**Ext2文件系统分析**

本文以Ext2文件系统为例分析了具体文件系统的磁盘数据结构，并且分析了从磁盘数据结构建立虚拟文件系统超级块、索引节点、目录项的过程，以及这些结构之间的关系。并介绍了Linux文件系统架构，最后大致介绍了页高速缓存，关于对页高速缓存的详细分析请查看页高速缓存的分析报告。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | 版本 | 修改内容简介 | 修改日期 | 修改人 |
| 1 | V1.0 | 完成ext2文件系统系分析 | 2016-03-22 | 李小龙 |



**目录**

[一、 1](#_Toc865952274)

[一、ext2磁盘数据结构 2](#_Toc1280447894)

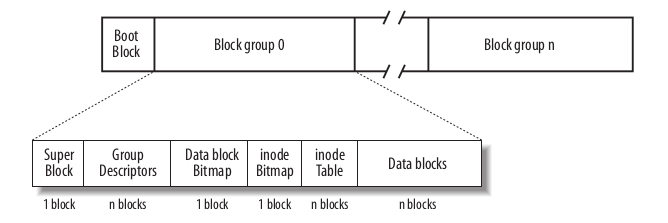
[二、 ext2数据机构和vfs数据结构的关系 4](#_Toc1500267165)

[三、 Linux文件系统架构 7](#_Toc981521196)

[四、 页高速缓存 8](#_Toc636776892)

**一、ext2磁盘数据结构**

ext2磁盘数据结构如下图所示，第0块是引导块，其后面的块被分为多个块组，每个块组中第一块是超级块，接下来n块是组描述符，然后是数据块位图和索引节点位图分别占一个块，然后是n块索引节点表，最后是数据块。



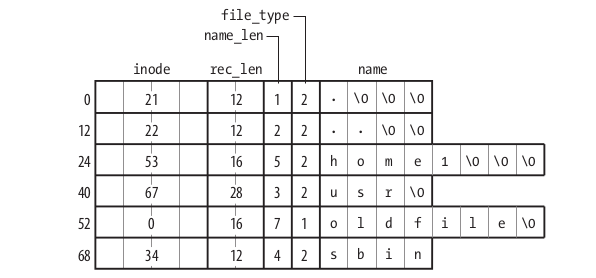
1. ext2\_super\_block：磁盘超级块和vfs超级块是完全不同的数据结构，当然创建vfs超级块时要依赖磁盘超级块。磁盘超级块中保存有文件系统中索引节点个数、文件系统块的个数、块大小（2的幂次方)、每组的块数、片数、索引节点数等。
2. ext2\_group\_desc：每个组块都保存一所有块组描述符，与该组相对应的快组描述符保存有该组中数据块位图所在的块号、inode位图所在块号、索引节点表的起始块号，另外还保存有空闲块、空闲索引节点个数以及组中目录个数等。
3. 索引节点表保存ext2\_inode：索引节点表保存一系列索引节点，每个索引节点128字节，其中主要保存文件的属性以及访问控制信息，还保存有文件的数据块所占块数，以及文件数据块块号。
4. 数据块位图和inode位图用于记录数据块和索引节点表的分配情况。

问题1：虚拟文件系统的inode结构中保存有inode节点号，如何才能找到对应的磁盘块号？

问题2：文件名和目录名分别保存在哪里？在ext2中目录也是文件的具体含义？

注：

1. 在ext2文件系统中文件（或目录）名和文件属性以及文件的数据存放在不同的数据结构中，因此也保存在不同的磁盘块中。文件名保存在目录对应的数据块中，文件属性保存在ext2\_inode节点中，文件数据保存在ext2\_inode的iblock字段指向的数据块中。
2. ext2文件系统中目录也是一个文件，只不过该目录对应的数据块中保存的是文件名（或目录名）和索引节号、文件类型和文件名长度组成的目录项ext2\_dir\_entry\_2结构。该结构如下图所示：



