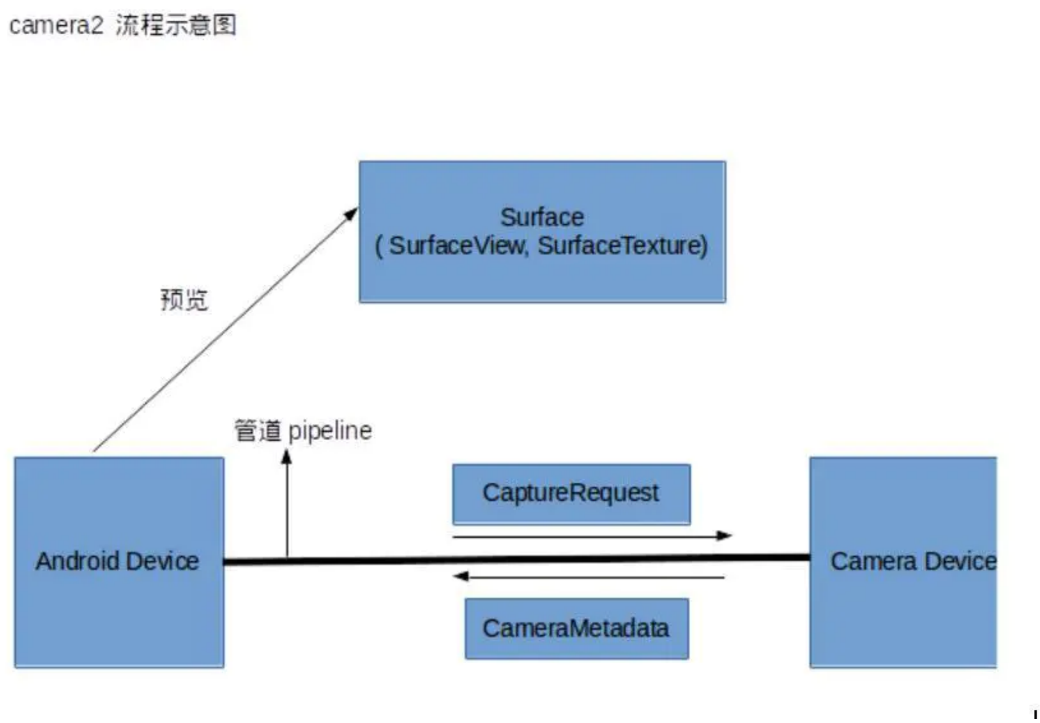
前言

在前几篇文章中介绍了如何调用系统相机拍照和使用Camera1的实现自定义相机拍照、人脸检测等功能

接下来的几篇文章中，我将给大家介绍如何使用Camera2实现自定义相机以及在实现过程中遇到的问题。(实现效果及源码在文末给出)

