# Linux文件系统的演进

------------------------------------------------------------------

renyl 2015/5/20

目录

[1 ext2文件系统 3](#_Toc379963970)

[1.1 文件系统的布局 3](#_Toc379963971)

[1.1.1 启动块（Boot Block） 3](#_Toc379963972)

[1.1.2 超级块（Super Block） 3](#_Toc379963973)

[1.1.3 块组描述符表（GDT，Group Descriptor Table） 4](#_Toc379963974)

[1.1.4 块位图（Block Bitmap） 5](#_Toc379963975)

[1.1.5 索引节点位图（inode Bitmap） 5](#_Toc379963976)

[1.1.6 索引节点表（inode Table） 6](#_Toc379963977)

[1.1.7 数据块（Data Block） 7](#_Toc379963978)

[1.2 数据块寻址 7](#_Toc379963979)

[1.2.1 直接寻址与间接寻址 8](#_Toc379963980)

[1.2.2 数据块寻址能力 9](#_Toc379963981)

[1.2.3 数据块寻址过程 9](#_Toc379963982)

[1.3 实例剖析 9](#_Toc379963983)

[1.3.1 查看文件系统的布局 9](#_Toc379963984)

[1.3.2 恢复已删除的文件 13](#_Toc379963985)

[2 ext3文件系统 19](#_Toc379963986)

[2.1 产生的背景 19](#_Toc379963987)

[2.2 日志文件系统 19](#_Toc379963988)

[2.2.1 日志文件系统的定义 19](#_Toc379963989)

[2.2.2 日志文件系统的原理 19](#_Toc379963990)

[2.3 日志的分类 20](#_Toc379963991)

[2.4 ext3与ext2的关系 20](#_Toc379963992)

[2.5 文件恢复 23](#_Toc379963993)

[3 ext4文件系统 27](#_Toc379963994)

[3.1 ext4产生的背景 27](#_Toc379963995)

[3.2 ext4的新特性 27](#_Toc379963996)

[3.2.1 更大的文件系统 27](#_Toc379963997)

[3.2.2 元组块 29](#_Toc379963998)

[3.2.3 延迟分配 30](#_Toc379963999)

[3.2.4 更多的子目录 31](#_Toc379964000)

[3.2.5 日志checksum 31](#_Toc379964001)

[3.2.6 更快的文件系统检查 31](#_Toc379964002)

[3.2.7 纳秒级时间戳 32](#_Toc379964003)

[3.2.8 extent 32](#_Toc379964004)

[3.3 ext4与ext3的关系 35](#_Toc379964005)

[3.4 文件恢复 36](#_Toc379964006)

[4 XFS文件系统 40](#_Toc379964007)

[5 Btrfs文件系统 40](#_Toc379964008)

# 1 ext2文件系统

## 1.1 文件系统的布局

我们知道，一个磁盘可以划分成多个分区，每个分区必须先用格式化工具（如mkfs命令）格式化成某种格式的文件系统，然后才能存储文件，格式化的过程中会在磁盘上写一些管理存储布局的信息。下图是一个磁盘分区格式化成ext2文件系统后的存储布局。



### 1.1.1 启动块（Boot Block）

启动块（Boot Block）是用来存储磁盘分区信息和启动信息，大小是固定的，为1KB，任何文件系统都不能使用启动块。启动块之后才是ext2文件系统的开始，ext2文件系统将整个分区划成若干个同样大小的块组（Block Group）。

### 1.1.2 超级块（Super Block）

1. 超级块描述整个分区的文件系统信息，如inode数、块数量、块大小、文件系统版本号、上次mount的时间等等。
2. 超级块在每个块组的开头都有一份拷贝，默认情况下，只有第一个块组的超级块会被系统内核使用，其他块组的超级块可以在e2fsck之类的程序对磁盘上的文件系统进行一致性检查时使用，这样当第一个块组的开头意外损坏时就可以使用其他拷贝来恢复，从而减少数据丢失。
3. 文件系统中存储的最小单位是块（Block），一个块究竟多大是在格式化时确定的（例如mke2fs的-b选项可以设定块大小为1024、2048或4096字节）。超级块占用1个块大小。
4. 在ext2文件系统中，超级块通过一个名为ext2\_super\_block的结构进行描述，如下所示（引用自Kernel的/root/include/linux/ext2\_fs.h）：

|  |
| --- |
| 338 /\*  若一个块大小为4KB，则文件系统的最大值为2^32 \* 4KB=16TB  339 \* Structure of the super block  340 \*/  341 struct [ext2\_super\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=ext2_super_block&type=reference" \o "Multiple refered from 19 places.) {  342 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_inodes\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_inodes_count&type=symbol); /\* Inodes count \*/  343 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_blocks\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_blocks_count&type=symbol); /\* Blocks count \*/  344 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_r\_blocks\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_r_blocks_count&type=symbol); /\* Reserved blocks count \*/  345 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_free\_blocks\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_free_blocks_count&type=symbol); /\* Free blocks count \*/  346 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_free\_inodes\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_free_inodes_count&type=symbol); /\* Free inodes count \*/  347 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_first\_data\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_first_data_block&type=symbol); /\* First Data Block \*/  348 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_log\_block\_size](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_log_block_size&type=symbol); /\* Block size \*/  349 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_log\_frag\_size](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_log_frag_size&type=symbol); /\* Fragment size \*/  350 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_blocks\_per\_group](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_blocks_per_group&type=symbol); /\* # Blocks per group \*/  351 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_frags\_per\_group](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_frags_per_group&type=symbol); /\* # Fragments per group \*/  352 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_inodes\_per\_group](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_inodes_per_group&type=symbol); /\* # Inodes per group \*/  353 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [s\_mtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=s_mtime&type=symbol); /\* Mount time \*/  …  411 }; |

### 1.1.3 块组描述符表（GDT，Group Descriptor Table）

1. 块组描述符表由多个块组描述符（GD，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，从哪里开始时数据块，空闲的inode和数据块还有多少个等等。
2. 和超级块类似，块组描述符表在每个块组的开头也都有一份拷贝，这些信息是非常重要的，一旦超级块意外损坏就会丢失整个分区的信息，一旦块组描述符意外损坏就会丢失整个块组的数据，因此它们都有多份拷贝。默认情况下，只有第一个块组的块组描述符会被系统内核使用，其他块组的超级块可以在e2fsck之类的程序对磁盘上的文件系统进行一致性检查时使用，这样当第一个块组的开头意外损坏时就可以使用其他拷贝来恢复，从而减少数据丢失。
3. 每个块组描述符占32个字节，块组描述表则占用多个块大小，具体数量不确定，由分区的大小和块大小确定。
4. 在ext2文件系统中，块组描述符通过一个名为ext2\_group\_desc的结构进行描述，如下所示（引用自Kernel的/root/include/linux/ext2\_fs.h）：

|  |
| --- |
| 133 /\*  134 \* Structure of a blocks group descriptor  135 \*/  136 **struct** [ext2\_group\_desc](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=ext2_group_desc&type=reference" \o "Multiple refered from 23 places.)  137 {  138 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [bg\_block\_bitmap](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_block_bitmap&type=symbol); /\* Blocks bitmap block \*/  139 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [bg\_inode\_bitmap](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_inode_bitmap&type=symbol); /\* Inodes bitmap block \*/  140 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [bg\_inode\_table](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_inode_table&type=symbol); /\* Inodes table block \*/  141 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [bg\_free\_blocks\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_free_blocks_count&type=symbol); /\* Free blocks count \*/  142 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [bg\_free\_inodes\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_free_inodes_count&type=symbol); /\* Free inodes count \*/  143 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [bg\_used\_dirs\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_used_dirs_count&type=symbol); /\* Directories count \*/  144 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [bg\_pad](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_pad&type=symbol);  145 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [bg\_reserved](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=bg_reserved&type=symbol)[3];  146 }; |

### 1.1.4 块位图（Block Bitmap）

1. 在ext2文件系统中块组中的块是这样利用的：超级块、块组描述表、块位图、inode位图、inode表这几部分存储该块组的描述信息；数据块（Data Block）存储所有文件的数据；
2. 假设某个分区的块大小为1024字节，某个文件时2049字节，那么就需要三个数据块来存储这个文件，即使第三个块只存了一个字节也需要占用一个整块。
3. 块位图就是用来描述整个块组中哪些块已用，哪些块是空闲的。块位图占一个块，其中每个bit代表本块组的一个块，这个bit为1表示该块已用，这个bit为0表示该块空闲可用。
4. 使用df命令统计分区的已用空间速度非常块是因为只需要查看每个块组的块位图即可，不需要搜索真个分区。相反，使用du命令查看一个较大目录的已用空间就非常慢，因为其不可避免地要搜索真个目录的所有文件。
5. 至于一个分区中到底有多少个块组，主要的限制在于块位图本身必须只占一个块，因此块组的个数这取决于两个因素：分区大小和块大小。最终的计算公式如下：

分区的块组数=分区大小/（块大小\*8）

注：格式化时可以用mkfs命令的-g参数指定一个块组有多少个块，但是通常不需要手动指定，mkfs工具会计算出最优的数值。

### 1.1.5 索引节点位图（inode Bitmap）

inode位图和块位图类似，用来描述整个块组中哪些inode已用，哪些inode未使用。inode位图本身占一个块，其中每个bit代表本块组的一个inode，这个bit为1表示该inode已用，这个bit为0表示该inode未使用。

