Android 深入浅出之 Audio Policy

一 目的

上回我们说了 AudioFlinger(AF)，总感觉代码里边有好多东西没说清楚，心里发毛。就看了看 AF 的流程，我们敢说自己深入了解了 Android 系统吗？AudioPolicyService（APS）是个什么东西？为什么要有它的存在？下层的 Audio HAL 层又是怎么结合到 Android 中来的？更有甚者，问个实在问题：插入耳机后，声音又怎么从最开始的外放变成从耳机输出了？调节音量的时候到底是调节 Music 的还是调节来电音量呢？这些东西，我们在 AF 的流程中统统都没讲到。但是这些他们又是至关重要的。从我个人理解来看，策略（Policy）比流程更复杂和难懂。
当然，遵循我们的传统分析习惯，得有一个切入点，否则我们都不知道从何入手了。
这里的切入点将是：

- AF 和 APS 系统第一次起来后，到底干了什么。
- 检测到耳机插入事件后，AF 和 APS 的处理。

大家跟着我一步步来看，很快就发现，啊哈，APS 也不是那么难嘛。
另外，这次代码分析的格式将参考《Linux 内核情景分析》的样子，函数调用的解析将采用深度优先的办法，即先解释所调用的函数，然后再出来继续讲。

```
我曾经数度放弃分析 APS，关键原因是我没找到切入点，只知道代码从头看到尾!
```

二 AF 和 APS 的诞生

这个东西，已经说得太多了。在 framework/base/media/MediaServer/Main_MediaServer 中。
我们看看。

```
int main(int argc, char** argv)
{
    sp<ProcessState> proc(ProcessState::self());
    sp<IServiceManager> sm = defaultServiceManager();
    //先创建 AF
    AudioFlinger::instantiate();
    //再创建 APS
    AudioPolicyService::instantiate();

    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
}
```

2.1 new AudioFlinger

前面说过，instantiate 内部会实例化一个对象，那直接看 AF 的构造函数。

```
AudioFlinger::AudioFlinger()
    : BnAudioFlinger(),//基类构造函数
        mAudioHardware(0), mMasterVolume(1.0f), mMasterMute(false), mNextThreadId(0)
{
  注意 mAudioHardware 和 mNextThreadId
   mHardwareStatus = AUDIO_HW_IDLE;
//创建 audio 的 HAL 代表
    mAudioHardware = AudioHardwareInterface::create();
    mHardwareStatus = AUDIO_HW_INIT;
    //下面这些不至于会使用 APS 吧? APS 还没创建呢!
    if (mAudioHardware->initCheck() == NO_ERROR) {
        setMode(AudioSystem::MODE_NORMAL);
        setMasterVolume(1.0f);
        setMasterMute(false);
    }
}
```

感觉上，AF 的构造函数就是创建了一个最重要的 AudioHardWare 的 HAL 代表。
其他好像是没干什么策略上的事情。
不过：AF 创建了一个 AudioHardware 的 HAL 对象。注意整个系统就这一个 AudioHardware 了。
也就是说，不管是线控耳机，蓝牙耳机，麦克，外放等等，最后都会由这一个 HAL 统一管理。
再看 APS 吧。

2.2 new AudioPolicyService

```
AudioPolicyService::AudioPolicyService()
    : BnAudioPolicyService() , mpPolicyManager(NULL)
{
  //  mpPolicyManager?策略管理器?可能很重要
  char value[PROPERTY_VALUE_MAX];

    // TonePlayback? 播放铃声的? 为什么放在这里? 以后来看看
    mTonePlaybackThread = new AudioCommandThread(String8(""));
    // Audio Command? 音频命令? 看到 Command, 我就想到设计模式中的 Command 模式了
    //Android 尤其是 MediaPlayerService 中大量使用了这种模式。
    mAudioCommandThread = new AudioCommandThread(String8("ApmCommandThread"));
```

```
#if (defined GENERIC_AUDIO) || (defined AUDIO_POLICY_TEST)
 //注意 AudioPolicyManagerBase 的构造函数，把 this 传进去了。
    mpPolicyManager = new AudioPolicyManagerBase(this);
    //先假设我们使用 Generic 的 Audio 设备吧。
#else
    ...

#endif
    // 根据系统属性来判断摄像机是否强制使用声音。这个...为什么会放在这里？
    //手机带摄像机好像刚出来的时候，为了防止偷拍，强制按快门的时候必须发出声音
    //就是这个目的吧？
    property_get("ro.camera.sound.forced", value, "0");
    mpPolicyManager->setSystemProperty("ro.camera.sound.forced", value);
}
```

