转自：http://www.cnblogs.com/QJohnson/archive/2011/06/24/2089414.html

struct inode——字符设备驱动相关的重要结构介绍

内核中用inode结构表示具体的文件，而用file结构表示打开的文件描述符。Linux2.6.27内核中，inode结构体具体定义如下：

```
struct inode {

    struct hlist_node    i_hash;
    struct list_head    i_list;
    struct list_head    i_sb_list;
    struct list_head    i_dentry;
    unsigned long        i_ino;
    atomic_t        i_count;
    unsigned int        i_nlink;
    uid_t            i_uid;
    gid_t            i_gid;
    dev_t            i_rdev;    //该成员表示设备文件的inode结构，它包含了真正的设备编
号。
    u64            i_version;
    loff_t            i_size;
#ifdef __NEED_I_SIZE_ORDERED
    seqcount_t        i_size_seqcount;
#endif
    struct timespec        i_atime;
    struct timespec        i_mtime;
    struct timespec        i_ctime;
    unsigned int        i_blkbits;
    blkcnt_t        i_blocks;
    unsigned short          i_bytes;
    umode_t            i_mode;
    spinlock_t        i_lock;    /* i_blocks, i_bytes, maybe i_size */
    struct mutex        i_mutex;
     struct rw_semaphore    i_alloc_sem;
    const struct inode_operations    *i_op;
    const struct file_operations    *i_fop;    /* former ->i_op->default_file_ops
*/
    struct super_block    *i_sb;
    struct file_lock    *i_flock;
    struct address_space    *i_mapping;
    struct address_space    i_data;
#ifdef CONFIG_QUOTA
    struct dquot        *i_dquot[MAXQUOTAS];
#endif
    struct list_head    i_devices;
    union {
        struct pipe_inode_info    *i_pipe;
        struct block_device    *i_bdev;
        struct cdev        *i_cdev; //该成员表示字符设备的内核的 内部结构。当inode指
```

```
向一个字符设备文件时，该成员包含了指向struct cdev结构的指针，其中cdev结构是字符设备结构体。
    };
    int             i_cindex;

    __u32            i_generation;

#ifdef CONFIG_DNOTIFY
    unsigned long        i_dnotify_mask; /* Directory notify events */
    struct dnotify_struct    *i_dnotify; /* for directory notifications */
#endif

#ifdef CONFIG_INOTIFY
    struct list_head    inotify_watches; /* watches on this inode */
    struct mutex        inotify_mutex;    /* protects the watches list */
#endif

    unsigned long        i_state;
    unsigned long        dirtied_when;    /* jiffies of first dirtying */

    unsigned int        i_flags;

    atomic_t        i_writecount;
#ifdef CONFIG_SECURITY
    void            *i_security;
#endif
    void            *i_private; /* fs or device private pointer */
};
```

struct file ——字符设备驱动相关重要结构

文件结构 代表一个打开的文件描述符，它不是专门给驱动程序使用的，系统中每一个打开的文件在内核中都有一个关联的struct file。它由内核在open时创建，并传递给在文件上操作的任何函数，知道最后关闭。当文件的所有实例都关闭之后，内核释放这个数据结构。

```
struct file {
     /*
     * fu_list becomes invalid after file_free is called and queued via
     * fu_rcuhead for RCU freeing
     */
     union {
         struct list_head    fu_list;
         struct rcu_head     fu_rcuhead;
     } f_u;
     struct path        f_path;
     #define f_dentry    f_path.dentry    //该成员是对应的 目录结构 。
     #define f_vfsmnt    f_path.mnt
     const struct file_operations    *f_op;  //该操作 是定义文件关联的操作的。内核在执行open时对这个 指针赋值。
     atomic_long_t        f_count;
```

```
    unsigned int         f_flags;    //该成员是文件标志。
    mode_t            f_mode;
    loff_t            f_pos;
    struct fown_struct    f_owner;
    unsigned int        f_uid, f_gid;
    struct file_ra_state    f_ra;

    u64            f_version;
#ifdef CONFIG_SECURITY
    void            *f_security;
#endif
    /* needed for tty driver, and maybe others */
    void            *private_data;//该成员是系统调用时保存状态信息非常有用的资源。

#ifdef CONFIG_EPOLL
    /* Used by fs/eventpoll.c to link all the hooks to this file */
    struct list_head    f_ep_links;
    spinlock_t        f_ep_lock;
#endif /* #ifdef CONFIG_EPOLL */
    struct address_space    *f_mapping;
#ifdef CONFIG_DEBUG_WRITECOUNT
    unsigned long f_mnt_write_state;
#endif
};
```

---

file结构体和inode结构体

- struct file结构体定义在include/linux/fs.h中定义。文件结构体代表一个打开的文件，系统中的每个打开的文件在内核空间都有一个关联的 struct file。它由内核在打开文件时创建，并传递给在文件上进行操作的任何函数。在文件的所有实例都关闭后，内核释放这个数据结构。在内核创建和驱动源码中，struct file 的指针通常被命名为file或filp。如下所示：

```
struct file {
    union {
        struct list_head fu_list; 文件对象链表指针linux/include/linux/list.h
        struct rcu_head fu_rcuhead; RCU(Read-Copy Update)是Linux 2.6内核中新的锁机
制
    } f_u;
    struct path f_path;  包含dentry和mnt两个成员，用于确定文件路径
    #define f_dentry  f_path.dentry  f_path的成员之一，当前文件的dentry结构
    #define f_vfsmnt  f_path.mnt  表示当前文件所在文件系统的挂载根目录
    const struct file_operations *f_op; 与该文件相关联的操作函数
    atomic_t  f_count; 文件的引用计数（有多少进程打开该文件）
    unsigned int  f_flags;  对应于open时指定的flag
    mode_t  f_mode; 读写模式：open的mod_t mode参数
    off_t  f_pos; 该文件在当前进程中的文件偏移量
    struct fown_struct f_owner; 该结构的作用是通过信号进行I/O时间通知的数据。
    unsigned int  f_uid, f_gid; 文件所有者id，所有者组id
    struct file_ra_state f_ra;  在linux/include/linux/fs.h中定义，文件预读相关
    unsigned long f_version;
```

```
#ifdef CONFIG_SECURITY
    void  *f_security;
#endif
    /* needed for tty driver, and maybe others */
    void *private_data;
#ifdef CONFIG_EPOLL
    /* Used by fs/eventpoll.c to link all the hooks to this file */
    struct list_head f_ep_links;
    spinlock_t f_ep_lock;
#endif /* #ifdef CONFIG_EPOLL */
    struct address_space *f_mapping;
};
```

