Rockchip Android MediaCodec 扩展参数支持说明

文件标识：RK-SM-YF-E12

发布版本：V1.0.2

日期：2025-05-26

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文档主要介绍 Rockchip Android 平台所有支持的厂商 MediaCodec 扩展参数，旨在帮助开发者了解平台编解码器支持的的扩展特性，在自己的 MediaCodec 应用中按需使用扩展参数。

> 本文档扩展参数介绍基于 MediaCodec Codec2，仅在 Android 12 及以上的版本适配支持。

芯片名称	内核版本
适配所有芯片	Linux-4.19、Linux-5.10、Linux-6.1

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	陈锦森	2024-09-02	初始版本
V1.0.1	陈锦森	2025-02-07	新增一些扩展参数配置
V1.0.2	陈锦森	2025-05-26	新增一些扩展参数配置

目录

---

1. 解码器平台扩展参数支持

1.1 低延时解码模式

平台低延迟解码模式用于快速输出解码图像，提高解码的实时性。

该模式使能解码器内部快速帧解析功能，并忽略帧的参考序列，立即输出解码图像。常用于对解码显示延迟比较敏感的场景，如投屏、直播等。

> 低延时解码模式是 "一进一出" 式，不适用于带 B 帧的码流（B 帧前后参考，立即输出会导致解码输出帧乱序）。

使能方式：

```
mediaFormat.setInteger("low-latency", 1);
```

确认生效日志：

```
c2_info("enable lowLatency, enable mpp fast-out mode");
```

1.2 tunneled 多媒体隧道模式

多媒体隧道模式下，解码器与显示驱动直接建立隧道，解码输出无需经 Adnroid SurfaceFlinger 显示框架流程处理，直接发送到显示屏。

应用场景：

1. **高码流高帧率视频播放** - 绕过系统的显示框架流程，减少系统流程执行产生的软件开销，释放系统资源，为高码流高帧率视频播放提供支持
2. **低延迟显示方案** - 解码输出绕开 SurfaceFlinger 显示框架直接同步到显示 VOP，显著降低显示延迟。

> 参考谷歌官网: 多媒体隧道模式

1) 补丁配置

环境需求 - 内核版本 >= 5.10

1. 在自己的板级 dts 使能 rkvtunnel 驱动

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3588s-evb1-lp4x.dtsi
b/arch/arm64/boot/dts/rockchip/rk3588s-evb1-lp4x.dtsi
index 8c1d0f37b8a3..8bd65c9ed85b 100644
--- a/arch/arm64/boot/dts/rockchip/rk3588s-evb1-lp4x.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3588s-evb1-lp4x.dtsi
@@ -846,6 +846,10 @@ &usbhost_dwc3_0 {
        status = "disabled";
 };

+&rkvtunnel {
+        status = "okay";
+};
+
 /* vp0 & vp3 are not used on this board */
 &vp0 {
        /delete-property/ rockchip,plane-mask;
```

2. 在解码器配置文件 media_codecs.xml 中公告该解码器能够进行隧道式播放

如下配置文件配置 RK3588 AVC 解码器支持隧道模式，其他芯片其他解码器类似配置支持。

```
~/3_android_14/vendor/rockchip/common/vpu/etc$ git diff .
diff --git a/vpu/etc/media_codecs_c2_rk3588.xml
b/vpu/etc/media_codecs_c2_rk3588.xml
index 28c1f947..d01da2d5 100644
--- a/vpu/etc/media_codecs_c2_rk3588.xml
+++ b/vpu/etc/media_codecs_c2_rk3588.xml
@@ -20,6 +20,7 @@
     </Settings>
     <Decoders>
         <MediaCodec name="c2.rk.avc.decoder" type="video/avc">
+            <Feature name="tunneled-playback" required="true"/>
             <Alias name="OMX.rk.video_decoder.avc" />
             <Limit name="size" min="64x64" max="7680x4320" />
             <Limit name="alignment" value="2x2" />
```

2) 应用说明

多媒体隧道模式音视频同步方式支持 AUDIO_HW_SYNC/HW_AV_SYNC/REALTIME，AUDIO_HW_SYNC 和 HW_AV_SYNC 方式依赖系统调谐器驱动。

目前仅支持实时渲染方式，即延迟优先，取到解码输出图像立刻通过隧道送显。程序伪代码如下：

