1 CameraManager讲解

`CameraManager` 是系统服务之一，专门用于 **检测** 和 **打开相机**，以及 **获取相机设备特性。**

官方文档其实说的蛮清楚的了，英文好的同学也可以直接看官方文档把：https://developer.android.google.cn/reference/android/hardware/camera2/CameraManager

二、获取 CameraManager 实例

通过 `Context` 类的 `getSystemService()` 方法来获取一个系统服务，参数使用 `Context.CAMERA_SERVICE` 或 `CameraManager.class` 都行。

```
// 方式一
CameraManager manager = (CameraManager)
context.getSystemService(Context.CAMERA_SERVICE);
// 方式二
CameraManager manager = (CameraManager)
context.getSystemService(CameraManager.class);
1234
```

三、内部类

CameraManager 中包含两个公有的内部类，分别为：

1. CameraManager.AvailabilityCallback

当一个相机设备的可用状态发生变化时，就会回调这个类的 `onCameraAvailable(String cameraId)` 和 `onCameraUnavailable(String cameraId)` 方法。

2. CameraManager.TorchCallback

当一个相机设备的闪光灯的 Torch 模式可用状态发生变化时，就会回调这个类的 `onTorchModeChanged(String cameraId, boolean enabled)` 和 `onTorchModeUnavailable(String cameraId)` 方法。

通过 `setTorchMode(String cameraId, boolean enabled)` 方法设置 Torch 模式。

四、常用方法

其中第二条和第三条是重点的重点，一定要掌握的。

1. String[] getCameraIdList()

获取当前连接的相机设备列表，这个 id 通常都是从 `0` 开始并依次递增的。

对于一般的手机而言:

> **后置摄像头一般为 “0”，常量值为** `CameraCharacteristics.LENS_FACING_FRONT`;
> **前置摄像头一般为 “1”，常量值为** `CameraCharacteristics.LENS_FACING_BACK`。

2. CameraCharacteristics getCameraCharacteristics(String cameraId)

根据 cameraId 获取对应相机设备的特征。返回一个 `CameraCharacteristics`，类比于旧 API 中的 `Camera.Parameter` 类，里面封装了相机设备固有的所有属性功能。

3. void openCamera(String cameraId, final CameraDevice.StateCallback callback, Handler handler)

打开指定的相机设备，该方法使用当前进程 uid 继续调用 `openCameraForUid(cameraId, callback, handler, USE_CALLING_UID)` 方法。

参数解释：

- cameraId： 需要打开的相机 id。
- callback： 回调类，常用如下几个回调方法。
  - onOpened(CameraDevice camera) 成功打开时的回调，此时 camera 就准备就绪，并且可以得到一个 `CameraDevice` 实例。
  - onDisconnected(CameraDevice camera) 当 camera 不再可用或打开失败时的回调，通常在该方法中进行资源释放的操作。
  - onError(CameraDevice camera, int error) 当 camera 打开失败时的回调，error 为具体错误原因，定义在 `CameraDevice.StateCallback` 类中。通常在该方法中也要进行资源释放的操作。
- handler： 指定回调执行的线程。传 null 时默认使用当前线程的 Looper，我们通常创建一个后台线程来处理。

使用示例：

```
    private int mCameraId = CameraCharacteristics.LENS_FACING_FRONT;
    private CameraManager mCameraManager; // 相机管理者
    private CameraDevice mCameraDevice; // 相机对象
    private Handler mBackgroundHandler;
    private HandlerThread mBackgroundThread;

    private CameraDevice.StateCallback mStateCallback = new
CameraDevice.StateCallback() {
        @Override
        public void onOpened(@NonNull CameraDevice camera) {
            mCameraDevice = camera;
            // 当相机成功打开时回调该方法，接下来可以执行创建预览的操作
        }
        @Override
        public void onDisconnected(@NonNull CameraDevice camera) {
            // 当相机断开连接时回调该方法，应该在此执行释放相机的操作
        }
        @Override
        public void onError(@NonNull CameraDevice camera, int error) {
            // 当相机打开失败时，应该在此执行释放相机的操作
        }
    };

    public void openCamera() {
        try {
            // 前处理
            mCameraManager.openCamera(Integer.toString(mCameraId),
mStateCallback, mBackgroundHandler);
        } catch (CameraAccessException e) {
```

```
            e.printStackTrace();
        }
    }
```

后台线程创建示例：

```
    private void startBackgroundThread() {
        mBackgroundThread = new HandlerThread("CameraBackground");
        mBackgroundThread.start();
        mBackgroundHandler = new Handler(mBackgroundThread.getLooper());
    }

    private void stopBackgroundThread() {
        mBackgroundThread.quitSafely();
        try {
            mBackgroundThread.join();
            mBackgroundThread = null;
            mBackgroundHandler = null;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
12345678910111213141516
```