### 1.1.6 索引节点表（inode Table）

1. 索引节点表由多个索引节点（inode）组成，每个索引节点大小为128个字节。
2. 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、权限、文件大小等等（也就ls –l命令看到的信息），这些信息都存在indoe中而不是数据块（Data Block）中。每个文件都有一个inode中，一个块组中的所有inode组成了inode表。
3. inode表占多少个块在格式化时就要决定并写入块组描述符中。mke2fs格式化工的默认策略是一个块组有多少个8KB就分配多少个inode。这样，若这个分区中平均每个文件的大小是8KB，当分区存满的时候inode表会得到充分的利用，数据块也不浪费；如果这个分区存的都是很大的文件，则数据块用完的时候inode则会有一些浪费；如果这个分区存的都是很小的文件，则有可能数据块还用完indoe就已经用完了，数据块可能有很大的浪费。
4. 如果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，可以使用mkfs命令的-i参数手动指定每多少个字节分配一个inode。
5. 在ext2文件系统中，索引节点通过一个名为ext2\_inode的结构进行描述，如下所示（引用自Kernel的/root/include/linux/ext2\_fs.h）：

|  |
| --- |
| 208 /\*  209 \* Structure of an inode on the disk  210 \*/  211 **struct** [ext2\_inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=ext2_inode&type=reference" \o "Multiple refered from 4 places.) {  212 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [i\_mode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_mode&type=symbol); /\* File mode \*/  213 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [i\_uid](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_uid&type=symbol); /\* Low 16 bits of Owner Uid \*/  214 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_size](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_size&type=symbol); /\* Size in bytes \*/  这里的块是以512字节来计算的，并非格式化文件系统指定的块大小，因此单个文件的最大值为2^32\*512B=2TB  215 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_atime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_atime&type=symbol); /\* Access time \*/  216 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_ctime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_ctime&type=symbol); /\* Creation time \*/  217 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_mtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_mtime&type=symbol); /\* Modification time \*/  218 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_dtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_dtime&type=symbol); /\* Deletion Time \*/  219 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [i\_gid](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_gid&type=symbol); /\* Low 16 bits of Group Id \*/  220 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [i\_links\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_links_count&type=symbol); /\* Links count \*/  该域保存了数据存放在数据块的位置  221 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_blocks](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_blocks&type=symbol); /\* Blocks count \*/  222 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_flags](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_flags&type=symbol); /\* File flags \*/  …  234 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [i\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=i_block&type=symbol)[[EXT2\_N\_BLOCKS](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/S/17006.html#L169)];/\* Pointers to blocks \*/  …  263 }; |

### 1.1.7 数据块（Data Block）

根据不同的文件类型有以下几种情况：

1. 对于常规文件，文件的数据存储在数据块中。
2. 对于目录，该目录下的所有文件名和目录名存储在数据块中。

注： a：目录也是一种文件，是一种特殊类型的文件。

b：文件名保存在它所在目录的数据块中，除文件名外，ls –l命令看到的其他信息都存在该文件的inode中。

1. 对于符号链接文件，如果目标路径名较短（小于60个字符）则直接保存在inode中以便快速地查找，如果目标路径名较长则分配一个数据块来保存。
2. 对于硬链接文件，与原文件采用相同的inode，仅在所属目录的数据块中有一个目录项记录。

注： a：不能对目录建立硬链接

b：不能建立跨文件系统的硬链接

1. 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。
2. 在ext2文件系统中，目录项通过一个名为ext2\_dir\_entry\_2的结构进行描述，如下所示（引用自Kernel的/root/include/linux/ext2\_fs.h）：

|  |
| --- |
| 517 /\*  518 \* The new version of the directory entry. Since EXT2 structures are  519 \* stored in intel byte order, and the name\_len field could never be  520 \* bigger than 255 chars, it's safe to reclaim the extra byte for the  521 \* file\_type field.  522 \*/  523 **struct** [ext2\_dir\_entry\_2](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=ext2_dir_entry_2&type=reference" \o "Multiple refered from 13 places.) {  524 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 3 places.) [inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/S/16909.html#L529); /\* Inode number \*/  525 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 3 places.) [rec\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=rec_len&type=symbol); /\* Directory entry length \*/  526 [\_\_u8](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__u8&type=definitions" \o "Multiple defined in 22 places.) [name\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=name_len&type=symbol); /\* Name length \*/  527 [\_\_u8](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=__u8&type=definitions" \o "Multiple defined in 22 places.) [file\_type](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/S/21264.html#L54);  528 **char** [name](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/cgi-bin/global.cgi?pattern=name&type=definitions" \o "Multiple defined in 36 places.)[[EXT2\_NAME\_LEN](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/S/17006.html#L508)]; /\* File name \*/   1. }; |

## 1.2 数据块寻址

在索引节点表中介绍的ext2\_inode结构保存文件的各种描述信息，其中i\_block域是一个大小为[EXT2\_N\_BLOCKS](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL5.2GA/HTML/S/17006.html#L169)（通常来说，其值为15）的数组，其中保存的就是真正存放文件数据的数据块位置。

在ext2文件系统中，采用了直接寻址和间接寻址两种方式来对数据进行寻址，原理如下图所示：（假设块大小为b个字节）



### 1.2.1 直接寻址与间接寻址

1. 对于i\_block的前12个元素（i\_block[0]到i\_block[11]）来说，其中存放的就是实际的数据块号，即对应于文件的0到11块。这种方式称为直接寻址。
2. 对于i\_block的第13个元素（i\_block[12]）来说，其中存放的是另外一个数据块的逻辑块号；这个块中并不存放真正的数据，而是存放真正保存数据的数据块的块号。由于每个块号需要使用4个字节表示，因此这种寻址方式可以访问的对应文件的块号范围为12到(b/4)+11，这种寻址方式称为间接寻址。
3. 对于i\_block的第14个元素（i\_block[13]）来说，其中存放的也是另外一个数据块的逻辑块号。不过其则利用了二级间接索引，第三级数组存放的才是真正保存文件的逻辑块号，这种寻址方式称为二次间接寻址，对应文件块号的寻址范围为b/4+12到(b/4)^2+(b/4)+11。
4. 对于i\_block的第15个元素（i\_block[14]）来说，则利用了三级间接寻址，第四级数组中存放的才是真正保存文件的块号，其寻址范围为(b/4)^2+12到(b/4)^3+(b/4)^12+(b/4)+11。
5. 可见，这种寻址方式对于访问不超过12个数据块的小文件事非常快的，访问文件中的任意数据只需要两次读盘操：一次读inode；一次读数据；而访问大文中的数据则需要最多五次读盘操作：indoe、一级间接寻址、二级间接寻址、三级间接寻址、数据块。

### 1.2.2 数据块寻址能力

ext2文件系统可以支持1024、2048和4096字节三种大小的块，其对应的寻址能力如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 块大小 | 直接寻址 | 间接寻址 | 二次间接寻址 | 三次间接寻址 |
| 1024B | 12KB | 268KB | 64.26MB | 16.06GB |
| 2048B | 24KB | 1.02MB | 513.02MB | 265.5GB |
| 4096B | 48KB | 4.04MB | 4GB | 4TB |

### 1.2.3 数据块寻址过程

假设有一个文件/opt/file，我们要使用cat命令把其内容打印到终端上，其数据块的寻址过程如下：

1. 读出inode表中第2项，也就是根目录的inode，从中找出根目录的数据块的位置
2. 从根目录的数据块中找出文件名为opt的记录，从记录中读出它的inode号
3. 读出opt目录的inode，从中找出它的数据块的位置
4. 从opt目录的数据块中找出文件名为file的记录，从记录中读出它的inode号
5. 读出file文件的inode，从中找出它的数据的位置
6. 输出其内容到终端

## 1.3 实例剖析

### 1.3.1 查看文件系统的布局

1. dumpe2fs工具

首先使用parted命令格式一个分区大小为1MB，然后使用mkfs.ext2来创建一个ext2文件系统，最后使用dumpe2fs命令来查看文件系统的超级块和块组描述符表中的信息，如下所示：

|  |
| --- |
| [root@localhost /]# dumpe2fs /dev/sde1  dumpe2fs 1.41.12 (17-May-2010)  Filesystem volume name: <none>  Last mounted on: <not available>  Filesystem UUID: 69525cc0-d9f7-4ea6-86b3-a0e1b94e22cf  Filesystem magic number: 0xEF53  Filesystem revision #: 1 (dynamic)  Filesystem features: ext\_attr resize\_inode dir\_index filetype sparse\_super  Filesystem flags: signed\_directory\_hash  Default mount options: (none)  Filesystem state: not clean  Errors behavior: Continue  Filesystem OS type: Linux  Inode count: 128  Block count: 976  Reserved block count: 48  Free blocks: 938  Free inodes: 117  First block: 1  Block size: 1024  Fragment size: 1024  Reserved GDT blocks: 3  Blocks per group: 8192  Fragments per group: 8192  Inodes per group: 128  Inode blocks per group: 16  Filesystem created: Wed Jan 22 02:04:10 2014  Last mount time: Wed Jan 22 02:04:23 2014  Last write time: Wed Jan 22 02:04:23 2014  Mount count: 1  Maximum mount count: 21  Last checked: Wed Jan 22 02:04:10 2014  Check interval: 15552000 (6 months)  Next check after: Mon Jul 21 03:04:10 2014  Reserved blocks uid: 0 (user root)  Reserved blocks gid: 0 (group root)  First inode: 11  Inode size: 128  Default directory hash: half\_md4  Directory Hash Seed: fbd896e5-47aa-4781-be0a-a0dd2e9bf7b3  Group 0: (Blocks 1-975)  Primary superblock at 1, Group descriptors at 2-2  Reserved GDT blocks at 3-5  Block bitmap at 6 (+5), Inode bitmap at 7 (+6)  Inode table at 8-23 (+7)  938 free blocks, 117 free inodes, 2 directories  Free blocks: 38-975  Free inodes: 12-128 |

从上图可以看出：

1. 块大小是1024字节，共有976个块
2. 第0个块是启动块，因此Group 0占据第1个块到第975个块
3. 块位图占一个块，共有1024\*8=8192个bit，足够表示这975个块了
4. 共有128 inode，每个inode占128字节

