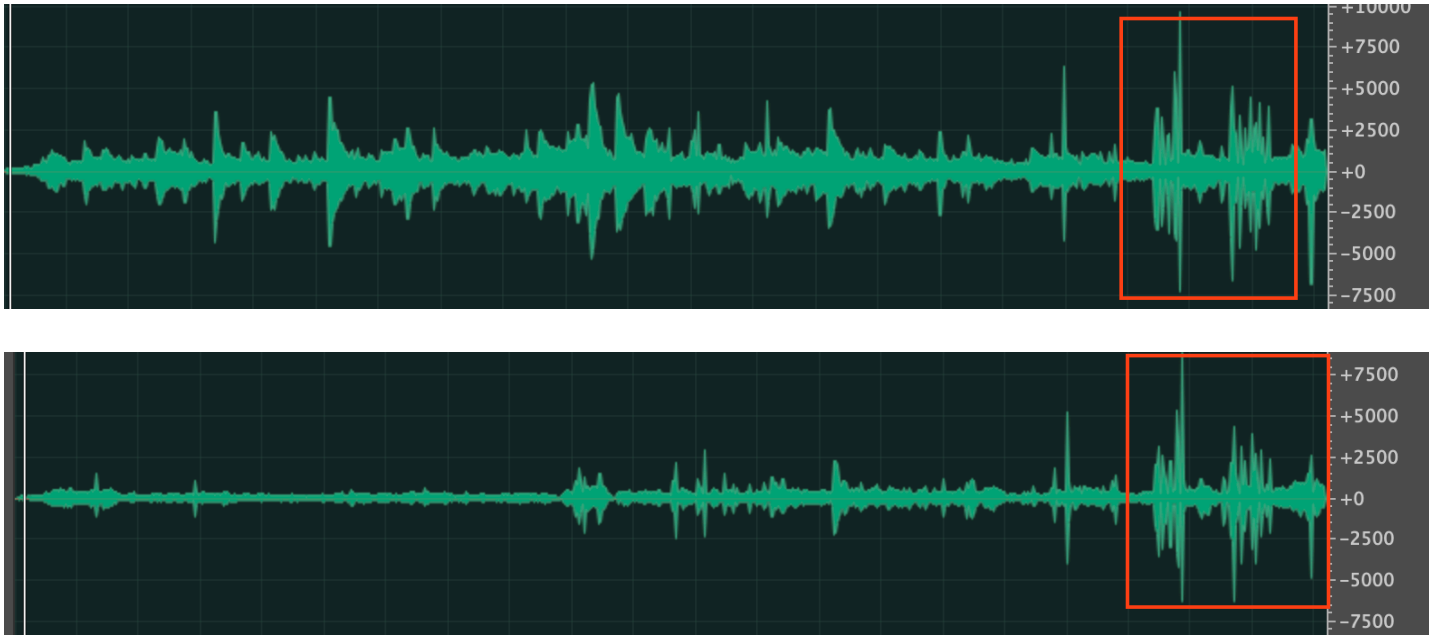
一次喜马拉雅APP录音卡顿问题解决过程

背景

- 喜马拉雅儿童版依赖录音二方库支持录音能力，喜马拉雅主APP也是
- 因此，若儿童版与主APP依赖同一版本录音二方库，在同一台设备上，表现也理应相同
- 然而，儿童版录制时出现了杂音，主APP表现正常

分析过程

- **录音有杂音**



上图为麦克风原始数据，下图为经过AEC回声消除后的数据，正常消除回声中的配乐后，音波趋于平整(如前半段所示)，然后接近末尾时，出现了不能消除的毛刺，试听有明显的顿挫感，判断原因有几：

