第9期内容：

问题交流和我个人微信： tu18879499804

●

1、本期内容结束sensor驱动源码的讲解

●

2、介绍一下v4l2的学习和v4l2\_subdev的介绍

●

3、梳理整个sensor驱动源码主要做了什么

●

4、第10期开始讲解mipi传输，怎么把sensor的数据传输到Isp那边去做处理

1、v4l2\_subdev的详细介绍：

很多设备都需要与子设备进行交互，通常情况下子设备用于音视频的编解码以及混合处理，对于网络摄像机来说子设备就是 sensors 和 camera 控制器。通常情况下它们都是 I2C 设备，但也有例外。v4l2\_subdev 结构体被用于子设备管理。

每一个子设备驱动都必须有一个 v4l2\_subdev 结构体，这个结构体可以作为独立的简单子设备存在，也可以嵌入到更大的结构体（自定义的子设备结构体）里面。通常会有一个由内核设置的低层次结构体（i2c\_client，也就是上面说的 i2c 设备），它包含了一些设备数据，要调用 v4l2\_set\_subdevdata 来设置子设备私有数据指针指向它，这样的话就可以很方便的从 subdev 找到相关的 I2C 设备数据（这个要编程实现的时候才能够了解它的用意）。另外也需要设置低级别结构的私有数据指针指向 v4l2\_subdev 结构体，方便从低级别的结构体访问 v4l2\_subdev 结构体，达到双向访问的目的，对于 i2c\_client 来说，可以用 i2c\_set\_clientdata 函数来设置，其它的需使用与之相应的函数来完成设置。

桥驱动器需要存储每一个子设备的私有数据，v4l2\_subdev 结构体提供了主机私有数据指针成员来实现此目的，使用以下函数可以对主机私有数据进行访问控制：

v4l2\_get\_subdev\_hostdata();

v4l2\_set\_subdev\_hostdata();

从桥驱动器的角度来看，我们加载子设备模块之后可以用某种方式获取子设备指针。对于 i2c 设备来说，调用 i2c\_get\_clientdata 函数即可完成，其它类型的设备也有与之相似的操作，在内核里面提供了不少的帮助函数来协助完成这部分工作，编程时可以多多使用。

每个 v4l2\_subdev 结构体都包含有一些函数指针，指向驱动实现的回调函数，内核对这些回调函数进行了分类以避免出现定义了一个巨大的回调函数集，但是里面只有那么几个用得上的尴尬情况。最顶层的操作函数结构体内部包含指向各个不同类别操作函数结构体的指针成员，如下所示：

struct v4l2\_subdev\_core\_ops {

int (\*log\_status)(struct v4l2\_subdev \*sd);

int (\*s\_io\_pin\_config)(struct v4l2\_subdev \*sd, size\_t n,

struct v4l2\_subdev\_io\_pin\_config \*pincfg);

int (\*init)(struct v4l2\_subdev \*sd, u32 val);

int (\*load\_fw)(struct v4l2\_subdev \*sd);

int (\*reset)(struct v4l2\_subdev \*sd, u32 val);

int (\*s\_gpio)(struct v4l2\_subdev \*sd, u32 val);

long (\*ioctl)(struct v4l2\_subdev \*sd, unsigned int cmd, void \*arg);

#ifdef CONFIG\_COMPAT

long (\*compat\_ioctl32)(struct v4l2\_subdev \*sd, unsigned int cmd,

unsigned long arg);

#endif

#ifdef CONFIG\_VIDEO\_ADV\_DEBUG

int (\*g\_register)(struct v4l2\_subdev \*sd, struct v4l2\_dbg\_register \*reg);

int (\*s\_register)(struct v4l2\_subdev \*sd, const struct v4l2\_dbg\_register \*reg);

#endif

int (\*s\_power)(struct v4l2\_subdev \*sd, int on);

int (\*interrupt\_service\_routine)(struct v4l2\_subdev \*sd,

u32 status, bool \*handled);

int (\*subscribe\_event)(struct v4l2\_subdev \*sd, struct v4l2\_fh \*fh,

struct v4l2\_event\_subscription \*sub);

int (\*unsubscribe\_event)(struct v4l2\_subdev \*sd, struct v4l2\_fh \*fh,

struct v4l2\_event\_subscription \*sub);

};

struct v4l2\_subdev {

#if defined(CONFIG\_MEDIA\_CONTROLLER)

struct media\_entity entity;

#endif

struct list\_head list;

struct module \*owner;

bool owner\_v4l2\_dev;

u32 flags;

struct v4l2\_device \*v4l2\_dev;

const struct v4l2\_subdev\_ops \*ops;

const struct v4l2\_subdev\_internal\_ops \*internal\_ops;

struct v4l2\_ctrl\_handler \*ctrl\_handler;

char name[V4L2\_SUBDEV\_NAME\_SIZE];

u32 grp\_id;

void \*dev\_priv;

void \*host\_priv;

struct video\_device \*devnode;

struct device \*dev;

struct fwnode\_handle \*fwnode;

struct list\_head async\_list;

struct v4l2\_async\_subdev \*asd;

struct v4l2\_async\_notifier \*notifier;