4. void registerAvailabilityCallback(AvailabilityCallback callback, Handler handler)

注册一个 `AvailabilityCallback` 回调，handle 指定处理回调的线程，传 null 时默认使用当前线程的 Looper。

```
CameraManager cameraManager = (CameraManager)
context.getSystemService(Context.CAMERA_SERVICE);
cameraManager.registerAvailabilityCallback(new
CameraManager.AvailabilityCallback() {
    @Override
    public void onCameraAvailable(@NonNull String cameraId) {
        super.onCameraAvailable(cameraId);
    }

    @Override
    public void onCameraUnavailable(@NonNull String cameraId) {
        super.onCameraUnavailable(cameraId);
    }
}, mBackgroundHandler);
```

5. void unregisterAvailabilityCallback(AvailabilityCallback callback)

注销 `AvailabilityCallback` 回调，当回调不再需要时一定要注销，否则将带来内存泄漏的问题。

6. void registerTorchCallback(TorchCallback callback, Handler handler)

注册一个 `TorchCallback` 回调，handle 解释如上。

```
CameraManager cameraManager = (CameraManager)
context.getSystemService(Context.CAMERA_SERVICE);
cameraManager.registerTorchCallback(new CameraManager.TorchCallback() {
    @Override
    public void onTorchModeUnavailable(@NonNull String cameraId) {
        super.onTorchModeUnavailable(cameraId);
    }

    @Override
    public void onTorchModeChanged(@NonNull String cameraId, boolean enabled) {
        super.onTorchModeChanged(cameraId, enabled);
    }
}, mBackgroundHandler);
```

7. void unregisterTorchCallback(TorchCallback callback)

注销 `TorchCallback` 回调，当回调不再需要时一定要注销，否则将带来内存泄漏的问题。

8. void setTorchMode(String cameraId, boolean enabled)

打开和关闭指定相机设备的闪光灯功能

2 NV21旋转

> Android Camera对象通过setPreviewCallback 函数，在onPreviewFrame(byte[] data,Camera camera)中回调采集的数据就是NV21格式。而x264编码的输入数据却为I420格式。

因此，当我们采集到摄像头数据之后需要将NV21转为I420。

NV21和I420都是属于YUV420格式。而NV21是一种two-plane模式，即Y和UV分为两个Plane(平面)，但是UV（CbCr）交错存储，2个平面，而不是分为三个。这种排列方式被称之为YUV420SP，而I420则称之为YUV420P。(Y:明亮度、灰度，UV:色度、饱和度)

下图是大小为4x4的NV21数据:Y1、Y2、Y5、Y6共用V1与U1,.....

yuv.png

而I420则是

I420.png

可以看出无论是哪种排列方式，YUV420的数据量都为: w*h*+*w*/2h/2+w/2*h*/2 *即为*wh*3/2

将NV21转位I420则为：

Y数据按顺序完整复制,U数据则是从整个Y数据之后加一个字节再每隔一个字节取一次。

手机摄像头的图像数据来源于摄像头硬件的图像传感器，这个图像传感器被固定到手机上后会有一个默认的取景方向，这个取景方向坐标原点于手机横放时的左上角。当应用是横屏时候：图像传感器方向与屏幕自然方向原点一致。而当手机为竖屏时:

方向.png

传感器与屏幕自然方向不一致，将图像传感器的坐标系逆时针旋转90度，才能显示到屏幕的坐标系上。所以看到的画面是逆时针旋转了90度的，因此我们需要将图像顺时针旋转90度才能看到正常的画面。而Camera对象提供一个`setDisplayOrientation`接口能够设置预览显示的角度：

```
* public static void setCameraDisplayOrientation(Activity activity,
*         int cameraId, android.hardware.Camera camera) {
*     android.hardware.Camera.CameraInfo info =
*             new android.hardware.Camera.CameraInfo();
*     android.hardware.Camera.getCameraInfo(cameraId, info);
*     int rotation = activity.getWindowManager().getDefaultDisplay()
*             .getRotation();
*     int degrees = 0;
*     switch (rotation) {
*         case Surface.ROTATION_0: degrees = 0; break;
*         case Surface.ROTATION_90: degrees = 90; break;
*         case Surface.ROTATION_180: degrees = 180; break;
*         case Surface.ROTATION_270: degrees = 270; break;
*     }
*
*     int result;
*     if (info.facing == Camera.CameraInfo.CAMERA_FACING_FRONT) {
*         result = (info.orientation + degrees) % 360;
*         result = (360 - result) % 360;  // compensate the mirror
*     } else {  // back-facing
*         result = (info.orientation - degrees + 360) % 360;
*     }
*     camera.setDisplayOrientation(result);
* }
```

