/dev目录下并没有一个/dev/misc的device，而是直接包含各个misc device

比如

```
crw-------  1 root root       10, 130  3月   8 08:22 watchdog
crw-------  1 root root       10,   1  3月   8 08:22 psaux
crw-------  1 root root       10, 227  3月   8 08:22 mcelog
crw-------  1 root root       10, 228  3月   8 08:22 hpet
crw-------  1 root root       10, 231  3月   8 08:22 snapshot
crw-------  1 root root       10, 237  3月   8 08:22 loop-control
crw-------  1 root root       10,  52  3月   8 08:23 vboxnetctl
crw-------  1 root root       10,  53  3月   8 08:23 vboxdrv
crw-------  1 root root       10,  54  3月   8 08:22 mei
crw-------  1 root root       10,  55  3月   8 08:22 network_throughput
crw-------  1 root root       10,  56  3月   8 08:22 network_latency
crw-------  1 root root       10,  57  3月   8 08:22 cpu_dma_latency
crw-------  1 root root       10,  58  3月   8 08:22 alarm
crw-------  1 root root       10,  59  3月   8 08:22 ashmem
crw-------  1 root root       10,  60  3月   8 08:22 binder
crw-------  1 root root       10,  61  3月   8 08:22 ecryptfs
crw-------  1 root root       10,  63  3月   8 08:22 vga_arbiter
```

这里device major为10的都是misc device。

misc device是通过 device minor来区分的。

当user space application open misc device时，首先得到调用的是"misc" device driver的file_operations中的.open callback,因为kernel是通过device major来定位driver的，而所有misc device共享10。

in drivers/char/misc.c

```c
static int __init misc_init(void)
{
        int err;

#ifdef CONFIG_PROC_FS
        proc_create("misc", 0, NULL, &misc_proc_fops);
#endif
        misc_class = class_create(THIS_MODULE, "misc");
        err = PTR_ERR(misc_class);
        if (IS_ERR(misc_class))
                goto fail_remove;

        err = -EIO;
        if (register_chrdev(MISC_MAJOR,"misc",&misc_fops))
                goto fail_printk;
        misc_class->devnode = misc_devnode;
        return 0;

fail_printk:
        printk("unable to get major %d for misc devices\n", MISC_MAJOR);
        class_destroy(misc_class);
fail_remove:
        remove_proc_entry("misc", NULL);
        return err;
}
```

"misc" char device注册的file_operations是misc_fops。

```c
static const struct file_operations misc_fops = {
        .owner          = THIS_MODULE,
        .open           = misc_open,
        .llseek         = noop_llseek,
};
```

"misc" driver的open handler的作用是再根据当前misc device的minor来找到该device driver的file_operations

来替换原来的misc_fops。这样下次user space application再access(read / write / io etc)该device时，就会被

route到正确的device driver的file_operations的callback handler上。这个逻辑是由misc_open()完成的。

比如/dev/tun的driver

in drivers/net/tun.c

```c
static int __init tun_init(void)
{
        int ret = 0;

        pr_info("%s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
        pr_info("%s\n", DRV_COPYRIGHT);

        ret = rtnl_link_register(&tun_link_ops);
        if (ret) {
                pr_err("Can't register link_ops\n");
                goto err_linkops;
        }

        ret = misc_register(&tun_miscdev);
        if (ret) {
                pr_err("Can't register misc device %d\n", TUN_MINOR);
                goto err_misc;
        }
        return  0;
err_misc:
        rtnl_link_unregister(&tun_link_ops);
err_linkops:
        return ret;
}
```

```c
static struct miscdevice tun_miscdev = {
        .minor = TUN_MINOR,
        .name = "tun",
        .nodename = "net/tun",
        .fops = &tun_fops,
};
```

```c
static const struct file_operations tun_fops = {
        .owner  = THIS_MODULE,
        .llseek = no_llseek,
        .read  = do_sync_read,
        .aio_read  = tun_chr_aio_read,
        .write = do_sync_write,
        .aio_write = tun_chr_aio_write,
        .poll   = tun_chr_poll,
        .unlocked_ioctl = tun_chr_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl = tun_chr_compat_ioctl,
#endif
        .open   = tun_chr_open,
        .release = tun_chr_close,
        .fasync = tun_chr_fasync,
#ifdef CONFIG_PROC_FS
        .show_fdinfo = tun_chr_show_fdinfo,
#endif
};
```

tun_fops是/dev/tun misc device的真正的file_operations。

in drivers/char/misc.c

```
static int misc_open(struct inode * inode, struct file * file)
{
        int minor = iminor(inode);                                       ①
        struct miscdevice *c;
        int err = -ENODEV;
        const struct file_operations *new_fops = NULL;

        mutex_lock(&misc_mtx);

        list_for_each_entry(c, &misc_list, list) {                    ②
                if (c->minor == minor) {
                        new_fops = fops_get(c->fops);
   ③
                        break;
                }
        }

        if (!new_fops) {
                mutex_unlock(&misc_mtx);
                request_module("char-major-%d-%d", MISC_MAJOR, minor);
                mutex_lock(&misc_mtx);

                list_for_each_entry(c, &misc_list, list) {
                        if (c->minor == minor) {
                                new_fops = fops_get(c->fops);
                                break;
                        }
                }
                if (!new_fops)
                        goto fail;
        }

        err = 0;
        replace_fops(file, new_fops);                                   ④
        if (file->f_op->open) {
                file->private_data = c;                                           ⑤
                err = file->f_op->open(inode,file);                         ⑥
        }
fail:
        mutex_unlock(&misc_mtx);
        return err;
}
```

①

minor是user space application访问的misc device的minor number

②

misc_list中包含了当前注册的所有misc device driver

③

找到由minor标识的miscdevice，variable c指向该miscdevice

new_fops是该miscdevice的file_operations

④

用当前miscdevice中的file_operation替换file*中的file_operations.

这样就完成了从general misc device的file_operations到特定miscdevice

的file_operation的转变

⑤

c是当前nisc device的struct miscdevice

⑥

调用真正的misc device的.open callback。

以后user space application在read / write 该misc device，调用的是该device的read / write，而绕过general misc的file_operations了。