Android音视频开发初探之AudioRecord与AudioTrack完成音频采集与播放

vvengzt 关注
0.393 2018.07.26 15:06:22 字数 2,922 阅读 4,320

> **有阵子没出文章，接下来争取这段时间持续更新，将沉淀的东西记录下来，废话不多说**
> 刚接触了音视频方面，趁热乎记录一下，欢迎大家指正

接下来会分为一下几点来介绍：

1. 基础知识准备
2. Android MediaRecorder和AudioRecord 与 MediaPlayer 和 AudioTrack 的介绍
3. PCM与WAV编码介绍与转化
4. 实例 Android AudioRecord 和 AudioTrack 的使用

基础知识准备

音频开发经常遇到的专业性词语

（1） 采样率

“ 音频采样率 ”是指录音设备在一秒钟内对声音信号的**采样次数**，采样频率越高声音的还原就越真实越自然。常用的音频采样频率有：8kHz、11.025kHz、22.05kHz、16kHz、37.8kHz、44.1kHz、48kHz、96kHz、192kHz等。在当今的主流采集卡上，采样频率一般共分为22.05KHz、44.1KHz、48KHz三个等级，22.05KHz只能达到FM广播的声音品质，44.1KHz则是理论上的CD音质界限，48KHz则更加精确一些。
通俗理解：每秒录取声音的次数。

（2） 量化精度（采样位数）

“ 采样位数 ”越大表示的值的范围也就越大
" 采样位数 "可以理解为采集卡处理声音的解析度。这个数值越大，解析度就越高，录制和回放的声音就越真实。**电脑中的声音文件是用数字0和1来表示的。**连续的模拟信号按一定的采样频率经数码脉冲取样后，每一个离散的脉冲信号被以一定的量化精度量化成一串二进制编码流，这串编码流的位数即为采样位数，也称为" 量化精度 "。
常见的位数为： `16bit` 和 `32bit`
通俗理解：每秒录取声音的精度，就像画面的分辨率，越高声音越真实

（3） 声道数

声道数 分别有： 单声道 的声道数为**1个声道**； 双声道 的声道数为2个声道；立体声道的声道数默认为**2个声道**；立体声道（4声道）的声道数为**4个声道。**
常见使用的是：单声道(MONO) 和 双声道 (STEREO)
通俗理解：声道数表示录制或者播放音频的声音源

（4） 比特率（码率）

比特率 （又叫做位速率或者码率）是指每秒传送的**比特（bit）数**。单位为bps(Bit Per Second），比特率越**高**，传送的数据越**大**。比特率表示经过编码（压缩）后的音、视频数据**每秒钟**需要用多少个比特来表示，而比特就是二进制里面最小的单位，要么是0，要么是1。比特率与音、视频压缩的关系，简单的说就是比特率越高，音、视频的质量就越好，但编码后的文件就越大；如果比特率越少则情况刚好相反。

若作为一种数字音乐压缩效率的参考性指标， 比特率 表示单位时间（1秒）内传送的**比特数bps（bit per second，位/秒）的速度。通常使用kbps（通俗地讲就是每秒钟1000比特）作为单位**。CD中的数字音乐比特率为1411.2kbps（也就是记录1秒钟的cd音乐，需要1411.2×1000比特的数据），音乐文件的BIT RATE高是意味着在单位时间（1秒）内需要处理的数据量（BIT）多，也就是音乐文件的音质好的意思。但是，BIT RATE高时文件大小变大，会占据很多的内存容量，音乐文件最常用的bit rate是128kbps，MP3文件可以使用的一般是8-320kbps，但不同MP3机在这方面支持的范围不一样，大部分的是32-256Kbps，这个指数当然是越广越好了，不过320Kbps是暂时最高等级了。<1B字节= 8 bit位 / 1024B = 1M>
（那一秒的CD音乐需要 1411.2 / 8 = 176.4KB/s的空间，那四分钟的CD音乐需要(1411.2kbps * 4 * 60)/ 8 / 1024 = 41.34373M）

码率计算公式
基本的算法是：【码率】（kbps)=【文件大小】（字节）X8/【时间】（秒）× 1000
音频文件专用算法：【比特率】（kbps)=【量化采样点】（kHz）×【位深】（bit/采样点）×【声道数量】（一般为2）

举例，D5的碟，容量4.3G，其中考虑到音频的不同格式，所以算为600M，（故剩余容量为4.3*1000-600=3700M），所以视频文件应不大于3.7G，本例中取视频文件的容量为3.446G，视频长度100分钟（6000秒），计算结果：码率约等于4933kbps。

码率几点原则
1、码率和质量成正比，但是文件体积也和码率成正比。
2、码率超过一定数值，对图像的质量没有多大影响。
3、DVD的容量有限，无论是标准的4.3G，还是超刻，或是D9，都有极限。

