Sec07 Linux 磁盘与文件系统管理

认识 Linux 文件系统

磁盘组成与分区

详见 Sec02

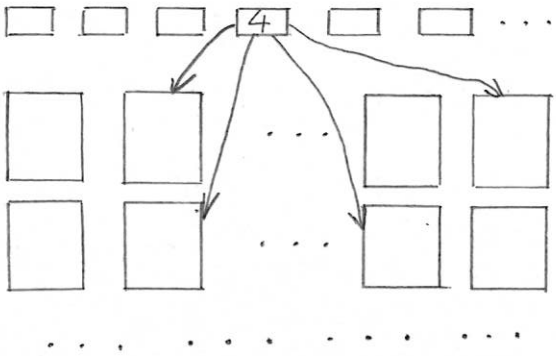
文件系统特性

磁盘分区完毕后需要进行格式化（format），之后操作系统才能够使用这个文件系统（filesystem）。

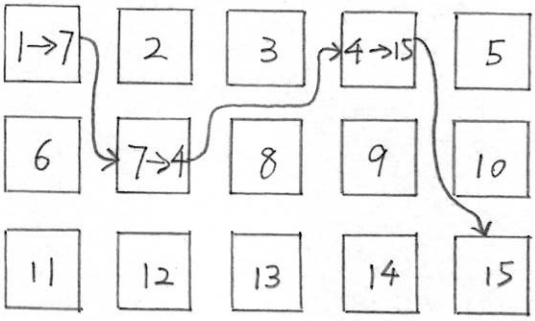
每种操作系统能够使用的文件系统并不相同。在默认的情况下，windows是不认识 linux的 Ext2 的。

文件数据 {
- 文件属性（权限、拥有者、群组、时间等） ⟶ inode：一个文件占用一个 inode，同时记录此文件的数据所在的 block 号码。
- 实际数据 ⟶ data block：若文件太大，会占用多个。

超级区块（superblock）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，以及文件系统的格式与相关信息等。



索引式（indexed allocation）文件系统



FAT 文件系统

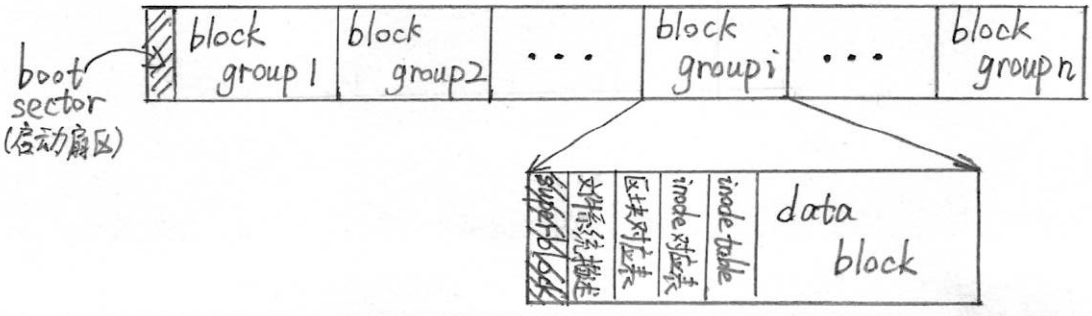
注：Ext2 是索引式文件系统。

FAT 的文件系统需要经常地"碎片整理"：将同一个文件的 blocks 汇整在一起。

Linux 的 Ext2 文件系统（inode）

"如果某个文件系统高达数百GB，那么将所有的 inode 与 block 通通放在一起是很不明智的。"

区块群组（block group）



文件系统最前面有一个启动扇区（boot sector），这个启动扇区可以安装开机管理程序，
如此设计可以使不同的开机管理程序安装到个别的文件系统最前端，而不用覆盖整个磁盘唯一的MBR，
这样便能够制作出多重引导的环境。

- data block

  放置文件内容数据；每个block都有编号，以方便inode的记录。
  除非重新格式化，block的大小与数量在格式化完成后就不能再改变了。
  每个block只属于一个文件，如果文件大于一个block的大小，则该文件会占用多个block，
  如果文件小于一个block的大小，则该block的剩余容量就不能再用了（浪费）。

