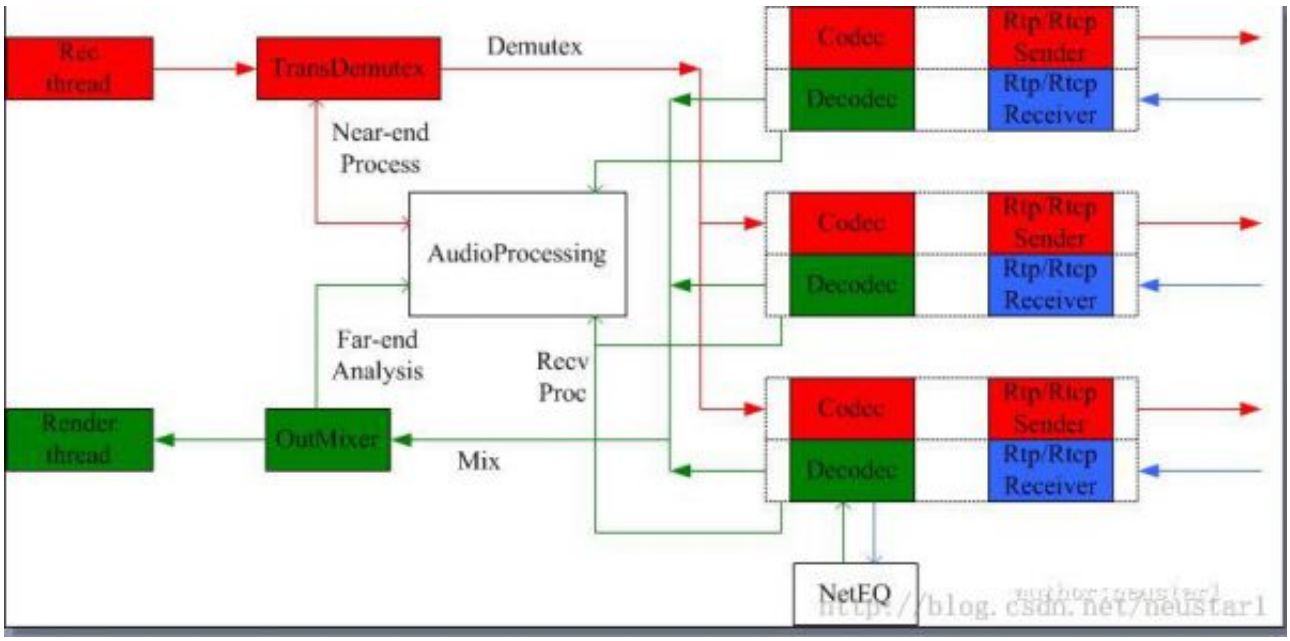
WebRTC源码分析：音频模块结构分析

一、概要介绍WebRTC的音频处理流程，见下图：



webRTC将音频会话抽象为一个通道Channel，譬如A与B进行音频通话，则A需要建立一个Channel与B进行音频数据传输。上图中有三个Channel，每个Channel包含编解码和RTP/RTCP发送功能。

以一个Channel而言，应用程序中将包含三个活动线程，录音线程，音频接收线程和播放线程。

1）录音线程：负责麦克风音频的采集，见图中红色路径，采集到音频后，缓存到一定长度，进行音频处理，主要包括EC，AGC和NS等。然后送到Channel，经过音频

Codec模块编码，封装成RTP包，通过Socket发送出去；

2）接收线程：见蓝色路径，负责接收远端发送过来的音频包，解封RTP包，解码音频数据，送入NetEQ模块缓存。

3）播放线程：负责耳机声音播放，见绿色路径。播放线程去OutMixer中获取要播放的音频数据，首先依次获取参与会话的Channel中NetEQ存储的音频帧，可以对其做AGC和NS处理；然后混合多个Channel的音频信号，得到混合音频，传递给AudioProcessing模块进行远端分析。最后播放出来。

如下为本地回环录音和播放代码：

```
VoiceEngine* ve = VoiceEngine::Create();
VoEBase* base = VoEBase::GetInterface(ve);
base->Init();
int chId = base->CreateChannel();
base->SetSendDestination(chId,3,"127.0.0.1",4);
base->SetLocalReceiver(chId,3,3001,"127.0.0.1");
base->StartPlayout(chId);
```

```
base->StartReceive(chId);
base->StartSend(chId);

    //....sleep...wait.....

base->StopSend(chId);

base->StopReveive(chId);

base->StopPlayout(chId);

base->Terminate();
```

本文介绍WebRTC音频模块组成和结构，详细介绍音频引擎的配置和启动，相信看完本文后，很多人可以利用WebRTC完成一个音频通话程序开发。

一、对外接口

音频部分的对外主要接口如下，各个接口之间的关系如图1所示。

1）VoiceEngine：负责引擎的所有接口查询，存储共享数据信息ShareData。

2）VoEBase：负责音频处理的基本操作。

3）VoEAudioProcessing：音频信号处理接口，设置各个音频处理项的参数。

4）VoECodec：音频编解码接口，提供支持的编解码器查询，音频编解码设置。

5）VoEHardware：音频硬件设备接口，负责音频硬件设备的设置。

其它的接口还有VoENetEqStats，VoENetwork，VoERTP_RTCP，VoEVideoSync，VoEVolumeControl，VoEFile，VoECallReport，VoEDtmf，VoEMeidaProcess和VoEEncryption。

