黄伟亮：ext4文件系统之裸数据的分析实践

原创：黄伟亮 Linux阅码场 2017-07-27

作者简介: 黄伟亮(Huang weller),毕业于苏州大学,就职于苏州博世汽车部件汽车多媒体事业部,从事汽车多媒体娱乐系统的平台开发工作六年有余, 接触Linux 系统近10年。感兴趣的方向有Linux系统性能优化,多媒体框架, 文件系统和存储器件, USB以及虚拟化等。

欢迎给Linuxer投稿(Linuxer只接受Linux方面的原创文章)，获赠三大社任意在售图书一本，详情点击：在Linuxer上把一个问题说清或者看懂有惊喜
感谢人民邮电异步社区对活动的大力支持！

开篇

笔者一直认为，文件系统就是构建在块设备上的数据库，文件名是检索数据库中数据的一种手段。原理永远是简单的，就像火箭上天只要有足够大的反向推力装置就好了，就像空调制冷就是利用汽化吸热的原理一样。但实现是复杂的.

笔者十分偏爱ext4这个文件系统，原因之一是一直和这个文件系统打交道，曾经一年多的时间没做别的事情，工作内容全是它。另一个原因，则是它是linux世界使用最广泛的文件系统，google的GFS也是构建在ext4之上的。而且ext4著名的maintainer Ted 也在google就职, 社区的maintainer回答问题也很积极和友好. 不过,因为是偏爱,其实不需要任何理由.

回顾一下笔者当年的学习过程，简单概括起来就是，有表入里，由浅入深。过程也是充满艰辛. 话说信息时代, 网络有千文， 但看别人的文章， 总不如自己实践一番来的体会深刻.

但是篇幅和精力所限,本文不做涉及文件系统技术细节的具体分析,比如jbd2在文件系统中的具体作用和分析等,本文 只是提供一个分析文件系统的实践过程.

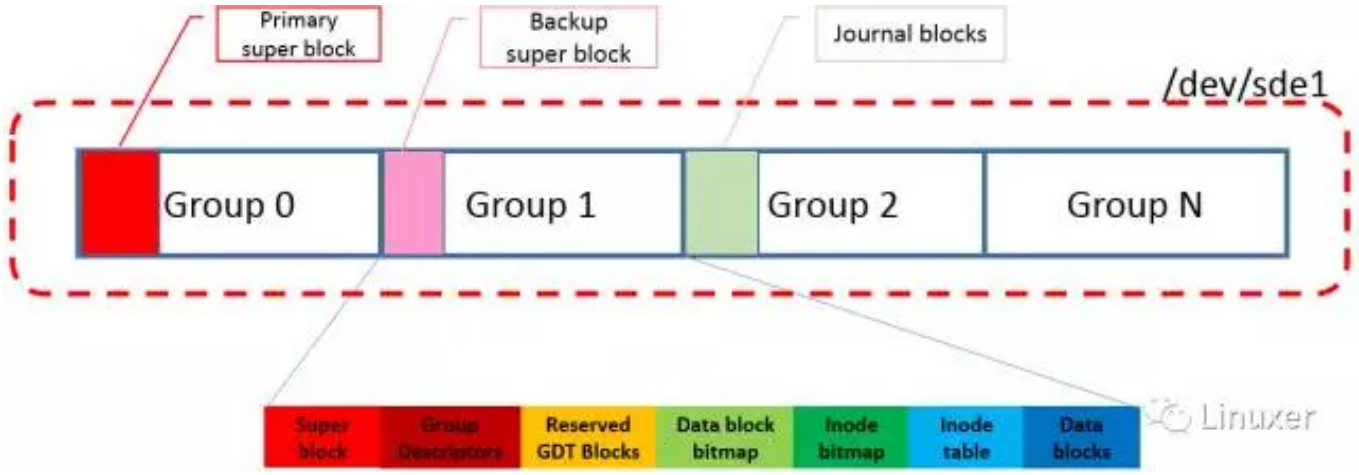
那么， 就拿张SD卡follow me吧.

Ext4 layout

Ext4的layout的详细内容这里就略过了，童鞋们可以到这个页面去了解:

https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Inline_Data

一定要看过以后再来看这篇拙文哦！ 如果那个链接实在看不下去， 那就看下面这张图吧:



在 mkfs.ext4命令格式化完块设备以后，ext4的layout 在块设备上大致如上图所描述， 包含了一下信息:

