第 7 章 Linux 磁盘和文件系统管理

一、认识 Linux 文件系统

1.文件系统的特性

由于每种操作系统所设定的文档属性和权限并不相同，为了存放这些文件所需的数据，因此就需要将分区槽进行格式化，以成为操作系统能够利用的文件系统格式(文件系统)。

windows 操作系统主要使用的是 FAT 和 NTFS，而 Linux 主要是 Ext2（Linuxsecondextended 文件系统，ext2fs）。默认情况下，Windows 是无法识别 EXT2 文件系统的。

现在 LVM 或软磁盘阵列可以将一个分区划分为多个文件系统，也可以将多个分区合成一个文件系统。所以目前格式化已经不再是针对分区而言，我们可以说**一个可被挂载的数据为一个文件系统而不是一个分区**。

文件系统是如何运行的呢？较新的操作系统的文件数据除了文件实际内容外，通常含有非常多的属性，例如 Linux 操作系统的文件权限(rwx)与文件属性(拥有者、用户组、时间参数等)。**文件系统通常会将这两部份的数据分别存放在不同的区块，权限与属性放置到 inode 中，至于实际数据则放置到 datablock 区块中**。另外，还有一个超级区块(superblock)会记录整个文件系统的整体信息，包括 inode 与 block 的总量、使用量、剩余量等。

每一个 inode 和 block 都要编号。inode、block、superblock 的意义如下：

（1）inode：记录文件的属性，一个文件占用一个 inode，同时记录此文件的数据所在的 block 编号；

（2）superblock：记录此文件系统的整体信息，包括 inode/block 的总量、使用量、剩余量，以及文件系统的格式与相关信息等；

（3）block：实际记录文件的内容，如果文件过于大，会使用多个 block。

我们将 inode 于 block 区块用图解来说明一下。如图 1.1 所示，文件系统先格式化出 inode 与 block 的区块，假设某一个文件的属性与权限数据是放置到 inode4，而这个 inode 记录了文件数据的实际放置点为 2,7,13,15 这四个 block 编号，此时我们的操作系统就能够根据此来排列磁盘的阅读顺序将四个 block 内容读出来，数据的读取就如同下图中的箭头示。

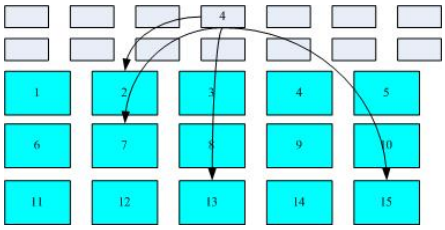

图 1.1inode 和 block 的演示

这种数据存取方式被称为**索引式文件系统**（indexedallocation）。

例如 FAT 格式这种文件系统并没有 inode 的存在，所以它没有办法把所有的 block 一口气读出来，只能读完一个 block，根据这个 block 中记录的编号去寻找下一个 block。如果同一个文件数据写入的 block 太分散，磁盘读取头将无法在磁盘转一圈就读到所有的数据，因此磁盘就会多转好几圈才能完整的读取到这个文件的内容。我们常说的磁盘整理，原因就是因为**文件写入的 block 太分散，会导致文件读取性能变差**。解决方法就是通过**碎片整理将同一个文件所属的 blocks 整理在一起**。

2.Ext2 文件系统（inode）

文件系统在最初就将 inode 和 block 规划好了，除非格式化，否则 inode 与 block 固定后

就不再变动。但是若文件系统高达数百 GB 时，那么将 inode 和 block 放在一起并不合适，为此，Ext2 文件系统在格式化的时候基本上是区分多个区块用户组（blockgroup）的，每个去看用户组都有自己的 inode/superblock/block 系统。如下图所示。

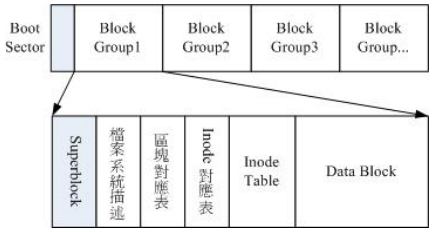

图 1.2ext2 文件系统示意图

系统最前面有一个启动扇区(bootsector)，这个启动扇区可以**安装开机管理程序**，这个设计非常重要，通过这个扇区我们可将不同的开机管理程序安装到不同的文件系统最前面，而不是覆盖整个磁盘唯一的 MBR，这样也才能够制作出多重引导的环境。

图 1.2 中每个 BlockGroup 里面都有 6 个部分，分别介绍如下：

（1）superblock（超级块）

它记录的信息主要有：

①block 与 inode 的总量；

②未使用与已使用的 inode/block 数量；

③block 与 inode 的大小(block 为 1,2,4K，inode 为 128bytes 或 256bytes)；

④文件系统的挂载时间、最近一次写入数据的时间、最近一次检验磁盘(fsck)的时间等文件系统的相关信息；

④一个 validbit 数值，若此文件系统已被挂载，则 validbit 为 0，若未被挂载，则 validbit 为 1。

Superblock 是非常重要的，因为我们这个文件系统的基本信息都写在这里，因此，如果 superblock 不存在了，你的文件系统可能就需要花费很多时间去恢复。一般来说，superblock 的大小为 **1024bytes**。

此外，每个 blockgroup 都可能含有 superblock。但是我们也说一个文件系统应该仅有一个 superblock 而已，那是怎么回事啊？事实上除了第一个 blockgroup 内会含有 superblock 之外，后续的 blockgroup 不一定含有 superblock，如果它含有，则该 superblock 主要是做为第一个 blockgroup 内 superblock 的备份。

（2）文件系统 Description（文件系统描述说明）

这个部分描述每个 blockgroup 的开始与结束的 Blcok 编码，同时说明每个区段（superblock/bitmap/inodemap/datablock）分别介于哪一个 block 编码之间。

