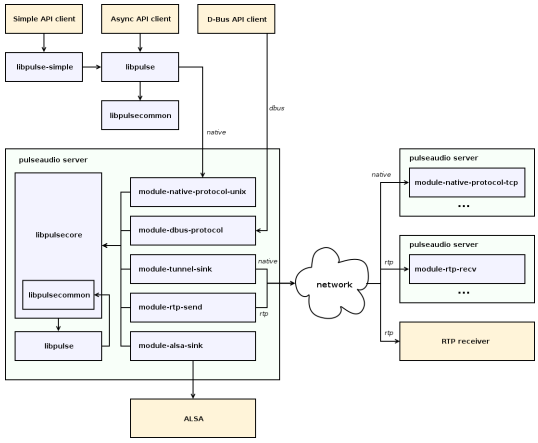
「音频采播」基于PulseAudio的Linux音频功能解析

一、背景

PulseAudio是Linux系统中通用的音频服务器，预装于大部分的Linux发行版中，因此系统兼容性较好。其位于Linux内核级的ALSA (Advanced Linux Sound Architecture)之上，设计目标包括硬件抽象、易于调用、灵活性好、可拓展等，整体框图如下所示：



在Webrtc的开源代码库中，linux端设备采集播放实现有两套方案，即ALSA和PulseAudio。本文将基于webrtc的audio_device框架以及实践过程中总结的经验，重点阐述以下两个方面：

a. 如何实现音频采集和播放功能；

b. 如何实现设备枚举、音量控制和音频事件监听这三种设备控制功能。

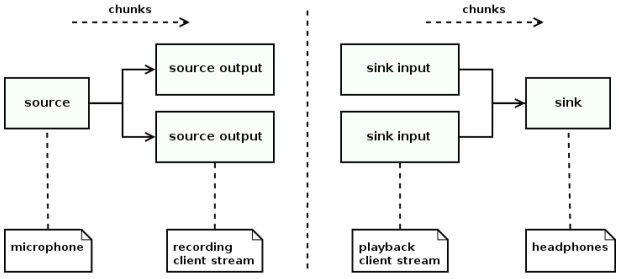
二、PulseAudio基本概念

在具体阐述功能实现细节之前，有必要先简单解释下PulseAudio中的基本概念，以期对整体音频数据从采集到播放的通路，和PulseAudio中几个重要角色有大致的了解，这对后文内容的理解有一定帮助。

2.1 设备与流

> 概念：source、source output、sink、sink input

下图表示的是从麦克风采集到数据，经过采集流通路，传递到播放流通路，最后从播放设备播放数据的完整链路。从虚线连接中可以清楚看出这几个概念之间的联系，即source代表采集设备，source output代表采集流，sink input代表播放流，sink代表播放设备，了解这层关系是非常重要的。



以具体的Linux笔记本操作举例。

① 利用pacmd工具，输入list-sources可以枚举出所有采集设备及其属性，一个物理声卡可以对应多个source，一个source可以对应多个port。类似的，输入list-sinks可以查看播放设备相关信息。【例】在ThinkPad X1的card 1默认profile中，包含着"HDMI/DisplayPort 1"，"HDMI/DisplayPort 2"，"HDMI/DisplayPort 3"以及"Speaker+Headphones"四种source；而针对最后一种source，其中又包含着Speaker和Headphones两种port，分别对应内置扬声器和3.5mm耳机播放孔。

② 输入list-source-outputs可以枚举数所有采集流相关信息，当没有应用在占用采集设备时，枚举数量为0。类似的，如果通过网页打开一个视频并且播放，输入list-sink-inputs就可以查询到播放流的相关信息。

概念	含义	单元类型	详细说明	呈现方式

Source	输入设备	active	每一个输入硬件设备都会自动创建一个source；其生产sample chunks，并且提供给source input	有输入设备时	
Source output	采集流	passive	应用开启采集流的时候会自动创建；被source驱动	录音 **[谷歌在线录音]**	
Sink	输出设备	active	每一个输出硬件设备都会自动创建一个sink；其消费来自sink input的sample chunks	有输出设备时	
Sink input	播放流	passive	应用开启播放流时候会自动创建	播放声音 **[网页打开视频播放]**	

