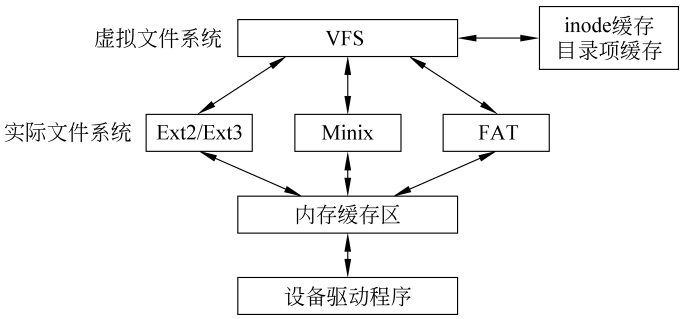

图 3-1　Linux 文件系统的分层结构

1. 设备驱动程序

文件系统需要利用存储设备来存储文件，因此，存储设备是文件系统的物质基础。除此之外，Linux 系统中的其他设备也作为文件，由文件系统统一管理。这些设备都由特定的设备驱动程序直接控制，它们负责设备的启动、数据传输控制和中断处理等工作。

Linux 系统中各种设备驱动程序都通过统一的接口与文件系统连接。文件系统向用户提供使用文件的接口，设备驱动程序则控制设备实现具体的文件 I/O 操作。

2. 实际文件系统

实际文件系统是以磁盘分区来划分的，每个磁盘分区由一个具体的文件系统管理，不同磁盘分区的文件系统可以不同。Linux 系统支持多种文件系统，除了专为 Linux 设计的 Ext2/Ext3、JFS、XFS、ReiserFS 和 NFS 之外，还支持以下文件系统：UNIX 系统的 sysv、ufs 和 bfs，Minix 系统的 minx 和 XIA，Windows 系统的 FAT16、FAT32 和 NTFS，以及 OS/2 系统的 hpfs，等等。这些文件系统都可以在 Linux 系统中工作。Linux 默认使用的文件系统是 Ext2/Ext3/Ext4。

3. 虚拟文件系统

实际文件系统通常是为特定的操作系统专门设计的，各种文件系统具有不同的组织结构和文件操作接口函数，相互之间往往差别很大。Linux 为了屏蔽各个文件系统之间的差别，为用户提供访问文件的统一接口，在具体的文件系统之上增加了一个称为虚拟文件系统（Virtual File System，VFS）的抽象层。

VFS 是 Linux 文件系统对外的接口。任何要使用文件系统的程序都必须经由这个接口来使用它。VFS 是一个异构文件系统之上的软件粘合层。有时也把 VFS 称为可堆叠的文件系统（stackable file system），因为 VFS 可以无缝地使用多个不同类型的文件系统，就像把多个文件系统堆叠在一起一样。通过 VFS，可以为访问文件系统的系统调用提供一个

统一的抽象接口。VFS 最早由 Sun 公司提出，以实现 NFS(Network File System，网络文件系统)，但是现在很多系统都采用了 VFS(包括 UNIX、Linux、FreeBSD、Solaris 等)。VFS 的原理如图 3-2 所示。

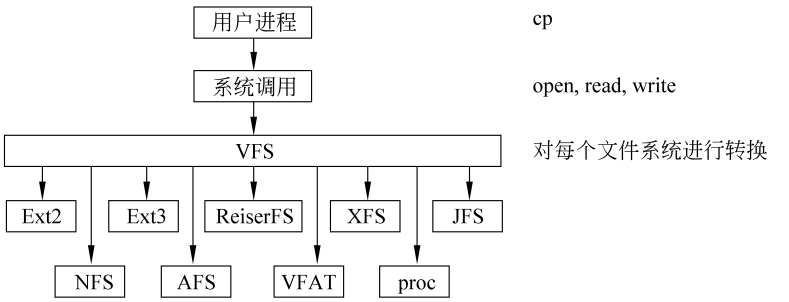

图 3-2　VFS 的原理示意图

虚拟文件系统运行在各个具体的文件系统之上，它采用一致的文件描述结构和文件操作函数，使得不同的文件系统能够按照同样的模式呈现在用户面前。有了 VFS，用户察觉不到文件系统之间的差异，可以使用同样的命令和系统调用来操作不同的文件系统，并可以在它们之间自由地复制文件。

VFS 的作用就是采用标准的 Linux 系统调用读写位于不同物理介质上的不同文件系统中的文件。VFS 是一个可以让 open、read、write 等系统调用不必关心底层的存储介质和文件系统类型就可以工作的粘合层。在 MS-DOS 操作系统中，要访问本地文件系统之外的文件系统，需要使用特殊的工具才能进行。而在 Linux 中，通过 VFS 这个抽象的通用访问接口屏蔽了底层的文件系统和物理介质的差异性。

每一种文件系统代码都隐藏了实现的细节。因此，对于 VFS 层和内核的其他部分而言，每一种文件系统看起来都是一样的。在 Linux 中，VFS 采用的是面向对象的编程方法。Linux 的 VFS 截取与文件系统有关的所有调用，为所有文件系统提供标准的接口或 API。对于用户而言，在使用 ls 命令列出不同物理介质上的文件时，实际上是没有区别的。

在 VFS 的公共文件模型中，目录被看作包含其他文件和其他目录列表的文件。某些文件系统(如 FAT16、FAT32 和 VFAT)在目录树中显示各个文件的存储位置。在这种情况下，目录不是普通的文件。因此，在文件系统上执行操作的过程中，文件系统必须将目录的视图动态建立为文件。显然，这样的视图实际上不是存储在文件系统中的，而是作为对象存在于内核空间中的。

此外，Linux 不会在实际的内核函数中执行与文件有关的系统调用。更确切地说，每个 read 和 icoctl 都将转化为各个文件系统处理该文件专用的函数的指针。为了实现这一点，VFS 的公共文件模型引入了公共文件系统控制块的概念，使各个文件系统与传统的 UNIX 方法相互协调。这些控制块如下：