…

1. hexdump命令

使用hexdump命令来查看文件系统的所有字节，来深入理解文件系统的布局，如下所示：

|  |
| --- |
| [root@localhost /]# hexdump -C /dev/sde1  00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  每个字段表示什么意义请参考超级块的数据结构ext2\_super\_block  从00000000开始的1KB是启动块，内容都为零  \*  inode count=128  block count=1024  00000400 80 00 00 00 d0 03 00 00 30 00 00 00 aa 03 00 00 |........0.......|  从00000400到000007ff的1KB是超级块  00000410 75 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 |u...............|  00000420 00 20 00 00 00 20 00 00 80 00 00 00 77 6d df 52 |. ... ......wm.R|  00000430 77 6d df 52 01 00 15 00 53 ef 00 00 01 00 00 00 |wm.R....S.......|  00000440 6a 6d df 52 00 4e ed 00 00 00 00 00 01 00 00 00 |jm.R.N..........|  00000450 00 00 00 00 0b 00 00 00 80 00 00 00 38 00 00 00 |............8...|  00000460 02 00 00 00 01 00 00 00 69 52 5c c0 d9 f7 4e a6 |........iR\...N.|  00000470 86 b3 a0 e1 b9 4e 22 cf 00 00 00 00 00 00 00 00 |.....N".........|  00000480 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  000004c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 00 |................|  000004d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  000004e0 00 00 00 00 00 00 00 00 00 00 00 00 fb d8 96 e5 |................|  000004f0 47 aa 47 81 be 0a a0 dd 2e 9b f7 b3 01 00 00 00 |G.G.............|  00000500 00 00 00 00 00 00 00 00 6a 6d df 52 00 00 00 00 |........jm.R....|  00000510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00000560 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  00000570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  每个字段表示什么意义请参考块组描述符的数据结构ext2\_group\_desc  \*  Block Bitmap at: 6  inode Bitmap at: 7  每个字段表示什么意义请参考超级块的数据结构ext2\_super\_block  从00000800开始的4KB是块组描述符表  00000800 06 00 00 00 07 00 00 00 08 00 00 00 aa 03 75 00 |..............u.|  00000810 02 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  00000820 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  前37个Block已用，空闲的Block编号从38号开始  从00001800开始的1KB是块位图  00001800 ff ff ff ff 1f 00 00 00 00 00 00 00 00 00 00 00 |................|  00001810 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00001870 00 00 00 00 00 00 00 00 00 80 ff ff ff ff ff ff |................|  00001880 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |................|  \*  前11位inode已用，空闲的inode编号从12到128  00001c00 ff 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  从00001c00开始的1KB是inode位图  00001c10 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |................|  \*  00002000 00 00 00 00 00 00 00 00 6a 6d df 52 6a 6d df 52 |........jm.Rjm.R|  从00002000开始的7KB是inode Table  00002010 6a 6d df 52 00 00 00 00 00 00 00 00 00 00 00 00 |jm.R............|  00002020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  每个字段表示什么意义请参inode的数据结构ext2\_inode  \*  User=0000,即root用户  st\_mode=040755  00002080 ed 41 00 00 00 04 00 00 6a 6d df 52 6a 6d df 52 |.A......jm.Rjm.R|  从00002080开始的128个字节是inode号为2的索引节点，即根目录的索引节点  00002090 6a 6d df 52 00 00 00 00 00 00 03 00 02 00 00 00 |jm.R............|  000020a0 00 00 00 00 00 00 00 00 18 00 00 00 00 00 00 00 |................|  000020b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00002300 80 81 00 00 00 30 04 04 6a 6d df 52 6a 6d df 52 |.....0..jm.Rjm.R|  00002310 6a 6d df 52 00 00 00 00 00 00 01 00 08 00 00 00 |jm.R............|  00002320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00002350 00 00 00 00 00 00 00 00 00 00 00 00 25 00 00 00 |............%...|  00002360 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00002500 c0 41 00 00 00 30 00 00 6a 6d df 52 6a 6d df 52 |.A...0..jm.Rjm.R|  00002510 6a 6d df 52 00 00 00 00 00 00 02 00 18 00 00 00 |jm.R............|  00002520 00 00 00 00 00 00 00 00 19 00 00 00 1a 00 00 00 |................|  00002530 1b 00 00 00 1c 00 00 00 1d 00 00 00 1e 00 00 00 |................|  00002540 1f 00 00 00 20 00 00 00 21 00 00 00 22 00 00 00 |.... ...!..."...|  00002550 23 00 00 00 24 00 00 00 00 00 00 00 00 00 00 00 |#...$...........|  每个字段表示什么意义请参目录项的数据结构ext2\_dir\_entry\_2  \*  inode号为2  目录项的长度为12  00006000 02 00 00 00 0c 00 01 02 2e 00 00 00 02 00 00 00 |................|  从00006000开始的1KB（目录大小为block的整数倍）为根目录的数据块  00006010 0c 00 02 02 2e 2e 00 00 0b 00 00 00 e8 03 0a 02 |................|  00006020 6c 6f 73 74 2b 66 6f 75 6e 64 00 00 00 00 00 00 |lost+found......|  00006030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00006400 0b 00 00 00 0c 00 01 02 2e 00 00 00 02 00 00 00 |................|  00006410 f4 03 02 02 2e 2e 00 00 00 00 00 00 00 00 00 00 |................|  00006420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00006800 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00 00 |................|  00006810 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \*  00006c00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00 00 |................|  00006c10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|  \* |

从上图中的各项数据来看，与之前的分析以及dumpe2fs命令的输出结果一致。

### 1.3.2 恢复已删除的文件

在ext2文件系统中，文件被删除时其对应的数据块并不会被真正的删除，因此通过一些技术手段完全有可能将被删除的文件恢复回来。接下来就来进行分析：

1. 删除文件的原理
2. 目录是一种特殊的文件，其索引节点的结构与普通文件一样，唯一的区别是目录中的数据都是按照ext2\_dir\_entry\_2结构存储在数据块的。先来看一个已准备好的目录：

|  |
| --- |
| [root@localhost / ]# debugfs /dev/sde1  debugfs 1.41.12 (17-May-2010)  debugfs: ls -l  2 40755 (2) 0 0 1024 23-Jan-2014 13:38 .  2 40755 (2) 0 0 1024 23-Jan-2014 13:38 ..  11 40700 (2) 0 0 12288 23-Jan-2014 13:35 lost+found  12 40755 (2) 0 0 1024 23-Jan-2014 13:36 testdir1  13 40755 (2) 0 0 1024 23-Jan-2014 13:36 testdir2  15 100644 (1) 0 0 12 23-Jan-2014 13:37 testfile  16 100644 (1) 0 0 75 23-Jan-2014 13:38 testfile2  debugfs: stat <2>  Inode: 2 Type: directory Mode: 0755 Flags: 0x0  Generation: 0 Version: 0x00000000  User: 0 Group: 0 Size: 1024  File ACL: 0 Directory ACL: 0  Links: 3 Blockcount: 2  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52e08b6d -- Thu Jan 23 11:24:29 2014  atime: 0x52e08b6e -- Thu Jan 23 11:24:30 2014  mtime: 0x52e08b6d -- Thu Jan 23 11:24:29 2014  根目录的数据块所在位置  BLOCKS:  (0):24  TOTAL: 1  debugfs: stat <15>  Inode: 15 Type: regular Mode: 0644 Flags: 0x0  Generation: 762856613 Version: 0x00000000  User: 0 Group: 0 Size: 12  File ACL: 0 Directory ACL: 0  Links: 1 Blockcount: 2  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52e0ab88 -- Thu Jan 23 13:41:28 2014  atime: 0x52e0aa84 -- Thu Jan 23 13:37:08 2014  mtime: 0x52e0aa84 -- Thu Jan 23 13:37:08 2014  BLOCKS:  (0):64  TOTAL: 1 |

1. 备份根目录的数据块

|  |
| --- |
| [root@localhost /]# dd if=/dev/sde1 of=/block.24.orig bs=1024 count=1 skip=24  1+0 records in  1+0 records out  1024 bytes (1.0 kB) copied, 0.00032152 s, 3.2 MB/s |

1. 写一个小程序，读取目录数据块中的数据

|  |
| --- |
| /\*Program:  \* read\_dir\_entry.c  \* read directory's data block and output to terminal  \*History:  \* renyl 2014/1/23 0.1version  \*/  #include <stdio.h>  #include <stdlib.h>  #include <ext2fs/ext2\_fs.h>  /\*custom data struct similar to ext2\_dir\_entry\_2\*/  struct ext2\_dir\_entry\_part{  \_\_u32 inode;  \_\_u16 rec\_len;  \_\_u8 name\_len;  \_\_u8 file\_type;  };  void usage()  {  printf("./read\_dir\_entry <dir\_entry\_filename> <dir\_entry\_size>\n");  exit(1);  }  int main(int argc, char\* argv[])  {  struct ext2\_dir\_entry\_part dep;  struct ext2\_dir\_entry\_2 de;  char \*filename = NULL;  FILE \*fp = NULL;  int de\_size = 0;  int len = 0;  int rtn = 0;  if (argc < 3)  {  printf("Too few parameters!\n");  usage();  }  filename = argv[1];  de\_size = atoi(argv[2]);  fp = fopen(filename,"r");  if (!fp)  printf("can't open file: %s\n",filename);    printf("offset | inode\_number | rec\_len | name\_len | file\_type | name\n");  printf("=========================================\n");  while ( rtn = fread(&dep, sizeof(struct ext2\_dir\_entry\_part), 1, fp) )  {  if (dep.rec\_len <=0)  {  fclose(fp);  exit(3);  }  fseek(fp, 0 - sizeof(struct ext2\_dir\_entry\_part), SEEK\_CUR);  /\*dir\_entry need 4Byte alignment.\*/  fread(&de,((int)(dep.name\_len+3)/4)\*4 \  + sizeof(struct ext2\_dir\_entry\_part), 1 ,fp);  de.name[de.name\_len]='\0';  printf("%6d: %10d%10d%10d%10d %s\n",\  len, de.inode, de.rec\_len, de.name\_len, de.file\_type, de.name);    len+=dep.rec\_len;  if (len >= de\_size - sizeof(struct ext2\_dir\_entry\_part))  {  fclose(fp);  return 0;  }  }  fclose(fp);  return 0;  } |

