

# Android手机直播（四）Android Media API



风从影 (/u/c3dcf28e69f2) [+ 关注](#)

2017.07.14 11:41\* 字数 4046 阅读 2479 评论 21 喜欢 56 赞赏 2

(/u/c3dcf28e69f2)

## 一、文章说明

最近工作实在太忙，很久没有更新文章了，收到很多小伙伴催更的消息，心中实在惭愧，趁着今天有空赶紧更新。

第一篇文章从总体上介绍了Android手机直播，之后两篇文章分别介绍了视频和音频采集，这篇文章便开始介绍编解码相关的知识。Android提供很多和视音频处理相关的类，熟练使用这些相关的类，其实是能实现很强大的功能。

视音频编解码一般分为两种，一种是硬编实现，一种是软编实现。这两种方式各有优缺点，硬编性能好，但是需要对兼容性进行相应处理；软编兼容性好，可以进行一些参数设置，但是软编一般性能较差，引入相关的编解码库往往会增大app的整体体积，而且还需要写相应的jni接口。

这篇文章主要讲述使用Android原生提供的各种类来实现对视音频进行处理，相信各位看完完整篇文章后会感受到这几个类配合使用的强大。

直播项目已经开源，开源地址：SopCastComponent (<https://github.com/LaiFeng-Android/SopCastComponent>)

Github地址：<https://github.com/SuperJim123> (<https://github.com/SuperJim123>)

## 二、几个类



很多时候我们往往会忽略很多事情，比如说Android系统已经给我们提供了对音视频的强大支持，我们往往还不知道，没有专心去研究。这篇文章先介绍几个和音视频相关的类，通过这几个类的组合使用，其实是能变换出许多音视频处理的相关功能，下面就对这几个类进行简单介绍。

(/apps/download?  
utm\_source=sbc)



MediaMetadataRetriever：用来获取视频的相关信息，例如视频宽高、时长、旋转角度、码率等等。

MediaExtractor：音视频分离器，将一些格式的视频分离出视频轨道和音频轨道。

MediaCodec：音视频相应的编解码类。

MediaMuxer：音视频合成器，将视频和音频合成相应的格式。

MediaFormat：音视频相应的格式信息。

MediaCodec.BufferInfo：存放ByteBuffer相应信息的类。

MediaCrypto：音视频加密解密处理的类。

MediaCodecInfo：音视频编解码相关信息的类。

MediaFormat和MediaCodec.BufferInfo是串起上面几个类的桥梁，上面几个音视频处理的类通过这两个桥梁建立起联系，从而变化出相应的功能，认真分析的话会感觉到Google设计的精妙。

### 三、MediaMetadataRetriever

MediaMetadataRetriever用来获取音视频的相关信息，MediaMetadataRetriever的使用十分简单，传入相应的文件路径创建MediaMetadataRetriever，之后便可以得到视频的相关参数。



```
MediaMetadataRetriever metadataRetriever = new MediaMetadataRetriever();
metadataRetriever.setDataSource(file.getAbsolutePath());
String widthString = metadataRetriever.extractMetadata(MediaMetadataRetriever.METADATA_EXTRACTOR_WIDTH);
if(!TextUtils.isEmpty(widthString)) {
    width = Integer.valueOf(widthString);
}
String heightString = metadataRetriever.extractMetadata(MediaMetadataRetriever.METADATA_EXTRACTOR_HEIGHT);
if(!TextUtils.isEmpty(heightString)) {
    height = Integer.valueOf(heightString);
}
String durationString = metadataRetriever.extractMetadata(MediaMetadataRetriever.METADATA_EXTRACTOR_DURATION);
if(!TextUtils.isEmpty(durationString)) {
    duration = Long.valueOf(durationString);
}
String bitrateString = metadataRetriever.extractMetadata(MediaMetadataRetriever.METADATA_EXTRACTOR_BITRATE);
if(!TextUtils.isEmpty(bitrateString)) {
    bitrate = Integer.valueOf(bitrateString);
}
String degreeStr = metadataRetriever.extractMetadata(MediaMetadataRetriever.METADATA_EXTRACTOR_ORIENTATION);
if (!TextUtils.isEmpty(degreeStr)) {
    degree = Integer.valueOf(degreeStr);
}
metadataRetriever.release();
```

(/apps/download?  
utm\_source=sbc)