1. 手机麦克风出现了问题，因为一手数据就存在了毛刺
2. AEC模块出现了问题，不能消除回声
3. 其他问题

手机自带录音APP录制正常，且前半段音频也正常，手机麦克风硬件出现问题概率小，排除原因1；

AEC消除需要1~2s计算收敛时间，因此这段时间内确实存在无法消除的回声，但只发生在录制开始时，而这里的录音是连续的，且在尾部，排除原因2

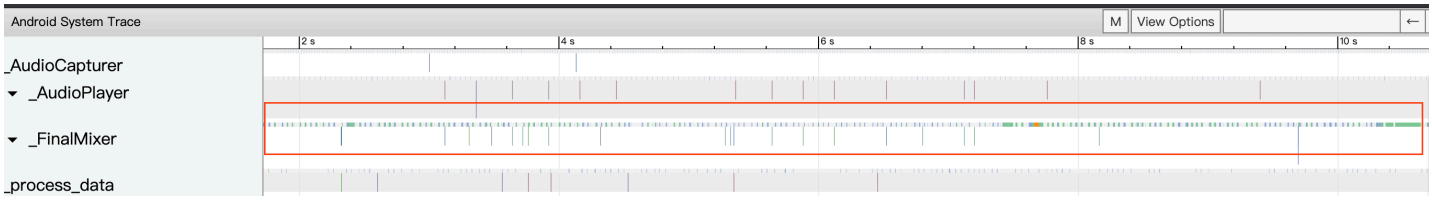
因此，麦克风、AEC运行都正常，怀疑其所在的线程在运行时受到了某种限制。

- **音频编码线程资源占用异常高**

```
  PID   TID USER     PR  NI CPU% S     VSS     RSS PCY Thread          Proc
  453  3570 logd     20   0   8% R  36472K  20364K unk logd.reader.per /system/bin/logd
21777 21777 u0_a301  12  -8   8% R 2239380K 337124K  ta ya.ting.android com.ximalaya.ting.android
21777 21947 u0_a301  10 -10   5% S 2239380K 337124K  ta RenderThread    com.ximalaya.ting.android
21777 22676 u0_a301  20   0   4% S 2239380K 337124K  ta _FinalMixer     com.ximalaya.ting.android
21558 21558 shell    20   0   3% R   9116K   3328K  fg top             top
21777 22677 u0_a301  20   0   1% S 2239380K 337124K  ta _process_data   com.ximalaya.ting.android
```

```
  PID   TID USER     PR  NI CPU% S     VSS     RSS PCY Thread          Proc
 6576 21580 u0_a300  20   0  12% R 2176264K 226800K  ta _FinalMixer     com.ximalaya.ting.kid
  453  3570 logd     20   0   8% S  34424K  18784K unk logd.reader.per /system/bin/logd
 6576  6576 u0_a300  12  -8   5% R 2176264K 226800K  ta malaya.ting.kid com.ximalaya.ting.kid
21558 21558 shell    20   0   3% R   9116K   3152K  fg top             top
 6576  6872 u0_a300  10 -10   2% S 2176264K 226800K  ta RenderThread    com.ximalaya.ting.kid
 6576 21581 u0_a300  20   0   0% S 2176264K 226800K  ta _process_data   com.ximalaya.ting.kid
 6576 21584 u0_a300  12  -8   0% S 2176264K 226800K  ta _AudioPlayer    com.ximalaya.ting.kid
```

上图是主app，下图是儿童版，“_FinalMixer”是音频编码线程名称， 通过"adb shell top -d 1 -m 10 -H"发现，儿童版资源占用较高，将近三倍于主app。



这是systrace下的表现，上图是主app，下图是儿童版。编码线程从间断的CPU占用，到连续的几乎打满CPU，差别之大也是奇怪。看了下_FinalMixer的实现，满足条件就唤醒线程从wait切换到running状态在doRealWork中执行音频编码，否则就置位pause，进入wait状态。

```
    @Override
    public void run() {
        while (!mIsStop){
            if (check2Pause() || mIsPause){
                onPause();
                synchronized (mLock){
                    try {
                        Log.v("XmRecorderr",getName() + " 进入等待...");
                        mLock.wait();
                        Log.v("XmRecorderr",getName() + " 结束等待，开始工作...");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }

            handleQueue();

            if (mIsStop) {
                break;
            }
            if (mIsPause) {
                continue;
            }

            onStart();

            try {
                doRealWork(); // 在子类执行具体的耗时任务，比如音频编码
            }catch (Exception e){
                e.printStackTrace();
                onError("doRealWork", e);
            }
        }
        onEnd();
    }
```

通过加日志，排除了死循环，但是doRealWork中除了音频编码又没有其他耗时操作，难道是编码耗时在同一个手机的两个app上不一样?

- **音频编码器编码耗时异常**(测试方法见“其他”)

对同样的原始音频进行编码，使用户相同的输出参数(双声道、44100Hz)，抱着试一试的心态，将其分别作为runnable放到两个app的后台运行：

```
主APP：
cost = 7339 ms
cost = 7407 ms
cost = 7343 ms

儿童版：
cost = 24366 ms
cost = 24219 ms
cost = 24944 ms
```

编码这段原始只有20s的音频，在主APP中耗时7s左右，在儿童版中24s，后者将近三倍多(联系到之前线程占用比类似)。

- **ijk动态库方法耗时异常**

我们知道了编码器耗时多，但还不知道为什么多，多在了什么地方。 通过Simpleperf来了解一下：

1.获得样本数据"perf.data"：(6576是儿童版app的进程号)

```
./simpleperf record -p 6576 --duration 30
```

1. 从"perf.data"解析执行时间最长的共享库为“libijkffmpeg-armeabi-v7a.so”：

```
./simpleperf report --sort dso

Cmdline: /data/data/com.ximalaya.ting.kid/simpleperf record -p 6576 --duration 30
Arch: arm64
Event: cpu-cycles (type 0, config 0)
Samples: 120167
Event count: 58438296274

Overhead  Shared Object
82.31%    /data/app/com.ximalaya.ting.kid-2/lib/arm/libijkffmpeg-armeabi-v7a.so
4.38%     /system/lib/libart.so
4.22%     /system/framework/arm/boot-framework.oat
1.79%     /system/lib/libm.so
1.53%     /dev/ashmem/dalvik-jit-code-cache (deleted)
1.50%     [kernel.kallsyms]
1.31%     /system/framework/arm/boot.oat
........
```