```
// 1. get tunneled codec name
format.setFeatureEnabled(MediaCodecInfo.CodecCapabilities.FEATURE_TunneledPlayba
ck, true);
MediaCodecList mcl = new MediaCodecList(MediaCodecList.ALL_CODECS);
String codecName = mcl.findDecoderForFormat(format);

// 2. start tunneled video playback
MediaCodec mCodec = MediaCodec.createByCodecName(codecName);
mCodec.configure(format, mSurface, null, 0);
mCodec.start();

while (!isEOS) {
    int inIndex = mCodec.dequeueInputBuffer(0);
    if (inIndex >= 0) {
        ByteBuffer buffer = mCodec.getInputBuffer(inIndex);
        int sampleSize = mExtractor.readSampleData(buffer, 0);
        if (sampleSize > 0) {
            mCodec.queueInputBuffer(inIndex, 0, sampleSize,
                    mExtractor.getSampleTime(), 0);
            mExtractor.advance();
        } else {
            mCodec.queueInputBuffer(inIndex, 0, 0,
                    0, MediaCodec.BUFFER_FLAG_END_OF_STREAM);
            isEOS = true;
        }
    }
}

mCodec.stop();
mCodec.release();
```

1.3 忽略 H264 码流的帧序列连续性检查（忽略丢帧标错）

【问题描述】

网络会议/监控等实时流传输场景，由于网络不稳定导致码流丢帧丢包。

解码器内部默认会检查码流帧序列的连续性，检测到码流 POC 不连续，将整个 GOP 序列帧标错，进而丢掉整个 GOP 序列的 error 帧，下一个关键帧进来才重新开始显示播放。

解码过程中异常日志如:

```
C2RKMpiDec: skip error frame with pts 0
```

【问题解决】

解码器硬件有纠错功能，码流轻微的丢帧丢包有时依赖硬件的纠错功能可以恢复过来，这种情况可以忽略 POC 连续性检查，码流也可以正常播放。

- 注意硬件纠错非无止境，过于严重的丢帧依旧会导致花屏。
- POC 连续性检测是码流解码的标准流程，因此仅作为用户配置支持该功能。

实际网络流传输场景，检测到轻微丢帧导致画面卡顿，可以使用下面的配置禁用码流 POC 连续性检查。

使能方式:

```
mediaFormat.setInteger("vendor.c2-dec-disable-dpb-check.value", 1);
```

确认生效日志:

```
c2_info("disable poc discontinuous check");
```

1.4 关闭解码器内部的错误处理

针对一些异常码流解码的兼容，如参考关系异常、码流语法异常等。

使用以下用户扩展参数配置关闭解码器内部的错误处理，一旦使能，解码器会无视码流的错误情况，输出全部的可解码图像，同时不对解码输出做 errinfo 标记。

> Note: 由于框架依赖解码器输出的 errinfo 做丢帧处理，无 errinfo 输出可能造成异常帧被输出显示，从而造成花屏显示。

使能方式:

```
mediaFormat.setInteger("vendor.c2-dec-disable-error-mark.value", 1);
```

确认生效日志:

```
c2_info("disable error frame mark");
```

1.5 解码器内存使用优化配置

在低内存设备或其他一些设备可用内存比较紧张的场景，提供用户配置用于绕开框架的限制，减少解码器申请的输出 buffer 个数，达到减少内存使用量的目的。

在大分辨率片源的解码播放场景，内存改善效果尤为显著。配置方式:

```
mediaFormat.setInteger("vendor.c2-dec-low-memory-mode.value", xxx);

// 目前支持两种配置选项:
// 配置选项 0x1: 绕开框架限制，减少 4 个 buffer 个数申请
// 配置选项 0x2: 主动检测并使用码流 sps 中的参考帧个数作为输出 buffer 申请个数

// 设置值 0x3，配置打开所有优化项
```

确认生效日志:

```
c2_info("in low memory mode %d, reduce output ref count", lowMemoryMode);
```

> 解码器默认根据码流 level 信息预测输出 buffer 个数，这种方式申请内存较多，可以满足所有片源。配置选项 0x2 申请内存通常更小，但可能存在片源 sps 参考帧数目不可信的情况，导致片源播放失败，因此配置这种方式需先经过片源兼容性测试。

1.6 关闭解码器 FBC 输出模式

FrameBuffer Compression（FBC）是基于硬件的帧缓存区压缩技术，设置 FBC 解码输出可以提高解码效率\降低 DDR 带宽负载，在 RK3576\RK3588\RK356X 等芯片大于 1080P 的片源默认会开启 FBC 输出。

Codec2 框架提供用户接口配置支持，用来关闭 FBC 解码输出。

使能方式:

```
mediaFormat.setInteger("vendor.c2-dec-fbc-disable.value", 1);
```

确认生效日志:

```
c2_info("got disable fbc request");
```

1.7 SurfaceTexture 解码器配置输出实宽实高

硬件解码器处理对齐过的 buffer 效率比较高，因此框架申请的解码输出 buffer 都是经过对齐的，即解码输出存在 stride 无效数据。如 H264 1920x1080 格式申请 buffer 规格为1920x1088，送显前需要先经过裁剪掉底部 8 个像素的虚高。

MediaCodec 应用配置 SurfaceView 的情况，走 Android SurfaceFlinger 渲染框架，Crop 裁剪参数由框架内部处理。而指定配置 SurfaceTexture 时，走 GL 纹理渲染，需要应用配合完成 Crop 操作，否则视频应用测试可能出现绿边。

兼容早期的 SurfaceTexture 视频应用，提供用户配置直接输出不带 stride 的输出来避免绿边问题。

> 相比较原先的流程，该方式在框架中手动拷贝了一个不带 stride 的解码输出，因此框架解码耗时可能略微增加。

使能方式:

```
mediaFormat.setInteger("vendor.c2-dec-output-crop-enable", 1);
```

确认生效日志:

```
c2_info("got request for output crop");
```

2. 编码器平台扩展参数支持

2.1 可分级 TSVC 编码模式支持

SVC（Scalable Video Coding，可分级视频编码）的工作原理是对视频进行分层处理。它允许视频流被分割成多个层级，每个层级可以独立解码，每一层都增加了特定的细节，如更高的分辨率或更流畅的动作，并且可以组合以提供不同级别的视频质量。

- 网络自适应性：网络连接不稳定时只解码第一层视频基本内容，网络流畅下则解码更高层增强信息，提供更清晰的分辨率等。
- 差错恢复： 底层码流可以利用高层的信息进行错误检测和纠正，如果在 SVC 流中检测到了错误，分辨率和帧率可以逐步降层直至基本层，SVC 的抗丢包率高达 20%。

> SVC 技术在视频监控和视频会议等领域天然优势，因此已被国内外厂商广泛使用，如腾讯 Rooms 会议、微软 Teams 会议等均使用 SVC 编码。

Rockchip 平台提供时域可分级编码 TSVC 的实现，支持配置 2 ~ 4 层编码:

```
mediaFormat->setString("ts-schema", "android.generic.3");
```

可通过下面的打印或查看生成码流确认生效:

```
c2_info("setupTemporalLayers: layers %d", layerCount);
```

2.2 QP 范围设置

QP 是码流编码质量的量化参数，可配置范围为 1 ~ 51。一般来说，QP 越小，视频质量越高，但码率也会相应的增加，反之，QP 值越大，视频质量越低，码率也会相对应的减小。因此，合理设置 QP 范围对于平衡视频质量和文件大小至关重要。

参考以下代码配置编码器 QP 范围，配置 I 帧 QP 范围为 10 ~ 40，P 帧 QP 范围为 10 ~ 30。

```
mediaFormat->setInt32("video-qp-i-min", 10);
mediaFormat->setInt32("video-qp-i-max", 40);
mediaFormat->setInt32("video-qp-p-min", 10);
mediaFormat->setInt32("video-qp-p-max", 30);
```

可通过下面的打印或查看生成码流确认生效：

```
c2_info("setupQp: qpInit %d i %d-%d p %d-%d", qpInit, iMin, iMax, pMin, pMax);
```

2.3 旋转编码支持

使能旋转角度配置支持，支持 90\180\270 度视频旋转角度，参考如下代码，配置编码器输出码流按 90 度顺时针旋转：

