

Rockchip Android Multimedia FAQ

文件标识: RK-PC-YF-310

发布版本: V1.0.0

日期: 2022-03-03

文件密级: 绝密 秘密 内部资料 公开

免责声明

本文档按“现状”提供, 瑞芯微电子股份有限公司 (“本公司”, 下同) 不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因, 本文档将可能在未经任何通知的情况下, 不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标, 归本公司所有。

本文档可能提及的其他所有注册商标或商标, 由其各自拥有者所有。

版权所有 © 2022 瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

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 多媒体平台常见的调试手段以及开发过程中常见的问题清单。相关工程师遇到有关本文档涉及的问题, 可尝试通过提供方法进行调试, 收敛问题, 提高解决问题的效率, 进一步解决问题。

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

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	2022-03-03	初始版本
V1.0.1	陈锦森	2023-06-04	更新格式,补充常见问题

目录

Rockchip Android Multimedia FAQ

1 通用媒体类

- 1.1 片源无法播放
 - 1.1.1 查询片源是否在芯片支持规格内
 - 1.1.2 查询音视频格式是否涉及版权
- 1.2 常见芯片编解码能力规格表
 - 1.2.1 解码能力规格表:
 - 1.2.2 编码能力规格表:
- 1.3 如何提高 VPU 频率
 - 1.3.1 VPU 频率查询
 - 1.3.2 VPU频率修改
- 1.4 如何抓取编解码输入输出
- 1.5 多路编解码支持路数计算
- 1.6 多媒体应用 CPU 使用率高
- 1.7 多媒体应用内存泄漏问题分析
- 1.8 平台 JPEG 硬件编解码参考 demo

2 视频编解码类

- 2.1 片源卡顿\音视频不同步问题
- 2.2 录像录屏输出模糊、马赛克
- 2.3 MediaCodec 编码码率设置失控\超码率
- 2.4 编解码器初始化失败日志中提示“mpp hal xxx init failed”
- 2.5 MediaCodec BufferMode 解码效率提升

3 应用实用类

- 3.1 Kodi\哔哩哔哩等应用视频播放未使用硬件解码器
- 3.2 webview 视频播放失败或顶部出现白边
- 3.3 爱奇艺视频播放老化拷机出现 APP 闪退
- 3.4 播放器应用在播放过程中实时获取缩略图失败
- 3.5 RK356X 录屏或录像编码绿屏, 日志提示 RGA 报错
- 3.6 视频播放器应用剔除自定义音视频格式支持

1 通用媒体类

1.1 片源无法播放

1.1.1 查询片源是否在芯片支持规格内

使用 Windows 或 Linux 提供的 MediaInfo 工具可以查询到片源的媒体参数信息，包含编码格式、分辨率、码率、扫描方式、位深等基本信息。对比平台提供的芯片 Datasheet 手册或 Codec Benchmark 可初步判断片源是否支持。

Note: Datasheet 手册和 Codec Benchmark 标定芯片的编解码能力，是排查芯片支持问题的必要条件。如需获取，请邮件联系曾雅敏 (sw.fae@rock-chips.com)。



1.1.2 查询音视频格式是否涉及版权

平台由于版权原因已知明确不支持的音视频格式包括：

音频：mlp、ac3、eac3、dts、dsp、heaac、其他杜比相关
视频：divx、xvid、rmvb、vp6、vc1、svq、iso蓝光

1.2 常见芯片编解码能力规格表

芯片编解码规格可在 Datasheet 或 Codec Benchmark 中查询获取。以下收录平台常见芯片编解码能力的标定规格表，方便快速查询确认。

实际测试编解码能力与当时的系统负载相关，如 CPU/DDR 负载比较重时，可能出现编解码能力略低于 Datasheet 标定规格的情况。

1.2.1 解码能力规格表：

	H264	H265	VP9	JPEG
RK3588	7680x4320@30f	7680x4320@60f	7680x4320@60f	1920x1080@280f
RK3562	1920x1080@60f	4096x2304@30f	4096x2304@30f	1920x1080@60f
RK3528	4096x2304@30f	4096x2304@60f	4096x2304@60f	1920x1080@120f
RK356X	4096x2304@30f	4096x2304@60f	4096x2304@60f	1920x1080@80f
RK3399	4096x2304@30f	4096x2304@60f	4096x2304@60f	1920x1080@30f
RK3328	4096x2304@30f	4096x2304@60f	4096x2304@60f	1920x1080@30f
RK3288	3840x2160@30f	4096x2304@60f	N/A	1920x1080@30f
RK3358/PX5	4096x2160@25f	4096x2304@60f	N/A	1920x1080@30f
RK3326/PX30	1920x1080@60f	1920x1080@60f	N/A	1920x1080@30f
RK312X	1920x1080@60f	1920x1080@60f	N/A	1920x1080@30f

其他需要注意的点:

- 1.RK3588 支持 AVS2(7680x4320@60f) 及 AV1(3840x2160@60f) 解码
- 2.RK3528 支持 AVS2(4096x2160@60f) 解码
- 3.RK3562 不支持 mpeg1/2/4、vp8、h263 等格式解码
- 4.除 RK3562 外, 其他芯片支持 mpeg1/2/4、vp8、h263 解码, 最大规格为 1080P

1.2.2 编码能力规格表:

	H264	H265	VP8	JPEG
RK3588	7680x4320@30f	7680x4320@30f	1920x1080@30f	3840x2160@30f
RK3562	1920x1080@30f	N/A	N/A	N/A
RK3528	1920x1080@60f	1920x1080@60f	N/A	3840x2160@30f
RK356X	1920x1080@60f	1920x1080@60f	1920x1080@30f	1920x1080@60f
RK3399	1920x1080@30f	N/A	1920x1080@30f	1920x1080@30f
RK3328	1920x1080@30f	1920x1080@30f	1920x1080@30f	1920x1080@30f
RK3288	1920x1080@30f	N/A	1920x1080@30f	1920x1080@30f
RK3368/PX5	1920x1080@30f	N/A	1920x1080@30f	1920x1080@30f
RK3326/PX30	1920x1080@30f	N/A	1920x1080@30f	1920x1080@30f
RK312X	1920x1080@30f	N/A	1920x1080@30f	1920x1080@30f

1.3 如何提高 VPU 频率

VPU (Video Processing Unit) 是硬件视频处理单元, 评估硬件编解码的性能问题, 常常需要操作 VPU 频率。

通常认为, 当性能达到瓶颈时, 提高 VPU 和 DDR 频率对硬件的编解码能力有一定提升。但过高的频率对机器稳定性有一定影响, 建议客户集成前进行充分测试。

提高 DDR 频率请参考 SDK RKDocs/common/DDR/Rockchip-Developer-Guide-DDR/ 目录下文档说明。本节主要介绍平台 VPU 频率操作。