so easy!，不至于吧？我们不应该放过任何一个疑问！这么多疑问，先看哪个呢？这里分析的是 Audio Policy，而构造函数中又创建了一个 AudioPolicyManagerBase，而且不同厂商还可以实现自己的 AudioPolicyManager，看来这个对于音频策略有至关重要的作用了。
不得不说的是，Android 代码中的这些命名在关键地方上还是比较慎重和准确的。
另外，AudioPolicyManagerBase 的构造函数可是把 APS 传进去了，看来又会有一些回调靠 APS 了。真绕。

2.3 AudioPolicyManagerBase

代码位置在 framework/base/libs/audioflinger/AudioPolicyManagerBase.cpp 中

```
AudioPolicyManagerBase::AudioPolicyManagerBase(AudioPolicyClientInterface
*clientInterface)
    :
 mPhoneState(AudioSystem::MODE_NORMAL), ---->这里有电话的状态？
 mRingerMode(0),
 mMusicStopTime(0),
 mLimitRingtoneVolume(false)
{
```

[--->mPhoneState(AudioSystem::MODE_NORMAL)]

AudioSystem 其实是窥视 Android 如何管理音频系统的好地方。位置在 framework/base/include/media/AudioSystem.h 中，定义了大量的枚举之类的东西来表达 Google 对音频系统的看法。我们只能见招拆招了。

下面是 audio_mode 的定义。这里要注意一个地方：
这些定义都和 SDK 中的 JAVA 层定义类似。实际上应该说先有 C++层的定义，然后再反映到 JAVA 层中。但是 C++层的定义一般没有解释说明，而 SDK 中有。所以我们不能不面对的一个痛苦现实就是：常常需要参考 SDK 的说明才能搞明白到底是什么。
关于 C++的 AudioSystem 这块，SDK 的说明在 AudioManager 中。

```
enum audio_mode {
//解释参考 SDK 说明，以下不再说明
        MODE_INVALID = -2, //无效 mode
        MODE_CURRENT = -1,//当前 mode，和音频设备的切换（路由）有关
        MODE_NORMAL = 0,//正常 mode，没有电话和铃声
        MODE_RINGTONE,//收到来电信号了，此时会有铃声
        MODE_IN_CALL,//电话 mode，这里表示已经建立通话了
        NUM_MODES  // Android 大量采用这种技巧来表示枚举结束了。
    };
```

好，继续：

```
...
mPhoneState(AudioSystem::MODE_NORMAL), ---->这里有电话的状态?
 mRingerMode(0),
 mMusicStopTime(0),
 mLimitRingtoneVolume(false)
{
  mpClientInterface = clientInterface;//BT，保存 APS 对象。
//forceUse? 这是个什么玩意儿?
    for (int i = 0; i < AudioSystem::NUM_FORCE_USE; i++) {
        mForceUse[i] = AudioSystem::FORCE_NONE;
    }
```

[---->AudioSystem::FORCE_NONE 和 AudioSystem::NUM_FORCE_USE]
注意，这里有两个枚举，太无耻了。先看看 FORCE_NONE 这个

```
enum forced_config {强制_配置，看名字好像是强制使用设备吧，比如外放，耳机，蓝牙等
        FORCE_NONE,
        FORCE_SPEAKER,
        FORCE_HEADPHONES,
        FORCE_BT_SCO,
```

```
        FORCE_BT_A2DP,
        FORCE_WIRED_ACCESSORY,
        FORCE_BT_CAR_DOCK,
        FORCE_BT_DESK_DOCK,
        NUM_FORCE_CONFIG,
        FORCE_DEFAULT = FORCE_NONE //这个，太无聊了。
    };
```

