MEDIASTREAMER2 分析研究

Ver 0.2

文档变更记录

日期	版本	作者	改动

1. 目的

本次分析研究 VOIP 中流媒体部分，是为将来在企业通讯客户端中实现软电话来呼叫外部电话；外部电话通过 IPPBX 上的语音提示来呼叫本地软电话；软电话和普通电话互通；不同办公地点之间音视频传输；语音邮件等。

在分析开源代码的基础上，扩展流媒体在传输过程中的编码和优化流媒体通讯过程中的语音，视频质量。

为满足上述目的，我们采用分析 LINPHONE 开源代码中流媒体部分，经过分析发现该流媒体部分在功能上完全能满足上述目的。该框架比较小且非常灵活，完全可以通过二次开发来优化传输过程中的流媒体质量。

2. 总体架构

2.1. 概述

LINPHONE 中流媒体处理分为 2 个部分 MEDIASTREAMER2 和 ORTP,前者负责媒体流的处理后者负责流媒体如何安全可靠的传输.

关于 MEDIASTREAMER2，以下简称 MS。

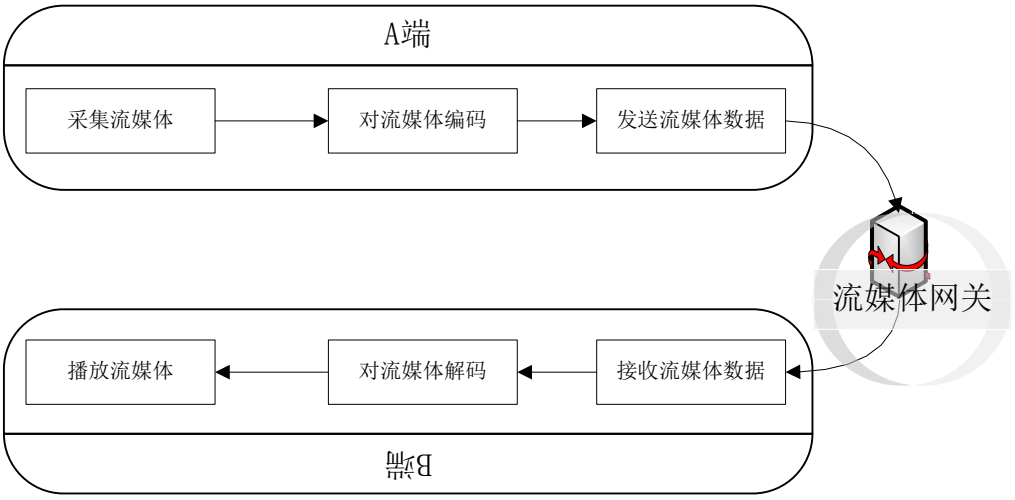
MS 和 ORTP 为二个项目文件，由于 MS 利用 ORTP 作为传输部分使用，因此在下面描述中认为 ORTP 是其中的一部分功能。

ORTP 实现了 RTP 协议,提供良好的 API 访问功能.多种 RTP 格式支持；发送接收的实时调度；单线程支持多路媒体流；自适应的缓冲区算法；实现了 RTCP。目前我们采用的是 ORTP 0.16.1 的最新版本.

2.2. 总体描述

2.2.1. 业务流程描述

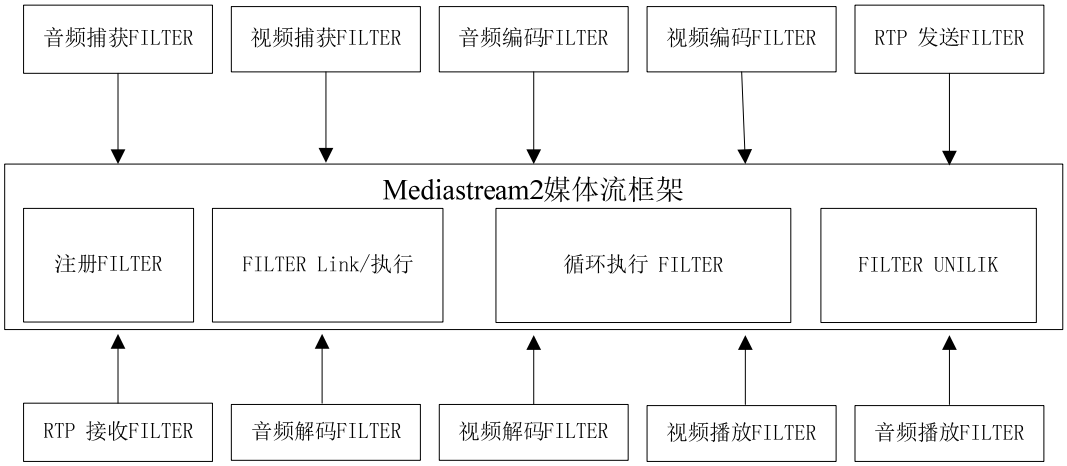
同很多软终端一样，在 MS 中实现的业务流程基本一样。以下图描述了一个单向的处理。MS 在传输过程中用一个全双工的 session 来传送视频或者音频，不管是本机还是远端主机，运行的都是同一个程序，一次只能选择一种 payload。