1.3.1 VPU 频率查询

VPU 频率可通过系统时钟树表查询 (cat /d/clk/clk_summary) , 以下列举常见 codec 的时钟树频率节点名称, 可以对照表格在 clk_summary 输出中查找。

	h264_dec	h265_dec	vp9_dec	jpeg_dec	h264_enc	h265_enc
RK3588	rkvdec	rkvdec	rkvdec	jpeg_decoder	rkvenc	rkvenc
RK3562	rkvdec	rkvdec	rkvdec	jdec	rkvenc	×
RK3528	rkvdec	rkvdec	rkvdec	jpeg_decoder	rkvenc	rkvenc
RK356X	rkvdec	rkvdec	rkvdec	jdec	rkvenc	rkvenc
RK3399	vdu	vdu	vdu	vcodec	vcodec	×
RK3328	rkvdec	rkvdec	rkvdec	vpu	h264	h265
RK3288	vcodec	hevc	×	×	vcodec	×
RK3368/PX5	video	video	×	video	video	×
RK3326/PX30	vpu	vpu	×	vpu	vpu	×
RK312X	vdpu	vdpu	×	vdpu	vdpu	×

举例说明 - 以下场景所需频率获取:

1.RK3588 H264 解码

```
$ cat /d/clk/clk_summary | grep rkvdec // <ac1k_rkvdec0> <ac1k_rkvdec1>
```

2.RK3288 h264 解码

```
$ cat /d/clk/clk_summary | grep vcodec // <ac1k_vcodec>
```

3.RK3328 H265编码

```
$ cat /d/clk/clk_summary | grep h265 // <ac1k_h265>
```

1.3.2 VPU频率修改

4.4 内核 (Android 7.1 ~ 9.0) 频率修改参考如下方式, 配置 vpu 频率跑 500M 测试, 4.4 内核驱动版本不支持单独配置 IP 的频率.

```
--- a/drivers/video/rockchip/vcodec/vcodec_service.c  
+++ b/drivers/video/rockchip/vcodec/vcodec_service.c  
@@ -2307,6 +2307,7 @@ static void vcodec_set_freq_default(struct vpu_service_info  
*pservice,  
{  
    enum VPU_FREQ curr = atomic_read(&pservice->freq_status);  
  
+    reg->freq = VPU_FREQ_500M;  
    if (curr == reg->freq)  
        return;
```

4.19/5.10 内核 (>=Android 10.0) 频率配置参考如下方式, 配置 rkvdec 频率为 500M 测试

1. mpp_service 驱动使用 dtsti 方式配置频率信息

芯片编解码器配置可在 dtsti 中查询，以下例子为 RK3399 配置解码器大于 4K 分辨率时提频至 500M。

```
<rk3399.dtsi>

rkvdec: rkvdec@ff660000{
    clock-names = "aclk_vcodec", "hclk_vcodec",
        "clk_cabac", "clk_core",;
    rockchip,normal-rates = <297000000>, <0>,
        <297000000>, <297000000>, ;
    rockchip,advanced-rates = <500000000>, <0>,
        <500000000>, <500000000>;
    rockchip,default-max-load = <2088960>; // 1920x1088
};

rockchip,normal-rates 为分辨率小于 1920x1088 时设置的 clock
rockchip,advanced-rates 为分辨率大于 1920x1088 时设置的 clock
```

2. 采用 debugfs 节点修改频率

用于临时测试，评估频率对编解码器的影响，可以采用如下方式：

```
$ echo 500000000 > /proc/mpp_service/rkvdec/aclk
$ echo 500000000 > /proc/mpp_service/rkvdec/clk_core
$ echo 500000000 > /proc/mpp_service/rkvdec/clk_cabac
```

1.4 如何抓取编解码输入输出

对花屏、绿屏类问题，抓取编解码的输入和输出有利于快速定位问题，缩小问题范围。

下面介绍平台保存解码输入输出的开关。

[应用上层]

1. 使用 MediaCodec API
 - a) 编码情况
 - 输入：queueInputBuffer 输入 buffer 前保存文件
 - 输出：dequeueOutputBuffer 后可以对编码输出 buffer 进行读写
 - b) 解码情况
 - 输入：queueInputBuffer 填输入 buffer 前保存文件
 - 输出：未配置 surface 的情况可以在 dequeueOutputBuffer 后对解码输出 buffer 进行读写

[框架层]

```
$ setenforce 0
$ mkdir /data/video/

1)Android 12 及以上的版本使用 Codec2 框架，按以下命令抓取
$ setprop vendor.dump.c2.log 0x000000f0

2)Android 11 及之前的版本使用 OMX 框架，按以下命令抓取
// 解码 dec_in*.bin
$ setprop vendor.omx.vdec.debug 0x01000000
$ setprop record_omx_dec_in 1
```

```
// 编码enc_in*.bin enc_out*.bin
$ setprop vendor.omx.venc.debug 0x03000000
$ setprop record_omx_enc_in 1
$ setprop record_omx_enc_out 1
```

如果以上命令在 /data/video/ 路径下无法生成文件，则可以使用系统底层编解码库的开关来抓取。

```
$ setenforce 0
$ mkdir /data/video/

$ setprop mpp_dump_in /data/video/mpp_dec_in.bin
$ setprop mpp_dump_out /data/video/mpp_dec_out.bin
$ setprop vendor.mpp_dump_in /data/video/mpp_dec_in.bin
$ setprop vendor.mpp_dump_out /data/video/mpp_dec_out.bin
$ setprop mpp_debug 0x600 && setprop vendor.mpp_debug 0x600
```

1.5 多路编解码支持路数计算

芯片硬编解码最大支持路数换算涉及硬件 pixel 算力，以一个具体例子说明：

问题：RK3399 最大能支持多少路 H264 1080P@30fps 规格解码？

查询 RK3399 规格表可知 H264 的解码能力为：4096x2304@30fps。

1) 硬件 pixel 算力：一秒钟4096x2304x30f

2) 换算 1080P@30fps，计算方式为：

$$(4096 \times 2304 \times 30) / (1920 \times 1088 \times 30) = 4.5$$

3) 路数越多，折算损耗也越大，一般最后值计算为向下取整，故支持 4 路 1080P@30fps

其他 codec 可参考类似计算，需要注意的是以上计算依据为高码率极限片源，因此可保证任何情况下都支持 H264 4 路 1080P@30fps 解码。

RK3588、RK356X、RK3399、RK3328 搭建高性能解码器 rkvddec (H264、H265、VP9)，对 H264\H265\VP9 解码，极限情况下普通的测试片源有机会突破计算极限。

