**Ext4文件系统架构分析(一)**

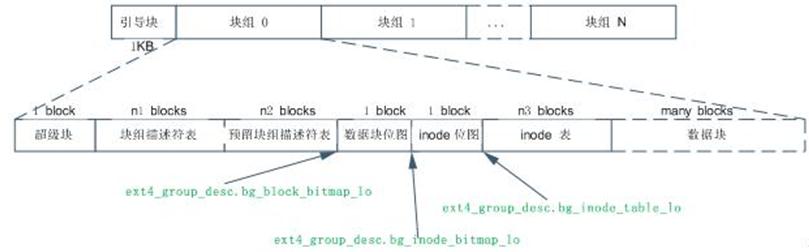
Balloc.c (kernel\fs\ext4)

本文描述Ext4文件系统磁盘布局和元数据的一些分析，同样适用于Ext3和Ext2文件系统，除了它们不支持的Ext4的特性外。整个分析分两篇博文，分别概述布局和详细介绍各个布局的数据结构及组织寻址方式等。感兴趣的看官敬请留意和指导！

## 1. Ext4文件系统布局综述

一个Ext4文件系统被分成一系列块组。为减少磁盘碎片产生的性能瓶颈，块分配器尽量保持每个文件的数据块都在同一个块组中，从而减少寻道时间。以4KB的数据块为例，一个块组可以包含32768个数据块，也就是128MB。

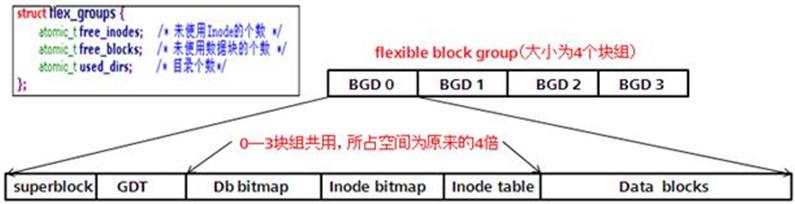
## 1.1 磁盘布局

Ext4文件系统的标准磁盘布局如下：  


Ext4文件系统主要使用块组0中的超级块和块组描述符表，在其他一些特定块组中有超级块和块组描述符表的冗余备份。如果块组中不含冗余备份，那么块组就以数据块位图开始。当格式化磁盘成为Ext4文件系统的时候，mkfs将在块组描述符表后面分配预留GDT表数据块(“Reserve GDT blocks”)以用于将来扩展文件系统。紧接在预留GDT表数据块后的是数据块位图与inode表位图，这两个位图分别表示本块组内的数据块与inode表的使用，inode表数据块之后就是存储文件的数据块了。在这些各种各样的块中，超级块、GDT、块位图、Inode位图都是整个文件系统的元数据，当然inode表也是文件系统的元数据，但是inode表是与文件一一对应的，我更倾向于将inode当做文件的元数据，因为在实际格式化文件系统的时候，除了已经使用的十来个外，其他inode表中实际上是没有任何数据的，直到创建了相应的文件才会分配inode表，文件系统才会在inode表中写入与文件相关的inode信息。

## 1.2 Flexible 块组（flex\_bg）

Flexible 块组（flex\_bg）是从Ext4开始引入的新特性。在一个flex\_bg中，几个块组在一起组成一个逻辑块组flex\_bg。Flex\_bg的第一个块组中的位图空间和inode表空间扩大为包含了flex\_bg中其他块组上位图和inode表。

比如flex\_bg包含4个块组，块组0将按序包含超级块、块组描述符表、块组0-3的数据块位图、块组0-3的inode位图、块组0-3的inode表，块组0中的其他空间用于存储文件数据。同时，其他块组上的数据块位图、inode位图、inode表元数据就不存在了，但是SB和GDT还是存在的。  
  