(1) 超级块结构。用于记录有关已装载的文件系统的信息。这个结构实际上匹配存储在设备上的文件系统控制块。

(2) inode(信息节点)对象。用于记录关于特定文件的信息。这个对象通常指向设备上的文件控制块。每一个这样的对象都有一个信息节点号码，该号码唯一地指向文件系统

中的一个文件。

(3) file(文件)对象。用于记录目前打开的文件以及访问它的进程的信息。这个对象只保留在内核中,并在关闭文件时消失。

(4) dentry(目录项)对象。它是目录项目与相关联的文件之间的连接信息。每个文件系统都有这个对象的具体实现方式。

4. 缓存机制

文件系统和存储设备进行数据传输时,采用了缓存机制来提高存储设备的访问效率。缓存区是在内存中划分的特定区域,每次从外部设备读取的数据都暂时存放在这里。下次读取数据时,首先搜索缓存区。如果有需要的数据,则直接从这里读取;如果缓存区中没有,则再启动存储设备,读取相应的数据。写入磁盘的数据也先放入缓存区中,然后再分批写入磁盘中,使用缓存机制使得大多数数据传输都直接在进程的内存空间和缓存区之间进行,减少了对存储设备的访问次数,提高了系统的整体性能。VFS 使用了内存缓存区、目录项缓存以及 inode 缓存等机制,使得整个文件系统具有相当高的效率。

3.1.3 Ext2 文件系统

Linux 的 Ext2 文件系统和 UNIX 使用的其他文件系统非常相似,但更接近于 BSD 系统所用的 FFS(Fast File System,快速文件系统)。Ext2 文件系统的特点是存取文件的性能极好,对于中小型的文件更具优势,这主要得益于它的簇快取层的优良设计。其单一文件大小与文件系统本身的容量上限以及簇的大小有关。在 x86 计算机系统中,簇最大为 4KB,则单一文件大小上限为 2048GB,而文件系统的容量上限为 16 384GB。

Ext2 文件系统采用了多重索引结构,用 inode 中的索引表描述,如图 3-3 所示。

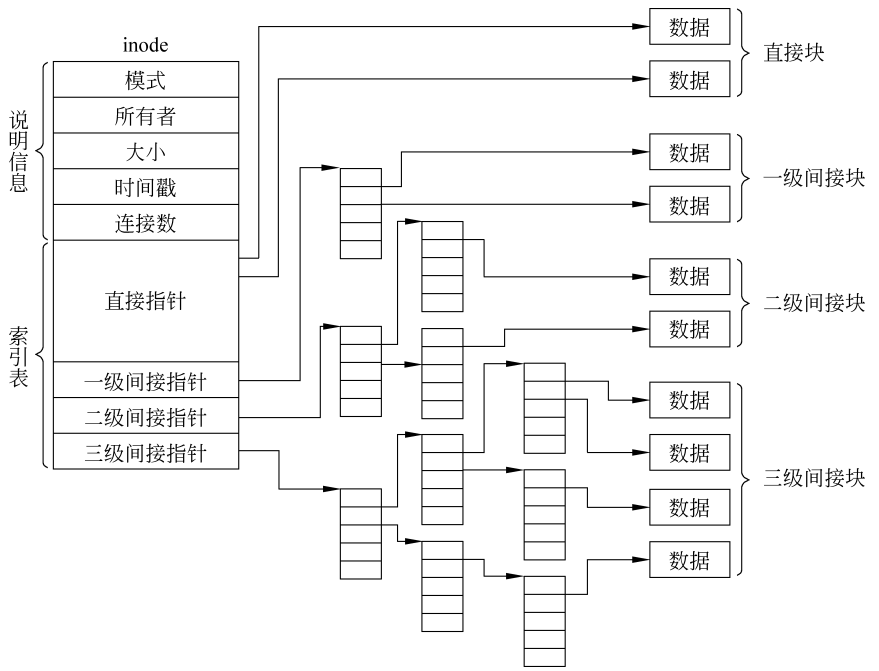

图 3-3 Ext2 文件的多重索引结构

索引表中前 12 个表项是直接指针，直接指向文件的数据块，这些块称为直接块。第 13 个表项是一个一级间接指针，它指向一个索引块，这个索引块中存放的是间接索引表。索引表的第 14 项和第 15 项提供了一个二次间接指针和一个三次间接指针，可提供对更多的间接块的索引。提供多级间接指针的目的是为了表达大型文件的结构。

Ext2 文件系统的默认块大小是 1KB。对于 12KB 以下的小文件，不需要使用间接索引，所有信息均在 inode 中，因此访问的速度非常快。较大的文件需要用到一个间接索引表。一个间接索引表含有 256 个间接指针，每个指针占 4B，则 1KB 的块可以容纳 256 个指针，可以索引 256 个间接块。因此，大小为 12～268KB 的文件需要一次间接索引，访问速度会有所降低。而对于大型的文件，可以使用二次间接指针甚至三次间接指针，得到最大约 16GB 的文件。

Ext2 的核心是两个内部数据结构，即超级块(superblock)和 inode。

超级块是一个包含文件系统重要信息(例如标签、大小、inode 的数量等)的表格，它是对文件系统结构的基础性、全局性描述，因此，没有了超级块的文件系统将不再可用。由于这个原因，Ext2 文件系统中不同位置存放着超级块的多个副本。

inode 是基本的文件级数据结构，文件系统中的每一个文件都可以在一个 inode 中找到它的描述。inode 描述的文件信息包括文件的创建和修改时间、文件大小、实际存放文件数据的块列表等。对于较大的文件，块列表可能包含附加数据块列表的磁盘位置(即间接块)，甚至有可能出现二级或三级间接块列表。文件名字通过目录项(directory entry，通常缩写为 dentry)关联到 inode，目录项由文件名和 inode 构成。

