

struct address\_space\*i\_mapping

inode

struct address\_space i\_data

address\_space

struct inode\*host

struct request \*last\_merge;

elevator\_t \*elevator;

struct request\_list rq;

request\_queue

struct list\_head queue\_head;

block\_device

page

**union struct {**

**unsigned long private;**

**struct address\_space \*mapping;**

**};**

**union {**

**pgoff\_t index**

**void \*freelist**

**};**

buffer\_head

**struct buffer\_head \*b\_this\_page**

**struct page \*b\_page**

**char \*b\_data;**

**struct block\_device \*b\_bdev**

0xc051bf00

0xc051c100

0xc051c300

0xc051c500

gendisk

request

struct list\_head queuelist

struct bio \*bio

struct bio \*biotail

bio

biotail

bio

biotail

bio

biotail

bio

biotail

struct bio

struct bio

struct bio

struct bio

unsigned short bi\_idx

struct bio\_vec \*bi\_io\_vec

struct cause\_list\_list \*cll

struct bio

struct page \*bv\_page;

unsigned int bv\_len;

unsigned int bv\_offset;

bio\_vec

page

page

page

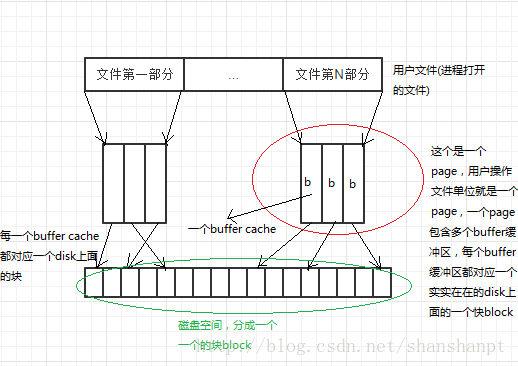
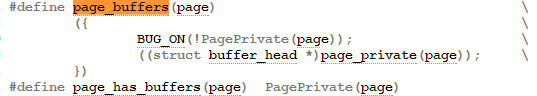
page

bio\_vec

bio\_vec

bio\_vec

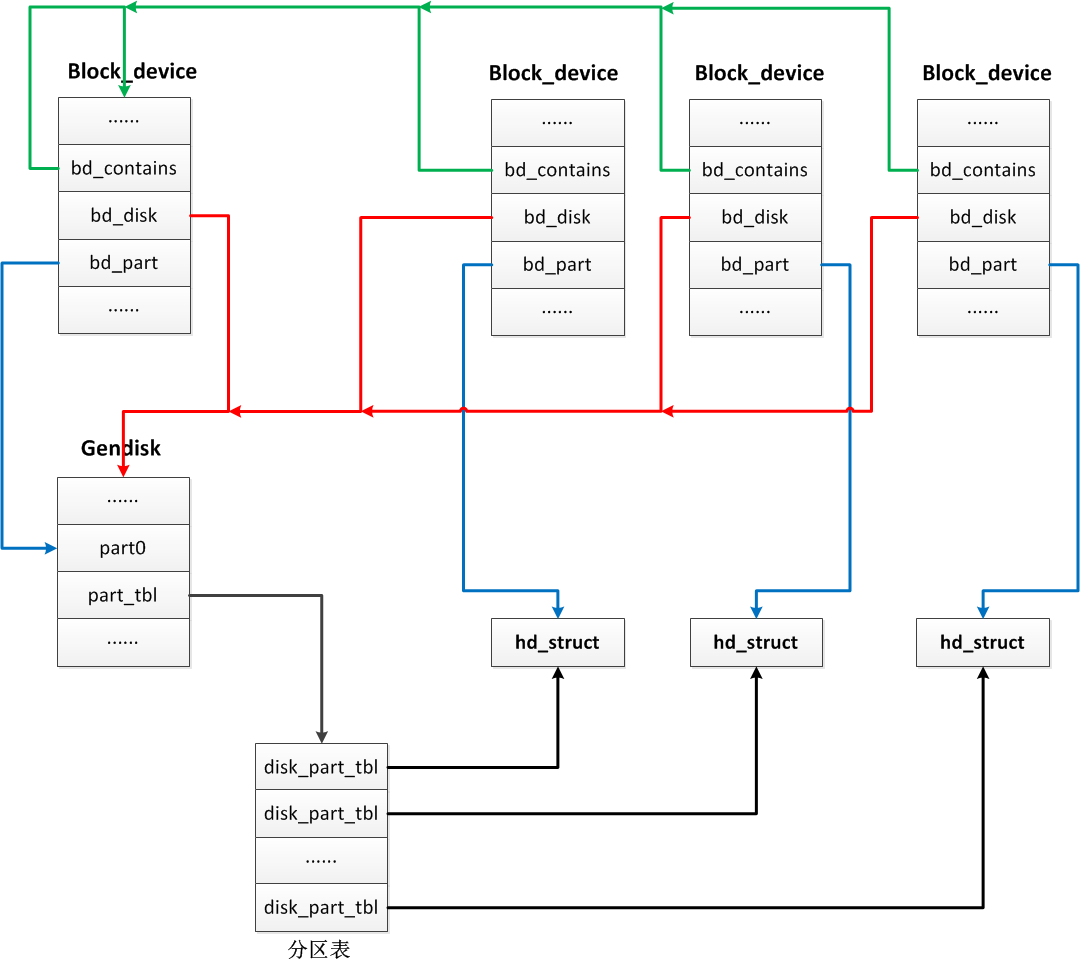
bio\_vec



高速缓存是linux内核实现的一种主要磁盘缓存，它主要用来减少对磁盘的IO操作，具体地讲，是通过把磁盘中的数据缓存到物理内存中，把对磁盘的访问变为对物理内存的访问。为什么要这么做呢？一，速度；二临时局部原理。有关这两个概念，相信熟悉操作系统的我们不会太陌生。页高速缓存是由RAM中的物理页组成的，缓存中的每一页都对应着磁盘中的多个块。每当内核开始执行一个页IO操作时，就先到高速缓存中找。这样就可以大大减少磁盘操作。

      一个物理页可能由多个不连续的物理磁盘块组成。也正是由于页面中映射的磁盘块不一定连续，所以在页高速缓存中检测特定数据是否已被缓存就变得不那么容易了。另外linux页高速缓存对被缓存页的范围定义的非常宽。缓存的目标是任何基于页的对象，这包含各种类型的文件和各种类型的内存映射。为了满足普遍性要求，linux使用定义在linux/fs.h中的结构体address\_space结构体描述页高速缓存中的页面。

其中的i\_mmap字段是一个优先搜索树，它的搜索范围包含了在address\_sapce中私有的和共享的页面。nrpages反应了address\_space空间的大小。address\_space结构往往会和某些内核对象关联。通常情况下，会与一个索引节点(inode)关联，这时host域就会指向该索引节点。如果关联对象不是一个索引节点的话，比如address\_space和swapper关联时，这是host域会被置为NULL。



首先block\_device既可以是gendisk的抽象，又可以是hd\_struct(分区)的抽象当作为分区的抽象时，bd\_contains指向了该分区所属的gendisk对应的block\_device

当作为gendisk的抽象时，bd\_contains指向自身的block\_device