如 RK3399 有机会能达到 1080P@30fps 8 路解码，具体评估方式如下：

前提：非高码率片源

1) 解码性能指标：VPU 驱动内核单帧解码时间

```
4.19/5.10 内核 (Android 10.0 及以上版本)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
$ cat /proc/kmsg
```

```
4.4 内核 (Android 7.1 到 9.0)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/debug
$ cat /proc/kmsg
```

要支持 8 路 1080P@30fps 需要的单帧解码时间为 (时间 / 总帧数)：

$$\rightarrow (1 \times 1000) / (8 * 30) \approx 4.16 \text{ ms}$$

抛开解码路上的时间损耗，一帧的解码时间在 4ms 以内可以符合需求，如果目前测试单帧解码时间达不到所需，按下面方式评估改善频率信息，通常提高 VPU 频率或 DDR 频率对硬编解码性能会有一定提升。

2) 频率信息

通常认为当性能达到瓶颈时，提高 VPU 和 DDR 频率对硬件的编解码能力有一定提升。因此查看测试过程中的 VPU 频率及 DDR 频率。如果查看频率尚有提升空间，可以尝试提高频率后继续按步骤 2 查看内核解码时间是否能达到需求。

```
/* VPU 频率 */
$ cat /d/clk/clk_summary | grep vdu      <ac1k_vdu>      rk3399
$ cat /d/clk/clk_summary | grep rkvddec <ac1k_rkvdec> rk3588\rk3328\rk356x

/* DDR 频率 */
$ cat /sys/class/devfreq/dmc/cur_freq
```

3) 提高 VPU 频率

请参考第 1.3 节的介绍提高 VPU 频率。

4) 提高 DDR 频率

```
echo performance > /sys/class/devfreq/dmc/governor // 将 DDR 频率设为
performance
```

具体提高 DDR 频率请参考 RKDocs/common/DDR/Rockchip-Developer-Guide-DDR/ 文档说明

1.6 多媒体应用 CPU 使用率高

【问题描述】

客户的定制化多媒体编解码应用，运行时整体 CPU 使用率高，导致系统卡顿。

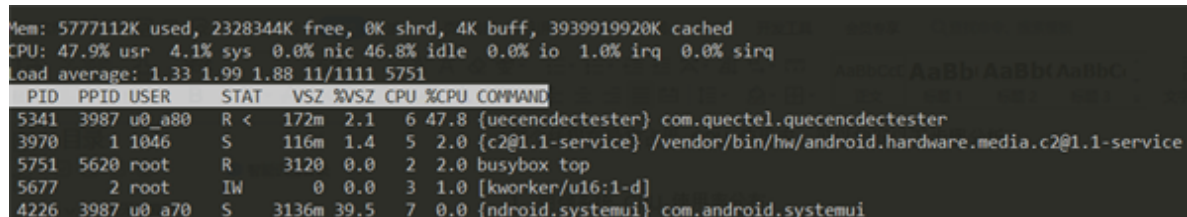
【问题分析】

媒体应用上大量的读写操作同样会占用 CPU。因此首先我们需要分责，需要细分到具体是哪个进程哪个线程在异常占用 CPU。可以参考以下步骤分析，从**进程->线程->函数**一步一步缩小分析范围。

1. 明确系统 CPU 使用率分布

问题场景下，使用系统 **adb shell busybox top** 实时显示各进程的 CPU 占用情况。

如下图所示，应用进程 com.quectel.quecncdectester 占用 47.8% 的 CPU 使用率，是需要重点分析的进程。



```
Mem: 5777112K used, 2328344K free, 0K shrd, 4K buff, 3939919920K cached
CPU: 47.9% usr 4.1% sys 0.0% nic 46.8% idle 0.0% io 1.0% irq 0.0% sirq
Load average: 1.33 1.99 1.88 11/1111 5751
  PID  PPID  USER  STAT  VSZ %VSZ CPU %CPU COMMAND
 5341  3987  u0_a80  R <   172m 2.1  6 47.8 {uecncdectester} com.quectel.quecncdectester
 3970   1  1046   S    116m 1.4  5  2.0 {c2@1.1-service} /vendor/bin/hw/android.hardware.media.c2@1.1-service
 5751  5620  root    R     3120 0.0  2  2.0 busybox top
 5677   2  root    IW     0  0.0  3  1.0 [kworker/u16:1-d]
 4226  3987  u0_a70   S    3136m 39.5  7  0.0 {ndroid.systemui} com.android.systemui
```

2. 确认异常进程中各个线程的CPU占用率情况

问题场景下，使用系统命令 **top -H -p \$(pidof com.quectel.quecncdectester)** 来查看应用进程中各个线程的 CPU 占用率情况，如果主要是应用中的线程 CPU 占用，则需要客户自行分析。如果是系统框架线程 CPU 占用，则转交 redmine 处理。

如下图所示，CodecLooper 为 MediaCodec 框架编解码处理线程，Sample 程序正在进行多路解码测试，此时框架编解码线程处理占用了绝大部分 CPU 使用率。

```
Mem: 8105456K total, 5833500K used, 2271956K free, 4268K buffers
Swap: 4052724K total, 0K used, 4052724K free, 3013984K cached
800%cpu 288%user 1%nice 117%sys 388%idle 0%iow 5%irq 1%irq 0%host
```

TID	USER	PR	NI	VIRT	RES	SHR	S[%CPU]	%MEM	TIME+	THREAD	PROCESS
5789	u0_a80	4	-16	16G	1.0G	82M	R 46.3	13.7	0:05.04	CodecLooper	com.quectel.quecencdectester
5783	u0_a80	4	-16	16G	1.0G	82M	S 45.3	13.7	0:05.51	CodecLooper	com.quectel.quecencdectester
5787	u0_a80	4	-16	16G	1.0G	82M	R 45.0	13.7	0:05.04	CodecLooper	com.quectel.quecencdectester
5777	u0_a80	4	-16	16G	1.0G	82M	S 45.0	13.7	0:05.59	CodecLooper	com.quectel.quecencdectester
5788	u0_a80	4	-16	16G	1.0G	82M	R 44.6	13.7	0:04.56	CodecLooper	com.quectel.quecencdectester
5781	u0_a80	4	-16	16G	1.0G	82M	R 44.3	13.7	0:05.46	CodecLooper	com.quectel.quecencdectester
5784	u0_a80	10	-10	16G	1.0G	82M	S 4.0	13.7	0:00.72	MediaCodec_loop	com.quectel.quecencdectester
5779	u0_a80	10	-10	16G	1.0G	82M	S 4.0	13.7	0:00.71	MediaCodec_loop	com.quectel.quecencdectester

