/home/alientek/RK3568/SDK/linux/rk3568\_linux\_sdk/kernel/drivers/media/imx415\_drv\_project/imx415\_drv.c

我们创建的IMX415驱动文件位于此目录下。参考gc2053.c代码的程序进行编写。

摄像头驱动核心就在 注册函数，注册就是说，他会跟设备树里面的compatable

#include <linux/clk.h>

#include <linux/device.h>

#include <linux/delay.h>

#include <linux/gpio/consumer.h>

#include <linux/i2c.h>

#include <linux/module.h>

#include <linux/pm\_runtime.h>

#include <linux/of.h>

#include <linux/of\_graph.h>

#include <linux/regulator/consumer.h>

#include <linux/sysfs.h>

#include <linux/slab.h>

#include <linux/pinctrl/consumer.h>

#include <linux/version.h>

#include <linux/rk-camera-module.h>

//下面的都是关于media（v4l2）的一些头文件，红色警告不用管

#include <media/v4l2-async.h>

#include <media/media-entity.h>

#include <media/v4l2-common.h>

#include <media/v4l2-ctrls.h>

#include <media/v4l2-device.h>

#include <media/v4l2-event.h>

#include <media/v4l2-fwnode.h>

#include <media/v4l2-image-sizes.h>

#include <media/v4l2-mediabus.h>

#include <media/v4l2-subdev.h>

//这些都是系统自带的头文件。

#define IMX415\_NAME "imx415"

//实现imx415\_pm\_ops：

static const struct dev\_pm\_ops imx415\_pm\_ops = {

SET\_RUNTIME\_PM\_OPS(imx415\_runtime\_suspend,

imx415\_runtime\_resume, NULL)

};

static int imx415\_pm\_resume(struct device \*dev)

//3.实现prob函数

static int imx415\_probe(struct i2c\_client \*client,const struct i2c\_device\_id \*id)

{

return 0;

}

//4.

static int imx415\_remove(struct i2c\_client \*client)

{

return 0;

}

//5. 定义匹配表

static const struct i2c\_device\_id imx415\_i2c\_driver[] =

{

{"imx415",0},

{ },

};

#if IS\_ENABLE(CONFIG\_OF)

static const struct of\_device\_id imx415\_of\_match[] = {

{.compatible = "fsl,imx415", },

{ },

};

MODULE\_DEVICE\_TABLE(of, imx415\_of\_match);

#endif

//1.定义和设置i2c\_drv取得结构体

/\*

driver：Device driver model driver主要包含驱动名称和与DTS注册设备进行匹配的

of\_match\_table。当of\_match\_table中的compatible域 和 dts文件的compatible域匹配时，·probe函数才会被调用

id\_table:List of I2C devices supported by this driver如果kernel没有使用of\_match\_table和

dts注册设备进行进行匹配，则kernel使用该table进行匹配

probe:Callback for device binding

remove:Callback for device unbinding

\*/

static struct i2c\_driver imx415\_i2c\_driver = {

.driver = {

.name = IMX415\_NAME,

.pm = &imx415\_pm\_ops,

.of\_match\_table = of\_match\_ptr(imx415\_of\_match),

},

.probe = &imx415\_probe,

.remove = &imx415\_remove,

.id\_table = imx415\_match\_id,

};

//2.注册imx415\_i2c\_driver结构体

static int \_\_init sensor\_mod\_init(void)

{

return i2c\_add\_driver(&imx415\_i2c\_driver);

}

static void \_\_exit sensor\_mod\_exit(void)

{

i2c\_del\_driver(&imx415\_i2c\_driver);

}

device\_initcall\_sync(sensor\_mod\_init);

module\_exit(sensor\_mod\_exit);

MODULE\_DESCRIPTION("IMX415 CMOS Image Sensor driver");

MODULE\_LICENSE("GPL v2");