## 四、MediaExtractor

MediaExtractor用来对视音频进行分离，对文件中的视频音频轨道进行分离能做大量的事情，比如说要写一个播放器，那么首先的第一个步骤是分离出视频音频轨道，然后进行相应的处理。MediaExtractor的创建和MediaMetadataRetriever一样十分简单，只需要传入相应的文件路径。通过getTrackCount()可以得到相应的轨道数量，一般情况下视音频轨道都有，有些时候可能只有视频，有些时候可能只有音频。轨道的序号从0开始，通过getTrackFormat(int index)方法可以得到相应的MediaFormat，而通过MediaFormat可以判断出轨道是视频还是音频。通过selectTrack(int index)方法选择相应序号的轨道。



```
public static MediaExtractor createExtractor(String path) throws IOException {
    MediaExtractor extractor;
    File inputFile = new File(path);    // must be an absolute path
    if (!inputFile.canRead()) {
        throw new FileNotFoundException("Unable to read " + inputFile);
    }
    extractor = new MediaExtractor();
    extractor.setDataSource(inputFile.toString());
    return extractor;
}

public static String getMimeTypeFor(MediaFormat format) {
    return format.getString(MediaFormat.KEY_MIME);
}

public static int getAndSelectVideoTrackIndex(MediaExtractor extractor) {
    for (int index = 0; index < extractor.getTrackCount(); ++index) {
        if (isVideoFormat(extractor.getTrackFormat(index))) {
            extractor.selectTrack(index);
            return index;
        }
    }
    return -1;
}

public static int getAndSelectAudioTrackIndex(MediaExtractor extractor) {
    for (int index = 0; index < extractor.getTrackCount(); ++index) {
        if (isAudioFormat(extractor.getTrackFormat(index))) {
            extractor.selectTrack(index);
            return index;
        }
    }
    return -1;
}

public static boolean isVideoFormat(MediaFormat format) {
    return getMimeTypeFor(format).startsWith("video/");
}

public static boolean isAudioFormat(MediaFormat format) {
    return getMimeTypeFor(format).startsWith("audio/");
}
```

(/apps/download?  
utm\_source=sbc)



选择好一个轨道后，便可以通过相应方法提取出相应轨道的数据。

extractor.seekTo(startTime, SEEK\_TO\_PREVIOUS\_SYNC)方法可以直接跳转到开始解析的位置。extractor.readSampleData(byteBuffer, 0)方法则可以将数据解析到byteBuffer



中。extractor.advance()方法则将解析位置进行前移，准备下一次解析。

下面是MediaExtractor一般的使用方法。

```
MediaExtractor extractor = new MediaExtractor();
extractor.setDataSource(...);
int numTracks = extractor.getTrackCount();
for (int i = 0; i < numTracks; ++i) {
    MediaFormat format = extractor.getTrackFormat(i);
    String mime = format.getString(MediaFormat.KEY_MIME);
    if (weAreInterestedInThisTrack) {
        extractor.selectTrack(i);
    }
}
ByteBuffer inputBuffer = ByteBuffer.allocate(...)
while (extractor.readSampleData(inputBuffer, ...) >= 0) {
    int trackIndex = extractor.getSampleTrackIndex();
    long presentationTimeUs = extractor.getSampleTime();
    ...
    extractor.advance();
}

extractor.release();
extractor = null;
```

(/apps/download?  
utm\_source=sbc)



## 五、MediaCodec

MediaCodec是Android视音频里面最为重要的类，它主要实现的功能是对视音频进行编解码处理。在编码方面，可以对采集的视音频数据进行编码处理，这样的话可以对数据进行压缩，从而实现以较少的数据量存储视音频信息。在解码方面，可以解码相应格式的视音频数据，从而得到原始的可以渲染的数据，从而实现视音频的播放。

一般场景下音频使用的是AAC-LC的格式，而视频使用的是H264格式。这两种格式在MediaCodec支持的版本（Api 16）也都得到了很好的支持。在直播过程中，先采集视频和音频数据，然后将原始的数据塞给编码器进行硬编，然后得到相应的编码后的AAC-LC和H264数据。

在Android系统中，MediaCodec支持的格式有限，在使用MediaCodec之前需要对硬编类型的支持进行检测，如果MediaCodec支持再进行使用。



## 1、检查

在使用硬编编码器之前需要对编码器支持的格式进行检查，在Android中可以使用MediaCodecInfo这个类来获取系统对音视频硬编的支持情况。

下面的代码是判断MediaCodec是否支持某个MIME：

```
private static MediaCodecInfo selectCodec(String mimeType) {
    int numCodecs = MediaCodecList.getCodecCount();
    for (int i = 0; i < numCodecs; i++) {
        MediaCodecInfo codecInfo = MediaCodecList.getCodecInfoAt(i);
        if (!codecInfo.isEncoder()) {
            continue;
        }
        String[] types = codecInfo.getSupportedTypes();
        for (int j = 0; j < types.length; j++) {
            if (types[j].equalsIgnoreCase(mimeType)) {
                return codecInfo;
            }
        }
    }
    return null;
}
```

根据之前的讲述，在Android系统中有着不同的颜色格式，有着各种类型的YUV颜色格式和RGB颜色格式。在摄像头采集的文章中已经讲述，需要设置摄像头采集的图像颜色格式，一般来说设置为ImageFormat.NV21，之后在摄像头PreView的回调中得到相应的图像数据。

