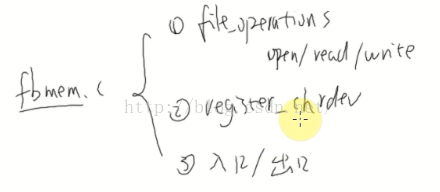
**1、对于复杂的字符设备驱动程序（以LCD为例）**

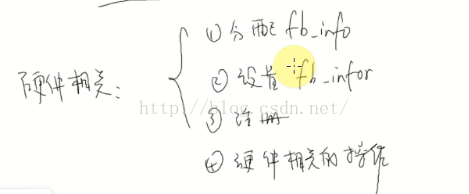
**引入分层的概念**

**(1)上层fbmem.c(对于LCD驱动程序来说，内核已经帮我们做好了，这个fbmem.c是内核提供的)**

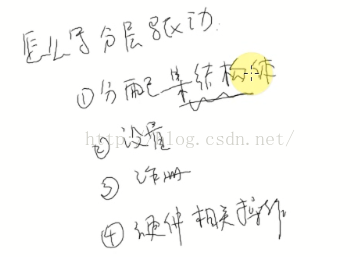
****

**(2)我们要做的是硬件相关这一层**

分配、设置、注册fb\_info结构体，这里的注册是把这个结构体告诉fbmem.c,当应用程序调用读写等函数操作LCD的时候，首先会调用fbmem.c的file\_operation结构体的读写函数，在这些读写函数里面就会调用之前的fb\_info结构体里面的函数或属性来操作硬件。

****

**（3）总结：如何写对于分层的设备驱动程序**

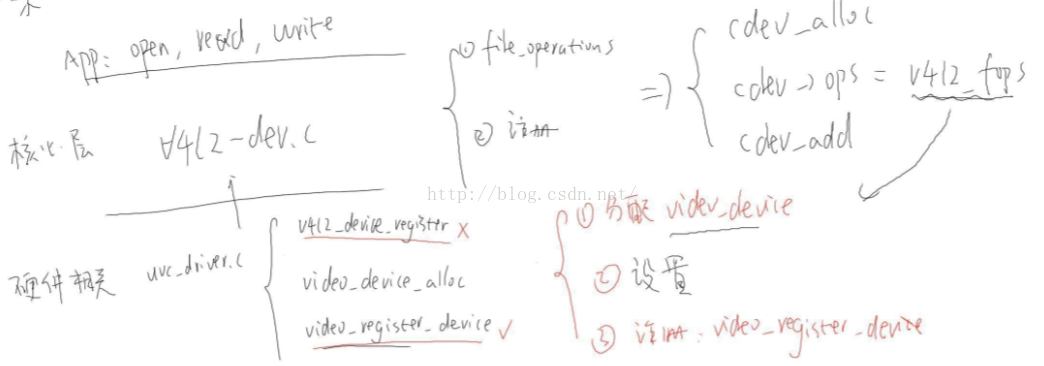
****

**2、V4L2框架和驱动**

**（1）框架**

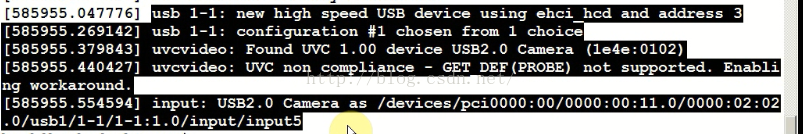
注册file\_operation结构体就是把这个结构体变成结构体cdev的ops成员。

应用程序调用读写函数，会调用v4l2\_fops结构体（也就是file\_operation结构体）里面的读写函数。v4l2\_fops结构体肯定会调用到video\_device结构体提供的各种函数或属性（如里面的file\_operation结构体）。

****

不知道代码是哪一个：

打开虚拟机，并位于前面，接上USB摄像头到电脑上去，然后看内核的输出信息（用dmesg命令查看）

****

**进入内核目录搜索关键词，最终会发现在uvc\_driver.c里面**

**https://img-blog.csdn.net/20161203204054895?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center**

**r表示递归,n表示显示行号**

根据搜索结果反推摄像头驱动框架，uvc是usb video class，是usb类的驱动程序，是硬件相关的驱动程序。

uvc\_driver.c会向核心层注册一个结构体usb\_driver，这个结构体里面会有一个probe函数，当发现支持的设备（通过id\_table比较）就会调用probe函数。

