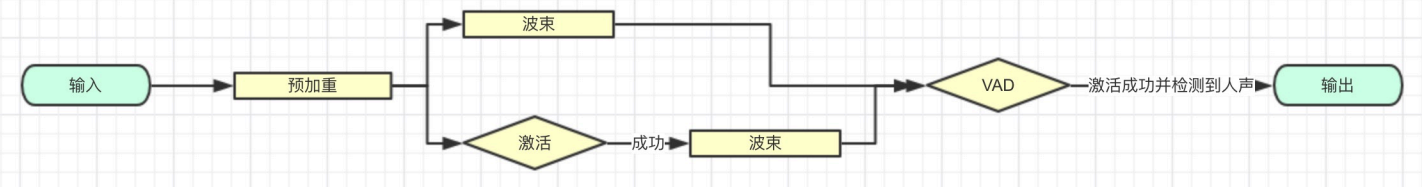
单独激活模块介绍&简单应用示例

一、简介

本章主要为你了解单独激活模块。单独激活模块拟为开发者提供一个通用、可靠、便捷的语音唤醒功能。下面看一张多路语音唤醒流程图:



首先接收外部输入语音，经过前期预处理以提高高频部分能量并传入激活算法，同时对多路语音做波数形成，经过波数和激活算法处理后，会输出单路语音和激活状态信息，然后做VAD(Voice Activity Detection)人声检测，当检测到人声并根据激活状态判断是否激活和对外输出VAD。此次相比之前的BlackSiren模块，去掉了一些不必要的语音信号处理和多进程，使CPU保持在7%到8%，内存消耗50M左右，并提供Java接口以降低部分学习成本。

二、下载和编译

克隆仓库地址(内部)

```
git clone ssh://username@s.rokid-inc.com:29418/open-platform/client/vsys-activation
```

Ubuntu/Linux编译

仅支持x86_64架构

从源码树的根路径执行:

```
make
```

Android编译

支持armeabi-v7a、arm64-v8a架构，兼容所有Android版本

从源码树的根路径执行:

```
export ANDROID_SDK_HOME=<Android SDK路径>
make android
```

运行Linux示例程序

main.cpp

从源码树的根路径执行:

```
make test
./run.sh

激活成功后输出激活词信息和和其它状态事件
02-29 12:54:55.001  134986 I activation: 若琪     3200      10240      24284.724609
02-29 12:54:55.001  134986 I activation: --------------------------------     105
02-29 12:54:55.001  134986 I activation: --------------------------------     102
02-29 12:54:55.002  134986 I activation: --------------------------------     100
02-29 12:54:55.002  134986 I activation: --------------------------------     106
02-29 12:54:55.002  134986 I activation: --------------------------------     107
02-29 12:54:55.002  134986 I activation: --------------------------------     107
```

运行Android示例程序

android/example

选择一部armeabi-v7a架构，Android 4.4及以上版本手机。

从源码树的根路径执行:

```
adb install -r android/example/RKVoiceActivationExample.apk
adb shell am start com.rokid.openvoice.example/.MainActivity
adb logcat -s VoiceActivation.example

Android示例程序获取真实麦克风数据，需要对着手机说"若琪"才能看到激活效果

D/VoiceActivation.example(16305): onVoiceTrigger: word = 若琪, start = 3200, end =
10720, energy = 1547.4436
D/VoiceActivation.example(16305): onAwake
D/VoiceActivation.example(16305): onVadStart: energy = 0.01, energy threshold = 92
1.20917
D/VoiceActivation.example(16305): onVadData: data is 26880 bytes
D/VoiceActivation.example(16305): onVadData: data is 640 bytes
```

三、基本使用

Linux使用介绍

这里采用从文件获取语音数据，数据格式为16000，16bit, 1通道。

准备工作

源码根路径下创建main.cpp，导入include目录下所有头文件。其中 `vsys_activation.h` 为接口定义， `vsys_types.h` 为相关参数和数据结构定义。

创建激活算法句柄

```
activation_param_t param;
memset(&param, 0, sizeof(activation_param_t));
param.sample_rate = AUDIO_SAMPLT_RATE_16K;
param.sample_size_bits = AUDIO_FORMAT_ENCODING_PCM_FLOAT;
param.num_mics = 1;
param.num_channels = CHANNEL_NUM;

VsysActivationInst handle =  VsysActivation_Create(&param, "./thirdparty/model/dnn
", true);
```

注册Callback

```
static void my_voice_event_callback(voice_event_t* voice_event, void* token){
    //
}

VsysActivation_RegisterVoiceEventCallback(handle, my_voice_event_callback, nullptr
);
```

获取语音数据

```
std::ifstream pcm_in("./data/sounds/pcm_16k_32f_1.pcm", std::ios::in | std::ios::b
inary);
```

做激活算法处理

```
while(pcm_in.good()){
    pcm_in.read(buff, FRAME_SIZE * CHANNEL_NUM * sizeof(AUDIO_TYPE));
    VsysActivation_Process(handle, (uint8_t *)buff, FRAME_SIZE * CHANNEL_NUM * siz
eof(AUDIO_TYPE));
}
```

释放激活算法句柄

```
VsysActivation_Free(handle);
```

注：完整代码见源码树根路径main.cpp文件

模型文件地址

```
thirdparty/model/dnn
```

添加编译依赖

在你的编译命令中加入如下选项：

```
-Lout/libs -lrkvacti
```

添加其他编译依赖

在你的编译命令中加入如下选项：

```
-Lthirdparty/libs/dnn/linux-x86_64 -lr2vt5 -lr2ssp -lztvad -lmkl_intel_lp64 -lmkl_
sequential -lmkl_core
```