在Android系统中不同手机中的编码器支持着不同的颜色格式，一般情况下并不直接支持NV21的格式，这时候需要将NV21格式转换成为编码器支持的颜色格式。在摄像头采集的文章中已经详细讲述YUV图像格式和相应的存储规则，YUV图像格式的转换可以使用LibYuv (<https://github.com/illuspas/libyuv-android>)。

这里说一下MediaCodec支持的图像格式。一般来说Android MediaCodec支持如下几种格式：

(/apps/download?  
utm\_source=sbc)



```
/**
 * Returns true if this is a color format that this test code understands (i.e.
 * to read and generate frames in this format).
 */
private static boolean isRecognizedFormat(int colorFormat) {
    switch (colorFormat) {
        // these are the formats we know how to handle for this test
        case MediaCodecInfo.CodecCapabilities.COLOR_FormatYUV420Planar:
        case MediaCodecInfo.CodecCapabilities.COLOR_FormatYUV420PackedPlanar:
        case MediaCodecInfo.CodecCapabilities.COLOR_FormatYUV420SemiPlanar:
        case MediaCodecInfo.CodecCapabilities.COLOR_FormatYUV420PackedSemiPlanar:
        case MediaCodecInfo.CodecCapabilities.COLOR_TI_FormatYUV420PackedSemiPlanar:
            return true;
        default:
            return false;
    }
}
```

(/apps/download?  
utm\_source=sbc)

之前有看过一篇文章，里面大致统计了Android各种手机MediaCodec支持的各种颜色格式，上面5个类型是比较常用的类型。

另外MediaCodec支持Surface的方式输入和输出，当编码的时候只需要在Surface上进行绘制就可以输入到编码器，而解码的时候可以将解码图像直接输出到Surface上，使用起来相当方便，需要在Api 18或以上。

## 2、创建

当需要使用MediaCodec的时候，首先需要根据视音频的类型创建相应的MediaCodec。在直播项目中视频使用了H264，而音频使用了AAC-LC。在Android中创建直播的音频编码器需要传入相应的MIME，AAC-LC对应的是audio/mp4a-latm，而H264对应的是video/avc。如下的代码展示了两个编码器的创建，其中视频编码器的输入设置成为了Surface的方式。



```
//Audio
public static MediaCodec getAudioMediaCodec() throws IOException {
    int size = AudioRecord.getMinBufferSize(44100, AudioFormat.CHANNEL_IN_MONO,
        MediaFormat.createAudioFormat("audio/mp4a-latm", 44100,
        format.setInteger(MediaFormat.KEY_AAC_PROFILE, MediaCodecInfo.CodecProfileLe
        format.setInteger(MediaFormat.KEY_BIT_RATE, 64 * 1000);
        format.setInteger(MediaFormat.KEY_SAMPLE_RATE, 44100);
        format.setInteger(MediaFormat.KEY_MAX_INPUT_SIZE, size);
        format.setInteger(MediaFormat.KEY_CHANNEL_COUNT, 1);
        MediaCodec mediaCodec = MediaCodec.createEncoderByType("audio/mp4a-latm");
        mediaCodec.configure(format, null, null, MediaCodec.CONFIGURE_FLAG_ENCODE);
        return mediaCodec;
    }
//Video
public static MediaCodec getVideoMediaCodec() throws IOException {
    int videoWidth = getVideoSize(1280);
    int videoHeight = getVideoSize(720);
    MediaFormat format = MediaFormat.createVideoFormat("video/avc", videoWidth,
    format.setInteger(MediaFormat.KEY_COLOR_FORMAT, MediaCodecInfo.CodecCapabili
    format.setInteger(MediaFormat.KEY_BIT_RATE, 1300 * 1000);
    format.setInteger(MediaFormat.KEY_FRAME_RATE, 15);
    format.setInteger(MediaFormat.KEY_I_FRAME_INTERVAL, 1);
    format.setInteger(MediaFormat.KEY_BITRATE_MODE, MediaCodecInfo.EncoderCapabil
    format.setInteger(MediaFormat.KEY_COMPLEXITY, MediaCodecInfo.EncoderCapabilit
    MediaCodec mediaCodec = MediaCodec.createEncoderByType("video/avc");
    mediaCodec.configure(format, null, null, MediaCodec.CONFIGURE_FLAG_ENCODE);
    return mediaCodec;
}

// We avoid the device-specific limitations on width and height by using values
// are multiples of 16, which all tested devices seem to be able to handle.
public static int getVideoSize(int size) {
    int multiple = (int) Math.ceil(size/16.0);
    return multiple*16;
}
```

(/apps/download?  
utm\_source=sbc)