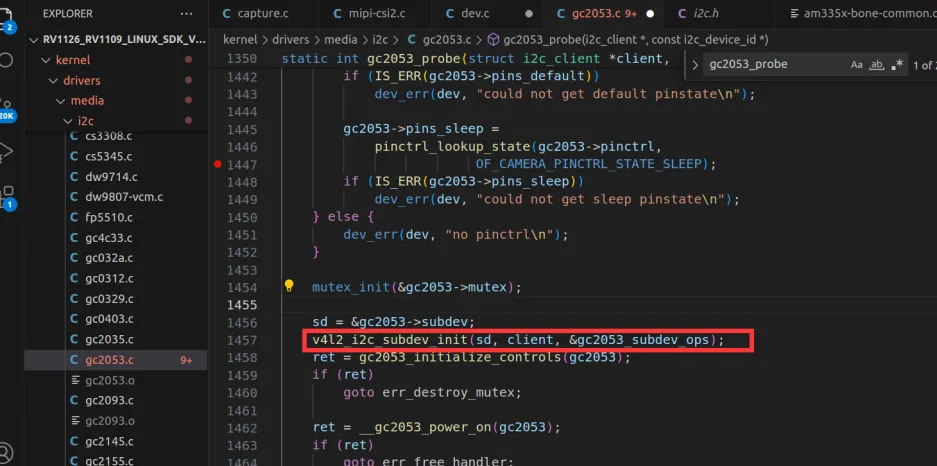
struct v4l2\_async\_notifier \*subdev\_notifier;

struct v4l2\_subdev\_platform\_data \*pdata;

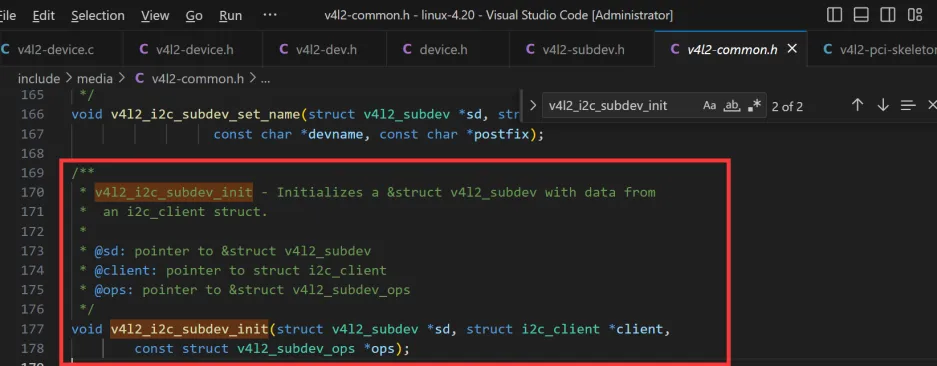
};

子设备驱动的初始化使用 v4l2\_subdev\_init 函数来完成（该函数只是初始化一些 v4l2\_subdev 的成员变量，内容比较简单），在初始化之后需要设置子设备结构体的 name 和 owner 成员（如果是 i2c 设备的话，这个在 i2c helper 函数里面就会被设置）.该部分 ioctl 可以直接通过用户空间的 ioctl 命令访问到（前提是该子设备在用户空间生成了子设备节点，这样的话就可以操作子设备节点来进行 ioctl）。内核里面可以使用 v4l2\_subdev\_call 函数来对这些回调函数进行调用，这个在 pipeline 管理的时候十分受用。

下面是gc2053里面的源码分析：



v4l2\_i2c\_subdev\_init接口的说明：



● v4l2\_i2c\_subdev\_init : **这是一个函数调用，用于初始化一个v4l2\_subdev结构，该结构表示一个V4L2视频子设备。**

●sd : **这是一个指向v4l2\_subdev结构的指针，表示要初始化的视频子设备。**

●client : **这是一个指向i2c\_client结构的指针，包含有关I2C设备的信息，如地址、总线等。**

●&gc2053\_subdev\_ops : **这是一个指向操作函数集的指针，用于指定视频子设备的操作函数，这些函数包括控制、配置等。**

●**这段代码的作用是使用给定的i2c\_client结构和操作函数集来初始化一个v4l2\_subdev结构，从而创建一个表示特定I2C视频子设备的数据结构。** 也就是说在Linux V4L2框架下，初始化一个型号为gc2053的I2C子设备，并为其注册一系列的操作函数，以便后续可以通过这些函数与子设备进行交互