再看看 AudioSystem::NUM_FORCE_USE 这个

```
enum force_use {
        FOR_COMMUNICATION,//这里是 for_xxx，不是 force_xxx。
        FOR_MEDIA,
        FOR_RECORD,
        FOR_DOCK,
        NUM_FORCE_USE
    };
```

不懂，两个都不懂。为何？能猜出来什么吗？也不行。因为我们没找到合适的场景！那好吧，我们去 SDK 找找。恩

我看到 AudioManager 这个函数 setSpeakerphoneOn (boolean on)。好吧，我

这么调用

```
setSpeakerphoneOn(true)，看看实现。
这次我没再浪费时间了，我用一个新的工具 coolfind，把搜索 framework 目录，寻找*.java 文件，匹配字符串 setSpeakerphone。终于，我在
framework/base/media/java/android/media/AudioService.java 中找到了。
public void setSpeakerphoneOn(boolean on){
        if (!checkAudioSettingsPermission("setSpeakerphoneOn()")) {
            return;
        }
        if (on) {
//看到这里，是不是明白十之八九了?下面这个调用是:
//强制通话使用 speaker! 原来是这么个意思!
            AudioSystem.setForceUse(AudioSystem.FOR_COMMUNICATION,
             AudioSystem.FORCE_SPEAKER);
            mForcedUseForComm = AudioSystem.FORCE_SPEAKER;
```

```
        } else {
            AudioSystem.setForceUse(AudioSystem.FOR_COMMUNICATION,
            AudioSystem.FORCE_NONE);
            mForcedUseForComm = AudioSystem.FORCE_NONE;
        }
    }
```

好了，说点题外话，既然 Android 源码都放开给我们了，有什么理由我们不去多搜搜呢？上网 google 也是搜，查源代码也是一样吗。不过我们要有目的：就是找到一个合适的使用场景。force_use 和 force_config 就不用我再解释了吧？

[--->AudioPolicyManagerBase::AudioPolicyManagerBase]

```
...
//下面这个意思就是把几种 for_use 的情况使用的设备全部置为 NONE。
//比如设置 FOR_MEDIA 的场景，使用的设备就是 FORCE_NONE
for (int i = 0; i < AudioSystem::NUM_FORCE_USE; i++) {
        mForceUse[i] = AudioSystem::FORCE_NONE;
    }

  // 目前可以的输出设备，耳机和外放
    mAvailableOutputDevices = AudioSystem::DEVICE_OUT_EARPIECE |
                        AudioSystem::DEVICE_OUT_SPEAKER;
 //目前可用的输入设备，内置 MIC
    mAvailableInputDevices = AudioSystem::DEVICE_IN_BUILTIN_MIC;
```

