Camera V4L2 架构分析 - xiaofengcanyue2013的专栏 - 博客频道

分类：
kernel（20）

V4L2相关代码位于Linux-3.4/drivers/media/video目录中。

v4l2-dev.c文件提供设备节点实现与用户层数据交流，设备节点在/dev/目录下以video0、video1等名字出现。注册字符设备的语句如下：

```
1. /* Part 3: Initialize the character device */
2. vdev->cdev = cdev_alloc();
3. if (vdev->cdev == NULL) {
4.         ret = -ENOMEM;
5.         goto cleanup;
6. }
7. vdev->cdev->ops = &v4l2_fops;
8. vdev->cdev->owner = owner;
9. ret = cdev_add(vdev->cdev, MKDEV(VIDEO_MAJOR, vdev->minor), 1);
10. if (ret < 0) {
11.         printk(KERN_ERR "%s: cdev_add failed\n", __func__);
12.         kfree(vdev->cdev);
13.         vdev->cdev = NULL;
14.         goto cleanup;
15. }
16. /* Part 4: register the device with sysfs */
17. vdev->dev.class = &video_class;
18. vdev->dev.devt = MKDEV(VIDEO_MAJOR, vdev->minor);
19. if (vdev->parent)
20.         vdev->dev.parent = vdev->parent;
21. dev_set_name(&vdev->dev, "%s%d", name_base, vdev->num);
22. ret = device_register(&vdev->dev);
```

只要调用函数video_register_device即可把一个video_device注册到V4L2架构中；但这只是一个总纲领，Camera驱动主要是实现V4L2的一个子系统的功能，子系统用一个v4l2_subdev结构体来描述，只有实现好了系统要求的相关函数操作，最后一步才是注册到V4L2中。
子系统实现的方式每个平台都有差别，这里分析的是全志A23平台的代码。

在sunxi-vfe/vfe.c文件中，以platform形式注册了前后摄像头的平台资源。匹配过程忽略，最终vfe_probe函数会被调用。在probe中，看到有函数v4l2_i2c_new_subdev_board：

```
1.         /* Create the i2c client */
2.         if (info->addr == 0 && probe_addrs)
3.                 client = i2c_new_probed_device(adapter, info, probe_addrs,
4.                                                NULL);
5.         else
6.                 client = i2c_new_device(adapter, info);
7. ..............................
8.         /* Register with the v4l2_device which increases the module's
9.              use count as well. */
10.         if (v4l2_device_register_subdev(v4l2_dev, sd))


11.                 sd = NULL;
```

这里的client获得在之前写的I2C驱动的文章已经分析过，可见驱动的知识都是环环相扣的；根据I2C驱动文章的分析可知获取client过程中，device_register(&client->dev)会被调用，而根据device和device_driver的模型关系可知，device所在的bus总线会进行匹配，device处于i2c总线下，根据I2C match函数的匹配，名字相同的device_driver将被匹配中；我们假设i2c_client的name为"ov5460"吧，这样ov5460.c中看看：

```
1. static int sensor_probe(struct i2c_client *client,


2.             const struct i2c_device_id *id)
```

```
{
    struct  v4l2_subdev  *sd;
    struct  sensor_info  *info;
//    int  ret;
    info  =  kzalloc(sizeof(struct  sensor_info),  GFP_KERNEL);
    if  (info  ==  NULL)
        return  -ENOMEM;
    sd  =  &info->sd;
    glb_sd  =  sd;
    v4l2_i2c_subdev_init(sd,  client,  &sensor_ops);
    info->fmt  =  &sensor_formats[0];
    info->af_first_flag  =  1;
    info->init_first_flag  =  1;
    info->auto_focus  =  0;
    return  0;
}
```

这里就看到了前面提到的v4l2_subdev结构体，v4l2_i2c_subdev_init函数会进入v4l2-subdev.c进行一系列初始化操作，并且用i2c_set_clientdata(client, sd);保存子系统指针，以便后续取出。这里sensor_ops结构体即为子系统支持的类型：

```
static  const  struct  v4l2_subdev_ops  sensor_ops  =  {


    .core  =  &sensor_core_ops,
    .video  =  &sensor_video_ops,
};
```