1. 对比文件testfile被删除前后索引节点的变化以及所属目录数据块的变化

|  |
| --- |
| [root@localhost /]# gcc -g read\_dir\_entry.c -o read\_dir\_entry  [root@localhost /]# ./read\_dir\_entry block.24 1024  offset | inode\_number | rec\_len | name\_len | file\_type | name  ======================================================  0: 2 12 1 2 .  12: 2 12 2 2 ..  24: 11 20 10 2 lost+found  该文件是在我编辑文件testfile2时，系统自动保存的  44: 12 16 8 2 testdir1  60: 13 40 8 2 testdir2  100: 0 24 14 1 .testfile2.swp  124: 15 16 8 1 testfile  140: 16 908s 9 1 testfile2  [root@localhost /]# cat /home/renyl/mountdir/testfile  重新挂载是为了确保删除操作会被同步到磁盘上的数据块中  hello world  [root@localhost /]# rm –rf /home/renyl/mountdir/testfile  [root@localhost /]# umount /dev/sde1  [root@localhost /]# mount /dev/sde1 /home/renyl/mountdir  [root@localhost /]# dd if=/dev/sde1 of=/block.24.deleted bs=1024 count=1 skip=24  1+0 records in  1+0 records out  1024 bytes (1.0 kB) copied, 0.00032152 s, 3.2 MB/s  [root@localhost /]# ./read\_dir\_entry block.24.deleted 1024  offset | inode\_number | rec\_len | name\_len | file\_type | name  =======================================================  0: 2 12 1 2 .  12: 2 12 2 2 ..  24: 11 20 10 2 lost+found  44: 12 16 8 2 testdir1  索引节点被清空了，这也就是为什么ls命令无法显示被删除后的文件  60: 13 56 8 2 testdir2  116: 0 24 14 1 .testfile2.swp  140: 0 16 8 1 testfile  156: 16 908 9 1 testfile2  [root@localhost / ]# debugfs /dev/sde1  debugfs 1.41.12 (17-May-2010)  debugfs: stat <15>  Inode: 15 Type: regular Mode: 0644 Flags: 0x0  Generation: 762856613 Version: 0x00000000  User: 0 Group: 0 Size: 12  File ACL: 0 Directory ACL: 0  Links: 0 Blockcount: 2  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52e0ab88 -- Thu Jan 23 13:41:28 2014  atime: 0x52e0aa84 -- Thu Jan 23 13:37:08 2014  mtime: 0x52e0aa84 -- Thu Jan 23 13:37:08 2014  dtime: 0x52e0ab88 -- Thu Jan 23 13:41:28 2014  BLOCKS:  (0):64  TOTAL: 1  [root@localhost /]# cd /home/renyl/mountdir/  [root@localhost mountdir]# ls -ali  total 20  2 drwxr-xr-x 5 root root 1024 Jan 23 13:41 .  32792 drwxr-xr-x. 7 root root 4096 Jan 23 06:55 ..  11 drwx------ 2 root root 12288 Jan 23 13:35 lost+found  12 drwxr-xr-x 2 root root 1024 Jan 23 13:36 testdir1  13 drwxr-xr-x 2 root root 1024 Jan 23 13:36 testdir2  16 -rw-r--r-- 1 root root 75 Jan 23 13:38 testfile2 |

由上表可以看出，文件testfile被删除后，索引节点和所在目录的数据块都发生了变化：

* 索引节点：

a）设置了i\_dtime域，该域只有文件被删除时才设置

b）硬链接数（link）被设置为零

c）将Block Bitmap和inode Bitmap中所对应的bit设置为0，即可用状态

* 所在目录数据块：

1. 将文件的索引节点号清空
2. 将文件的rec\_len长度合并到前一项

由于文件被删除时，其数据块没有被删除，以及索引节点中的重要信息没有被删除（如文件属性、大小、访问权限以及数据块的块号），因此可以把已删除的文件恢复出来。

2）删除文件的恢复

|  |
| --- |
| [root@localhost mountdir]# debugfs /dev/sde1  debugfs 1.41.12 (17-May-2010)  debugfs: lsdel  Inode Owner Mode Size Blocks Time deleted  14 0 100600 12288 12/ 12 Thu Jan 23 13:38:04 2014  15 0 100644 12 1/ 1 Thu Jan 23 13:41:28 2014  2 deleted inodes found.  debugfs: dump <15> testfile.bak  debugfs: q  [root@localhost mountdir]# ls -ali  total 21  2 drwxr-xr-x 5 root root 1024 Jan 23 15:34 .  32792 drwxr-xr-x. 7 root root 4096 Jan 23 06:55 ..  11 drwx------ 2 root root 12288 Jan 23 13:35 lost+found  12 drwxr-xr-x 2 root root 1024 Jan 23 13:36 testdir1  13 drwxr-xr-x 2 root root 1024 Jan 23 13:36 testdir2  16 -rw-r--r-- 1 root root 75 Jan 23 13:38 testfile2  14 -rw-r--r-- 1 root root 12 Jan 23 15:34 testfile.bak  [root@localhost mountdir]# cat testfile.bak  hello world |

debugfs的lsdel命令去扫描磁盘上索引节点表中的所有索引节点，其中i\_dtime不为空的项就被认为是已经删除的文件对对应的索引节点。使用dump命令可以恢复已删除的文件。

Linux文件系统还存在一些特殊文件，如目录、链接文件，空洞文件等，对这些文件的恢复思路与普通文件类似。恢复系统中删除的文件是一个非常繁琐的过程，而工具e2undel可以用来方便地恢复文件系统中已删除的各种文件。

# 2 ext3文件系统

## 2.1 产生的背景

1. ext2文件系统具有较好的文件存取性能，是由于其采用了“文件系统缓存”的概念，可以加速文件的读写速度。然而，如果“文件系统缓存”中的数据尚未写入磁盘，机器就发生了断电等意外状况，就会造成磁盘数据不一致的情况，这会破坏磁盘数据的完整性（文件数据与元数据不一致）。
2. 为了确保数据的完整，在系统引导时，会自动检查文件系统上次是否是正常卸载的。如果是非正常卸载或者已经使用到一定的次数，就会自动运行fsck之类的程序强制进行一致性检查，并修复存在问题的地方，使ext2文件系统恢复到新的一致性状态。
3. 随着硬盘技术的发展，磁盘容量变得越来越大，对磁盘进行一致性检查可能会占用很长时间，这对于一些关键应用来说是无法忍受的，于是日志文件系统（Journal File System）的概念就应用而生了。

## 2.2 日志文件系统

### 2.2.1 日志文件系统的定义

日志文件系统：是指在文件系统发生变化时，先把相关信息写入一个被称为日志的区域，然后再把变化写入主文件系统的文件系统。在文件系统发生故障（如内核崩溃或突然停电）时，日志文件系统更容易保持一致性，并且可以较快恢复。

具体来说，就是在修改文件系统内容的同时（或之前），将修改变化记录到日志中，这样就可以在意外发生的情况下，就可以根据日志将文件系统恢复到一致状态。这些操作完全可以在重新挂载文件系统时来完成。因此，在重新启动机器时，并不需要对文件系统进行一致性检查，这样可以提高系统的可用程度。

### 2.2.2 日志文件系统的原理

1. 对文件系统进行修改时，需要进行很多操作。这些操作可能中途被打断，也就是说，这些操作不是“不可中断”（atomic）的，如果操作被打断，就可能造成文件系统出现不一致的状态。
2. 例如删除文件时，先要从目录树中移除文件的标示，然后收回文件占用空间。如果在这两步之间操作被打断，文件占用的空间就无法收回。文件系统认为它是被占用的，但实际上目录树种已经找不到使用它的文件了。
3. 在非日志文件系统中，要检查并修复类似的错误就必须对整个文件系统的数据结构进行检查。一般在挂载文件系统前，操作系统会检查它上次是否被成功卸载，如果没有，就会对其进行检查。如果文件系统很大，就会花费很长时间。
4. 为了避免这样的问题，日志文件系统分配了一个称为日志的区域来提前记录要对文件系统做的更改。在系统崩溃或突然掉电后，只要读取日志重新执行未完成的操作，文件系统就可以恢复一致。这种恢复是“原子操作”的，有如下几种情况：
   1. 不需要重新执行：这个事务被标记为已经完成
   2. 成功重新执行：根据日志记录，这个事务被重新执行
   3. 无法重新执行：这个事务会被撤销，就如同这个事务从来没有发生过
   4. 日志本身不完成：事务还没有被完全写入日志，它会被简单忽略

## 2.3 日志的分类

ext3文件系统提供了3种日志模式：

1. 回写（data=writeback）：只有元数据被记录到日志中，数据被直接写入主文件系统中，这种模式速度最快，提供较好的性能，但也有较大的风险。例如，在增大文件时，数据还没写入就发生崩溃，那么文件系统恢复后文件后面就可能出现垃圾数据。
2. 顺序（data=ordered）：只有元数据被记录到日志中，但在日志被标记为提交前，数据会被写入文件系统。在这种模式下，如果在增大文件时，数据还未写入就发生崩溃，那么在恢复时这个事务会被简单的撤销，文件保持原来的状态。
3. 数据（data=journal）：元数据和文件内容都先被写入到日志中，然后再提交到主文件系统。这提高了安全性，但损失了性能，因为所有数据要写入两次。在这种模式下，如果增大文件时，发生崩溃，那么可能有两种情况：
4. 日志完成：这时事务会被重新执行，修改会被提交到主文件系统
5. 日志不完成：这时文件系统还未被修改，只需要简单放弃这个事务

不管哪种日志模式，这对于用户来说都是透明的，用户根本就察觉不到日志文件的存在，只是内核在挂载文件系统时会自动检查日志的内容，并采取相应的操作，将尚未提交到磁盘上的操作重新写入磁盘，从而确保文件系统的一致性。