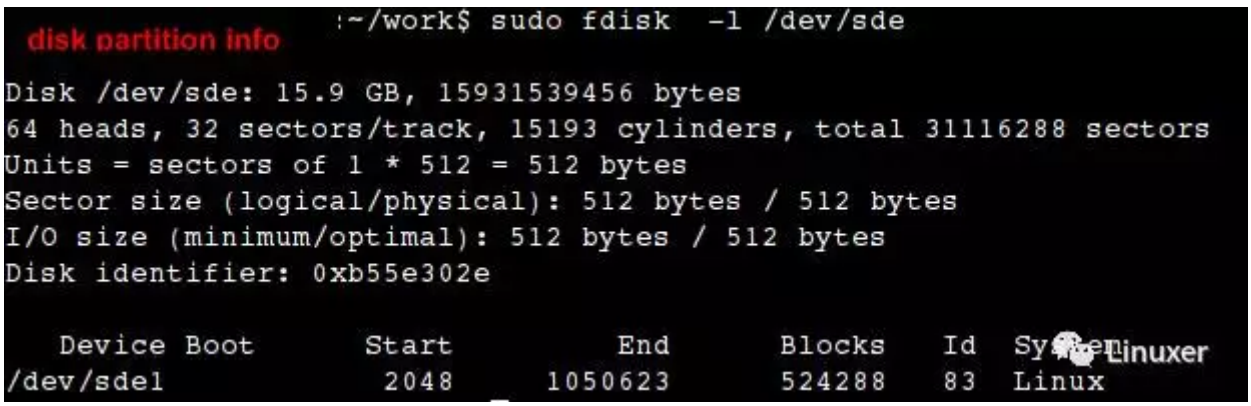
- *块设备被按group 来划分,每个group有对应的group descriptor*
- *Super block在Group 0中， 非group 0中的super block是backup super block*
- *Journal block的位置并不是在最后一个group.（本例中,journal block在group 2）*

定位文件系统超级块superblock

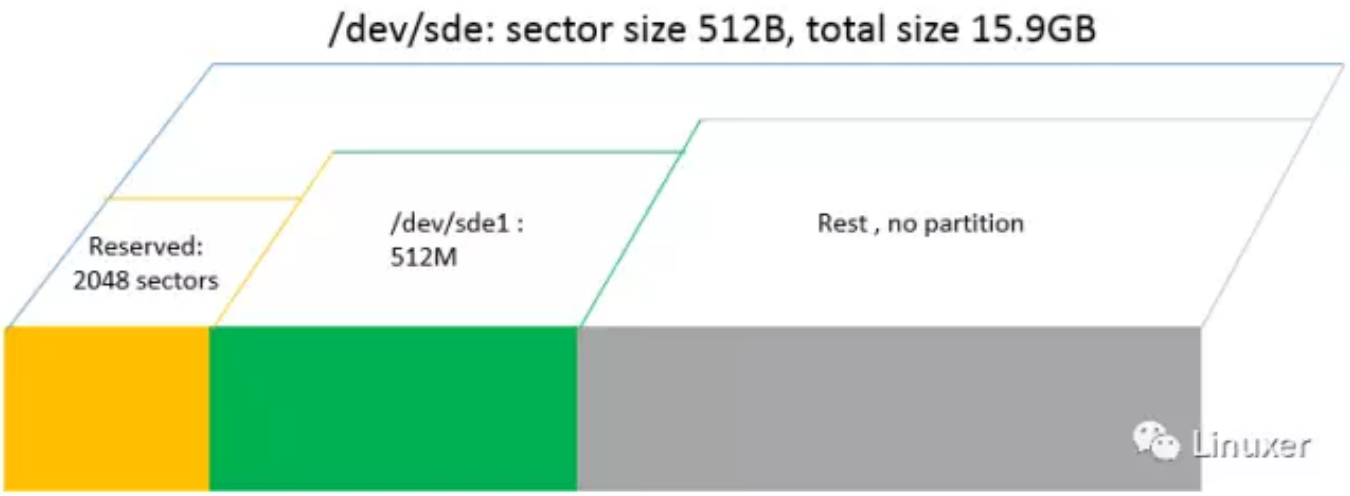
首先, 如果磁盘是DOS的分区格式, 那就用fdisk命令查看一下磁盘的分区信息. 从磁盘的分区信息我们可以得到以下内容:

- 整个磁盘有多大
- 磁盘被分成了多少个分区
- 每个分区的大小和起始结束的位置

下图给出了一个例子, 后面的文件系统分析也是基于该磁盘: /dev/sde1.



下面这张图清晰的描述了测试所用的SD卡的分区信息, 对块设备/dev/sde来说, 最小的单元为sector, sector size通常为512, SD卡的大小可以通过sector来描述.