又得来看看 AudioSystem 是怎么定义输入输出设备的了。

[--->mAvailableOutputDevices = AudioSystem::DEVICE_OUT_EARPIECE]

```
enum audio_devices {
        // output devices
        DEVICE_OUT_EARPIECE = 0x1,
        DEVICE_OUT_SPEAKER = 0x2,
        DEVICE_OUT_WIRED_HEADSET = 0x4,
        DEVICE_OUT_WIRED_HEADPHONE = 0x8,
        DEVICE_OUT_BLUETOOTH_SCO = 0x10,
        DEVICE_OUT_BLUETOOTH_SCO_HEADSET = 0x20,
        DEVICE_OUT_BLUETOOTH_SCO_CARKIT = 0x40,
```

```
        DEVICE_OUT_BLUETOOTH_A2DP = 0x80,
        DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES = 0x100,
        DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER = 0x200,
        DEVICE_OUT_AUX_DIGITAL = 0x400,
        DEVICE_OUT_DEFAULT = 0x8000,
        DEVICE_OUT_ALL = (DEVICE_OUT_EARPIECE | DEVICE_OUT_SPEAKER |
 DEVICE_OUT_WIRED_HEADSET | DEVICE_OUT_WIRED_HEADPHONE | DEVICE_OUT_BLUETOOTH_SCO |
DEVICE_OUT_BLUETOOTH_SCO_HEADSET |DEVICE_OUT_BLUETOOTH_SCO_CARKIT |
 DEVICE_OUT_BLUETOOTH_A2DP | DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES |
 DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER | DEVICE_OUT_AUX_DIGITAL | DEVICE_OUT_DEFAULT),
   DEVICE_OUT_ALL_A2DP = (DEVICE_OUT_BLUETOOTH_A2DP |
  DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES |DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER),


        // input devices
        DEVICE_IN_COMMUNICATION = 0x10000,
        DEVICE_IN_AMBIENT = 0x20000,
        DEVICE_IN_BUILTIN_MIC = 0x40000,
        DEVICE_IN_BLUETOOTH_SCO_HEADSET = 0x80000,
        DEVICE_IN_WIRED_HEADSET = 0x100000,
        DEVICE_IN_AUX_DIGITAL = 0x200000,
        DEVICE_IN_VOICE_CALL = 0x400000,
        DEVICE_IN_BACK_MIC = 0x800000,
        DEVICE_IN_DEFAULT = 0x80000000,
        DEVICE_IN_ALL = (DEVICE_IN_COMMUNICATION | DEVICE_IN_AMBIENT |
  DEVICE_IN_BUILTIN_MIC |DEVICE_IN_BLUETOOTH_SCO_HEADSET | DEVICE_IN_WIRED_HEADSET |
  DEVICE_IN_AUX_DIGITAL | DEVICE_IN_VOICE_CALL | DEVICE_IN_BACK_MIC |
  DEVICE_IN_DEFAULT)
    };
```

一些比较容易眼花的东西我标成红色的了。这么多东西，不过没什么我们不明白的了。得嘞，继续走。

[--->AudioPolicyManagerBase::AudioPolicyManagerBase]

```
// 目前可以的输出设备，又有耳机又有外放，配置很强悍啊。
//注意这里是 OR 操作符，最终 mAvailableOutputDevices = 0X3
```

```
    mAvailableOutputDevices = AudioSystem::DEVICE_OUT_EARPIECE |
                        AudioSystem::DEVICE_OUT_SPEAKER;
 //目前可用的输入设备，内置 MIC，mAvailableInputDevices 为 0x4000，不过我们不关注 input
 mAvailableInputDevices = AudioSystem::DEVICE_IN_BUILTIN_MIC;


...
  下面东西就很少了，我们一气呵成。
//创建一个 AudioOutputDescriptor，并设置它的 device 为外设 0x2
    AudioOutputDescriptor *outputDesc = new AudioOutputDescriptor();
    outputDesc->mDevice = (uint32_t)AudioSystem::DEVICE_OUT_SPEAKER;


    //调用 APS 的 openOutput，得到一个 mHardwareOutput 东东。这是个 int 型
    //不过保不准是一个指针也不一定喔。
    //而且，下面的参数都是指针类型（flags 除外），难道？有人会改 value 吗？
    mHardwareOutput = mpClientInterface->openOutput(&outputDesc->mDevice,
                                    &outputDesc->mSamplingRate,
                                    &outputDesc->mFormat,
                                    &outputDesc->mChannels,
                                    &outputDesc->mLatency,
                                    outputDesc->mFlags);


  //这个...估计是把 int 和指针加入到一个 map 了，方便管理。
    addOutput(mHardwareOutput, outputDesc);
    //不知道干嘛，待会看。
    setOutputDevice(mHardwareOutput, (uint32_t)AudioSystem::DEVICE_OUT_SPEAKER, true);
    //不知道干嘛，待会看。
    updateDeviceForStrategy();
```

好了，上面还有一系列函数，等着我们调用呢。我们一个一个看。
提前说一下，这块可是 AudioManagerBase 的核心喔。
[---->AudioOutputDescriptor *outputDesc = new AudioOutputDescriptor()]
AudioOutputDescriptor 是个什么？我不是神，我也得看注释。

```
// descriptor for audio outputs. Used to maintain current configuration of each opened audio output
```

```
// and keep track of the usage of this output by each audio stream type.
```

明白了么？大概意思就是它，是这么一个东西：

- 描述 audio 输出的，可以用来保存一些配置信息。
- 跟踪音频 stream 类型使用这个 output 的一些情况。

没明白吧？以后碰到场景就明白了。

它的构造函数干了如下勾当：

```
    AudioPolicyManagerBase::AudioOutputDescriptor::AudioOutputDescriptor()

        : mId(0), mSamplingRate(0), mFormat(0), mChannels(0), mLatency(0),

        mFlags((AudioSystem::output_flags)0), mDevice(0), mOutput1(0), mOutput2(0)

{}

//很好，统统都置零了。上面这些东西不用我解释了吧？命名规则也可以看出来。
```

OK，go on.