图 3-4 展示了 Ext2 文件系统的数据结构。

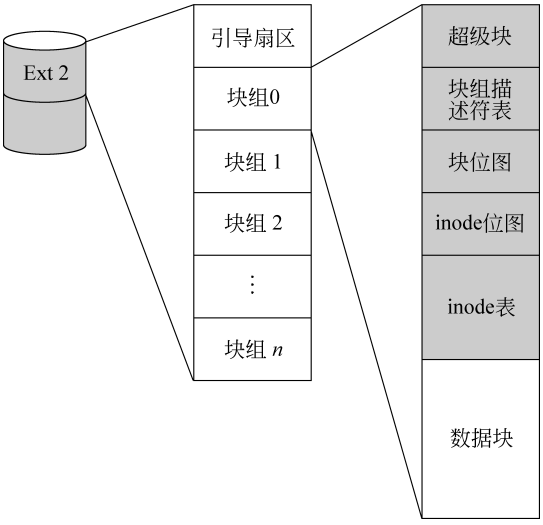

图 3-4　Ext2 文件系统的数据结构

Ext2 文件系统以引导扇区(boot sector)开始，紧接着是块组(block group)。因为 inode 表和存储用户数据的数据块(data block)可以位于磁盘相邻位置，这样就可以减少寻道的时间。为了性能的提升，整个文件系统被分为多个块组。一个块组由下面几项组成：

- 超级块：存储文件系统信息，必须位于每个块组的顶部。
- 块组描述符：存储块组的相关信息。
- 块位图：用于管理未使用的数据块。
- inode 位图：用户管理未使用的 inode。
- inode 表：每个文件都有一个相应的 inode 表，用来保存文件的元数据，例如文件模式、uid、gid、atime、ctime、mtime、dtime 和数据块的指针。
- 数据块：存储实际的用户数据。

每个块组组成的详细说明如下。

1. 超级块

超级块描述整个分区的文件系统信息，例如块大小、文件系统版本号、上次挂载的时间等。这些都是文件系统挂载、检查、分配、检索等操作的基本参数，是文件系统中最重要的数据，如果超级块损坏，则整个分区的文件系统就不再可用。超级块在每个块组的开头都有一个副本。

2. 块组描述符表

块组描述符表(Group Descriptor Table，GDT)由很多块组描述符(group descriptor)组成，整个分区分成多少个块组，就有多少个块组描述符。每个块组描述符存储一个块组的描述信息，例如，在这个块组中从哪里开始是 inode 表，从哪里开始是数据块，空闲的 inode 和数据块还有多少个，等等。和超级块类似，块组描述符表在每个块组的开头也都有一个副本。这些信息是非常重要的，一旦块组描述符意外损坏，就会丢失整个块组的数据，因此它有多个副本。通常内核只用到第 0 个块组中的块组描述符表，当执行 e2fsck 命令检查文件系统一致性时，第 0 个块组中的超级块和块组描述符表就会复制到其他块组。这样，当第 0 个块组的开头意外损坏时，就可以用其他副本来恢复，从而减小损失。

3. 块位图

一个块组中的块是这样被利用的：数据块存储文件中的数据。例如，某个分区的块大小是 1024B，某个文件是 2049B，那么就需要 3 个数据块，即使第三个块只存了 1B，也需要占用一个整块；超级块、块组描述符表、块位图、inode 位图、inode 表这几部分存储该块组的描述信息。块位图(block bitmap)是用来描述整个块组中哪些块已用、哪些块空闲的，它本身占一个块，其中的每个二进制位代表本块组中的一个块，这个二进制位为 1 表示该块已用，这个二进制位为 0 表示该块空闲。

4. inode 位图

inode 位图(inode bitmap)和块位图类似，本身占一个块，其中每个二进制位表示一个 inode 是否空闲。

5. inode 表

一个文件除了数据需要存储之外，一些描述信息也需要存储，例如文件类型(常规文件、目录、符号链接等)、权限、文件大小、创建/修改/访问时间等。利用 ls-l 命令看到的那些信息都保存在 inode 中，而不是数据块中。每个文件都有一个 inode，一个块组中的所有 inode 组成了 inode 表(inode table)。

inode 表占多少个块在格式化时就已决定，并写入块组描述符中。格式化工具 mke2fs 的默认策略是一个块组有多少个 8KB 就分配多少个 inode。由于数据块占整个块组的绝大

部分，也可以近似认为一个数据块有多少个 8KB 就分配多少个 inode。换句话说，如果平均每个文件的大小是 8KB，当分区存满的时候 inode 表会得到比较充分的利用，数据块也不浪费。如果这个分区保存的都是很大的文件，则数据块用完的时候 inode 会有一些浪费；如果这个分区保存的都是很小的文件，则有可能数据块还没用完 inode 就已经用完了，数据块可能有很大的浪费。如果用户在格式化时能够对这个分区以后要存储的文件大小做一个预测，也可以用 mke2fs 的-i 参数指定每多少字节分配一个 inode。

6. 数据块

数据块根据不同的文件类型有以下几种情况：

- 对于常规文件，文件的数据存储在数据块中。
- 对于目录，其下的所有文件名和目录名存储在数据块中。文件名保存在它所在目录的数据块中，除文件名之外，ls -l 命令看到的其他信息都保存在该文件的 inode 中。目录也是一种文件，是一种特殊类型的文件，即目录文件。
- 对于符号链接，如果目标路径名较短，则直接保存在 inode 中，以便快速查找；如果目标路径名较长，则分配一个数据块来保存它。

设备文件、FIFO 和 socket 等特殊文件没有数据块。设备文件的主设备号和次设备号保存在 inode 中。

为了找到组成文件的数据块，内核首先查找文件的 inode。当一个进程请求打开/var/log/messages 时，内核解析文件路径并查找根目录的目录项，获得其下的文件及目录信息。下一步，内核继续查找/var 的 inode 并查看/var 的目录项，它也包含其下的文件及目录信息。内核继续使用同样的方法，直到找到指定文件的 inode。Linux 内核使用文件对象缓存（如目录项缓存或 inode 缓存）来加快查找相应 inode 的速度。

