2 Clang 交叉编译

添加NDK编译环境变量

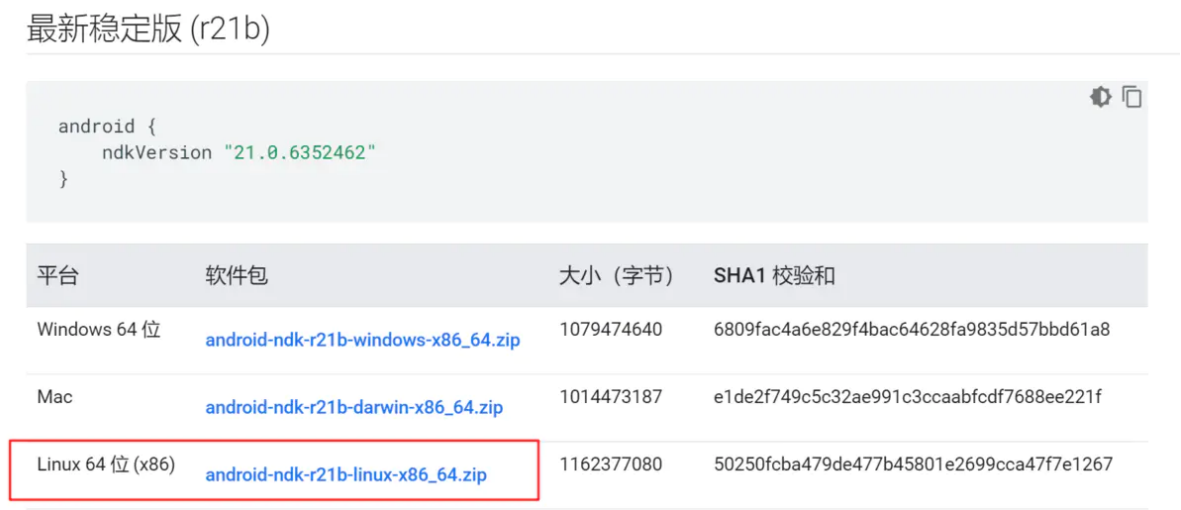
2.1 下载NDK

> wget https://dl.google.com/android/repository/android-ndk-r21d-linux-x86_64.zip

2.2 安装NDK

从下面的链接下载NDK，并解压：
https://developer.android.google.cn/ndk/downloads/

这里下载了 android-ndk-r21b，解压到 /home/temp/programs/android-ndk-r21b



最新稳定版 (r21b)

```
android {
    ndkVersion "21.0.6352462"
}
```

平台	软件包	大小（字节）	SHA1 校验和
Windows 64 位	android-ndk-r21b-windows-x86_64.zip	1079474640	6809fac4a6e829f4bac64628fa9835d57bbd61a8
Mac	android-ndk-r21b-darwin-x86_64.zip	1014473187	e1de2f749c5c32ae991c3ccaabfcdf7688ee221f
Linux 64 位 (x86)	android-ndk-r21b-linux-x86_64.zip	1162377080	50250fcba479de477b45801e2699cca47f7e1267

2.3 添加系统环境变量

> vim etc/profile
>
> export PATH=$PATH:/root/ndk/android-ndk-r21d
> export SYSROOT="$NDK/toolchains/llvm/prebuilt/linux-x86_64/sysroot/"
> export ANDROID_GCC="$NDK/toolchains/llvm/prebuilt/linux-x86_64/bin/x86_64-linux-android24-clang"

2.4 生效环境变量

source /etc/profile

```
$NDK/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android26-clang++
main.cpp -o hello
```

这里NDK用的是r19及以上的版本。

下载faac

> wget https://nchc.dl.sourceforge.net/project/faac/faac-src/faac-1.29/faac-1.29.9.2.tar.gz
>
> #下载完成后解压
>
> tar xvf faac-1.29.9.2.tar.gz
>
> #进入facc目录
>
> cd faac-1.29.9.2