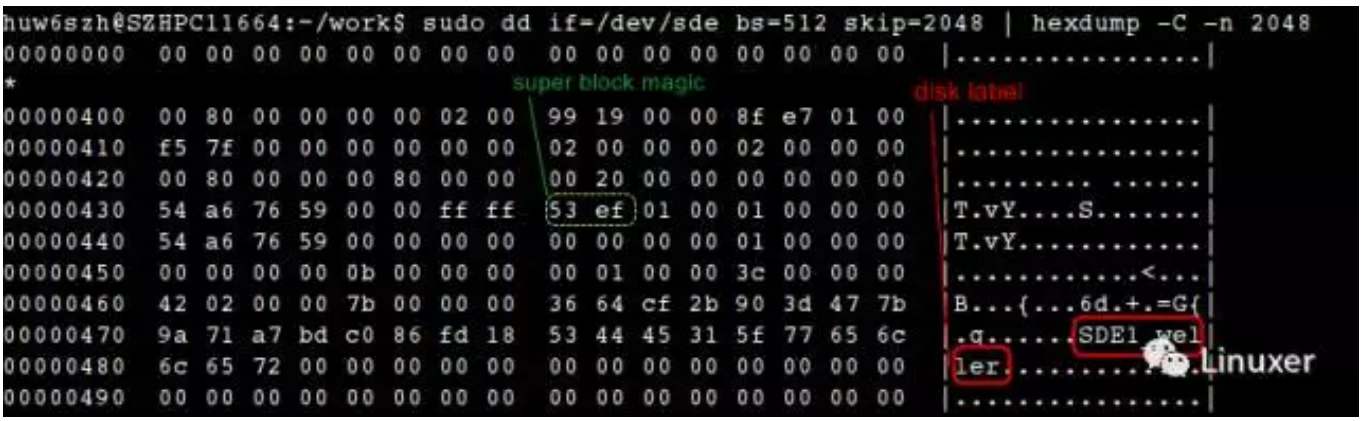


那么，接下来就要开始找ext4的super block了． ext4的superblock包含一个magic number : __le16 s_magic = 0xEF53, 我们可以用过它来确认superblock.

通过Linux的常用命令dd + hexdump来查看想要查看的block device 任意位置的内容.
如下图所示, dd 命令:

```
sudo dd if=/dev/sde bs=512 skip=2048 | hexdump -C -n 2048
```

skip到块设备 sde1的offset 2048 x 512B(这里的512即sector size)位置， hexdump出后面2KB的内容． 在0x438位置，找到了ext4的super block magic word， 同时， 也看到了在使用mkfs.ext4 时， -L参数指定的disk label: "SDE1_weller"



定位journal block 和 inode table

Ext4离开jbd2能活吗？可以！但会变得非常不可靠． 关于jbd2和文件系统的关系， 会在以后的文章中再做详述. 不过我们要知道journaling block device是通用的journaling layer, 为其他文件系统提供journaling服务.

因为文件的写操作和挂载过程和jbd2紧密相关， 所以在分析文件系统问题的时候， 我们通常需要查询journal block里的内容做分析, 因此请跟随下面的内容来定位journal block.

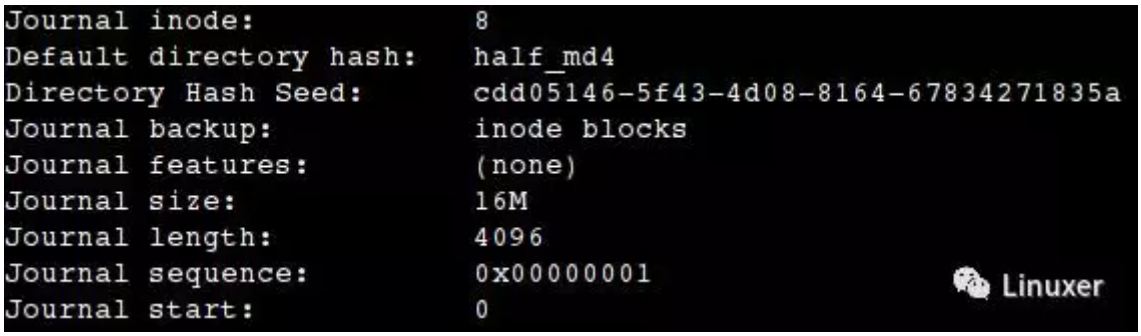
首先使用dumpe2fs来查看sde1块设备上的ext4文件系统的信息, 这也是本文唯一使用的ext4原生工具.

如下图所示可知如下信息：

- *journal的位置信息存储在inode table 的 offset 8 上(inode index = 8)*
- *journal的整个大小是16MB*
- *journal的长度则是和文件系统的block size相同， 也是4KB.*
- *Inode size 是256B*

命令：

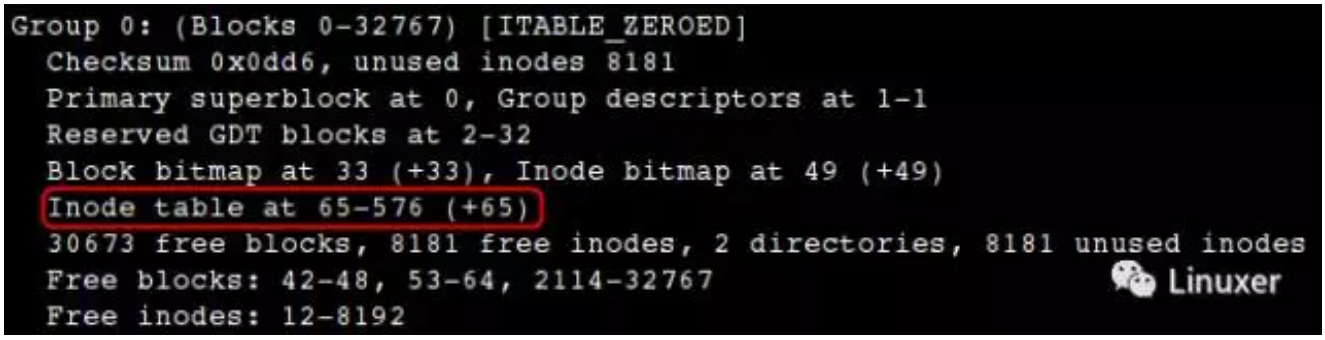
```
sudo dumpe2fs /dev/sde1
```



那么，问题来了，为了找到journal的起始位置，我们不得不先去找inode index 8所在的位置. 其实，用hexdump直接search journal的magic word也可以找到journal block的起始位置，但是这是笔者初学时的方法，现在想来实在太low.