2.2 其他重要角色

> 概念：mainloop、context、operation、lock

PulseAudio基于C&S架构，开发者调用API扮演者Client的角色，而系统则以**mainloop**作为Server。

常用的两套API分别是Simple API和Asynchronous API。前者是异步封装的同步调用，仅能满足简单的采集和播放需求，因此若要实现复杂功能，通常使用后者的异步调用。这涉及到另一个概念**context**，其作为异步调用中与服务端沟通的桥梁，会出现在各种异步调用函数形参中。

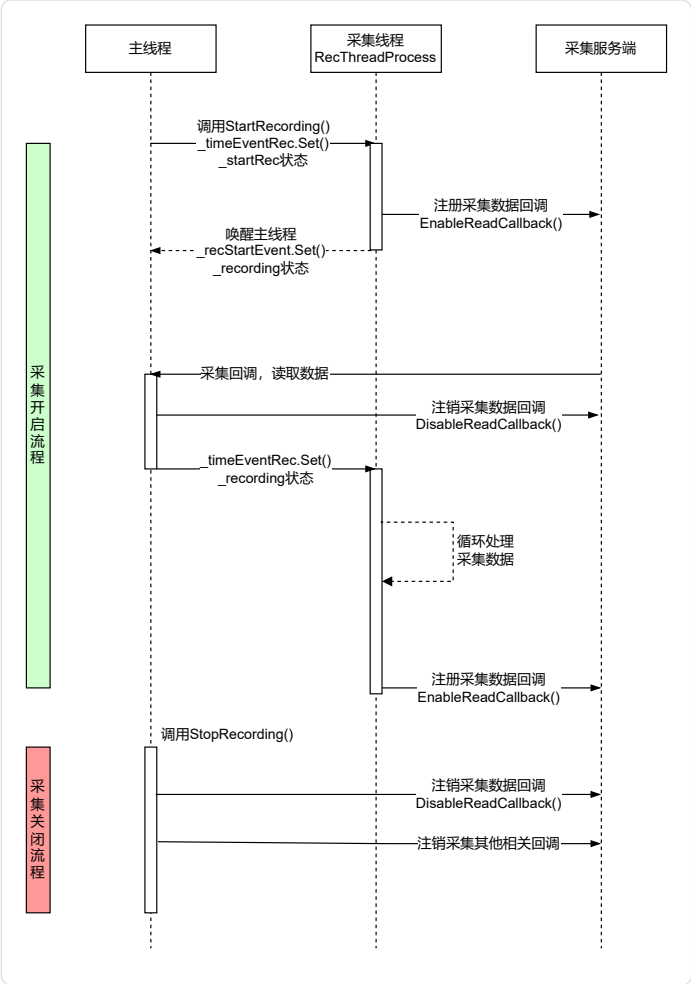
此外，异步调用中还有两个常见的概念分别是**operation**和**lock**。前者在诸如查询设备信息、流信息、server信息等带context函数的异步调用中作为返回值，一个很大用处就是将异步转同步处理，但是注意其引用属性，每次使用完需要unref；后者是PulseAudio中的递归锁，由于maininloop, context, stream和operation等资源不支持并发调用，因此每次对资源的使用前后需要上锁和解锁，此外，该锁不支持在mainloop中调用，否则会触发crash。

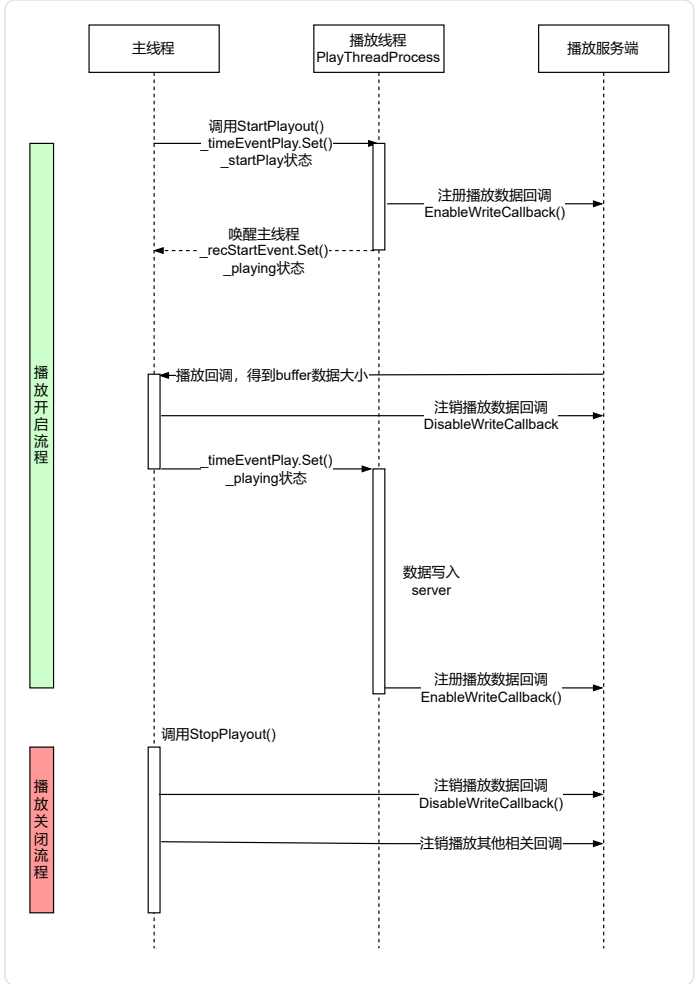
```
// 上锁
pa_threaded_mainloop_lock(_paMainloop);

// 异步转同步
pa_operation* paOperation = pa_context_get_source_info_by_name(_paContext, nam
while (pa_operation_get_state(paOperation) == PA_OPERATION_RUNNING) {
    pa_threaded_mainloop_wait(_paMainloop);
}
pa_operation_unref(paOperation);

// 解锁
pa_threaded_mainloop_lock(_paMainloop);
```