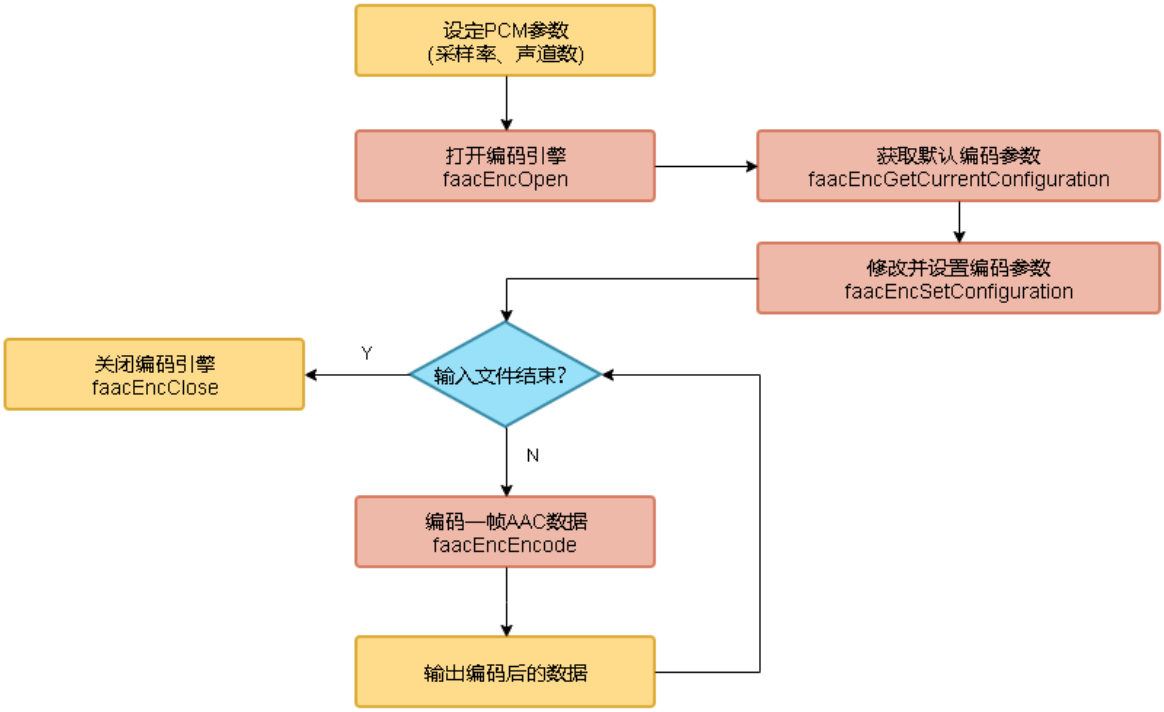
AAC全称为Advanced Audio Coding，目前比较主流的AAC开源编码器主要有Nero和Faac。接下来我们将使用Faac实现音频PCM至AAC的音频格式转换，并使用Emscripten编译成WebAssembly模块。

run.sh脚本内容

```
#!/bin/bash
PREFIX=$pwd/android/armeabi-v7a

TOOLCHAIN=$NDK/toolchains/llvm/prebuilt/linux-x86_64

export CFLAGS="--target=armv7-none-linux-androideabi21 --gcc-toolchain=$TOOLCHAIN
-g -DANDROID -fdata-sections -ffunction-sections -funwind-tables -fstack-
protector-strong -no-canonical-prefixes -fno-addrsig -march=armv7-a -mthumb -Wa,
--noexecstack -Wformat -Werror=format-security -Oz -DNDEBUG -fPIC "

export CC=$TOOLCHAIN/bin/armv7a-linux-androideabi21-clang

export CXX=$TOOLCHAIN/bin/armv7a-linux-androideabi21-clang++


./configure --prefix=$PREFIX --with-pic=yes --host=arm-linux-androideabi --
enable-shared=no --enable-static=yes  -with-sysroot=$TOOLCHAIN/sysroot
```

二、实现步骤

使用Faac实现音频编码，主要有以下步骤：



2.1 主要函数

- **faacEncOpen**

```
faacEncHandle FAACAPI faacEncOpen(unsigned long sampleRate,
    unsigned int numChannels,
    unsigned long *inputSamples,
    unsigned long *maxOutputBytes
);
```