定位inode table

回到`sudo dumpe2fs /dev/sde1`这条命令的输出结果上来，在Group 0的描述中，Inode table的起始block number是65.



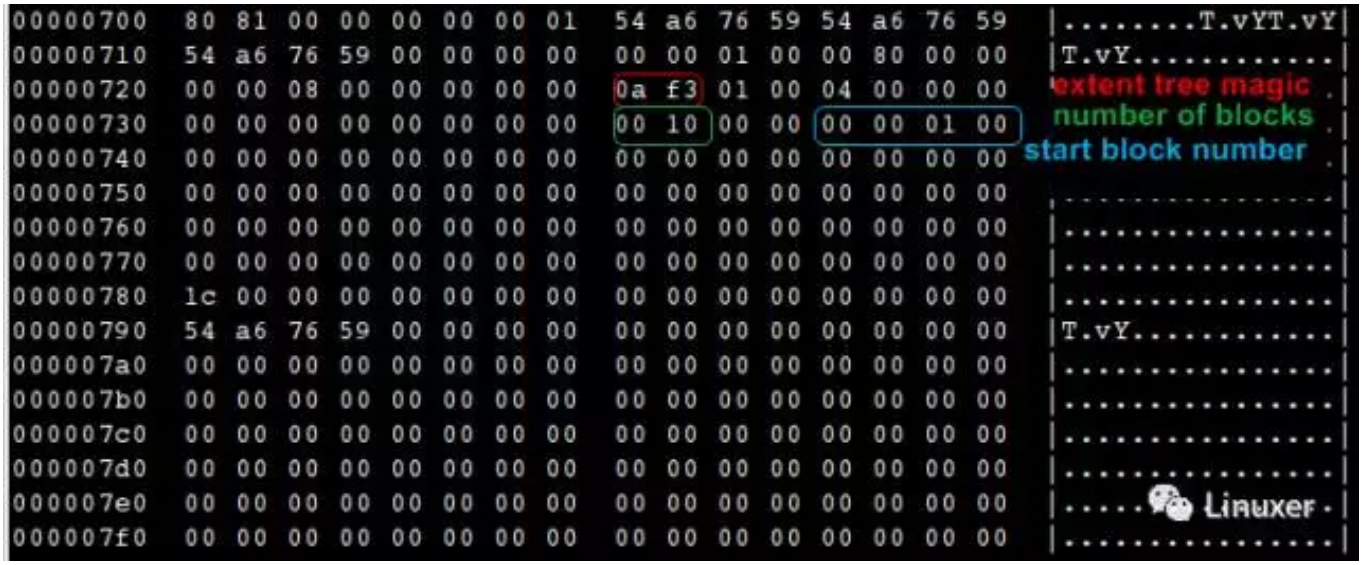
已知该文件系统的block size是 4KB, Inode size 是256B.

那么inode index 8 在inode table中的 offset 为 (8-1) x 256B = 0x700

使用如下命令dump出来，截取0x700开始的256B内容：

命令:

```
sudo dd if=/dev/sde1 bs=4096 skip=65 | hexdump -Cv -n 2048
```



由此可知，journal block 的起始位置是0x00010000 x 4KB，也就是block number 为0x10000的block，大小为0x1000 x 4KB(16MB).

如何判断结果上述的结果准确呢?

Journal block对数据格式也是有定义的，判断依据是，/dev/sde1设备的0x10000 block offset处的数据内容应当是journal的super block，block type为4意为 Journal superblock v2.

