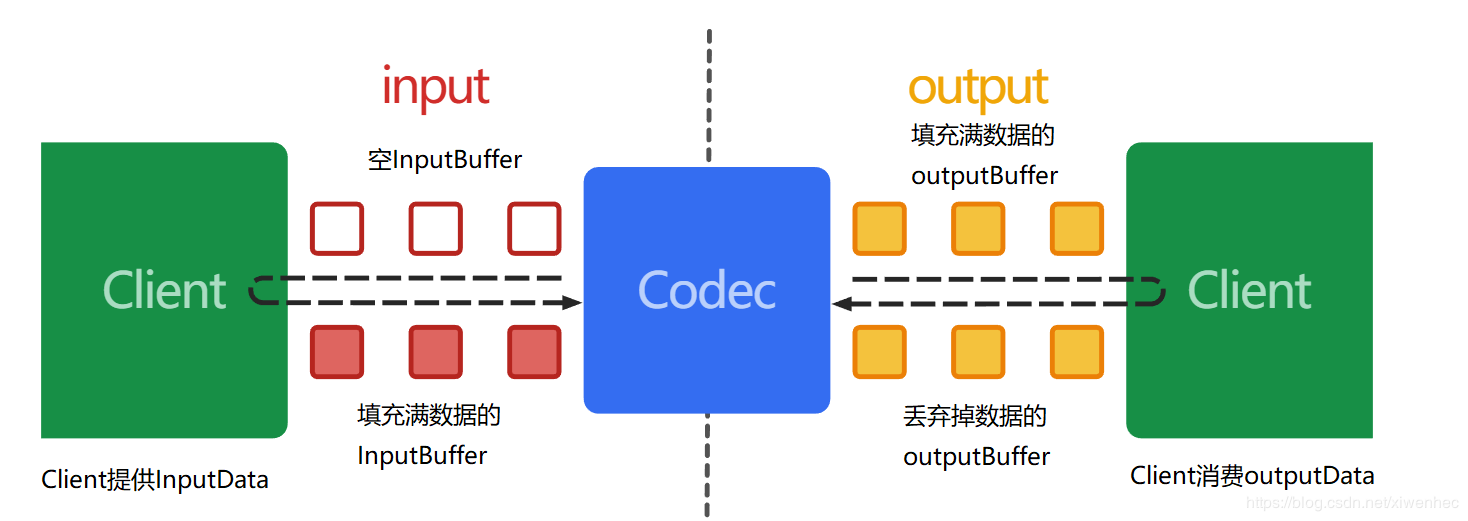
Android MediaCodec 使用说明

MediaCodec简述

[MediaCodec](https://developer.android.google.cn/reference/android/media/MediaCodec)类为开发者提供了能访问到Android底层媒体**Codec**（Encoder/Decoder）的能力，它是Android底层多媒体基础架构的一部分（通常和MediaExtractor、MediaSync、MediaMuxer、MediaCrypto、MediaDrm、Image、Surface、AudioTrack一起使用）。



从广义上来讲，Codec就是处理输入数据产生输出数据。它使用一组输入、输出缓冲区异步的处理数据，简而言之，你请求一个空的input Buffer，给它填充数据，然后将这个buffer送入Codec进行处理，Codec会消耗掉输入的数据，然后将InputBuffer转换成一空的Buffer以供下次请求时使用。最后，你请求一个填充满数据的outputBuffer，消费掉Buffer里的内容，然后释放掉这个Buffer返回给Codec。

**接收的数据**

MediaCodec 接受三种数据格式：压缩数据，原始音频数据和原始视频数据。

这三种类型的数据都可以通过[ByteBuffer](https://developer.android.google.cn/reference/kotlin/java/nio/ByteBuffer?hl=en)来传递给Codec，但是对于原始视频数据我们建议使用Surface来传递，这样可以提高Codec的性能，Surface使用的是native video buffer，不用映射或者拷贝成ByteBuffer，因此这样的方式更高效。当你使用Surface来传递原始视频数据时，也就无法获取到了原始视频数据，Android 提供了[ImageReader](https://developer.android.google.cn/reference/android/media/ImageReader)帮助你获取到解码后的原始视频数据。这种方式可能仍然有要比ByteBuffer的方式更加高效，因为某些native video buffer会直接映射成byteBuffer。当然如果你ByteBuffer的模式，你可以使用Image类提供的getInput/OutputImage(int)来获取原始视频数据。

