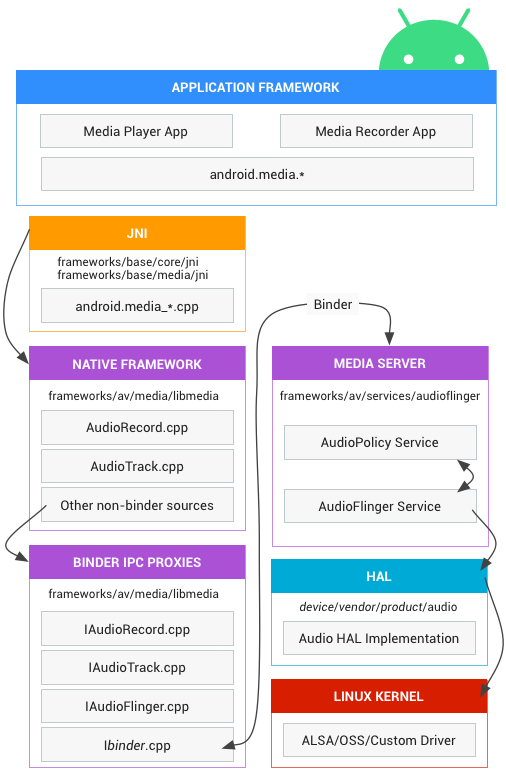
**一．技术背景**

借助5G高带宽、低时延的特性，结合云端大容量、高性能的运算及存储资源，终端的运算和存储能力逐渐迁移到云端，本地和云端只需进行指令、音频、视频流的交互，便可实现远程操控云端系统，畅想云端无限存储、计算资源的效果。云-端音频传输是云侧和端侧协同的关键技术，同时可为云手机、云游戏、云办公等场景赋能，提供更流程、更安全、体验更佳的5G云服务。

**二. 技术调研**

**1. android 音频整体设计框架概述**

Android 音频架构定义了音频功能的实现方式，并指出实现中所涉及的相关源代码。



**应用框架**

应用框架包含应用代码，该代码可使用 [android.media](http://developer.android.com/reference/android/media/package-summary.html) API 与音频硬件进行交互。在内部，此代码会调用相应的 JNI 粘合类，以访问与音频硬件交互的原生代码。

**JNI**

与 android.media 关联的JNI 代码可调用较低级别原生代码，以访问音频硬件。JNI位于 frameworks/base/core/jni/ 和 frameworks/base/media/jni 中。

**原生框架**

原生框架可提供相当于 android.media 软件包的原生软件包，从而调用 Binder IPC 代理以访问媒体服务器的特定于音频的服务。原生框架代码位于 frameworks/av/media/libmedia 中。

**Binder IPC**

Binder IPC 代理用于促进跨越进程边界的通信。代理位于 frameworks/av/media/libmedia 中，并以字母“I”开头。

**媒体服务器**

媒体服务器包含音频服务，这些音频服务是与您的 HAL 实现进行交互的实际代码。媒体服务器位于 frameworks/av/services/audioflinger 中。

