LINUX NAND 驱动核心层分析

系统的 nand 驱动程序在系统中以两种形式呈现，一种以字符的形式呈现，字符驱动程序，第二种是以块设备的方式进行呈现，一种对应与用户端进行按照字符设备的操作方式进行操作，比如获得 NAND 的设备信息，另外一种要结合文件系统的方式进行操作，比如是格式化之后,进行常规文件的创建操作。

<1> nand 的块设备创建和流程分析

在对应的驱动程序中，NAND 的块设备程序系统已经帮我们做完了几乎所有的无差异部分，只剩下差异相关的部分需要用户进行填充了，这里当然包括块设备创建的时候的分区创建，和差异的 NAND 芯片差异的读写方式。在 NAND 中一个 NAND 设备对应的一个 MTD 设备结构体，这里主要包括设备的差异读写函数：在 MTD 中的私有的 nand_chip 作为识别和读写 NAND 的核心部分。

```
    /* 1. 分配一个nand_chip结构体 */
    s3c_nand = kzalloc(sizeof(struct nand_chip), GFP_KERNEL);

    s3c_nand_regs = ioremap(0x4E000000, sizeof(struct s3c_nand_regs));

    /* 2. 设置nand_chip */
    /* 设置nand_chip是给nand_scan函数使用的, 如果不知道怎么设置, 先看nand_scan怎么使用
     * 它应该提供:选中,发命令,发地址,发数据,读数据,判断状态的功能
     */
    s3c_nand->select_chip = s3c2440_select_chip;
    s3c_nand->cmd_ctrl    = s3c2440_cmd_ctrl;
    s3c_nand->IO_ADDR_R   = &s3c_nand_regs->nfdata;
    s3c_nand->IO_ADDR_W   = &s3c_nand_regs->nfdata;
    s3c_nand->dev_ready   = s3c2440_dev_ready;
    s3c_nand->ecc.mode    = NAND_ECC_SOFT;
```

构造完 nand_chip 这块 nand 在系统差异的部分其实已经完成了，剩下的就只剩下系统无差异的部分进行的设备节点的创建，字符设备的创建，设备对应的块设备的创建，这里主要是给对应的块设备的读写队列中添加对这块 nand 的操作，至此设备驱动的整个工作也简要分析了一遍。一下是块设备的创建和与相关分区结合起来的流程分析。

一下是 MTD 的分区情况：

```
static struct mtd_partition s3c_nand_parts[] = {
    [0] = {
        .name   = "bootloader",
        .size   = 0x00040000,
        .offset = 0,
    },
    [1] = {
        .name   = "params",
        .offset = MTDPART_OFS_APPEND,
        .size   = 0x00020000,
    },
    [2] = {
        .name   = "kernel",
        .offset = MTDPART_OFS_APPEND,
        .size   = 0x00200000,
    },
    [3] = {
        .name   = "root",
        .offset = MTDPART_OFS_APPEND,
        .size   = MTDPART_SIZ_FULL,
    }
};
```

对应的分区描述，通过 add_mtd_partition 来进行设备分区的创建和设备节点的创建。

```
/* 4. 使用: nand_scan */
s3c_mtd = kzalloc(sizeof(struct mtd_info), GFP_KERNEL);
s3c_mtd->owner = THIS_MODULE;
s3c_mtd->priv  = s3c_nand;

nand_scan(s3c_mtd, 1);  /* 识别NAND FLASH, 构造mtd_info */

/* 5. add_mtd_partitions */
add_mtd_partitions(s3c_mtd, s3c_nand_parts, 4);
```

调用的核心是 add_mtd_partition

<2> add_mtd_partition 块设备创建的核心

在 add_mtd_Partion 中函数做了大多数块设备相关的操作，这里的操作主要包括设备节点的创建，和块设备的创建，块设备的队列创建，这样就可以以分区为单位进行设备分区的创建，达到的目的每个分区一个 块设备队列，一个设备节点，这样就将单个的 NAND 进行了多分区，多使用的方式。

NAND 读写根源还是在 NAND 抽象的块设备中，也就是系统中的统一的抽象块设备 gendisk 。和对用的设备读写队列。

```
    for (i = 0; i < nbparts; i++)
    {
        slave = add_one_partition(master, parts + i, i, cur_offset);

        if (!slave)
            return -ENOMEM;

        cur_offset = slave->offset + slave->mtd.size;
    }
```

安装所属的分区个数进行设备的分区创建与节点的创建，这里开始分析设备的节点创建流程。

add_one_partition

add_mtd_device(&slave->mtd);

```
dev_set_name(&mtd->dev, "mtd%d", i);
dev_set_drvdata(&mtd->dev, mtd);
if (device_register(&mtd->dev) != 0)
    goto fail_added;

if (MTD_DEVT(i))
    device_create(&mtd_class, mtd->dev.parent,
                  MTD_DEVT(i) + 1,
                  NULL, "mtd%dro", i);
```

```
list_for_each_entry(not, &mtd_notifiers, list)
    not->add(mtd);
```

进入创建设备的关键步骤：

1. 开始对设备的一些参数的检查
2. 在 add_mtd_device 中进行设备节点的创建
3. 通知上层中的设备创建

分析 not->addd 中设备的 块设备的创建

```
static void blktrans_notify_add(struct mtd_info *mtd)
{
    struct mtd_blktrans_ops *tr;

    if (mtd->type == MTD_ABSENT)
        return;

    list_for_each_entry(tr, &blktrans_majors, list)
        tr->add_mtd(tr, mtd);
}
```

最终调用的函数指针是

```
static void mtdblock_add_mtd(struct mtd_blktrans_ops *tr, struct mtd_info *mtd)
{
    struct mtd_blktrans_dev *dev = kzalloc(sizeof(*dev), GFP_KERNEL);

    if (!dev)
        return;

    dev->mtd    = mtd;
    dev->devnum = mtd->index;

    dev->size = mtd->size >> 9;
    dev->tr   = tr;
    dev->readonly = 1;

    if (add_mtd_blktrans_dev(dev))
        kfree(dev);
}
```

<3>add_mtd_blktrans_dev

gd = alloc_disk(1 << tr->part_bits); ***分配一个 gendisk***

new->rq=blk_init_queue(mtd_blktrans_request,&new->queue_loc ***初始化 gendisk 队列***

add_disk(gd); *添加队列。*

这样从文件系统到读写队列整个链路都已将通畅了，其实还是利用 nand 的差异的读写函数，和对应 nand 的分区情况这样就可以创建队列的底层的 gendisk 块设备和对应的 gendisk 的读写队列了。