**压缩数据**

压缩数据可以作为解码器的输入数据或者编码器的输出数据，需要指定数据格式，这样 codec 才能知道如何处理这些压缩数据。

对于视频数据而言，通常是一帧数据；音频数据，一般是单个处理单元。

**原始音频数据**

原始音频数据即编码器的输入数据，解码器的输出数据。包含整个 PCM 音频数据帧，是每一个通道按着通道顺序的采样数据。每一个采样按16Bit量化。。

**原始视频数据**

原始视频数据也是编码器的输入数据，解码器的输出数据。即yuv数据，MediaCodec主要支持的格式为：

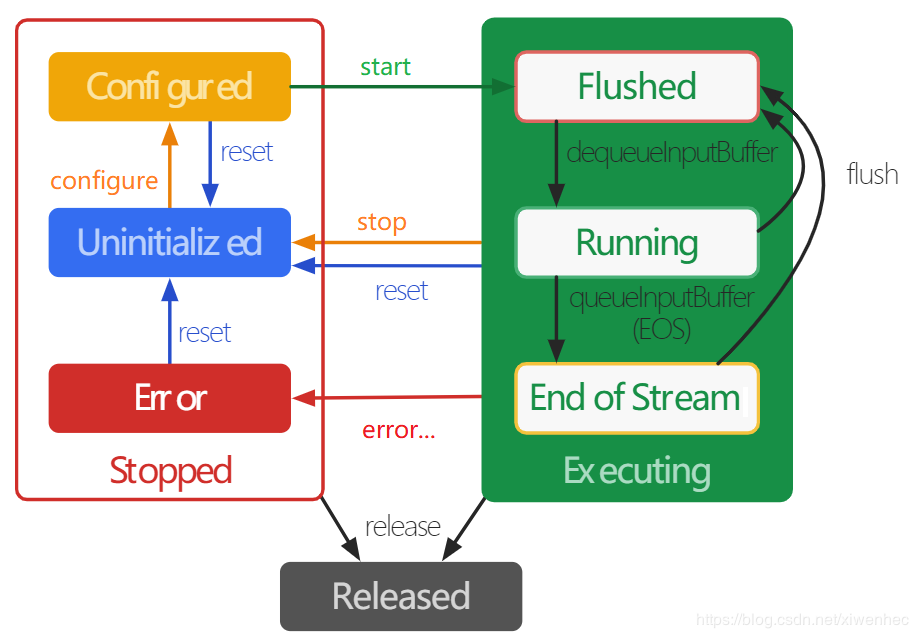
* native raw video format : COLOR\_FormatSurface，用来处理 Surface 模式的数据输入输出
* flexible YUV buffers : 例如 COLOR\_FormatYUV420Flexible
* specific formats: 支持ByteBuffer模式，有一些厂家会定制

**Codec的状态**

Codec的声明周期存在三种状态：**Stopped，Executing，Released**。

Stopped状态是一个集合状态，它聚合了三种状态：Uninitialized，Configured，和Error；

同时Executing状态的处理也是通过三个子状态来完成：Flushed，Running，End-of-Stream。

  
创建MediaCodec后，Codec就处于Uninitialized状态。

调用MediaCode.Configure(...)后会将Codec转换成Configured状态。

调用MediaCode.start()函数，将Codec转换成Executing状态的Flushed子状态，在这个状态下我们才能通过buffer处理数据。

调用MediaCode. dequeueInputBuffer()只要第一个Input buffer被dequeued，Codec就转变成Running子状态，这个状态占据了Codec的生命周期的绝大部分。

当入队一个带有end-of-stream标志的InputBuffer后，Codec将转换成End of Stream子状态，在这个状态下，Codec将不会再接收任何输入的数据，但是仍然会产生output buffer ,直到end-of-Stream标记的buffer被输出。在Executing状态的任何时候,调用flush()函数，将Codec切换成Flushed状态。

调用stop()函数会将Codec返回到Uninitialized状态，这样我们就可以对Codec进行重新配置，当用完了Codec后，必须要调用release()函数去释放这个Codec

在极少数情况下，Codec可能也会遇到错误，此时Codec将会切换到Error状态，我们可以通过queuing操作获取到一个无效的返回值，或者有时会通过异常来的得知Codec发生了错误。通过调用reset()函数，将Codec进行重置，这样Codec将切换成Uninitalized状态，我们可以在任何状态下调用rest()函数将Codec将切换成Uninitalized`状态。

MediaCodec使用