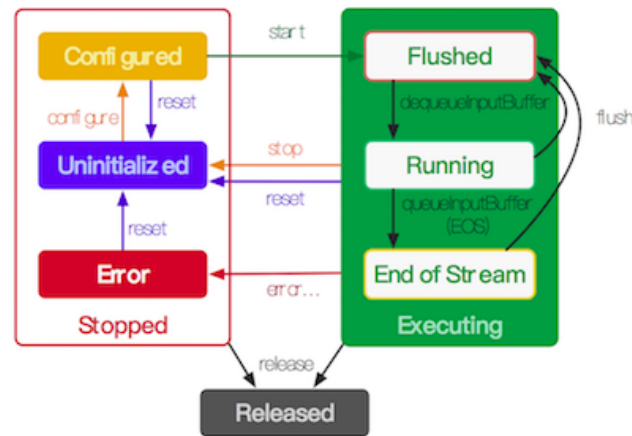
### 3、使用

MediaCodec创建之后，需要通过start()方法进行开启。MediaCodec有输入缓冲区队列和输出缓冲区队列，不断通过往输入缓冲区队列传递数据，经过MediaCodec处理后就可以得到响应的输出数据。当在编码的时候，需要向输入缓冲区传入采集到的原始的视音频数据，然后获取输出缓冲区的数据，输出出来的数据也就是编码处理后的数据。当在解码的时候，往输入缓冲区输入需要解码的数据，然后获取输出缓冲区的数据，输出出来的数据也就是解码后得到的原始的视音频数据。当需要清空输入和输出缓冲区的时候



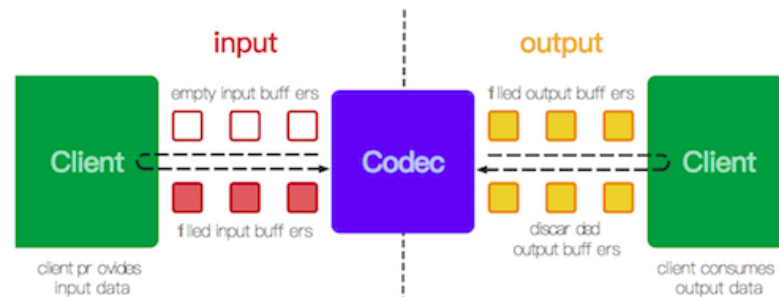


候，可以调用MediaCodec的flush()方法。当编码或者解码结束时，通过往输入缓冲区输入带结束标记的数据，然后从输出缓冲区可以得到这个结束标记，从而完成整个编解码过程。下面一张图片很好地展示了MediaCodec的状态变化。



MediaCodec状态

对于MediaCodec通过处理输入的数据，从而得到输出数据。MediaCodec通过一系列的输入和输出缓冲区来处理数据。如下图所示，输入客户端通过查询得到空的输入缓冲区，然后往里面填充数据，然后将输入缓冲区传递给MediaCodec；输出客户端通过查询得到塞满的输出缓冲区，然后得到里面的数据，然后通知MediaCodec释放这个输出缓冲区。



MediaCodec过程

在API 21及以后可以通过下面这种异步的方式来使用MediaCodec。

```
MediaCodec codec = MediaCodec.createByCodecName(name);
MediaFormat mOutputFormat; // member variable
codec.setCallback(new MediaCodec.Callback() {
    @Override
    void onInputBufferAvailable(MediaCodec mc, int inputBufferId) {
        ByteBuffer inputBuffer = codec.getInputBuffer(inputBufferId);
        // fill inputBuffer with valid data
        ...
        codec.queueInputBuffer(inputBufferId, ...);
    }

    @Override
    void onOutputBufferAvailable(MediaCodec mc, int outputBufferId, ...) {
        ByteBuffer outputBuffer = codec.getOutputBuffer(outputBufferId);
        MediaFormat bufferFormat = codec.getOutputFormat(outputBufferId); // option A
        // bufferFormat is equivalent to mOutputFormat
        // outputBuffer is ready to be processed or rendered.
        ...
        codec.releaseOutputBuffer(outputBufferId, ...);
    }

    @Override
    void onOutputFormatChanged(MediaCodec mc, MediaFormat format) {
        // Subsequent data will conform to new format.
        // Can ignore if using getOutputFormat(outputBufferId)
        mOutputFormat = format; // option B
    }

    @Override
    void onError(...) {
        ...
    }
});
codec.configure(format, ...);
mOutputFormat = codec.getOutputFormat(); // option B
codec.start();
// wait for processing to complete
codec.stop();
codec.release();
```

(/apps/download?  
utm\_source=sbc)