```
mediaFormat->setInt32("rotation", 90);
```

可通过下面的打印确认生效：

```
c2_info("setupPreProcess: rotation degrees %d", degrees);
```

2.4 缩放编码支持

平台硬件本身不支持缩放编码，该功能依赖平台图形处理硬件 RGA 对编码输入做缩放前处理，因此编码耗时可能会略微增加。

参考如下代码，配置编码器码流按 720P 分辨率输出：

```
mediaFormat->setInt32("vendor.c2-enc-input-scalar.width", 1280);
mediaFormat->setInt32("vendor.c2-enc-input-scalar.height", 720);
```

可通过下面的打印确认生效：

```
c2_info("setupBaseCodec: coding %s w %d h %d hor %d ver %d",
        toStr_Coding(mCodingType), mSize->width, mSize->height, mHorStride,
mVerStride);
```

2.5 多 slice 配置支持

H264\H265 编码以宏块为最小单位（简称 MB），多个连续的宏块组成一个 slice，每个 slice 编码输出生成一个 NALU，默认情况下一帧图像编码放在一个 slice 里。

- 因为划分 slice 的概念，一个 slice 包含某一帧的全部或部分数据。
- 划分 slice 可以按照固定宏块个数的方式，也可以按照字节大小也就是宏块累计的字节数。
- 划分 slice 之后，每个 slice 都有自己专有的包头等元信息，因此最终的编码体积会相对应的变大，划分越细，整体码率越高。

适用于多 slice 的业务场景主要有几类：

1. MTU 大小受限的网络流传输场景，如一次最大发送数据包是 1500 字节，则可以划分 slice，一个 slice 按 1500 大小划分。
2. 多 slice 的编码是独立的，对有多核编码器的芯片如 RK3588，可以并发地对多个 slice 进行编码，提升编码性能。
3. 抗网损功能，划分 slice 之后，实施纠错机制，对应所需要处理的区域就可以较小，减少消耗。

目前默认支持按字节大小划分 slice，可参考如下配置：

```
mediaFormat->setInt32("vendor.c2-enc-slice-size.value", 1500);
```

可通过下面的打印确认生效：

```
c2_info("setupSliceSize: slice-size %d", c2Size->value);
```

2.6 超大帧重编码支持

超大帧对网络冲击较大，该配置适用于需要平滑码率刷新的场景。

需注意，超大帧重编次数越多单帧编码时间可能越长，请根据实际需求配置。配置编码器超大帧至多重编三次，可参考如下配置：

```
mediaFormat->setInt32("vendor.c2-enc-reenc-times.value", 1500);
```

可通过下面的打印确认生效：