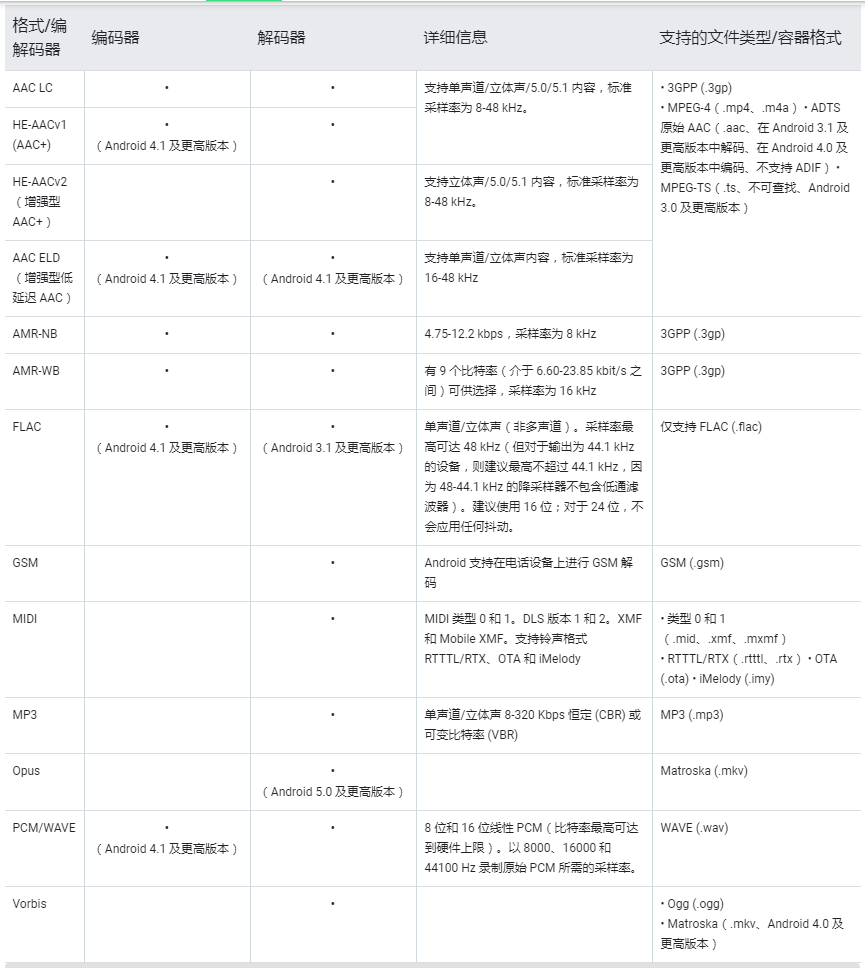
**HAL**

HAL 定义了由音频服务调用且您必须实现以确保音频硬件功能正常运行的标准接口。音频 HAL 接口位于 hardware/libhardware/include/hardware 中。

**内核驱动程序**

音频驱动程序可与您的硬件和 HAL 实现进行交互。您可以使用高级 Linux 音频架构 (ALSA)、开放声音系统 (OSS) 或自定义驱动程序（HAL 与驱动程序无关）。

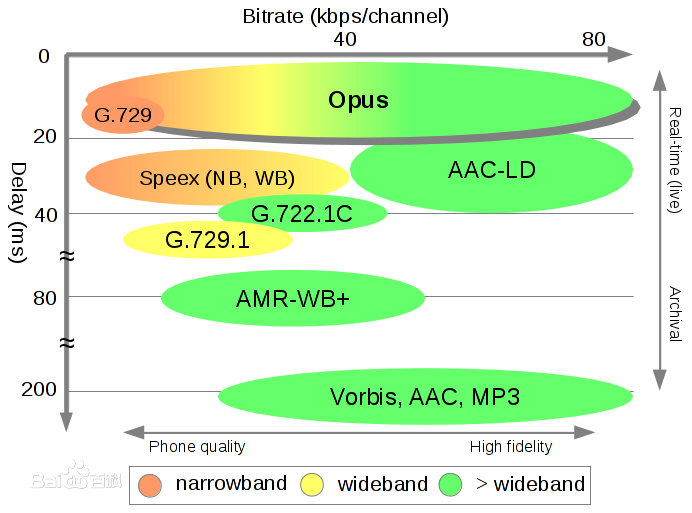
1. **android 支持的音频格式以及编解码器**

****

1. **opus 编解码介绍**

Opus是一个有损声音编码的格式，由Xiph.Org基金会开发，之后由IETF互联网工程任务组进行标准化，目标用希望用单一格式包含声音和语音，取代Speex和Vorbis，且适用于网上上低延迟的即时声音传输，标准格式定义于RFC6716文件。Opus格式是一个开放格式，使用上没有任何专利或限制。