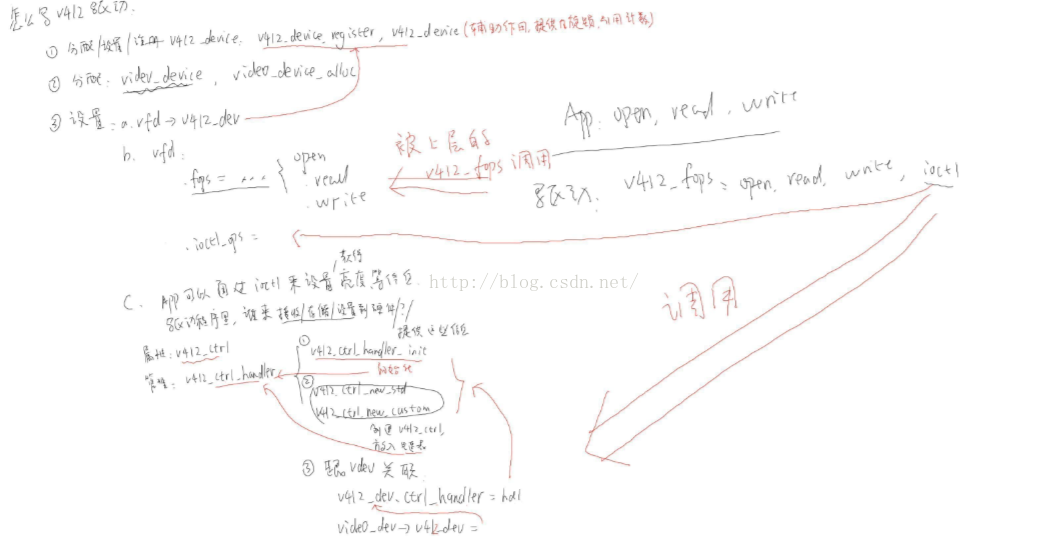
在这个probe函数里面分配、设置、注册

**（2）驱动编写**

分配、设置、注册v4l2\_device结构体

分配、设置、注册video\_device（vfd）结构体,这个结构体的v4l2\_device成员指向

上面的v4l2\_device结构体

****

**3、分析V4l2内部实现细节（通过vivi.c(是一个虚拟的视频驱动)）**

**（1）虚拟视频驱动vivi.c分析:**  
1．分配video\_device  
2．设置  
3．注册：video\_register\_device  
  
**入口函数分析**  
vivi\_init  
    vivi\_create\_instance  
        v4l2\_device\_register   // 不是主要, 只是用于初始化一些东西，比如自旋锁、引用计数

        //分配video\_device  
        video\_device\_alloc  
        // 设置  
          1. vfd:  
            .fops           = &vivi\_fops,  //file\_operation结构体  
            .ioctl\_ops = &vivi\_ioctl\_ops,//ioctl操作函数相关结构体  
            .release = video\_device\_release,  
          2.  
            vfd->v4l2\_dev = &dev->v4l2\_dev;  
          3. 设置"ctrl属性"(用于APP的ioctl)：  
            v4l2\_ctrl\_handler\_init(hdl, 11);//初始化ctrl\_handler,下面是设置这个ctrl\_handler（类似于链表），下面函数是创建各个属性并填充到这个链表  
            dev->volume = v4l2\_ctrl\_new\_std(hdl, &vivi\_ctrl\_ops,  
            V4L2\_CID\_AUDIO\_VOLUME, 0, 255, 1, 200);//音量  
            dev->brightness = v4l2\_ctrl\_new\_std(hdl, &vivi\_ctrl\_ops,  
            V4L2\_CID\_BRIGHTNESS, 0, 255, 1, 127);//明暗度  
            dev->contrast = v4l2\_ctrl\_new\_std(hdl, &vivi\_ctrl\_ops,//对比度  
            V4L2\_CID\_CONTRAST, 0, 255, 1, 16);

        //注册   （根据VFL\_TYPE\_GRABBER这个类型自动创建设备节点）根据类型得到不同的名字、次设备号等

  
        video\_register\_device(video\_device, type:VFL\_TYPE\_GRABBER, nr)  
            \_\_video\_register\_device  
                vdev->cdev = cdev\_alloc(); //分配cdev结构体  
                vdev->cdev->ops = &v4l2\_fops;//cdev结构体的ops成员等于v4l2\_fops（file\_operation结构体）  
                cdev\_add //注册cdev结构体  
                  
                video\_device[vdev->minor] = vdev;   //video\_device数组以次设备号为下标把video\_device结构体放进去  
  
  
        if (vdev->ctrl\_handler == NULL)  //包含V4L2\_ctrl结构体的ctrl\_handler跟video\_device关联  
        vdev->ctrl\_handler = vdev->v4l2\_dev->ctrl\_handler;