变量名	变量含义
sampleRate	输入PCM的采样率。
numChannels	输入PCM的通道数。
inputSamples	编码一帧AAC所需要的字节数，打开编码器后获取，故声明时不需赋值。
maxOutputBytes	编码后的数据输出的最大长度。

- **faacEncEncode**

```
int FAACAPI faacEncEncode(faacEncHandle hEncoder,
    int32_t * inputBuffer,
    unsigned int samplesInput,
    unsigned char *outputBuffer,
    unsigned int bufferSize
);
```

变量名	变量含义
hEncoder	faacEncOpen返回的编码器句柄
inputBuffer	PCM缓冲区
samplesInput	faacEncOpen编码后的数据长度inputSamples，即PCM缓冲区长度
outputBuffer	编码后输出数据
bufferSize	输出数据的长度，对应faacEncOpen的maxOutputBytes

2.2 编码器参数

与Faac编码器相关的配置在faaccfg.h中声明。主要参数的含义如下：

```
// 生成的mpeg版本，如果需要录制MP4则设置为MPEG4，如果希望得到未封装的AAC裸流，则设置为MPEG2
// 0-MPEG4 1-MPEG2
unsigned int mpegVersion;

// AAC编码类型
// 1-MAIN 2-LOW 3-SSR 4-LTP
unsigned int aacObjectType;

// 是否允许一个通道为低频通道
// 0-NO 1-YES
unsigned int useLfe;

// 是否使用瞬时噪声定形(temporal noise shaping，TNS)滤波器
// 0-NO 1-YES
unsigned int useTns;

// AAC码率，可参考常见AAC码率，单位bps
unsigned long bitRate;

// AAC频宽
unsigned int bandWidth;

// AAC编码质量
// lower<100 default=100 higher>100
unsigned long quantqual;

// 输出的数据类型，RAW不带adts头部
// 0-RAW 1-ADTS
unsigned int outputFormat;

// 输入PCM数据类型
// PCM Sample Input Format
// 0    FAAC_INPUT_NULL         invalid, signifies a misconfigured config
// 1    FAAC_INPUT_16BIT        native endian 16bit
// 2    FAAC_INPUT_24BIT        native endian 24bit in 24 bits      (not
implemented)
// 3    FAAC_INPUT_32BIT        native endian 24bit in 32 bits      (DEFAULT)
// 4    FAAC_INPUT_FLOAT        32bit floating point
unsigned int inputFormat;
```

2.3 编码器初始化

```
unsigned long inputSample = 0;
unsigned long maxOutputBytes = 0;
faacEncHandle encoder;

EM_PORT_API(void) turn_on_encoder() {
    unsigned int numChannels = 1;
    unsigned long sampleRate = 8000;

    faacEncConfigurationPtr config;

    encoder = faacEncOpen(sampleRate, numChannels, &inputSample,
&maxOutputBytes);

    // EM_ASM_({
    //  console.log('inputSample', $0);
    //  console.log('maxOutputBytes', $1)
    // }, (unsigned int)inputSample, (unsigned int)maxOutputBytes);

    config = faacEncGetCurrentConfiguration(encoder);

    config->aacObjectType = LOW;
    config->useTns = 1;
    config->allowMidside = 1;
    config->bitRate = 8000;
    config->outputFormat = 1;
    config->inputFormat = FAAC_INPUT_16BIT;

    faacEncSetConfiguration(encoder, config);
}
```

在之前的Emscripten的介绍中，已经给出宏EM_PORT_API的定义。值得注意的是，因为inputSample、maxOutputBytes的数据类型是unsigned long，使用64位存储，为了避免C与JS进行数据交互时，发生内存不对齐的情况，此处将数据类型转为32位的unsigned int类型。

2.4 编码

```
EM_PORT_API(unsigned char*) pcm_2_aac(unsigned char* inputBuffer) {

    byteLength = 0;

    unsigned char* outputBuffer = (unsigned char*)malloc(maxOutputBytes);

    do {
        byteLength = faacEncEncode(encoder, (int32_t*)inputBuffer, inputSample,
outputBuffer, maxOutputBytes);
        if (byteLength > 0) break;
    } while (byteLength <= 0);

    return outputBuffer;
}
```