（3）blockbitmap（区块对照表）

blockbitmap 中记录着哪些 block 是空的，哪些 block 是使用中的。

（4）inodebitmap（inode 对照表）

这个是记录使用与未使用的 inode 的编号。

（5）inodetable（inode 表格）

inode 记录的文件数据至少有以下几项：

①该文件的存取模式(read/write/excute)；

②该文件的拥有者与用户组(owner/group)；

③该文件的容量；

④该文件建立或状态改变的时间(ctime)；

⑤最近一次的读取时间(atime)；

⑥最近修改的时间(mtime)；

⑦定义文件特性的标志(flag)，如 SetUID 等；

⑧该文件真正内容的指向(pointer)。

inode 的**数量和大小也是在格式化时就已经固定**另外它还有以下特点：

①每个 inode 大小均固定为 128bytes(新的 ext4 与 xfs 可设定到 256bytes)；

②每个文件都仅会占用一个 inode 而已，因此文件系统能够建立的文件数量与 inode 的数量有关；

③系统读取文件时需要先找到 inode，并分析 inode 所记录的权限与用户是否符合，若符合才能够开始实际读取 block 的内容。

接下来，以 EXT2 为例来分析下 inode 和 block 的关系。inode 大小为 128bytes，其中 block 号码需要 4bytes，假如我们有一个 400MB 的文件，且每个 block 为 4k，那么我们需要记录 10W 个 block 号码，128bytes 是远远不够的。为了解决这一问题，inode 记录 block 编号的区域被定义为了 **12 个直接、1 个间接、1 个双间接、1 个三间接**的记录区，如图 1.3 所示。

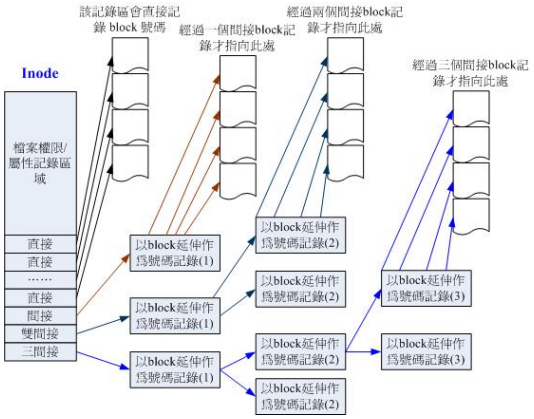

图 1.3inode 记录 block 的区域

上图中的 inode 最上面是记录文件权限和属性的地方，下面就是 12 个直接指向 block 区域的编号。下面一个就是所谓的间接区，它就是**拿一个 block 来当作记录 block 号码的记录区**，如果文件太大的时候，就会使用间接的 block 来记录编号，当文件持续增大，就会使用双间接乃至三间接。

假设我们的 block 大小为 1k，那么就可以有以下的情况：

①12 个直接：12*1K=12K。直接指向，故可指向 12 个 block。

②一个间接：256*1K=256K。由于间接区利用一个 block 来记录 block 编号，每个编号 4bytes，所以可以记录 256 个编号。

③双间接：256*256*1K=65536K。双间接的第一层有 256 个区，第二层有 256 个区。

④三间接：256*256*256*1K=16777216K=16384MB=16G。如（3）同理。

⑤总额：12+256+256*256+256*256*256（K）≈16G。

（6）datablock（数据块）

datablock 是放置文件内容数据的地方，在 EXT2 中所支持的 block 大小有 **1K**、**2K**、**4K**。因 block 大小而产生的 EXT2 文件系统限制如下：

Block 大小	1KB	2KB	4KB
最大单一文件限制	16GB	256B	2TB
最大文件系统总容量	2TB	8TB	16TB

block 块有以下特点：

①原则上，block 的大小与数量在格式化完就不能够再改变了（除非重新格式化）；

②每个 block 内最多只能够放置一个文件的数据；

③如果文件大于 block 的大小，则一个文件会占用多个 block 数量；若文件小于 block，则该 block 的剩余容量就不能够再被使用了(磁盘空间会浪费)。

每个区段与 superblock 的信息都可以使用 **dumpe2fs** 这个命令来查询。

[root@study~]#dumpe2fs[-bh]设备文件名 选项与参数： -b：列出保留为坏道的部分。 -h：仅列出 superblock 的数据，不会列出其他的区段内容。

输入 dumpe2fs/dev/sda 后的结果一部分如下。

```
Filesystem OS type:       Linux
Inode count:              1966080
Block count:              7863808
Reserved block count:     393190
Free blocks:              5618013
Free inodes:              1784717
First block:              0
Block size:               4096
Fragment size:            4096
Group descriptor size:    64
Reserved GDT blocks:      1024
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8192
Inode blocks per group:   512
Flex block group size:    16
Filesystem created:       Wed Sep  5 22:02:05 2018
```

图 1.4 测试结果

```
组 0：(块 0-32767) 校验值 0xf559 [ITABLE_ZEROED]
  主 超级块位于 0，组描述符位于 1-4
  保留的GDT块位于 5-1028
  块位图位于 1029 (+1029)，校验值 0x9ff24d81
  Inode 位图位于 1045 (+1045)，校验值 0x3bc3d891
  Inode表位于 1061-1572 (+1061)
  4352 个可用 块，8177 个可用inode，2 个目录 ，8176个未使用的inodes
  可用块数： 15750-16018，20497-24575，32764-32767
  可用inode数： 16-8192
```

图 1.5blockgroup0 的信息

从上图可以发现：

①Goup0 所占用的 block 号码是 0 到 32767，校验值为 0xf559，superblock 位于 0 号的 block 块内。

②blockbitmap 与 inodebitmap 则分别在编号为 1029 和 1045 的 block 上。

③inodetable 分布于 1061-1572 的 block 上。

④这个组目前可以用的块有 4352 个（15750-16018，20497-24575,32764-32767），可用的 inode 有 8177 个（16-8192）。

3.与目录树的关系

（1）目录

当我们在 Linux 下的文件系统建立一个目录时，文件系统会分配一个 inode 与至少一个 block 给该目录。其中，inode 记录该目录的相关权限与属性，并可记录分配到的那个 block 号码，而 block 则是记录在这个目录下的文件名与文件名占用的 inode 号码信息。也就是说目录所占用的块内容在记录如下的信息：