以上为一完整的 VOIP 软终端的实现流程，采集语音和视频的信息，经过一定的编码，通过 RTP 协议传输流媒体数据到网关（IPPBX），流媒体网关可能会对数据进行处理，也可能是直接连接流媒体，这时候对方可以接收流媒体的数据开始播放。这样就实现连单方的通话，把整个流程反过来就实现了一个完整的流媒体传输通道。

2.2.2. 总体功能模块描述

MS 的总体逻辑关系非常简单，它利用函数指针将一系列的 FILTER 组合起来，通过一个线程来调度这些 FILTER 上的函数指针。



以上为整个 MS 的功能模块图，可以看出 MS 框架除了具有维护 FILTER 功能之外处理事情非常少，它利用函数指针来管理这些 FILTER，而这些 FILTER 之间的逻辑关系则通过系统启动的时候配置加载，也就是说如果实现一个电话软终端，我们可以根据 SIP 协议和对方协商的 SDP 配置包来决定我们来加载语音或者是视频流。

2.3. 功能描述

根据总体功能模块描述分以下几个小的功能:

2.3.1. 注册 FILTER

注册分为二部分，一部分为 RTP 协议中的注册，由于 ORTP 基本上实现了 RTP 协议上的所有规范，在 ORTP 部分先要注册与协议相关的信息，而在 MS 部分则主要注册 FILTER 的 ID；FILTER 的函数指针；以及该函数指针和外部的 API 的调用关系。这部分要涉及到写程序，实现各种 FILTER 和外部函数之间的一一对应关系。

2.3.2. FILTER Link/执行

由于有太多的 FILTER，我们无法确定所有用户之间通讯都采用一种编码的方式，而且 FILTER 之间的逻辑关系也无法确定，因此需要一个配置能决定我们采用的编码，FILTER 之间的关系.基本上每个 FILTER 都具备一个入一个出的属性，但是在采集和播放的时候 FILTER 只有一个属性。

在 FILTER LINK 之前基本上决定了流媒体的类型，LINK 之后我们又知道了每个 FILTER 之间的关系。但是我们在采集媒体信息的时候要设置声音效果，质量，回音处理等。所以 FILTER 还要提供一次性的方法来配置这些属性。在 FILTER LINK 之前通过函数指针来调用 FILTER 的附加函数指针，该调用方式一般是一次行为。非 FILTER 的标准函数指针。

2.3.3. 循环执行 FILTER

这个功能非常重要，在上面定义了 FILTER,定义了 FILTER 的先后逻辑关系.现在要通过一个线程来持续不断的处理这些逻辑.由于 FILTER 之间的关系通过一个非循环的双链表来关联. 这个过程并非连续循环执行，循环过程只是从队列头到队列尾的一个过程. 语音流程有 2 个队列头,一个以采集为起点到发送为终点;一个以接收为起点到播放为终点.而视频则不这样。该功能将在下面详细描述。

2.3.4. FILTER UNILIK

取消每个 FILTER 之间的关系。

2.3.5. RTP 发送/接收 FILTER

MS 通过对 ORTP 的封装，在调用的时候认为 ORTP 传输部分的 FILTER，按照实现方式来说，我们可以采用其它的 RTP 传输库来做 FILTER。通过 ORTP 库中集成的 RTP 库可以猜测，该功能是可以实现的。

2.3.6. 音频编解码 FILTER

音频的编码和解码一般为一对,在 MS 中支持的有 g711u, g711a, speex 和 gsm 等编码,目前已经加入了 G729A.,也能够通过动态的办法加载新的编码.

2.3.7. 视频编解码 FILTER

视频的 FILTER 目前支持 H263-1998, MP4V-ES, theora.同时也可以通过插件的方式来添加 H264 的编码,目前的库是 X264。

2.3.8. 音视频播放 FILTER

这个 FILTER 只是通过 WINDOWS API 来从设备上读数据和写数据到设备上.

2.3.9. 音视频捕获 FILTER

2.3.10. ORTP 的功能描述

多种 RTP 格式支持

ortp 有一个 RtpProfile 的全局变量，该变量在 ortp 初始化的时候赋值，该变量中有一个 PayloadType 的数组，每个 PayloadType 对应一种媒体流，PayloadType 中包含了媒体流的特性，包括名称，采样率，声音位数，静音格式，占用带宽等。这些定义都在 avprofile.c 中。其中的几个参数中，采样率用的比较多，主要用来计算实时性.。

