## 一：背景介绍

ijkplayer是基于ffmpeg实现的，前期版本使用ffmpeg的Decoder。在n0.2.1版本开始支持MediaCodec(API 16+, android4.1开始支持MediaCodec)。

ijkplayer在JAVA层提供了MediaCodec的配置开关：

ijkMediaPlayer的setOption接口，category是OPT\_CATEGORY\_PLAYER, 可设置的key-value有：

(参考Ff\_ffplay\_options.h里的ffp\_context\_options)

key: mediacodec

value: 0 or 1

help: MediaCodec: enable H264 (deprecated by 'mediacodec-avc')

key: mediacodec-all-videos

value: 0 or 1

help: MediaCodec: enable all videos

key: mediacodec-avc

value: 0 or 1

help: MediaCodec: enable H264

key: mediacodec-hevc

value: 0 or 1

help: MediaCodec: enable HEVC

key: mediacodec-mpeg2

value: 0 or 1

help: MediaCodec: enable MPEG2VIDEO

key: mediacodec-auto-rotate

value: 0 or 1

help: MediaCodec: auto rotate frame depending on meta

## 二：MediaCodec配置开关作用流程

这里，用到了ffmpeg的AVOption，结构成员管理系统。查看Ff\_ffplay\_options.h里的ffp\_context\_options数据结构：

|  |
| --- |
| { "mediacodec", "MediaCodec: enable H264 (deprecated by 'mediacodec-avc')",  OPTION\_OFFSET(mediacodec\_avc), OPTION\_INT(0, 0, 1) },  { "mediacodec-auto-rotate", "MediaCodec: auto rotate frame depending on meta",  OPTION\_OFFSET(mediacodec\_auto\_rotate), OPTION\_INT(0, 0, 1) },  { "mediacodec-all-videos", "MediaCodec: enable all videos",  OPTION\_OFFSET(mediacodec\_all\_videos), OPTION\_INT(0, 0, 1) },  { "mediacodec-avc", "MediaCodec: enable H264",  OPTION\_OFFSET(mediacodec\_avc), OPTION\_INT(0, 0, 1) },  { "mediacodec-hevc", "MediaCodec: enable HEVC",  OPTION\_OFFSET(mediacodec\_hevc), OPTION\_INT(0, 0, 1) },  { "mediacodec-mpeg2", "MediaCodec: enable MPEG2VIDEO",  OPTION\_OFFSET(mediacodec\_mpeg2), OPTION\_INT(0, 0, 1) }, |

可以看到，通过JAVA层设置的key参数都会对应FFPlay数据结构里面的其中一个变量。

JAVA层的setOption接口对应于Natvie层的ffp\_set\_option\_int，最后调用ffmpeg的av\_dict\_set\_int来更改对应Option的值，也就改变了FFPlay里面的变量的值。

## 三：什么时候使用MediaCodec

这里需要提到一个IJKFF\_Pipeline，引入MediaCodec后，它使用的是ffpipeline\_create\_from\_android来创建的。

在IJKFF\_Pipeline里的func\_open\_video\_decoder指向的函数实现：

|  |
| --- |
| if (ffp->mediacodec\_all\_videos || ffp->mediacodec\_avc || ffp->mediacodec\_hevc || ffp->mediacodec\_mpeg2)  node = ffpipenode\_create\_video\_decoder\_from\_android\_mediacodec(ffp, pipeline, opaque->weak\_vout);  if (!node) {  node = ffpipenode\_create\_video\_decoder\_from\_ffplay(ffp);  } |

可以看到，如果设置了MediaCodec的配置，Video Codec会优先通过MediaCodec来获取，如果获取不到合适的Codec，则再从ffmpeg里获取Decoder。

## 四：MediaCodec支持的Video类型

在ffpipenode\_create\_video\_decoder\_from\_android\_mediacodec里可以看到：

|  |
| --- |
| #define SDL\_AMIME\_VIDEO\_AVC "video/avc" //- H.264/AVC video  #define SDL\_AMIME\_VIDEO\_HEVC "video/hevc" //- H.265/HEVC video  #define SDL\_AMIME\_VIDEO\_MPEG2VIDEO "video/mpeg2" //- MPEG2 video |

目前支持这三类的Video的硬解。其他类型的Video都是用ffmpeg的软解，如"video/mp4v-es", "video/3gpp"。

对于AVC视频来说，ffmpeg的libavcodec在Demuxer提取Video流的时候，会设置不同的profile和level，有些profile在MediaCodec里也是没有支持的，如下图，红色的profile在MediaCodec是不支持的，会使用ffmpeg的Codec。

|  |
| --- |
| #define FF\_PROFILE\_H264\_BASELINE 66  #define FF\_PROFILE\_H264\_CONSTRAINED\_BASELINE (66|FF\_PROFILE\_H264\_CONSTRAINED)  #define FF\_PROFILE\_H264\_MAIN 77  #define FF\_PROFILE\_H264\_EXTENDED 88  #define FF\_PROFILE\_H264\_HIGH 100  #define FF\_PROFILE\_H264\_HIGH\_10 110  #define FF\_PROFILE\_H264\_HIGH\_10\_INTRA (110|FF\_PROFILE\_H264\_INTRA)  #define FF\_PROFILE\_H264\_HIGH\_422 122  #define FF\_PROFILE\_H264\_HIGH\_422\_INTRA (122|FF\_PROFILE\_H264\_INTRA)  #define FF\_PROFILE\_H264\_HIGH\_444 144  #define FF\_PROFILE\_H264\_HIGH\_444\_PREDICTIVE 244  #define FF\_PROFILE\_H264\_HIGH\_444\_INTRA (244|FF\_PROFILE\_H264\_INTRA)  #define FF\_PROFILE\_H264\_CAVLC\_444 44 |