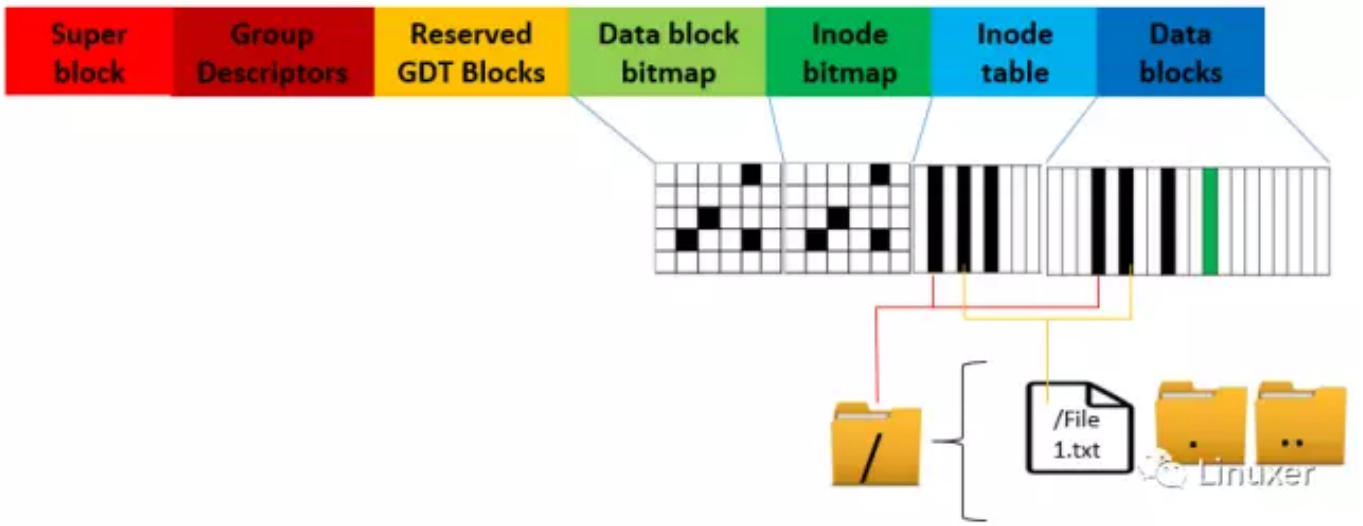
来一窥真容:

命令:

```
sudo dd if=/dev/sde1 bs=4096 skip=65536 | hexdump -Cv -n 2048
```

定位文件内容位置

要找到一个文件内容在磁盘上的位置，我们需要先了解一下文件在ext4上的建立过程. Data block bitmap和inode bitmap想像记账员一样，记录着data block和inode table的使用情况. Inode table用来描述文件内容所在的block number 和number of block.



首先我们在块设备sde1 mount的目录下创建一个文件file1.txt. 然后用 ls -i 命令获得file1.txt的inode index.

下例中，file1.txt的inode index为12.

```
SDE1_weller$ ls -i file1.txt
12 file1.txt
```

inode 12 在inode table中的 offset为 (12-1) x256B = 0xB00. 用下面的命令dump inode table并且截取0xB00位置的256B字节:

命令:

```
sudo dd if=/dev/sde1 bs=4096 skip=65 | hexdump -Cv -n 4096
```

```
00000b00  b4 81 5a 0c 2c 00 00 00  e5 df 76 59 e4 df 76 59  |..Z.,.....vY..vY|
00000b10  e4 df 76 59 00 00 00 00  5a 0c 01 00 08 00 00 00  |..vY....Z.......|
00000b20  00 00 08 00 01 00 00 00  0a f3 01 00 04 00 00 00  |................|
00000b30  00 00 00 00 00 00 00 00  01 00 00 00 21 80 00 00  |............!...|
00000b40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000b50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000b60  00 00 00 00 51 79 5f c5  00 00 00 00 00 00 00 00  |....Qy_.........|
00000b70  00 00 00 00 00 00 00 00  c3 60 c3 60 00 00 00 00  |.........`.`....|
00000b80  1c 00 00 00 5c c0 0d 93  5c c0 0d 93 80 16 c8 be  |....\...\.......|
00000b90  e4 df 76 59 5c c0 0d 93  00 00 00 00 00 00 00 00  |..vY\...........|
00000ba0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000bb0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000bc0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000bd0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000be0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....Linuxer.|
00000bf0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

由图可知，文件的数据块在0x00008021的地方，我们dump出来看看吧，猜猜我在文件里写了哪些内容呢?

命令:

```
sudo dd if=/dev/sde1 bs=4096 skip=32801 | hexdump -Cv -n 4096
```

```
                -/work$ sudo dd if=/dev/sde1 bs=4096 skip=32801 | hexdump -Cv -n 4096
00000000  63 72 65 61 74 65 20 61  20 66 69 6c 65 20 61 74  |create a file at|
00000010  20 2f 6d 65 64 69 61 2f  68 75 77 36 73 7a 68 2f  | /media/l      /|
00000020  53 44 45 31 5f 77 65 6c  6c 65 72 0a 00 00 00 00  |SDE1_weller.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```


定位文件名位置

不知道你会不会好奇,这个文件是在根目录下， 那么文件名又是存在什么位置的呢 ？ ext4是怎么知道文件夹它包含哪些文件的呢?

首先根目录也是目录,目录在文件系统里面的表示和文件的表示是一样一样的.

同样可以找到根目录的inode index： `ls -i . -a`

我们可以看到,当前目录， 也就是"."的inode index为 2.

```
     2 .   262771 ..        12 file1.txt      11 lost+found
```

Inode index 2 内容(命令忽略了,可以思考一下命令是怎样的):

```
00000100  ff 41 00 00 00 10 00 00  e5 df 76 59 e4 df 76 59  |.A........vY..vY|
00000110  e4 df 76 59 00 00 00 00  00 00 03 00 08 00 00 00  |..vY............|
00000120  00 00 08 00 01 00 00 00  0a f3 01 00 04 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  01 00 00 00 25 00 00 00  |............%...|
```


根据inode table描述， 根目录的数据块在block offset 0x25， 长度为1个block.