3. 分析函数的CPU使用情况

Android 上的 CPU 性能分析工具很多，可以使用 Android Studio IDE 集成的 TraceView\Systrace 分析工具。本章节介绍 SDK 自带的 simpleperf 工具来统计 CPU 使用率。

紧接上面 Sample 程序的分析，多路解码应用中框架线程 CodecLooper CPU 使用率较高。我们可以使用 simpleperf 工具具体到各个函数的 CPU 使用率占比。

```
1. 使用以下命令实时记录 CPU 事件，1349 为进程 pid，duration 10 代表记录 10 秒
simpleperf record -o /sdcard/perf.data -g -p 1349 --duration 10

2. 记录结束，使用以下命令生成分析报告
simpleperf report -i /sdcard/perf.data -g caller > /sdcard/perfdata
```

/sdcard/perfdata 为生成的分析报告，如下图所示，CodecLooper 为 CPU 占用率最多的线程，15.31% 的时间 CPU 都在处理该线程。而该线程中大部分的时间又集中在处理 libyuv::X420ToI420 函数。

```
Cmdline: /system/bin/simpleperf record -o /sdcard/perf.data -g -p 5901 --duration 10
Arch: arm64
Event: cpu-cycles (type 0, config 0)
Samples: 142336
Event count: 73178921430
```

Children	Self	Command	Pid	Tid	Shared	Symbol
Object	15.31%	CodecLooper	5901	6314		/apex/com.android.runtime/lib64/bionic/libc.so
						__start_thread
						__pthread_start(void*)
						thread_data_t::trampoline(thread_data_t const*)
						android::Thread::_threadLoop(void*)
						...
						...
						android::OutputBuffersArray::registerBuffer(std::__1::shared_ptr<C2Buffer> const&, unsigned long*, android::sp<android::MediaImage2D> const&, android::ConstGraphicBlockBuffer::copy(std::__1::shared_ptr<C2Buffer> const&)
						android::ImageCopy(unsigned char*, android::MediaImage2 const*, C2GraphicView const&) (.cfi)
						NV12ToI420.cfi
						libyuv::X420ToI420(unsigned char const*, int, int, unsigned char const*, int, unsigned char*, int, int, int)

该函数作用为 MediaCodec Buffer Mode 解码模式下将框架解码输出拷贝到应用外部，由于应用无法处理框架的 dma 解码输出，因此这部分 CPU 拷贝工作无法避免。本文档 <MediaCodec BufferMode 解码效率提升> 章节中，将 CPU 拷贝替换成平台硬件 RGA 拷贝，可改善该场景下的 CPU 使用率问题。

同样的分析步骤可以排查应用中其他 CPU 占用率问题。

1.7 多媒体应用内存泄漏问题分析

【问题描述】

多媒体应用运行过程中系统可用内存越来越少，最终OOM (Out Of Memery) 系统重启。

【问题分析】

内存泄露问题可参考以下步骤排查:

1. 确认是哪个进程的内存泄漏

首先需要明确是定制应用还是系统进程存在的泄漏，可以通过确认问题出现前后 meminfo 信息，对比各个进程 RSS\PSS 内存占用，确认是哪个进程内存占用一直增多，即该进程存在泄漏的可能。

```
dumpsys meminfo
```

2. 确认内存泄漏类型

明确泄漏类型有助于我们更好的定位问题点。视频播放类型主要可能存在的泄漏类型为 malloc 或者 dmabuf 类型，dmabuf 用于解码输出及显示，是零拷贝视频播放的基础，通过 dma buffer 的接口 (ION\DRM\dmaBufferHeap) 申请。malloc 就是 alloc、malloc 函数申请的内存。

Linux showmap 命令用于定位进程申请的内存 map 表。

```
showmap $(pidof xxx)
```

其中 RSS\PSS 为进程占用内存，单位为K，找出使用比较大的内存块，明确泄漏类型，如果是 dmaBuf 类型泄漏就主要检查视频播放初始化、销毁以及 info-change 时的 buffer 申请释放。如果是 malloc 内存泄漏，可以使用 Android Malloc Debug 定位泄漏堆栈。

virtual size	RSS	PSS	shared clean	shared dirty	private clean	private dirty	swap	swa
113328	10628	6005	5352	228	1960	3088	0	

Map 内存 object 类型:

- /dmabuf: dma buffer 类型内存
- malloc 或 anon:scudo malloc 类型内存

3. Android Malloc Debug定位内存泄漏增长点堆栈

Malloc Debug 是 Android 原生提供用于调试 native memory 内存泄露\内存损坏\内存释放问题的工具。相关使用说明文档在 SDK bionic/libc/malloc_debug/README.md。

以下为示例使用，假定我们需要内存检测的程序名为 myTest。

```
$ adb shell stop
$ adb shell setprop libc.debug.malloc.program myTest
$ adb shell setprop libc.debug.malloc.options backtrace
$ adb shell start
$ 开始测试复现问题...
$ kill -9 $(pidof myTest)
```

默认抓取到的内存快照保存在路径: /data/local/tmp/backtrace_heap.3302.txt，其中 3302 为 myTest 程序的进程 pid。

生成的内存快照可以使用 SDK native_heapdump_viewer.py 工具解析生成泄露堆栈。

```
$ python development/scripts/native_heapdump_viewer.py --symbols $(OUT)/symbols backtrace_heap.3378.txt > out_heap.txt
```

根据 out_heap 内存堆栈信息所示的代码，修复内存泄露点。

1.8 平台 JPEG 硬件编解码参考 demo

Google 原生 MediaCodec 通路不支持 JPEG 编解码，需要在系统多媒体应用集成 JPEG 硬编解码功能可以使用 MPP JPEG 封装库 libhwjpeg。

相关代码在 SDK 路径: hardware/rockchip/libhwjpeg

libhwjpeg 用于支持 Rockchip 平台 JPEG 硬编解码，是平台 MPP(Media Process Platform)库 JPEG 编解码逻辑的封装。

其中 MjpegEncoder 类封装了硬编码相关操作，MjpegDecoder 类封装了硬解码相关操作，用于支持图片或 MJPEG 码流解码。工程主要目录：

- src: 库实现代码
- inc: 应用接口头文件
- test: 用户测试实例

具体使用方法请参考目录下 readme.txt 说明。

2 视频编解码类

2.1 片源卡顿\音视频不同步问题

视频播放卡顿、声音卡顿、音视频不同步等可以归为流畅性问题。流畅性问题的分析需要依托显示帧率 FPS 及内核单帧解码时间。

```
// 显示帧率 FPS
$ setprop debug.sf.fps 1
$ logcat -c ;logcat | grep mFps

// 内核单帧解码时间
4.19/5.10 内核 (Android 10.0 及以上版本)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/mpp_dev_debug
$ cat /proc/kmsg

4.4 内核 (Android 7.1 到 9.0)
$ echo 0x0100 > /sys/module/rk_vcodec/parameters/debug
$ cat /proc/kmsg
```