Flexible块组的作用是：

(1)  聚集元数据，加速元数据载入；

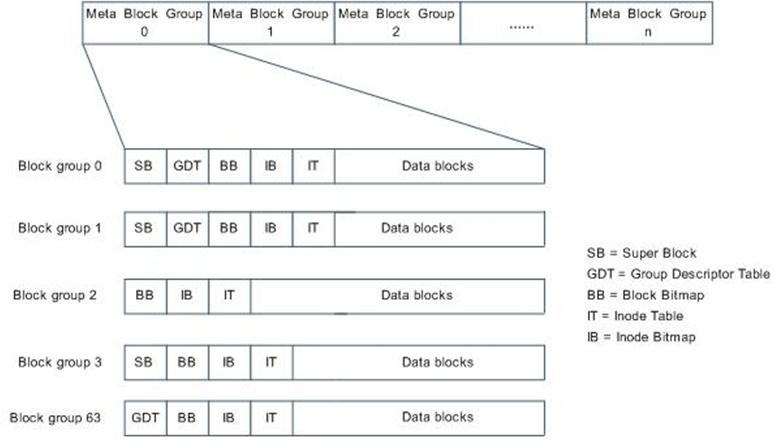
(2)  使得大文件在磁盘上尽量连续；

即使开启flex\_bg特性，超级块和块组描述符的冗余备份仍然位于块组的开头。 Flex\_bg中块组的个数由2^ext4\_super\_block.s\_log\_groups\_per\_flex 给出。

## 1.3 元块组（Meta Block Groups）

通常，在每个冗余备份的超级块的后面是一个完整的(包含所有块组描述符的)块组描述符表的备份。这样会产生一个限制，以Ext4的块组描述符大小64 Bytes计算，文件系统中最多只能有2^21个块组，也就是文件系统最大为256TB。

使用元块组Meta Block Groups特性，每个块组都包含该块组自己的描述符的冗余备份。因此可以创建2^33个块组，也就是文件系统最大1EB。48位数据块，每个块组128MB，因而可以创建2^33个块组。

元块组实际上是可以用一个块组描述符块来进行描述的块组集，简单的说，它由一系列块组组成，同时这些块组对应的块组描述符存储在一个块中。它的出现使得Ext3和Ext4的磁盘布局有了一定的变化，以往超级块后紧跟的是变长的GDT块，现在是超级块依然决定于是否是3,5,7的幂，而一个块组描述符块则存储在元块组的第一个，第二个和最后一个块组的开始处(见下图)  
  
在两种情况下我们可能会用到这种新布局：

(1)  文件系统创建时。用户可以指定使用这种布局。

(2)  当文件系统增长而且预留的组描述符块耗尽时。目前超级块中有一个域s\_first\_meta\_bg用于描述第一个使用元块组的块组。

 当增加新块组时，我们不需要给组描述符表预留空间，而是在当前文件系统后面直接添加新的元块组就可以了。

## 1.4 Lazy 块组初始化

如果块组中的相应标志已设置，那么块组中的inode位图和inode表将不被初始化。这样可以减少mkfs时间，如果开启了块组描述符校验和功能，甚至连块组都可以不初始化。

## 1.5 特殊inodes

Ext4预留了一些inode做特殊特性使用，见下表：

表 1  Ext4的特殊inode

Inode号    用途

0      不存在0号inode

1      损坏数据块链表

2      根目录

3      ACL索引

4      ACL数据

5      Boot  loader

6      未删除的目录

7      预留的块组描述符inode

8      日志inode

11     第一个非预留的inode，通常是lost+found目录

## 1.6 数据块和Inode分配策略

在机械磁盘上，保持相关的数据块相互接近可以总的磁头移动时间，因而可以加速磁盘IO。在SSD上虽然没有磁头转动，数据局部性可以增加每次IO请求的传输的数据大小，因而减少响应IO请求的传输次数。数据的局部性对单个擦除块的写入产生影响，可以加速文件重写的速度。因而尽可能减少碎片是必要的。inode和数据块的分配策略可以保证数据的局部集中。以下为inode和数据块的分配策略：

(1)  多块分配可以减少磁盘碎片。当文件初次创建的时候，块分配器预测性地分配8KB的磁盘空间给文件。当文件关闭的时候，未使用的空间当然也就释放了。但是如果推测是正确的，那么文件数据将写到一个多个块的extent中。

(2)  延迟分配。当一个文件需要更多的数据块引起写操作时，文件系统推迟决定新数据在磁盘上的存放位置，直到脏的buffer写到磁盘为止。

(3)  尽量保持文件的数据块与其inode在同一个块组中。可以减少磁盘寻道时间.