发送接收的实时调度。

发送和接收主要是两个函数 rtp_session_send_with_ts 和 rtp_session_recv_with_ts。以 rtp_session_recv_with_ts 为例：内部接收数据使用的是 rtp_session_recvm_with_ts，首先，会接收所有 scoket 上的数据，然后将 rtp 包存放在一个队列之中，一系列处理之后，有一个 pthread_mutex_lock 的线程锁，将线程锁住。此时，由 rtp_scheduler_schedule 线程进行调度（该线程在协议栈初始化）时创建。rtp_scheduler_schedule 会遍历所有的 media session（媒体流），然后判断其中的 timestamp（时间戳），如果计算的时间到达，则让 rtp_session_recvm_with_ts 继续处理。

时间戳的算法是以第一个打到的 rtp 数据包为准，然后根据其中的时间，进行推算。假如第一个包是 10 点整来的，然后 ptime 又是 20ms，那么下一个包的时间就是 10 点又 20 毫秒。

media session 是一个 RtpSession 对象，包含多种属性和方法。RtpScheduler 中包含一个 RtpSession 的队列，用来支持多媒体流。

值得一提的是，rtp_scheduler_schedule 中有一个独特的"sleep"，该 sleep 可以停顿 10ms。并且这个时间是绝对的，如果中间因为处理或者其他原因延迟了 2ms，那么这个 sleep 停顿的就是 8ms。具体函数可以看一下 posixtimer.c 中的 posix_timer_do 实现。精确的计时使用 select，精确

时间的取得很多使用 gettimeofday 这个函数，该函数在精确计时的时候非常有用。

单线程支持多路媒体流

对外是有一个 sessionset，完全模拟了 select 的做法，对外提供的接口，也和标准 select 几乎一样。主要的处理实现，还是在 rtp_scheduler 里面完成。模拟 select 的唤醒，使用了 pthread_cond_wait。

自适应的缓冲区算法

主要的实现都在 jitterctl.c 里面。

RTCP 的实现

rtcp 是不算很复杂的东西。ortp 中，在 RtpSession 里面，即包含了 rtcp 的一些相关属性。需要的时候，直接取出，然后组包发送就可以了。不过还要涉及到 SDES。