[--->mHardwareOutput = mpClientInterface->openOutput()]:

这里调用的是 APS 的 openOutput，看看去：

[--->AudioPolicyService::openOutput]

```
audio_io_handle_t AudioPolicyService::openOutput(uint32_t *pDevices,

                                uint32_t *pSamplingRate,

                                uint32_t *pFormat,

                                uint32_t *pChannels,

                                uint32_t *pLatencyMs,

                                AudioSystem::output_flags flags)

{

    sp<IAudioFlinger> af = AudioSystem::get_audio_flinger();

    //娘希匹，搞到 AF 去了

    return af->openOutput(pDevices, pSamplingRate, (uint32_t *)pFormat, pChannels,

            pLatencyMs, flags);

}
```

[----->AudioFlinger::openOutput]

```
int AudioFlinger::openOutput(uint32_t *pDevices,

                                uint32_t *pSamplingRate,

                                uint32_t *pFormat,

                                uint32_t *pChannels,

                                uint32_t *pLatencyMs,

                                uint32_t flags)
```

```
{
//我们思考下传进来的值吧
//*pDevices=0x2,代表外放
//其他都是 0。 嘿嘿，有了值，这不就知道下面该怎么走了吗?
    status_t status;
    PlaybackThread *thread = NULL;
    mHardwareStatus = AUDIO_HW_OUTPUT_OPEN;
    uint32_t samplingRate = pSamplingRate ? *pSamplingRate : 0;
    uint32_t format = pFormat ? *pFormat : 0;
    uint32_t channels = pChannels ? *pChannels : 0;
    uint32_t latency = pLatencyMs ? *pLatencyMs : 0;

     Mutex::Autolock _l(mLock);
    //HAL 对象得到一个 AudioStreamOut，传进去的值会改吗?
    AudioStreamOut *output = mAudioHardware->openOutputStream(*pDevices,
                                                              (int *)&format,
                                                              &channels,
                                                              &samplingRate,
                                                              &status);
      mHardwareStatus = AUDIO_HW_IDLE;

    if (output != 0) {
    //走哪个分支? 我把答案告诉大家吧。
    //刚才那个 mAudioHardware->openOutputStream 确实会更改指针对应的 value。
    //当然，我们说了，AF 使用的是 GENERIC 的 Audio 硬件。大家有兴趣可以去看看它的实现。
    //我待会再贴出它的内容。反正到这里。
    //那几个值变成: format 为 PCM_16_BIT，channels 为 2，samplingRate 为 44100
    //这样的话，那只能走 else 分支了。
        if ((flags & AudioSystem::OUTPUT_FLAG_DIRECT) ||
            (format != AudioSystem::PCM_16_BIT) ||
            (channels != AudioSystem::CHANNEL_OUT_STEREO)) {
            thread = new DirectOutputThread(this, output, ++mNextThreadId);
```

```
            } else {
//还记得前两节分析的同学，看到这里是不是明白了？恩，原来
//open 一个 Output，就会在 AF 中创建一个混音线程。设计得真好。
//想象下，所有设置为外放的程序，它的输出都是这个外放 stream 混音线程来工作
//所有设置为耳机的程序，它的输出都是这个耳机 stream 混音线程来完成。
//为什么对 stream 特加强调呢，没看见
//我们调用的是 mAudioHardware->openOutputStream(0x2,,,)嘛。返回的
//是一个 AudioStreamOut，可不是设备喔。Android 把这些个东西都交给 HAL 层去实现了。
//不用自己来管理系统上有什么耳机，外设，蓝牙真实设备之类的东东，它反正用 AudioStreamOut
来表示它想要的就可以了。例如Generic的Audio Hal只支持一个OutputStream。--> only my opinion
            thread = new MixerThread(this, output, ++mNextThreadId);
  }
//好了，又多得了一个线程，
        mPlaybackThreads.add(mNextThreadId, thread);
        if (pSamplingRate) *pSamplingRate = samplingRate;
        if (pFormat) *pFormat = format;
        if (pChannels) *pChannels = channels;
        if (pLatencyMs) *pLatencyMs = thread->latency();
//从这里返回的是混音线程的索引。
        return mNextThreadId;
    }
    return 0;//如果没创建成功线程，则返回零。
}
```