让我们一起看看block number 为0x25的内容吧， 我们可以看到".", "..", lost+found 和file1.txt 等文件名. 至于这一块的数据格式定义， 请童鞋自行学习吧.

命令:

```
sudo dd if=/dev/sde1 bs=4096 skip=37 | hexdump -Cv -n 4096
```

```
                        sudo dd if=/dev/sde1 bs=4096 skip=37 | hexdump -Cv -n 4096
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  d4 0f 09 01 66 69 6c 65  31 2e 74 78 74 00 00 00  |....file1.txt...|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```


添加softlink后的文件系统分析

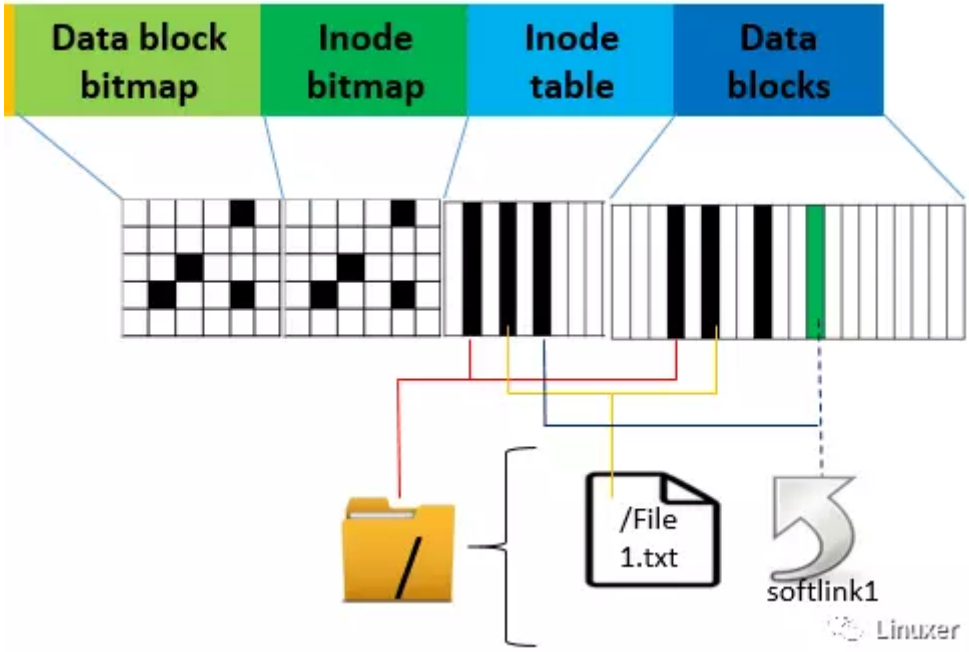
在测试目录下添加一个软链接文件,指向文件file1.txt