（5） PCM编码与WAV格式

PCM （Pulse Code Modulation—-脉码调制录音)。所谓PCM录音就是将声音等模拟信号变成符号化的脉冲列，再予以记录。PCM信号是由[1]、[0]等符号构成的数字信号，而未经过任何编码和压缩处理。与模拟信号比，它不易受传送系统的杂波及失真的影响。动态范围宽，可得到音质相当好的影响效果。也就是说，**PCM就是没有压缩的编码方式**，PCM文件就是采用PCM这种没有压缩的编码方式编码的音频数据文件。
PCM 约定俗成了 无损编码 ，因为PCM代表了数字音频中最佳的保真水准，并不意味着PCM就能够确保信号绝对保真，PCM也只能做到 最大程度 的无限接近。

WAV 为微软公司（Microsoft)开发的一种声音文件格式，它符合RIFF(Resource Interchange File Format)文件规范，用于保存Windows平台的音频信息资源，被Windows平台及其应用程序所广泛支持，该格式也支持MSADPCM，CCITT A LAW等多种压缩运算法，支持多种音频数字，取样频率和声道，标准格式化的WAV文件和CD格式一样，也是44.1K的取样频率，16位量化数字，因此在声音文件质量和CD相差无几!
在Windows平台下，基于PCM编码的WAV是被支持得最好的音频格式，所有音频软件都能完美支持，由于本身可以达到较高的音质的要求，因此，WAV也是音乐编辑创作的首选格式，适合保存音乐素材。因此，基于PCM编码的WAV被作为了一种中介的格式，常常使用在其他编码的相互转换之中，例如MP3转换成WMA。

通俗理解： PCM 是一种没有压缩且无损的编码方式， WAV 是微软开发的一种无损的音频文件格式 ， 而 WAV 是通过 PCM 数据的基础上 添加头部信息 而生成的一种音频格式，当然而可以基于其他如ADPCM编码添加头部信息生成WAV

（6） 音频数字化的过程

过程：获取音频源 —— 将模拟信号进行离散化的样本采集（**采样**）—— 取样的离散音频要转化为计算机能够表示的数据范围（**量化**）——（**编码**）按一定格式记录采样和量化后的数字数据，用二进制表示每个采样的量化值

模拟音频信号转化为数字音频信号：模拟音频信号是一个在时间上和幅度上都连续的信号，它的数字化过程如下所述：
1、**采样**：在时间轴上对信号数字化。也就是，按照固定的时间间隔抽取模拟信号的值，这样，采样后就可以使一个时间连续的信息波变为在时间上取值数目有限的离散信号。
2、**量化**：在幅度轴上对信号数字化。也就是，用有限个幅度值近似还原原来连续变化的幅度值，把模拟信号的连续幅度变为有限数量的有一定间隔的离散值。
3、**编码**：用二进制数表示每个采样的量化值（十进制数）。

简述（摘自）：
音频数字化通常经过三个阶段，即**采样—量化—编码**。音频数字化过程的具体步骤如下：**第一步**，将话筒转化过来的模拟电信号以某一频率进行离散化的样本采集，这个过程就叫采样；**第二步**，将采集到的样本电压或电流值进行等级量化处理，这个过程就是量化；**第三步**，将等级值变换成对应的二进制表示值（0和1），并进行存储，这个过程就是编码。通过这三个环节，连续的模拟音频信号即可转换成离散的数字信号——二进制的0和1 。

Android MediaRecorder和AudioRecord 与 MediaPlayer 和 AudioTrack 的介绍

> 官方 提供**两种API**用于音频开发，分别为 **MediaRecorder** 和 **AudioRecord** 用与音频的采集，**MediaPlayer** 和 **AudioTrack** 用于音频的播放

API	作用	优点	缺点
AudioRecord	音频采集	可以实时获取音频数据做到边录边播，更偏向底层更加灵活，可以对获取的音频做出处理，如 压缩、网络传输、算法处理等	由于输出的数据是原始数据PCM，播放器是不能识别播放的，需要通过AudioTrack处理播放
MediaRecorder	音频采集	官方将音频的录制、编码、压缩等都封装成API供使用、方便快捷	不能实时处理音频、输出的音频格式不多,有AMR/ACC/VORBIS 而这些PCM都可以处理生成
MediaPlayer	播放音频	MediaPlayer可以播放多种格式的声音文件，例如MP3，AAC，WAV，OGG，MIDI等。MediaPlayer会在framework层创建对应的音频解码器。	资源占用量较高、延迟时间较长、不支持多个音频同时播放等。这些缺点决定了MediaPlayer在某些场合的使用情况不会很理想，例如在对时间精准度要求相对较高的游戏开发中。