```
c2_info("setupReencTimes: reenc-times %d", reencTime->value);
```

2.7 RK3576\RK3588 智能编码模式支持(智能降码率)

超级编码模式依据客户省带宽\画质增强等需求开发。目前平台主要开发三个版本：

- 1.0 版本: 依据帧间\帧内复杂度信息、ROI 区域占比、最大最小码率，自适应调整量化参数，优化编码策略达到省码率的目的，对大运动\小运动\静止场景码率优化分别达到 21%+\85%+\90%+。
- 2.0 版本: 更加丰富的统计信息，更平稳的码控策略。进一步优化静态弱纹理场景的码率和清晰度。
- 3.0 版本: 加入 AI 神经网络智能识别 ROI 区域，达到画质增强的目的。

> 需要注意的是，目前仅 RK3576\RK3588 支持智能编码模式。

目前 MediaCodec 扩展参数集支持配置智能编码 1.0 和 3.0 版本。

平台提供了多种策略用于码控调优，以下代码配置运动场景下画质优先模式:

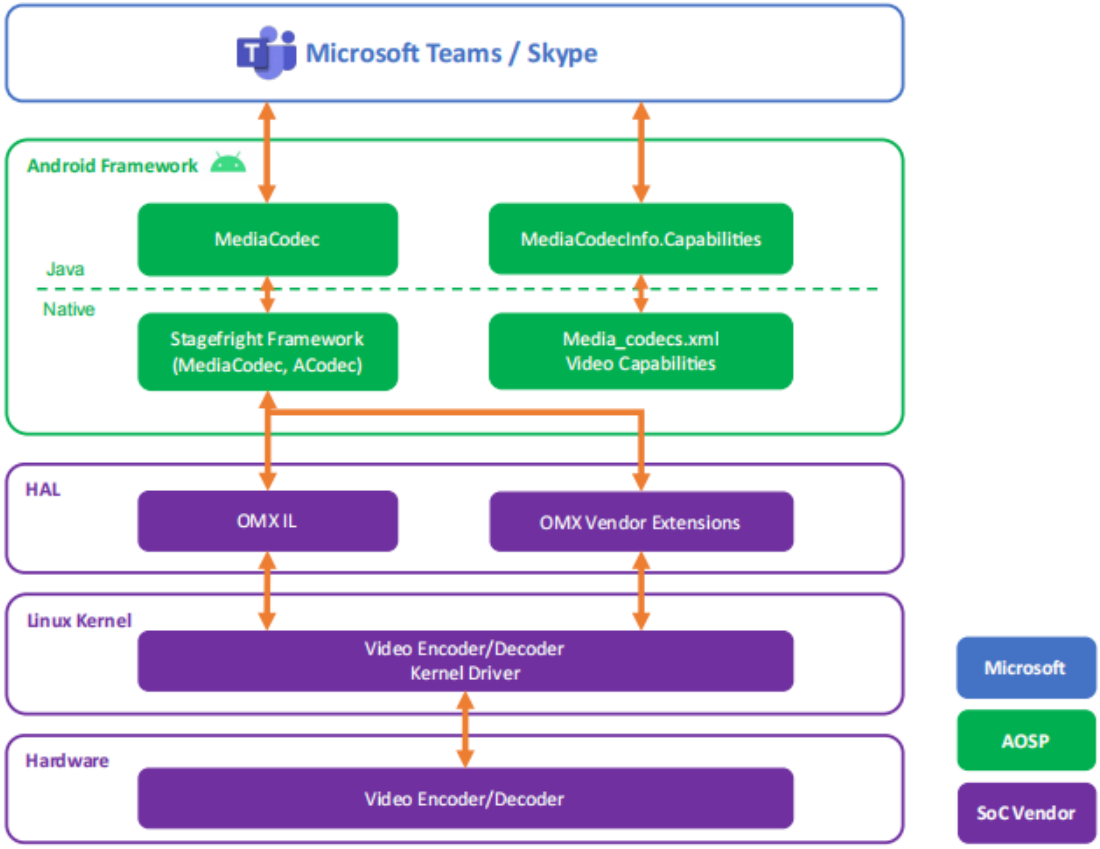
```
mediaFormat->setInt32("vendor.c2-enc-super-encoding.mode", 1);

// 设置值:
// 1    ===== 画质优先，追求更高的 psnr 和 vmaf 值
// 2    ===== 码率优先，追求更高的视频压缩率
// 3    ===== smart v3.0 模式，ROI 区域自适应识别，增强画质
```

可通过下面的打印确认生效：

```
c2_info("setupSuperMode: setup super mode %d", superMode);
```

2.8 微软 MVCT 协议认证支持



MVCT（Microsoft Codec Validation Tool）是微软 Teams 会议系统认证的编解码协议支持部分，适配与实现微软要求的扩展协议接口，实现如 TSVC\长短参考帧配置\帧级别QP设置\多slice切分\动态码率分辨率等功能。

微软 Teams 认证是一个严格且周期比较长的过程，对硬件设备 media\camera\audio 性能均有要求，客户认证设备需要分别通过：

- MVCT 编解码协议认证，适配支持微软 MVCT 扩展协议，添加编码 prefix nal 信息，完整通过微软 MVCT 测试认证。
- VCT 认证，压力测试认证，要求设备在多路编解码多路显示压测场景下，性能\CPU使用率\内存占用率等指标始终保持正常。
- 设备硬件认证，包括 Audio\Camera\光学等硬件认证。

Rockchip 平台目前已成功适配 RK3588\RK3399\PX30 芯片支持微软 MCVT 编解码协议支持，并支持相关设备通过 Teams，其中 RK3588 设备性能强劲，进入微软国产芯片推荐选型。

MVCT测试Tool

2.9 SEI 编码补充增强信息禁用支持

补充增强信息（Supplemental Enhancement Information）是码流范畴里面的概念，提供了向视频码流中加入信息的办法，是 H264/H265 视频压缩标准的特性之一。

SEI 通常集成在视频码流 SPS/PPS/IDR 前，添加用户自定义信息，如编码器参数\视频版权信息\摄像头参数等，这些信息对解码过程(容错、纠错)有帮助，但不是解码过程的必须项。

由于 Google GMS 的要求，RK 编码器默认会在码流初始位置插入一帧 SEI，但有发现一些老旧的解码器不支持 SEI 而导致解码失败的情况，因此提供用户扩展参数支持用户禁用 SEI 信息输出。