Inodenumber	文件名
53735697	anaconda-ks.cfg
53745858	initial-setup-ks.cfg

（2）文件

当我们在 Linux 下的 ext2 建立一个一般文件时，ext2 会分配一个 inode 与相对于该文件大小合适的数量的 block 给该文件。

（3）目录树读取

```
root@xiaoao-virtual-machine:/home/xiaoao# ll -di / /etc /etc/passwd
      2 drwxr-xr-x  24 root root  4096 9月   5 22:08 //
1048577 drwxr-xr-x 126 root root 12288 12月  4 09:59 /etc/
1054775 -rw-r--r--   1 root root  2408 12月  4 09:59 /etc/passwd
```

图 1.6 目录树读取

我们通过图 1.6 中可以看到以下信息：

①/的 inode：透过挂载点的信息找到 inode 为 2 的根目录 inode，且 inode 规范的权限让 xiaoao 可以读取该 block 的内容；

②/的 block：经过上面的步骤获得 block 的号码，然后进入/的 block，找到里面 etc 的 inode 号码（1048577）；

③/etc 的 inode：读取 1048577 号 inode 得知 xiaoao 具有 r 和 x 的权限，因此可以读取 etc/的 block 的内容；

④/etc 的 block：在该文件中找到 passwd 文件的 inode 号码（1054775）；

⑤passwd 的 inode：读取 1054775 号 inode 得知 xiaoao 具有 r 的权限，因此可以读取 passwd 的 block 内容；

⑥passwd 的 block：最后将 block 内容的数据读出来。

（4）文件系统大小与磁盘读取性能

虽然我们的 ext2 在 inode 处已经将该文件所记录的 block 号码做记录，数据可以一次性读取，但是如果文件真的太过分散，还是会发生读取效率低的问题。因为硬盘读取头还是得要在整个文件系统中来回读取。因此，我们可以将整个内的资料全部复制出来，将该文件系统重新格式化，再将资料复制回去即可解决这个问题。

此外，如果文件系统真的太大了，那么当一个文件分别记录在这个文件系统的最前面与最后面的 block 号码中，此时会造成硬盘的机械手臂移动幅度过大，也会造成资料读取性能的下降，而且读取头在搜寻整个文件系统时，也会花费比较多的时间去搜索。因此，分区的规划并不是越大越好，而是真的要针对您的主机用途来进行规划才可以。

4.EXT2/EXT3/EXT4 文件的存取与日志式文件系统的功能

如果我们需要新建一个文件/目录，文件系统又是怎么处理的呢？这里就需要使用 blockbitmap 和 inodebitmap 了。假设我们要新增一个文件，那么文件系统将要做以下事情：

①先确定用户对于欲新增文件的目录是否具有 w 与 x 的权限，若有则继续往下，否则停止；

②根据 inodebitmap 找到没有使用的 inode 号码，并将新文件的权限/属性写入；

③根据 blockbitmap 找到没有使用的 block 号码，并将实际的数据写入 block 中，且更新 inode 的 block 指向数据；

④将刚刚写入的 inode 与 block 数据同步更新 inodebitmap 与 blockbitmap，并更新 superblock 的内容。

一般来说，我们将 inodetable 与 datablock 称为数据存放区域，至于其他例如 superblock、blockbitmap 与 inodebitmap 等区段就被称为 metadata（中间数据）。这是因为 superblock，inodebitmap 及 blockbitmap 的数据是经常变动的，每次新增、移除、编辑时都可能会影响到这三个部分的数据，因此才被称为中间数据。

（1）数据的不一致（Inconsistent）状态

如果文件在写入系统的时候，因突发原因导致出错，所写入数据只有 inodetable 和 datablock，最后一个同步更新 metadata 的过程并没有完成，就会出现 **metadata 内容与实际数据存放区不一致**的情况。

在早期的 EXT2 系统中，系统在重新启动的时候，会有 superblock 中的 vaildbit（是否

有挂载）和文件系统 state（clean 与否）等状态来判断是否强制进行数据一致性检验，若有需要检查可以以 e2fsck 来进行。不过若文件系统极大，这样将非常耗时，日志式文件系统应运而生。

（2）日志式文件系统

所谓日志式文件系统是指，在文件系统中规划处一个区块专门记录写入或修订文件时的步骤，它主要分为 3 个过程：

①预备：当系统要写入一个文件时，会先在日志记录区块中记录某个文件准备要写入的信息；

②实际写入：开始写入文件的权限与数据；开始更新 metadata 的数据；

③结束：完成数据与 metadata 的更新后，在日志记录区块当中完成该文件的记录。

5.Linux 文件系统的操作

为了解决编辑较大文件时，磁盘写入速度要慢于内存速度的问题，Linux 采用的是**异步处理**（asynchronously）方式。

这种方式是：当系统载入一个文件到内存后，如果该文件没有被更动过，则在内存区段的文件数据会被设定为**干净(clean)**的。**但如果内存中的文件数据被更改过了，此时该内存中的数据会被设定为脏的(Dirty)。此时所有的动作都还在内存中执行，并没有写入到磁盘中。**

另外，Linux 系统的文件系统与内存的互动有以下几点：

①系统会不定时的将内存中设定为“Dirty”的数据写回磁盘，以保持磁盘与内存数据的一致性。

②系统会将常用的文件数据放置到主存储器的缓冲区，以加速文件系统的读/写。但这样 Linux 的物理内存最后都会被用光，不过这是正常的情况，它可加速系统效能。

③你可以手动使用 sync 来强迫内存中设定为“Dirty”的文件回写到磁盘中；

④若正常关机时，关机指令会主动呼叫 sync 来将内存的数据回写入磁盘内；

⑤但若不正常关机（如停电、死机或其他不明原因），由于数据尚未回写到磁盘内，因此重新启动后可能会花很多时间在进行磁盘检验，甚至可能导致文件系统的损毁（非磁盘损毁）。

6.挂载点的意义

将文件系统与目录树的结合称为“**挂载**”。挂载点一定是目录，该目录为进入该文件系统的入口。必须挂载到目录树的某个目录后，文件系统才能被正常使用。

7.其它 Linux 支持的文件系统与 VFS

输入“ls -l /lib/modules/$(uname -r)/kernel/fs”，可以得到如图 1.7 所示，Ubuntu 18.04 支持的文件系统。由于支持较多，在这里只截取一部分做演示。