**（2）分析vivi.c的open,read,write,ioctl过程**

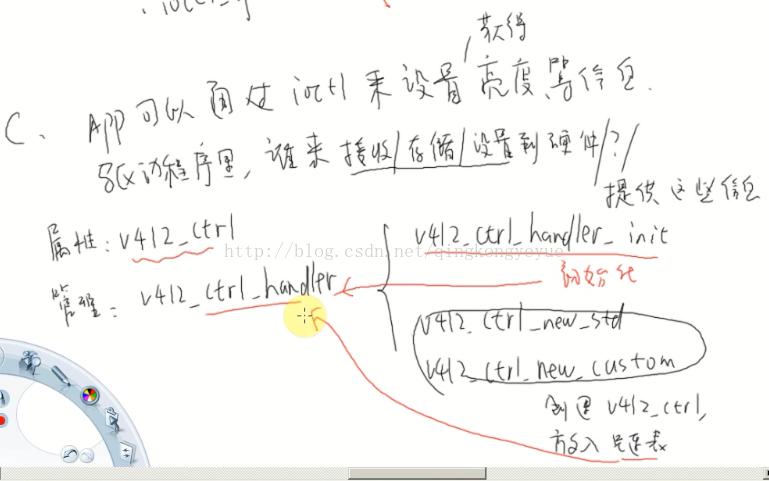
1. open  
app:     open("/dev/video0",....)  
---------------------------------------------------  
drv:     v4l2\_fops.v4l2\_open  //v4l2\_fops是cdev结构体下的file\_operation结构体，调用其open函数  
            vdev = video\_devdata(filp);  // 根据次设备号从数组中得到video\_device  
            ret = vdev->fops->open(filp); //调用video\_device结构体里面的file\_operatoin结构体的open函数，最终调用v4l2\_ioctl\_ops结构体里面的open函数即 v4l2\_fh\_open

                        vivi\_ioctl\_ops.open  
                            v4l2\_fh\_open  
  
  
2. read  
app:    read ....  
---------------------------------------------------  
drv:    v4l2\_fops.v4l2\_read  //v4l2\_fops是cdev结构体下的file\_operation结构体，调用其read函数

            struct video\_device \*vdev = video\_devdata(filp); // 根据次设备号从数组中得到video\_device  
            ret = vdev->fops->read(filp, buf, sz, off);//调用video\_device结构体里面的file\_operatoin结构体的read函数  
  
  
3. ioctl  
app:   ioctl  
----------------------------------------------------  
drv:   v4l2\_fops.unlocked\_ioctl v4l2\_fops是cdev结构体下的file\_operation结构体，调用其ioctl函数  
v4l2\_ioctl  
                struct video\_device \*vdev = video\_devdata(filp);/ /根据次设备号从数组中得到video\_device  
                ret = vdev->fops->unlocked\_ioctl(filp, cmd, arg);//调用video\_device结构体里面的file\_operatoin结构体的ioctl函数  
                            video\_ioctl2  
                                video\_usercopy(file, cmd, arg, \_\_video\_do\_ioctl);//复制用户空间的参数arg，命令cmd ,然后调用\_\_video\_do\_ioctl函数  
                                    \_\_video\_do\_ioctl  
                                        struct video\_device \*vfd = video\_devdata(file);  
                                        根据APP传入的cmd来获得、设置"某些属性",

这些属性由谁提供  
  
  
v4l2\_ctrl\_handler的使用过程:  
    \_\_video\_do\_ioctl  
        struct video\_device \*vfd = video\_devdata(file);//获得video\_device  
  
  
    case VIDIOC\_QUERYCTRL:  
    {  
    struct v4l2\_queryctrl \*p = arg;  
      
    if (vfh && vfh->ctrl\_handler)   
    ret = v4l2\_queryctrl(vfh->ctrl\_handler, p);  
    else if (vfd->ctrl\_handler)  // 在哪设置？在video\_register\_device，video\_device里面设置了ctrl\_handler  
    ret = v4l2\_queryctrl(vfd->ctrl\_handler, p);  
               // 根据ID在ctrl\_handler里找到v4l2\_ctrl，返回它的值  
**4、V4L2\_ctrl结构体 （每个结构体对应一项（音量、亮度））**

**（1）作用**



**（2）这些结构体的管理通过v4l2\_ctrl\_handler（类似链表）**

**5、总结**

**（1）video\_device结构体的v4l2\_dev等于我们创建的v4l2\_device结构体**

**（2）vdev->ctrl\_handler = vdev->v4l2\_dev->ctrl\_handler;**

**（3）v4l2\_ctrl\_handler（类似链表）包含V4L2\_ctrl结构体 （代表各种属性）**