好，我们回到 AudioManagerBase 中。

[--->AudioPolicyManagerBase::AudioPolicyManagerBase]

```
mHardwareOutput = mpClientInterface->openOutput(&outputDesc->mDevice,
                                    &outputDesc->mSamplingRate,
                                    &outputDesc->mFormat,
                                    &outputDesc->mChannels,
                                    &outputDesc->mLatency,
                                    outputDesc->mFlags);
//上面实际就返回一个线程 index。我有点疑惑，难道 APS 就只这么一个实际是线程 index 的东西
就就行了吗？虽然它把这个 index 当成 hardware 的标识了。
```

```
  //这个...估计是把 int 和指针加入到一个 map 了，方便管理。不看了。
    addOutput(mHardwareOutput, outputDesc);
    //不知道干嘛，待会看。
setOutputDevice(mHardwareOutput, (uint32_t)AudioSystem::DEVICE_OUT_SPEAKER, true);
```

[--->setOutputDevice(mHardwareOutput,...)]

这个函数，很重要！另外，再传点技巧。不要老在 source insight 中后退后退了，直接找到 window 菜单，里边列出了最近打开的文件，找到我们的 AudioManagerBase.cpp，不就行了吗？

```
void AudioPolicyManagerBase::setOutputDevice(audio_io_handle_t output, uint32_t device,
bool force, int delayMs)
{
    //注意我们的参数:
    // output = 1,
    //device 为 AudioSystem::DEVICE_OUT_SPEAKER
    // force 为 true, delayMs 用默认值 0

    //map 吧？刚才通过 addOutput 已经加进去了
    AudioOutputDescriptor *outputDesc = mOutputs.valueFor(output);

    if (outputDesc->isDuplicated()) {
        setOutputDevice(outputDesc->mOutput1->mId, device, force, delayMs);
        setOutputDevice(outputDesc->mOutput2->mId, device, force, delayMs);
        return;
    }
    //还记得 addOutput 前设置的 device 吗？对了，为 0X3，外放|耳机
    uint32_t prevDevice = (uint32_t)outputDesc->device();
    现在设置的是外设，
    if ((device == 0 || device == prevDevice) && !force) {
        return;
    }
//喔，设置这个 outputDesc 为外放
    outputDesc->mDevice = device;
    popCount 为 2，因为 device=0x2=0010
//另外，我对下面这个 output== mHardwareOutput 尤其感兴趣。还记得我们刚才的疑问吗？
```

```
// mHardwareOutput 实际上是 AF 返回的一个线程索引，那 AMB 怎么根据这样一个东西来
//管理所有的线程呢？果然，这里就比较了 output 是不是等于最初创建的线程索引
//这就表明。虽然只有这么一个 mHardwareOutput，但实际上还是能够操作其他 output 的！
    if (output == mHardwareOutput && AudioSystem::popCount(device) == 2) {
        setStrategyMute(STRATEGY_MEDIA, true, output);
         usleep(outputDesc->mLatency*2*1000);
    }

    // 晕，又冒出来一个 AudioParameter，不过意思却很明白
    //说我们要设置路由，新的输出设备为外放
    //等我们以后讲由外放切换到耳机，再来看这个问题。
    AudioParameter param = AudioParameter();
    param.addInt(String8(AudioParameter::keyRouting), (int)device);
    mpClientInterface->setParameters(mHardwareOutput, param.toString(), delayMs);
    // update stream volumes according to new device
    applyStreamVolumes(output, device, delayMs);

    // if changing from a combined headset + speaker route, unmute media streams
    if (output == mHardwareOutput && AudioSystem::popCount(prevDevice) == 2) {
      //这里说，把 media 的音量置为 0。以后再说。
      setStrategyMute(STRATEGY_MEDIA, false, output, delayMs);
    }
}
```