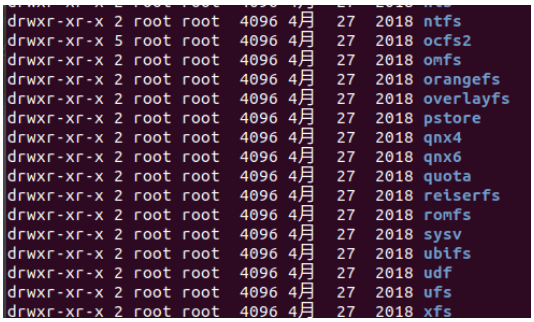

图 1.7 Ubuntu 支持的文件系统

Linux 核心是怎样管理这些认识的文件系统呢？其实 Linux 是通过名为 Virtual 文件系统 Switch 的核心功能去读取文件系统的。

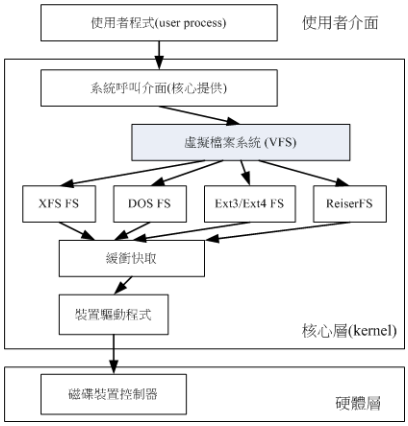

图 1.8 VFS 文件系统的示意图

8.XFS 文件系统

XFS 就是一个**日志式文件系统**。XFS 文件系统在数据分布上，分为**数据区（data section）**、**文件系统活动登录区（log section）**和**实时运行区（realtime section）**。它们的内容如下：

（1）数据区

也包括 inode/data block/superbloc 等数据，也是分为多个存储区群组来分别放置文件系统所需要的数据，每个存储区组包括了：整个文件系统的 superblock、剩余空间的管理机制、inode 分配与追踪。此外，inode 与 block 都是系统用到的时候才动态配置产生。

另外，**与 EXT 不同的是，xfs 的 block 和 inode 有多种不同容量可以设置，block 容量可以在 512bytes~64KB 之间调整（Linux 环境下最高位 4K），inode 有 256bytes 到 2M 之间**。

（2）文件系统活动登录区

记录文件系统的变化，直到该变化完整的写入到数据区后，该次记录才会结束。因为系统所有动作的时候都会在这个区块做记录，因此这个区块的活动相当频繁。xfs 允许用户指定外部磁盘来作为 xfs 文件系统的日志区块。

（3）实时运行区

当有文件要被建立时，xfs 会在这个区段里面找一个到数个的 extent 区块，将文件放置在这个区块内，等到分配完毕后，再写入到 data section 的 inode 与 block 去！这个 extent 区块的大小得要在格式化的时候就先指定，最小值是 4K 最大可到 1G。一般非磁盘阵列的磁盘默认为 64K 容量，而具有类似磁盘阵列的 stripe 情况下，则建议 extent 设定为与 stripe 一样大。这个 extent 最好不要乱动，因为可能会影响到实体磁盘的性能。

xfs 可以使用 xfs_info 去观察 superblock 内容。

二、文件系统的简单操作

1.磁盘与目录的容量：df，du

（1）df

df 是列出文件系统的整体磁盘使用量。

[root@study ~]# df [-ahikHTm] [目录或文件名]
选项与参数：
-a：列出所有的文件系统，包括系统特有的/proc 等文件系统；
-k：以 KBytes 的容量显示各文件系统；
-m：以 MBytes 的容量显示各文件系统；
-h：以人们较易阅读的 GBytes,MBytes,KBytes 等格式自行显示；
-H：以 M=1000K 取代 M=1024K 的进位方式；

-T：连同该 partition 的文件系统名称(例如 xfs)也列出；
-i：不用磁盘容量，而以 inode 的数量来显示。

注意，**/dev/shm/目录是利用内存虚拟出来的硬盘空间，通常是总物理内存的一半**。由于我分配给虚拟机的内存为 2G，所以大小为 986MB。在这个目录下面新建的任何数据文件访问速度非常快，但是缺点是一旦关机，数据就会消失。

（2）du

du 是评估文件系统的的磁盘使用量（常用在推测估计目录所占容量）。

[root@study~]#du [-ahskm] 文件或目录名称
选项与参数：
-a：列出所有的文件与目录容量，因为默认仅统计目录底下的文件量而已。
-h：以人们较易读的容量格式(G/M)显示；
-s：列出总量而已，而不列出每个各别的目录占用容量；
-S：不包括子目录下的总计，与-s 有点差别。
-k：以 KBytes 列出容量显示；
-m：以 MBytes 列出容量显示。

2.硬连接与符号连接：ln

在 Linux 下连接文件有两种，一种是类似于 Windows 快捷方式的文件，可以快速的连接到目标文件（目录）；另一种则是通过文件系统的 inode 连接来产生新文件名，而不是产生新文件。后者称为**硬链接**（hard link）。

（1）硬连接：**在某个目录下新增一个文件名连接到某个 inode 号码的关联记录**。

```
root@xiaoao-virtual-machine:/home/xiaoao# ll -i /etc/crontab
1048716 -rw-r--r-- 1 root root 722 11月 16  2017 /etc/crontab
root@xiaoao-virtual-machine:/home/xiaoao# ln /tec/crontab .
ln: failed to access '/tec/crontab': 没有那个文件或目录
root@xiaoao-virtual-machine:/home/xiaoao# ln /etc/crontab
root@xiaoao-virtual-machine:/home/xiaoao# ln /etc/crontab .
ln: 无法创建硬链接'./crontab': 文件已存在
root@xiaoao-virtual-machine:/home/xiaoao# ll -i /etc/crontab crontab
1048716 -rw-r--r-- 2 root root 722 11月 16  2017 crontab
1048716 -rw-r--r-- 2 root root 722 11月 16  2017 /etc/crontab
```