```
9 Jul 25 15:03 softlink1 -> file1.txt
```

用之前的方法找到softlink1这个链接文件的inode.

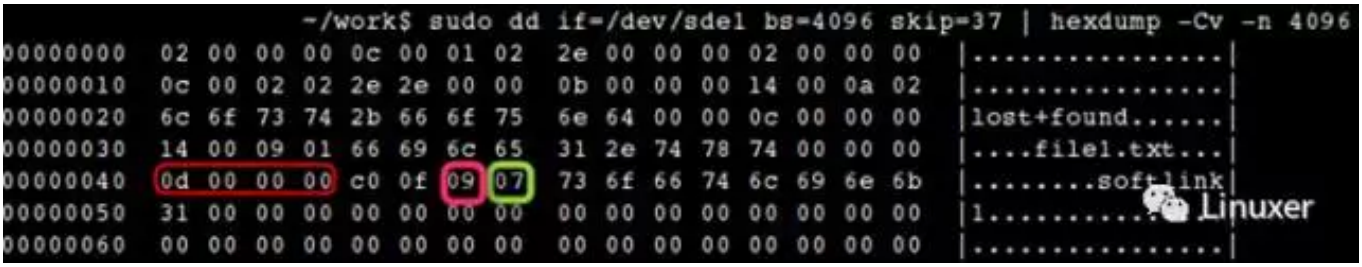
```
                                /SDE1_weller$ ls -i softlink1
13 softlink1
```

软连接， 普通文件和文件夹一样, 都通过inode table来描述:

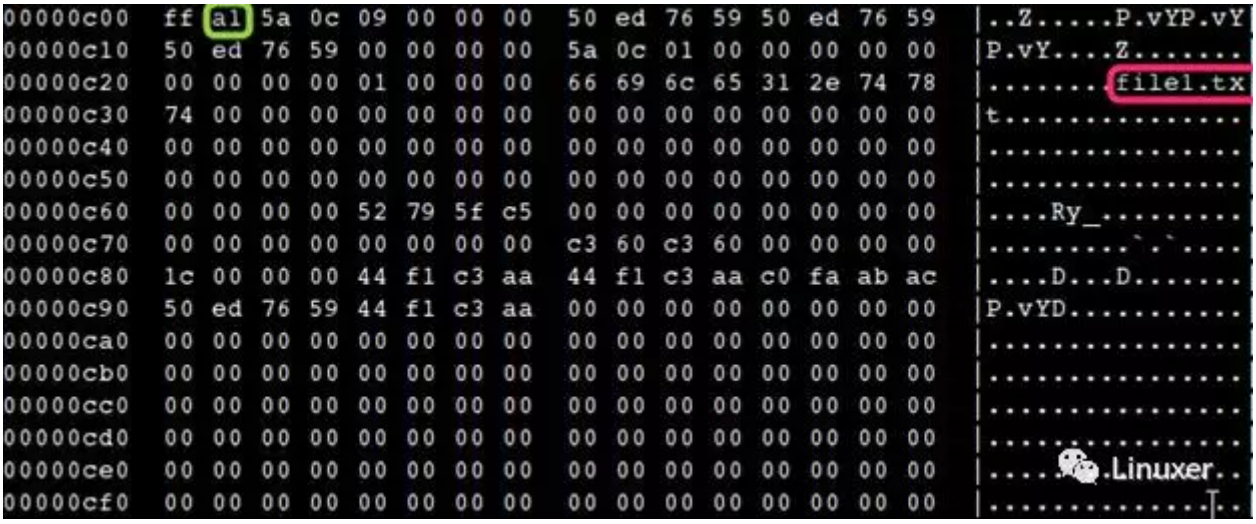


通过上面章节相同的方法dump 根目录的inode指向的数据块，我们可以看到一个新的文件softlink1已经被添加. 下图中, 对于softlink1这个文件的描述是这样的:

- 0x0000000d 意为这个文件的inode index.
- 0x09意为文件名字的长度
- 0x07意为文件的类型是soft link.



接着，我们来dump inode index 0x0d上的文件内容:



0xa1ff中的0xa意为这个文件是一个softlink类型的文件(详情请见inode数据格式定义)，对于这种类型的文件，ext4的处理方式是这样的:

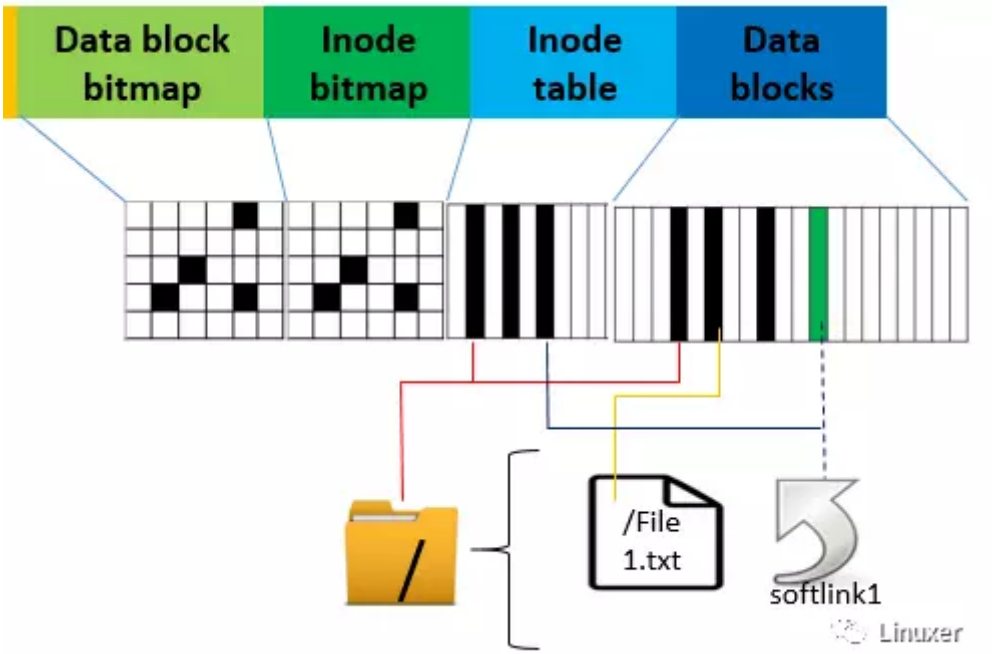
> The target of a symbolic link will be stored in this field if the target string is less than 60 bytes long. Otherwise, either extents or block maps will be used to allocate data blocks to store the link target.

意思就是，如果目标文件名的大小< 60B，则存放在inode的i_block的数据结构中，否则就分配一个extents 去存放长度更长的目标文件名.

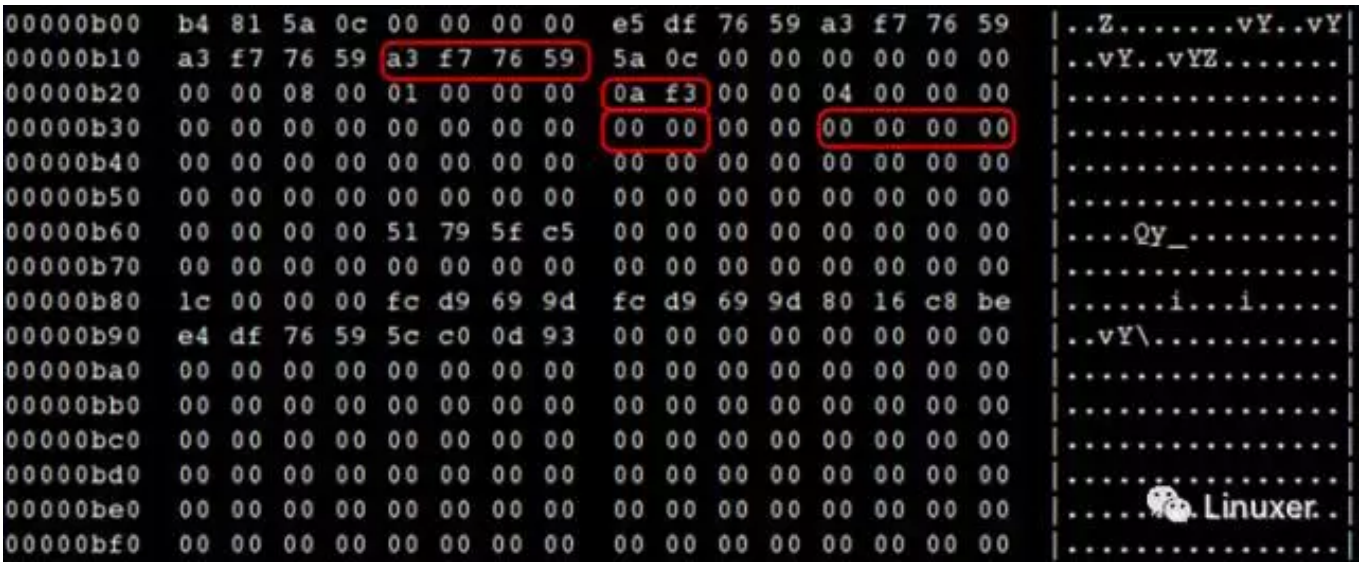
删除文件后的文件系统分析

文件删除操作,作为文件创建的反向操作，大致的原理是找到文件的inode，修改文件的inode，释放inode(free inode number)和data block.

在将跟目录下的文件file1.txt删除后，我们分别dump一下被删除文件的inode内容，被删除文件的文件内容，根目录的extent内容.



删除文件的inode: extent的block number 和number of block变为 0. 删除时间由原先的0变更为0x5976f7a3(epoch format)，也就是标记这个inode被delete了，它没有指向任何数据，解除了和原先文件file1.txt的关系(如上图).



文件内容则依然存在,只是这个原先的extent占用的块已经被释放了:

```
                ~/work$ sudo dd if=/dev/sde1 bs=4096 skip=32801 | hexdump -Cv -n 400
00000000  63 72 65 61 74 65 20 61  20 66 69 6c 65 20 61 74  |create a file at|
00000010  20 2f 6d 65 64 69 61 2f  68 75 77 36 73 7a 68 2f  | /media/huw6szh/|
00000020  53 44 45 31 5f 77 65 6c  6c 65 72 0a 00 00 00 00  |SDE1_weller.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

根目录的extent内容没有发生变化:

```
                ~/work$ sudo dd if=/dev/sde1 bs=4096 skip=37 | hexdump -Cv -n 400
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 28 00 0a 02  |............(...|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  14 00 09 01 66 69 6c 65  31 2e 74 78 74 00 00 00  |....file1.txt...|
00000040  0d 00 00 00 c0 0f 09 07  73 6f 66 74 6c 69 6e 6b  |........softlink|
00000050  31 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |1...............|
```


也就是说, 删除文件其实就是操作了对应文件的inode table.

终了

本文使用了dd,hexdump, dumpe2fs, ls四个命令完成了对ext4文件系统的简单的分析实践, 之所以说简单, 是因为本文只包含了对文件系统基本操作的分析实践, 但是触类旁通, 希望这篇文章能够给大家带来一定的帮助, 同时, 也希望大牛能够给本文指出不足, 小弟先谢谢大家了.

另外, 笔者之所以没有使用debugfs工具来分析ext4是为了让大家更直观的了解分析实践的过程.

>>> Linuxer往期精彩回顾

设备树：《深入探究Linux的设备树》的直播改期到8月14日

航天二院Linux讲座的一些手绘的图

Linux硬实时和Preempt-RT补丁(中断、软中断、调度、内存与调试)

丁增贤: glibc堆探秘系列之fastbin ——上集

丁增贤: glibc堆探秘系列之fastbin ——下集

陈然： 容器生态系统的发展与演变之我见

《Linux总线、设备、驱动模型》直播PPT分享

徐西宁： 码农小马与Docker不得不说的故事

让天堂的归天堂，让尘土的归尘土——谈Linux的总线、设备、驱动模型

...