- inode table

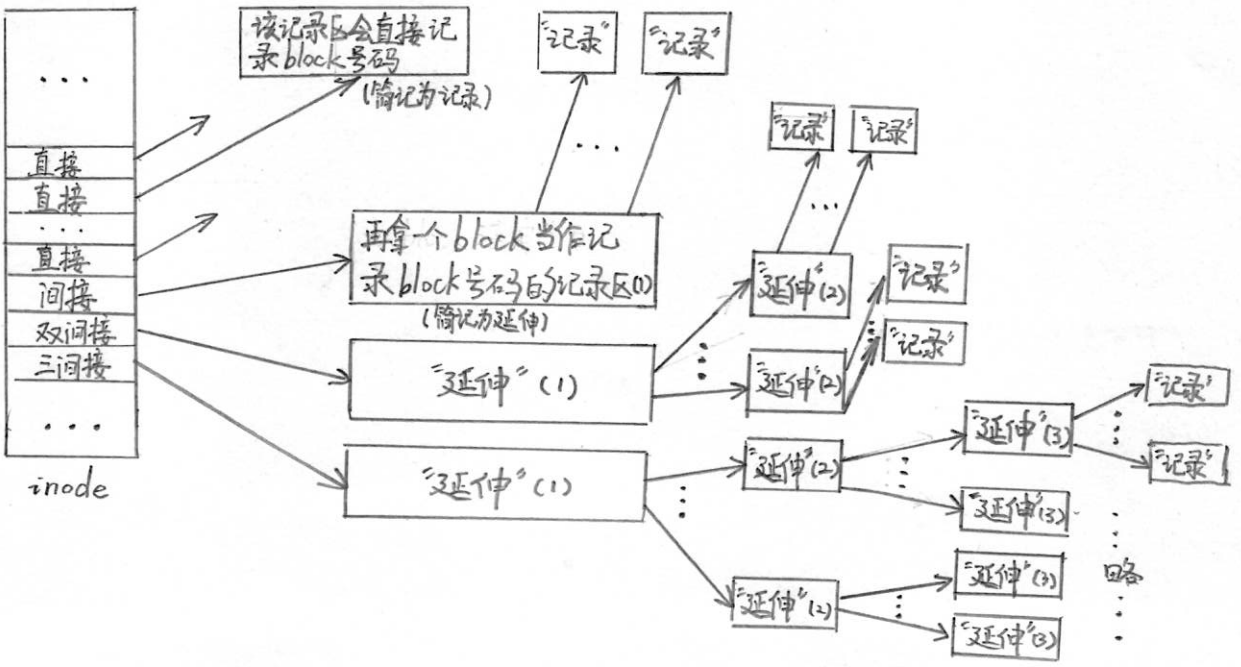
  inode：
  - 该文件的权限（read/write/execute）
  - 该文件的拥有者与群组（owner/group）
  - 该文件的容量
  - 该文件建立或状态改变的时间（ctime）
  - 最近一次的读取时间（atime）
  - 最近修改的时间（mtime）
  - 定义文件特性的flag，如SetUID等
  - 该文件实际数据的指向（pointer），即实际数据位于哪几号block内

  inode的数量与大小在格式化时就已经固定了。
  每个inode的大小均固定。
  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。
  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，只有符合才
  会开始实际地读取block的内容。

问题：

由于inode的大小是固定的，而inode每记录一个block号码都要花掉一点空间，
所以文件的大小是被inode大小与block号码限制地很死？

NO! inode记录block号码的区域有12个直接，1个间接，1个双间接与1个三间接记录区。



所以文件大小的上限是直接、间接、双间接、三间接的总额。

- superblock
  记录整个 filesystem 相关信息的地方。
  - block与inode的总量
  - 未使用与已使用的 inode/block 数量
  - block与inode的大小
  - filesystem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)的时间等文件系统相关信息
  - 一个 valid bit 数值，若此文件系统已被挂载，则 valid bit 为0，反之为1