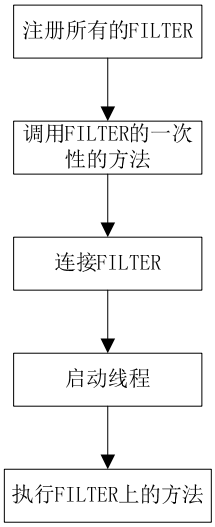
SRTP 的实现

见 SRTP 的说明

2.4.程序运行逻辑

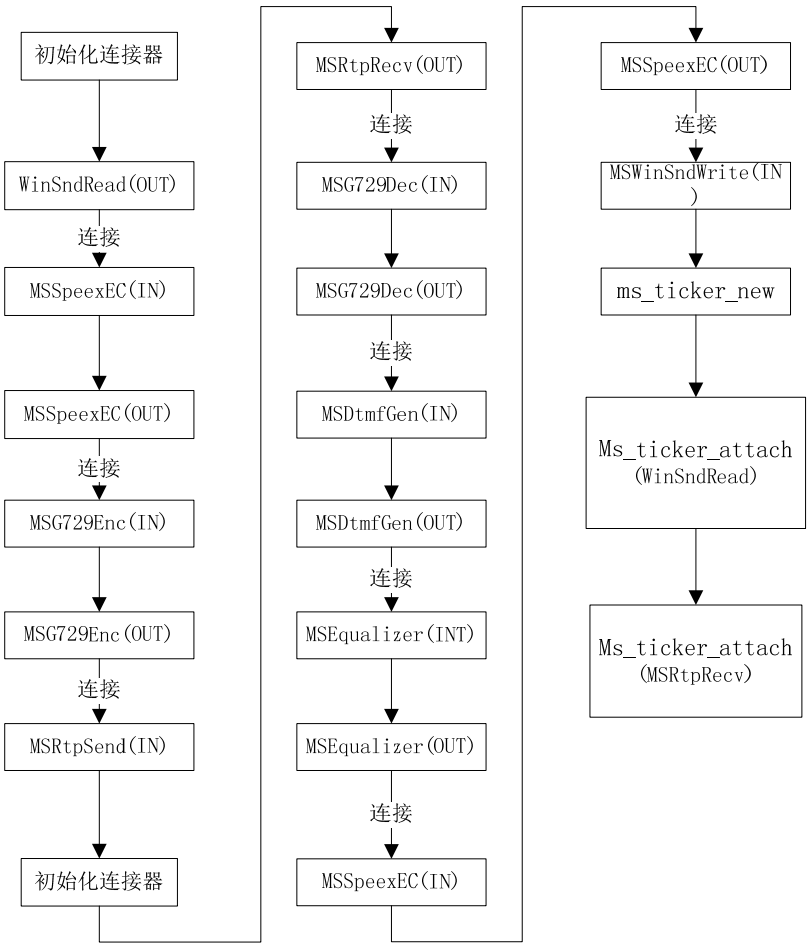
2.4.1. MEDIASTREAMER2

在这里我们简要的分析下程序加载的逻辑，从前面的描述过程中我们可以很容易看出 MS 的程序运行逻辑。



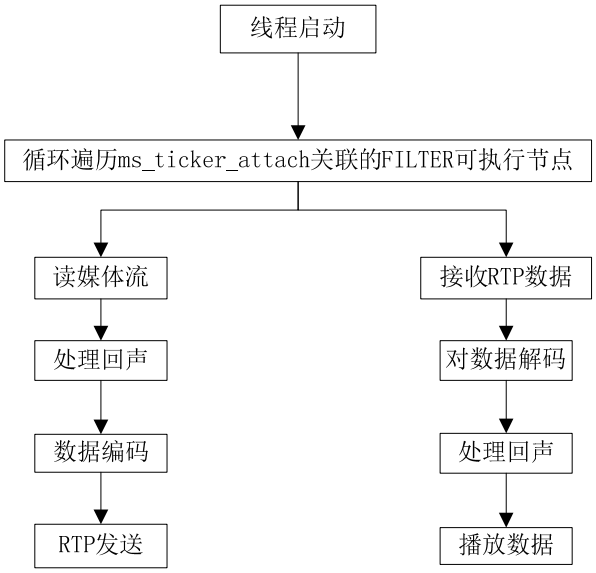
上图简要的描述了 MS 的程序加载逻辑,当然在调用的过程中具体的细节实现根据具体应用有所区别。

注册很好理解,见二次扩展部分就很容易理解,下面先描述下如何定义 FILTER 的前后逻辑流程.我们以一个语音通讯的具体例子来描述.



根据上面流程描述，定义整个流程非常简单，只是需要注意最后的二个 ATTACH，它是为了找到二个 FILTER 的入口，一个为从采集语音的数据，是发起的源头，另外一个为 RTP 接收的 FILTER，为接收的源头。

流程定义完成后,下面我们将细化一下并描述这些 FILTER 被调用的过程。



通过上图可以知道，在线程启动后，首先找到 ATTACH 上的二个 FILTER 的节点指针，然后通过一个

递归来执行到最后一个节点。执行完成后再重新开始。下面的二个分支是不确定的，根据不同的流程设置，应该调用的 FILTER 也不同。

2.4.2. ORTP

ORTP 的发送处理流程

1,初始化 ortp_init();
2,初始化调度线程 ortp_scheduler_init();
3,设置媒体传输过程中有 DTMF 数据 rtp_profile_set_payload;
4,创建一个发送的 SESSIONsession=rtp_session_new(RTP_SESSION_SENDONLY);
5,设置是否使用调度任务 rtp_session_set_scheduling_mode(session,1);
6,设置阻塞模式 rtp_session_set_blocking_mode(session,1);
7,设置对端的地址 rtp_session_set_remote_addr();
7,设置默认的编码格式 rtp_session_set_send_payload_type(session,0);
8,设置退出的信号
9,得到数据发送数据 rtp_session_send_with_ts
10,发送 DTMF 信号
11,关闭 SESSIONrtp_session_destroy(session);
12,退出 ortp_exit();

其实以上处理流程根本不需要用调度任务,因为只有一个 SESSION,个人感觉没有必要.在多个 SESSION 的状态下用调度任务比较合适.,结合语音的框架建议用非任务调度的方式,该描述在语音模块中具体再说.

ORTP 的接收处理流程

1,初始化 ortp_init();
2,初始化 ortp_scheduler_init();
3, 设置媒体传输过程中有 DTMF 数据.
4,创建一个接收的 session=rtp_session_new(RTP_SESSION_RECVONLY);
5,设置是否使用调度任务 rtp_session_set_scheduling_mode(session,1);
6,设置阻塞模式 rtp_session_set_blocking_mode(session,1);
7,设置本地地址 rtp_session_set_local_addr();
8,设置默认的编码格式 rtp_session_set_payload_type(session,0); pcm
9,设置回调 rtp_session_signal_connect();
10,接收数据 rtp_session_recv_with_ts(session,buffer,160,ts,&have_more);
11,关闭 rtp_session_destroy(session);
12,ortp_exit();

3. 编译流程

3.1. 平台描述

MS 的代码由于是纯粹的 C 编写，很容易支持各种平台，目前我在 WINXP 上编译。而且目前的应用估计 90%都会用到 WINDOWS 系统。所以下面的描述都是基于这个平台。

3.2. 依赖环境

这里的依赖环境是说要编译 MS 和 ORTP 库还需要那些库文件的支持。如果不考虑各种编码，其依赖的库也不太多。

ORTP 依赖的库主要是 SRTP 和 OPENSSL 的库。

SRTP:

简单介绍：安全的适时传输协议,一个 Cisco 公司人写的

下载地址：http://srtp.sourceforge.net/srtp-1.4.2.tgz

注意事项：由于 ORTP 中要集成了 SRTP 库，所以在编译 ORTP 的时候要找 SRTP 的头文件。

OPENSSL：到处都有，在这里最简单的办法是引用 OPENSSL 的头文件和加载二进制的代码。

RTP: 见 RTP 协议简单描述。

MS 库主要依赖的库主要有：swscale quartz dmoguids strmiids strmbasd libtheora vfw32.lib libavcodec libgsmcodec ortp speex speexdsp

quartz dmoguids strmiids strmbasd ：这四个库主要是 DirectShow 中产生的。主要用来处理一些与音视频有关的多媒体任务，比如音视频采集、回放等。

具体要安装 DirectShow，我下载的是 dx90bsdk.exe，安装后编译 DX9 SDK\Samples\C++\DirectShow\BaseClasses 这个目录下的文件，产生 strmbasd 这个库，别的库文件安装之后就存在了。

Swscale,libavcodec：从 FFMPEG 中提取，我们可以直接用这个库的二进制文件和头文件。FFmpeg 是一套可以用来记录、转换数字音频、视频，并能将其转化为流。

下载地址： svn checkout svn://svn.ffmpeg.org/ffmpeg/trunk ffmpeg

或者 git clone git://git.ffmpeg.org/ffmpeg/

cd ffmpeg

git clone git://git.ffmpeg.org/libswscale/

libtheora：Theora 是一个开源的视频编解码器，属于 Ogg 项目的一部分。

下载地址：http://downloads.xiph.org/releases/theora/libtheora-1.1.1.zip

Speex：是一套专门用于压缩声音的库,由于其专门针对声音,所以压缩声音的性能非常高.同时也可以用于回音处理,关于回声处理,我以前以为在 MS 中是通过微软的 API 处理的,经过研究发现还是通过 SPEEX 来实现,只不过版本不同而已.

下载地址：http://www.speex.org/downloads/

3.3.编译设置

将上面的依赖库文件编译出来后，编译 MS 库和 ORTP 库就非常简单了，将头文件的路径指向这些头文件目录，就可以了。

Ortp:

头文件目录：..\..\..\win32-bin\include" /I "..\..\include" /I "..\..\include\ortp" /I "..\..\src" /I "..\..\build\win32native\include"

库文件目录：..\..\..\win32-bin\lib

库文件：libeay32.lib libsrtp.lib

Ms 库：

头文件目录:：..\..\..\win32-bin\include" /I "..\..\include" /I "..\..\include\ortp" /I "..\..\src" /I "..\..\build\win32native\include"

库文件目录：..\..\..\win32-bin\lib

库文件：libswscale.a quartz.lib dmoguids.lib strmiids.lib strmbasd.lib libtheora.lib vfw32.lib libavcodec.dll.a libgsmcodec.lib ortp.lib libspeexd.lib libspeexdsp.lib

4.二次扩展

4.1.功能修改

在输出部分的修改：

目前为调试方便将所有的信息输出,采用 DEBUGVIEW 查看的方式,而且对每条信息标志出了线程号,时间,错误级别等。

设备检测的修改：

去掉动态检测声卡设备的线程，该线程每 5 秒检测一次声卡设备，感觉这种动态的应用非常少，而且占用大量的 CPU，因此屏蔽了该功能。

4.2.编/解码的扩展

在 MS 库中做编码的添加修改，非常方便，但是个人感觉还是不太简单，因为很多时候对编码的修改不是一个流媒体输入和流媒体的输出那样简单。必须对编码应该有所了解。

目前我只对音频部分做了 G729 的添加，在视频部分后续将继续介绍。

4.2.1. ORTP 扩展

个人认为一个好的 RTP 协议应该不象 ORTP 协议配置流媒体类型这样复杂，仅仅因为要支持一种新的媒体编码就需要将程序编译一遍，感觉还是不太容易接受。

以下以 G729A 为例：添加 G729 编码.,G729 在这个库中已经定义好了。以下的说明是为描述如何让 ORTP 支持一个新的编码.

ORTP 部分具体过程如下:

1，在 AVPROFILE.C 中添加 G729 的定义.描述:

```
PayloadType payload_type_g729={
    TYPE( PAYLOAD_AUDIO_PACKETIZED),
    CLOCK_RATE(8000),
    BITS_PER_SAMPLE( 0),
    ZERO_PATTERN(NULL),
    PATTERN_LENGTH( 0),
    NORMAL_BITRATE( 8000),
    MIME_TYPE ("G729"),
    CHANNELS(1)
};
```

简单描述: 类型为一个音频的数据包,时钟频率为 8000,类型为 729,占用一个通道.表示 RTP 分组第一个字节的取样时刻。其初值为随机数，每个采用周期加 1。如果每次传送 20ms 的数据，由于音频的采样频率为 8000Hz，即每 20ms 有 160 次采样，则每传送 20ms 的数据，时戳增加 160.