从API 21开始，可以使用下面这种同步的方式来使用MediaCodec。



```
MediaCodec codec = MediaCodec.createByCodecName(name);
codec.configure(format, ...);
MediaFormat outputFormat = codec.getOutputFormat(); // option B
codec.start();
for (;;) {
    int inputBufferId = codec.dequeueInputBuffer(timeoutUs);
    if (inputBufferId >= 0) {
        ByteBuffer inputBuffer = codec.getInputBuffer(...);
        // fill inputBuffer with valid data
        ...
        codec.queueInputBuffer(inputBufferId, ...);
    }
    int outputBufferId = codec.dequeueOutputBuffer(...);
    if (outputBufferId >= 0) {
        ByteBuffer outputBuffer = codec.getOutputBuffer(outputBufferId);
        MediaFormat bufferFormat = codec.getOutputFormat(outputBufferId); // option A
        // bufferFormat is identical to outputFormat
        // outputBuffer is ready to be processed or rendered.
        ...
        codec.releaseOutputBuffer(outputBufferId, ...);
    } else if (outputBufferId == MediaCodec.INFO_OUTPUT_FORMAT_CHANGED) {
        // Subsequent data will conform to new format.
        // Can ignore if using getOutputFormat(outputBufferId)
        outputFormat = codec.getOutputFormat(); // option B
    }
}
codec.stop();
codec.release();
```

(/apps/download?  
utm\_source=sbc)



在API版本21之前，获取缓冲区的方式有所不同，不能直接得到相应的缓冲区，需要根据索引序号从缓冲区列表中得到相应的缓冲区，具体的代码如下所示：



```
MediaCodec codec = MediaCodec.createByCodecName(name);
codec.configure(format, ...);
codec.start();
ByteBuffer[] inputBuffers = codec.getInputBuffers();
ByteBuffer[] outputBuffers = codec.getOutputBuffers();
for (;;) {
    int inputBufferId = codec.dequeueInputBuffer(...);
    if (inputBufferId >= 0) {
        // fill inputBuffers[inputBufferId] with valid data
        ...
        codec.queueInputBuffer(inputBufferId, ...);
    }
    int outputBufferId = codec.dequeueOutputBuffer(...);
    if (outputBufferId >= 0) {
        // outputBuffers[outputBufferId] is ready to be processed or rendered.
        ...
        codec.releaseOutputBuffer(outputBufferId, ...);
    } else if (outputBufferId == MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED) {
        outputBuffers = codec.getOutputBuffers();
    } else if (outputBufferId == MediaCodec.INFO_OUTPUT_FORMAT_CHANGED) {
        // Subsequent data will conform to new format.
        MediaFormat format = codec.getOutputFormat();
    }
}
codec.stop();
codec.release();
```

(/apps/download?  
utm\_source=sbc)



当数据输入结束的时候，通过queueInputBuffer的输入标记设置为BUFFER\_FLAG\_END\_OF\_STREAM来通知MediaCodec结束编码。当MediaCodec作为编码器的时候，dequeueOutputBuffer方法能够得到当前编码输出缓冲区数据的相关信息，这些信息存储在bufferInfo里面，通过bufferInfo信息能够得到数据的真实长度，当前数据为关键帧或者非关键帧等等信息。

当使用Output Surface作为解码的输出的时候，可以根据以下情况来设置是否将视频渲染到Surface上。

```
releaseOutputBuffer(bufferId, false) //不渲染buffer里面的数据
releaseOutputBuffer(bufferId, true) //渲染buffer里面的数据
releaseOutputBuffer(bufferId, timestamp) //在特定时间渲染buffer里面的数据
```



当使用Input Surface作为编码器输入的时候，不允许使用dequeueInputBuffer。当输入结束的时候，使用signalEndOfInputStream()来使得编码器停止。

## 六、MediaMuxer

前面讲述了MediaExtractor（视音频分离器），现在讲述MediaMuxer（视音频合成器）。MediaMuxer是Android提供的视音频合成器，目前只支持mp4和webm两种格式的视音频合成。一般来时视音频媒体都有视频轨道和音频轨道，有些时候也还有字母轨道，MediaMuxer将这些轨道糅合在一起存储在一个文件中。

MediaMuxer在Android中一个最常使用的场景是录制mp4文件。一般来说当存储为mp4文件时，视频轨道一般是经过编码处理后的h264视频，音频轨道一般是经过编码后处理的aac音频。前面已经讲述了如何对采集的视频和音频进行硬编，那么这时候如果对硬编后的视频和音频使用MediaMuxer进行合成，那么就可以合成为mp4文件。

下面是MediaMuxer一般的使用方法。

(/apps/download?  
utm\_source=sbc) ×

```
MediaMuxer muxer = new MediaMuxer("temp.mp4", OutputFormat.MUXER_OUTPUT_MPEG_4);
// More often, the MediaFormat will be retrieved from MediaCodec.getOutputFormat()
// or MediaExtractor.getTrackFormat().
MediaFormat audioFormat = new MediaFormat(...);
MediaFormat videoFormat = new MediaFormat(...);
int audioTrackIndex = muxer.addTrack(audioFormat);
int videoTrackIndex = muxer.addTrack(videoFormat);
ByteBuffer inputBuffer = ByteBuffer.allocate(bufferSize);
boolean finished = false;
BufferInfo bufferInfo = new BufferInfo();

muxer.start();
while(!finished) {
    // getInputBuffer() will fill the inputBuffer with one frame of encoded
    // sample from either MediaCodec or MediaExtractor, set isAudioSample to
    // true when the sample is audio data, set up all the fields of bufferInfo,
    // and return true if there are no more samples.
    finished = getInputBuffer(inputBuffer, isAudioSample, bufferInfo);
    if (!finished) {
        int currentTrackIndex = isAudioSample ? audioTrackIndex : videoTrackIndex;
        muxer.writeSampleData(currentTrackIndex, inputBuffer, bufferInfo);
    }
};
muxer.stop();
muxer.release();
```