  不是每个 block group 都有 superblock，一般只有第一个 block group 有。 dumpe2fs 指令
- 文件系统描述
  - 每个 block group 的开始与结束的 block 号码
  - 每个区段（superblock、bitmap、inodemap、data block）分别介于哪一个 block 号码之间
- block bitmap（区块对照表）
  记录哪些 block 是空的、哪些不是。（记录的是 block 的号码）
- inode bitmap（inode对照表）
  记录使用与未使用的 inode 号码。
- dumpe2fs
  查询 Ext 家族 superblock 信息的指令。

与目录树的关系

目录与文件在文件系统中是如何被记录的呢？

when 建立一个目录，
  文件系统会分配一个 inode 与至少一块 block 给该目录，
  其中 inode 记录该目录的相关权限与属性，并记录被分配到的那块 block 的号码，
  而 block 则是记录在这个目录下的文件名与该文件名占用的 inode 号码数据。
  (文件或目录) (文件或目录)

when 建立一个文件，
  文件系统会分配一个 inode 与 文件大小的 block 数目给该文件。

由上，inode 本身并不记录文件名，文件名的记录是在目录的 block 当中。

(∴ 新建、删除、更名文件名与目录的 w 权限有关)

例：读取 /etc/passwd

① / 的 inode，得到 / 的 block 号码
② 由①获得 / 的 block 号码，读取 / 的 block，并找到 etc/ 目录的 inode 号码
③ 由②，进入 etc/ 的 inode，获得 etc/ 的 block 号码
④ 读取 etc/ 的 block，并找到 passwd 文件的 inode 号码
⑤ 由④，读取 passwd 的 inode，获取 passwd 的 block 号码
⑥ 读取 passwd 的 block 的内容

EXT2/EXT3/EXT4 文件的存取与日志式文件系统的功能

新增一个文件时，此时文件系统的行为是：

① 确定用户对于欲新增文件的目录是否具有 w 与 x 的权限，有的话才能新增

② 根据 inode bitmap 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 block bitmap 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

④ 将刚刚写入的 inode 与 block 数据同步更新到 inode bitmap 与 block bitmap 中，并更新 superblock 的内

inode table, data block } 数据存放区域

superblock, block bitmap, inode bitmap } metadata

隐患：数据的不一致 (inconsistent)

①②③ × 还：metadata 的内容与实际数据存放区不一致！

⇒ 日志式文件系统 (journaling filesystem)

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，在日志记录区块中完成对该文件的记录。

在数据的记录过程中万一发生了什么问题，那么系统只要去检查日志记录区块，就可以知道哪个文件发生了问题。

ext3/ext4 这个文件系统就可以！(在 superblock 下)

Linux 文件系统的运作

效率问题：在编辑一个大文件时，要频繁地从内存写入磁盘中，而 $V_{disk} \ll V_{mem}$。

⇒ 异步处理 (asynchronously)

系统加载一个文件到内存后，

如果该文件没有被动过，

则在内存区段的文件数据会被设为 clean，

如果内存中的文件被更改了，

则该内存中的数据会被设为 dirty。

系统会不定时地将内存中的 dirty 数据写回磁盘，以保持磁盘与内存数据的一致性。(也可以通过 sync 手动

挂载点的意义 (mount point)

一个文件系统要能够链接到目录树才能被我们使用，将文件系统与目录树结合的动作称为"挂载"。

注：挂载点一定是目录，该目录为进入该文件系统的入口。

文件系统的简单操作

(容量、连结档 linkfile)

磁盘与目录的容量

文件系统的整体数据是在 superblock 区块中，每个文件的容量则是在(被记录) inode 当中。

df [-ahikHTm] <目录或文件名>　　#列出文件系统的整体磁盘使用量

[-a] 列出所有的文件系统，包括系统特有的 /proc 等文件系统 (比较特殊的文件系统几乎都在内存中，不占磁盘空间)