一、Camera2架构概述

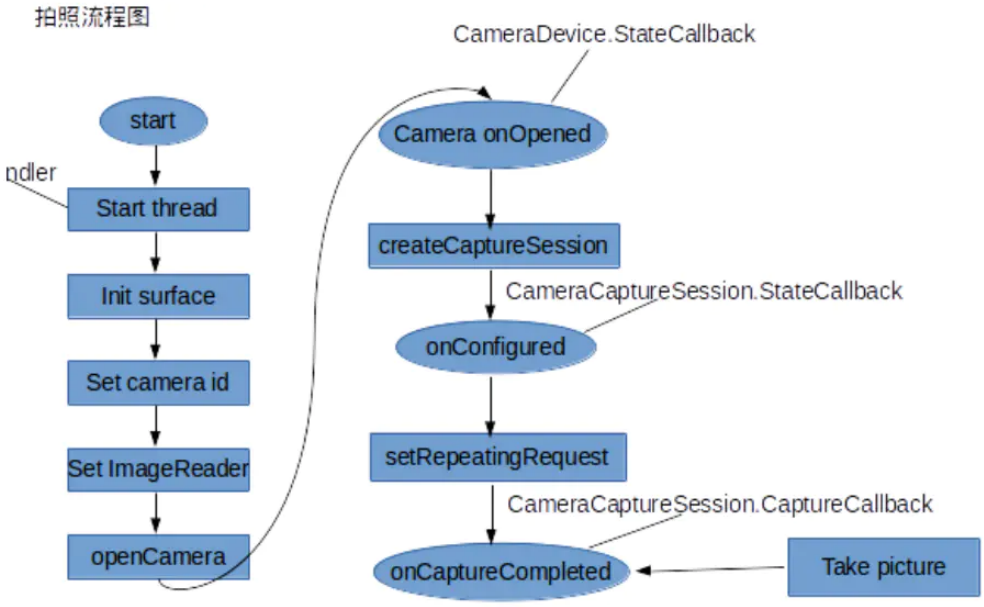
Camera2架构图：



camera2架构.jpg

Camera2引用了管道的概念将安卓设备和摄像头之间联通起来，系统向摄像头发送 Capture 请求，而摄像头会返回 CameraMetadata。这一切建立在一个叫作 CameraCaptureSession 的会话中。

Camera2拍照流程图：



Camera2拍照流程图.png

二、 Camera2中比较重要的类及方法

1. CameraManager

摄像头管理器，用于打开和关闭系统摄像头

- **getCameraIdList()**：
  返回当前设备中可用的相机列表
- **getCameraCharacteristics(String cameraId)**：
  根据摄像头id返回该摄像头的相关信息
- **openCamera(String cameraId, final CameraDevice.StateCallback callback,Handler handler)**:
  打开指定cameraId的相机。参数callback为相机打开时的回调，参数handler为callback被调用时所在的线程

2. CameraDevice

描述系统摄像头，类似于早期的Camera

- **createCaptureRequest(int templateType)**:
  创建一个新的Capture请求。参数templateType代表了请求类型，请求类型一共分为六种，分别为：

> 1. TEMPLATE_PREVIEW : 创建预览的请求
> 2. TEMPLATE_STILL_CAPTURE: 创建一个适合于静态图像捕获的请求，图像质量优先于帧速率
> 3. TEMPLATE_RECORD : 创建视频录制的请求
> 4. TEMPLATE_VIDEO_SNAPSHOT : 创建视视频录制时截屏的请求
> 5. TEMPLATE_ZERO_SHUTTER_LAG : 创建一个适用于零快门延迟的请求。在不影响预览帧率的情况下最大化图像质量
> 6. TEMPLATE_MANUAL : 创建一个基本捕获请求，这种请求中所有的自动控制都是禁用的(自动曝光，自动白平衡、自动焦点)

- **createCaptureSession(List outputs,CameraCaptureSession.StateCallback callback,Handler handler)**:

创建CaptureSession会话。第一个参数 outputs 是一个 List 数组，相机会把捕捉到的图片数据传递给该参数中的 Surface 。第二个参数 StateCallback 是创建会话的状态回调。第三个参数描述了 StateCallback 被调用时所在的线程

3. CameraCharacteristics

描述摄像头的各种特性，类似于Camera1中的CamerInfo。通过CameraManager的getCameraCharacteristics(String cameraId)方法来获取

- **get(Key key)**：
  通过制定的key获取相应的相机参数。

常用的key值有：

> 1. CameraCharacteristics.LENS_FACING：
>   获取摄像头方向。前置摄像头（LENS_FACING_FRONT）或 后置摄像头（LENS_FACING_BACK）
> 2. CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL：
>   获取当前设备支持的相机特性
> 3. CameraCharacteristics.SENSOR_ORIENTATION：
>   获取摄像头方向
> 4. CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP：
>   获取StreamConfigurationMap，它是管理摄像头支持的所有输出格式和尺寸
> 5. CameraCharacteristics.FLASH_INFO_AVAILABLE：
>   是否支持闪光灯
> 6. CameraCharacteristics.STATISTICS_INFO_MAX_FACE_COUNT:
>   同时检测到人脸的数量
> 7. CameraCharacteristics.STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES:
>   相机支持的人脸检测模式