问题2：如何知道文件名对应的索引节点号？

问题3：存不存在0号索引节点？

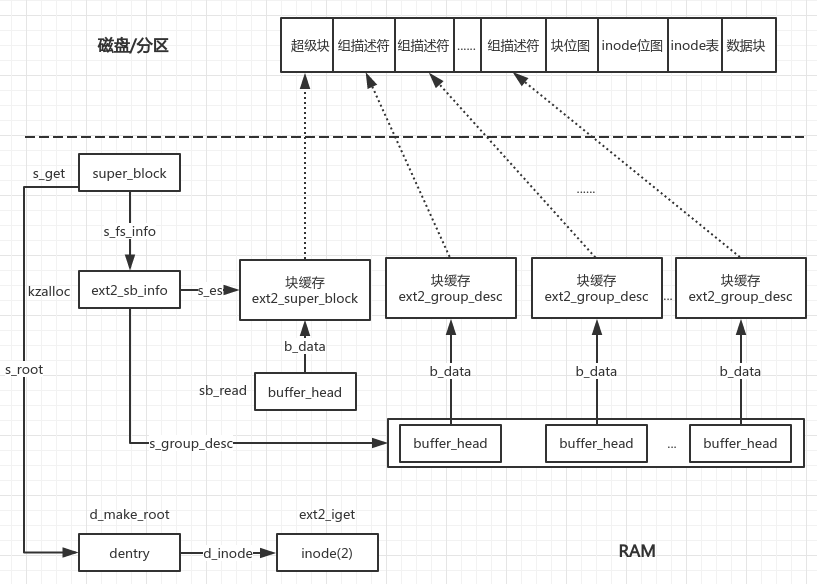
问题4：索引节点号对于vfs是唯一的吗？

1. **ext2数据机构和vfs数据结构的关系**

vfs文件系统是对具体文件系统的抽象，当创建vfs数据结构时，需要从磁盘读入相关的数据来初始化vfs数据结构，并且为了减小对磁盘的访问，内核将频繁读取的数据保存在高速缓存中。对磁盘数据的读取总是以块为单位进行的，读取到的数据首先是保存在缓冲区页中的。

1. 超级块结构之间的关系

super\_block的创建是在挂载文件系统时进行的，下图反应了从磁盘读入磁盘超级块ext2\_super\_block并创建vfs超级块super\_block的过程及其数据结构之间的关系。



在挂载文件系统的过程中，会调用sget()函数创建一个vfs超级块super\_block（假设该磁盘没有被挂载过，即内存中不存在其对应的超极块、根目录dentry、和根目录inode），接着在后续过程中会调用到kzalloc创建一个ext2\_sb\_info，这个数据结构是ext2磁盘超级块ext2\_super\_block的内存数据结构。ext2\_super\_block由sb\_read()函数从磁盘读入到页高速缓存的块缓存中，并创建buffer\_head的缓冲区首部，其中的b\_data字段指向ext2\_super\_block所在的块缓存，之后会用ext2\_super\_block的内容设置ext2\_sb\_info中对应的大部分字段，s\_group\_desc是一个buffer\_head类型的指针数组，每个bufer\_head的b\_data字段指向从磁盘读入的组描述符所在的块缓存。之后还要为超级块创建根目录dentry和与其对应的inode节点（根目录对应的inode节点号是2）。

1. vfs索引节点inode的创建

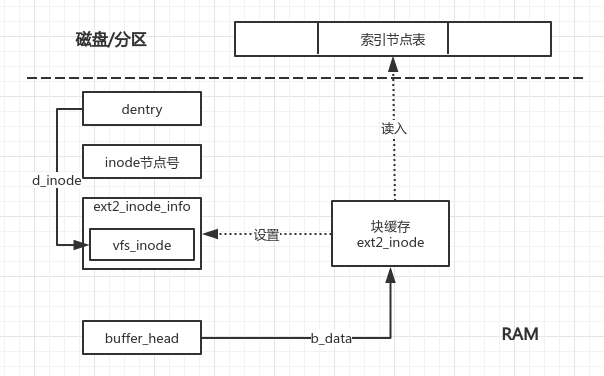
根据第一部分的内容我们知道确定一个文件在磁盘中的外置的大概过程是：文件名->inode节点号->磁盘/分区块。例如：在usr/目录下查找bin/目录（假设我们已经找到了/usr对应的目录项，并且其索引节点已经建立并与之关联），首先找到usr对应的inode，由于是目录，调用inode->i\_op->lookup()函数，该函数在ext2文件系统中指向ext2\_lookup函数，ext2\_lookup函数调用ext2\_inode\_by\_name()函数获得name是bin的ext2\_dir\_entry\_2结构的inode节点号（参考第一部分），然后调用ext2\_iget()函数确定该inode节点号的ext2\_inode结构所在的号所在磁盘的块号，并将其读入页高速缓存中。