设置方向.png

根据文档，配置完Camera之后预览确实正常了，但是在onPreviewFrame中回调获得的数据依然是逆时针旋转了90度的。所以如果需要使用预览回调的数据，还需要对onPreviewFrame回调的byte[] 进行旋转。

旋转前：

yuv.png

后置摄像头需要顺时针旋转90度，旋转后：

yuv顺时针旋转90度.png

前置摄像头需要逆时针旋转90度，旋转后：

yuv逆时针旋转90度.png

前置摄像头可能还需要进一步镜像处理，镜像后：

yuv逆时针旋转90度&镜像.png

3 yuv详解

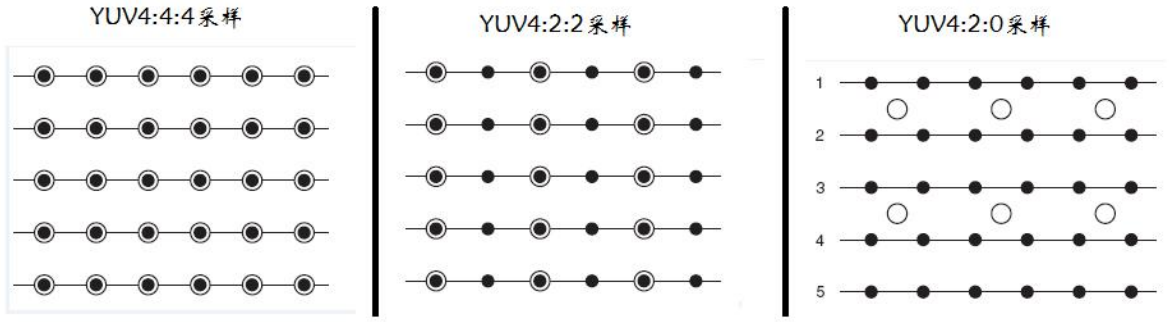
YUV格式有两大类：planar和packed。
对于planar的YUV格式，先连续存储所有像素点的Y，紧接着存储所有像素点的U，随后是所有像素点的V。
对于packed的YUV格式，每个像素点的Y,U,V是连续交*存储的。

YUV，分为三个分量，“Y”表示明亮度（Luminance或Luma），也就是灰度值；而“U”和“V” 表示的则是色度（Chrominance或Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。

与我们熟知的RGB类似，YUV也是一种颜色编码方法，主要用于电视系统以及模拟视频领域，它将亮度信息（Y）与色彩信息（UV）分离，没有UV信息一样可以显示完整的图像，只不过是黑白的，这样的设计很好地解决了彩色电视机与黑白电视的兼容问题。并且，YUV不像RGB那样要求三个独立的视频信号同时传输，所以用YUV方式传送占用极少的频宽。

YUV码流的存储格式其实与其采样的方式密切相关，主流的采样方式有三种，YUV4:4:4，YUV4:2:2，YUV4:2:0，关于其详细原理，可以通过网上其它文章了解，这里我想强调的是如何根据其采样格式来从码流中还原每个像素点的YUV值，因为只有正确地还原了每个像素点的YUV值，才能通过YUV与RGB的转换公式提取出每个像素点的RGB值，然后显示出来。

用三个图来直观地表示采集的方式吧，以黑点表示采样该像素点的Y分量，以空心圆圈表示采用该像素点的UV分量。



先记住下面这段话，以后提取每个像素的YUV分量会用到。

1. YUV 4:4:4采样，每一个Y对应一组UV分量。
2. YUV 4:2:2采样，每两个Y共用一组UV分量。
3. YUV 4:2:0采样，每四个Y共用一组UV分量。

2. 存储方式

下面我用图的形式给出常见的YUV码流的存储方式，并在存储方式后面附有取样每个像素点的YUV数据的方法，其中，Cb、Cr的含义等同于U、V。

（1） YUVY 格式 （属于YUV422）

start + 0:	Y'_{00}	Cb_{00}	Y'_{01}	Cr_{00}	Y'_{02}	Cb_{01}	Y'_{03}	Cr_{01}
start + 8:	Y'_{10}	Cb_{10}	Y'_{11}	Cr_{10}	Y'_{12}	Cb_{11}	Y'_{13}	Cr_{11}
start + 16:	Y'_{20}	Cb_{20}	Y'_{21}	Cr_{20}	Y'_{22}	Cb_{21}	Y'_{23}	Cr_{21}
start + 24:	Y'_{30}	Cb_{30}	Y'_{31}	Cr_{30}	Y'_{32}	Cb_{31}	Y'_{33}	Cr_{31}