WebRTC使用继承实现接口转换和查询，接口之间的数据共享是通过ShareData完成，首先VoiceEngineImpl继承各个对外接口的实现，所以可以从VoiceEngineImpl很容易获取其他对外接口。而VoiceEngineImpl本身也继承ShareData，当从VoiceEngineImpl获取其他对外接口的同时，隐式的传递了ShareData指针，因此各个接口可以很方便的获取到ShareData的数据信息。因此虽然类与类之间的关系看起来比较混乱，但是使用上比较方便。

利用VoiceEngine获取对外接口：VoEInterfaceXX* pInterf = VoEInterfaceXX:GetInterface(pVoiceEngine);

二、模块组成

主要由五大模块组成：AudioDeviceModule音频设备模块，AudioProcess音频处理模块，AudioCodingModule音频编码模块，AudioConferenceMixer混音模块和RtpRtcp传输模块。

ShareData用于粘合各个模块之间的关系，负责管理全局的对象，包括AudioDeviceModule，

TransmitMixer，OutputMixer，ChannelManager和AudioProcess。

录音流程：AudioDeviceWinCore负责采集音频数据，传递到AudioDeviceBuffer中缓存，AudioDeviceBuffer则将数据送入TransmixMixer，首先交给AudioProcess进行近端音频处理，完成后分发到各个Channel中，Channel则通过AudioCodingModule进行编码，编码后再交付到RtpRtcp中经由RTPSender发送出去。

接收流程：RTPReceiver负责接收音频RTP包，接收到RTP包后交给Channel，Channel转交给AudioCodingModule中的ACMNetEQ模块，进行解码缓存。

播放流程：Channel从ACMNetEQ模块中取出缓存的解码音频数据，如果需要进行远端数据处理的话，传递给AudioProcess处理。最后所有Channel都汇入到OutputMixer中进行混音，混音后再传递到AudioProcess进行远端音频分析。最后送入AudioDeviceModule中的AudioDevceWinCore播放。

三、配置

1、音频引擎创建与删除

```
VoiceEngine*pVoeEngine = VoiceEngine::Create();

VoiceEngine::Delete(pVoeEngine);
```

2、音频收发

1）音频通话链路创建

WebRTC中的Channel，为一路音频。作为网络语音通信，至少要创建一路音频Channel。

Channel没有提供对外接口，是有VoEBase来管理的，通过索引号来选定对应的Channel。

```
VoEBase*base = VoEBase::GetInterface(pVoeEngine);

int ch0 =base->CreateChannel();
```

2）网络端口设置

音频通过RTP和RTCP发送出去，RTP和RTCP使用UDP实现，需要配置网络端口和地址。

```
//设置发送给.2机器的3端口

base->SetSendDestination(ch0,3,”192.168.8.2”);

//在本机的3端口接收RTP包

base->SetLocalReceiver(ch0,3);
```

3）音频编码选择

VoECodec负责编解码的配置。

```
VoECodec*codec = VoEBase::GetInterface(pVoeEngine);
```

设置Channel的编码类型之前，要查询支持的编码列表。

```
CodecInstinst;
Intnum = codec->NumOfCodecs();
for(int i=0; i<num; ++i)
{
    Codec->GetCodec(I,inst);
   //打印编码信息
}
//设置编码0
Codec->GetCodec(0,inst);
Codec->SetSendCodec(ch0,inst);
```

WebRTC自动识别编码类型，因此解码不需要设置。

4）启动

启动播放：base->StartPlayout(ch0);该操作含义是将通话ch0进行混音输出。

启动接收：base->StartReceive(ch0);开始接收后，每增加一路通话，引擎会将音频进行混音再输出。

启动发送：base->StartSend(ch0);启动发送的时候，会检查是否正在录音，如果已经开启录音，则不再开启；否则会执行音频设备录音操作。

3、音频处理的配置

VoEAudioProcessing负责音频处理的配置。

```
VoEAudioProcessing*pAudioProc = VoEAudioProcessing::GetInterface(pVoeEngine);
//启动AGC功能
pAudioProc->SetAgcStatus(true);
```

4、音频设备的配置

VoEHardware接口可以查看录音和播放设备，可以选择指定的设备进行音频通话。

```
VoEHardware*pHardware=VoEAudioProcessing::GetInterface(pVoeEngine);
Int numin =pHardware->GetNumOfRecordingDevices();
For(int i=0;i<numin; ++i)
{
    pHardware->GetRecordingDeviceNames(…)
    //打印录音设备
}
//选择设备0作为录音设备
pHardware->SetRecordingDevice(0);
```

播放设备配置类似。