对于视频播放不流畅，主要原因可以归结为以下几点，工程师遇到相关问题时可以参考以下方式进行排查：

1. 检查是否使用平台硬解

一些视频网站或应用程序，由于代码及使用 API 未知，可能使用软解来进行视频格式解码。因此这类问题卡顿发生可以先在 Logcat 日志中查找相应的视频格式，如果确定视频格式及规格都在芯片解码能力范围之内而未使用平台硬解，请先检查程序或网站是否存在配置用来控制使用硬解与否。

判断是否使用平台硬解可以键入查询内核单帧解码时间，经过内核硬件处理则会有值打印，说明当前走硬件解码，否则说明走软解。

2. 确认问题点在解码还是显示

视频播放存在两个动作：解码与显示，解码输出提交给显示做渲染，因此请先查询第 1.2 节芯片编解码能力规格表，确定在解码能力范围内的片源用上面的命令查询下内核单帧解码时间及显示帧率。

如果单帧解码时间足够（30 帧源 33 ms 之内/60 帧源 16 ms 之内），则可先归纳为显示合成效率不够，按第 3 小节排查。

3. 显示合成效率不够

屏幕刷新率配置异常会导致 hwc 合成效率低下，因此首先检查 dts 中屏幕刷新率是否正确配置。

```
rk3566_r:/ # cat /d/dri/0/summary
Video Port0: DISABLED
Video Port1: ACTIVE
  Connector: DSI-1
    bus_format[100a]: RGB888_1X24
    overlay_mode[0] output_mode[0] color_space[0]
  Display mode: 1080x1920p60
    clk[132000] real_clk[132000] type[48] flag[a]
  H: 1080 1095 1097 1127
```

屏幕刷新率一般配置为 60 fps，如果有问题按以下公式纠正 dts 配置中屏对应的 panel-timing 设置。

```
clock-frequency = (hactive + hback-porch + hfront-porch + hsync-len) * (vactive +
vback-porch + vfrontporch + vsync-len) * fps
```

非屏参配置问题，原因通常是以下几点：

- 视频显示输出存在角度，也就是屏幕旋转输出
- 场景 Surface 层数过多，存在多个图层
- 视频格式不支持

可以抓取以下日志并提交 redmine 指派对应的工程师处理。

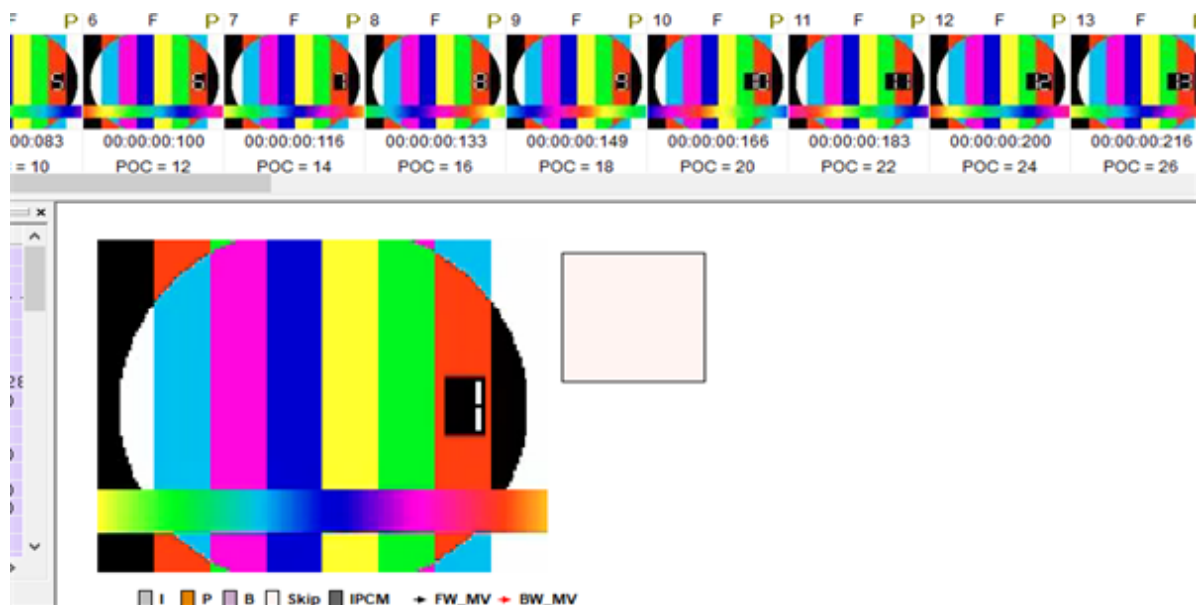
```
// 通过 SurfaceFlinger Services 检查合成策略是否正常
dumpsys SurfaceFlinger

// 若不正常，通过打印 HWC log 查看不正常原因
adb shell "setprop sys.hwc.log 51"
adb shell "logcat -c ;logcat" > hwc
```

4. 视频解码过程存在解码错误

日志中出现 "error frame" 等相关的打印，则说明硬解过程出错，出现了丢帧。此类问题请先确认送给解码器的码流无误且无丢帧，媒体框架中提供了几种 Dump 解码输入的方式，请参考第 4 章节命令进行操作。

抓取的输入为纯视频轨，可以使用 PC 工具查看分析，如 H264 格式可使用 eseye 或 Vega H264 Analyzer 分析 H265 格式使用 HEVCAnalyzer，通常这些工具可以判断出码流本身是否存在问题或丢帧。如下面的 vega 分析工具预览，POC 值连续且预览无误则说明码流正常且无丢帧。



确认解码输入正确，则说明目前的解码框架对该码流适配出现问题，可提交对应的码流并附上 Logcat 日志提交 redmine 指派相应工程师处理。

2.2 录像录屏输出模糊、马赛克

【问题描述】

录像录屏输出文件模糊\马赛克\不清晰

【问题分析】

1. 平台编码属于有损压缩编码，编码过程中存在图像数据的丢失，因此编码输出与原图像存在一定差异。
2. 模糊、马赛克现象产生的原因通常是局部编码质量过低，而 QP 是编码质量量化参数，范围在 1 ~ 51，QP 值越小编码质量越高。

在用户接口不限定 QP 范围的情况下，QP 范围由用户设定码率决定。码率越大，编码质量越高，单帧 QP 值就越小。所以编码马赛克问题通常排查步骤如下：

1. 检查编码输出的 QP 范围

eseye 或 Vega H264 Analyzer 等 PC 工具提供了单帧图像质量查询，如下图所示该帧的 QP 范围为 16~29，大于 40 QP 值的宏块可能会引起模糊、马赛克的出现，如果出现了 40 几的 QP 质量值，需要继续往下排查码率和编码质量策略是否需要调整，如果图像 QP 范围正常且出现马赛克，则需要抓取编码输入来排查是否输入本身就存在马赛克。

Info...