```
mediaFormat->setInt32("vendor.c2-enc-disable-sei.value", 1);
```

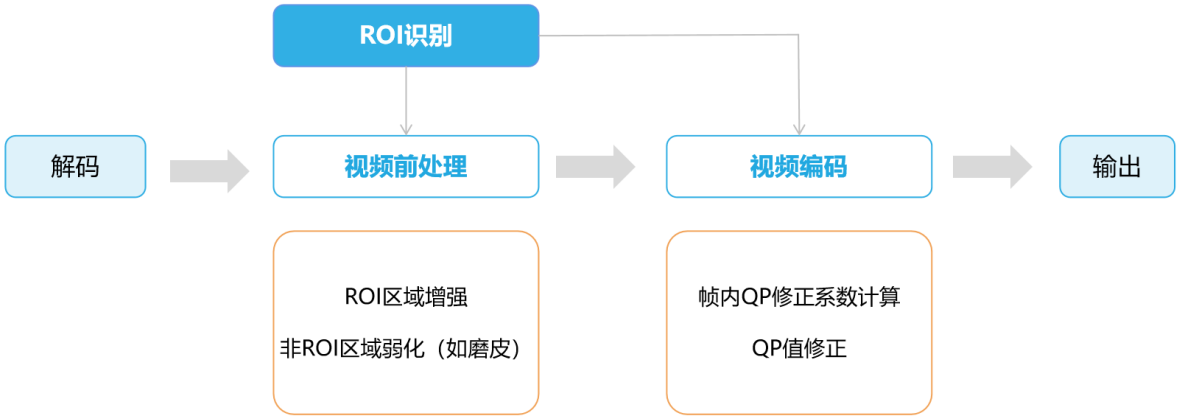
可通过下面的打印确认生效：

```
c2_info("disable sei info output");
```

2.10 编码 ROI 感兴趣区域设置

ROI（region of interest）是基于感兴趣区域的视频编码技术，即对图像中感兴趣的区域降低量化参数值，从而分配更多码率以提高编码质量，而对不感兴趣的区域则提高量化参数值，从而分配更少码率，在相同的码率限制下，获得更好的主观视觉效果。

ROI 视频编码 MediaCodec 应用的设计思路是，视频编码前处理识别 ROI 区域（人脸\中心区域等），并通过 MediaCodec->setParameter 传递 ROI 区域及编码质量修正参数。



参数配置说明:

1. 最多支持配置 4 个 ROI 区域，使用 vendor 扩展参数前缀 "roi-region-config" ~ "roi-region4-config" 标识
2. key 参数传递具体的 ROI 区域及编码量化参数，各参数含义如下:
    - left: ROI 区域左上角的水平坐标值
    - right: ROI 区域左上角的垂直坐标值
    - width: ROI 区域的宽度
    - height: ROI 区域的高度
    - force-intra: 是否指定 ROI 区域编码为 I-块
    - qp-mode： 配置值 0 - qpVal 为相对值 / 1 - qpVal 为绝对值
    - qp-val: ROI 区域内的宏块的 qp 值
3. qp-mode 为相对值时，设置的 qp 值为码率控制产生的 qp 与用户设定的 qp 偏移值的和
4. 参数配置为 "一次性"，每次 queueInputBuffer 前都需要更新区域配置

配置范例，配置两个 160x160 的 ROI 区域，并指定 QP 值为 10:

```
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.left", 0);
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.right", 0);
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.width", 160);
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.height", 160);
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.force-intra", 0);
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.qp-mode", 1);
mediaFormat->setInt32("vendor.c2-enc-roi-region-config.qp-val", 10);

mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.left", 500);
mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.right", 500);
mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.width", 160);
mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.height", 160);
mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.force-intra", 0);
mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.qp-mode", 1);
mediaFormat->setInt32("vendor.c2-enc-roi-region2-config.qp-val", 10);
```

可通过下面的打印确认生效：