## 五：MediaCodec的Codec匹配规则

根据设置好的ijkmp\_mediacodecinfo\_context（包括mime\_type， profile， level， codec\_name），调用ffpipeline\_select\_mediacodec\_l接口查找最合适的Codec。

ffpipeline\_select\_mediacodec\_l调用回调函数mediacodec\_select\_callback，再通过JAVA发射机制，调用JAVA层的接口onSelectCodec，最终调用到onMediaCodecSelect。

onMediaCodecSelect接口里面，首先通过Android标准SDK接口MediaCodecList，获取系统支持的Codec列表，MediaCodecList实际上是读取/system/etc/media\_codecs.xml文件（这个文件每个平台都会定制）。

根据Native传递上来的mime\_type信息，遍历MediaCodecList，找出所有符合要求的Codec,接着，**ijkplayer这里做了一个各个平台的适配操作：IjkMediaCodecInfo.setupCandidate。**

**做这个适配操作就是为了针对各个平台的不同硬解能力的一个优化改进（这个后面是可以根据需要进行定制优化）。**

IjkMediaCodecInfo.setupCandidate 会给各个Codec定等级，级别如下：

|  |
| --- |
| public static final int RANK\_MAX = 1000;  public static final int RANK\_TESTED = 800;  public static final int RANK\_ACCEPTABLE = 700;  public static final int RANK\_LAST\_CHANCE = 600;  public static final int RANK\_SECURE = 300;  public static final int RANK\_SOFTWARE = 200;  public static final int RANK\_NON\_STANDARD = 100;  public static final int RANK\_NO\_SENSE = 0; |

值越大，等级越高，优先选择。

具体类型的Codec的等级定义如下：

|  |
| --- |
| 非 "omx." RANK\_NO\_SENSE  omx.pv RANK\_SOFTWARE  omx.google. RANK\_SOFTWARE  omx.ffmpeg. RANK\_SOFTWARE  omx.k3.ffmpeg. RANK\_SOFTWARE  omx.avcodec. RANK\_SOFTWARE  omx.ittiam. RANK\_NO\_SENSE  omx.mtk. RANK\_TESTED  //如果不是上面这些Codec类型，就根据已经收集的具体Codec进行等级设置：  OMX.Nvidia.h264.decode RANK\_TESTED  OMX.Nvidia.h264.decode.secure RANK\_SECURE  OMX.Intel.hw\_vd.h264 RANK\_TESTED + 1  OMX.Intel.VideoDecoder.AVC RANK\_TESTED  OMX.qcom.video.decoder.avc RANK\_TESTED  OMX.ittiam.video.decoder.avc RANK\_NO\_SENSE  OMX.SEC.avc.dec RANK\_TESTED  OMX.SEC.AVC.Decoder RANK\_TESTED - 1  OMX.SEC.avcdec RANK\_TESTED - 2  OMX.SEC.avc.sw.dec RANK\_SOFTWARE  OMX.Exynos.avc.dec RANK\_TESTED  OMX.Exynos.AVC.Decoder RANK\_TESTED - 1  OMX.k3.video.decoder.avc RANK\_TESTED  OMX.IMG.MSVDX.Decoder.AVC RANK\_TESTED  OMX.TI.DUCATI1.VIDEO.DECODER RANK\_TESTED  OMX.rk.video\_decoder.avc RANK\_TESTED  OMX.amlogic.avc.decoder.awesome RANK\_TESTED  OMX.MARVELL.VIDEO.HW.CODA7542DECODER RANK\_TESTED  OMX.MARVELL.VIDEO.H264DECODER RANK\_SOFTWARE  OMX.sprd.soft.h264.decoder RANK\_SOFTWARE |

如果不属于以上这些Codec类型，会调用Android的标准SDK接口MediaCodecInfo的getCapabilitiesForType，获取CodecCapabilities，获取到，则是RANK\_ACCEPTABLE，否则是RANK\_LAST\_CHANCE。

选择到的最佳的Codec信息会保存到FFPlayer里的audio\_codec\_info, 在JAVA层可以通过\_getVideoCodecInfo接口来获取。

## 六. MediaCodec适配后的Codec如何被用来解码

JAVA层的prepare接口调用后，执行到Native里的ffp\_prepare\_async\_l，根据url信息，调用stream\_open，这个时候，会创建一个线程

|  |
| --- |
| is->read\_tid = SDL\_CreateThreadEx(&is->\_read\_tid, read\_thread, ffp, "ff\_read"); |

在read\_thread里，先利用ffmpeg的av\_find\_input\_format查找合适的Demuxer, 通过avformat\_open\_input根据选择的Demuxer来提取资源的开始部分数据，解析，获取到Audio/Video流，同时设置Codec ID, Codec Type等相关信息, 然后执行

|  |
| --- |
| ret = stream\_component\_open(ffp, st\_index[AVMEDIA\_TYPE\_VIDEO]); |

在stream\_component\_open里，利用ffmpeg的avcodec\_find\_decoder根据前面的Codec ID找到Codec(AVCodec), 找到表示这个Codec ID正常，否则异常，退出。

接着，根据codec\_type，如果是AVMEDIA\_TYPE\_VIDEO的话（Audio直接使用前面获取到的ffmpeg的Decoder），调用ffpipeline\_open\_video\_decoder，也就是调用了func\_open\_video\_decoder（第三点提到的函数），得到Codec信息保存到FFPlayer的node\_vdec里。接着，执行解码线程：

|  |
| --- |
| decoder\_start(&is->viddec, video\_thread, ffp, "ff\_video\_dec") |

在video\_thread里，执行ffpipenode\_run\_sync，调用node\_vdec的func\_run\_sync，里面执行解码过程。