图 2.1 硬连接

两个文件名都连接到了 1048716 这个 inode 号码，这两个文件名的权限和属性完全一致。再注意一下，root 前面的数字在第一行为 1，第二行为 2，其实意思是**有多少个文件名连接到了这个 inode 号码**。这个过程就如图 2.2 所示的这样一个结构。

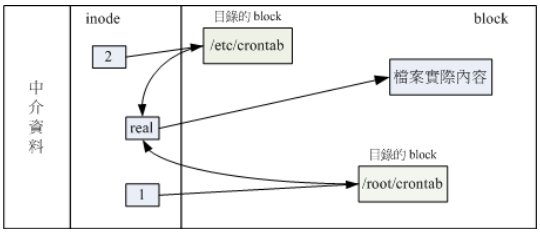

图 2.2 硬连接的文件读取示意图

可以看出，从 1 或 2 的目录的 inode 指定的 block 找到了两个不同的文件名，但是它们都指向了同一个 inode（名为 real 的 inode），然后 inode 再去读取最终数据。这样的好处是非常**安全**，即**如果将任何一个文件名删除，inode 和 block 都还是存在的，依旧可以通过另一个文件名来读取到正确的文件数据；此外，不论用哪个文件名编辑，最终结果都会写入到相同的 inode 和 block 中**。

hard link 有以下限制：

①Hard link 不能跨文件系统；

②Hard link 不能指向目录

（2）Symbolic Link（符号连接）

符号连接亦称软连接（Soft link），它是**建立一个独立的文件，而这个文件会让数据的读取指向其连接的文件的文件名**。当来源文件被删除之后，symbolic link 就会打不开。其读取示意图如图 2.3 所示。

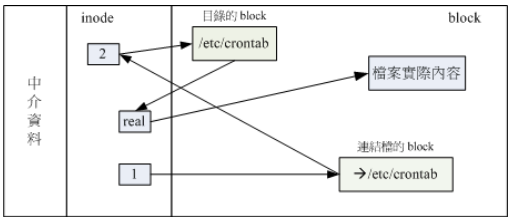

图 2.3 符号连接的文件读取示意图

这里需要注意的是，**Symbolic Link 与 Windows 快捷方式可以划等号，由 Symbolic link 所建立的文件为一个独立的新文件，会占用掉 inode 和 block。**

下面介绍一下 ln 这个指令。

[root@study ~]# ln [-sf] 来源文件 目标文件 选项与参数： -s：如果不加任何参数就进行连结，那就是 hardlink，至于-s 就是 symboliclink -f：如果目标文件存在时，就主动的将目标文件直接移除后再建立！

（3）关于目录的 link 数量

一个空目录里面至少会存在什么呢？存在.与..这两个目录，那么我们新建一个/tmp/testing 的目录的时候，基本会有三个东西，分别是：

/tmp/testing

/temp/testing/.

/temp/testing/..

其中/tmp/testing 和/tmp/testing/,是一样的，而/tmp/testing/..则代表/tmp 目录。所以我们在新建一个目录的时候，**新的目录的 link 数为 2，而上层目录的 link 数则会增加 1**。

三、磁盘的分区、格式化、检查与挂载

如果我们想要在系统里面新增一个磁盘时，应该有哪些动作需要做的呢？

①对磁盘进行分区，以建立可用的 partition；

②对该 partition 进行格式化，以建立系统可用的文件系统；

③（可选）对刚刚建立好的文件系统进行校验；

④在 linux 系统上，需要建立挂载点，并将其挂在上来。

1.查看磁盘分区状态

（1）lsblk 可以看做为“list block device”的缩写。使用方法如下：

[root@study ~]# lsblk [-dfimpt] [device] 选项与参数： -d：仅列出磁盘本身，并不会列出该磁盘的分区数据 -f：同时列出该磁盘内的文件系统名称 -i：使用 ASCII 的线段输出，不要使用复杂的编码(再某些环境下很有用) -m：同时输出该设备在/dev 底下的权限数据(rwx 的数据) -p：列出该设备的完整文件名！而不是仅列出最后的名字而已。