当 Linux 内核找到文件的 inode 后，它将尝试访问实际的数据块。inode 保存了数据块的指针。内核通过指针可以获得数据块。对于大文件，Ext2 提供了直接或间接的数据块参照，如图 3-5 所示。

当建立一个新的文件时，Ext2 文件系统要为它分配一个 inode 和一定数目的数据块；当该文件被删除时，Ext2 文件系统将回收其占有的 inode 和数据块；当该文件在读写过程中增加或减少了内容时，Ext2 文件系统也需要动态地为它分配和回收 inode 和数据块。

分配时根据 inode 位图和块位图的记录为文件分配 inode 和数据块。分配策略在一定程度上决定了文件系统的整体效率。Ext2 文件系统会尽可能把同一个文件所使用的块、同一个目录所关联的 inode 存放在相邻的单元中，至少是在同一个块组中，以提高文件的访问效率。

另外，Ext2 文件系统还采用了预分配机制来保证文件内容扩展时空闲块的分配效率。在文件建立的时候，如果有足够的空闲块，就在相邻的位置为文件分配多于当前需要的块，称为预分配块；当文件内容扩展时，优先使用预分配块，可以提高分配效率，也可以保证这些块相邻。如果预分配块用完或者根本没有启动预分配机制，分配新块时也要尽可能保证与原有块相邻。

3.1.4 Ubuntu 的目录结构

无论何时，当前工作目录中的所有文件都是可以直接存储的，并且通过名字可以直接引用文件。而对于非当前目录中的文件，必须在文件名之前加上各级目录构成的路径才能访

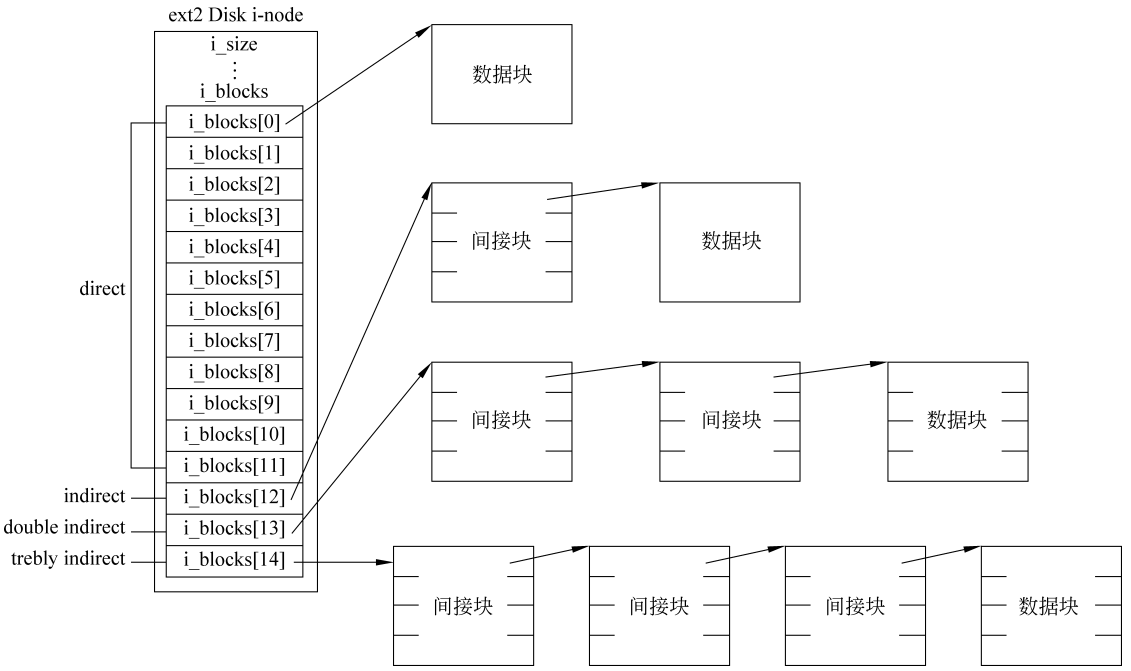

图 3-5　Ext2 文件系统直接或间接的数据块参照

问。文件的路径名指的就是从某个目录开始，穿过整个文件系统，直至到达目标文件而经过的目录层次。例如，从根目录开始，中间经过 usr 和 bin 两级子目录到达 find 文件的路径就是 find 文件的路径名，把上述访问路径写成 Linux 文件系统中的路径名就是/usr/bin/find。

每个目录中均包含以句点(.)和双句点(..)命名的两个特殊的目录文件，分别表示当前目录及其父目录。这两个特殊目录把文件系统中的各级目录有机地联系在一起。“.”是当前目录的别名，要访问当前目录中的文件时，都可以直接使用“.”，而不必明确给出当前目录名。“..”是当前目录的父目录的别名，从任何目录位置开始，都可以使用“..”逐层攀升到文件系统层次结构的最上层。

路径名要么是由斜线字符分隔的一系列名字，要么是单个名字。在一系列名字中，最后一个名字是实际的文件，其他名字均为目录。文件可以是任何类型的文件。

Linux 文件系统采用树状分层结构，且只有一个根节点。与 Windows 一样，在 Linux 中路径分为绝对路径和相对路径。

(1) 绝对路径。指文件的准确位置且以根目录为起点。例如，/usr/game/gnect 就是一个绝对路径，表示位于/usr/game/下的四子连线游戏。

(2) 相对路径。是相对于当前目录的一个文件或目录的位置，还以上面的/usr/game/gnect 为例，如果用户现在处于/usr 中，只需要 game/gnect 就可以确定这个文件，而不需要将根目录和 usr 目录写出。