4. CaptureRequest

描述了一次操作请求，拍照、预览等操作都需要先传入CaptureRequest参数，具体的参数控制也是通过CameraRequest的成员变量来设置

- **addTarget(Surface outputTarget)**：
  给此次请求添加一个Surface对象作为图像的输出目标
- **set(Key key, T value)**：
  设置指定的参数值。

```
// 自动对焦
captureRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE)
// 闪光灯
captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH)
// 根据摄像头方向对保存的照片进行旋转，使其为"自然方向"
captureRequestBuilder.set(CaptureRequest.JPEG_ORIENTATION,
mCameraSensorOrientation)
// 人脸检测模式
captureRequestBuilder.set(CaptureRequest.STATISTICS_FACE_DETECT_MODE,
CameraCharacteristics.STATISTICS_FACE_DETECT_MODE_SIMPLE)
```

5. CameraCaptureSession

当需要拍照、预览等功能时，需要先创建该类的实例，然后通过该实例里的方法进行控制（例如：拍照capture()）

- **setRepeatingRequest(CaptureRequest request, CaptureCallback listener, Handler handler)**:
  根据传入的 CaptureRequest 对象开始一个无限循环的捕捉图像的请求。第二个参数 listener 为捕捉图像的回调，在回调中可以拿到捕捉到的图像信息
- **capture(CaptureRequest request, CaptureCallback listener，Handler handler)**:
  拍照。第二个参数为拍照的结果回调

6. CaptureResult

描述拍照完成后的结果

7. ImageReader

用于接收拍照结果和访问拍摄照片的图像数据。
得到一个ImageReader对象的方法为newInstance(int width, int height, int format, int maxImages)。前两个参数是保存图片的宽高，第三个参数为保存图片的格式，第四个参数代表用户可以同时访问到的最大图片数量

> 注意:
> 这个参数应该根据具体需业务需求尽可能的小，因为它的数值越大意味着需要消耗的内存就越高

- **acquireNextImage()**:
  得到ImageReader图像队列中的下一张图片，返回值是一个Image对象

8. Image

一个完整的图片缓存

- **getPlanes()**:
  获取该图像的像素平面数组。这个数组的大小跟图片的格式有关，如 JPEG格式数组大小为1

9. Plane

图像数据的单色平面

- **getBuffer()**:
  获取包含帧数据的ByteBuffer。通过这个ByteBuffer我们就可以把图片保存下来

三、具体实现步骤

一、申请权限:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

二、在xml布局文件中定义一个TextureView

```
<TextureView
  android:id="@+id/textureView"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

三、创建一个CameraHelper类，并给TextureView对象添加回调函数

```
class Camera2Helper(val mActivity: Activity, private val mTextureView:
TextureView) {

    companion object {
        const val PREVIEW_WIDTH = 720         //预览的宽度
        const val PREVIEW_HEIGHT = 1280       //预览的高度
        const val SAVE_WIDTH = 720            //保存图片的宽度
        const val SAVE_HEIGHT = 1280          //保存图片的高度
    }

    private lateinit var mCameraManager: CameraManager
    private var mImageReader: ImageReader? = null
    private var mCameraDevice: CameraDevice? = null
    private var mCameraCaptureSession: CameraCaptureSession? = null

    private var mCameraId = "0"
    private lateinit var mCameraCharacteristics: CameraCharacteristics

    private var mCameraSensorOrientation = 0        //摄像头方向
    private var mCameraFacing = CameraCharacteristics.LENS_FACING_BACK        //
默认使用后置摄像头
    private val mDisplayRotation =
mActivity.windowManager.defaultDisplay.rotation  //手机方向

    private var canTakePic = true                   //是否可以拍照
    private var canExchangeCamera = false           //是否可以切换摄像头

    private var mCameraHandler: Handler
    private val handlerThread = HandlerThread("CameraThread")

    private var mPreviewSize = Size(PREVIEW_WIDTH, PREVIEW_HEIGHT) //预览大小
    private var mSavePicSize = Size(SAVE_WIDTH, SAVE_HEIGHT)       //保存图片大小

    init {
        handlerThread.start()
        mCameraHandler = Handler(handlerThread.looper)

        mTextureView.surfaceTextureListener = object :
TextureView.SurfaceTextureListener {
            override fun onSurfaceTextureSizeChanged(surface: SurfaceTexture?,
width: Int, height: Int) {
```

