date: 2017-01-18 16:00:24 title: Camera2录像流程 # 这是标题 description: Camera2录像流程 categories: # 这里写的分类会自动汇集到 categories 页面上，分类可以多级

- 博客
- Camera2 tags: # 这里写的标签会自动汇集到 tags 页面上
- 博客

Android M版本及以上需要申请权限

申请 `CAMERA` 和 `RECORD_AUDIO` 权限。

在TextureView可用的时候，打开Camera，通过 `CameraDevice.StateCallback` 回调获取打开结果。

其中openCamera函数声明如下（ `cameraId`： Camera的id，通常0表示后置摄像头，1表示前置摄像头； `callback`: `CameraDevice.StateCallback` 相机状态回调; `handler` :执行回调操作的hanlder，为空时，在主线程中执行，如果传入异步线程的handler,则在异步线程中执行）：

```
@RequiresPermission(android.Manifest.permission.CAMERA)
   public void openCamera(@NonNull String cameraId,
           @NonNull final CameraDevice.StateCallback callback, @Nullable Handler
handler)
           throws CameraAccessException {
           }
```

完整打开Camera代码如下:

```
/**
 * Tries to open a {@link CameraDevice}. The result is listened by `mStateCallback`.
 */
private void openCamera(int width, int height) {
    if (!hasPermissionsGranted(VIDEO_PERMISSIONS)) {
        requestVideoPermissions();
        return;
    }
    final Activity activity = getActivity();
    if (null == activity || activity.isFinishing()) {
        return;
    }
    //获取CameraManager对象
    CameraManager manager = (CameraManager)
activity.getSystemService(Context.CAMERA_SERVICE);
    try {
        Log.d(TAG, "tryAcquire");
        //做打开超时判断，超过2.5秒则抛出异常，对应release()结束
        if (!mCameraOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS)) {
            throw new RuntimeException("Time out waiting to lock camera opening.");
        }
        String cameraId = manager.getCameraIdList()[0];//获取后摄id

        // Choose the sizes for camera preview and video recording
        CameraCharacteristics characteristics =
manager.getCameraCharacteristics(cameraId);
        //类似之前的Parameters
        StreamConfigurationMap map = characteristics
                .get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
        mSensorOrientation =
characteristics.get(CameraCharacteristics.SENSOR_ORIENTATION);
        //选择录像尺寸
        mVideoSize = chooseVideoSize(map.getOutputSizes(MediaRecorder.class));
        //选择合适的预览尺寸
        mPreviewSize = chooseOptimalSize(map.getOutputSizes(SurfaceTexture.class),
                width, height, mVideoSize);

        int orientation = getResources().getConfiguration().orientation;
        if (orientation == Configuration.ORIENTATION_LANDSCAPE) {
            mTextureView.setAspectRatio(mPreviewSize.getWidth(),
mPreviewSize.getHeight());
        } else {
            mTextureView.setAspectRatio(mPreviewSize.getHeight(),
mPreviewSize.getWidth());
        }
        configureTransform(width, height);
        mMediaRecorder = new MediaRecorder();
        //打开Camera，通过CameraDevice.StateCallback回调打开结果
        manager.openCamera(cameraId, mStateCallback, null);
    } catch (CameraAccessException e) {
        Toast.makeText(activity, "Cannot access the camera.",
Toast.LENGTH_SHORT).show();
        activity.finish();
    } catch (NullPointerException e) {
        // Currently an NPE is thrown when the Camera2API is used but not supported
on the
        // device this code runs.
        ErrorDialog.newInstance(getString(R.string.camera_error))
                .show(getChildFragmentManager(), FRAGMENT_DIALOG);
    } catch (InterruptedException e) {
        throw new RuntimeException("Interrupted while trying to lock camera
opening.");
    }
}
```

调用 `openCamera` 方法后会回调 `CameraDevice.StateCallback` 这个接口，在该接口里重写 `onOpened` 函数，开启预览。

```
/**
 * {@link CameraDevice.StateCallback} is called when {@link CameraDevice} changes its
status.
 */
private CameraDevice.StateCallback mStateCallback = new CameraDevice.StateCallback()
{
    @Override
    public void onOpened(CameraDevice cameraDevice) {//打开Camera成功
        mCameraDevice = cameraDevice;
        startPreview();//开始预览
        mCameraOpenCloseLock.release();//释放掉超时判断
        if (null != mTextureView) {
            configureTransform(mTextureView.getWidth(), mTextureView.getHeight());
        }
    }

    @Override
    public void onDisconnected(CameraDevice cameraDevice) {//Camera断开连接
        mCameraOpenCloseLock.release();
        cameraDevice.close();
        mCameraDevice = null;
    }

    @Override
    public void onError(CameraDevice cameraDevice, int error) {//打开Camera失败
        mCameraOpenCloseLock.release();
        cameraDevice.close();
        mCameraDevice = null;
        Activity activity = getActivity();
        if (null != activity) {
            activity.finish();
        }
    }

};
```

开启预览主要就是使用 `CameraDevice` 来创建会话 `createCaptureSession` 。

声明如下（ `outputs` :预览输出载体，比如TextureView的getSurfaceTexture()，拍照时ImageReader的getSurface()，录像时MediaRecorder的getSurface()； `callback` :摄像头采集状态回调； `handler` :同openCamera）：

```
public abstract void createCaptureSession(@NonNull List<Surface> outputs,
        @NonNull CameraCaptureSession.StateCallback callback, @Nullable Handler
handler)
        throws CameraAccessException;
```