API	作用	优点	缺点
AudioTrack	播放音频	对于数据量小、延时要求高的音频处理可以使用AudioTrack的MODE_STATIC传输模式	AudioTrack只能播放已经解码的PCM流，如果是文件的话只支持wav格式的音频文件，因为wav格式的音频文件大部分都是PCM流。AudioTrack不创建解码器，所以只 能播放不需要解码的wav文件。

小知识点：

1. 在用MediaRecorder进行录制音视频时，最终还是会创建AudioRecord用来与AudioFlinger进行交互。
2. MediaPlayer在framework层还是会创建AudioTrack，把解码后的PCM数流传递给AudioTrack，AudioTrack再传递给AudioFlinger进行混音，然后才传递给硬件播放。所以是MediaPlayer包含了AudioTRack。

PCM编码转化为WAV音频格式

上面可知WAV是一种音频格式，而所有的WAV格式都有特定文件头，文件头储存着 RIFF文件标志、WAVE文件标志、采样率、声道数等信息，**PCM数据只需要加上WAV的文件头即可转化为WAV音频格式**

参考自文章作者 河北-宝哥

这里给出的是关于下面实例的转化WAV代码

```
/**
 * 将pcm文件转化为可点击播放的wav文件
 * @param inputPath pcm路径
 * @param outPath wav存放路径
 * @param data
 */
private void PcmtoWav(String inputPath ,String outPath ,byte[] data){
    FileInputStream in;//读取
    FileOutputStream out;
    try{
    in = new FileInputStream(inputPath);
    out = new FileOutputStream(outPath);
    //添加头部信息
    writeWavFileHeader(out,in.getChannel().size(),SAMPLE_RATE_HERTZ,CHANNEL_CONFIG
    while(in.read(data)!= -1){
      out.write(data);
    }
    //关流
    in.close();
    out.close();
    }catch (IOException e){
        e.printStackTrace();
    }
}

/**
 * @param out            wav音频文件流
 * @param totalAudioLen  不包括header的音频数据总长度
 * @param longSampleRate 采样率,也就是录制时使用的频率
 * @param channels       audioRecord的频道数量
 * @throws IOException 写文件错误
 */
private void writeWavFileHeader(FileOutputStream out, long totalAudioLen, long lon
                                int channels) throws IOException {
    byte[] header = generateWavFileHeader(totalAudioLen, longSampleRate, channels)
    //写头
    out.write(header, 0, header.length);

}

/**
 * 任何一种文件在头部添加相应的头文件才能够确定的表示这种文件的格式，
```

```
     * wave是RIFF文件结构，每一部分为一个chunk，其中有RIFF WAVE chunk，
     * FMT Chunk，Fact chunk,Data chunk,其中Fact chunk是可以选择的
     *
     * @param totalAudioLen  不包括header的音频数据总长度
     * @param longSampleRate 采样率,也就是录制时使用的频率
     * @param channels       audioRecord的频道数量
     */
    private byte[] generateWavFileHeader(long totalAudioLen, long longSampleRate, int 
        long totalDataLen = totalAudioLen + 36;
        long byteRate = longSampleRate * 2 * channels;
        byte[] header = new byte[44];
        header[0] = 'R'; // RIFF
        header[1] = 'I';
        header[2] = 'F';
        header[3] = 'F';
        header[4] = (byte) (totalDataLen & 0xff);//数据大小
        header[5] = (byte) ((totalDataLen >> 8) & 0xff);
        header[6] = (byte) ((totalDataLen >> 16) & 0xff);
        header[7] = (byte) ((totalDataLen >> 24) & 0xff);
        header[8] = 'W';//WAVE
        header[9] = 'A';
        header[10] = 'V';
        header[11] = 'E';
        //FMT Chunk
        header[12] = 'f'; // 'fmt '
        header[13] = 'm';
        header[14] = 't';
        header[15] = ' ';//过渡字节
        //数据大小
        header[16] = 16; // 4 bytes: size of 'fmt ' chunk
        header[17] = 0;
        header[18] = 0;
        header[19] = 0;
        //编码方式 10H为PCM编码格式
        header[20] = 1; // format = 1
        header[21] = 0;
        //通道数
        header[22] = (byte) channels;
        header[23] = 0;
        //采样率，每个通道的播放速度
        header[24] = (byte) (longSampleRate & 0xff);
        header[25] = (byte) ((longSampleRate >> 8) & 0xff);
        header[26] = (byte) ((longSampleRate >> 16) & 0xff);
        header[27] = (byte) ((longSampleRate >> 24) & 0xff);
        //音频数据传送速率,采样率*通道数*采样深度/8
        header[28] = (byte) (byteRate & 0xff);
        header[29] = (byte) ((byteRate >> 8) & 0xff);
        header[30] = (byte) ((byteRate >> 16) & 0xff);
        header[31] = (byte) ((byteRate >> 24) & 0xff);
        // 确定系统一次要处理多少个这样字节的数据，确定缓冲区，通道数*采样位数
        header[32] = (byte) (2 * channels);
        header[33] = 0;
        //每个样本的数据位数
        header[34] = 16;
        header[35] = 0;
        //Data chunk
        header[36] = 'd';//data
        header[37] = 'a';
        header[38] = 't';
        header[39] = 'a';
        header[40] = (byte) (totalAudioLen & 0xff);
        header[41] = (byte) ((totalAudioLen >> 8) & 0xff);
        header[42] = (byte) ((totalAudioLen >> 16) & 0xff);
        header[43] = (byte) ((totalAudioLen >> 24) & 0xff);
        return header;
    }
```

