user访问/dev/ttyXXX的interface是file_operations，而tty device的核心interface是tty_operations。

当user access /dev/ttyXXX device时，从file_operations到tty_operations的route呢？

比如对串口tty而言

in drivers/tty/serial/serial_core.c

```
1.  int uart_register_driver(struct uart_driver *drv)
2.  {
3.          struct tty_driver *normal;
4.          int i, retval;
5.
6.          BUG_ON(drv->state);
7.
8.          /*
9.           * Maybe we should be using a slab cache for this, especially if
10.          * we have a large number of ports to handle.
11.          */
12.         drv->state = kzalloc(sizeof(struct uart_state) * drv->nr, GFP_KERNEL);
13.         if (!drv->state)
14.                 goto out;
15.
16.         normal = alloc_tty_driver(drv->nr);
17.         if (!normal)
18.                 goto out_kfree;
19.
20.         drv->tty_driver = normal;
21.
22.         normal->driver_name     = drv->driver_name;
23.         normal->name            = drv->dev_name;
24.         normal->major           = drv->major;
25.         normal->minor_start     = drv->minor;
26.         normal->type            = TTY_DRIVER_TYPE_SERIAL;
27.         normal->subtype         = SERIAL_TYPE_NORMAL;
28.         normal->init_termios    = tty_std_termios;
29.         normal->init_termios.c_cflag = B9600 | CS8 | CREAD | HUPCL | CLOCAL;
30.         normal->init_termios.c_ispeed = normal->init_termios.c_ospeed = 9600;
31.         normal->flags           = TTY_DRIVER_REAL_RAW | TTY_DRIVER_DYNAMIC_DEV;
32.         normal->driver_state    = drv;
33.         tty_set_operations(normal, &uart_ops);          ①
34.
35.         /*
36.          * Initialise the UART state(s).
37.          */
38.         for (i = 0; i < drv->nr; i++) {                 ②
39.                 struct uart_state *state = drv->state + i;
40.                 struct tty_port *port = &state->port;
41.
42.                 tty_port_init(port);
43.                 port->ops = &uart_port_ops;
44.                 port->close_delay     = HZ / 2; /* .5 seconds */
45.                 port->closing_wait    = 30 * HZ;/* 30 seconds */
46.         }
47.
48.         retval = tty_register_driver(normal);           ③
49.         if (retval >= 0)
50.                 return retval;
51.
52.         for (i = 0; i < drv->nr; i++)
53.                 tty_port_destroy(&drv->state[i].port);
```

```
54.             put_tty_driver(normal);
55.     out_kfree:
56.             kfree(drv->state);
57.     out:
58.             return -ENOMEM;
59.     }
```

①

这里的uart_ops就是整个uart tty(/dev/ttySXXX)的tty_operations

tty driver的核心structure

```
1.  static const struct tty_operations uart_ops = {
2.          .open           = uart_open,
3.          .close          = uart_close,
4.          .write          = uart_write,
5.          .put_char       = uart_put_char,
6.          .flush_chars    = uart_flush_chars,
7.          .write_room     = uart_write_room,
8.          .chars_in_buffer= uart_chars_in_buffer,
9.          .flush_buffer   = uart_flush_buffer,
10.         .ioctl          = uart_ioctl,
11.         .throttle       = uart_throttle,
12.         .unthrottle     = uart_unthrottle,
13.         .send_xchar     = uart_send_xchar,
14.         .set_termios    = uart_set_termios,
15.         .set_ldisc      = uart_set_ldisc,
16.         .stop           = uart_stop,
17.         .start          = uart_start,
18.         .hangup         = uart_hangup,
19.         .break_ctl      = uart_break_ctl,
20.         .wait_until_sent= uart_wait_until_sent,
21.  #ifdef CONFIG_PROC_FS
22.         .proc_fops      = &uart_proc_fops,
23.  #endif
24.         .tiocmget       = uart_tiocmget,
25.         .tiocmset       = uart_tiocmset,
26.         .get_icount     = uart_get_icount,
27.  #ifdef CONFIG_CONSOLE_POLL
28.         .poll_init      = uart_poll_init,
29.         .poll_get_char  = uart_poll_get_char,
30.         .poll_put_char  = uart_poll_put_char,
31.  #endif
32.  };
```