(/apps/download?  
utm\_source=sbc)



其实上面的注释很好说明了MediaMuxer的使用场景。视音频轨道的初始化需要传入MediaFormat，而MediaFormat可以通过MediaCodec.getOutputFormat()获取（采集后进行硬编得到MediaFormat），也可以通过MediaExtractor.getTrackFormat()获取（分离器分离出视音频得到MediaFormat）。上面包含了两个应用场景，一个是采集，一个是转码。

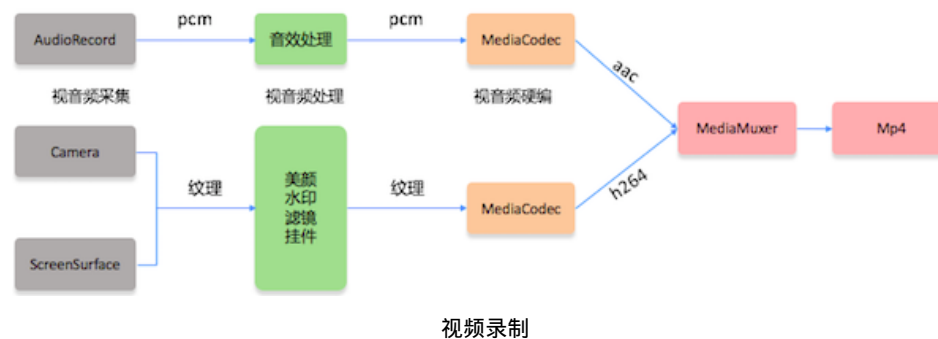
## 七、结合

MediaExtractor和MediaCodec结合使用可以实现视频的播放功能，MediaCodec和MediaMuxer结合使用可以实现视频的录制功能，MediaExtractor、MediaCodec和MediaMuxer三者一起使用可以实现视频的转码功能。下面讲述一下这几个功能的实现。

### 1、视音频录制



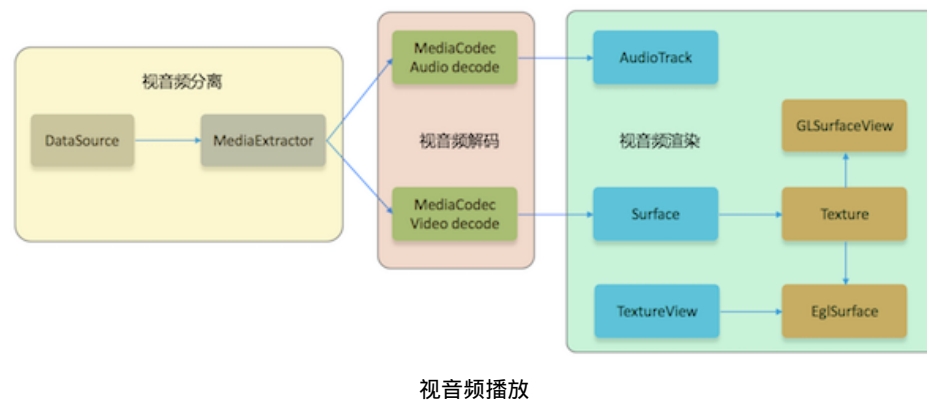
之前讲述了视频的采集和音频的采集，将采集到的视音频通过MediaCodec进行编码处理，之后将编码数据传递到MediaMuxer进行合成，也就完成了视音频录制的功能。



(/apps/download?  
utm\_source=sbc)

根据视音频采集的相关参数创建MediaCodec，当MediaCodec的outputBufferId为INFO\_OUTPUT\_FORMAT\_CHANGED时，可以通过codec.getOutputFormat()得到相应的MediaFormat，之后便可以用这个MediaFormat为MediaMuxer添加相应的视音频轨道。通过codec.dequeueOutputBuffer(...)可以得到编码后的数据的bufferInfo信息和相应的数据，之后将这个数据和bufferInfo通过muxer.writeSampleData(currentTrackIndex, inputBuffer, bufferInfo)传递给Muxer，也就将整个视音频数据合成到了mp4中。