Opus集成了两种声音编码的技术：以语音编码为导向的SILK和低延迟的CELT。Opus可以无缝调节高低比特率。在编码器内部它在较低比特率时使用线性预测编码在高比特率时候使用变换编码（在高低比特率交界处也使用两者结合的编码方式）。Opus具有非常低的算法延迟（默认为22.5 ms），非常适合用于低延迟语音通话的编码，像是网上上的即时声音流、即时同步声音旁白等等，此外Opus也可以透过降低编码码率，达成更低的算法延迟，最低可以到5 ms。在多个听觉盲测中，Opus都比MP3、AAC、HE-AAC等常见格式，有更低的延迟和更好的声音压缩率。

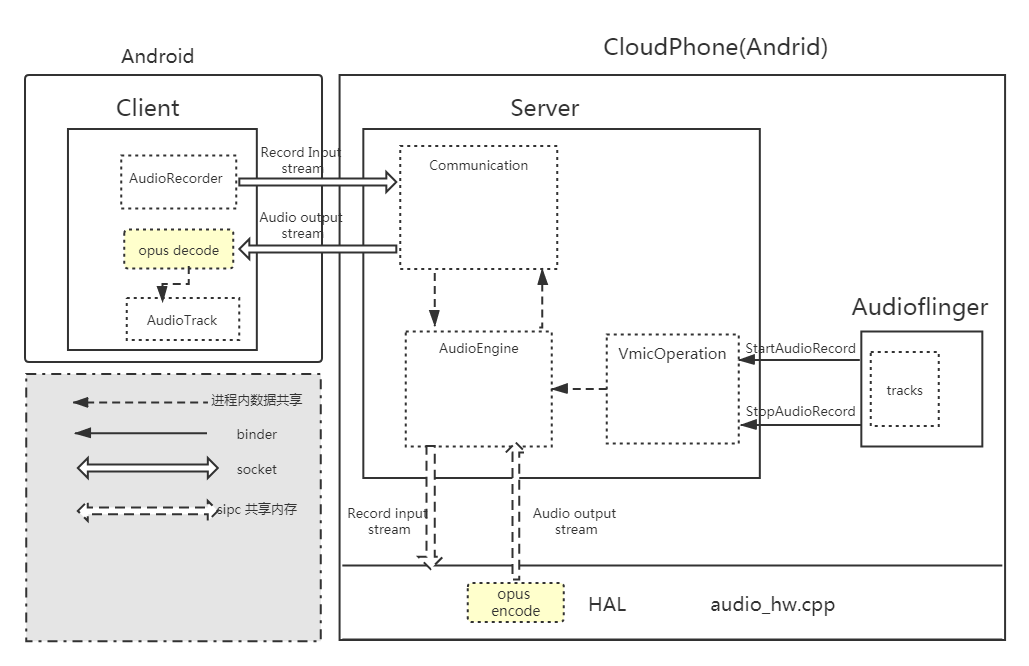
****

**Opus概述图**

由于Opus具有的低延时和高压缩率的特性，相比于其他的音频编解码格式，更符合我们的应用场景。

1. **音频传输方案整体框架**

服务端（安卓侧+协议侧），客户端（协议）整体架构图。



音频部分主要实现两个功能：音频注入,音频输出。

音频注入：采集客户端麦克风的输入数据并注入到服务端。

音频输出：获取服务端产生的音频输出数据，先进行编码，然后将编码后的数据发送到客户端。客户端收到数据后先进行解码然后播放。

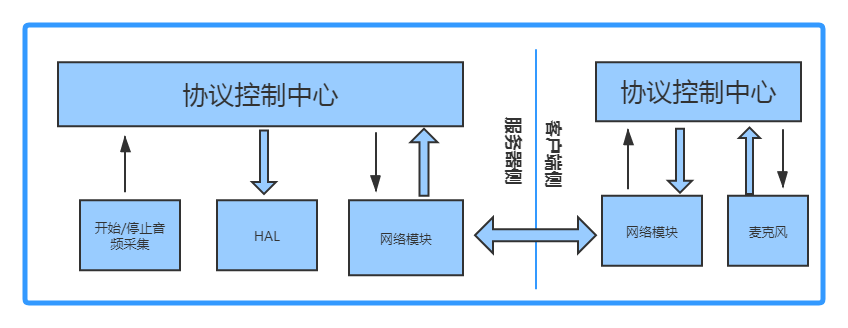
如上图所示，本方案主要由服务端和客户端两部分组成。而这两个部分又分别由自己的两个小模块组成，分别为：服务端 Android 侧，服务端协议侧，客户端 Android 侧，客户端协议侧。

服务端 Android 侧: 当有音频输出数据时，从 Android 的 HAL层获取音频输出数据，然后对原始数据进行编码，并通过共享内存的方式将编码后的音频数据发送到服务端协议侧。当需要采集音频注入数据时，从 Audioflinger 发送开始采集音频数据的命令给服务端的协议侧，或者发送停止采集音频数据给服务端的协议侧。