Camera当然是是video的了，core是核心，应该是不可少的操作吧。其实v4l2_subdev所支持的类型很多，其全部类型定义如下：

```
struct  v4l2_subdev_video_ops  {
        int  (*s_routing)(struct  v4l2_subdev  *sd,  u32  input,  u32  output,  u32  config);
        int  (*s_crystal_freq)(struct  v4l2_subdev  *sd,  u32  freq,  u32  flags);
        int  (*s_std_output)(struct  v4l2_subdev  *sd,  v4l2_std_id  std);
        int  (*g_std_output)(struct  v4l2_subdev  *sd,  v4l2_std_id  *std);
        int  (*querystd)(struct  v4l2_subdev  *sd,  v4l2_std_id  *std);
        int  (*g_tvnorms_output)(struct  v4l2_subdev  *sd,  v4l2_std_id  *std);
        int  (*g_input_status)(struct  v4l2_subdev  *sd,  u32  *status);
        int  (*s_stream)(struct  v4l2_subdev  *sd,  int  enable);
        int  (*cropcap)(struct  v4l2_subdev  *sd,  struct  v4l2_cropcap  *cc);
        int  (*g_crop)(struct  v4l2_subdev  *sd,  struct  v4l2_crop  *crop);
        int  (*s_crop)
    (struct  v4l2_subdev  *sd,  struct  v4l2_crop  *crop);
        int  (*g_parm)(struct  v4l2_subdev  *sd,  struct  v4l2_streamparm  *param);
        int  (*s_parm)(struct  v4l2_subdev  *sd,  struct  v4l2_streamparm  *param);
        int  (*g_frame_interval)(struct  v4l2_subdev  *sd,
                                 struct  v4l2_subdev_frame_interval  *interval);
        int  (*s_frame_interval)(struct  v4l2_subdev  *sd,
                                 struct  v4l2_subdev_frame_interval  *interval);
        int  (*enum_framesizes)(struct  v4l2_subdev  *sd,  struct  v4l2_frmsizeenum  *fsize);
        int  (*enum_frameintervals)(struct  v4l2_subdev  *sd,  struct  v4l2_frmivalenum  *fival);
        int  (*enum_dv_presets)  (struct  v4l2_subdev  *sd,
                        struct  v4l2_dv_enum_preset  *preset);
        int  (*s_dv_preset)(struct  v4l2_subdev  *sd,
                        struct  v4l2_dv_preset  *preset);
        int  (*g_dv_preset)(struct  v4l2_subdev  *sd,
                        struct  v4l2_dv_preset  *preset);
        int  (*query_dv_preset)(struct  v4l2_subdev  *sd,
                        struct  v4l2_dv_preset  *preset);
        int  (*s_dv_timings)(struct  v4l2_subdev  *sd,
                        struct  v4l2_dv_timings  *timings);
        int  (*g_dv_timings)(struct  v4l2_subdev  *sd,
```

```
32.                         struct  v4l2_dv_timings  *timings);
33.         int  (*enum_mbus_fmt)(struct  v4l2_subdev  *sd,  unsigned  int  index,
34.                                 enum  v4l2_mbus_pixelcode  *code);
35.         int  (*enum_mbus_fsizes)(struct  v4l2_subdev  *sd,
36.                                 struct  v4l2_frmsizeenum  *fsize);
37.         int  (*g_mbus_fmt)(struct  v4l2_subdev  *sd,
38.                             struct  v4l2_mbus_framefmt  *fmt);
39.         int  (*try_mbus_fmt)(struct  v4l2_subdev  *sd,
40.                               struct  v4l2_mbus_framefmt  *fmt);
41.         int  (*s_mbus_fmt)(struct  v4l2_subdev  *sd,
42.                             struct  v4l2_mbus_framefmt  *fmt);
43.         int  (*g_mbus_config)(struct  v4l2_subdev  *sd,
44.                                 struct  v4l2_mbus_config  *cfg);
45.         int  (*s_mbus_config)(struct  v4l2_subdev  *sd,
46.                                 const  struct  v4l2_mbus_config  *cfg);
47. };
```

太多了，这就是一个具体摄像头主要实现的操作，而ov5640只是支持其中一部分操作而已：