-t：列出该磁盘设备的详细数据，包括磁盘队列机制、预读写的数据量大小等

如图所示 3.1 所示是输入 lsblk 命令后我的硬盘的情况的一部分。图中的各列解释如下：

①NAME：设备名称。会省略/dev 等前导目录。

②MAJ:MIN：主要:次要设备代码。

③RM：是否为可移除设备。如 U 盘等。

④SIZE：容量。

⑤RO：是否为只读设备。

⑥TYPE：是磁盘（disk）、分区（partition，part）、只读存储器（rom）、loop 设备（loop）。

⑦MOUNTPOINT：挂载点。

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0 140.9M  1 loop /snap/gnome-3-26-1604/70
loop1    7:1    0 140.7M  1 loop /snap/gnome-3-26-1604/74
loop2    7:2    0    13M  1 loop /snap/gnome-characters/117
loop3    7:3    0  88.2M  1 loop /snap/core/5897
```

图 3.1 lsblk 情况

（2）blkid：列出设备的 UUID 参数。UUID 是全局单一标识符（universally unique identifier）的意思。

（3）parted：列出磁盘的分区表类型和分区信息。格式如下：

[root@study ~]# parted device_name print

2.磁盘分区：gdisk/fdisk

MBR 分区使用 fdisk，GPT 分区使用 gdisk。partprobe 可以用来更新 Linux 核心的分区表信息。

3.磁盘格式化

格式化，又被称为“制作文件系统（make 文件系统）”，所以这个综合指令名为 mkfs。对于 xfs 来说为 mkfs.xfs，对于 ext4 来说为 mkfs.ext4。

（1）XFS 文件系统：mkfs.xfs

[root@study ~]# mkfs.xfs [-b bsize] [-d parms] [-i parms] [-l parms] [-L label] [-f] \
[-r parms] 设备名称

选项与参数：

关于单位：底下只要谈到“数值”时，没有加单位则为 bytes 值，可以用 k,m,g,t,p(小写)等来解释。比较特殊的是 s 这个单位，它指的是 sector 的个数。

-b：后面接的是 block 容量，可由 512 到 64k，不过最大容量限制为 Linux 的 4k。

-d：后面接的是重要的 data section 的相关参数值，主要的值有：

agcount=数值：设定需要几个储存群组的意思(AG)，通常与 CPU 有关；

agsize=数值：每个 AG 设定为多少容量的意思，通常 agcount/agsize 只选一个设定即可；

file：指的是『格式化的设备是个文件而不是个设备』的意思！(例如虚拟磁盘)；

size=数值：data section 的容量，亦即你可以不将全部的设备容量用完的意思；

su=数值：当有 RAID 时，那个 stripe 数值的意思，与底下的 sw 搭配使用；

sw=数值：当有 RAID 时，用于储存数据的磁盘数量(须扣除备份盘与备用盘)；

sunit=数值：与 su 相当，不过单位使用的是『几个 sector(512bytes 大小)』的意思；

swidth=数值：就是 su*sw 的数值，但是以『几个 sector(512bytes 大小)』来设定。

-f：如果设备内已经有文件系统，则需要使用这个-f 来强制格式化才行。

-i：与 inode 有较相关的设定，主要的设定值有：

size=数值：最小是 256bytes 最大是 2k，一般保留 256 就足够使用了；

internal=[0|1]：log 设备是否为内建？默认为 1 内建，如果要用外部设备，使用底下设定；
logdev=device：log 设备为后面接的那个设备上头的意思，需设定 internal=0 才可；
size=数值：指定这块登录区的容量，通常最小得要有 512 个 block，大约 2M 以上才行。
-L：后面接这个文件系统的标头名称 Labelname 的意思。
-r：指定 realtimesection 的相关设定值，常见的有：
extsize=数值：就是那个重要的 extent 数值，一般无需设定，但有 RAID 时，最好设定与 swidth 的数值相同较佳！最小为 4K 最大为 1G。

（2）EXT4 文件系统 mkfs.ext4

[root@study ~]# mkfs.ext4 [-b size] [-L label] 设备名称
选项与参数：
-b：设定 block 的大小，有 1K、2K、4K 的容量。
-L：后面接这个装置的标头名称。

4.文件系统检查

（1）xfs_repair：当有 xfs 文件系统错乱的时候用这个指令。

[root@study ~]# xfs_repair [-fnd] 设备名称
选项与参数：
-f ：后面的设备其实是个文件而不是硬件。
-n ：单纯检查并不修改文件系统的任何数据 (检查而已)
-d ：通常用在单人维护模式底下，针对根目录 (/) 进行检查与修复的动作！很危险！不要随便使用。

xfs_repair 可以检查/修复文件系统，不过，因为修复文件系统是个很庞大的任务！因此，修复时该文件系统不能被挂载。Linux 系统有个装置无法被卸除，这就是根目录啊。如果你的根目录有问题怎办？这时得要进入单人维护或救援模式，然后通过-d 这个选项来处理。加入-d 这个选项后，系统会强制检验该装置，检验完毕后就会自动重新启动。

（2）fsck.ext4：fsck 是一个综合指令，针对 ext4 的话，使用 fsck.ext4 检测比较好。

[root@study ~]# fsck.ext4 [-pf] [-b superblock] 设备名称
选项与参数：
-p：当文件系统在修复时，若有需要回复 y 的动作时，自动回复 y 来继续进行修复动作。
-f：强制检查。一般来说，如果 fsck 没有发现任何 unclean 的标志，不会主动进入检查的，如果您想要强制 fsck 进入检查，就得加上-f 标志。
-D：针对文件系统下的目录进行优化配置。
-b：后面接 superblock 的位置！一般来说这个选项用不到。但是如果你的 superblock 因故损坏时，通过这个参数即可利用文件系统内备份的 superblock 来尝试救援。一般来说，superblock 备份在：1Kblock 放在 8193,2Kblock 放在 16384,4Kblock 放在 32768。

无论是 xfs_repair 或 fsck.ext4，这都是用来检查与修正文件系统错误的指令。注意：**只有身为 root 且你的文件系统有问题的时候才使用这个指令**，否则在正常状况下使用此指令可能会造成对系统的危害。

另外，如果你怀疑刚刚格式化成功的磁盘有问题的时后，也可以使用 xfs_repair/fsck.ext4 来检查磁盘。此外，由于 xfs_repair/fsck.ext4 在扫瞄磁盘的时候，可能会造成部分文件系统的修订，所以『执行 xfs_repair/fsck.ext4 时，被检查的 partition 务必**不能挂载到系统**上，也就是说需要在**卸除**的状态。

5.文件系统的挂载与卸载

挂载前的注意事项：

①单一文件系统不应该被重复挂载在不同的挂载点(目录)中；

②单一目录不应该重复挂载多个文件系统；

③要作为挂载点的目录，理论上应该都是空目录才是。当然，若没有使用空目录，那么挂载后该目录中的文件会被暂时隐藏，当卸载的时候文件就会再次显示出来。

（1）挂载使用 mount，下面简单做一些介绍。

