OpenSL ES

OpenSL ES (Open Sound Library for Embedded Systems)

OpenSL ES™ 是无授权费、跨平台、针对嵌入式系统精心优化的硬件音频加速API。它为嵌入式移动多媒体设备上的本地应用程序开发者提供标准化, 高性能,低响应时间的音频功能实现方法，并实现软/硬件音频性能的直接跨平台部署，降低执行难度，促进高级音频市场的发展。

简单来说，OpenSL ES就是嵌入式跨平台免费的音频处理库。

为什么要使用OpenSL ES

OpenGL ES能够播放与录制PCM音频。在Android SDK的Java接口中，如果需要录制PCM音频需要使用`AudioRecord`，而播放则需要使用`AudioTrack`。因此我们在C++中使用FFmpeg解码完成获取的PCM数据实际上也是可以拷贝给Java进行播放。

但是，如果使用Java实现，数据需要从native拷贝给Java的AudioTrack播放，而AudioTrack在播放时又会需要将音频数据从 Java 拷贝到 native 层 。如果希望减少拷贝，开发更加高效的 Android 音频应用，则建议使用 Android NDK 提供的 OpenSL ES API 接口，它支持在 native 层直接处理音频数据。这样就避免音频数据频繁在 native 层和 Java 层拷贝，提高效率 。

OpenSL ES 的使用

OpenSL ES 通过 Object 和 Interface使用，但是这里的Object和Interface 与Java的对象与接口并不相同。

- Object 可能会存在一个或者多个 Interface，官方为每一种 Object 都定义了一系列的 Interface；
- Object 对象提供了各种操作，如果希望使用该对象支持的功能函数，则必须通过其 GetInterface 函数拿到 Interface 接口，然后通过 Interface 来访问功能函数 。

OpenSL ES的基本开发流程

由于OpenSL ES是Native层提供的API，在使用前须注意添加头文件和链接选项，在需要用到OpenSL ES API的源文件中添加：

```
#include <SLES/OpenSLES.h>
#include <SLES/OpenSLES_Android.h>
```

而在CMakeLists.txt中需要链接:OpenSLES

```
target_link_libraries(
                       dnplayer
                       avfilter avformat    avcodec  avutil  swresample swscale rtmp
                         z
                         OpenSLES
                         android
                       log )
```

接下来播放的流程为：

1. 创建引擎对象
2. 设置混音器
3. 创建播放器
4. 开始播放
5. 停止播放

创建引擎对象

```
// 创建引擎engineObject
SLObjectItf engineObject = NULL;
SLresult result;
result = slCreateEngine(&engineObject, 0, NULL, 0, NULL, NULL);
if (SL_RESULT_SUCCESS != result) {
    return;
}
result = (*engineObject)->Realize(engineObject, SL_BOOLEAN_FALSE);
if (SL_RESULT_SUCCESS != result) {
    return;
}
// 获取引擎接口engineInterface
SLEngineItf engineInterface = NULL;
result = (*engineObject)->GetInterface(engineObject, SL_IID_ENGINE,
                                       &engineInterface);
if (SL_RESULT_SUCCESS != result) {
    return;
}
```

设置混音器

```
//通过引擎接口创建婚姻器
SLObjectItf outputMixObject = NULL;
result = (*engineInterface)->CreateOutputMix(engineInterface, &outputMixObject, 0, 0, 0);
if (SL_RESULT_SUCCESS != result) {
    return;
}

// 初始化混音器outputMixObject
result = (*outputMixObject)->Realize(outputMixObject, SL_BOOLEAN_FALSE);
if (SL_RESULT_SUCCESS != result) {
    return;
}
```

创建播放器

```
//创建buffer缓冲类型的队列作为数据定位器（获取播放数据） 2个缓冲区
SLDataLocator_AndroidSimpleBufferQueue android_queue =
{SL_DATALOCATOR_ANDROIDSIMPLEBUFFERQUEUE,2};
//pcm数据格式：pcm、声道数、采样率、采样位、容器大小、通道掩码(双声道)、字节序(小端)
SLDataFormat_PCM pcm = {SL_DATAFORMAT_PCM, 2, SL_SAMPLINGRATE_44_1,
SL_PCMSAMPLEFORMAT_FIXED_16,SL_PCMSAMPLEFORMAT_FIXED_16,SL_SPEAKER_FRONT_LEFT |
SL_SPEAKER_FRONT_RIGHT, SL_BYTEORDER_LITTLEENDIAN};

//数据源 （数据获取器+格式）
SLDataSource slDataSource = {&android_queue, &pcm};


//设置混音器
SLDataLocator_OutputMix outputMix = {SL_DATALOCATOR_OUTPUTMIX, outputMixObject};
SLDataSink audioSnk = {&outputMix, NULL};
//需要的接口
const SLInterfaceID ids[1] = {SL_IID_BUFFERQUEUE};
const SLboolean req[1] = {SL_BOOLEAN_TRUE};
SLObjectItf bqPlayerObject = NULL;
//创建播放器
(*engineInterface)->CreateAudioPlayer(engineInterface, &bqPlayerObject, &slDataSource,
                                      &audioSnk, 1,
                                      ids, req);
//初始化播放器
(*bqPlayerObject)->Realize(bqPlayerObject, SL_BOOLEAN_FALSE);
```

这里的配置有点多，但是实际上目的就是为了获得一个播放器对象，获得对象后，需要使用播放器的功能，则需要获取播放器对应的Interface接口。

开始播放

```
//获得播放数据队列操作接口
SLAndroidSimpleBufferQueueItf bqPlayerBufferQueue = NULL;
(*bqPlayerObject)->GetInterface(bqPlayerObject, SL_IID_BUFFERQUEUE,
                                &bqPlayerBufferQueue);
```

```
//设置回调（启动播放器后执行回调来获取数据并播放）
(*bqPlayerBufferQueue)->RegisterCallback(bqPlayerBufferQueue, bqPlayerCallback, this);

//获取播放状态接口
SLPlayItf bqPlayerInterface = NULL;
(*bqPlayerObject)->GetInterface(bqPlayerObject, SL_IID_PLAY, &bqPlayerInterface);
// 设置播放状态
(*bqPlayerInterface)->SetPlayState(bqPlayerInterface, SL_PLAYSTATE_PLAYING);

//需要手动调用一次播放回调
bqPlayerCallback(bqPlayerBufferQueue, this);

void bqPlayerCallback(SLAndroidSimpleBufferQueueItf bq, void *context) {
    AudioChannel *audioChannel = static_cast<AudioChannel *>(context);
    //... 获得播放数据 audioChannel->buffer 与数据长度 datalen
    if (datalen > 0) {
        //加入队列并播放
        (*bq)->Enqueue(bq, audioChannel->buffer, datalen);
    }
}
```

停止播放

```
    //设置停止状态
    if (bqPlayerInterface) {
        (*bqPlayerInterface)->SetPlayState(bqPlayerInterface, SL_PLAYSTATE_STOPPED);
        bqPlayerInterface = 0;
    }
    //销毁播放器
    if (bqPlayerObject) {
        (*bqPlayerObject)->Destroy(bqPlayerObject);
        bqPlayerObject = 0;
        bqPlayerBufferQueue = 0;
    }
    //销毁混音器
    if (outputMixObject) {
        (*outputMixObject)->Destroy(outputMixObject);
        outputMixObject = 0;
    }
    //销毁引擎
    if (engineObject) {
        (*engineObject)->Destroy(engineObject);
        engineObject = 0;
        engineInterface = 0;
    }
```