```
            }

            override fun onSurfaceTextureUpdated(surface: SurfaceTexture?) {
            }

            override fun onSurfaceTextureDestroyed(surface: SurfaceTexture?):
Boolean {
                releaseCamera()
                return true
            }

            override fun onSurfaceTextureAvailable(surface: SurfaceTexture?,
width: Int, height: Int) {
                initCameraInfo()
            }
        }
    }
}
```

各个参数都加的有注释，应该都能看得懂哈~

简单说几点：

1. 因为打开相机和创建会话等都是耗时操作，所以我们启动一个HandlerThread在子线程中来处理
2. 有两个关于尺寸的变量，一个是预览尺寸（在屏幕上显示），一个是保存图片的尺寸（保存到sd卡中图片的尺寸）
3. 有两个方向，一个是手机方向（如果是竖屏应用的话此方向为0），另一个是摄像头方向（一般来说，前置摄像头方向为270，后置摄像头方向为90）

> 注：
> 如果对手机方向和摄像头方向还不太理解的小伙伴，建议看一下Android: Camera相机开发详解(上) —— 知识储备，里面有对这两个方向的讲解。

四、初始化相关参数

```
    /**
     * 初始化
     */
    private fun initCameraInfo() {
        mCameraManager = mActivity.getSystemService(Context.CAMERA_SERVICE) as
CameraManager
        val cameraIdList = mCameraManager.cameraIdList
        if (cameraIdList.isEmpty()) {
            mActivity.toast("没有可用相机")
            return
        }

        for (id in cameraIdList) {
            val cameraCharacteristics =
mCameraManager.getCameraCharacteristics(id)
            val facing =
cameraCharacteristics.get(CameraCharacteristics.LENS_FACING)

            if (facing == mCameraFacing) {
                mCameraId = id
```

```
                mCameraCharacteristics = cameraCharacteristics
            }
            log("设备中的摄像头 $id")
        }

        val supportLevel =
mCameraCharacteristics.get(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL)
        if (supportLevel ==
CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY) {
            mActivity.toast("相机硬件不支持新特性")
        }

        //获取摄像头方向
        mCameraSensorOrientation =
mCameraCharacteristics.get(CameraCharacteristics.SENSOR_ORIENTATION)
        //获取StreamConfigurationMap，它是管理摄像头支持的所有输出格式和尺寸
        val configurationMap =
mCameraCharacteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP
)

        val savePicSize = configurationMap.getOutputSizes(ImageFormat.JPEG)
     //保存照片尺寸
        val previewSize =
configurationMap.getOutputSizes(SurfaceTexture::class.java) //预览尺寸

        val exchange = exchangeWidthAndHeight(mDisplayRotation,
mCameraSensorOrientation)

        mSavePicSize = getBestSize(
                if (exchange) mSavePicSize.height else mSavePicSize.width,
                if (exchange) mSavePicSize.width else mSavePicSize.height,
                if (exchange) mSavePicSize.height else mSavePicSize.width,
                if (exchange) mSavePicSize.width else mSavePicSize.height,
                savePicSize.toList())

        mPreviewSize = getBestSize(
                if (exchange) mPreviewSize.height else mPreviewSize.width,
                if (exchange) mPreviewSize.width else mPreviewSize.height,
                if (exchange) mTextureView.height else mTextureView.width,
                if (exchange) mTextureView.width else mTextureView.height,
                previewSize.toList())

        mTextureView.surfaceTexture.setDefaultBufferSize(mPreviewSize.width,
mPreviewSize.height)

        log("预览最优尺寸 : ${mPreviewSize.width} * ${mPreviewSize.height}, 比例
${mPreviewSize.width.toFloat() / mPreviewSize.height}")
        log("保存图片最优尺寸 : ${mSavePicSize.width} * ${mSavePicSize.height}, 比例
${mSavePicSize.width.toFloat() / mSavePicSize.height}")

        //根据预览的尺寸大小调整TextureView的大小，保证画面不被拉伸
        val orientation = mActivity.resources.configuration.orientation
        if (orientation == Configuration.ORIENTATION_LANDSCAPE)
            mTextureView.setAspectRatio(mPreviewSize.width, mPreviewSize.height)
        else
            mTextureView.setAspectRatio(mPreviewSize.height, mPreviewSize.width)
```