## 2、视音频播放



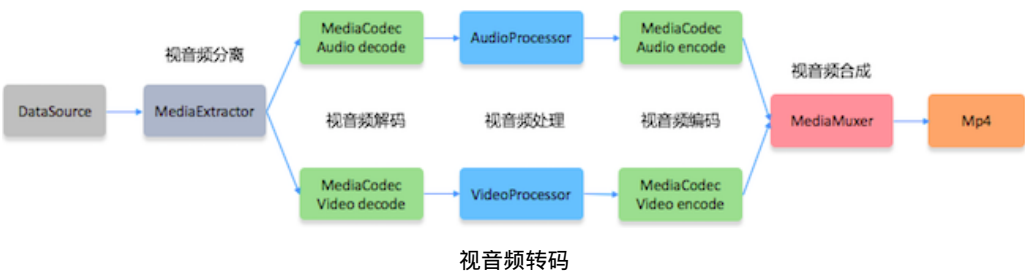
利用Android提供的Media API来实现一个播放器也是可以的，实际上Google著名的开源项目ExoPlayer就是这么做的。

上面的示意图简要描述了一个简单的本地播放器的结构。利用MediaExtractor分离视音频文件，得到相应的音频轨道和视频轨道。之后通过MediaExtractor从相应的轨道中获取数据，并且将这些数据传递给MediaCodec的输入缓冲区，经过MediaCodec的解码便可以得到相应的原始数据。音频解码后可以得到PCM数据，从而可以传递给AudioTrack进行播放。视频解码后可以渲染到相应的Surface，这个Surface可以通过SurfaceTexture创建，而SurfaceTexture是可以通过纹理创建的，从而将解码后的视频数据传递到纹理上了。

MediaExtractor解析视音频文件，可以得到相应数据的pts，之后pts可以传输到MediaCodec，之后在MediaCodec的输出里面可以得到相应的pts，之后在根据视音频的pts来控制视音频的渲染，从而实现视音频的同步。

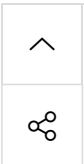
3、视音频转码

视音频的转码，其实就是通过MediaExtractor解析相应的文件，之后得到相应的视频轨道和音频轨道，之后将轨道里的数据传输到MediaCodec进行解码，然后将解码后的数据进行相应的处理（例如音频变声、视频裁剪、视频滤镜），之后将处理后的数据传递给MediaCodec进行编码，最后利用MediaMuxer将视频轨道和音频轨道进行合成，从而完成了整个转码过程。



八、展望

(/apps/download?utm\_source=sbc) ×





文中讲述了，如何使用Media API进行相应的录制、播放、转码，讲述了如何将视音频编解码和纹理相结合，但是由于篇幅原因并未讲述如何通过OpenGL对视频进行相应的处理，在之后的文章中会进一步讲述。

文中讲述了很多如何实现的原理和过程，但是并未提供相应的实现源码，其实这些程序的编写并不复杂，之后会写相应的开源代码，写完之后会更新到这篇文章上。

(/apps/download?utm\_source=sbc) ×

九、相关链接

- Android手机直播（一）总览 (http://www.jianshu.com/p/7ebbcc0c5df7)
- Android手机直播（二）摄像机 (http://www.jianshu.com/p/39a015f2996e)
- Android手机直播（三）声音采集 (http://www.jianshu.com/p/2cb75a71009f)
- Android手机直播（四）Android Media API (http://www.jianshu.com/p/d26e7d788c0e)

十、结束语

终于写完了，各位看官觉得文章不错的话不妨点个喜欢~

📖 安卓直播 (/nb/8464631) 举报文章 © 著作权归作者所有



风从影 (/u/c3dcf28e69f2)

写了 30717 字，被 660 人关注，获得了 696 个喜欢

(/u/c3dcf28e69f2)

+ 关注

如切如磋，如琢如磨。

小礼物走一走，来简书关注我

赞赏支持



(/u/7b03323aa12)6e6)

^

🔗

♥ 喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button)

56



(/apps/download?utm\_source=sbc)

(http://cwb.assets.jianshu.io/notes/images/10486619/weibo/image\_1

✕



(/apps/download?utm\_source=nbc)



登录 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-comment-form)

登录后发表评论

21条评论

只看作者

按喜欢排序 按时间正序 按时间倒序



徐有容陈长生 (/u/1ac3df5fdc5)

2楼 · 2017.07.14 11:51

(/u/1ac3df5fdc5)

支持你😊😊😊

👍 赞    💬 回复

风从影 (/u/c3dcf28e69f2) : 😊

2017.07.14 18:12    💬 回复

✍ 添加新评论



zhh\_happig (/u/d82bd37b1d29)

3楼 · 2017.07.14 11:55

(/u/d82bd37b1d29)

终于更新了



👍 赞    💬 回复

风从影 (/u/c3dcf28e69f2) : 😊

2017.07.14 18:11    💬 回复

✎ 添加新评论

(/apps/download?  
utm\_source=src) ✕



More\_Efficient (/u/45a75e771244)

4楼 · 2017.07.18 15:16

(/u/45a75e771244)

楼主真是厉害啊，膜拜大神 🙏

👍 赞    💬 回复



YeMengSky (/u/d7dad0efa430)

