Android4.x 的 **RTSP** 框架学习

—— **NuPlayer** 介绍

本文介绍如下内容：

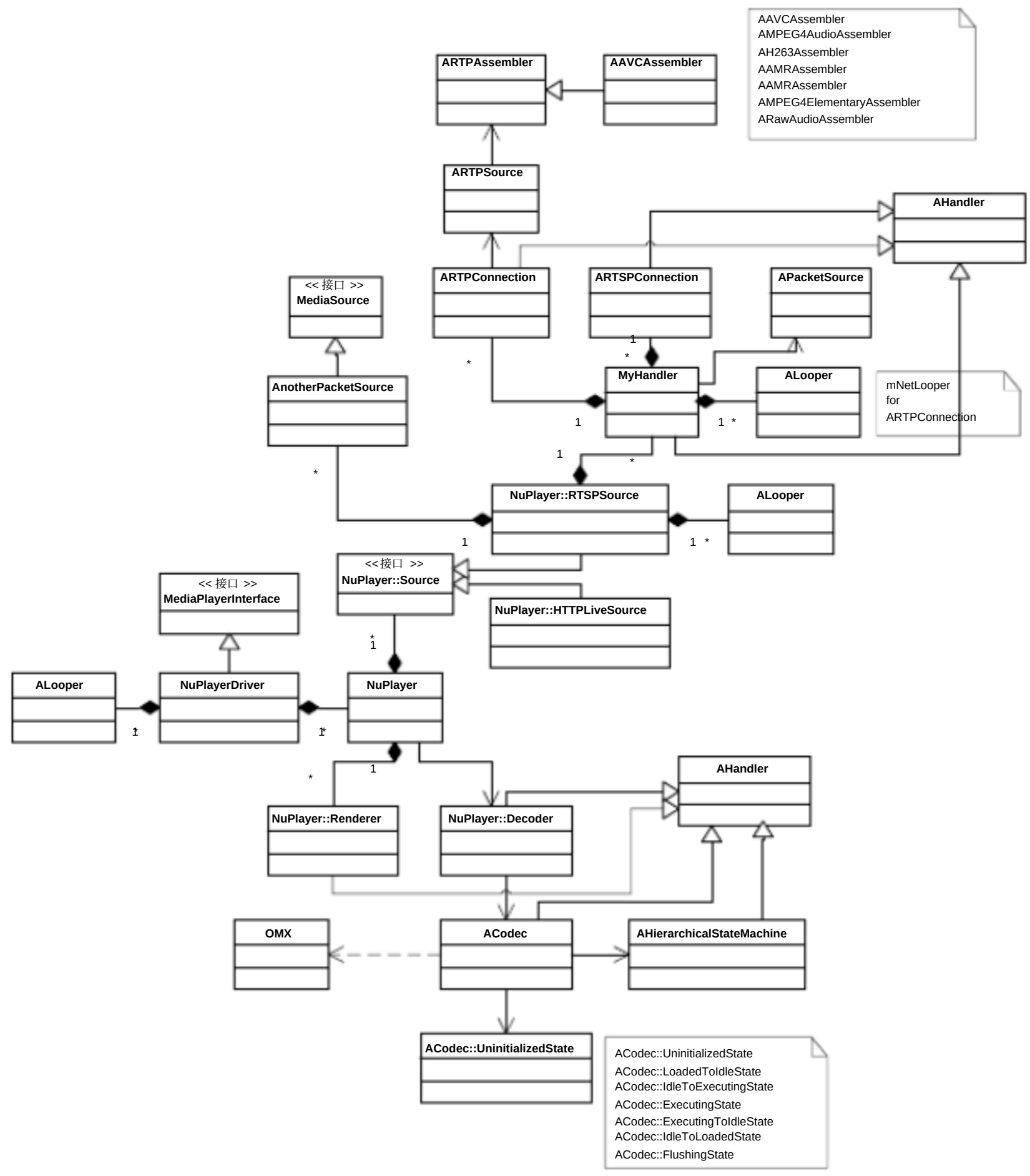
播放框架介绍

RTSP 源介绍

HTTP 流媒体的区别

要研究的点

NuPlayer 框图：



概述：

1. 整个播放框架是基于 ALooper、AHandler 和 Amessage 实现的消息机制的。
2. 从 MediaPlayerService 看，有 RTSP 和 http 且为 m3u8 的 url 会用 NuPlayerDriver，前者就走 RTSPSource 后者走 HTTPLiveSource(还没看呢)。
3. 分播放框架和 RTSP 源两部分来介绍，他们之间耦合很低，可以分开看代码。

播放框架：

1. NuPlayerDriver 是对 NuPlayer 的封装，前者继承 MediaPlayerInterface 接口，后者实现播放的功能。
2. NuPlayer 通过 RTSPSource 接口得到数据流的信息和解码数据本身，RTSPSource 的接口有 start、getFormat、getDuration、seekTo、dequeueAccessUnit、feedMoreTSData 和 stop。后面会对 NuPlayer 做详细介绍。
3. Decoder 的核心是 ACodec，后者相当于 stagefright 的 OMXCodec，实例化一个 OMX 的 Client，调用 OMX 组件，实现 Decode 功能。
4. ACodec 有几个跟 OMX 状态对应的内部类，这里有个状态机的概念。如果一个 msg 的 Handler 为 Acodec，那么他对应的处理函数就对应其所在状态的 onMessageReceived 实现。看 ACodec 代码要了解 OMX 的一些概念，否则找不到调用关系了，有些流程是要靠 OMX 回调来串联的。
5. Render，奇怪，我始终没能找到 google 组件软解的显示是在哪里实现的，期待 debug。
6. Android4.0 引入了 SurfaceTexture 的概念，在给组件分配输出 buffer 时有两种方式可选，分配一段内存 buffer，或是从显示模块分配 buffer，后者显然更高效。前者还需要再做一次格式转换，后者貌似由 OMX 完成。
7. OMX 组件，在 qcom 平台上有两套组件 plugin，qcom 的和 google 的，qcom 在 libstagefrighthw.so 中。google 的组件和库命名规则为 "OMX.google. aac.decaac.decoder"// 对应动态库 libstagefright_soft_aacdec.so

RTSP 源：

1. RTSPSource 的与播放框架接口，是播放框架的数据源。其主要接口有：

```
RTSPSource(