```
[root@study ~]# mount -a
[root@study ~]# mount [-l]
[root@study ~]# mount [-t 文件系统] LABEL="  挂载点
[root@study ~]# mount [-t 文件系统] UUID="   挂载点（推荐）
[root@study ~]# mount [-t 文件系统] 设备文件名  挂载点
选项与参数：
-a：依照配置文件/etc/fstab 的数据将所有未挂载的磁盘都挂载上来。
-l：单纯的输入 mount 会显示目前挂载的信息。加上-l 可增列 Label 名称。
-t：可以加上文件系统种类来指定欲挂载的类型。常见的 Linux 支持类型有：xfs，ext3，
ext4，reiserfs，vfat，iso9660(光盘格式)，nfs，cifs，smbfs(后三种为网络文件系统类型)。
-n：默认情况下，系统会将实际挂载情况实时写入/etc/mtab，助于其他程序运行。但在某
些情况下(例如单人维护模式)为了避免问题会刻意不写入。此时就得要使用-n 选项。
-o：后面可以接一些挂载时额外加上的参数！比方说账号、密码、读写权限等：
async，sync:此文件系统是否使用同步写入(sync)或异步(async)的内存机制，预设为 async；
atime，noatime:是否修订文件的读取时间(atime)。为了效能，某些时刻可使用 noatime
ro，rw:挂载文件系统成为只读(ro)或可擦写(rw)；
auto，noauto:允许此文件系统被以 mount-a 自动挂载(auto)；
dev，nodev:是否允许此文件系统上，可建立装置文件，dev 为可允许；
suid，nosuid:是否允许此文件系统含有 suid/sgid 的文件格式；
exec，noexec:是否允许此文件系统上拥有可执行 binary 文件；
user，nouser:是否允许此文件系统让任何使用者执行 mount？一般来说，mount 仅有 root
可以进行，但下达 user 参数，则可让一般 user 也能够对此 partition 进行 mount；
defaults:默认值为：rw，suid，dev，exec，auto，nouser，andasync。
remount:重新挂载，这在系统出错，或重新更新参数时，很有用。
```

Linux 十分聪明，我们不需要加上-t 选项，系统会自动分析最恰当的文件系统来尝试挂载你的设备。Linux 是利用分析文件系统 superblock 搭配 linux 自己的驱动程序来测试挂载，如果成功了，就立刻自动使用该类型的文件系统挂载起来。系统是参考下面两个文件进行挂载：

①/etc/文件系统 s：系统指定的测试挂载文件系统类型的优先级；

②/proc/文件系统 s：Linux 系统已经加载的文件系统类型。

注意：

①光驱挂载成功以后就无法退出光盘，除非将其卸载才可以退出。

②如果参数要改变或者根目录出现“只读”状态，根目录需要重新挂载，可以通过使用“mount -o remount, rw, auto”来进行。

（2）卸载挂载使用 umount。

```
[root@study ~]# umount [-fn] 设备文件名称或挂载点
选项与参数：
-f：强制卸载。可用在类似网络文件系统(NFS)无法读取到的情况下。
-l：立刻卸载文件系统，比-f 还强。
-n：不更新/etc/mtab 情况下卸载。
```

对于有其它挂载的项目，一定要用挂载点来写出才可以。

6.磁盘/文件系统参数修订

Linux 下所有设备都以文件来代表，通过文件的 major 于 minor 数值来代替，这两个数值是由特殊意义的。

磁盘文件名	Major	Minor
/dev/sda	8	0-15
/dev/sdb	8	16-31
/dev/loop0	7	0
/dev/loop1	7	1

手动处理设备文件使用 mknod。

```
[root@study ~]# mknod  设备文件名  [bcp] [Major] [Minor]
选项与参数：
装置种类：
b：设定装置名称成为一个外部存储设备文件，例如磁盘等。
c：设定装置名称成为一个外部输入设备文件，例如鼠标/键盘等。
p：设定装置名称成为一个 FIFO 文件。
Major：主要装置代码。
Minor：次要装置代码。
```

xfs_admin 可以修改 XFS 文件系统的 UUID 和 Label name。

```
[root@study ~]# xfs_admin [-lu] [-L label] [-U uuid]  设备文件名
选项与参数：
-l：列出这个设备的 labelname；
-u：列出这个设备的 UUID；
-L：设定这个设备的 Labelname；
-U：设定这个设备的 UUID。
```

tune2fs 可以修改 ext4 的 label name 和 UUID。

```
[root@study ~]# tune2fs [-l] [-L Label] [-U uuid]  设备文件名
选项与参数：
-l：类似 dumpe2fs-h 的功能，将 superblock 内的数据读出来；
-L：修改 LABEL name；
-U：修改 UUID。
```

四、设定开机挂载

系统挂载的一些限制：

①根目录/是必须挂载的，而且一定要先于其它 mount point 被挂载进来。

②其它 mount point 必须为已建立的目录，可任意指定，但一定要遵守必须的系统目录架构原则(FHS)

③所有 mount point 在同一时间之内，只能挂载一次。

④所有 partition 在同一时间之内，只能挂载一次。

⑤如若进行卸除，您必须先将工作目录移到 mountpoint(及其子目录)之外。

1.开机挂载/etc/fstab 及/etc/mtab

我们查看一下/etc/fstab 这个文件的内容，如图 4.1 所示。