File	Picture	Headers	MB
mb count	:	8 160	
picture size (bits)	:	534 390 [66.798]	
transform coeff size (bits)	:	408 519 [76.45%]	
mv size (bits)	:	50 930 [9.53%]	
max mb size (bits)	:	496 [3762]	
max qp	:	29 [47]	
min qp	:	16 [6840]	
max mv x	:	742 [2619]	
max mv y	:	240 [11]	
min mv x	:	-650 [3762]	
min mv y	:	-241 [608]	

2. 检查码率参数合理性

不同编码器的压缩性能各异，没有标准的码率对照表，界定码率是否合理，可以对照编码输出的图像质量。

- a) 如果实际码率 \geq 应用设定的码率，且出现图像模糊 QP 值过大的情况判定为用户设定码率过小，建议调大码率测试。
- b) 如果实际码率 $<$ 应用设定的码率，且出现图像模糊 QP 值过大的情况检查应用接口是否设置了可变码率模式，BITRATE_MODE_VBR
- c) 如果实际码率与应用设定的码率相符，则可能需要手动调整图像 QP 策略。

3. 调整图像 QP 策略

首先如果当前为 baseline 编码，可以尝试切换成 high profile，high profile 使用 cabac 熵编码，整体压缩率更高，同等码率图像质量更好。

在 Android 12 及之后的版本，MediaCodec 支持图像质量控制，用户接口可以界定 QP 范围。设定 Max QP 不大于 40，可有效改善模糊、马赛克问题。

应用层 MediaCodec 接口代码可参考如下设置（其中 qp-i-min 代表 I 帧的最小 QP 值，qp-p-max 代表 P 帧的最大 QP 值）：

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

需要注意的是，调整 QP 范围可能会出现码率失控的情况，需要额外留意编码输出码率情况。Android 11 及之前的版本用户接口暂不支持 QP 范围控制，可以通过修改 MPP 参数配置接口来对 QP 范围进行限定。

```
diff --git a/mpp/legacy/vpu_api_legacy.cpp b/mpp/legacy/vpu_api_legacy.c
index 9a4f4067..1982af83 100644
--- a/mpp/legacy/vpu_api_legacy.cpp
+++ b/mpp/legacy/vpu_api_legacy.cpp
@@ -146,9 +146,9 @@ static MPP_RET vpu_api_set_enc_cfg(MppCtx mpp_ctx, M
     mpp_enc_cfg_set_s32(enc_cfg, "h264:cabac_idc", 0);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_init", -1);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_min", 10);
-    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max", 51);
+    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max", 40);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_min_i", 10);
-    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max_i", 51);
+    mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_max_i", 40);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_step", 4);
     mpp_enc_cfg_set_s32(enc_cfg, "h264:qp_delta_ip", 3);
 } break;
```

2.3 MediaCodec 编码码率设置失控\超码率

【问题描述】

在 ≥ 12 的 Android 版本中，实际码流码率比设置值偏大，问题场景通常是用户设置码率偏小，如用户 1080P 分辨率编码码率设置 1Mbps，但实际编码码流是 4Mbps 多。

【问题原因】

Android 12 版本中新使能了 media format shaping，用于对一些不合理配置 QP、bitrate 等做纠正。

1Mbps 码率对于 1080P 分辨率被框架认定为不合理，框架 VQApply 模块根据用户设置码率、宽、高、bpp 等计算出一个参考码率，并纠正赋值给编码器使用，这个纠正值通常大于 $w * h * bpp$ 。

日志中通常会有以下 VQApply 打印，代表用户设置码率为 1258291 bps，VQApply 框架纠正为 4727808 bps，并赋值给编码器作为真正的码率使用。因此就出现了实际码率偏大的场景。

```
D VQApply : minquality/target bitrate raised from 1258291 to 4727808 bps
```

【问题解决】

谷歌提供了属性值 `debug.stagefright.enableshaping` 用于关闭框架的 format shaping 纠正。如果不需要这个特性，可以设置属性关闭 `setprop debug.stagefright.enableshaping 0`。

同样可以直接在代码中关闭。

Path: frameworks/av

```
diff --git a/media/libstagefright/MediaCodec.cpp b/media/libstagefright/MediaCodec.cpp
index da581fb7f1..eb1a6fbf90 100644
--- a/media/libstagefright/MediaCodec.cpp
+++ b/media/libstagefright/MediaCodec.cpp
@@ -1490,7 +1490,7 @@ status_t MediaCodec::setOnFirstTunnelFrameReadyNotification(const sp<AMessage> &
 * MediaFormat Shaping forward declarations
 * including the property name we use for control.
 */
-static int enableMediaFormatShapingDefault = 1;
+static int enableMediaFormatShapingDefault = 0;
static const char enableMediaFormatShapingProperty[] = "debug.stagefright.enableshaping";
```

2.4 编解码器初始化失败日志中提示“mpp hal xxx init failed”

【问题描述】

片源播放\录像录屏\拍照等场景编解码器初始化失败，且日志中提示“mpp hal xxx init failed”

【问题分析】

```
E HAL_JPEG_VDPU2: hal_jpegd_vdpu2_init mpp_dev_init failed. ret: -1
E mpp_hal : mpp_hal_init hal jpegd init failed ret -1
```

编解码器初始化失败，问题日志中提示 mpp hal xxx init failed，此类问题通常是客户 dts 未做正确配置，没有使能相应的编解码器节点。在客户 dts 里增加使能相应的节点即可解决问题。

如上日志 hal_jpegd_vdpu2 提示两点关键信息：

- 1. jpegd 也就是 jpeg 解码器初始化失败
- 2. vdpu 为相应需要使能的节点

在 MPP 驱动设备目录下确认 vdpu 节点是否正确使能。

```
ls -al /proc/mpp_service/
```

如确认未使能，可参考 kernel 下对应芯片的 dtsi 配置，如 rk3588-evb.dtsi \ rk3399-android.dtsi。

上述的问题日志增加的对应配置为：

```
&mpp_srv {
    status = "okay";
};

&vdpu {
    status = "okay";
};
```

另一个类似的报错问题日志：

```
E hal_h264e_vepu541: hal_h264e_vepu541_init mpp_dev_init failed. ret: -1
E mpp_enc_hal: mpp_enc_hal_init hal hal_h264e init failed ret -1
E mpp_enc_hal: mpp_enc_hal_init could not found coding type 7
```

h264e 也就是 264 编码器初始化失败，需要增加使能的节点名称是 vepu。因此相对应需要增加的配置为：

```
&mpp_srv {
    status = "okay";
};

&vepu {
    status = "okay";
};
```