三、采集与播放实现详解





Webrtc的采集和播放的实现需要分别创建采集线程和播放线程，与应用设备线程以及服务器线程之间交互的时序图如上，其中起到同步线程的工具是rtc::Event，具体作用如下：

- _timeEventRec：从主线程切到采集线程
- _timeEventPlay：从主线程切到播放线程
- _recStartEvent：从采集线程切到主线程
- _playStartEvent：从播放线程切到主线程

上述流程图给出的是大概框架，若需要了解其中具体执行内容，请看下面三小节。

3.1 采集与播放三步走策略

采集与播放的整体流程类似，启动可分为init和start两步，而播放只有stop一步，下面以这三步为出发点，针对每一步的主要工作展开叙述：

(1) 初始化：主要负责相关参数的设置和流的创建

- 设置采样率、通道数、数据格式、流相关参数、buffer相关参数
- 创建流、申请buffer空间、注册流状态回调

(2) 启动：主要负责流资源准备和数据传输控制

- 准备流程：流连接到设备、注册采集/播放回调
- 运行流程：取数据、用数据

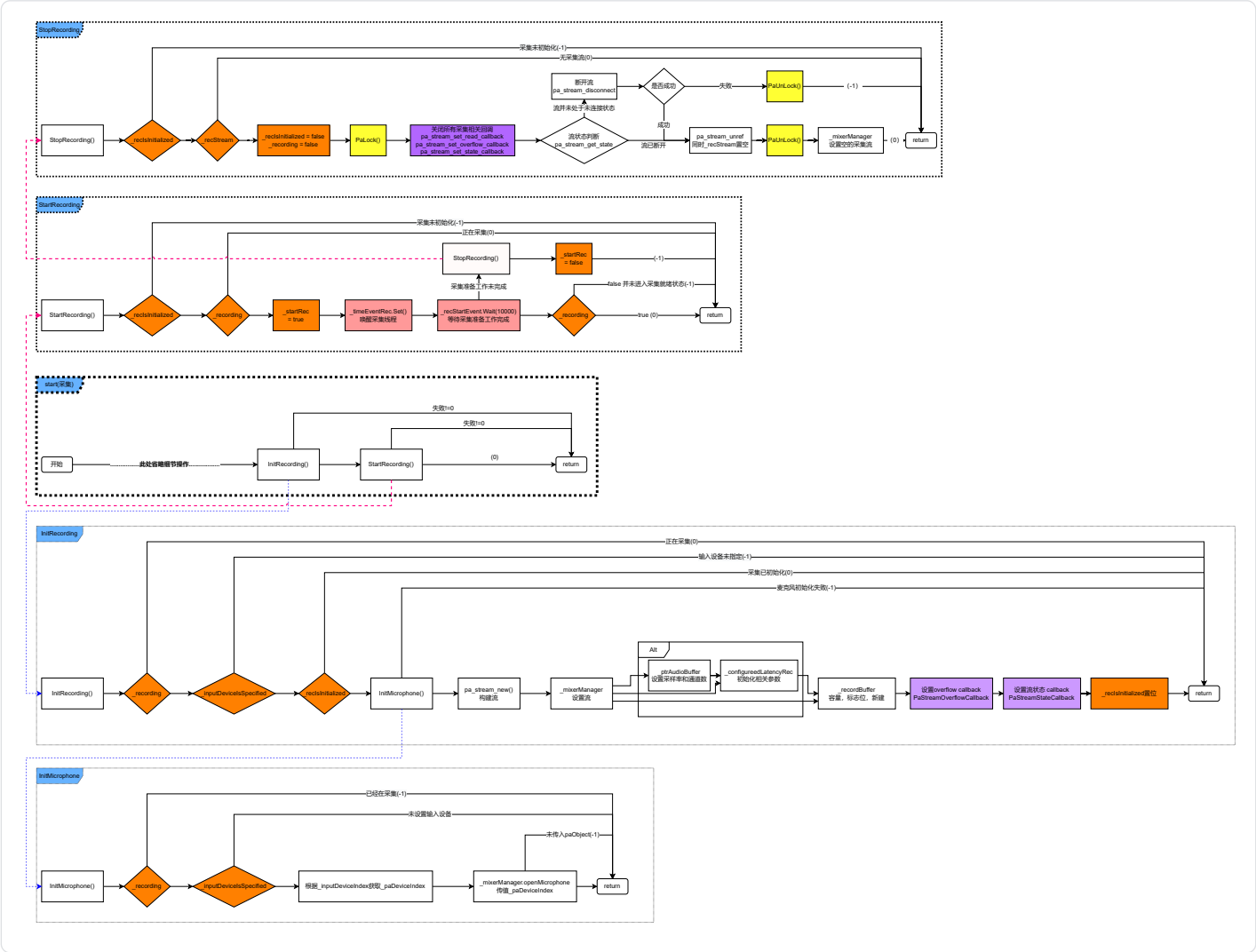
(3) 停止：主要负责资源清理

- 重置状态位、注销回调、断开流、注销流、清除buffer

3.2 设备线程实现流程图

> 图例：Mainloop、Callback、状态标志、锁、回调相关、线程控制

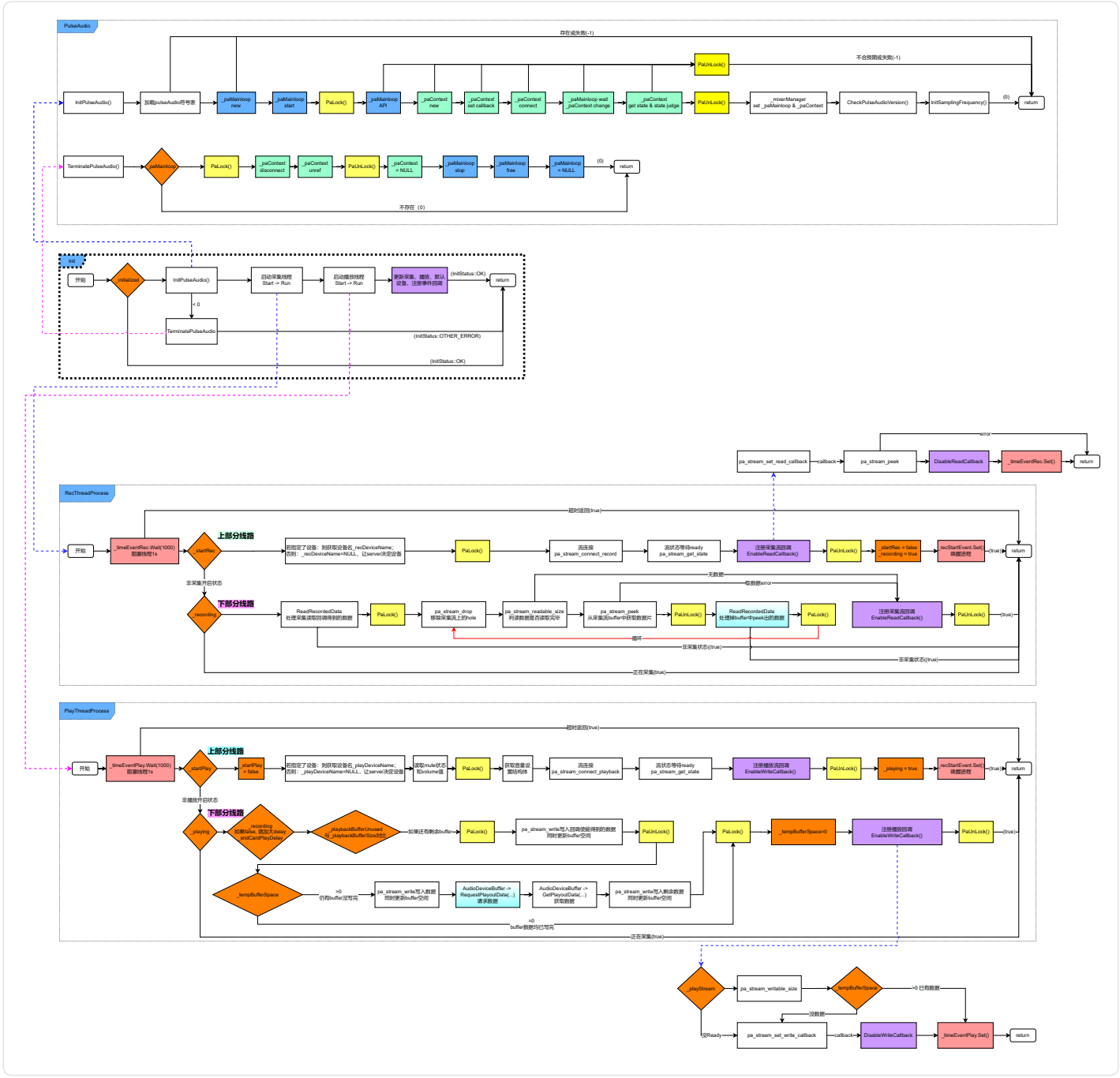
以采集为例，从下图中部加粗框图可以看到，start包含的初始化和启动分为initRecording和StartRecording两部分，具体执行上一节阐述的内容以及标志位的设置。如果StartRecording失败了，会调用StopRecording执行清理工作。