```
root@xiaoao-virtual-machine:/home/xiaoao# cat /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=cbc8bed7-0124-4e65-a103-2e3e34fc2f11 /               ext4    errors=remount-ro 0       1
/swapfile                                 none            swap    sw                0       0
```

图 4.1 /etc/fstab 的内容

磁盘被手动挂载之后都必须把挂载信息写入/etc/fstab 这个文件中，否则下次开机启动时仍然需要重新挂载。

系统开机时会主动读取/etc/fstab 这个文件中的内容，根据文件里面的配置挂载磁盘。这样我们只需要将磁盘的挂载信息写入这个文件中我们就不需要每次开机启动之后手动进行挂载了。

除此之外，/etc/fstab 还加入了 dump 这个备份用的命令支持，与开机时是否进行文件系统检查 fsck 等指令有关。

在鸟哥的书里面，这个文件的六列内容分别是：

[设备/UUID 等] [挂载点] [文件系统] [文件系统参数] [dump] [fsck]

但是在我的 Ubuntu 18.04 中，六列内容分别是：

[文件系统] [挂载点] [类型] [选项] [dump] [fsck]

下面继续以鸟哥的内容来讲。

（1）第一列：磁盘设备文件名/UUID/LABEL 名称：

这个字段可以填写的数据主要有三个项目：

①文件系统或磁盘的装置文件名，如 /dev/vda2 等；

②文件系统的 UUID 名称，如 UUID=xxx；

③文件系统的 LABEL 名称，例如 LABEL=xxx。

（2）第二列：挂载点。

（3）文件系统：如 xfs/ext4/vaf/nfs 等。

（4）文件系统参数

文件系统参数主要有以下几点

参数	内容意义
async/sync 异步/同步	设定磁盘是否以异步方式运作。预设为 async(效能较佳)
auto/noauto 自动/非自动	当下达 mount -a 时，此文件系统是否会被主动测试挂载。预设为 auto。
rw/ro 可擦写/只读	让该分区以可擦写或者是只读的型态挂载上来，如果你想要分享的数据是不给用户随意变更的，这里也能够设定为只读。
exec/noexec 可执行/不可执行	限制在此文件系统内是否可以进行执行操作。如果纯粹用来储存数据，那么可以设定为 noexec 会比较安全。不过，这个参数也不能随便使用，因为你不知道该目录下是否默认会有可执行程序。

user/nouser （不）允许用户挂载	是否允许用户使用 mount 指令来挂载呢？一般而言，我们当然不希望一般身份的 user 能使用 mount。
suid/nosuid （不）具有 suid 权限	该文件系统是否允许 SUID 的存在？如果不是执行文件放置目录，也可以设定为 nosuid 来取消这个功能。
defaults	同时具有 **rw,suid,dev,exec,auto,nouser,async** 等参数。基本上，预设情况使用 defaults 设定即可

（5）能否被 dump 备份指令作用。一般默认输入 0。

（6）是否可以以 fsck 检查分区。

/etc/fstab 是开机时的配置文件，不过，实际文件系统的挂载是记录到/etc/mtab 与/proc/mounts 这两个文件当中的。每次我们在更动文件系统的挂载时，也会同时更动这两个文件。但是，万一发生你在/etc/fstab 输入的数据错误，导致无法顺利开机成功，而进入单人维护模式当中，那时候的/可是 readonly 的状态，当然你就无法修改/etc/fstab，也无法更新/etc/mtab，那怎么办？

答案是使用

mount -n -o remount,rw /

2.特殊设备 loop 挂载（镜像文件不刻录就挂载使用）

（1）挂载光盘/DVD 镜像文件

下面以鸟哥将 CentOS 7.x 的 DVD 镜像文件挂到测试机上面的命令来做显示。

```
[root@study ~]# mkdir /data/centos_dvd
[root@study ~]# mount -o loop /tmp/CentOS-7.0-1406-x86_64-DVD.iso /data/centos_dvd
[root@study ~]# df /data/centos_dvd
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/loop0        4050860 4050860         0 100% /data/centos_dvd
[root@study ~]# umount /data/centos_dvd/
# 测试完成，卸载镜像文件
```

（2）新建大文件以制作 loop 设备文件

既然能够挂载 DVD 的镜像文件，那么我能不能制作出一个大文件，然后将这个文件格式化后挂载呢？好问题!这是个有趣的事情,能够帮助我们解决很多系统的分割不好的情况。

①建立大文件

假设我要在/srv/loopdev 建立一个空的文件：

```
[root@study ~]# dd if=/dev/zero of=/srv/loopdev bs=1M count=512
#这个指令的简单意义如下：
#if 是 input file，输入文件。那个/dev/zero 是会一直输出 0 的设备!
#of 是 output file，将一堆零写入到后面接的文件中。
#bs 是每个 block 大小，就像文件系统那样的 block 意义；
#count 则是总共几个 bs 的意思。所以 bs*count 就是这个文件的容量了!
```

②大型文件格式化

预设 xfs 不能格式化文件，所以要加入特殊的参数。

```
[root@study ~]# mkfs.xfs -f /srv/loopdev
[root@study ~]# blkid /srv/loopdev
```

③挂载

```
[root@study ~]# mount -o loop UUID="7dd97bd2-4446-48fd-9d23-a8b03ffdd5ee" /mnt
[root@study ~]# df /mnt
```

五、内存交换空间（swap）的构建

1.使用物理分区构建 swap

（1）分区：使用 gdisk 在硬盘分出一个分区给系统作 swap。**由于 Linux 的 gdisk 预设会将分区的 ID 设定为 Linux 的文件系统，所以你可能还得要设定一下 system ID**。

（2）格式化：利用新建的 swap 格式的“**mkswap 设备文件名**”就可以格式化该分区称为 swap 格式。

（3）使用：最后将该 swap 设备启动，方法为：**swapon 设备文件名**。

（4）观查看：最后通过 free 与 swapon -s 来观察下内存使用情况。

执行命令如下：

2.使用文件构建 swap（以建立一个 128MB 的 swap 空间为例）

（1）使用 dd 命令新增一个 128MB 的文件，在/tmp 下面：

```
[root@study ~]# dd if=/dev/zero of=/tmp/swap bs=1M count=128
```

（2）使用 mkswap 将/tmp/swap 格式化为 swap 的文件格式：

```
[root@study ~]# mkswap /tmp/swap
```

（3）使用 swapon 来讲/tmp/swap 启动

```
[root@study ~]# swapon /tmp/swap
```

（4）使用 swapoff 关掉 swap file，并设定自动启动

①设定自动启动：

```
[root@study ~]# nano /etc/fstab /tmp/swap   swap   swap   defaults   0   0
```

②关闭 swap file

```
[root@study ~]# swapoff /tmp/swap /dev/vda6
```

六、文件系统的特殊查看与操作

1.磁盘空间的浪费问题

我们在前面提到过，一个 block 只能放置一个文件，因此太多小文件将会浪费非常多的磁盘容量。另外，整个文件系统中包括 superblock，inode table 以及其他 metadata 等其实都会浪费磁盘容量，所以我们建立 xfs/ext4 文件系统的时候，一挂载就有很多容量被用到了。

2.利用 GNU 的 parted 进行分区

虽然你可以使用 gdisk/fdisk 很快速的将你的分区分隔妥当，不过 gdisk 主要针对 GPT 而 fdisk 主要支持 MBR，对 GPT 的支持还不够。所以使用不同的分区格式时，得要先查询到正确的分区表才能用适合的指令。那有没有同时支持的指令呢？答案是肯定的，那就是使用 parted 来进行分区。

parted 常用语法如下所示：

```
[root@study ~]# parted [设备] [指令  [参数]]
选项与参数：
指令功能：
新增分区：mkpart [primary|logical|extended] [ext4|vfat|xfs]  开始  结束
显示分区：print
删除分区：rm [partition]
```