在这个函数中，使用了malloc为编码后的数据缓冲区outputBuffer动态分配内存空间，为了避免内存泄漏，在不需要outputBuffer时，需要手动将内存释放。因而增加以下函数，可在JS中调用函数进行释放。

```
EM_PORT_API(void) free_buf(void* buf) {
    free(buf);
}
```

三、如何在JS中使用

使用Emscripten编译生成WebAssembly模块和胶水代码，假设为faac.wasm与faac.js，加入到JS项目中。
下面是我自己写的一个由PCM转AAC的例子：

```
/**
 * PCM转AAC
 * @param {ArrayBuffer} buffer PCM数据，有符号16位
 */
pcm_2_aac(buffer) {
    var pcmBuf = new Uint8Array(buffer);

    // 创建PCM数据在HEAP中的指针变量
    var pcmPtr = Faac._malloc(pcmBuf.byteLength);
    Faac.HEAPU8.set(Array.from(pcmBuf), pcmPtr);

    /**
     * Faac._pcm_2_aac(inputBuffer)
     * @param {Number} inputBuffer PCM数组在HEAP中的首地址
     */
    var aacPtr = Faac._pcm_2_aac(pcmPtr);
    var byteLen = Faac._getByteLen();
    var arrBuf = Uint8Array.from(Faac.HEAPU8.subarray(aacPtr, aacPtr +
byteLen));

    // 清除缓存
    Faac._free(pcmPtr);
    Faac._free_buf(aacPtr);

    return arrBuf;
}
```

注意到我们是从HEAPU8中取出编码后的AAC数据的，此处的HEAP事实上是指C环境的整个内存空间。在调用该函数进行编码时，需要将大块的数据送入C环境下，此时我们可以在JS中分配内存并装入数据，然后将数据指针传入，调用C函数进行处理。这种做法借助了C的导出函数*malloc*/free实现的。

另外，HEAPU8实际上对应的数据类型是Uint8Array。

TMP直播，音频编码采用AAC时，需要把帧头的数据去掉。

第一个数据包，发送4个字节，前面两个是0xAF、0x00，我看有文章写的是这个0xAF的A代表的是AAC，说明如下：

> 0 = Linear PCM, platform endian
> 1 = ADPCM
> 2 = MP3
> 3 = Linear PCM, little endian
> 4 = Nellymoser 16 kHz mono
> 5 = Nellymoser 8 kHz mono
> 6 = Nellymoser
> 7 = G.711 A-law logarithmic PCM

> 8 = G.711 mu-law logarithmic PCM
> 9 = reserved
> 10 = AAC
> 11 = Speex
> 14 = MP3 8 kHz
> 15 = Device-specific sound

低4位的前2位代表抽样频率，二进制11代表44kHZ。第3位代表 音频用16位的。第4个bit代表声道数，0单声道，1双声道。尽管如此，实际使用发现这个AF00根本不需要更改，我用的24K采样、单声道，这个数据也是AF00没问题，关键是后面两个字节。

> 后面两个字节叫做AudioSpecificConfig，从最高位到最低位，分别表示：

前5位，表示编码结构类型，AAC main编码为1，LOW低复杂度编码为2，SSR为3。

接着4位，表示采样率。 按理说，应该是：0 ~ 96000， 1~88200， 2~64000， 3~48000， 4~44100，5~32000， 6~24000， 7~ 22050， 8~16000...)，通常aac固定选中44100，即应该对应为4，

最后3位，固定为0吧。

在程序里面，发送rtmp的时候，初始化以后，不用自己填充这一堆数据，直接调用faacEncGetDecoderSpecificInfo，就能返回这个配置信息，很简单，2个字节的数据。

```
char *buf;

int len;

faacEncGetDecoderSpecificInfo(hEncoder, &buf, &len);

rtmp_sendaac_spec(buf, len);

free(buf);
```


后面的AAC数据发送的时候，前面7个字节都是帧头数据，不用发送，