## 2.4 ext3与ext2的关系

ext3最大的特性在于它完全兼容ext2文件系统，ext2和ext3文件系统之间可以无缝地进行变换，二者在磁盘上采用完全相同的数据格式进行存储。

1）创建一个ext3和一个ext2文件系统，如下所示：

|  |
| --- |
| [root@localhost /]# mkfs.ext3 /dev/sde2  mke2fs 1.41.12 (17-May-2010)  Filesystem label=  OS type: Linux  Block size=4096 (log=2)  Fragment size=4096 (log=2)  Stride=0 blocks, Stripe width=0 blocks  61056 inodes, 243968 blocks  12198 blocks (5.00%) reserved for the super user  First data block=0  Maximum filesystem blocks=251658240  8 block groups  32768 blocks per group, 32768 fragments per group  7632 inodes per group  Superblock backups stored on blocks:  创建日志  32768, 98304, 163840, 229376  Writing inode tables: done  Creating journal (4096 blocks): done  Writing superblocks and filesystem accounting information: done  This filesystem will be automatically checked every 31 mounts or  180 days, whichever comes first. Use tune2fs -c or -i to override.  [root@localhost /]# mkfs.ext2 /dev/sde3  mke2fs 1.41.12 (17-May-2010)  Filesystem label=  OS type: Linux  Block size=4096 (log=2)  Fragment size=4096 (log=2)  Stride=0 blocks, Stripe width=0 blocks  61056 inodes, 243968 blocks  12198 blocks (5.00%) reserved for the super user  First data block=0  Maximum filesystem blocks=251658240  8 block groups  32768 blocks per group, 32768 fragments per group  7632 inodes per group  Superblock backups stored on blocks:  没有创建日志  32768, 98304, 163840, 229376  Writing inode tables: done  Writing superblocks and filesystem accounting information: done  This filesystem will be automatically checked every 37 mounts or  180 days, whichever comes first. Use tune2fs -c or -i to override. |

从上表可以看出，mkfs.ext3命令额外在文件系统中使用4096个数据块创建了日志。

1. ext2文件系统转换成ext3文件系统

|  |
| --- |
| [root@localhost /]# tune2fs -j /dev/sde3  创建日志  tune2fs 1.41.12 (17-May-2010)  Creating journal inode: done  This filesystem will be automatically checked every 37 mounts or  180 days, whichever comes first. Use tune2fs -c or -i to override.  [root@localhost /]# dumpe2fs /dev/sde3  Filesystem volume name: <none>  Last mounted on: <not available>  Filesystem UUID: 83da648b-4c0d-42cd-9890-320922b14d56  Filesystem magic number: 0xEF53  Filesystem revision #: 1 (dynamic)  Filesystem features: has\_journal ext\_attr resize\_inode dir\_index filetype sparse\_super large\_file  Filesystem flags: signed\_directory\_hash  Default mount options: (none)  Filesystem state: clean  Errors behavior: Continue  Filesystem OS type: Linux  Inode count: 61056  Block count: 243968  Reserved block count: 12198  Free blocks: 235724  Free inodes: 61045  First block: 0  Block size: 4096  Fragment size: 4096  Reserved GDT blocks: 59  Blocks per group: 32768  Fragments per group: 32768  Inodes per group: 7632  Inode blocks per group: 477  Filesystem created: Fri Jan 24 11:33:32 2014  Last mount time: n/a  Last write time: Fri Jan 24 11:35:25 2014  Mount count: 0  Maximum mount count: 37  Last checked: Fri Jan 24 11:33:32 2014  Check interval: 15552000 (6 months)  Next check after: Wed Jul 23 11:33:32 2014  Reserved blocks uid: 0 (user root)  Reserved blocks gid: 0 (group root)  First inode: 11  Inode size: 256  Required extra isize: 28  Desired extra isize: 28  Journal inode: 8  Default directory hash: half\_md4  Directory Hash Seed: ddc8cace-78d7-47f9-9998-996594fe77b8  Journal backup: inode blocks  日志的详细信息（大小、位置等）  Journal features: (none)  Journal size: 16M  Journal length: 4096  Journal sequence: 0x00000001  Journal start: 0  … |

从上表可以看出，ext2文件系统可以使用tune2fs命令平滑地转换成ext3文件系统。

## 2.5 文件恢复

1. 对于恢复删除的文件来说，需要关心的是文件在磁盘上的存储格式。实际上，ext3在这方面完全兼容ext2，以至于大部分支持ext2文件系统的工具都可以在ext3文件系统上使用。以存储目录项和索引节点使用的数据结构来看，如下所示：（引用自Kernel的/root/include/linux/ext3\_fs.h）

|  |
| --- |
| 285 /\*  286 \* Structure of an inode on the disk  287 \*/  288 **struct** [ext3\_inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext3_inode&type=reference" \o "Multiple refered from 10 places.) {  289 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [i\_mode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_mode&type=symbol); /\* File mode \*/  290 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [i\_uid](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_uid&type=symbol); /\* Low 16 bits of Owner Uid \*/  291 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_size](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_size&type=symbol); /\* Size in bytes \*/  292 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_atime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_atime&type=symbol); /\* Access time \*/  293 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_ctime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_ctime&type=symbol); /\* Creation time \*/  294 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_mtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_mtime&type=symbol); /\* Modification time \*/  295 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_dtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_dtime&type=symbol); /\* Deletion Time \*/  296 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [i\_gid](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_gid&type=symbol); /\* Low 16 bits of Group Id \*/  297 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [i\_links\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_links_count&type=symbol); /\* Links count \*/  298 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_blocks](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_blocks&type=symbol); /\* Blocks count \*/  299 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_flags](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_flags&type=symbol); /\* File flags \*/  300 **union** {  301 **struct** {  302 [\_\_u32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u32&type=definitions" \o "Multiple defined in 6 places.) [l\_i\_reserved1](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=l_i_reserved1&type=symbol);  303 } [linux1](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=linux1&type=symbol" \o "Multiple used in 7 places.);  304 **struct** {  305 [\_\_u32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u32&type=definitions" \o "Multiple defined in 6 places.) [h\_i\_translator](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=h_i_translator&type=symbol);  306 } [hurd1](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=hurd1&type=symbol" \o "Multiple used in 6 places.);  307 **struct** {  308 [\_\_u32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u32&type=definitions" \o "Multiple defined in 6 places.) [m\_i\_reserved1](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=m_i_reserved1&type=symbol);  309 } [masix1](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=masix1&type=symbol" \o "Multiple used in 6 places.);  310 } [osd1](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=osd1&type=symbol" \o "Multiple used in 13 places.); /\* OS dependent 1 \*/  311 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_block&type=symbol)[[EXT3\_N\_BLOCKS](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/27761.html#L151)];/\* Pointers to blocks \*/  312 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_generation](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_generation&type=symbol); /\* File version (for NFS) \*/  …  671 /\*  672 \* The new version of the directory entry. Since EXT3 structures are  673 \* stored in intel byte order, and the name\_len field could never be  674 \* bigger than 255 chars, it's safe to reclaim the extra byte for the  675 \* file\_type field.  676 \*/  677 **struct** [ext3\_dir\_entry\_2](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext3_dir_entry_2&type=reference" \o "Multiple refered from 58 places.) {  678 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/27662.html#L758); /\* Inode number \*/  679 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [rec\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=rec_len&type=symbol); /\* Directory entry length \*/  680 [\_\_u8](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u8&type=definitions" \o "Multiple defined in 4 places.) [name\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=name_len&type=symbol); /\* Name length \*/  681 [\_\_u8](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u8&type=definitions" \o "Multiple defined in 4 places.) [file\_type](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/30919.html#L68);  682 **char** [name](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=name&type=definitions" \o "Multiple defined in 35 places.)[[EXT3\_NAME\_LEN](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/27761.html#L662)]; /\* File name \*/  683 };  … |

从上表可以看出，ext3使用的两个数据结构ext3\_dir\_entry\_2和ext3\_inode与ext2并没有根本的区别，这正是ext2和ext3文件系统可以实现自由转换的基础。

ext3和ext2文件系统具有这么好的兼容性和相似性，以至于数据块的寻址过程，文件系统的布局都基本一致。那么，文件的删除操作发生了变化吗？

1. 查看一下ext3文件系统删除文件前后索引节点信息的变化，如下所示：

|  |
| --- |
| [root@localhost mountdir]# debugfs /dev/sde2  debugfs 1.41.12 (17-May-2010)  debugfs: ls -l  2 40755 (2) 0 0 4096 24-Jan-2014 12:36 .  2 40755 (2) 0 0 4096 24-Jan-2014 12:36 ..  11 40700 (2) 0 0 16384 24-Jan-2014 11:23 lost+found  7633 40755 (2) 0 0 4096 24-Jan-2014 12:35 dir1  22897 40755 (2) 0 0 4096 24-Jan-2014 12:35 dir2  12 100644 (1) 0 0 19 24-Jan-2014 12:36 testfile2  13 100644 (1) 0 0 13 24-Jan-2014 12:35 testfile  debugfs: stat <13>  Inode: 13 Type: regular Mode: 0644 Flags: 0x0  Generation: 342839017 Version: 0x00000000  User: 0 Group: 0 Size: 13  File ACL: 0 Directory ACL: 0  Links: 1 Blockcount: 8  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52e1ed9e -- Fri Jan 24 12:35:42 2014  atime: 0x52e1eda1 -- Fri Jan 24 12:35:45 2014  mtime: 0x52e1ed9e -- Fri Jan 24 12:35:42 2014  Size of extra inode fields: 4  BLOCKS:  (0):30728  TOTAL: 1  重新挂载是为了确保删除操作会被同步到磁盘上的数据块中  debugfs: q  [root@localhost mountdir]# rm -rf testfile  [root@localhost mountdir]# cd /  [root@localhost /]# umount /dev/sde2  [root@localhost /]# mount /dev/sde2 /home/renyl/mountdir/  [root@localhost /]# debugfs /dev/sde2  debugfs 1.41.12 (17-May-2010)  debugfs: ls -l  2 40755 (2) 0 0 4096 24-Jan-2014 12:36 .  2 40755 (2) 0 0 4096 24-Jan-2014 12:36 ..  11 40700 (2) 0 0 16384 24-Jan-2014 11:23 lost+found  7633 40755 (2) 0 0 4096 24-Jan-2014 12:35 dir1  22897 40755 (2) 0 0 4096 24-Jan-2014 12:35 dir2  12 100644 (1) 0 0 19 24-Jan-2014 12:36 testfile2  lsdel命令已无法显示删除的文件  debugfs: lsdel  Inode Owner Mode Size Blocks Time deleted  0 deleted inodes found.  debugfs: stat <13>  Inode: 13 Type: regular Mode: 0644 Flags: 0x0  Generation: 342839017 Version: 0x00000000  文件大小、硬链接数以及占用块数都被清空了  User: 0 Group: 0 Size: 0  File ACL: 0 Directory ACL: 0  Links: 0 Blockcount: 0  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52e1edeb -- Fri Jan 24 12:36:59 2014  atime: 0x52e1eda1 -- Fri Jan 24 12:35:45 2014  mtime: 0x52e1edeb -- Fri Jan 24 12:36:59 2014  dtime: 0x52e1edeb -- Fri Jan 24 12:36:59 2014  Size of extra inode fields: 4  BLOCKS:  最为关键的是，存储数据所在数据块的块号被清空了  debugfs: q  [root@localhost mountdir]# |

