# 1.3 V4L2 接口获取一张 YUV 图片_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 1.3 V4L2 接口获取一张 YUV 图片涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

V4L2 接口表面上很复杂，结构体成员信息众多，但是熟悉后发现只有 ioctl(fd, request, ...) 一套逻辑：

1）打开一个摄像头文件，获取 fd； 2）找到需要的 request 及相应的结构体，通过函数 ioctl( ) 便能够读取 / 配置相机属性参数；

3）内存映射，将相机数据和文件描述符建立关联，监听 fd 获得相机数据；

| request（IO 请求） | 功能含义 | ...（结构体 / 共用体 / 宏） |
| --- | --- | --- |
| VIDIOC_QUERYCAP | 设备支持的操作 | struct v4l2_capability |
| VIDIOC_ENUM_FMT | 枚举出来支持的帧格式 | struct v4l2_fmtdesc |
| VIDIOC_G_FMT | 获取设置支持的视频格式 | struct v4l2_format |
| VIDIOC_S_FMT | 设置捕获视频的格式 | struct v4l2_format |
| VIDIOC_REQBUFS | 申请内存缓冲区 | struct v4l2_requestbuffers |
| VIDIOC_QUERYBUF | 查询申请到的内存信息 | struct v4l2_buffer |
| VIDIOC_QBUF | 将空闲的内存加入可捕获视频的队列 | struct v4l2_buffer |
| VIDIOC_STREAMON | 打开视频流 | enum v4l2_buf_type |
| VIDIOC_DQBUF | 将已经捕获好视频的内存拉出已捕获视频的队列 | struct v4l2_buffer |
| VIDIOC_STREAMOFF | 关闭视频流 | enum v4l2_buf_type |
| ... | ... | ... |

什么意思呢？举个简单的例子，我想要设置相机捕获视频的格式，那么通过函数 ioctrl() 进行配置的过程如下：

```
struct v4l2_format stream_fmt = {` `    .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,//帧为视频帧捕获` `    .
if(-1 == ioctl(fd, VIDIOC_S_FMT, &stream_fmt)) {` `     perror("Fail to ioctl");` `     exit(EXIT_
```

知道了如何去查询和配置相机设备，那么完整的拍照步骤就可以展示出来了：

1）打开摄像头设备，比如打开“/dev/video0”这个摄像头；

```
int fd = open("/dev/video0", O_RDWR|O_NONBLOCK); //读写非阻塞
```

2）检查摄像头能力，拍照需要的能力是 V4L2_CAP_VIDEO_CAPTURE；

```
struct v4l2_capability cap; //存储设备信息的结构体` `ioctl(fd, VIDIOC_QUERYCAP, &cap); //fd：打开设备返
```

3）检查相机在某种能力下的输出格式，并选择一个适合的格式；

```
struct v4l2_fmtdesc fmt; //存储帧描述信息的结构体` `ioctl(fd, VIDIOC_ENUM_FMT, &fmt);
```

4）配置相机采集数据的格式；

```
struct v4l2_format format; //帧格式相关参数.` `ioctl(fd, VIDIOC_S_FMT, &format);
```

5）向驱动申请内存缓冲区（内核空间）；

```
struct v4l2_requestbuffers reqbuf;` `ioctl(fd, VIDIOC_REQBUFS, &reqbuf);
```

6）将内核缓冲区映射到用户空间（先查询后映射）；

```
for(int i = 0; i < reqbuf.count; i ++) {` `    struct v4l2_buffer buf;` `    ioctl(fd, VIDIOC_QUER
```

7）将申请的内核缓冲区放入输入队列中；

```
for(int i = 0; i < reqbuf.count; i ++) {` `    struct v4l2_buffer buf;` `    ioctl(fd, VIDIOC_QBUF
```

8）启动相机采集数据；

```
enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;` `ioctl(fd, VIDIOC_STREAMON, &type);
```

9）监测 fd 的读事件，并在可读时读取数据；

```
fd_set fds;` `FD_ZERO(&fds);` `FD_SET(fd, &fds);` `int r = select(fd + 1, &fds, NULL, NULL, &tv);`
```

10）处理相机数据结束，关闭相机；

```
enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;` `ioctl(fd, VIDIOC_STREAMOFF, &type);
```

函数 mmap() 的功能是：把文件映射到进程的虚拟内存空间。通过对这段内存的读取和修改，可以实现对文件的读取和修改，而不需要用 read 和 write 函数。函数原型是：

void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset); @addr：指定映射的起始地址，通常设为 NULL，由内核来分配； @length：代表将文件中映射到内存的部分的长度； @prot：映射区域的保护方式。可以为以下几种方式的组合： PROT_EXEC 映射区域可被执行； PROT_READ 映射区域可被读取； PROT_WRITE 映射区域可被写入； PROT_NONE 映射区域不能存取； @flags：映射区的特性标志位，常用的两个选项是： MAP_SHARD：写入映射区的数据会复制回文件，且与其他进程共享； MAP_PRIVATE：对映射区的写入操作会产生一个映射区的复制，对此区域的修改不会写原文件； @fd：要映射到内存中的文件描述符，由函数 open() 打开文件时返回的值。 @offset：文件映射的偏移量，offset 必须是分页大小的整数倍，设置为 0 代表从文件起始对应。 返回值：实际分配的内存的起始地址。

select、poll、epoll 常用来监测多文件的读写事件，常用于并发服务器管理多个套接字接收的请求数据。上面的相机获取数据过程中，也可以用来监测是否有相机数据产生。select 函数原型如下：

int select(int maxfd, fd_set* readset, fd_set* writeset, fd_set* exceptset, const truct timeval* timeout); @maxfd：监视对象文件描述符数量（范围）； @readset：记录所有可能存在待读数据的文件描述符； @writeset： 记录所有可能无阻塞写入数据的文件描述符； @exceptset：记录所有可能发生异常的文件描述符； @timeout：超时时间； 返回值：错误返回 – 1，超时返回 0。当关注的事件返回时，返回大于 0 的值，该值是发生事件的文件描述符数

预留问题

现在获取到了相机一帧指定格式的图像数据，如何把这个 YUV 图像数据转为图片呢?

全文完