[-k] 以 KB 为单位显示各文件系统

[-m] 以 MB 为单位显示各文件系统

[-h] 以 human 易读的 GB、MB、KB 显示

[-i] 不以磁盘容量，而以 inode 的数量来显示

[-T] 连同该 partition 的 filesystem 名称 (如 xfs) 列出

du [-ahikHTm] <目录或文件名>

[-a] 列出所有的文件与目录大小，因为默认只统计目录底下的文件而已

[-h] 以 human 易读的格式显示

[-s] 仅显示目录占了多少磁盘

[-S] 不包括子目录下的总计

硬链接与软链接

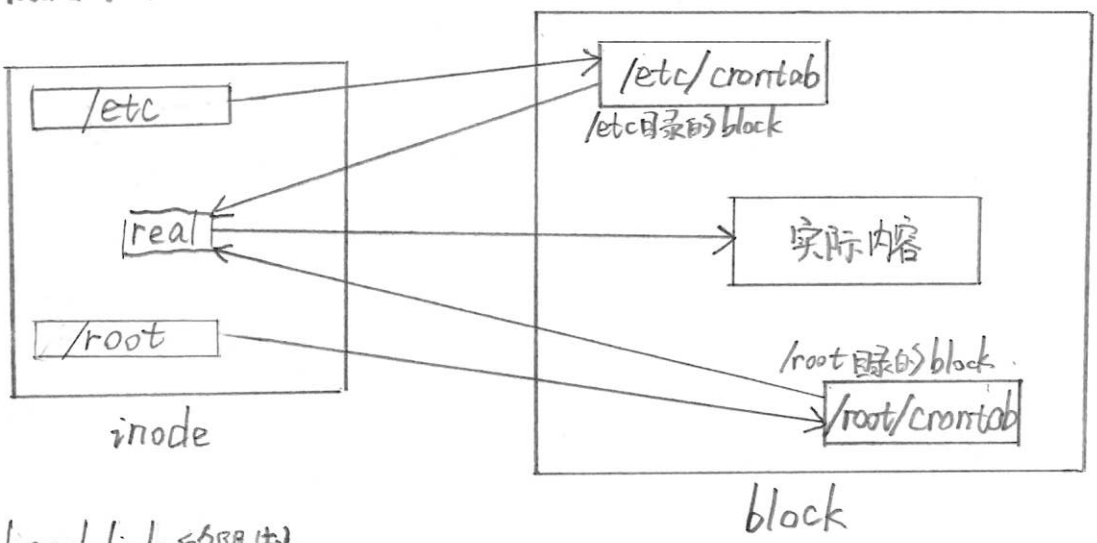
∵ 每个文件会占用一个 inode、文件内容由 inode 来指向

(当读取文件时，必须要经过目录记录的文件名来找到正确的 inode 号码才能读取)

∴ 文件名只与目录有关、文件内容只与 inode 有关

因此，可能有多个文件名对应到同一个 inode 号码

hard link 只是在某个目录下新增一个文件名并链接到某 inode 号码的关连记录而已。



hard link 的限制

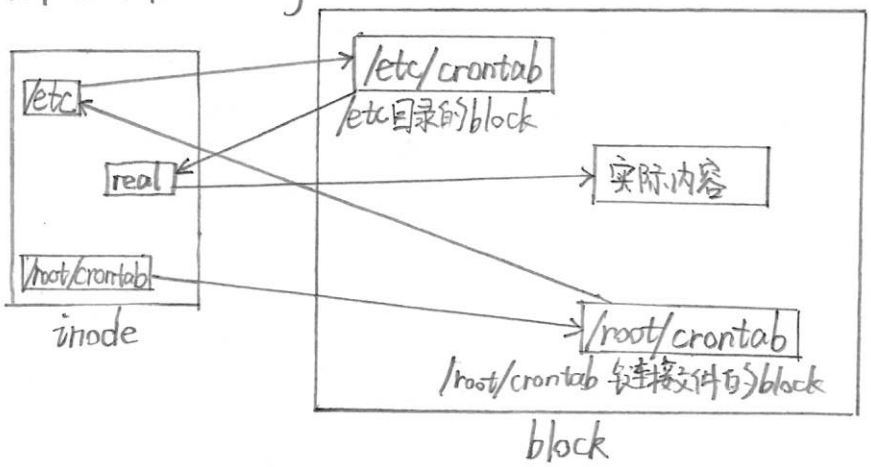
① 不能跨 filesystem (inode 不能指向 filesystem 之外的 block)