(4)  尽量保持同一个目录中的所有inodes与目录位于同一个块组中。这样的假设前提是一个目录中的文件是相关的。

(5)  磁盘卷被分成128MB的块组。当在根目录中创建目录时，inode分配器扫描块组并将新目录放到它找到的使用负荷最小的块组中。这可以保证目录在磁盘上的分散性。

(6)  即使上述机制无效，仍然可以使用e4defrag整理碎片文件。

**Ext4文件系统架构分析(二)**

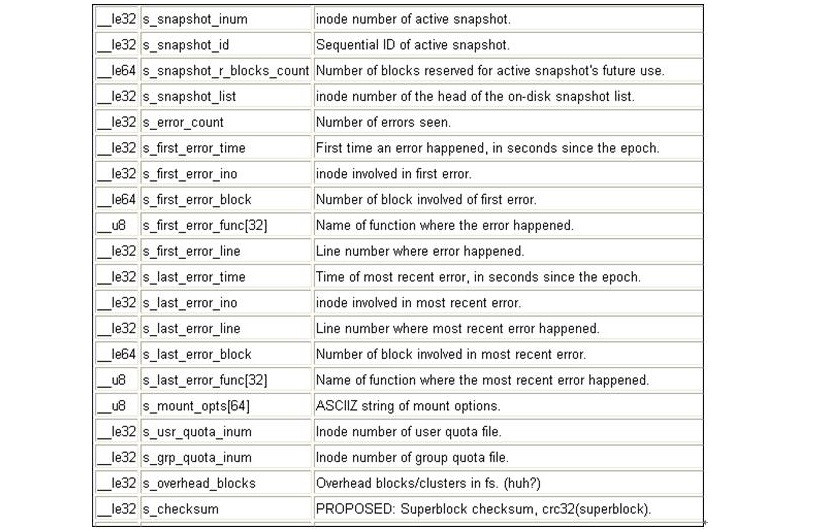
**1.7 超级块**

超级块记录整个文件系统的大量信息，如数据块个数、inode个数、支持的特性、管理信息，等待。

如果设置sparse\_super特性标志，超级块和块组描述符表的冗余备份仅存放在编号为0或3、5、7的幂次方的块组中。如果未设置sparse\_super特性标志，冗余备份存在与所有的块组中。以下是2.6.32.18内核中对Ext4超级块的描述：

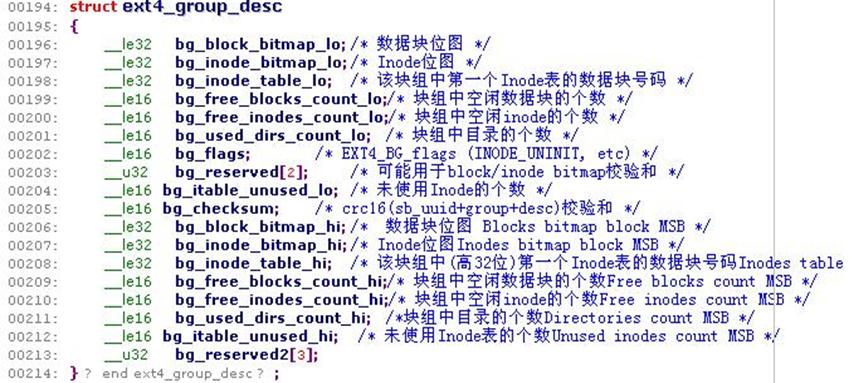
  
  


3.0的内核中，Ext4的超级块加入了以下相关元数据：快照、文件系统错误处理相关、挂载选项、配额文件inode、超级块校验和等，见下图。目前没有深入研究这些新的元数据。



**1.8 块组描述符**

一个块组中，具有固定位置的数据结构是超级块和块组描述符。其他数据结构位置都可以不固定。Flex\_bg机制使用这个性质将几个块组聚合成一个flex块组，将flex\_bg中所有位图和inode 表放到flex\_bg的第一个块组中。详细情况可以参考我的上一篇Ext4分析博文的Flexible 块组（flex\_bg）部分。

  
      如果设置了meta\_bg特性标志，几个块组结合成一个meta group。在meta\_bg的情况下，在meta group中的第一个和最后两个块组中仅包含meta group中的块组的块组描述符。Flex\_bg和Meta\_bg互斥因而不能共同出现。