从上表可以看出，ext3在删除文件时做了如下操作：

这两点这正是ext3文件系统与ext2文件系统在删除文件时最重要的区别

1. 设置了删除时间i\_dtime域
2. 将硬链接数设置为0
3. 将文件大小(size)和占用块数（Blockcount）都设置为0
4. 清空存储文件数据的数据（i\_block数组）
5. 将父目录项中该文件对应的项中的索引节点号设置为0，并扩展前一项，使其包含该项所占用的空间

b）和c）两部分对于恢复被删除文件的用途至关重要。因为缺少了文件大小和数据块位置的信息，尽管文件数据依然完好的保存在磁盘上，但是没有任何线索能够说明这个文件的数据块被存储在哪个磁盘块上以及这些数据的相互顺序等信息。这也是使用debugfs的dump命令在ext3文件系统中并不能恢复已删除文件的原因。

这样，是否意味着ext3文件系统中删除的文件就无法恢复了呢？基于这样一个事实：“在删除文件时，并不会将文件数据真正从磁盘上删除”，因此可以采用一些技术手段来尝试恢复已删除的文件。

1. 解决方法

在删除文件的同时，除了记录文件名、磁盘设备、索引节点信息之外，把文件大小、数据块位置等重要信息也同时记录下来，这样就可以根据日志文件中的元数据信息完美地恢复文件了。

具体操作可以通过设置环境变量LD\_PRELOAD=\*\*\*.so 来捕捉一些底层的系统调用（如unlink、rmdir），修改删除文件的具体操作，把文件大小和数据位置这样的重要信息保存起来，这样就完全可以把删除的文件恢复出来。

这里推荐一个工具ext3grep可以完美地恢复所删除的文件，具体信息参考这里（<http://carlo17.home.xs4all.nl/howto/undelete_ext3.html>）

# 3 ext4文件系统

## 3.1 ext4产生的背景

1. 存储容量问题：随着硬盘存储容量越来越大以及在线重新调整特性的支持，ext3面临的可扩充性压力越来越大。在ext3文件系统中，如果使用4KB大小的数据库，所支持的最大文件系统为16TB，这是由于它使用了32位的块号所决定的（2^32 \* 4KB=16TB）。
2. 性能方面问题：ext3文件系统修复时（fsck）花费时间太长以及读写速度要慢于其他文件系统（如XFS、JFS）。

## 3.2 ext4的新特性

### 3.2.1 更大的文件系统

1. ext4采用48位的块号取代ext3原来的32位块号，这样可以增大文件系统的容量。如果使用4KB大小的数据块时，ext4可以支持最大2^48 \* 4KB=2^50KB=1EB的文件系统。
2. 将块号从32位修改为48位之后，存储元数据的结构都必须相应地发生变化，主要包括超级块和组描述符。如下所示（引用自Kernel的/root/fs/ext4/ext4.h）:

|  |
| --- |
| 207 /\*  208 \* Structure of a blocks group descriptor  209 \*/  210 **struct** [ext4\_group\_desc](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_group_desc&type=reference)  211 {  212 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions) [bg\_block\_bitmap\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_block_bitmap_lo&type=symbol); /\* Blocks bitmap block \*/  213 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions) [bg\_inode\_bitmap\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_inode_bitmap_lo&type=symbol); /\* Inodes bitmap block \*/  214 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions) [bg\_inode\_table\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_inode_table_lo&type=symbol); /\* Inodes table block \*/  215 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [bg\_free\_blocks\_count\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_free_blocks_count_lo&type=symbol);/\* Free blocks count \*/  216 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [bg\_free\_inodes\_count\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_free_inodes_count_lo&type=symbol);/\* Free inodes count \*/  217 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [bg\_used\_dirs\_count\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_used_dirs_count_lo&type=symbol); /\* Directories count \*/  218 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [bg\_flags](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_flags&type=symbol); /\* EXT4\_BG\_flags (INODE\_UNINIT, etc) \*/  219 [\_\_u32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u32&type=definitions) [bg\_reserved](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_reserved&type=symbol)[2]; /\* Likely block/inode bitmap checksum \*/  Most Significant Bit  220 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [bg\_itable\_unused\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_itable_unused_lo&type=symbol); /\* Unused inodes count \*/  221 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [bg\_checksum](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_checksum&type=symbol); /\* crc16(sb\_uuid+group+desc) \*/  222 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions) [bg\_block\_bitmap\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_block_bitmap_hi&type=symbol); /\* Blocks bitmap block MSB \*/  223 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [bg\_inode\_bitmap\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_inode_bitmap_hi&type=symbol); /\* Inodes bitmap block MSB \*/  224 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [bg\_inode\_table\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_inode_table_hi&type=symbol); /\* Inodes table block MSB \*/  225 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [bg\_free\_blocks\_count\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_free_blocks_count_hi&type=symbol);/\* Free blocks count MSB \*/  226 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [bg\_free\_inodes\_count\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_free_inodes_count_hi&type=symbol);/\* Free inodes count MSB \*/  227 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [bg\_used\_dirs\_count\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_used_dirs_count_hi&type=symbol); /\* Directories count MSB \*/  228 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [bg\_itable\_unused\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_itable_unused_hi&type=symbol); /\* Unused inodes count MSB \*/  229 [\_\_u32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u32&type=definitions" \o "Multiple defined in 6 places.) [bg\_reserved2](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=bg_reserved2&type=symbol)[3];  230 };  …  888 /\*  889 \* Structure of the super block  890 \*/  891 **struct** [ext4\_super\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_super_block&type=reference" \o "Multiple refered from 29 places.) {  892 /\*00\*/ [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_inodes\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_inodes_count&type=symbol); /\* Inodes count \*/  893 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_blocks\_count\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_blocks_count_lo&type=symbol); /\* Blocks count \*/  894 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_r\_blocks\_count\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_r_blocks_count_lo&type=symbol); /\* Reserved blocks count \*/  895 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_free\_blocks\_count\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_free_blocks_count_lo&type=symbol); /\* Free blocks count \*/  896 /\*10\*/ [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_free\_inodes\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_free_inodes_count&type=symbol); /\* Free inodes count \*/  897 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_first\_data\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_first_data_block&type=symbol); /\* First Data Block \*/  898 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_log\_block\_size](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_log_block_size&type=symbol); /\* Block size \*/  899 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) s\_obso\_log\_frag\_size; /\* Obsoleted fragment size \*/  …  962 /\* 64bit support valid if EXT4\_FEATURE\_COMPAT\_64BIT \*/  963 /\*150\*/ [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_blocks\_count\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_blocks_count_hi&type=symbol); /\* Blocks count \*/  964 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_r\_blocks\_count\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_r_blocks_count_hi&type=symbol); /\* Reserved blocks count \*/  965 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [s\_free\_blocks\_count\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=s_free_blocks_count_hi&type=symbol); /\* Free blocks count \*/  …  978 }; |

有上图可知：

1. 在ext4\_group\_desc结构体中引入了7个字段（红色字体），它们分别表示相应字段的高16位，将它们扩充到48位。
2. 在ext4\_super\_desc结构体中引入了3个字段（红色字体），它们分别表示相应字段的高32位，将它们扩充64位。

另外，由于日志中要记录所修改数据块的块号，因此JBD（Journaling Block Device）也需要相应地支持48位的块号。

1. 采用48位块号取代原来的32位块号之后，文件系统的最大值还受文件系统中最多块数的制约，这是由于ext3原来采用的结构决定的。回想一下，在ext3中，所有的块组描述符信息都被保存在第一个块组中，因此以缺省的128MB（2^27B）大小的块组为例，最多能够支持2^21个块组（由于ext4每个组描述符占64个字节，2^27B / 64B = 2^21），最大支持的文件系统大小为2^21 \* 2^27B = 2^48=256TB。为了解决这个问题，ext4采用了下面将介绍的元组块。

### 3.2.2 元组块

所谓元组块是指：将块组描述符存储在一个数据块中的一些连续块组。以128MB的块组（数据块为4KB）为例，ext4中每个元块组可以包括4096B / 64B = 64个块组（ext4中每个组描述符占64个字节），因此每个元组块的大小是64 \* 128MB = 8GB。

ext4文件系统的在磁盘上的存储布局如下所示：



元组块的使用，使得ext3和ext4的磁盘布局有了变化：

a）：ext3时超级块后紧跟的是变长的GDT块

b）：ext4时超级块（取决于是否是3、5、7的幂）后面跟的是定长（一个数据块大小）的GDT（且只存储在第一个、第二个和最后一个块组中）。

采用元块组的ext4文件系统的大小将不会受块组描述符的影响，并且这种设计对文件系统的扩展非常有利。当需要扩充文件系统的大小，可以在现有数据块之后新添数据块，并将这些数据块也按照元块组的方式进行管理即可。