        const char *url,
        const KeyedVector<String8, String8> *headers,
        bool uidValid = false,
        uid_t uid = 0);

virtual void start();
virtual void stop();

virtual status_t feedMoreTSData();

virtual sp<MetaData> getFormat(bool audio);
virtual status_t dequeueAccessUnit(bool audio, sp<ABuffer> *accessUnit);

virtual status_t getDuration(int64_t *durationUs);
virtual status_t seekTo(int64_t seekTimeUs);
virtual bool isSeekable();

void onMessageReceived(const sp<AMessage> &msg);
```

2. AnotherPacketSource 在 RTSPSource 中作为 mAudioTrack 和 mVideoTrack，他虽然继承了 MediaSource 接口，但是并没有使用 read 来读数据，而是通过 dequeueAccessUnit 接

口， Server 端的压缩流通过 queueAccessUnit 保存到这里。

3. MyHandler 是核心，其中包含 ARTSPConnection 和 ARTPConnection 两大部分。MyHandler 负责向 Server 端发送 Request 和处理 Response。
4. ARTSPConnection 负责维护 RTSP socket，发送 Request，循环接收 Server 端数据，响应 Server 的 Request。这里只是接收 Response，真正的处理在 MyHandler 侧。

```
void connect(const char *url, const sp<AMessage> &reply);
void disconnect(const sp<AMessage> &reply);

void sendRequest(const char *request, const sp<AMessage> &reply);
void observeBinaryData(const sp<AMessage> &reply);

static bool ParseURL(
        const char *url, AString *host, unsigned *port, AString *path,
        AString *user, AString *pass);
```