同样，Ubuntu 的目录众多，但所有目录都在根目录下，所以在安装时一定要有一个与根目录对应的磁盘分区才能安装系统。有序的组织结构有助于用户访问、管理和维护 Ubuntu 系统。Ubuntu 的目录结构如图 3-6 所示。

Linux 目录结构的描述如表 3-1 所示。

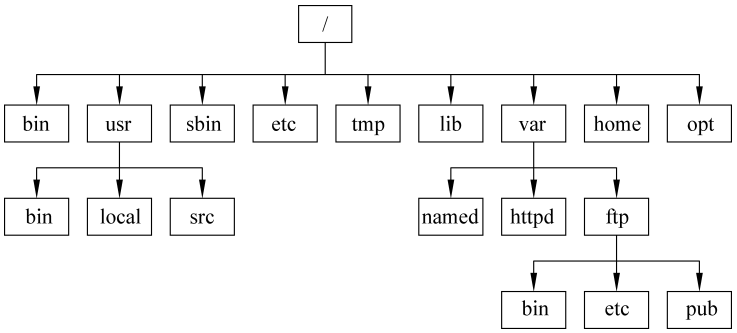

图 3-6　Ubuntu 的目录结构

表 3-1　Linux 目录结构的描述

目　录	描　　述
/	Linux 文件系统的根目录
/bin	存放系统中最常用的可执行文件(二进制文件)。系统所需要的基本命令位于此目录下，也是最小系统所需要的命令，例如 ls、cp、mkdir 等命令。/usr/bin 目录中的文件是普通用户可以使用的命令
/boot	存放 Linux 内核和系统启动文件，包括 grub、lilo 启动程序
/dev	存放所有设备文件，包括硬盘分区、键盘、鼠标、USB、tty 等
/etc	存放系统所有配置文件，例如，passwd 存放用户账户信息，hostname 存放主机名。/etc/fstab 用于开机自动挂载一些分区，在里面写入一些分区的信息，就能实现开机挂载分区
/home	用户主目录的默认位置
/initrd	存放启动时挂载 initrd.img 映像文件以及所需设备模块的目录
/lib	存放共享的库文件，包含许多被/bin 和/sbin 中的程序使用的库文件
/lost+found	在 Ext2 或 Ext3 文件系统中，当系统意外崩溃或意外关机时产生的一些文件碎片放在这里。在系统启动的过程中，fsck 工具会检查这里，并修复已经损坏的文件系统。有时系统发生问题，很多文件被移到这个目录中，可以用手工的方式修复文件或将其移到原来的位置上
/media	在这个目录下自动创建即插即用型存储设备的挂载点。例如，U 盘系统自动挂载后，会在这个目录下产生一个目录；CD-ROM/DVD-ROM 自动挂载后，也会在这个目录下创建一个目录，存放临时读入的文件
/mnt	通常用于作为被挂载的文件系统的挂载点
/opt	作为可选文件和程序的存放目录，有些自定义软件包也会被安装在这里，用户自己编译的软件包也可以安装在这个目录中
/proc	存放所有标识为文件的进程，它们通过进程号或其他的系统动态信息进行标识。例如，CPU、硬盘分区、内存信息等就存放在这里
/root	根用户(超级用户)的主目录
/sbin	存放涉及系统管理的命令，也就是 root 用户可执行的命令。这个目录和/usr/sbin、/usr/X11R6/sbin、/usr/local/sbin 目录是相似的
/srv	存放系统所提供的服务数据

续表

目　录	描　述
/sys	用于将系统设备组织成层次结构，并向用户提供详细的内核数据信息
/tmp	临时文件目录。/var/tmp 目录与之相似
/usr	用于存放与系统用户直接有关的文件和目录，如应用程序及支持系统的库文件。其主要的子目录如下：/usr/X11R6：X Window 系统。 /usr/bin：用户管理员的标准命令。 /usr/include：C/C++ 等开发工具语言环境的标准 include 文件。 /usr/lib：应用程序及程序包的链接库。 /usr/local：系统管理员安装的应用程序。 /usr/local/share：系统管理员安装的共享文件。 /usr/sbin：用户和管理员的标准命令。 /usr/share：使用手册等共享文件。 /usr/share/dict：词表的。 /usr/share/man：系统使用手册。 /usr/share/misc：一般数据。 /usr/share/sgml：SGML 数据。 /usr/share/xml：XML 数据
/var	通常用于存放长度可变的文件，例如日志文件和打印机文件。其主要的子目录如下： /var/cache：应用程序缓存目录。 /var/crash：系统错误信息。 /var/games：游戏数据。 /var/lib：各种状态数据。 /var/lock：文件锁定记录。 /var/log：日志记录。 /var/mail：电子邮件。 /var/opt：/opt 目录的变量数据。 /var/run：进程的标识数据。 /var/spool：存放电子邮件、打印任务等的队列。 /var/tmp：临时文件目录

注意：与 Windows 不同，在 Ubuntu 中是严格区分大小写的。例如，文件 File.txt，FILE.txt、FILE.TXT 是 3 个不同的文件，在文件处理时要特别注意。

同时，还有与 Windows 不同的是，在 Windows 中后缀名是一个非常重要的标识符，是根据后缀名来判断文件的类型并进行处理的。例如，A.c 是一个 C 语言的源程序，而 A.txt 是一个纯文本文件。而在 Linux 系统中，文件类型与后缀名是没有直接关系的。

3.2　创建、挂载与卸载文件系统

3.2.1　创建文件系统

在安装操作系统时，安装程序将会引导用户提供必要的数据或作出相应的选择，然后自动划分磁盘分区，创建文件系统。在日常的应用过程中，仅当需要增加新盘或使用移动硬盘，改变现有的磁盘分区结构时，才有可能需要自己手工创建文件系统。在磁盘等存储设备