3.3 采集播放线程实现流程图

> 图例：Mainloop、Callback、状态标志、锁、回调相关、线程控制

由于采集和播放线程是在整个设备模块的初始化中建立，为了方便理解，引入设备线程中执行的加粗框图部分。从图中可以看出，采集和播放线程的执行任务均有两条并行线路，上部分线路由_startRec和_startPlay标志位控制，设备线程的StartRecording和StartPlayout执行过程会触发这条线路。而之后整体运转靠系统回调驱动，走的是下部分线路。



四、其他功能实现参考

在实际使用过程中，我们的需求往往不会满足于仅实现采集和播放，还需要具备一定的控制设备能力，比如能够显示系统支持使用的设备、能够调节采集的音量或播放的音量大小、能够自主切换麦克风或播放器、能够感知设备的插拔等，这就是本章内容阐述的重点。

4.1 实现设备枚举

常用的设备枚举功能可分为获取默认设备和当前使用设备、获取设备和流属性信息、获取设备数量、主动设置当前使用设备，通过组合调用API的具体实现如下：

(1) 获取默认设备和当前使用设备

如果需要获取默认设备，使用的API为pa_context_get_server_info，在该函数的回调中可以访问服务端信息pa_server_info，其中包含的变量default_sink_name和default_source_name便分别对应着系统默认播放设备和默认采集设备的名字。

如果需要获取当前使用设备，这依赖于应用自身对设备的存储结构，而无法通过系统API调用得到结果。

(2) 获取设备和流属性信息

获取设备和流属性信息的方法有几种，以采集为例，可以调用的API如下。前两个方法都比较常用，第三种方法适合用于获取设备数量。相比于设备信息，流信息的获取有两个API，即无法通过流的name去实现。

- pa_context_get_source_info_by_index：通过唯一值index属性指定访问设备
- pa_context_get_source_info_by_name：通过唯一值name属性指定访问设备
- pa_context_get_source_info_list：系统自动根据设备数量触发对应次数的回调

设备的属性信息中，通常我们需要知道设备的index，name，vendor id，product id，transport type(设备接口类型)等，前四点信息能直接从系统获取，但transport type的获取需要多做一些工作，详见Linux音频设备Transport判断方法。

(3) 获取设备数量

上一点中有提及，pa_context_get_xxx_info_list类的函数时候用于枚举所有设备，因此只需要全局设定一个计数器，每次触发回调的时候将计数器加一，循环结束即可得到设备数量信息。

但考虑到该功能的使用频次高，为避免频繁的系统调用产生如卡死之类的问题，实际开发中需要存一个设备表，该表在有设备变更的情况下进行刷新，而获取设备数量的功能简化为返回设备列表的大小即可。