上述过程涉及两个查找：

①、在目录数据块所在的缓存中查找名为name的文件名的inode节点。

②、根据inode节点号获取ext2\_inode节点所在的数据块。

下图是从磁盘建立inode节点的过程：



注：

①、在路径查找过程中如果所查找的分量不在目录项高速缓存中，则建立dentry，接着还要根据磁盘中有无该文件建立对应的inode节点。

②、现在我们知道该文件的文件名，接下来就是要得到该文件名对应的inode节点号。通过父目录dentry找到父目录dentry对应的inode节点，调用inode节点的lookup()操作函数，对于ext2文件系统该函数是ext2\_lookup()。

③、ext2\_lookup()函数调用ext2\_inode\_by\_name()函数查找目录名为要查找的路径分量的inode节点号。如果父目录dentry对应的数据块不在页高速缓存中，首先会从磁盘读取该数据块到页高速缓存中，然后在页高速缓存中查找，最后返回目标文件的inode节点号。

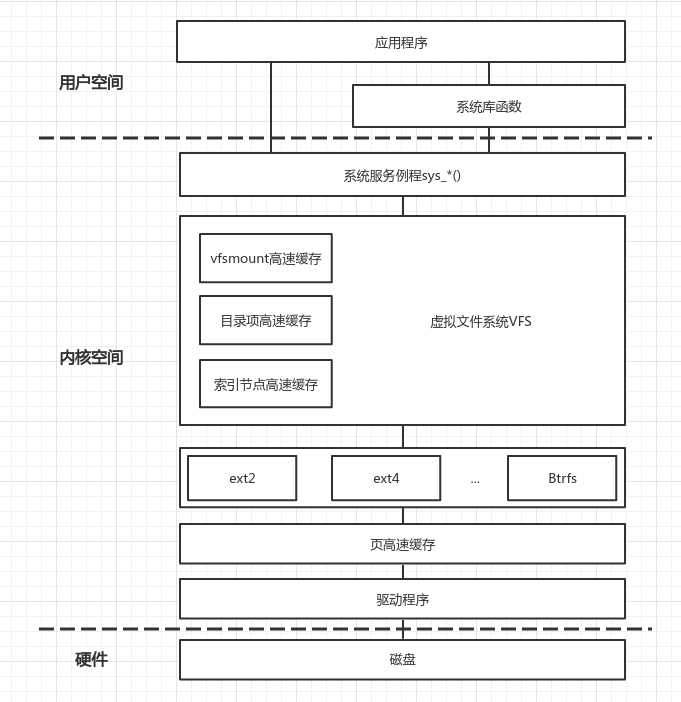
④、ext2\_lookup()函数根据获得的inode节点号，调用ext2\_iget()函数获取该inode节点号对应的inode节点。ext2\_iget()首先在inode高速缓存中查找inode，找不到时就会读取磁盘中inode表到页高速缓存中，并根据其建立一个新的inode节点，然后返回。

⑤、调用d\_splice\_alias()函数将新建立的dentry和inode节点相关联。

1. **Linux文件系统架构**

经过之前对vfs文件系统的分析和本文对ext2文件系统的分析我们得出了整个Linux文件系统的架构图。根据文件系统的架构图，以及之前对文件系统的分析，得出对文件系统性能有较大影响的几个点：

1. VFS路径查找过程。
2. 页高速缓存的管理。
3. 文件在磁盘上的块分布。



1. **页高速缓存**

页高速缓存和dentry、inode、vfsmount缓存一样也是一种磁盘高速缓存机制，只是其采用不同的管理机制，是提高文件系统速度的关键。与dentry、inode、vfsmount高速缓存不一样的一点是，页高速缓存由具体文件系统使用，而不是直接被vfs层所使用。

**附：散列表查找**

struct hlist\_head \*head = m\_hash(mnt, dentry);

static inline struct hlist\_head \*m\_hash(struct vfsmount \*mnt, struct dentry \*dentry)

{

unsigned long tmp = ((unsigned long)mnt / L1\_CACHE\_BYTES);

tmp += ((unsigned long)dentry / L1\_CACHE\_BYTES);

tmp = tmp + (tmp >> m\_hash\_shift);

return &mount\_hashtable[tmp & m\_hash\_mask];

}

struct hlist\_head \*head = inode\_hashtable + hash(sb, ino);

struct hlist\_bl\_head \*b = d\_hash(parent, hashlen\_hash(hashlen));

static inline struct hlist\_bl\_head \*d\_hash(const struct dentry \*parent,

unsigned int hash)

{

hash += (unsigned long) parent / L1\_CACHE\_BYTES;

return dentry\_hashtable + hash\_32(hash, d\_hash\_shift);

}

【参考】

1、http://blog.csdn.net/sanwenyublog/article/details/50830957