好了，返回了。

```
setOutputDevice(mHardwareOutput, (uint32_t)AudioSystem::DEVICE_OUT_SPEAKER, true);
这个调研，更新了 mHardwareOutput 对应的输出路由设备，而且还发了一个命令给 APS，说你给我
更新对应混音线程的输出路由设备。
```

[--->AudioPolicyManagerBase::AudioPolicyManagerBase]

```
    .....
    addOutput(mHardwareOutput, outputDesc);
        setOutputDevice(mHardwareOutput, (uint32_t)AudioSystem::DEVICE_OUT_SPEAKER,
        true);
```

```
  //只剩下最后一个函数了
  updateDeviceForStrategy();
```

[----->updateDeviceForStrategy()]

```
  void AudioPolicyManagerBase::updateDeviceForStrategy()
{
    for (int i = 0; i < NUM_STRATEGIES; i++) {
        mDeviceForStrategy[i] = getDeviceForStrategy((routing_strategy)i, false);
    }
}
```

晕，又出来一个枚举。我们看看
[---->for (int i = 0; i < NUM_STRATEGIES; i++)]
NUM_STRATEGIES 在 hardware/libhardware_legacy/include/hardware_legacy/AudioPolicyManagerBase.h 中定义。

```
enum routing_strategy {
//好像很好理解
            STRATEGY_MEDIA,
            STRATEGY_PHONE,//通话音吗?
            STRATEGY_SONIFICATION,//除了其他三个外的，可以是铃声，提醒声等。
            STRATEGY_DTMF,//好像是拨号音
            NUM_STRATEGIES
        };
```