5. 代码使用的 RTSP 请求有 DESCRIBE，其 Response 中有 SDP 信息，接下来是 SETUP，将本端的端口信息等发给服务器，然后是 PLAY。为了保持 RTSP 链接，还要周期性发 OPTIONS。
6. ARTPConnection 负责 RTP 和 RTCP 两个 socket, 接收 RTP 和 RTCP 包, 周期性发送 RTCP 包。

Public 接口：

```
void addStream(
        int rtpSocket, int rtcpSocket,
        const sp<ASessionDescription> &sessionDesc, size_t index,
        const sp<AMessage> &notify,
        bool injected);

void removeStream(int rtpSocket, int rtcpSocket);

void injectPacket(int index, const sp<ABuffer> &buffer);

// Creates a pair of UDP datagram sockets bound to adjacent ports
// (the rtpSocket is bound to an even port, the rtcpSocket to the
// next higher port).
static void MakePortPair(
        int *rtpSocket, int *rtcpSocket, unsigned *rtpPort);
```

Private 接口：

```
List<StreamInfo> mStreams;

bool mPollEventPending;
int64_t mLastReceiverReportTimeUs;

void onAddStream(const sp<AMessage> &msg);
void onRemoveStream(const sp<AMessage> &msg);
void onPollStreams();
void onInjectPacket(const sp<AMessage> &msg);
void onSendReceiverReports();

status_t receive(StreamInfo *info, bool receiveRTP);

status_t parseRTP(StreamInfo *info, const sp<ABuffer> &buffer);
status_t parseRTCP(StreamInfo *info, const sp<ABuffer> &buffer);
status_t parseSR(StreamInfo *info, const uint8_t *data, size_t size);
status_t parseBYE(StreamInfo *info, const uint8_t *data, size_t size);

sp<ARTPSource> findSource(StreamInfo *info, uint32_t id);

void postPollEvent();
```

7. 每个 RTP 数据流都有一个 ARTPSource，后者会创建一个 ARTPAssembler。依据处理数据流的压缩格式，实例化对应格式的 Assembler。

ARTPAssembler 的 Public 接口：

ARTPAssembler 的 Private 接口：

```
AssemblyStatus addNALUnit(const sp<ARTPSource> &source);
void addSingleNALUnit(const sp<ABuffer> &buffer);
AssemblyStatus addFragmentedNALUnit(List<sp<ABuffer> > *queue);
bool addSingleTimeAggregationPacket(const sp<ABuffer> &buffer);

void submitAccessUnit();
```

8. ARTPAssembler 对 ARTPConnection 接收到的数据进行处理， 如 AVC 数据，他会把单一 NAL，NAL 分片和复合 NAL 分别处理后， 都以单独 NAL 的形式回调传给 RTSPSource，存放在 AnotherPacketSource 中，供 decoder 端使用。

HTTP 的比较：

1. HTTPLive 还没看。
2. 普通的 http 和 rtsp 最大的区别是什么呢？

HTTP 流媒体是本地解析，而 RTSP 流媒体是 server 段解析。HTTP 流媒体通过 HTTP 协议， 下载一定 buffer 量的 server 端文件到本地， 利用本地的 parser，像播放文件一样解析，播放。而 RTSP 流媒体，从 server 端得到的就是可解码流。

HTTP 在实现 seek 时，是丢掉当前所有 buffer 内容的。

要研究的点：

1, RTSP 协议， request 和 response 数据包的解析。
2, RTSP 播放过程的管理。
Seek 过程为， 先向 Server 端发一个 PAUSE，带响应后再发一个带时间点的 PLAY 请求。
3, SDP 协议。
4, RTP 和 RTCP 协议，要了解接收到的 RTP 包头字段的信息。
5, RTSP 协议，要了解接收到 RTCP SR(SendReprot)和发送出的 RR(Receive Report) 信息。
6, 如 AVC 要了解 NAL 单元信息。