/\*

到这里，整个sensor驱动框架就已经搭建完成了，

接下来就需要去完善细节，重点就是在probe函数里面，

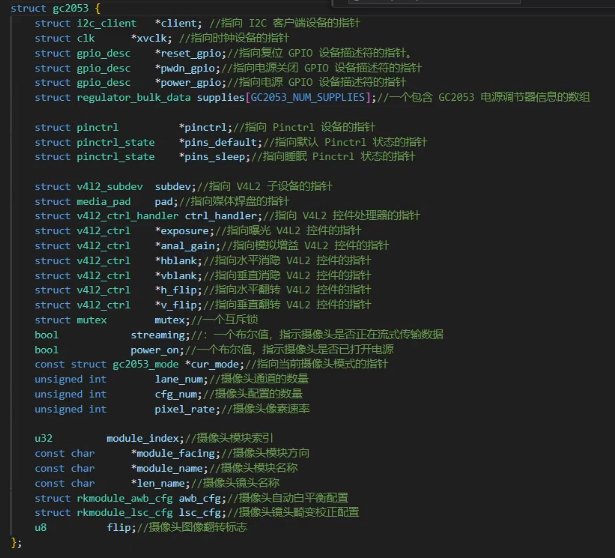
probe接口就会涉及到v4l2驱动框架相关的知识点。

在probe函数里面实现v4l2也需要参考gc20的驱动框架的内容。

\*/

本节先实现v4l2\_subdev和device这两个，剩下的慢慢实现。

Struct gc2053 \*gc2053这个结构体是我们自己定义的一个结构体，里面包含的就是摄像头的一些属性：



就是后面我们需要对摄像头做的一些操作。

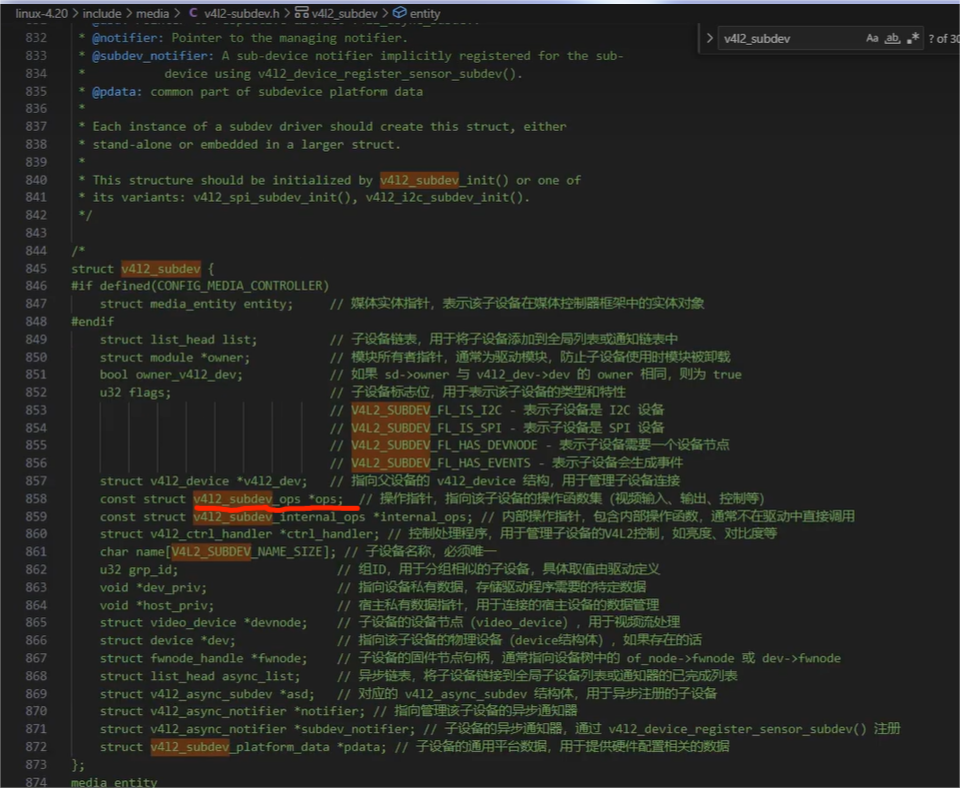
在这里先对我们常用的一些和device相关的结构体 进行说明：

struct device :不是表示视频设备，而是linux内核里面表示所有的设备的一个模型

struct video\_device：专门对视频设备进行访问的一个结构体

struct v4l2\_device：要使用v4l2的驱动框架，就必须要有这样一个结构体。

struct v4l2\_subdev：这个是摄像头里面的控制模块，比如上层要操作曝光等参数，就需要使用这个结构体。



这个结构体里面就有一个v4l2\_subdev\_ops结构体，通过这个结构体，我们可以操作不同类型的设备，我们现在操作的是视频设备，但有时候，我们不一定操作的是视频设备，也有可能是音频设备，如果说对视频设备进行操作，那具体操作哪些？有哪些接口呢？

(该结构体位于include/media/v4l2-subdev.h文件里面。)