2.5 MediaCodec BufferMode 解码效率提升

【问题描述】

应用接口使用 MediaCodec Buffer Mode 即不配置 surface 解码，表现为单帧解码时间变长且CPU使用率变高。问题发生时通常应用场景本身 CPU 负载就偏大。

【问题原因】

应用接口 MediaCodec 不配置 Surface 进行解码，意味着外部应用接口需要取到解码输出 buffer 做额外处理，而框架解码输出为 dmaBuffer 无法直接提供给应用，因此 Buffer Mode 模式下，需要额外的一次拷贝，将解码输出从框架拷贝到应用接口。

相对应的 Surface Mode 也就是配置 Surface 解码显示的场景，我们通常称之为零拷贝场景。这种模式下，解码和显示不需要经过任何拷贝，直接使用 dmaBuffer 传递，效率比 Buffer Mode 要高一些。

Buffer Mode 解码模式由于额外的一次 CPU 拷贝处理，单帧的解码处理时间变长且 CPU 使用率变高。

【问题解决】

使用系统 RGA 硬件拷贝来代替 CPU 拷贝，可以显著提升解码时间，并降低场景的 CPU 使用率。

具体的修改补丁请查阅 rk 补丁简报: <https://redmine.rock-chips.com/issues/418670>

3 应用实用类

3.1 Kodi\哔哩哔哩等应用视频播放未使用硬件解码器

【问题描述】

在 Androi 12 以上的版本，使用 github ijkmedia 播放框架的应用（目前已知 Kodi 和哔哩哔哩应用），视频播放时无法使用硬件解码器。

【问题原因】

ijkmedia 框架 MediaCodec 选择解码器时只过滤 OMX. 打头的 codec name 作为硬件解码器。而 Andorid 12 使用 codec2 新框架，硬解码器 codec name 为 c2.rk.avc.decoder，因此无法选择到平台硬件解码器。

【解决方式】

修改增加 "OMX.c2" 解码器，用于支持适配 ijkmedia 框架，如将 c2.rk.avc.decoder 重定向为 OMX.c2.rk.avc.decoder。

除了 Kodi\哔哩哔哩等应用，客户自定义的应用走 ijkmedia 媒体框架同样无法走到硬解。可以通过日志是否有 "IJKMEDIA" 字样打印确认。

具体的修改补丁请查阅 rk 补丁简报: <https://redmine.rock-chips.com/issues/396998>

该问题判定为 ijkmedia 框架问题，后续规范解决还是需要通过修改框架代码，因此先作为临时补丁提供，不提交到版本。

3.2 webview 视频播放失败或顶部出现白边

【问题描述】

在带 afbc 解码输出功能的 soc 芯片(rk3566\rk3568\rk3588\rk3528)，客户应用或网页使用 webview 视频播放失败或顶部出现白边。

【问题原因】

背景：fbc 解码格式下，硬件固定输出上扩 4，因此解码输出带 offset 4 行高度偏移。

webview 视频播放最后使用 ImageReader\NdkImageReader 获取 video surface 的 buffer 用于渲染，而 fbc 模式下，video buffer 带 4 行的高度偏移，因此显示时需要正确处理 buffer 的 crop，否则 4 行 offset 高度显示出来就会出现白边。

【问题解决】

高于 1080P 分辨率视频播放才会使用 fbc 模式，因此以下补丁适用于 rk356x\rk3588\rk3528 webview 1080P 以上分辨率视频播放。

1. 针对 webview 视频播放失败

报错日志为:

```
E NdkImageReader: Crop left top corner [0, 4] not at origin
```

默认代码中，ImageReader 对 buffer 的 left-top corner 做了限制，只能为 0，而 fbc 模式下的解码输出带 4 行的高度偏移，因此 top corner 为 4，可以将该限制解除即可解决播放失败问题。

补丁路径: frameworks/av

```
diff --git a/media/ndk/NdkImageReader.cpp b/media/ndk/NdkImageReader.cpp
index 92704995e0..3d78bf7f6b 100644
--- a/media/ndk/NdkImageReader.cpp
+++ b/media/ndk/NdkImageReader.cpp
@@ -451,7 +451,8 @@ AImageReader::acquireImageLocked(/*out*/AImage** image, /*ou
    Point lt = buffer->mCrop.leftTop();
    if (lt.x != 0 || lt.y != 0) {
        ALOGE("Crop left top corner [%d, %d] not at origin", lt.x, lt.y);
-       return AMEDIA_ERROR_UNKNOWN;
+       // for support afbc mode
+       //return AMEDIA_ERROR_UNKNOWN;
    }
}
```

2. 针对 webview 视频播放出现白边

针对已经带上述限制移除补丁的机器，大于 1080P webview 视频播放顶部出现白边。

本地播放器播放无白边的原因在 source crop 在播放器端自主可控，可以给 surface 传递我们想要选择和裁剪的区域。

而 webview 渲染通过 ImageReader 获取 surface buffer，应用框架进而调用 getHardwareBuffer 直接获取到 GraphicBuffer。因此 crop 的处理工作移交到了 webview 应用代码中去，框架暂时无法解决白边。如果客户无法接受白边，可以在 codec 组件中关闭 fbc 解码模式，代价是非 fbc 模式解码性能可能略有下降。

Android 11 及之前的补丁 (hardware/rockchip/omx_il) :

```
diff --git a/osal/Rockchip_OSAL_Android.cpp b/osal/Rockchip_OSAL_Android.cpp
index 2f07785..7e6edf2 100755
--- a/osal/Rockchip_OSAL_Android.cpp
+++ b/osal/Rockchip_OSAL_Android.cpp
@@ -258,6 +258,7 @@ OMX_BOOL Rockchip_OSAL_Check_Use_FBCMode(OMX_VIDEO_CODINGTY
    OMX_U32 width, height;

+   return OMX_FALSE;
    Rockchip_OSAL_GetEnvU32("omx_fbc_disable", &pValue, 0);
    if (pValue == 1) {
        return OMX_FALSE;
    }
}
```

Android 12 及之后的版本补丁 (vendor/rockchip/hardware/interfaces/codec2) :

```
diff --git a/osal/C2RKChipFeaturesDef.cpp b/osal/C2RKChipFeaturesDef.cpp
index 47e3a49..a75a506 100755
--- a/osal/C2RKChipFeaturesDef.cpp
+++ b/osal/C2RKChipFeaturesDef.cpp
@@ -257,6 +257,7 @@ static const int C2ChipFeatureSize = sizeof(FeattrueInfos
 int C2RKChipFeaturesDef::getFbcOutputMode(MppCodingType codecId) {
     RKChipInfo *chipInfo = getChipName();

+   return 0;
     if (chipInfo == NULL)
         return 0;
```