2，同时在这个文件中的 av_profile_init 函数中加入

```
rtp_profile_set_payload(profile,18,&payload_type_g729);
```

3，定义好后我们必须把其导出供其它应用来使用.在 PAYLOADTYPE.H 中加入

```
VAR_DECLSPEC PayloadType payload_type_g729;
```

4.2.2. MEDIASTREAMER2 扩展

在 ORTP 中加入完成后，下面将要扩展 MS2 的库，这个过程其实也可以简单的这样理解，函数提供一个媒体流的输入接口和一个媒体流的输出接口，来处理媒体信息。在处理前和处理后可能有一系列的处理，如在处理过程中需要共享部分信息。

1， 首先定义一个结构体

```
typedef struct G729State
```

在上面这个结构体中我们主要关注这个 BUF，用于来存储数据。

2， 初始化编码的函数和函数指针

```
MSFilterDesc ms_g729_enc_desc={
MS_G729_ENC_ID,
"MSG729Enc",
N_("The G729 full-rate codec"),
MS_FILTER_ENCODER,
"g729",
1,
1,
enc_init,
NULL,
enc_process,
NULL,
enc_uninit,
NULL
```

};

这里至少要定义三个函数，初始化，处理，注销。这几个函数处理函数必须不为空，除此之外我们还可以在处理前和处理后添加一些处理。其它描述见数据结构说明。

3, 初始化（enc_init）

首先我们需要初始化一个 BUF，用于处理中来存放数据，由于这个地方是用动态分配内存的方式，所以对应的注销中有释放处理，同时要初始化 G729 库的一些处理。

4, 处理 enc_process

具体的处理方式和编码算法有很大的关系，该函数并非每次都要有输出流和输入流的处理，如果输入的数据和编码算法要输入的数据包大小不一致的时候，可以不处理，当数据包的大小满足条件的时候则处理。

1， 首先从输入数据的指针中取数据放到我们在初始化 BUF 中。
2， 判断目前的 BUF 是否能满足算法所要求的数据。
3， 如果不满足则等下次来处理。
4， 如果满足则每次取等量的输入包，开始编码，处理后的包放到输出队列中。
5， 如果发现还有大小相同的包，则循环处理，直到输入的包小于要处理的包为止。
6， 退出过程。
7， 在这个函数成功的执行后该编码过程就完成了，后面的发送 FILTER 则可以判断函数的输出是否为空，如果不为空则从输出队列上取数据，发送出去。

5, 注销(enc_uninit)

当 FILTER 被 UNLINK 的时候会释放掉，先前初始化的操作。

6, 导出函数指针

MS_FILTER_DESC_EXPORT(ms_gsm_enc_desc)

7, 上面 2-6 只定义了 G729 的编码函数，下面需要定义接收部分收到数据包后如何处理（解包处理）。

8, 初始化解码的函数和函数指针

```
MSFilterDesc ms_g729_dec_desc={
    MS_G729_DEC_ID,
    "MSG729Dec",
    N_("The G729 codec"),
    MS_FILTER_DECODER,
    "g729",
    1,
    1,
    dec_init,
    NULL,
    dec_process,
    NULL,
    dec_uninit,
    NULL
};
```

9, 解包初始化（dec_init）

10, 解包处理（dec_process）

简单的说就是得到输入包上的数据，解包后放到输出上就可以，输出的数据可以后续处理。

11, 在 ALLFILTERS。H 定义函数类型的声明

MS_G729_ENC_ID,MS_G729_DEC_ID

12, 在 ALLCODESCS。H 中声明函数

```
extern MSFilterDesc ms_g729_dec_desc;
extern MSFilterDesc ms_g729_enc_desc;(编码,上面没有说)
```

13, 初始化结构体 ms_filter_descs[]，在末尾加上

```
&ms_g729_dec_desc,
&ms_g729_enc_desc,
```

以上为 MS 中添加 G729 的过程完成。

4.2.3. 调用

在 ORTP 和 MS 中这些部分完成后，我们所要做的就是开始调用和调试了。这个过程很简单。

```
stream->encoder=ms_filter_create_encoder(18);
//18为ORTP中关于G729的定义
stream->decoder=ms_filter_create_decoder(18);
```

这样 G729 的编码就加上去了,并可以使用了。

4.3. 插件的扩展

4.3.1. ORTP 中的配置

4.3.2. MS 中的配置

4.3.3. 遵循的函数接口标准

4.3.4. 调用

5. 数据结构

5.1. 框架数据结构

首先需要定义一些函数指针来连接和外部的函数接口，在这里通过几个简单函数指针来实现的。以下是关于所有 FILTER 的管理的数据结构，通过这些数据结构和控制的 API 来管理 FILTERS。

5.1.1. 函数指针定义：

```
typedef void (*MSFilterFunc)(struct _MSFilter *f);
```

作用： 定义 FILTER 初始化（INIT），处理前（PREPROCESS），处理（PROCESS），处理后（POSTPROCESS）和注销（UNINIT）的函数指针。

```
typedef int (*MSFilterMethodFunc)(struct _MSFilter *f, void *arg);
```

作用：定义 FILTER 和设置 FILTER 选项

```
typedef void (*MSFilterNotifyFunc)(void *userdata , unsigned int id, void *arg);
```