```
        mImageReader = ImageReader.newInstance(mPreviewSize.width,
mPreviewSize.height, ImageFormat.JPEG, 1)
        mImageReader?.setOnImageAvailableListener(onImageAvailableListener,
mCameraHandler)

        if (openFaceDetect)
            initFaceDetect()

        openCamera()
    }


   /**
     * 根据提供的屏幕方向 [displayRotation] 和相机方向 [sensorOrientation] 返回是否需要
交换宽高
     */
    private fun exchangeWidthAndHeight(displayRotation: Int, sensorOrientation:
Int): Boolean {
        var exchange = false
        when (displayRotation) {
            Surface.ROTATION_0, Surface.ROTATION_180 ->
                if (sensorOrientation == 90 || sensorOrientation == 270) {
                    exchange = true
                }
            Surface.ROTATION_90, Surface.ROTATION_270 ->
                if (sensorOrientation == 0 || sensorOrientation == 180) {
                    exchange = true
                }
            else -> log("Display rotation is invalid: $displayRotation")
        }

        log("屏幕方向  $displayRotation")
        log("相机方向  $sensorOrientation")
        return exchange
    }


    /**
     *
     * 根据提供的参数值返回与指定宽高相等或最接近的尺寸
     *
     * @param targetWidth    目标宽度
     * @param targetHeight   目标高度
     * @param maxWidth       最大宽度(即TextureView的宽度)
     * @param maxHeight      最大高度(即TextureView的高度)
     * @param sizeList       支持的Size列表
     *
     * @return  返回与指定宽高相等或最接近的尺寸
     *
     */
    private fun getBestSize(targetWidth: Int, targetHeight: Int, maxWidth: Int,
maxHeight: Int, sizeList: List<Size>): Size {
        val bigEnough = ArrayList<Size>()     //比指定宽高大的Size列表
        val notBigEnough = ArrayList<Size>()  //比指定宽高小的Size列表

        for (size in sizeList) {

            //宽<=最大宽度  &&  高<=最大高度  &&  宽高比 == 目标值宽高比
```

```
            if (size.width <= maxWidth && size.height <= maxHeight
                    && size.width == size.height * targetWidth / targetHeight) {

                if (size.width >= targetWidth && size.height >= targetHeight)
                    bigEnough.add(size)
                else
                    notBigEnough.add(size)
            }
            log("系统支持的尺寸: ${size.width} * ${size.height} ,  比例 :
${size.width.toFloat() / size.height}")
        }

        log("最大尺寸 : $maxWidth * $maxHeight, 比例 : ${targetWidth.toFloat() /
targetHeight}")
        log("目标尺寸 : $targetWidth * $targetHeight, 比例 :
${targetWidth.toFloat() / targetHeight}")

        //选择bigEnough中最小的值  或 notBigEnough中最大的值
        return when {
            bigEnough.size > 0 -> Collections.min(bigEnough,
CompareSizesByArea())
            notBigEnough.size > 0 -> Collections.max(notBigEnough,
CompareSizesByArea())
            else -> sizeList[0]
        }
    }
```

这个方法有点长，不过思路还是很清晰的。主要做了以下几件事：

1. 首先，通过mActivity.getSystemService(Context.CAMERA_SERVICE) as CameraManager 获取到CameraManager 实例
2. 通过循环遍历设备中可用的相机，通过 mCameraManager.getCameraCharacteristics(id) 获取到相机的各种信息
3. mCameraCharacteristics.get(CameraCharacteristics.SENSOR_ORIENTATION) 获取到相机传感器的方向
4. 通过 configurationMap.getOutputSizes(ImageFormat.JPEG) 和 configurationMap.getOutputSizes(SurfaceTexture::class.java) 获取到相机支持的预览尺寸和保存图片的尺寸
5. exchangeWidthAndHeight(displayRotation: Int, sensorOrientation: Int)方法的作用是根据屏幕方向和摄像头方向确定是否需要交换宽高

> 比如我们手机竖屏放置，设置的预览宽高是 720 * 1280 ，我们希望设置的是宽为 720，高为 1280 。 而后置摄像头相对于竖直方向是 90°，也就说 720 相对于是摄像头来说是它的高度，1280 是它的宽度，这跟我们想要设置的刚好相反。所以，我们通过exchangeWidthAndHeight这个方法得出来是否需要交换宽高值，如果需要，那变成了把 1280 * 720 设置给摄像头，即它的宽为 720，高为 1280 。这样就与我们预期的宽高值一样了

1. 通过 getBestSize(targetWidth: Int, targetHeight: Int, maxWidth: Int, maxHeight: Int, sizeList: List) 方法获取到最优的宽和高。 根据传入的 目标宽高值、最大宽高值（即屏幕大小）和 相机支持的尺寸列表，从相机支持的尺寸列表中得到一个最优值。
2. 通过mTextureView.surfaceTexture.setDefaultBufferSize(mPreviewSize.width, mPreviewSize.height) 方法用来设置TextureView的预览尺寸
3. mImageReader = ImageReader.newInstance(mSavePicSize.width, mSavePicSize.height, ImageFormat.JPEG, 1)
   mImageReader?.setOnImageAvailableListener(onImageAvailableListener, mCameraHandler)

创建一个ImageReader对象，并设置回调函数。前两个参数代表保存图片的宽高，第三个参数是保存图片的格式，第四个参数代表用户同时可以得到的图片最大数

在onImageAvailableListener中处理得到的图像数据，具体代码在后面给出

五、打开相机