当然，为了使用这些新增加的空间，在 superblock 结构中需要增加一些字段来记录相关信息（ext4\_super\_block 结构中增加了一个 s\_first\_meta\_bg 字段用来引用第一个元组块的位置）。

### 3.2.3 延迟分配

延迟分配是指：通过延迟使得竟可能多的相邻数据块一次被分配，这样既减少系统碎片又提高分配效率从而提高性能。通过一个示意图来说明，如下所示：

时间

ext3文件系统：



分配数据块但未写入数据

分配数据块且写入数据

ext4文件系统：

时间



注：在ext4文件系统中，延迟分配默认是有效的，使用挂载选项nodealloc可以禁用。

### 3.2.4 更多的子目录

在ext3文件系统中，一个目录最多只能包含32,000个子目录。在ext4文件系统中，取消了这个限制，一个目录可以包含任意多个子目录。

### 3.2.5 日志checksum

在ext3文件系统中，添加了日志功能。ext4文件系统则给日志数据添加了checksum功能，这样更加提高了系统的安全性和可靠性。

### 3.2.6 更快的文件系统检查

fsck命令在检查文件系统时有如下5个步骤：

1）Checking inodes、blocks and sizes

2）Checking directory structure

3）Checking directory connectivity

4）Checking reference counts

5）Checking group summary information

在ext3文件系统中，fsck在检查文件系统时会对整个磁盘进行扫描检查。在ext4文件系统中，允许fsck在检查中跳过磁盘中未使用的部分（主要为步骤一），因此加快了文件系统的检查。通过一个示意图来说明，如下所示：

ext3文件系统：



ext4文件系统：



### 3.2.7 纳秒级时间戳

在ext3文件系统中，时间戳的精确只能达到秒级。随着硬件性能的提升，这种精度已经无法区分在同一秒钟创建的文件的时间戳差异。在ext4文件系统中，通过扩充索引节点使得时间戳的精度达到了纳秒级的精度，如下所示（引用自Kernel的/root/fs/ext4/ext4.h）:

|  |
| --- |
| 524 /\*  525 \* Structure of an inode on the disk  526 \*/  527 **struct** [ext4\_inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_inode&type=reference) {  528 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [i\_mode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_mode&type=symbol); /\* File mode \*/  529 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions) [i\_uid](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_uid&type=symbol); /\* Low 16 bits of Owner Uid \*/  530 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_size\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_size_lo&type=symbol); /\* Size in bytes \*/  iflags标志i\_block\_lo是以扇区大小512B还是数据块大小为单位进行计算  531 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_atime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_atime&type=symbol); /\* Access time \*/  532 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_ctime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_ctime&type=symbol); /\* Inode Change time \*/  533 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_mtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_mtime&type=symbol); /\* Modification time \*/  534 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_dtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_dtime&type=symbol); /\* Deletion Time \*/  535 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [i\_gid](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_gid&type=symbol); /\* Low 16 bits of Group Id \*/  此数组的使用方式已改变，不再使用ext3那种间接寻址方式，而是用作extent树  536 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [i\_links\_count](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_links_count&type=symbol); /\* Links count \*/  537 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_blocks\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_blocks_lo&type=symbol); /\* Blocks count \*/  538 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_flags](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_flags&type=symbol); /\* File flags \*/  …  550 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_block&type=symbol)[[EXT4\_N\_BLOCKS](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/30889.html#L271)];/\* Pointers to blocks \*/  551 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_generation](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_generation&type=symbol); /\* File version (for NFS) \*/  新增的5个字段  …  578 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) i\_ctime\_extra; /\* extra Change time (nsec << 2|epoch) \*/  579 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) i\_mtime\_extra; /\* extra Modification time(nsec << 2|epoch) \*/  580 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) i\_atime\_extra; /\* extra Access time (nsec << 2|epoch) \*/  581 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_crtime](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_crtime&type=symbol); /\* File Creation time \*/  582 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) i\_crtime\_extra;/\* extra FileCreationtime (nsec << 2|epoch) \*/  583 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [i\_version\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=i_version_hi&type=symbol); /\* high 32 bits for 64-bit version \*/  584 }; |

### 3.2.8 extent

在ext4文件系统中引入了extent的概念来表示文件数据所在的位置。所谓extent就是描述保存文件数据使用的连续物理块的一段范围。每个extent都是一个ext4\_extent类型的结构，大小为12字节，如下所示（引用自Kernel的[root](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/files.html)/[fs](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/files/32849.html)/[ext4](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/files/32873.html)/ext4\_extents.h）：

|  |
| --- |
| 68 /\*  69 \* This is the extent on-disk structure.  70 \* It's used at the bottom of the tree.  71 \*/  72 **struct** [ext4\_extent](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_extent&type=reference" \o "Multiple refered from 41 places.) {  73 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [ee\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ee_block&type=symbol); /\* first logical block extent covers \*/  74 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [ee\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ee_len&type=symbol); /\* number of blocks covered by extent \*/  75 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [ee\_start\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ee_start_hi&type=symbol); /\* high 16 bits of physical block \*/  76 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [ee\_start\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ee_start_lo&type=symbol); /\* low 32 bits of physical block \*/  77 }; |

每个ext4\_extent结构可以表示该文件从ee\_block开始的ee\_len个数据块，它们在磁盘上的位置是从ee\_start\_hi<<32+ee\_start\_lo到ee\_start\_hi<<32+ee\_start\_lo+ee\_len-1，数据块全部是连续的。

注：ee\_len 是一个16位的无符号整数，但其最高位被在预分配特性中用来标识这个extent是否被初始化过了，因此一个extent可以表示 2^15个连续的数据块，如果采用4KB大小的数据块，就相当于 128MB。

如果文件大小超过了一个ext4\_extent结构能够表示的范围，或者其中有不连续的数据块，就需要使用多个ext4\_extent结构来表示。为此，还需要两个与extent密切相关的结构体来辅助，如下所示（引用自Kernel的[root](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/files.html)/[fs](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/files/32849.html)/[ext4](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/files/32873.html)/ext4\_extents.h）：

|  |
| --- |
| 79 /\*  80 \* This is index on-disk structure.  81 \* It's used at all the levels except the bottom.  82 \*/  83 **struct** [ext4\_extent\_idx](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_extent_idx&type=reference" \o "Multiple refered from 17 places.) {  84 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [ei\_block](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ei_block&type=symbol); /\* index covers logical blocks from 'block' \*/  85 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [ei\_leaf\_lo](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ei_leaf_lo&type=symbol); /\* pointer to the physical block of the next \*  86 \* level. leaf or next index could be there \*/  87 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [ei\_leaf\_hi](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ei_leaf_hi&type=symbol); /\* high 16 bits of physical block \*/  88 [\_\_u16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u16&type=definitions" \o "Multiple defined in 4 places.) ei\_unused;  89 };  90  91 /\*  表示在extent树中的位置，叶子节点该值为0，之上每层索引节点依次加1  92 \* Each block (leaves and indexes), even inode-stored has header.  93 \*/  94 **struct** [ext4\_extent\_header](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_extent_header&type=reference" \o "Multiple refered from 34 places.) {  95 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [eh\_magic](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=eh_magic&type=symbol); /\* probably will support different formats \*/  96 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [eh\_entries](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=eh_entries&type=symbol); /\* number of valid entries \*/  97 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [eh\_max](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=eh_max&type=symbol); /\* capacity of store in entries \*/  98 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [eh\_depth](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=eh_depth&type=symbol); /\* has tree real underlying blocks? \*/  99 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) eh\_generation; /\* generation of the tree \*/  100 }; |

ext4使用一个extent树结构来对文件的数据块进行寻址，其中：

a)：结构体ext4\_extent\_idx用来表示下一层的数据块位置且不位于最低层。

b）：结构体ext4\_extent\_header出现在每层的开头，用来表示本层的深度等相关信息。

c）：结构体 ext4\_extent用来表示文件数据块所在的磁盘位置。

用一个示意图来表示，如下所示：



在extent树中，节点一共有两类：

1）叶子节点：保存文件数据的磁盘块信息

2）索引节点：保存了下一层节点的位置

ext4文件系统采用extent树结构对文件进行寻址比ext3文件系统具有元数据减少、更加灵活等优点，用一个示意图来表示，如下所示：

ext3文件系统：



ext4文件系统：



关于使用extent树进行文件寻址需要注意以下几点：

1. 尽管索引节点中的i\_block[]字段保持不变，但对这个数组的使用方式已经改变了。其前3个元素一定是一个ext4\_extent\_header结构，后续每3个元素可能是一个ext\_extent或ext4\_extent\_idx，这取决于所表示文件的大小。
2. i\_block[]为一个60 字节的数组，最多可以保存一个 ext4\_extent\_header 结构以及4个ext4\_extent 结构。对于小文件来说，只需要一次寻址就可以获得保存文件数据块的位置；而超出此限制的文件只能通过遍历 extent 树来获得数据块的位置。对于连续存放的大文件来说，这种方法表示起来很高效，但是对于包含碎片非常多的文件或者稀疏文件来说，这种方法寻址起来要多次读盘。
3. 在ext3中i\_blocks是以扇区（512B）大为单位进行计算的，因此单个文件的最大值为2^32 \* 512B=2TB。而在ext4中i\_blocks\_lo可以以数据块大小为单位进行计算，因为单个文件的最大值可以扩充到16TB（数据块大小为4KB）。

注：为了避免需要对整个文件系统都需要进行类似转换，还引入了一个 EXT4\_HUGE\_FILE\_FL 标志，i\_flags 中不包含这个标志的索引节点的 i\_blocks\_lo 依然以 512 字节为单位进行计算。当文件所占用的磁盘空间大小增大到不能够用以512字节为单位的i\_blocks\_lo来表示时，ext4自动激活EXT4\_HUGE\_FILE\_FL标志，以数据块为单位重新计算i\_blocks\_lo的值。该转换是自动进行的，对用户透明。