服务端协议侧：接收服务端 Android 侧的音频输出数据，并通过 socket 将数据发送到客户端协议侧。监听服务端 Android 侧的开始或者停止音频数据采集命令，并将命令通过 socket 发送到客户端协议侧，然后接收客户端协议侧发来的音频采集数据，最终将数据通过共享内存注入到服务端 Android 侧的 HAL层。

客户端协议侧：接收服务端协议侧发来的音频数据然后进行解码，最终通过 AudioTrack 对解码数据进行播放。通过 AudioRecord 采集客户端麦克风输入数据，并将采集到的数据通过 socket 发送到服务端的协议侧。

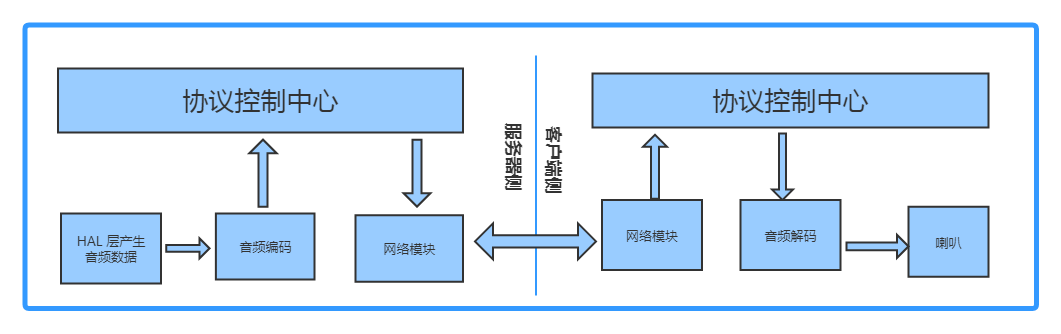
1. **音频注入方案实现**
2. **方案实现框架**

****

1. **方案实现概述**

开始和停止音频采集的控制命令由 audioflinger 触发。因为audioflinger 和协议端处在两个进程中，因此两个进程间的通信通过 binder 进程间调用实现。在协议端实现 binder 服务端，audioflinger 实现 binder 客户端。控制命令最终由网络模块由协议服务端传到协议客户端。客户端收到命令后通过麦克风采集数据，并最终将采集到的数据发送到协议服务端。

1. **音频输出方案实现**
2. **方案实现框架**



1. **方案实现概述**

服务端 Android 侧首先获取 HAL 层产生的原始 pcm音频数据，然后对pcm数据进行编码,编码后生成 opus 格式的数据。编码后的数据最终由网络模块发送到协议客户端，客户端收到 opus格式数据后对数据进行解码，解码后的数据最终通过喇叭进行播放。