②

这里的drv->nr是有多少个物理串口interface。每个physical uart interface都由一个

struct uart_state和struct tty_port对应。

③

in drivers/tty/tty_io.c

```
1.  int tty_register_driver(struct tty_driver *driver)
2.  {
3.          int error;
4.          int i;
5.          dev_t dev;
6.          struct device *d;
7.
8.          if (!driver->major) {
9.                  error = alloc_chrdev_region(&dev, driver->minor_start,
10.                                              driver->num, driver->name);
11.                 if (!error) {
12.                         driver->major = MAJOR(dev);
13.                         driver->minor_start = MINOR(dev);
14.                 }
15.         } else {
16.                 dev = MKDEV(driver->major, driver->minor_start);
17.                 error = register_chrdev_region(dev, driver->num, driver->name);
18.         }
19.         if (error < 0)
20.                 goto err;
21.
22.         if (driver->flags & TTY_DRIVER_DYNAMIC_ALLOC) {          ④
23.                 error = tty_cdev_add(driver, dev, 0, driver->num);
24.                 if (error)
25.                         goto err_unreg_char;
26.         }
27.
28.         mutex_lock(&tty_mutex);
29.         list_add(&driver->tty_drivers, &tty_drivers);
30.         mutex_unlock(&tty_mutex);
31.
32.         if (!(driver->flags & TTY_DRIVER_DYNAMIC_DEV)) {         ⑤
33.                 for (i = 0; i < driver->num; i++) {
34.                         d = tty_register_device(driver, i, NULL);
35.                                 error = PTR_ERR(d);
36.                                 goto err_unreg_devs;
37.                         }
38.                 }
39.         }
40.         proc_tty_register_driver(driver);
41.         driver->flags |= TTY_DRIVER_INSTALLED;
42.         return 0;
43.
44. err_unreg_devs:
45.         for (i--; i >= 0; i--)
46.                 tty_unregister_device(driver, i);
47.
48.         mutex_lock(&tty_mutex);
49.         list_del(&driver->tty_drivers);
50.         mutex_unlock(&tty_mutex);
51.
52. err_unreg_char:
53.         unregister_chrdev_region(dev, driver->num);
```

```
54.   err:
55.           return error;
56.   }
```

④

对uart tty driver而言

normal->flags = TTY_DRIVER_REAL_RAW | TTY_DRIVER_DYNAMIC_DEV;

该条件不满足

⑤

normal->flags = TTY_DRIVER_REAL_RAW | TTY_DRIVER_DYNAMIC_DEV;

该条件同样不满足

所以在uart_register_driver()中并不会去创建character device(/dev/ttySXXX)

以drivers/tty/serial/pxa.c uart driver为例

```
static int __init serial_pxa_init(void)
{
        int ret;

        ret = uart_register_driver(&serial_pxa_reg);            ①
        if (ret != 0)
                return ret;

        ret = platform_driver_register(&serial_pxa_driver);     ②
        if (ret != 0)
                uart_unregister_driver(&serial_pxa_reg);

        return ret;
}
```

①

这时虽然调用了tty_register_driver()，但还没有create char devices。

②

```
static struct platform_driver serial_pxa_driver = {
        .probe          = serial_pxa_probe,             ③
        .remove         = serial_pxa_remove,

        .driver         = {
                .name   = "pxa2xx-uart",
                .owner  = THIS_MODULE,
#ifdef CONFIG_PM
                .pm     = &serial_pxa_pm_ops,
#endif
                .of_match_table = serial_pxa_dt_ids,
        },
};
```

```
static int serial_pxa_probe(struct platform_device *dev)
{
        struct uart_pxa_port *sport;
        struct resource *mmres, *irqres;
        int ret;

    ......

uart_add_one_port(&serial_pxa_reg, &sport->port);    ④
        platform_set_drvdata(dev, sport);

        return 0;

    ......
}
```

③

向driver framework注册serial_pxa_probe()

④

uart_add_one_port()才会create "/dev/ttySXXX" char devices

这里可以理解每一个pxa uart device，则probe()就会运行一次，uart_add_one_port()就会运行一次，一个新的/dev/ttySXX device就会被创建。

uart_add_one_port()

|

|

\|/

tty_port_register_device_attr()

|

|

\|/

tty_register_device_attr()

|

|

\|/

tty_cdev_add()

in drivers/tty/tty_io.c

```
static int tty_cdev_add(struct tty_driver *driver, dev_t dev,
                unsigned int index, unsigned int count)
{
        /* init here, since reused cdevs cause crashes */
        cdev_init(&driver->cdevs[index], &tty_fops);    ⑤
        driver->cdevs[index].owner = driver->owner;
        return cdev_add(&driver->cdevs[index], dev, count);
}
```

⑤

创建/dev/ttySXXX时的file_operations是tty_fops

in drivers/tty/tty_io.c

```
static const struct file_operations tty_fops= {
        .llseek         = no_llseek,
        .read           = tty_read,
        .write          = tty_write,
        .poll           = tty_poll,
        .unlocked_ioctl = tty_ioctl,
        .compat_ioctl   = tty_compat_ioctl,
        .open           = tty_open,
        .release        = tty_release,
        .fasync         = tty_fasync,
};
```

比如open "dev/ttyS0" char device

tty_open() (file_operations callback)

|

|

\|/

uart_open() (tty_operations callback)

|

|

\|/

struct uart_port.startup() (uart_port callback, serial_pxa_startup() in pxa.c)

比如read "dev/ttyS0" char device

tty_read() (file_operations callback)

|

|

\|/

line discipline->tty_ldisc_ops->read() (n_tty_read)

由于read()完全是异步的，所以看不到直接调用hardware uart driver的function。

在tty_operations中没有read callback!

比如write "dev/ttyS0" char device

tty_write() (file_operations callback)

|

|

\|/

do_tty_write()

|

|

\|/

line discipline->tty_ldisc_ops->write() (n_tty_write)

|

|

\|/

uart_write() (tty_operations)，serial_core.c

|

|

\|/

uart_start()

|

|

\|/

uart_port.start_tx() (对应serial_pxa_start_tx() in pxa.c)

比如close "/dev/ttyS0" char device

tty_release()　　(file_operations callback)

　　|

　　|

　　\|/

tty_operations.close() (uart_close in serial_core.c)

　　|

　　|

　　\|/

uart_port.uart_ops.stop_rx() (serial_pxa_stop_rx() in pxa.c)