## 3.3 ext4与ext3的关系

ext4引入了大量新功能，但最重要的是与ext3的向后和向前兼容性。为了最大程度的实现兼容性，ext4尽量保持索引节点、组描述符、目录项等结构不发生大的变化。

ext4与ext3是向前兼容的，这样就可以将ext4文件系统挂载为ext3文件系统。ext4与ext3是向后兼容的，这样就可以将ext3文件系统挂载为ext4文件系统。用一个示意图表示，如下所示：



有两点需要注意下：

1. 将ext3文件系统挂载为 ext4 时，内核开始使用特定于ext4文件系统的功能，如 extent。一旦在这个文件系统中写入文件之后，文件系统所使用的特性中就包含了extent特性，这样将导致不能再把文件系统挂载为 ext3 文件系统。

注：挂载时可以使用-O ^extent选项来禁用extent功能。

1. 尽管把 ext3 文件系统挂载为 ext4 能够实现特定于 ext4 的功能，但挂载本身并不把旧数据结构转换为新的格式。例如，现有文件保持块状方式分配， 而不是使用 extent 进行分配。

## 3.4 文件恢复

在ext4文件系统中删除文件时，不会真正修改存储文件数据所使用的磁盘数据块的内容，而是仅仅删除或修改了相关的元数据信息，使文件数据无法正常索引，从而实现删除文件的母的。因此，在ext4文件系统中恢复删除的文件也是完全可能的。

要想恢复已删除的文件需要关心的是文件在磁盘上的存储格式，以目录项的定义来看，如下所示（引用自Kernel的/root/fs/ext4/ext4.h）：

|  |
| --- |
| 1297 /\*  1298 \* Structure of a directory entry  1299 \*/  1300 #define [EXT4\_NAME\_LEN](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=EXT4_NAME_LEN&type=reference" \o "Multiple refered from 5 places.) 255  1301  1302 **struct** ext4\_dir\_entry {  1303 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/27662.html#L758); /\* Inode number \*/  1304 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [rec\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=rec_len&type=symbol); /\* Directory entry length \*/  1305 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [name\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=name_len&type=symbol); /\* Name length \*/  1306 **char** [name](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=name&type=definitions" \o "Multiple defined in 35 places.)[[EXT4\_NAME\_LEN](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/30889.html#L1300)]; /\* File name \*/  1307 };  1308  1309 /\*  1310 \* The new version of the directory entry. Since EXT4 structures are  1311 \* stored in intel byte order, and the name\_len field could never be  1312 \* bigger than 255 chars, it's safe to reclaim the extra byte for the  1313 \* file\_type field.  1314 \*/  1315 **struct** [ext4\_dir\_entry\_2](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=ext4_dir_entry_2&type=reference" \o "Multiple refered from 57 places.) {  1316 [\_\_le32](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le32&type=definitions" \o "Multiple defined in 2 places.) [inode](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/27662.html#L758); /\* Inode number \*/  1317 [\_\_le16](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__le16&type=definitions" \o "Multiple defined in 2 places.) [rec\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=rec_len&type=symbol); /\* Directory entry length \*/  1318 [\_\_u8](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u8&type=definitions" \o "Multiple defined in 4 places.) [name\_len](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=name_len&type=symbol); /\* Name length \*/  1319 [\_\_u8](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=__u8&type=definitions" \o "Multiple defined in 4 places.) [file\_type](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/30919.html#L68);  1320 **char** [name](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/cgi-bin/global.cgi?pattern=name&type=definitions" \o "Multiple defined in 35 places.)[[EXT4\_NAME\_LEN](http://ltech.fnst.cn.fujitsu.com/source/line/RHEL6.1GA/HTML/S/30889.html#L1300)]; /\* File name \*/  1321 }; |

从上表可以看出，ext4使用的目录项数据结构ext4\_dir\_entry\_2与ext3并没有根本的区别，这正是ext4和ext3互相兼容的基础。

ext4和ext3文件系统具有这么好的兼容性，那么文件的删除操作发生了变化吗？

对比查看一下ext4文件系统删除文件前后索引节点信息的变化，如下所示：

|  |
| --- |
| [root@localhost mountdir]# df -hT  Filesystem Type Size Used Avail Use% Mounted on  /dev/sda2 ext3 15G 14G 519M 97% /  tmpfs tmpfs 4.0G 0 4.0G 0% /dev/shm  /dev/sda1 ext3 965M 831M 85M 91% /boot  /dev/sde3 ext4 938M 18M 874M 2% /mountdir  [root@localhost mountdir]# debugfs /dev/sde3  debugfs 1.41.12 (17-May-2010)  debugfs: ls -l  2 40755 (2) 0 0 4096 11-Feb-2014 13:05 .  2 40755 (2) 0 0 4096 11-Feb-2014 13:05 ..  11 40700 (2) 0 0 16384 11-Feb-2014 13:00 lost+found  这个就是一会将删除的文件  7633 40755 (2) 0 0 4096 11-Feb-2014 13:05 dir1  7634 40755 (2) 0 0 4096 11-Feb-2014 13:05 dir2  13 100644 (1) 0 0 12 11-Feb-2014 13:05 testfile  14 100644 (1) 0 0 22 11-Feb-2014 13:05 testfile2  debugfs: stat <2>  Inode: 2 Type: directory Mode: 0755 Flags: 0x0  Generation: 0 Version: 0x00000000:00000010  User: 0 Group: 0 Size: 4096  File ACL: 0 Directory ACL: 0  Links: 5 Blockcount: 8  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52f9afa0:dd160f78 -- Tue Feb 11 13:05:36 2014  atime: 0x52f9afb6:b2e2d658 -- Tue Feb 11 13:05:58 2014  mtime: 0x52f9afa0:dd160f78 -- Tue Feb 11 13:05:36 2014  crtime: 0x52f9ae65:00000000 -- Tue Feb 11 13:00:21 2014  Size of extra inode fields: 28  BLOCKS:  bs=4096是因为根目录占一个数据块的大小  根目录所对应的数据块号为169  (0):69  TOTAL: 1  debugfs: q  [root@localhost mountdir]# dd if=/dev/sde3 of=/home/renyl/testdir/block.69.orig bs=4096 count=1 skip=69  1+0 records in  这个是之前介绍的读取目录项的程序  1+0 records out  4096 bytes (4.1 kB) copied, 8.2647e-05 s, 49.6 MB/s  [root@localhost mountdir]# cd /home/renyl/testdir/  [root@localhost testdir]# ./ read\_dir\_entry block.69.orig 4096  offset | inode\_number | rec\_len | name\_len | file\_type | name  ===============================================================  0: 2 12 1 2 .  12: 2 12 2 2 ..  24: 11 20 10 2 lost+found  44: 7633 12 4 2 dir1  56: 7634 12 4 2 dir2  68: 13 40 8 1 testfile  108: 12 24 14 1 .testfile2.swp  132: 14 3988 9 1 testfile2  重新挂载时确保删除操作同步到磁盘上的数据块中  [root@localhost testdir]# rm -rf /mountdir/testfile  [root@localhost testdir]# umount /mountdir/  [root@localhost testdir]# mount /dev/sde3 /mountdir/  [root@localhost testdir]# debugfs /dev/sde3  lsdel命令已无法显示被删除的文件  debugfs: lsdel  Inode Owner Mode Size Blocks Time deleted  0 deleted inodes found.  debugfs: stat <13>  Inode: 13 Type: regular Mode: 0644 Flags: 0x80000  Generation: 3131709158 Version: 0x00000000:00000001  文件大小、硬链接数和占用块数都被清空  User: 0 Group: 0 Size: 0  File ACL: 0 Directory ACL: 0  Links: 0 Blockcount: 0  Fragment: Address: 0 Number: 0 Size: 0  ctime: 0x52f9b2a2:21b06828 -- Tue Feb 11 13:18:26 2014  atime: 0x52f9af84:d4fade54 -- Tue Feb 11 13:05:08 2014  mtime: 0x52f9b2a2:21b06828 -- Tue Feb 11 13:18:26 2014  数据块所在位置被清空  设置删除时间  crtime: 0x52f9af84:d4fade54 -- Tue Feb 11 13:05:08 2014  dtime: 0x52f9b2a2 -- Tue Feb 11 13:18:26 2014  Size of extra inode fields: 28  试着恢复文件testfile  EXTENTS:  debugfs: dump <13> testfile.bak  debugfs: q  可以看出，恢复失败。因为文件大小、位置都被清空了，debugfs无法进行恢复。  [root@localhost testdir]# cat testfile. bak  [root@localhost testdir]# ls -al testfile.bak  -rw-r--r-- 1 root root 0 Feb 11 14:29 testfile.bak  [root@localhost testdir]# dd if=/dev/sde3 of=block.69.deleted bs=4096 count=1 skip=69  [root@localhost testdir]# ./a.out block.69.deleted 4096  offset | inode\_number | rec\_len | name\_len | file\_type | name  testfile文件的目录项长度被合并到前一项上  ==================================================================  0: 2 12 1 2 .  12: 2 12 2 2 ..  24: 11 20 10 2 lost+found  44: 7633 12 4 2 dir1  该文件的目录项没有发生任何变化  56: 7634 52 4 2 dir2  108: 13 40 8 1 testfile  148: 12 24 14 1 .testfile2.swp  172: 14 3988 9 1 testfile2  [root@localhost testdir]# |

由上图可知：

当目录项被删除时，该目录项的所有相关信息都没有改变，只是将该目录项的空间合并到上一个目录项中，这使得在恢复文件时，文件名可以通过查找目录项中匹配的索引节点号得以正确恢复。

尽管ext4文件系统是基于extent来管理空间的，但是其恢复原理同ext3文件系统一致。

在删除文件的同时，除了记录文件名、磁盘设备、索引节点信息之外，把文件大小、数据块位置等重要信息也同时记录下来，这样就可以根据日志文件中的元数据信息完美地恢复文件了。

具体操作可以通过设置环境变量LD\_PRELOAD=\*\*\*.so 来捕捉一些底层的系统调用（如unlink、rmdir），修改删除文件的具体操作，把文件大小和数据位置这样的重要信息保存起来，这样就完全可以把删除的文件恢复出来。

# 4 XFS文件系统

# 5 Btrfs文件系统