分区之后，即可着手创建文件系统。

创建文件系统时，主要有两种方法：

（1）使用最基本的、通用的 mkfs 命令。在选定的磁盘分区中创建指定的文件系统。

（2）利用各种特定的工具，如 mke2fs、mkfs.ext2 或 mkfs.vfat 等，在选定的磁盘分区上直接创建特定类型的文件系统。

创建文件系统时，最基本的方法是使用 mkfs 命令。在 Linux 系统中，mkfs 命令实际上只是各种文件系统创建程序的一个总控程序，根据指定的文件系统类型，mkfs 将会调用特定文件系统的创建程序 mkfs.fs，其中 fs 可以是 ext2、ext3 或 vfat 等。因此，在创建一个具体的文件系统时，可以使用特定的文件系统创建命令。例如，为了创建一个 Ext2/Ext3 文件系统，可以直接使用 mkfs.ex2 或 mkfs.ext3 等命令，也可以使用等价的 mke2fs 命令。对于 Ext2/Ext3 文件系统而言，mke2fs 命令是最基本的工具。

用于创建 Ext2/Ext3 文件系统的 mkfs 与 mke2fs 命令的语法格式如下。

（1）mkfs 命令：

```
mkfs [-V] [-t fstype] [fs-options] device [size]
```

（2）mke2fs 命令：

```
mke2fs [-c|-l filename] [-b block-size] [-f fragment-size]
       [-i bytes-per-inode] [-I inode-size] [-J journal-options]
       [-G meta group size] [-N number-of-inodes]
       [-m reserved-blocks-percentage] [-o creator-os]
       [-g blocks-per-group] [-L volume-label] [-M last-mounted-directory]
       [-O feature[,...]] [-r fs-revision] [-E extended-option[,...]]
       [-T fs-type] [-U UUID] [-jnqvFSV] device [blocks-count]
```

其中的主要参数说明如下：

- -b：指定块大小，单位为字节。
- -c：检查是否有损坏的块。
- -f：指定不连续段的大小，单位为字节。
- -i：指定每多少字节创建一个 inode。
- -N：指定要建立的 inode 数目。
- -L：设置文件系统的标签名称。
- -m：指定给管理员保留的块的比例，默认值为 5%。
- -M：记录最后一次挂入的目录。
- -r：指定要建立的 Ext2 文件系统版本。
- -V：显示命令的版本号。

下面以一个例子来说明 mkfs 命令的执行过程。首先在虚拟机中添加一块 512MB 的未分区硬盘。添加完毕后重启系统，然后利用 fdisk -l 命令查看新增硬盘的信息。其中 Disk/dev/sdb 即为新增的硬盘，因为没有创建文件系统，所有没有包含任何有效的分区表。

```
user@user-desktop:~$ sudo fdisk -l
[sudo] password for user:
```

```
Disk /dev/sda: 10.7 GB, 10737418240 bytes
255 heads, 63 sectors/track, 1305 cylinders
Units =cylinders of 16065 * 512 =8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000e8668
Device Boot    Start  End    Blocks  Id  System
/dev/sda1   *       1  1244  9990144  83  Linux
/dev/sda2       1244  1306   492545   5  Extended
/dev/sda5       1244  1306   492544  82  Linux swap / Solaris
Disk /dev/sdb: 536 MB, 536870912 bytes
64 heads, 32 sectors/track, 512 cylinders
Units =cylinders of 2048 * 512 =1048576 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
Disk /dev/sdb doesn't contain a valid partition table
user@user-desktop:～$
```

接下来使用 mkfs -t ext2 /dev/sdb 命令对其进行创建文件系统的操作,以下为具体执行过程:

```
user@user-desktop:～$sudo mkfs -t ext2 /dev/sdb
mke2fs 1.41.11 (14-Mar-2010)
/dev/sdb is entire device, not just one partition!
无论如何也要继续? (y,n) y
文件系统标签=
操作系统: Linux
块大小=4096 (log=2)
分块大小=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
32768 inodes, 131072 blocks
6553 blocks (5.00%) reserved for the super user
第一个数据块=0
Maximum filesystem blocks=134217728
4 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304
正在写入 inode 表: 完成
Writing superblocks and filesystem accounting information: 完成
This filesystem will be automatically checked every 28 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
user@user-desktop:～$
```

最后要说明的是，在使用 mkfs 命令创建文件系统时，如果未使用-L 选项指定文件系统的卷标，在创建文件系统后，也可以使用 e2label 命令指定文件系统的卷标。e2label 命令的主要功能就是显示和命名未安装文件系统的卷标。

e2label 命令的格式如下：

```
e2label device [newlabel]
```

其中，device 为磁盘分区的设备名称，例如/dev/sda6；newlabel 表示文件系统的卷标。如果未指定 newlabel，e2label 命令将输出文件系统的卷标。需要指出的是，卷标不能超过 16 个字符。下面将刚才创建的文件系统的卷标命名为 data1：

```
$ sudo e2label /dev/sdb
$ sudo e2label /dev/sdb data1
$ sudo e2label /dev/sdb
data1
```

3.2.2 挂载文件系统

在 Ubuntu 系统中，对于一个文件系统或分区而言，如果想要使用它，必须先对其进行挂载操作。挂载就是把新建的文件系统安装到 Linux 文件系统目录层次结构的某个挂载点上，然后才能使用新建的文件系统。也就是说，挂载点必须是目录，可以将挂载看成一个链接动作。

例如，如果想使用/dve/sdb 分区，需要对该分区进行读写数据的操作，但是不能直接对其进行操作，需要建立一个目录，使用挂载操作建立它与分区的对应关系。而这个目录就代表了/dve/sdb 分区，挂载后对于目录的数据读写操作本质上就是对/dve/sdb 分区的读写操作。与之对应，卸载就是断开目录与/dve/sdb 分区之间的对应关系，但并不删除目录，/dve/sdb 分区也继续存在，如果要读写数据，需要再次进行挂载操作。