- struct dentry dentry 的中文名称是目录项，是Linux文件系统中某个索引节点(inode)的链接。这个索引节点可以是文件，也可以是目录。inode（可理解为ext2 inode）对应于物理磁盘上的具体对象，dentry是一个内存实体，其中的d_inode成员指向对应的inode。也就是说，一个inode可以在运行的时候链接多个dentry，而d_count记录了这个链接的数量。

```
struct dentry {
    atomic_t d_count; 目录项对象使用计数器,可以有未使用态,使用态和负状态
    unsigned int d_flags; 目录项标志
    struct inode * d_inode; 与文件名关联的索引节点
    struct dentry * d_parent; 父目录的目录项对象
    struct list_head d_hash; 散列表表项的指针
    struct list_head d_lru; 未使用链表的指针
    struct list_head d_child; 父目录中目录项对象的链表的指针
    struct list_head d_subdirs;对目录而言，表示子目录目录项对象的链表
    struct list_head d_alias; 相关索引节点（别名）的链表
    int d_mounted; 对于安装点而言，表示被安装文件系统根项
    struct qstr d_name; 文件名
    unsigned long d_time; /* used by d_revalidate */
    struct dentry_operations *d_op; 目录项方法
    struct super_block * d_sb; 文件的超级块对象
    vunsigned long d_vfs_flags;
    void * d_fsdata;与文件系统相关的数据
    unsigned char d_iname [DNAME_INLINE_LEN]; 存放短文件名

};
```

- 索引节点对象由inode结构体表示，定义文件在linux/fs.h中。

```
struct inode {
    struct hlist_node       i_hash; 哈希表
    struct list_head        i_list;    索引节点链表
    struct list_head        i_dentry; 目录项链表
    unsigned long           i_ino;   节点号
    atomic_t                i_count; 引用记数
    umode_t                 i_mode; 访问权限控制
    unsigned int            i_nlink; 硬链接数
```

```
    uid_t                   i_uid;  使用者id
    gid_t                   i_gid;  使用者id组
    kdev_t                  i_rdev; 实设备标识符
    loff_t                  i_size;  以字节为单位的文件大小
    struct timespec         i_atime; 最后访问时间
    struct timespec         i_mtime; 最后修改(modify)时间
    struct timespec         i_ctime; 最后改变(change)时间
    unsigned int            i_blkbits; 以位为单位的块大小
    unsigned long           i_blksize; 以字节为单位的块大小
    unsigned long           i_version; 版本号
    unsigned long           i_blocks; 文件的块数
    unsigned short          i_bytes; 使用的字节数
    spinlock_t              i_lock; 自旋锁
    struct rw_semaphore     i_alloc_sem; 索引节点信号量
    struct inode_operations *i_op; 索引节点操作表
    struct file_operations  *i_fop; 默认的索引节点操作
    struct super_block      *i_sb; 相关的超级块
    struct file_lock        *i_flock; 文件锁链表
    struct address_space    *i_mapping; 相关的地址映射
    struct address_space    i_data; 设备地址映射
    struct dquot            *i_dquot[MAXQUOTAS];节点的磁盘限额
    struct list_head        i_devices; 块设备链表
    struct pipe_inode_info  *i_pipe; 管道信息
    struct block_device     *i_bdev; 块设备驱动
    unsigned long           i_dnotify_mask;目录通知掩码
    struct dnotify_struct   *i_dnotify; 目录通知
    unsigned long           i_state; 状态标志
    unsigned long           dirtied_when;首次修改时间
    unsigned int            i_flags; 文件系统标志
    unsigned char           i_sock; 套接字
    atomic_t                i_writecount; 写者记数
    void                    *i_security; 安全模块
    __u32                   i_generation; 索引节点版本号
    union{
        void            *generic_ip;文件特殊信息
    } u;
};
```

inode 译成中文就是索引节点。每个存储设备或存储设备的分区（存储设备是硬盘、软盘、U盘 ... ... ）被格式化为文件系统后，应该有两部份，一部份是inode，另一部份是Block，Block是用来存储数据用的。而inode呢，就是用来存储这些数据的信息，这些信息包括文件大小、属主、归属的用户组、读写权限等。inode为每个文件进行信息索引，所以就有了inode的数值。操作系统根据指令，能通过inode值最快的找到相对应的文件。做个比喻，比如一本书，存储设备或分区就相当于这本书，Block相当于书中的每一页，inode 就相当于这本书前面的目录，一本书有很多的内容，如果想查找某部份的内容，我们可以先查目录，通过目录能最快的找到我们想要看的内容。

当我们用ls 查看某个目录或文件时，如果加上-i 参数，就可以看到inode节点了；比如ls -li lsfile.sh ，最前面的数值就是inode信息.