一、第8期内容讲解：

1、了解mipi、uvc、dvp接口：

- USB：是串行通用串行总线（Universal Serial Bus）的简称

- MIPI是移动行业处理器接口（Mobile Industry Processor Interface）

mipi接口的特点：

MIPI是差分串口传输，速度快，抗干扰。主流手机模组现在都是用MIPI传输，传输时使用4对差分信号传输图像数据和一对差分时钟信号；

最初是为了减少LCD屏和主控芯片之间连线的数量而设计的，后来发展到高速了，支持高分辨率的显示屏，现在基本上都是MIPI接口了。

MIPI摄像头有三个电源：VDDIO（IO电源），AVDD（模拟电源），DVDD（内核数字电源），不同sensor模组的摄像头供电不同，

AVDD有2.8V或3.3V的；DVDD一般使用1.5V或更高，不同厂家的设计不同，1.5V可能由sensor模组提供或外部供给，

可以使用外部供电则建议使用外部供，电压需大于内部的DVDD；VDDIO电压应与MIPI信号线的电平一致，

若信号线是2.8V电平，则VDDIO也应供2.8V，有些sensor模组也可以不供VDDIO，由内部提供。

- DVP是数字视频端口（digital video port）的简称

dvp接口特点：

DVP总线PCLK极限约在96M左右，而且走线长度不能过长，所有DVP最大速率最好控制在72M以下，PCB layout较容易画；

MIPI总线速率lvds接口耦合，走线必须差分等长，并且需要保护，

故对PCB走线以及阻抗控制要求高一点（一般来讲差分阻抗要求在85欧姆~125欧姆之间）。

DVP是并口，需要PCLK、VSYNC、HSYNC、D[0：11]——可以是8/10/12bit数据，具体情况要看ISP或baseband是否支持；

MIPI是LVDS低压差分串口，只需要要CLKP/N、DATAP/N——最大支持4-lane，一般2-lane可以搞定。MIPI接口比DVP的接口信号线少，

由于是低压差分信号，产生的干扰小，抗干扰能力也强。最重要的是DVP接口在信号完整性方面受限制，速率也受限制。

500W还可以勉强用DVP，800W及以上都采用MIPI接口。

- CSI是相机串行接口（CMOS Sensor Interface）的简称

价格因素汇总：MIPI 摄像头和 DVP 摄像头价格，从接口上分不出高低。比如 OV5640 它同时支持两种接口，主要还是像素尺寸、

像素点的个数有区别。此外，一些高分辨率摄像头只支持 MIPI，一些低分辨率摄像头只支持 SPI\DVP 接口。

同等的 USB 摄像头会略微贵一些。

开机速度：通常 MIPI\DVP 开机启动速度快，USB 开机启动速度慢。

数据格式：原生的 MIPI\DVP 摄像头可以直接通过 DMA，然后可以传递给 ISP\SoC，可以拿到 raw 数据。

低功耗：MIPI 的耗电可以非常低，usb 耗电较高，并口的 DVP接口 功耗最高。

开发成本：MIPI\DVP 的摄像头依赖额外的驱动程序来工作。这意味着对不同图像传感器的支持有限，

除非有专门的摄像头厂商或开发人员真正推动它！而 USB 摄像头对用户而言是免驱的。

总结

1）USB、DVP、MIPI 都可以对接摄像头，他们的特点不同，其中 USB 是异步串行接口、MIPI 是同步串行接口、DVP是同步并行串口。

2）不同摄像头方案对摄像头的需求不同，因此在考虑抗干扰能力、分辨率大小、开发成本、设备体积的基础上对摄像头接口进行选型，进而选择合适的摄像头是非常重要的。

v4l2驱动框架，简介：

几乎所有的设备都有多个 IC 模块，它们可能是实体的（例如 USB 摄像头里面包含 ISP、sensor 等）、也可能是抽象的（如 USB 设备里面的抽象拓扑结构），它们在 /dev 目录下面生成了多个设备节点，并且这些 IC 模块还创建了一些非 v4l2 设备：DVB、ALSA、FB、I2C 和输入设备。正是由于硬件的复杂性，v4l2 的驱动也变得非常复杂。特别是 v4l2 驱动要支持 IC 模块来进行音/视频的混合/编解码操作，这就更加使得 v4l2 驱动变得异常复杂。通常情况下，有些IC模块通过一个或者多个 I2C 总线连接到主桥驱动上面，同时其它的总线仍然可用，这些 IC 就称为 ‘sub-devices’，比如摄像头设备里面的 sensor 传感器就是使用 I2C 来进行命令沟通，