作用：定义 FILTER 中回掉函数的入口

定义 FILTER 的类别

MS_FILTER_OTHER, 其它类型

MS_FILTER_ENCODER 编码

MS_FILTER_DECODER 解码

5.1.2. MSFilterMethod

定义 FILTER 的函数列表，为一次性调用，如果 FILTER 上的 5 个主要函数指针不能满足需求的时候使用。

字段名	类型	描述
id	Int	函数 ID
method	MSFilterMethodFunc	函数方法

5.1.3. MBLK_T

MBLK_T 的主要作用是将一系列的数据流组装在一起,形成一个大的数据流,每个节点中的 BUF,又有指向 BUF 头和尾的指针

字段名	类型	长度	描述
b_prev	struct msgb *	Sizeof(msgb)	前一个数据包

b_next	struct msgb *	Sizeof(msgb)	后一个数据包
b_cont	struct msgb *	Sizeof(msgb)	
b_datap	struct datab *	Sizeof(datab)	数据
b_rptr	unsigned char *		首指针
b_wptr	unsigned char *		尾指针

5.1.4. MSFilterDesc

FILTER 的函数指针定义,下面这个结构为 FILTER 的核心描述，所有的 FILTER 的方法和属性都在这里定义。

字段名	类型	描述
id	MSFilterId	filter 的 ID 号,为唯一的 ID,
name	const char *	filter 的名称
text	const char *	描述,这个 FILTER 是实现什么功能
category	MSFilterCategory	类别,定义了编码,解码或者是其它
enc_fmt	const char *	必须被设置,在编解码的 FILTER 中
ninputs	int	定义输入的数字,如果是是播放的 FILTER 则为 0,如果是录音则为 1
noutputs	int	同上面的输入描述
init	MSFilterFunc	每个 FILTER 初始化的函数指针
preprocess	MSFilterFunc	在被调用前的函数指针
process	MSFilterFunc	过程的函数处理
postprocess	MSFilterFunc	处理完成后的指针
uninit	MSFilterFunc	结束,释放资源
methods	MSFilterMethod *	一些可选的函数指针,当上述函数指针不能处理的时候,可以通过该指针来实现

5.1.5. MSFilter

FILTER的完整定义

字段名	类型	描述
desc	MSFilterDesc *	函数指针
inputs	MSQueue **	输入的队列
outputs	MSQueue **	输出的队列
notify	MSFilterNotifyFunc	注册回调函数
ticker	struct _MSTicker *	调度,执行列表上的 FILTER,为一个线程

上面二个数据结构完整的定义了一个 FILTER，但是我们在执行过程中则需要通过数据结构来将这些 FILTER 串连起来。该结构体通过 create/link/unlink/destroy 方法来管理和维护。

5.1.6. MSConnectionPoint

本来上面的定义就可以完全的描述和管理好所有的 FILTER 了，但是在最新的版本中又提供了一种数据结构来管理 FILTER 的流程图。

字段名	类型	描述
filter	MSFilter *	Filter 对象
pin	int	FILTER 的输出,可以有多个

5.2. 传输数据结构

字段名	类型	描述
type	int	包类型 ,语音或者是视频
clock_rate	int	RTP 的时钟频率 如 8000,4400 等
bits_per_sample	char	连续音频数据
mime_type	char *	actually the submime, ex: pcm, pcma, gsm
channels	int	通道数
recv_fmtp	char *	用于 DTMF 数字信号、电话音和电话信号的 RTP 负载格式????
send_fmtp	char *	见注释

如果在 SDP 消息中包含有 a=fmtp 字段，则表示发送方有能力接受 DTMF（events 0 through 15），拨号和回铃音。

例如：若 payload-type 为 100， 则 a=fmtp:100 0-15,66,70

当接受方在 invite 请求中收到 a=fmtp 信息，

如果接受方不接受其中的任何一种信息，则在响应消息中不包含 a=fmtp 信息；

如果接受方不支持 a=fmtp 中的其中一种（如不支持 66，70），则在响应的消息中只包含有 1-15 字段；

6. API 描述

6.1. 传输 API

ortp_init(); 主要初始化SOCKET库文件,加载一个配置信息.
ortp_set_log_level_mask:设置LOG输出方式.
rtp_profile_set_payload:设置传输编码
rtp_profile_clone_full:拷贝一个配置环境
ortp_ev_queue_new:创建一个事件队列
rtp_session_register_event_queue:注册事件队列,.
rtp_session_compute_recv_bandwidth:显示接收的带宽,在不同的编码下可以比较性能.

rtp_session_compute_send_bandwidth :发送时的带宽.
rtp_profile_destroy() 释放环境变量.