```
    /**
     * 打开相机
     */
    private fun openCamera() {

        if (ContextCompat.checkSelfPermission(mActivity,
Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
            mActivity.toast("没有相机权限！")
            return
        }

        mCameraManager.openCamera(mCameraId, object :
CameraDevice.StateCallback() {
            override fun onOpened(camera: CameraDevice) {
                log("onOpened")
                mCameraDevice = camera
                createCaptureSession(camera)
            }

            override fun onDisconnected(camera: CameraDevice) {
                log("onDisconnected")
            }

            override fun onError(camera: CameraDevice, error: Int) {
                log("onError $error")
                mActivity.toast("打开相机失败！$error")
            }
        }, mCameraHandler)
    }
```

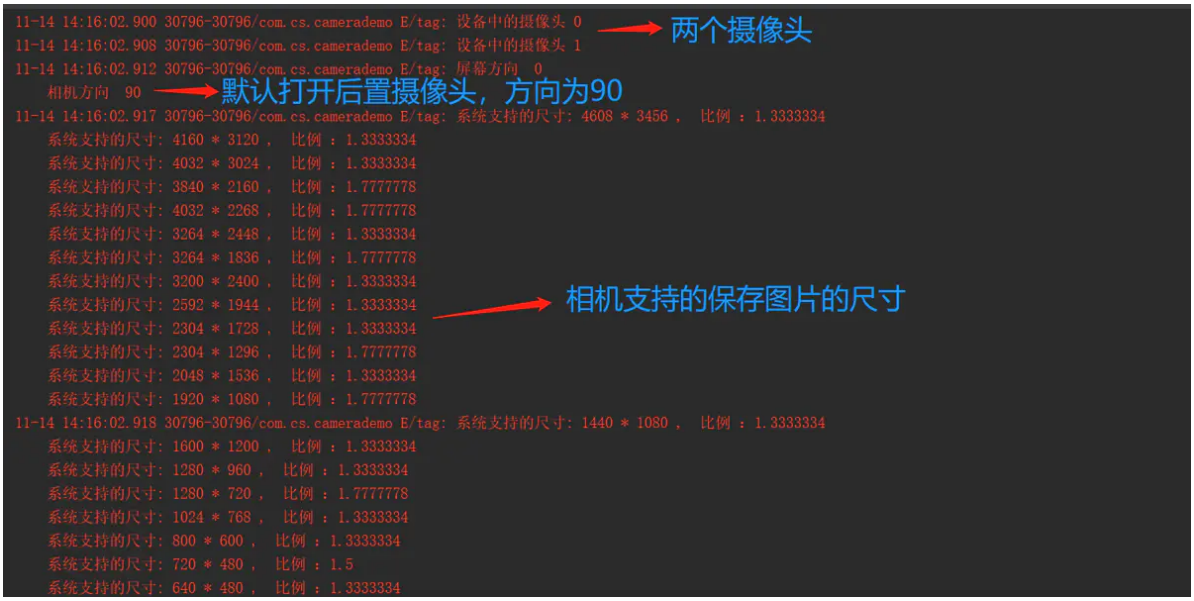
六、创建预览会话