```
c2_info("setup roi done, ctx %p regionCount %d", mRoiCtx, regionCount);
```

2.11 编码前处理支持，镜像\上下翻转

支持编码器的前处理，镜像和上下翻转，参考如下代码，使用扩展参数接口使能编码器前处理:

```
format->setInt32("vendor.c2-enc-preprocess.mirror", 1)
format->setInt32("vendor.c2-enc-preprocess.flip", 1)
```

可通过下面的打印确认生效：

```
c2_info("setupPreProcess: mirroring");
c2_info("setupPreProcess: flip");
```

2.12 超大帧处理，丢弃或重编

该功能适用于一些对网络带宽要求较高的场景，如视频序列中有些帧特别是 I 帧，可能超过预设值，比如单帧超过 1024KB，此时容易导致码率过冲而丢帧，丢帧继而引发花屏问题。系统支持设置超大帧的丢弃或重编来解决这一问题。

```
format->setInt32("vendor.c2-enc-super-process.mode", 1);
format->setInt32("vendor.c2-enc-super-process.super-i-thd", xxx);
format->setInt32("vendor.c2-enc-super-process.super-p-thd", xxx);
format->setInt32("vendor.c2-enc-super-process.max-reenc-times", 3);
```

参数说明:

1. mode: 超大帧处理模式，取值为 0~2，0 表示无特殊策略，1 表示丢弃超大帧，2 表示重编超大帧
2. super-i-thd: I 帧超大阈值，单位为 bps，另需注意， I 帧无法丢弃，但可设置超大 I 帧重编
3. super-p-thd: P 帧超大阈值，单位为 bps
4. max-reenc-times: 最大重编次数，建议设置为 2 或者 3，重编次数过多容易造成编码延时。

可通过下面的打印确认生效：

```
c2_info("setupSuperProcess, mode %d iThd %d pThd %d reencTimes %d",
        mode, iThd, pThd, reencTimes);
```

2.13 帧内刷新编码支持

常规编码器 I 帧通常比 P 帧分配更大的画质，因此 I 帧整体码率更高。在使用传统 IPPP..IPPP 编码架构模式时，两个 GOP 之间的 P 帧较小，而 I 帧出现时码率突然变高，这种码率突变不利于网络传输。

帧内刷新技术是视频编码中的一项关键技术，用于在不插入关键帧（I 帧）的情况下，通过周期性刷新部分区域来维持视频流的解码可靠性，因此帧内刷新也被称为 **视频渐近刷新 GDR(Gradual decoder refresh)**

1. 帧内刷新编码的结构为 IPPPPPPP，在一段时间周期内的所有 P 帧轮转刷新帧内编码块，直至覆盖全帧，替代传统的全帧关键帧编码，可以减少码率波动，有利于网络传输
2. 提升抗丢包能力，避免因单帧丢失导致的长时间画面错误

Android Codec2 原生框架支持帧内刷新编码的配置：

```
format->setFloat("intra-refresh-period", 20.0f);
```

可通过下面的打印确认生效：

```
c2_info("setupIntraRefresh: period(frames) %.1f refreshRowsPerFrame %d",
        intraRefresh->period, refreshRowsPerFrame);
```

参数配置说明：

1. 配置 peroid 为帧内刷新的帧间隔，单位为 frames，配置为 0 时代表关闭，如配置 20.0f 代表在 20 帧内轮转刷新完成。
2. 帧内编码默认为行刷新方式，如 h264 1080P 编码配置 20 帧帧内刷新间隔，则每帧刷新 ceil(ceil(1080 / 16) / 20) = 4，每帧刷新 4 行
3. 配置的刷新间隔需小于当前编码 GOP，刷新间隔帧数的配置建议如下。

	刷新间隔帧数大	**刷新间隔帧数小**
特点	每帧刷新宏块少	每帧刷新宏块多
优点	码率占用/码率波动更低	错误恢复更快
适用场景	4K 流媒体存储等	实时会议等需要优先保证抗丢包的场景 |