完整编译命令

```
g++ main.cpp -std=c++11 -Iinclude -Isrc -Lout/libs -lrkvacti -Lthirdparty/libs/dnn
/linux-x86_64 -lr2ssp -lztvad -lr2vt5 -lmkl_core -lmkl_intel_lp64 -lmkl_sequential
```

Android使用介绍

准备工作

确保使用第二章介绍的方式编译Java API，然后使用eclipse IDE创建Android APP工程，复制源码根路径的 `out/libs` 下的目录和文件到APP工程 `libs` 目录，并复制 `thirdparty/model/dnn/workdir_cn` 目录到APP工程 `assets` 目录下。

复制模型文件

在运行时将 `assets` 目录下模型文件夹复制到 `sdcard` 下以获取模型文件夹相对路径

实现Callback接口

查看 `android/java/com/rokid/openvoice/VoiceActivation.java` 文件的Callback定义

创建激活算法对象&注册Callback

```
VoiceActivationBuilder builder = new VoiceActivationBuilder();
    // 设置输入pcm流的采样率，位宽
    // 麦克风数
    // 本地vad模式
    // 设备上workdir_cn所在目录(算法模块需要读取此目录下模型文件)
    // 单通道，通道高级参数不设置，全部忽略
    // 回调对象
    VoiceActivation va = builder
            .setSampleRate(16000)
            .setBitsPerSample(VoiceActivationBuilder.AudioFormat.ENCODING_PCM_16BI
T)
            .setChannelNumber(1)
            .enableVad(true)
            .setBasePath(Environment.getExternalStorageDirectory().getPath())
            .addMic(0, 0, 0, 0, 0).setMicParamMask(0)
            .setCallback(this).build();
```

获取语音数据

```
int min = AudioRecord.getMinBufferSize(16000, 16, 2);
AudioRecord ar = new AudioRecord(MediaRecorder.AudioSource.VOICE_COMMUNICATION, 16
000, 16, 2, min * 5);
```

做激活算法处理

```
byte[] buf = new byte[min];
int c;

ar.startRecording();
while (true) {
    c = ar.read(buf, 0, buf.length);
    if (c > 0) {
        va.process(buf, 0, c);
    }
}
```

四、详细介绍

1. 音频格式

主要支持16bit、24bit、32bit、单精度浮点，请参见 `audio_format_t` 定义

2. 采样率

只支持16000采样

3. 通道数

通道没有限制，可以输入一路语音数据或多路麦克风阵列数据。如果是多路麦克风数据，需要对麦克风做精确配置，请参见 `mic_param_t` 定义，暂不支持线麦。

五、多路语音参数配置

六、接口说明

接口主要分为六类：创建/销毁句柄、注册回掉、激活处理函数、激活/休眠控制、激活算法配置、激活词操作。

```
/**
 * 创建激活算法句柄
 *
 * @param param 激活算法配置参数(包括：采样率、比特率、麦克风数、通道数、麦克风精确配置、mask)
 * @param path 模型文件相对路径
 * @param vad_enable 是否开启本地VAD
 *
 * @return 成功:激活算法句柄；失败:0
 */
VsysActivationInst VsysActivation_Create(const activation_param_t* param, const ch
ar* path, bool vad_enable);

/**
 * 释放激活算法句柄
 *
 * @param handle 激活算法句柄
 */
void VsysActivation_Free(VsysActivationInst handle);

/**
 * 注册激活事件回调
 *
 * @param handle 激活算法句柄
 * @param callback 回调函数指针
 * @param token 附加参数
 */
void VsysActivation_RegisterVoiceEventCallback(VsysActivationInst handle, voice_ev
ent_callback callback, void* token);

/**
 * 激活处理函数
 *
 * @param handle 激活算法句柄
 * @param input 音频裸数据，pcm格式
 * @param byte_size 音频流长度
```

```
 *
 * @return 成功:0；失败:-1
 */
int32_t VsysActivation_Process(VsysActivationInst handle, const uint8_t* input, co
nst size_t byte_size);

/**
 * 激活状态控制接口
 *
 * @param handle 激活算法句柄
 * @param action 控制意图
 *
 * @return 成功:0；失败:-1
 */
int32_t VsysActivation_Control(VsysActivationInst handle, active_action action);

/**
 * 激活算法配置
 *
 * @param handle 激活算法句柄
 * @param key 键
 * @param val 值
 *
 * @return 成功:0；失败:-1
 */
//int32_t VsysActivation_Config(VsysActivationInst handle, active_param key, const
 void* val);

/**
 * 添加激活词
 *
 * @param handle 激活算法句柄
 * @param vt_word 激活词信息
 *
 * @return 成功:0；失败:-1
 */
int32_t VsysActivation_AddVtWord(VsysActivationInst handle, const vt_word_t* vt_wo
rd);

/**
 * 删除激活词
 *
 * @param handle 激活算法句柄
 * @param word 激活词中文字符串，UTF-8编码
 *
 * @return 成功:0；失败:-1
 */
int32_t VsysActivation_RemoveVtWord(VsysActivationInst handle, const char* word);
```

```
/**
 * 查询激活词
 *
 * @param handle 激活算法句柄
 * @param vt_words_out 传入参数
 *
 * @return 成功:激活词个数；失败:-1
 */
int32_t VsysActivation_GetVtWords(VsysActivationInst handle, vt_word_t** vt_words_
out);
```