同时使用 MIPI 或者 LVDS 等接口进行图像数据传输。在很长一段时间内，该框架（指老旧的 V4L2 框架）仅限于通过 video\_device 结构体创建 v4l 设备节点和 video\_buf 来处理视频数据。这意味着所有的驱动都必须对设备实例进行设置并将其映射到子设备上。有些时候这些操作步骤十分复杂，很难正确完成，并且有些驱动程序从来没有正确的按照这些操作步骤编写。由于缺少一个框架，有很多通用代码就没有办法被重构，从而导致这部分代码被重复编写，效率比较低下。

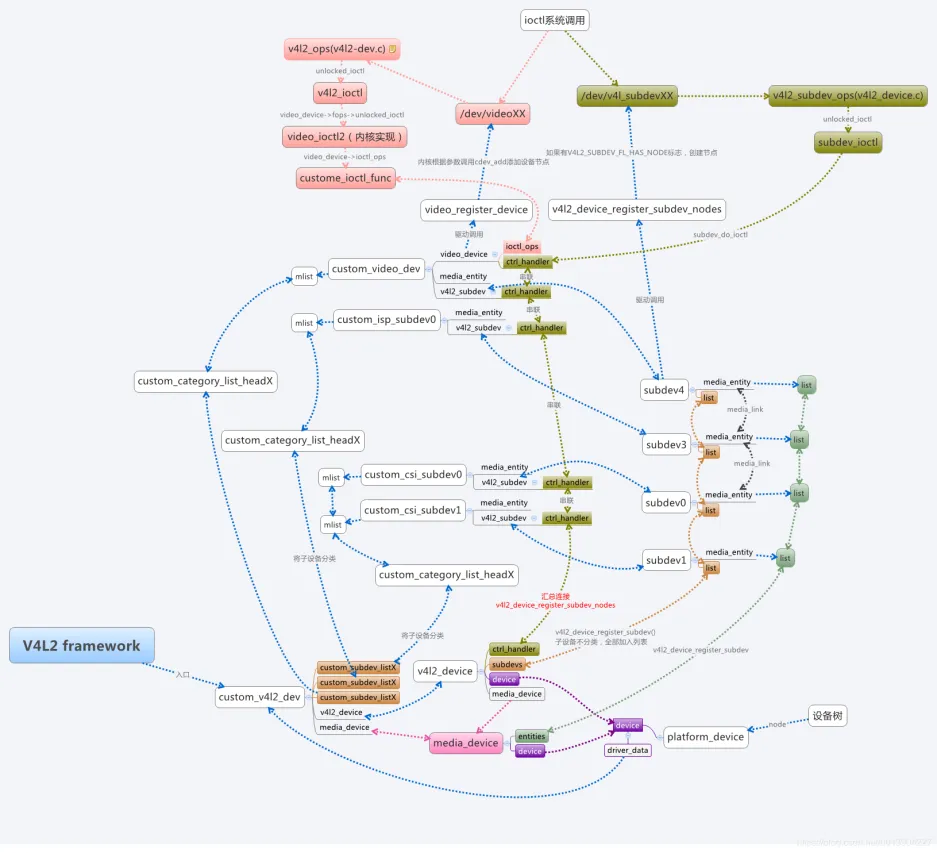
因此，本框架抽象构建了所有驱动都需要的代码并封装为一个个的模块，

简化了设备驱动通用代码的重构。v4l2-pci-skeleton.c 是个非常好的参考例程，它是一个PCI采集卡的驱动框架。该例程演示了如何使用 v4l2 驱动框架，并且该例程可以作为一个 PCI 视频采集卡的驱动模板使用。