3.3 爱奇艺视频播放老化拷机出现 APP 闪退

【问题描述】

在 >= Android 10.0 的版本，使用爱奇艺 APP 做视频播放老化拷机，内存泄露导致 lowmem 低内存，最终系统异常 APP 闪退。

【问题原因】

泄露类型为 dma buffer 内存，泄露发生在正片广告播放时，应用使用 MediaPlayer 接口，surface 设置时机比较晚，因此播放器先使用 buffer mode 解码，而后 surface 设置请求到来，触发 surface change。播放器需要针对新 surface 申请新的解码 buffer，并同步释放旧的解码 buffer。

这种情况下旧解码输出 buffer 的引用计数没有清零，导致这些 buffer 无法被最终释放。

【问题解决】

由于泄露类型为 dma buffer，因此可通过系统 dma_buf 状态信息判断。如果拷机过程中，挂载到 VPU 的 dma buffer 个数持续增加无法释放，则最终确认该问题。

```
cat /d/dma_buf/bufinfo
```

具体的修改补丁请查阅 rk 补丁简报：<https://redmine.rock-chips.com/issues/423914>

3.4 播放器应用在播放过程中实时获取缩略图失败

【问题描述】

播放器应用在视频播放过程中，使用 MediaMetadataRetriever 获取缩略图失败。日志提示：

```
MediaMetadataRetrieverJNI( 1574): getFrameAtTime: videoFrame is a NULL pointer
```

【问题原因】

目前播放器框架中，为了省带宽，让后台的缩略图解析不影响前台视频播放，默认在视频播放过程中禁用了缩略图功能，因此出现播放过程中无法缩略图。

【问题解决】

Android 12 以上的版本中提供属性开关，用于打开这一限制。如果客户应用开发需要，可以将以下属性加到系统 prop 文件中去。Android 12 以下的版本请提 redmine 找对应工程师更新库。

```
setprop rt_retriever_enable 1
```

3.5 RK356X 录屏或录像编码绿屏，日志提示 RGA 报错

【问题描述】

RK356X 设备录屏或录像编码绿屏，日志如下 RGA 报错，提示不支持 4G 地址以外的 buffer。

```
E/rga_mm ( 0): RGA_MMU unsupported Memory larger than 4G!  
E/rga_mm ( 0): scheduler core[4] unsupported mm_flag[0x0]!  
E/rga_mm ( 0): rga_mm_map_buffer map dma_buf error!  
E/rga_mm ( 0): job buffer map failed!  
E/rga_mm ( 0): src channel map job buffer failed!  
E/rga_mm ( 0): failed to map buffer
```

【问题原因】

1. RK356X 使用硬件 RGA2 进行图像处理，RGA2 硬件设计只能处理 32 位的地址空间，因此送 RGA 处理的 buffer 需要分配 4G 以内的空间。
2. RK356X 编码器只能处理对齐过的 buffer，因此 Codec 组件依赖 RGA 做编码前处理，将编码输入 buffer 对齐之后再送硬件 VPU 编码器。

报错的原因是 Codec 组件 RGA 前处理的 src 或 dst buffer 地址空间超过 4G，而 Android 框架 Dma buffer 申请最终都通过 Gralloc，因此需要让该场景下 Gralloc 分配 4G 以内的 buffer。

【问题解决】

RGA 前处理的 dst buffer，最终送入硬件编码器，修改使用 GraphicBufferAllocator，将标识需要分配 4G buffer 的 GRALLOC_USAGE_WITHIN_4G flag 传递给 Gralloc 即可。

Codec 组件 RGA 前处理的 src buffer 来自 surface，场景属于 BufferQueue 生产者消费者模型，Codec 组件为消费者端而 Surface 为生产者端。

Surface buffer 由生产者端控制分配，但消费者端设置的 usage 会最终应用于生产者端的 Surface buffer 分配。在 Codec 组件中会使用 usage GRALLOC_USAGE_HW_VIDEO_ENCODER 用于标识当前为硬件编码器场景。

因此使用此 usage 做区分，buffer 分配时 usage 带 GRALLOC_USAGE_HW_VIDEO_ENCODER 且为 RK356X 平台，则控制 Gralloc 为该场景分配 4G buffer。

具体的修改补丁请查阅 rk 补丁简报：<https://redmine.rock-chips.com/issues/425094>

3.6 视频播放器应用剔除自定义音视频格式支持

【问题描述】

本地应用播放器或其他 MediaPlayer 播放器需要剔除一些音视频格式支持。如国外 VP9 视频格式有版权，需要剔除 VP9 视频格式支持，而国内机器则不影响。

【问题解决】

提供配置文件方式支持客户剔除自定义的音视频格式，如需要在 RK3588 设备上剔除 VC1\VP9 视频格式以及 AAC\MP3 音频格式支持，SDK 按如下补丁修改 (path: device/rockchip/common)：

```
diff --git a/rt_video_config.xml b/rt_video_config.xml  
index 738ce594..c5380858 100755  
--- a/rt_video_config.xml  
+++ b/rt_video_config.xml  
@@ -206,6 +206,7 @@  
    <chip name="RK3566,RK3568">  
        <include name="mpeg1,mpeg2,mpeg4,vp8,h264_8k_10bit_hi  
        <include name="vp9_4k_10bit"/>  
+        <forbid name="vc1,vp9">^M  
    </chip>
```

```
diff --git a/rt_audio_config.xml b/rt_audio_config.xml
index 1f64d98e..8970b6f4 100644
--- a/rt_audio_config.xml
+++ b/rt_audio_config.xml
@@ -40,4 +40,10 @@
     <format>MLP</format>
   </formats>
 </bitstream>
+ <forbid>
+   <formats>
+     <format>AAC</format>
+     <format>MP3</format>
+   </formats>
+ </forbid>
</sound>
```

手动修改验证: 更新 /system/etc 路径下rt_audio_config.xml/rt_video_config.xml 配置文件, 重新开始播放即可确认生效。

其他可禁用的音视频格式:

Video 可选项:

mpeg1、mpeg2、h263、mpeg4、wmv1、wmv2、wmv3、h264、vp8、vp9、hevc、vc1、avs、avs+、avs2、flv1、av1

Audio 可选项:

AAC、APE、MP3、WMALOSSLESS、WMAPRO、WMAV1、WMAV2、ADPCM_IMA_QT、VORBIS、PCM_S16LE、PCM_S24LE、FLAC、MP1、MP2、AMR_WB、AMR_NB