对于任何文件系统——FAT 16/FAT 32 类型的 DOS 文件系统、NTFS 类型的 Windows 文件系统以及 iso9660 格式的 CD/DVD 等，都需要先进行挂载，然后才能使用。

挂载文件分区的命令是 mount，详见第 4 章。可以通过 mount 命令查看当前系统的挂载信息：

```
user@user-desktop:~$ mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
none on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
none on /dev type devtmpfs (rw,mode=0755)
none on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
none on /dev/shm type tmpfs (rw,nosuid,nodev)
none on /var/run type tmpfs (rw,nosuid,mode=0755)
none on /var/lock type tmpfs (rw,noexec,nosuid,nodev)
```

```
none on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,
nodev)
gvfs-fuse-daemon on /home/user/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,
nodev,user=user)
user@user-desktop:~$
```

可以看到,刚才建立的512MB硬盘分区/dev/sdb并未挂载,因此无法使用。下面将建立data1目录,并将/dev/sdb挂载到data1目录下:

```
user@user-desktop:~$sudo mkdir /data1
user@user-desktop:~$sudo mount /dev/sdb /data1/
```

挂载完毕后,再次使用mount命令查看挂载信息,可以看到,分区/dev/sdb已经被挂载到data1目录下。

```
user@user-desktop:~$mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
none on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
none on /dev type devtmpfs (rw,mode=0755)
none on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
none on /dev/shm type tmpfs (rw,nosuid,nodev)
none on /var/run type tmpfs (rw,nosuid,mode=0755)
none on /var/lock type tmpfs (rw,noexec,nosuid,nodev)
none on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,
nodev)
gvfs-fuse-daemon on /home/user/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,
nodev,user=user)
/dev/sdb on /data1 type ext2 (rw)
user@user-desktop:~$
```

还可以通过查看/etc/mtab文件获得挂载信息。具体操作如下:

```
user@user-desktop:/$cat /etc/mtab
/dev/sda1 / ext4 rw,errors=remount-ro 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
none /sys sysfs rw,noexec,nosuid,nodev 0 0
none /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
none /dev devtmpfs rw,mode=0755 0 0
none /dev/pts devpts rw,noexec,nosuid,gid=5,mode=0620 0 0
none /dev/shm tmpfs rw,nosuid,nodev 0 0
```

```
none /var/run tmpfs rw,nosuid,mode=0755 0 0
none /var/lock tmpfs rw,noexec,nosuid,nodev 0 0
none /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
binfmt_misc /proc/sys/fs/binfmt_misc binfmt_misc rw,noexec,nosuid,nodev 0 0
gvfs-fuse-daemon /home/user/.gvfs fuse.gvfs-fuse-daemon rw,nosuid,nodev,user=
user 0 0
/dev/sdb /data1 ext2 rw 0 0
user@user-desktop:/$
```

在 data1 目录下建立测试目录 testdir，并用 ls -l 查看目录信息，可以看到挂载成功，目录被正确建立。

```
user@user-desktop:/$sudo mkdir /data1/testdir
user@user-desktop:/$ls -l /data1/
总计 20
drwx------2 root root 16384 2012-05-01 00:44 lost+found
drwxr-xr-x 2 root root 4096 2012-05-01 01:08 testdir
user@user-desktop:/$
```

3.2.3 卸载文件系统

卸载文件系统意味着把文件系统从挂载点移走，删除/etc/mtab 文件中的挂载项。在文件系统的管理与维护工作中，有些工作只能在未挂载的文件系统中执行，因此需要进行卸载文件系统的操作。此外，当不再需要临时挂载的文件系统时，也应及时进行卸载操作，避免数据的误操作。

要卸载的文件系统必须处于空闲状态。一般来说，不能卸载一个尚处于工作中的文件系统。也就是说，当用户正在访问文件系统中的某个目录，或者进程打开了文件系统中的某个文件，或者文件系统正处于共享状态时，就无法卸载文件系统。卸载文件系统的命令是 umount。

下面卸载刚才挂载的文件系统/dev/sdb，具体操作如下：

```
user@user-desktop:/$sudo umount /dev/sdb
user@user-desktop:/$mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
none on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
none on /dev type devtmpfs (rw,mode=0755)
none on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
none on /dev/shm type tmpfs (rw,nosuid,nodev)
none on /var/run type tmpfs (rw,nosuid,mode=0755)
none on /var/lock type tmpfs (rw,noexec,nosuid,nodev)
none on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
```

```
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,
nodev)
gvfs-fuse-daemon on /home/user/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,
nodev,user=user)
user@user-desktop:/$
```

可以看到,文件系统已经被卸载。查看/etc/mtab 文件可以得到同样的结果。

```
user@user-desktop:/$cat /etc/mtab
/dev/sda1 / ext4 rw,errors=remount-ro 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
none /sys sysfs rw,noexec,nosuid,nodev 0 0
none /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
none /dev devtmpfs rw,mode=0755 0 0
none /dev/pts devpts rw,noexec,nosuid,gid=5,mode=0620 0 0
none /dev/shm tmpfs rw,nosuid,nodev 0 0
none /var/run tmpfs rw,nosuid,mode=0755 0 0
none /var/lock tmpfs rw,noexec,nosuid,nodev 0 0
none /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
binfmt_misc /proc/sys/fs/binfmt_misc binfmt_misc rw,noexec,nosuid,nodev 0 0
gvfs-fuse-daemon /home/user/.gvfs fuse.gvfs-fuse-daemon rw,nosuid,nodev,user=
user 0 0
user@user-desktop:/$
```

再次用 ls -l 查看 data1 目录,会发现 testdir 目录已经不存在了,说明卸载成功。

```
user@user-desktop:/$ls -l /data1/
总计 0
user@user-desktop:/$
```

本章小结

本章介绍了 Linux 的文件系统以及 Ubuntu 的文件系统中的目录结构。对于文件系统的具体使用,以创建文件系统、挂载文件系统和卸载文件系统为例进行了说明。

实 验 3

题目:文件系统的创建、挂载和卸载。

要求:

(1) 在虚拟机中增加一个新硬盘,利用 mkfs 命令为它创建一个文件系统。

(2) 将它挂载到 data1 目录下。显示挂载后的结果。

(3) 卸载该文件系统。显示卸载后的结果。

习 题 3

1. 什么是文件系统？
2. 什么是文件目录？文件目录包括哪些具体内容？
3. 比较下面几种文件系统的不同：FAT16、FAT32、NTFS。
4. Linux 系统可以支持哪些文件系统类型？其中默认的是什么文件系统类型？
5. 什么是虚拟文件系统？
6. 什么是绝对路径、相对路径？
7. 详细解释 Ubuntu 系统的目录结构中各部分的主要功能。
8. 如何创建、挂载、卸载文件系统？

第 4 章　Linux 常用命令

Linux 系统提供了一整套十分完备的命令，利用这些命令可以高效地完成所有的操作任务。使用命令方式进行操作，具有比图形化操作更加快捷高效的特点，但是命令方式不够直观，需要用户熟练掌握命令的用法、格式以及选项和参数等内容，只有通过不断使用才能得心应手。

4.1　Linux 命令

4.1.1　Shell 程序的启动

要使用命令，必须先启动 Shell 程序。当用户成功登录字符终端后，Shell 也同时启动。当用户登录到图形桌面后，在系统中找到终端即可启动 Shell。例如，对于 Ubuntu 18.04 LTS 桌面系统，用户可以通过使用 Ctrl＋Alt＋T 组合键启动 Shell。或者，可以单击桌面左侧下角的显示应用程序图标，在搜索栏输入 terminal，单击“搜索”按钮，系统会自动搜索出终端并启动它，同时，左侧 dock 面板上也将出现终端图标。为了方便后续的使用，用户可以在此时右击左侧 dock 面板上的终端图标，在快捷菜单中选择“添加到收藏夹”命令，把终端图标添加至 dock 面板内。这样，以后在需要使用终端时，直接单击 dock 面板上的终端图标即可。

终端提供了在图形环境下运行的字符命令行界面。Shell 启动后，显示命令提示符，普通用户默认的提示符是 $，root 用户默认的提示符是 #。普通用户的 Shell 启动界面如图 4-1 所示，root 用户的 Shell 启动界面如图 4-2 所示。

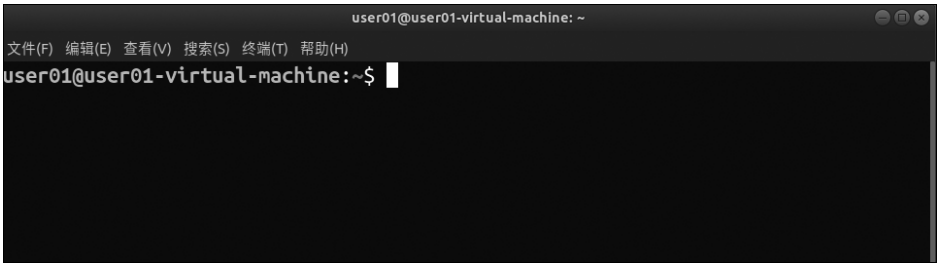

图 4-1　普通用户的 Shell 启动界面

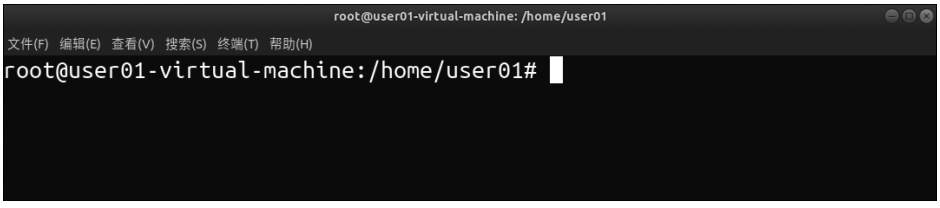

图 4-2　root 用户的 Shell 启动界面

4.1.2 命令的格式

Shell 命令是由命令名和多个选项以及参数组成的，各部分之间用空格分隔。Shell 命令是严格区分大小写的，因此，在使用 Shell 命令时应特别注意。

Shell 命令的格式如下：

命令名 ［-选项…］［参数…］

命令名是命令的名称，通常为小写。选项是以“—”(减号)作为前缀的，减号代表选项的开始。一个命令可能会有多个选项，可以单独使用，也可以组合使用。

4.2 目录操作基本命令

4.2.1 ls 命令

ls 命令是最常用的命令之一。它与 MS-DOS 系统的 dir 命令类似，用户可以利用 ls 命令查看某个目录下的所有内容。默认情况下，显示的条目按字母顺序排列。

【功能】 列出当前目录下的所有内容。ls 是 list 的缩写。

【格式】 ls ［选项］［文件名或目录名］

【说明】 各选项的作用如表 4-1 所示。

表 4-1 ls 命令各选项的作用

选 项	作 用
-s	显示每个文件的大小
-S	按文件的大小排序
-a	显示目录中的全部文件，包括隐藏文件
-l	使用长列表格式，显示文件详细信息
-t	按文件修改的时间排序显示
-F	显示文件类型描述符。例如，* 为可执行的普通文件，/为目录文件

在使用 ls 命令显示文件及目录的信息时，会发现文件及目录列表中有多种颜色出现。Linux 中不同的颜色代表不同的含义。

- 黑色(默认色)代表普通文件。
- 绿色代表可执行文件。
- 红色代表 tar 包文件，即 Linux 下的压缩打包文件。
- 蓝色代表目录文件，即一个目录，在 Windows 中称为文件夹。
- 水红色代表图像文件。
- 青色代表链接文件，此类文件相当于快捷方式。
- 黄色代表设备文件，即一个设备。