这个，反正我在 SDK 上没找到对应说明，我们待到以后看看会不会柳暗花明呢？
[----->getDeviceForStrategy((routing_strategy)i, false)]
看这个函数名的意思是，为各种策略找到它对应的设备。

```
uint32_t AudioPolicyManagerBase::getDeviceForStrategy(routing_strategy strategy, bool fromCache)
{
  //  fromCache 为 false
//放眼望去，这个函数好像涉及到很对策略方面的事情。
//我们大概讲解下，至于系统为什么要这么做，问 Google 吧。
  uint32_t device = 0;

    switch (strategy) {
    case STRATEGY_DTMF:
```

```
        if (mPhoneState != AudioSystem::MODE_IN_CALL) {
            //如果在打电话过程中，你再按按键，则和 MEDIA 走一个设备
            device = getDeviceForStrategy(STRATEGY_MEDIA, false);
            break;
        }
        //注意这里没有 break，所以在其他 mode 下，DTMF 和 PHONE 用一个策略
    case STRATEGY_PHONE:
```

```
        //还得判断用户是不是强制使用了输出设备。
```

```
        switch (mForceUse[AudioSystem::FOR_COMMUNICATION]) {
        case AudioSystem::FORCE_BT_SCO:
            if (mPhoneState != AudioSystem::MODE_IN_CALL || strategy != STRATEGY_DTMF) {
                device = mAvailableOutputDevices &
                        AudioSystem::DEVICE_OUT_BLUETOOTH_SCO_CARKIT;
                if (device) break;
            }
            device = mAvailableOutputDevices &
                   AudioSystem::DEVICE_OUT_BLUETOOTH_SCO_HEADSET;
            if (device) break;
            device = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_BLUETOOTH_SCO;
            if (device) break;
```

```
          // if SCO device is requested but no SCO device is available, fall back to default
```

```
           // case
            // FALL THROUGH
       //我们还记得强制设置那里吗？对了，此时都是 FORCE_NONE
      //而且，mAvailableOutputDevices 是 0X3 (外放|耳机)
        default:    // FORCE_NONE
            device = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_WIRED_HEADPHONE;
            if (device) break;
            device = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_WIRED_HEADSET;
            if (device) break;
    //看，下面这句会成立。啥意思？如果有耳机的话，那么输出设备就是耳机
//太正确了。实际手机是不是就是这样的呢？
```

```
            device = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_EARPIECE;
            break;
     //再验证下我们刚才说的，如果强制使用外放的话，
        case AudioSystem::FORCE_SPEAKER:
            if (mPhoneState != AudioSystem::MODE_IN_CALL || strategy != STRATEGY_DTMF) {
                device = mAvailableOutputDevices &
                        AudioSystem::DEVICE_OUT_BLUETOOTH_SCO_CARKIT;
                if (device) break;
            }
       //果然，会强制使用外放。
            device = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_SPEAKER;
            break;
        }
    break;
    case STRATEGY_SONIFICATION://分析方法同上，我不说了。
        if (mPhoneState == AudioSystem::MODE_IN_CALL) {
            device = getDeviceForStrategy(STRATEGY_PHONE, false);
            break;
        }
        device = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_SPEAKER;
        // 同样没有 break，说明 SONIFICATION 受 MEDIA 策略影响。
    case STRATEGY_MEDIA: {
        uint32_t device2 = mAvailableOutputDevices &
AudioSystem::DEVICE_OUT_AUX_DIGITAL;
        if (device2 == 0) {
            device2 = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_WIRED_HEADPHONE;
        }
        if (device2 == 0) {
            device2 = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_WIRED_HEADSET;
        }
   //可惜，上面那些高级设备我们都没有
        if (device2 == 0) {
```

```
            device2 = mAvailableOutputDevices & AudioSystem::DEVICE_OUT_SPEAKER;
        }
  //假设我们没有从 SONIFICATION 下来，那么 device 最终会= DEVICE_OUT_SPEAKER。
//假设我们从 SONIFICATION 下来，那么 device 还是等于 DEVICE_OUT_SPEAKER
//奇怪，如果有耳机的话为何会走外放呢？普通耳机和线控耳机还能区分？
  device |= device2;
        } break;


    default:
        break;
    }
     return device;
}
```

好了，回到
[---->AudioPolicyManagerBase::updateDeviceForStrategy()]

```
void AudioPolicyManagerBase::updateDeviceForStrategy()
{
    for (int i = 0; i < NUM_STRATEGIES; i++) {
        mDeviceForStrategy[i] = getDeviceForStrategy((routing_strategy)i, false);
    }
}
这个函数完了，表明各种策略下使用的对应设备也准备好了。
```

真爽，一路回去，APS 的构造就完了。

```
留个纪念:
AudioPolicyManagerBase::AudioPolicyManagerBase(AudioPolicyClientInterface
*clientInterface)
{
  ....
 updateDeviceForStrategy();
}
AudioPolicyService::AudioPolicyService()
    : BnAudioPolicyService() , mpPolicyManager(NULL)
{
```

```
 #if (defined GENERIC_AUDIO) || (defined AUDIO_POLICY_TEST)
    mpPolicyManager = new AudioPolicyManagerBase(this);
    LOGV("build for GENERIC_AUDIO - using generic audio policy");
...
#endif
    property_get("ro.camera.sound.forced", value, "0");
    mpPolicyManager->setSystemProperty("ro.camera.sound.forced", value);
}
```

2.4 总结

总结下吧，AF,APS 都创建完了，得到什么了吗？下面按先后顺序说说。

- AF 创建了一个代表 HAL 对象的东西
- APS 创建了两个 AudioCommandThread，一个用来处理命令，一个用来播放 tone。我们还没看。
- APS 同时会创建 AudioManagerBase，做为系统默认的音频管理
- AMB 集中管理了策略上面的事情，同时会在 AF 的 openOutput 中创建一个混音线程。同时，AMB 会更新一些策略上的安排。

另外，我们分析的 AMB 是 Generic 的，但不同厂商可以实现自己的策略。例如我可以设置只要有耳机，所有类型声音都从耳机出。

上面关于 AMB 方面，我们还只是看了看它的代码，还没有一个实际例子来体会。