6.2.语音控制 API

audio_stream_start_with_files;播放文件
audio_stream_play; 播放流文件
audio_stream_record录音;

audio_stream_play_received_dtmfs播放DTMF的声音
audio_stream_enable_echo_limiter 激活回声限制

audio_stream_set_mic_gain设置MIC的阀值
audio_stream_enable_noise_gate激活噪音处理
audio_stream_enable_equalizer 平衡处理，没有研究过？
audio_stream_stop 停止处理
ring_start 响铃
ring_stop 停止震铃
audio_stream_send_dtmf 发送DTMF
audio_stream_set_default_card设置默认的声卡
audio_stream_start_full 开始一个音频通道

6.3.视频控制 API

6.4.编/解码 API

6.5.FILTER 管理 API

注册插件.具体见插件的扩展流程
void ms_filter_register(MSFilterDesc *desc);

根据一个名称得到 FILTER 的编码函数指针,如果失败则返回空指针.如 PCMU,PCMA,SPEEX,GXM
MSFilterDesc * ms_filter_get_encoder(const char *mime);

根据一个函数名得到 FILTER 编码的函数指针,如果失败则返回空
MSFilterDesc * ms_filter_get_decoder(const char *mime);

根据一个名称创建一个 FILTER,空则表示失败
```
MSFilter * ms_filter_create_encoder(const char *mime);
```

根据一个名称创建一个解码的 FILTER
```
MSFilter * ms_filter_create_decoder(const char *mime);
```

检查是否支持该名称的编解码
```
bool_t ms_filter_codec_supported(const char *mime);
```

根据 ID 来创建一个 FILTER
```
MSFilter *ms_filter_new(MSFilterId id);
```

创建一个编码根据 FILTER 名称
```
MSFilter *ms_filter_new_from_name(const char *name);
```

创建一个 FILTER 根据 FILTER 函数,一般用于内部描述.
```
MSFilter *ms_filter_new_from_desc(MSFilterDesc *desc);
```

将 FILTER 之间关联起来,连接一个输出的插脚到另外一个 FILTER 输入的插脚,将一个 FILTER 的数据输出流转移到数据的输入流中.
输出数据流的 FILTER,输出的索引,输入流的 FILTER 对象,输出的索引,0 成功,-1 表示失败
```
int ms_filter_link(MSFilter *f1, int pin1, MSFilter *f2, int pin2);
```

解除 FILTER 之间的流程关系,并停止 FILTER 的工作
```
int ms_filter_unlink(MSFilter *f1, int pin1, MSFilter *f2, int pin2);
```

调用 FILTER 上的设置或者得到设置的方法.F 表示该 FILTER,ID 表示该 FILTER 的 ID,ARG 则表示该 FILTER 该方法所需要带的参数.
```
int ms_filter_call_method(MSFilter *f, unsigned int id, void *arg);
```

调用 FILTER 的设置方法,该函数常用于无数据传入出的 FILTER 方法.
```
int ms_filter_call_method_noarg(MSFilter *f, unsigned int id);
```

在 FILTER 上设置一个 CALLBACK 函数.
```
void ms_filter_set_notify_callback(MSFilter *f, MSFilterNotifyFunc fn, void *userdata);
```

得到 FILTER 的 ID 号
```
MSFilterId ms_filter_get_id(MSFilter *f);
```

通过一个 FILTER 得到该 FILTER 图上邻近的 FILTER,当调用者不需要的时候由其释放.
```
MSList * ms_filter_find_neighbours(MSFilter *me);
```

释放 FILTER 对象

```
void ms_filter_destroy(MSFilter *f);
```

初始化一个连接器,该方法为后期提供.

```
void ms_connection_helper_start(MSConnectionHelper *h);
```

7. MS 与 SIP 集成

1， **注册**

目的:通过 SIP 协议注册到,服务器上.(这个应该和语音模块关系不大)

个人理解:由于我们的服务器可能采用第三方的商业 IPPBX,而我们的软终端肯定要注册到自己的域服务器,需不需要到 IPPBX 上再注册,这可能和对方的实现有关.

2， **发起呼叫**

目的:通过在线列表呼叫一个软终端或者桌面电话或者移动电话.

个人理解:在这里 SIP 协议应该提供一个简单的 API 来让 MS 部分来生成 SDP 包,同时提供从 SDP 取编码方案的 API.也提供协商成功或者失败的控制.在这里应该不关注是 INVITE 还是 ACK 包的处理.

1,发送呼叫请求.生成 SDP 包

2,返回成功或者失败

3,如果成功则取 SDP 包,判断

4,开始连接媒体流

3， 取 SIP 协议中 SDP 包的信息

目的:取 SDP 包中的编码个数,和编码的具体描述等.

个人理解:能提供一个 API

4， 组装 SDP 包

目的:让对端知道本端具有什么样的编码格式.

个人理解:也需要一个封装好的 API

8. MS 提供给界面控制的函数

1， 编码设置

编码设置: 名称,采样频率(HZ),状态,最小速率,参数设置

2， 声音控制

播放设备,捕获,铃声

3， 视频控制

输入设备,视频大小

4， 声卡的检测

5， 摄像头的检测

6， 噪音控制

7， 回音控制

8, 语音通道的控制
9, 暂停通话
10, 停止通话
11, 注销

9. MS 中其它描述