1. 从最长的共享库查找执行时间最长的函数：

```
./simpleperf report --dsos /data/app/com.ximalaya.ting.kid-2/lib/arm/libijkffmpeg-armeabi-v7a.so --sort symbol

Cmdline: /data/data/com.ximalaya.ting.kid/simpleperf record -p 6576 --duration 30
Arch: arm64
Event: cpu-cycles (type 0, config 0)
Samples: 95849
Event count: 48098065822

Overhead  Symbol
28.09%    __addsf3
15.45%    __aeabi_fmul
4.57%     __eqsf2
4.38%     __aeabi_cfcmpeq
2.52%     __aeabi_l2f
2.08%     __floatsisf
1.97%     __aeabi_f2iz
1.71%     __muldf3
1.65%     __aeabi_dadd
1.20%     __aeabi_cfrcmple
0.90%     __aeabi_fcmpgt
0.72%     __aeabi_fcmpge
0.71%     __aeabi_fcmplt
0.56%     libijkffmpeg-armeabi-v7a.so[+2b8ec]
0.50%     __aeabi_fcmpeq
0.43%     libijkffmpeg-armeabi-v7a.so[+29568]
0.42%     libijkffmpeg-armeabi-v7a.so[+29520]
....
```

可以看到，占用时间较长的函数是“`__addsf3、__aeabi_fmul、__eqsf2`”等。 这些函数并不常见，查阅得知为浮点计算相关.

因此，得出在音频编码中ijk动态库耗时最长；在ijk动态库执行中，浮点相关的软计算耗时最多。

联想到动态库so在编译时，输出的几种CPU类型： armeabi、armeabi-v7a、armeabi-v8a、x86、x86_64等。

armeabi 仅支持软件浮点计算；armeabi-v7a 开始支持硬件浮点计算(参考：Android ABI)

- **验证两APP中的ijk动态库ABI类型**：

```
儿童版：
arm-linux-androideabi-readelf -A libijkffmpeg-armeabi.so
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "5TE"
  Tag_CPU_arch: v5TE
  Tag_ARM_ISA_use: Yes
  Tag_THUMB_ISA_use: Thumb-1
  Tag_FP_arch: VFPv2
....

主APP：
arm-linux-androideabi-readelf -A libijkffmpeg-armeabi-v7a-app.so
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "ARM v7"
  Tag_CPU_arch: v7
  Tag_CPU_arch_profile: Application
  Tag_ARM_ISA_use: Yes
  Tag_THUMB_ISA_use: Thumb-2
  Tag_FP_arch: VFPv3
```

解决方法

将儿童版ijk so库"armeabi"类型换成“armeabi-v7a”类型即可。

经验教训

- 移动端动态库abi类型优先使用armeabi-v7a及以上，而不是armeabi
- 使用“arm-linux-androideabi-readelf -A libxxx.so”判断动态库类型，而不是“file”
    - 用“file xxx.so”不准确，armeabi、armeabi-v7a的结果一样，无法区分

其他

- 音频编码测试用例

```
 AacEncoder aacEncoder = new AacEncoder();
String pcm = "/sdcard/_bgm_out.pcm";
DataInputStream dis = new DataInputStream(new BufferedInputStream(new FileIn

int buffer_size_in_short = 2048;
short[] buffer = new short[buffer_size_in_short];

String outAac = "/sdcard/123.aac";
Log.d("todo", "outAac = " + outAac);
int ret = aacEncoder.Init(outAac, Constants.sample_rate_in_Hz,
        Constants.nb_channels_single, Constants.sample_rate_in_Hz, Constants
Utils.checkFailThrow(ret, "Init");
long start = System.currentTimeMillis();
while (dis.available() > 0){
    for (int i = 0; i < buffer_size_in_short; i++) {
        buffer[i] = dis.readShort();
    }
    ret = aacEncoder.EncodeAudioFrame(buffer, buffer_size_in_short);
    Utils.checkFailThrow(ret, "EncodeAudioFrame");
}
Log.d("todo", "cost = " + (System.currentTimeMillis() - start) + " ms");
aacEncoder.FlushAndCloseFile();
System.out.println("FlushAndCloseFile end.");
```

- SimplePerf的主要用法
  - 事件数摘要：
    - ./simpleperf stat -p 进程号 --duration 检测进程的持续时间(秒)
  - 记录样本：
    - ./simpleperf record -p 进程号 -o 输出文件(默认perf.data) --duration 监测进程的持续时间(秒)
    - 偶现会失败报错“failed to open record file 'perf.data': Too many open files”，拔掉usb线重连就行
  - 根据样本，分析报告：
    - ./simpleperf report --dsos 选定动态共享对象(so库) -f 记录文件(默认perf.data) --sort 用于排序和打印报告的键 -n
  - SimplePerf命令默认在host上，需要push到Android设备的可执行目录
    - 命令路径：~/ndk-bundle/simpleperf/*
- 官方Simpleperf
- Simpleper Android的CPU分析，性能优化利器