5楼 · 2017.07.30 16:31

(/u/d7dad0efa430)

MediaMetadataRetriever 和 MediaExtractor 都能获取到视频中的具体某一帧，这两者的区别是什么呢？比如我的场景是判断视频中有没有人脸的存在，这种场景是下，哪种比较合适呢？

👍 赞    💬 回复

倾城\_之泪 (/u/36b23cb5edb5) : MediaMetadataRetriever 获取到的是 Bitmap，MediaExtractor 则是 ByteBuffer 元数据。根据你的需求 MediaMetadataRetriever 更加符合一些

2017.08.11 15:19    💬 回复

✎ 添加新评论



cloudLy (/u/dcf1db4693a9)


7楼 · 2017.08.24 18:23


(/u/dcf1db4693a9)

支持楼主,想知道楼主的流程是用什么工具画的,很漂亮

👍 赞    💬 回复

风从影 (/u/c3dcf28e69f2) : @cloudLy (/users/dcf1db4693a9) ppt画的

2017.08.24 21:33  回复

 添加新评论



欧弟\_55ab (/u/78ecf5a699be)

8楼 · 2017.08.30 09:56

(/u/78ecf5a699be)

大牛快快更新博文，很是期待，受益匪浅，谢谢，谢谢，谢谢！！！！

 赞  回复



大大大大大的大大 (/u/eb01968a6673)


9楼 · 2017.08.30 16:49

(/u/eb01968a6673)


请问 关键帧间隔需要自己设置吗 如果需要要设置多大；现在用了七牛的直播sdk，关键帧间隔设置为帧率的3倍

 赞  回复


大大大大大的大大 (/u/eb01968a6673)：是这个吗  $\text{DEFAULT\_IFI} = 2$


2017.08.30 17:46  回复

大大大大大的大大 (/u/eb01968a6673)：@大大大大大的大大 (/users/eb01968a6673) 是这个吗  $\text{DEFAULT\_IFI} = 2$  2秒钟有一个关键帧 帧间隔=帧率\*2

2017.08.30 17:51  回复

风从影 (/u/c3dcf28e69f2)：@大大大大大的大大 (/users/eb01968a6673) 是的

2017.08.30 17:53  回复

 添加新评论



不会飞的扫把 (/u/ae658aa148f3)

10楼 · 2017.09.08 22:08

(/u/ae658aa148f3)

请问一下，怎么给视频添加水印呢

 赞  回复

(/apps/download?  
utm\_source=sbc)



风从影 (/u/c3dcf28e69f2)：@不会飞的扫把 (/users/ae658aa148f3) 把图片转换成纹理，然后用OpenGL绘制上去就好了

2017.09.09 07:53 回复

添加新评论

(/apps/download?  
utm\_source=sbc)



CAT1024 (/u/85394f69a998)

11楼 · 2017.09.14 10:40

(/u/85394f69a998)

开源的直播项目怎么八个月没更新的。。。大佬，快去维护啊

赞 回复



哎疯 (/u/899e2d74bdeb)

12楼 · 2017.09.22 11:03

(/u/899e2d74bdeb)

请问用MediaCodec编码后的h264播放出来视频速度非常快，请教大师

赞 回复



擒贼先擒王 (/u/7bd39fc3aad3)

13楼 · 2017.09.27 14:19

(/u/7bd39fc3aad3)

给我提供了新思路，谢谢大神！

强烈建议赶快出OpenGL篇！

赞 回复



取鸽帅气的昵称吧 (/u/edd764911d81)

14楼 · 2017.10.19 15:47

(/u/edd764911d81)

催更我只在小说干过，你是第一个android让我催更的，哈哈，快更啊，大佬~👍

赞 回复





小默森 (/u/72dd0825e2cd)

15楼 · 2017.10.31 14:25

(/u/72dd0825e2cd)

如果投屏随着时间的推移，延时越来越严重，大概是什么原因啊

赞 回复

[\(/apps/download?  
utm\\_source=sbc\)](/apps/download?utm_source=sbc)

被以下专题收入，发现更多相似内容



Android开发 (/c/d1591c322c89?utm\_source=desktop&amp;utm\_medium=notes-included-collection)



Android知识 (/c/3fde3b545a35?utm\_source=desktop&amp;utm\_medium=notes-included-collection)



程序员 (/c/NEt52a?utm\_source=desktop&amp;utm\_medium=notes-included-collection)



首页投稿 (/c/bDHhpK?utm\_source=desktop&amp;utm\_medium=notes-included-collection)



Android... (/c/4688bad2bca2?utm\_source=desktop&amp;utm\_medium=notes-included-collection)



Android... (/c/3b7b8627fed1?utm\_source=desktop&amp;utm\_medium=notes-included-collection)



其他学习 (/c/4ed1e1069e24?utm\_source=desktop&amp;utm\_medium=notes-included-collection)

展开更多