完整的startPreview函数如下：

```
/**
 * Start the camera preview.
 */
private void startPreview() {
    if (null == mCameraDevice || !mTextureView.isAvailable() || null == mPreviewSize)
{
        return;
    }
    try {
        closePreviewSession();
        SurfaceTexture texture = mTextureView.getSurfaceTexture();//获取TextureView的
SurfaceTexture，作为预览输出载体
        assert texture != null;
        texture.setDefaultBufferSize(mPreviewSize.getWidth(),
mPreviewSize.getHeight());
        //创建CameraRequest.Builder，当程序调用setRepeatingRequest()方法进行预览时，
        // 或调用capture()方法进行拍照时，都需要传入CameraRequest参数.
        // CameraRequest代表了一次捕获请求，用于描述捕获图片的各种参数设置，
        // 比如对焦模式、曝光模式……程序需要对照片所做的各种控制，都通过CameraRequest参数进行设
置。
        // 可以理解一个请求参数一样，CameraRequest.Builder则负责生成CameraRequest对象。
        mPreviewBuilder =
mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);

        Surface previewSurface = new Surface(texture);
        mPreviewBuilder.addTarget(previewSurface);

        mCameraDevice.createCaptureSession(Arrays.asList(previewSurface), new
CameraCaptureSession.StateCallback() {

                @Override
                public void onConfigured(CameraCaptureSession cameraCaptureSession) {
                    mPreviewSession = cameraCaptureSession;
                    mPreviewBuilder.set(CaptureRequest.CONTROL_MODE,
CameraMetadata.CONTROL_MODE_AUTO);
                    //不停的发送获取图像请求，完成连续预览
                    mPreviewSession.setRepeatingRequest(mPreviewBuilder.build(),
null, mBackgroundHandler);
                }

                @Override
                public void onConfigureFailed(CameraCaptureSession
cameraCaptureSession) {
                    Activity activity = getActivity();
                    if (null != activity) {
                        Toast.makeText(activity, "Failed",
Toast.LENGTH_SHORT).show();
                    }
                }
            }, mBackgroundHandler);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }
```

setRepeatingRequest和capture方法其实都是向相机设备发送获取图像的请求，但是capture就获取那么一次，而setRepeatingRequest就是不停的获取图像数据，所以呢，使用capture就像拍照一样，图像就停在那里，但是setRepeatingRequest一直在发送和获取，所以需要连拍的时候就调用它，然后在onCaptureCompleted中保存图像就行了。（注意了，图像的预览也是用的setRepeatingRequest，只是不处理数据）

至此，Camera已经完成预览，等待用户输入开始录像事件。

当用户点击录像按钮时，先配置好`MediaRecorder`相关参数，然后重启预览，并在预览开始时调用`MediaRecorder.start()`,正式开始录像：

```
private void startRecordingVideo() {
    if (null == mCameraDevice || !mTextureView.isAvailable() || null == mPreviewSize)
{
        return;
    }
    try {
        closePreviewSession();//先关闭预览，因为需要添加一个预览输入载体到
MediaRecorder.getSurface()
        setUpMediaRecorder();//设置MediaRecorder相关参数
        SurfaceTexture texture = mTextureView.getSurfaceTexture();//获取TextureView的
输出载体
        assert texture != null;
        texture.setDefaultBufferSize(mPreviewSize.getWidth(),
mPreviewSize.getHeight());
        //准备创建CaptureRequest
        mPreviewBuilder =
mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_RECORD);
        List<Surface> surfaces = new ArrayList<>();

        // Set up Surface for the camera preview
        Surface previewSurface = new Surface(texture);
        surfaces.add(previewSurface);
        mPreviewBuilder.addTarget(previewSurface);

        // Set up Surface for the MediaRecorder
        mRecorderSurface = mMediaRecorder.getSurface();//获取MediaRecorder预览输出载体
        surfaces.add(mRecorderSurface);
        mPreviewBuilder.addTarget(mRecorderSurface);

        // Start a capture session
        // Once the session starts, we can update the UI and start recording
        mCameraDevice.createCaptureSession(surfaces, new
CameraCaptureSession.StateCallback() {

            @Override
            public void onConfigured(@NonNull CameraCaptureSession
cameraCaptureSession) {
                mPreviewSession = cameraCaptureSession;
                mPreviewBuilder.set(CaptureRequest.CONTROL_MODE,
CameraMetadata.CONTROL_MODE_AUTO);
                //不停的发送获取图像请求，完成连续预览
                mPreviewSession.setRepeatingRequest(mPreviewBuilder.build(), null,
mBackgroundHandler);
                getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        // UI
                        mButtonVideo.setText(R.string.stop);
                        mIsRecordingVideo = true;

                        // Start recording
                        mMediaRecorder.start();//正式开始录像
                    }
                });
            }

            @Override
            public void onConfigureFailed(@NonNull CameraCaptureSession
cameraCaptureSession) {
                Activity activity = getActivity();
                if (null != activity) {
                    Toast.makeText(activity, "Failed", Toast.LENGTH_SHORT).show();
                }
            }
        }, mBackgroundHandler);
    } catch (CameraAccessException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

当用户再点击停止录像按钮时，就停止录像了，释放掉一些资源：

```
private void stopRecordingVideo() {
    // UI
    mIsRecordingVideo = false;
    mButtonVideo.setText(R.string.record);
    // Stop recording
    mMediaRecorder.stop();
    mMediaRecorder.reset();

    Activity activity = getActivity();
    if (null != activity) {
        Toast.makeText(activity, "Video saved: " + mNextVideoAbsolutePath,
                Toast.LENGTH_SHORT).show();
        Log.d(TAG, "Video saved: " + mNextVideoAbsolutePath);
    }
    mNextVideoAbsolutePath = null;
    startPreview();
}
```