(4) 主动设置当前使用设备

主动设置当前使用的采集设备或播放设备，需要根据设备的name分别调用pa_stream_connect_record和pa_stream_connect_playback实现。若要实现此功能，需要注意的是，WebRTC的源码中两处需要修改：

a. pa_stream_connect_xxx：不要传入NULL作为设备名

b. pa_stream_flags_t：streamFlag加上PA_STREAM_DONT_MOVE标志位

4.2 实现音量控制

音量控制可以是设备的音量，也可以是流的音量。以采集为例：

- pa_context_set_source_volume_by_index：通过index设置采集设备音量
- pa_context_set_source_volume_by_name：通过name设置采集设备音量
- pa_context_set_source_mute_by_index：通过index设置采集设备mute状态
- pa_context_set_source_mute_by_name：通过name设置采集设备mute状态

类似于设备信息获取，流的控制无法指定name，只能通过指定index来实现。

4.3 实现音频事件处理

常用的音频事件主要分为**设备插入、设备拔出、默认设备变化和音量变化**这四种。

首先可以通过两个系统API即pa_context_set_subscribe_callback和pa_context_subscribe的配合，实现系统事件的订阅，示例代码如下，可放置于设备模块初始化的最后调用。

```
// 上锁
pa_threaded_mainloop_lock(_paMainloop);

// 事件订阅
pa_operation* paOperation = NULL;
pa_context_set_subscribe_callback(_paContext, PaContextSubcribeCallback, this)
if (!(paOperation = pa_context_subscribe(
              _paContext,
              (pa_subscription_mask_t)(PA_SUBSCRIPTION_MASK_SINK | PA_SUBSCRIF
                                       PA_SUBSCRIPTION_MASK_SERVER | PA_SUBSCF
              NULL, NULL))) {
}
if (paOperation != NULL){
    pa_operation_unref(paOperation);
}


// 解锁
pa_threaded_mainloop_lock(_paMainloop);
```

上述代码保证应用可以收到系统上报的相关事件，注意代码中有一个回调**PaContextSubcribeCallback**，其作用在于具体逻辑处理的分发。

```
void xxxxx::PaContextSubcribeCallback(pa_context* c, const pa_subscription_eve
```

```
                                                uint32_t paIndex, void*
    switch (t & PA_SUBSCRIPTION_EVENT_FACILITY_MASK) {
        case PA_SUBSCRIPTION_EVENT_SERVER: {
            pa_operation* paOperation = NULL;
            if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK) == PA_SUBSCRIPTION_EVENT
                if (!(paOperation = pa_context_get_server_info(c, PaDefaultCha
                    ... // 默认设备变化的处理逻辑
                    return;
                }
                pa_operation_unref(paOperation);
            }
        } break;

        case ... // 其他事件处理

        default:
            break;
    }
}
```

在本节的最后，总结部分音频事件处理方式，若对象改为流的处理，组合方法类似。

	A	B	C
1	**事件**	**event_type**	**mask**
2	**采集设备插入事件**	PA_SUBSCRIPTION_EVENT_SOURCE	PA_SUBSCRIPTION_EVENT_NEW
3	**采集设备拔出事件**	PA_SUBSCRIPTION_EVENT_SOURCE	PA_SUBSCRIPTION_EVENT_REMOVE
4	**采集设备音量变化、静音事件**	PA_SUBSCRIPTION_EVENT_SOURCE	PA_SUBSCRIPTION_EVENT_CHANGE
5	**播放设备插入事件**	PA_SUBSCRIPTION_EVENT_SINK	PA_SUBSCRIPTION_EVENT_NEW
6	**播放设备拔出事件**	PA_SUBSCRIPTION_EVENT_SINK	PA_SUBSCRIPTION_EVENT_REMOVE
7	**播放设备音量变化、静音事件**	PA_SUBSCRIPTION_EVENT_SINK	PA_SUBSCRIPTION_EVENT_CHANGE
8	**系统默认采集\播放设备变化事件**	PA_SUBSCRIPTION_EVENT_SERVER	PA_SUBSCRIPTION_EVENT_CHANGE
9	...	...	...

五、优质参考资料

官网：freedesktop.org: PulseAudio

开发接口文档：PulseAudio 16.0

深入理解PulseAudio：PulseAudio under the hood

Jan Newmarch书籍：LinuxSoundProgramming.pdf

Archlinux wiki：基础内容　Examples　配置相关　Troubleshooting(中文 英文)