```
 /**
     * 创建预览会话
     */
    private fun createCaptureSession(cameraDevice: CameraDevice) {

        val captureRequestBuilder =
cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW)

        val surface = Surface(mTextureView.surfaceTexture)
        captureRequestBuilder.addTarget(surface)  // 将CaptureRequest的构建器与
Surface对象绑定在一起
        captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH)      // 闪光灯
```

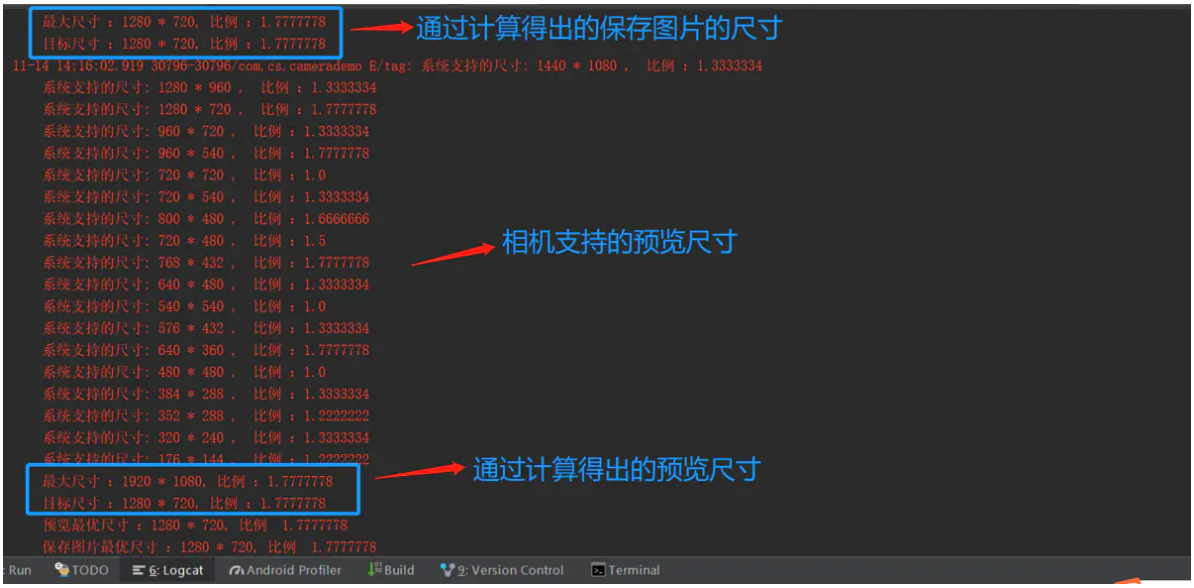
```
        captureRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE) // 自动对焦

        // 为相机预览，创建一个CameraCaptureSession对象
        cameraDevice.createCaptureSession(arrayListOf(surface,
mImageReader?.surface), object : CameraCaptureSession.StateCallback() {
            override fun onConfigureFailed(session: CameraCaptureSession?) {
                mActivity.toast("开启预览会话失败！")
            }

            override fun onConfigured(session: CameraCaptureSession) {
                mCameraCaptureSession = session
                session.setRepeatingRequest(captureRequestBuilder.build(),
mCaptureCallBack, mCameraHandler)
            }

        }, mCameraHandler)
    }

    private val mCaptureCallBack = object :
CameraCaptureSession.CaptureCallback() {

        override fun onCaptureCompleted(session: CameraCaptureSession, request:
CaptureRequest?, result: TotalCaptureResult) {
            super.onCaptureCompleted(session, request, result)
            canExchangeCamera = true
            canTakePic = true
        }