② 不能 link 目录 (否则得对被链接目录下的所有数据都建立链接，复杂度太大)

symbolic link 符号链接、软链接、快捷方式

就是建立一个独立的文件，而这个文件会让数据的读取指向它链接的那个文件的文件名。

当来源档被删除之后，symbolic link 会"失效"。



ln [-sf] <来源文件><目标文件>

[-s] 有了此选项，就是建立 symbolic link，否则就是建立 hard link

[-f] 如果<目标文件>存在时，就进行覆盖

hard link 一般情况下不会增加 inode、也不会耗用 block，毕竟只是在某个目录的 block 中写入数据，但如果这笔记录刚好将目录的 block 填满、则会进行"再次映射"，导致磁盘空间的变化。

symbolic link 会建立一个新文件，所以要占用 inode 与 block，此文件大小为 <来源文件> 的文件名长度。

磁盘的分区、格式化、检验与挂载

如果系统管理者要在系统里新增一个磁盘，

① 对磁盘进行分区，以建立可用的 partition（磁盘分区槽） ← "partition多大？"

② 对该 partition 进行格式化（format），以建立系统可用的 filesystem ← "是否加入journal功能？" "inode与block如何规划？"

③ 在 Linux 系统上建立挂载点（即目录），并将文件系统挂载上去

观察磁盘分区的状态

lsblk := list block device 列出系统上的所有磁盘列表

blkid := block id 列出装置的 UUID 等参数（universally unique identifier）

parted 列出磁盘的分区表类型与分区信息

磁盘分区：gdisk、fdisk

MBR 分区 → fdisk

GPT 分区 → gdisk

磁盘格式化（建立文件系统）

mkfs := make filesystem

文件系统检验

xfs_repair

fsck.ext4

文件系统的挂载与卸除

挂载点的意义：挂载点是目录，而这个目录是进入磁盘分区槽（其实是文件系统）的入口。

注：

同一个文件系统不应被重复挂载在不同的挂载点（目录）中

同一个目录不应挂载多个文件系统

要作为挂载点的目录，理论上应该都是空目录

若用来挂载的目录不是空的，那么挂载了文件系统之后，原目录下的东西会暂时地消失。

```
mount -a
mount [-l]      l是long的意思，用于显示目前挂载的信息
mount [-t <文件系统的信息>] LABEL='...' <挂载点>
                            UUID='...'
                            <装置文件名>
```

[-n] 不写入 /etc/mtab

umount [-fn] <装置文件名或挂载点>

[-f] 强制卸除，可用在 NFS 无法读取到的情况下等

[-n] 不更新 /etc/mtab

磁盘或文件系统的参数修订

略

设定开机挂载

开机挂载 /etc/fstab 及 /etc/mtab

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

③ 所有 mount point 只能挂载一次

④ 所有 partition 只能挂载一次

⑤ 在卸除之前，必须将工作目录移到 mount point 之外

/etc/fstab

filesystem table

[装置/UUID等] [挂载点] [文件系统类型] [文件系统参数] [dump] [fsck]

"/dev/vda2"	xfs	rw/ro	是否以 fsck 检验扇区
"UUID=..."	ext4	async/sync	
"LABEL=..."	vfat	defaults	
	nfs		

特殊装置 loop 挂载（映象文件不刻录就挂载使用）

内存置换空间（swap）的建立

CPU 能读取的数据都来自于内存，当内存不足时，为了让后续的程序可以顺利地运行，因此在内存中暂不使用的程序与数据就会被挪到 swap 中。（swap 是磁盘用来暂时放置内存的数据的）

使用实体分区槽建立 swap

建立一个虚拟内存的文件

文件系统的特殊观察与操作

磁盘空间的浪费问题

一个 block 只能放一个文件，因此太多的小文件会浪费非常多的磁盘容量。

```
root@vsa11779357:~# ll -sh
total 680K
8.0K -rw------- ... 6.4K ...
4.0K ...        ... 861  ...
4.0K ...        ... 1.1K ...
648K ...        ... 645K ...
...  ...        ...  ...  ...
```

占用的磁盘空间的大小　　实际的大小

利用 GNU 的 parted 进行分区行为

略