在最开始的时候也可以参照这个代码编写方式进行联系，当然最适合的代码还是 drivers/media/video/omap3isp 文件夹里面的代码，

这个代码基本上可以作为一个完整的输入设备实例代码（因为它包含了 ISP、CSI、video 等设备，并且有着一个完整的数据流 pipeline，

几乎用到了 V4L2 框架的方方面面，参考价值极大）来进行参考编写自己的设备驱动代码。



v4l2\_device：这个是整个输入设备的总结构体，可以认为它是整个 V4L2 框架的入口，充当驱动的管理者以及入口监护人。由该结构体引申出来 v4l2\_subdev。用于视频输入设备整体的管理，有多少输入设备就有多少个v4l2\_device抽象（比如一个USB摄像头整体就可以看作是一个 V4L2 device）。再往下分是输入子设备，对应的是例如 ISP、CSI、MIPI 等设备，它们是从属于一个 V4L2 device 之下的。

●

media\_device：用于运行时数据流的管理，嵌入在 V4L2 device 内部，

运行时的意思就是：一个 V4L2 device 下属可能有非常多同类型的子设备（两个或者多个 sensor、ISP 等），那么在设备运行的时候我怎么知道我的数据流需要用到哪一个类型的哪一个子设备呢。这个时候就轮到 media\_device 出手了，

它为这一坨的子设备建立一条虚拟的连线，建立起来一个运行时的 pipeline（管道），并且可以在运行时动态改变、管理接入的设备。

●

v4l2\_ctrl\_handler：控制模块，提供子设备（主要是 video 和 ISP 设备）在用户空间的特效操作接口，比如你想改变下输出图像的亮度、对比度、饱和度等等，都可以通过这个来完成。

●

vb2\_queue：提供内核与用户空间的 buffer 流转接口，输入设备产生了一坨图像数据，在内核里面应该放在哪里呢？能放几个呢？是整段连续的还是还是分段连续的又或者是物理不连续的？用户怎么去取用呢？都是它在管理。

●

可以看到图中的入口 custom\_v4l2\_dev，它是由用户定义的一个结构体，重要的不是它怎么定义的，重要的是它里面有一个 v4l2\_device 结构体，上文说到，这个结构体总览全局，运筹帷幄，相当于中央管理处的位置，那么中央决定了，

它就是整个输入设备整体的抽象（比如整个 USB 摄像头输入设备，比如整个 IPC 摄像头输入设备）。它还有一个 media\_device 结构体，上文也说道，，它是管数据流线路的，属于搞结构路线规划管理的。

●

往后 v4l2\_device 里面有一个链表，它维护了一个巨大的子设备链，所有的子设备都通过内核的双向循环链表结构以v4l2\_device 为中心紧紧团结在一起。另外 media\_device 在往里面去就是一个个的 media\_entity

（现在不需要了解它的具体含义，只需要知道它就是类似电路板上面的元器件一样的抽象体），media\_entity 之间建立了自己的小圈子，在它们这个小圈子里面数据流按照一定的顺序畅通无阻，恣意遨游。

●

到结尾处，抽象出来了 /dev/videoX 设备节点，这个就是外交部的角色，它负责提供了一个内核与用户空间的交流枢纽。

需要注意的是，该设备节点的本质还是一个字符设备，其内部的一套操作与字符设备是一样的，只不过是进行了一层封装而已。

●

到此为止，一个 V4L2 大概的四层结构就抽象出来了，如下图所示：

media framework 其中一个目的是：在运行时状态下发现设备拓扑并对其进行配置。为了达到这个目的，media framework 将硬件设备抽象为一个个 entity，它们之间通过 links 连接。

media\_device，media\_entity，media\_link，media\_pad

1、entity：硬件设备模块抽象（类比电路板上面的各个元器件、芯片）

2、pad：硬件设备端口抽象（类比元器件、芯片上面的管脚）

3、link：硬件设备的连接抽象，link 的两端是 pad（类比元器件管脚之间的连线）