        override fun onCaptureFailed(session: CameraCaptureSession?, request:
CaptureRequest?, failure: CaptureFailure?) {
            super.onCaptureFailed(session, request, failure)
            log("onCaptureFailed")
            mActivity.toast("开启预览失败！")
        }
    }
```

1. 通过cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW) 创建一个用于预览的Builder对象
2. 为该Builder对象添加一个Surface对象，并设置各种相关参数
3. 通过cameraDevice.createCaptureSession创建一个会话，第一个参数中传了一个 surface 和 mImageReader?.surface。这表明了这次会话的图像数据的输出到这两个对象
4. 当会话创建成功时，通过 session.setRepeatingRequest(captureRequestBuilder.build(), mCaptureCallBack, mCameraHandler) 发起预览请求

到这一步，程序已经能够正常跑起来了。下面是我的手机跑起来时打印的日志：



手机log1.png



手机log2.png

> **注意：**
> Camera2在一些低端机器上会出现预览画面拉伸问题。
> 在android 5.0，硬件兼容级别为legacy时，Camera2输出的宽高比和Camera Sensor保持一致。
> 也就是说我们设置的预览宽高 720 * 1280 并不起作用，所以出现了画面拉伸。
> 对于这个问题，我在网上看到的答案是如果遇到这种情况放弃使用Camra2，使用旧的Camera1。
> 这并不是一种优雅的解决方法，如果小伙们们有更好的解决方法的话欢迎提出来

七、拍照、保存

```
    /**
     * 拍照
     */
    fun takePic() {
        if (mCameraDevice == null || !mTextureView.isAvailable || !canTakePic)
return

        mCameraDevice?.apply {
```

```
            val captureRequestBuilder =
createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE)
            captureRequestBuilder.addTarget(mImageReader?.surface)

            captureRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE) // 自动对焦
            captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH)     // 闪光灯
            captureRequestBuilder.set(CaptureRequest.JPEG_ORIENTATION,
mCameraSensorOrientation)      //根据摄像头方向对保存的照片进行旋转，使其为"自然方向"
            mCameraCaptureSession?.capture(captureRequestBuilder.build(), null,
mCameraHandler)
                    ?: mActivity.toast("拍照异常！")
        }
    }

    private val onImageAvailableListener = OnImageAvailableListener {

        val image = it.acquireNextImage()
        val byteBuffer = image.planes[0].buffer
        val byteArray = ByteArray(byteBuffer.remaining())
        byteBuffer.get(byteArray)
        it.close()
        BitmapUtils.savePic(byteArray, mCameraSensorOrientation == 270, {
savedPath, time ->
            mActivity.runOnUiThread {
                mActivity.toast("图片保存成功！ 保存路径：$savedPath 耗时：$time")
            }
        }, { msg ->
            mActivity.runOnUiThread {
                mActivity.toast("图片保存失败！ $msg")
            }
        })
    }
```

1. 通过createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE) 创建一个拍照请求的Builder对象
2. 然后设置各种参数。注意，captureRequestBuilder.set(CaptureRequest.JPEG_ORIENTATION, mCameraSensorOrientation)用来设置保存照片的旋转方向。如果不设置的话，保存的照片不是"自然方向"

> 在 Android: Camera相机开发详解(上) —— 知识储备 中有讲解

1. 拍照的结果是在 OnImageAvailableListener 对象中得到的。首先通过 acquireNextImage() 方法获取到一个Image对象，然后通过 image.planes[0].buffer 得到 byteBuffer，将这个 byteBuffer 转换成 byteArray 。这个 byteArray 就是拍照所得到的图像数据。然后就可以把这个 byteArray 保存成图片到手机存储中

八、 释放相机及线程

```
    fun releaseCamera() {
        mCameraCaptureSession?.close()
        mCameraCaptureSession = null
```

```
        mCameraDevice?.close()
        mCameraDevice = null

        mImageReader?.close()
        mImageReader = null

        canExchangeCamera = false
    }

    fun releaseThread() {
        handlerThread.quitSafely()
    }
```

效果展示

效果展示.gif

完整代码

https://github.com/smashinggit/Study

> 注：此工程包含多个module，本文所用代码均在CameraDemo下

在下一篇文章中将会介绍如何使用Camera2实现人脸检测及实时显示人脸位置功能

获取 CameraId 列表通过调用 CameraManager 类 getCameraIdList() 实现。

getCameraIdList() 按标识符返回当前连接的摄像头设备列表，包括其他 camera API 客户端可能正在使用的摄像头。

不可移动摄像头的标识符使用以 0 开头的整数，而可移动摄像头即使是同一型号，也为每个设备都分配唯一的标识符。