**1.9 数据块位图与inode位图**

数据块位图跟踪块组中数据块使用情况。Inode位图跟踪块组中Inode使用情况。每个位图一个数据块，每一位用0或1表示一个块组中数据块或inode表中inode的使用情况。如果一个数据块大小是4KB的话，那一个位图块可以表示4\*1024\*8个数据块的使用情况，这也是单个块组具有的最大数据块个数。这样可以算出一个块组大小是128MB。当然一个位图块也可以表示4\*1024\*8个inode的使用情况，但是实际上一个块组中即使存满了文件，也不会用到这么多的inode，因为实际系统中基本不会出现所有文件大小都小于等于1个数据块大小的情况。实际上一个块组中有多少个inode，在块组描述符中是确定的，在文件系统格式化过程中也会看到这个数值，如果没记错的话，大概是每4个还是8个数据块分配一个inode空间。

**1.10 Inode表**

为了找到与一个文件相关的信息，必须遍历目录文件找到与文件相关的目录项，然后加载inode找到该文件的元数据。Ext4在目录项中用一位存储了文件类型(通常存储在inode中)的拷贝，这对性能提升有益。Inode表的大小为ext4\_super\_block.s\_inode\_size \* ext4\_super\_block.s\_inodes\_per\_group Bytes。   
  
  
       
 Ext4的inode的数据结构大小为156 bytes，但是Ext4的标准inode的大小是256 bytes。

**1.11 查找inode**

每个块组包含ext4\_super\_block.s\_inodes\_per\_group个inodes。因为0号inode不存在，可以通过如下的算式计算inode所在的块组：

bg=(inode\_num -1)/ ext4\_super\_block.s\_inodes\_per\_group

inode在块组中inode表中的索引index可以通过如下的算式计算：

index=(inode\_num -1) % ext4\_super\_block.s\_inodes\_per\_group

inode在inode表中的地址偏移为：

offset=index \* ext4\_super\_block.s\_inode \_size

**1.12 inode.i\_block0[]s的内容**

取决于文件类型，inode.i\_blocks[]使用的方式不同。一般来说，常规文件和目录用inode.i\_blocks[]作为文件数据块索引信息，特殊文件将inode.i\_blocks[]用于特殊用途。常规文件用inode.i\_blocks[]作为文件数据块索引信息的三级索引结构会在后面直接、间接块地址中详细介绍。

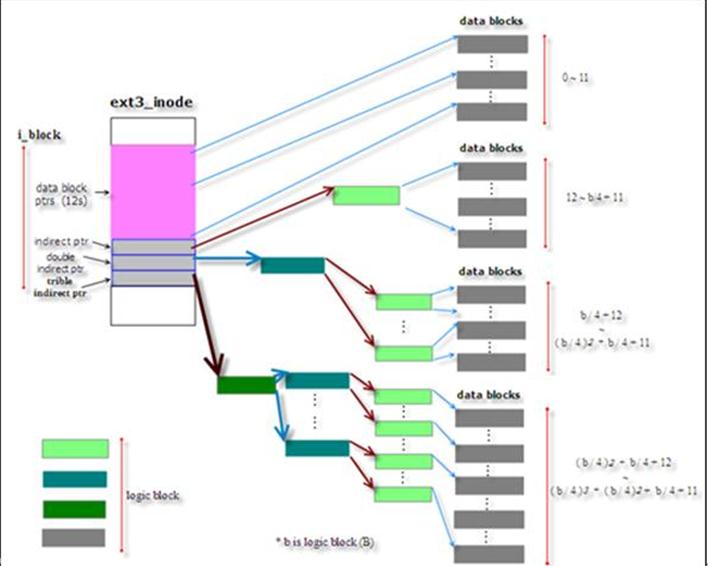
**1.13  符号链接**

如果符号链接的目标字符串长度小于60字节，那么就将其存储在inode.i\_blocks[]中，inode中inode.i\_blocks[]占据的大小刚好是60KB。这里要注意到的是，有些文件其内容是跟文件的元数据放在一起的，因而就没有了数据块。也就是说不是每个文件数据都必然占据着一个数据块。