YUYV为YUV422采样的存储格式中的一种，相邻的两个Y共用其相邻的两个Cb、Cr，分析，对于像素点Y'00、Y'01 而言，其Cb、Cr的值均为 Cb00、Cr00，其他的像素点的YUV取值依次类推。 **（2） UYVY 格式 （属于YUV422）**

start + 0:	Cb_{00}	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{01}	Y'_{02}	Cr_{01}	Y'_{03}
start + 8:	Cb_{10}	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{11}	Y'_{12}	Cr_{11}	Y'_{13}
start + 16:	Cb_{20}	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{21}	Y'_{22}	Cr_{21}	Y'_{23}
start + 24:	Cb_{30}	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{31}	Y'_{32}	Cr_{31}	Y'_{33}

UYVY格式也是YUV422采样的存储格式中的一种，只不过与YUYV不同的是UV的排列顺序不一样而已，还原其每个像素点的YUV值的方法与上面一样。

（3） YUV422P（属于YUV422）

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cb_{01}		
start + 18:	Cb_{10}	Cb_{11}		
start + 20:	Cb_{20}	Cb_{21}		
start + 22:	Cb_{30}	Cb_{31}		
start + 24:	Cr_{00}	Cr_{01}		
start + 26:	Cr_{10}	Cr_{11}		
start + 28:	Cr_{20}	Cr_{21}		
start + 30:	Cr_{30}	Cr_{31}		

YUV422P也属于YUV422的一种，它是一种Plane模式，即平面模式，并不是将YUV数据交错存储，而是先存放所有的Y分量，然后存储所有的U（Cb）分量，最后存储所有的V（Cr）分量，如上图所示。其每一个像素点的YUV值提取方法也是遵循YUV422格式的最基本提取方法，即两个Y共用一个UV。比如，对于像素点Y'00、Y'01 而言，其Cb、Cr的值均为 Cb00、Cr00。

（4）YV12，YU12格式（属于YUV420）

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cr_{00}	Cr_{01}		
start + 18:	Cr_{10}	Cr_{11}		
start + 20:	Cb_{00}	Cb_{01}		
start + 22:	Cb_{10}	Cb_{11}		

YU12和YV12属于YUV420格式，也是一种Plane模式，将Y、U、V分量分别打包，依次存储。其每一个像素点的YUV数据提取遵循YUV420格式的提取方式，即4个Y分量共用一组UV。注意，上图中，Y'00、Y'01、Y'10、Y'11共用Cr00、Cb00，其他依次类推。

（5）NV12、NV21（属于YUV420）

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 20:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

NV12和NV21属于YUV420格式，是一种two-plane模式，即Y和UV分为两个Plane，但是UV（CbCr）为交错存储，而不是分为三个plane。其提取方式与上一种类似，即Y'00、Y'01、Y'10、Y'11共用Cr00、Cb00

YUV420 planar数据， 以720×488大小图象YUV420 planar为例，

其存储格式是： 共大小为(720×480×3>>1)字节，

分为三个部分:Y,U和V

Y分量： (720×480)个字节

U(Cb)分量：(720×480>>2)个字节

V(Cr)分量：(720×480>>2)个字节

三个部分内部均是行优先存储，三个部分之间是Y,U,V 顺序存储。

即YUV数据的0 - - 720×480字节是Y分量值，

720×480 - - 720×480×5/4字节是U分量

720×480×5/4 - - 720×480×3/2字节是V分量。

4：2： 2 和4：2：0 转换：

最简单的方式：

YUV4:2:2 ---> YUV4:2:0 Y不变，将U和V信号值在行(垂直方向)在进行一次隔行抽样。 YUV4:2:0 ---> YUV4:2:2 Y不变，将U和V信号值的每一行分别拷贝一份形成连续两行数据。

在YUV420中，一个像素点对应一个Y，一个4X4的小方块对应一个U和V。对于所有YUV420图像，它们的Y值排列是完全相同的，因为只有Y的图像就是灰度图像。YUV420sp与YUV420p的数据格式它们的UV排列在原理上是完全不同的。420p它是先把U存放完后，再存放V，也就是说UV它们是连续的。而420sp它是UV、UV这样交替存放的。(见下图) 有了上面的理论，我就可以准确的计算出一个YUV420在内存中存放的大小。 width * hight =Y（总和） U = Y / 4 V = Y / 4

所以YUV420 数据在内存中的长度是 width * hight * 3 / 2，

假设一个分辨率为8X4的YUV图像，它们的格式如下图：

YUV420sp格式如下图

转存失败重新上传取消
转存失败重新上传取消
转存失败重新上传取消

YUV420p数据格式如下图

转存失败重新上传取消
转存失败重新上传取消
转存失败重新上传取消