**1.14 直接/间接块地址**

 Ext2/Ext3中数据块映射方式如下表





**1.15   Extent 树**

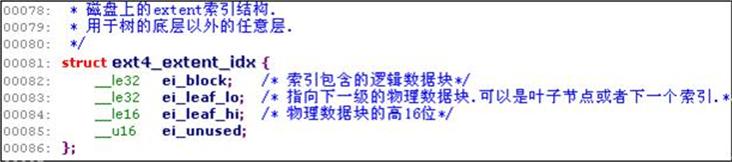
Ext4中用extent树代替了逻辑块映射。使用extents，用一个struct ext4\_extent结构就可以映射多个数据块，减少元数据块的使用。如果设置了flex\_bg，甚至可以用一个extent分配一个非常大的文件。使用extent特性，inode必须设置extents flag。

Extents以树的方式安排。Extent树的每个节点都以一个ext4\_extent\_header开头，如果节点是内部节点(ext4\_extent\_header.eh\_depth>0)，ext4\_extent\_header后面紧跟的是ext4\_extent\_header .eh\_entries个索引项struct ext4\_extent\_idx，每个索引项指向该extent树中一个包含更多的节点的数据块。如果节点是叶子节点(ext4\_extent\_header.eh\_depth==0)，ext4\_extent\_header后面紧跟的是ext4\_extent\_header .eh\_entries个struct ext4\_extent数据结构。这些ext4\_extent结构指向文件数据块。Extent树的根结点存储在inode.i\_blocks中，可以存储文件的前4个extents而不需额外的元数据块。

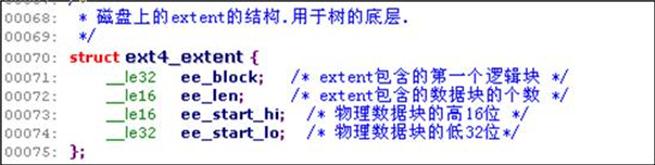
ext4\_extent\_header：

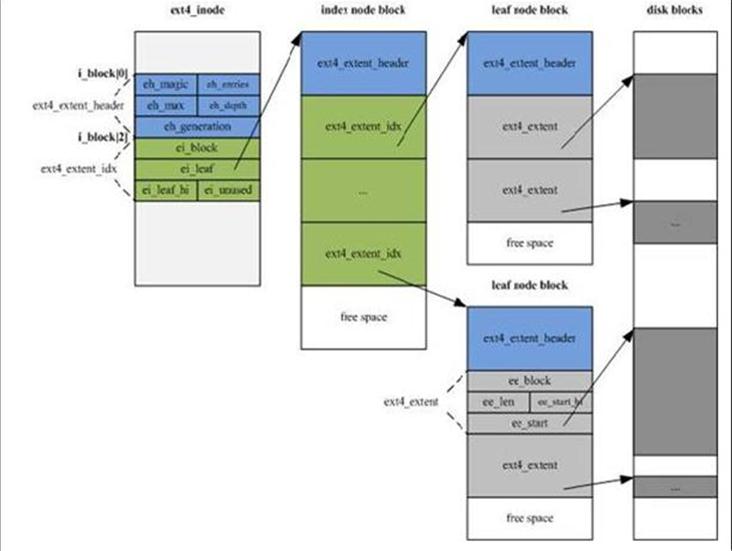


struct ext4\_extent\_idx：extent树的内部节点，也称为索引节点。



ext4\_extent：extent树的叶子节点。





**1.16 Extent树数据块校验和：可能加入的新元数据**

由于extent树的根在inode中，因而Extent树数据块指extent树的除根据节点外的所有内部节点和叶子节点。Extent的树根节点和叶子节点的数据块中存储完xt4\_extent\_idx和xt4\_extent数据结构后至少会留下4 ((2^x%12)>=4) bytes的空间。因而可以加入一个结构struct ext4\_extent\_tail，其中存储32位的校验和。位于inode中的4个extents无需校验和，因为inode已经做了校验和。

**1.17 目录项**

Ext4文件系统中，一个目录差不多是一个平面文件，映射任意长度的字符串到文件系统中的一个inode。文件系统中存在多个目录项引用同一个inode——硬链接，这也是硬链接不能链接其他文件系统中的文件的原因。

**1.18 线性（经典）目录**

缺省地，目录文件中包含一个线性的目录项数组。未使用的目录项标记为inode =0。Ext4文件系统默认地使用struct ext4\_dir\_entry\_2记录目录项，除非没有设置filetype特性标志。在没有设置filetype特性标志的情况下，使用struct ext4\_dir\_entry记录目录项。  
  