实例Android AudioRecord 和 AudioTrack 的使用

> 首先给出我们将音频的采集和播放封装成一个AudioRecordManager音频管理类 （代码有详细注释）

```
package com.example.medialearn.test2;

import android.media.AudioAttributes;
import android.media.AudioFormat;
```

```
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.os.Build;
import android.os.Environment;
import android.support.annotation.RequiresApi;
import android.util.Log;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

/**
 * @author vveng
 * @version version 1.0.0
 * @date 2018/7/24 16:03.
 * @email vvengstuggle@163.com
 * @instructions 说明
 * @describe 描述
 * @features 功能
 */
public class AudioRecordManager {

    private static final String TAG = "AudioRecordManager";
    private static final String DIR_NAME = "arm";
    private static String AudioFolderFile; //音频文件路径
    private static AudioRecordManager mAudioRecordManager;
    private File PcmFile = null ; //pcm音频文件
    private File WavFile = null;  //wav格式的音频文件
    private AudioRecordThread mAudioRecordThead; //录制线程
    private AudioRecordPlayThead mAudioRecordPlayThead;//播放线程
    private boolean isRecord = false;
    /**
     * 采样率，现在能够保证在所有设备上使用的采样率是44100Hz, 但是其他的采样率（22050, 16000, 110
     */
    public static final int SAMPLE_RATE_HERTZ = 44100;

    /**
     * 声道数。CHANNEL_IN_MONO and CHANNEL_IN_STEREO. 其中CHANNEL_IN_MONO是可以保证在所有设
     */
    public static final int CHANNEL_CONFIG = AudioFormat.CHANNEL_IN_STEREO;

    /**
     * 返回的音频数据的格式。 ENCODING_PCM_8BIT, ENCODING_PCM_16BIT, and ENCODING_PCM_FLOA
     */
    public static final int AUDIO_FORMAT = AudioFormat.ENCODING_PCM_16BIT;


    public static AudioRecordManager NewInstance() {
        if (mAudioRecordManager == null) {
            synchronized (AudioRecordManager.class) {
                if (mAudioRecordManager == null) {
                    mAudioRecordManager = new AudioRecordManager();
                }
            }
        }
        return mAudioRecordManager;
    }


    /**
     * 播放音频
     */
    @RequiresApi(api = Build.VERSION_CODES.M)
    public synchronized void playRecord() {
        //可防止重复点击录制
        if (true == isRecord) {
            Log.d(TAG, "无法开始播放，当前状态为：" + isRecord);
            return;
        }
        isRecord = true;
        mAudioRecordPlayThead = new AudioRecordPlayThead(PcmFile);
        mAudioRecordPlayThead.start();
    }

    /**
     * 停止播放
     */
    public void stopPlayRecord() {
```

```
        if (null != mAudioRecordPlayThead) {
            mAudioRecordPlayThead.interrupt();
            mAudioRecordPlayThead = null;
        }
        isRecord = false;
    }

    /**
     * 播放音频线程
     */
    private class AudioRecordPlayThead extends Thread {
        AudioTrack mAudioTrack;
        int BufferSize = 10240;
        File autoFile = null; //要播放的文件

        @RequiresApi(api = Build.VERSION_CODES.M)
        AudioRecordPlayThead(File file) {
            setPriority(MAX_PRIORITY);
            autoFile = file;
            //播放缓冲的最小大小
            BufferSize = AudioTrack.getMinBufferSize(SAMPLE_RATE_HERTZ,
                    AudioFormat.CHANNEL_OUT_STEREO, AUDIO_FORMAT);
           // 创建用于播放的 AudioTrack
            mAudioTrack = new AudioTrack.Builder()
                    .setAudioAttributes(new AudioAttributes.Builder()
                            .setUsage(AudioAttributes.USAGE_ALARM)
                            .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
                            .build())
                    .setAudioFormat(new AudioFormat.Builder()
                            .setEncoding(AUDIO_FORMAT)
                            .setSampleRate(SAMPLE_RATE_HERTZ)
                            .setChannelMask(AudioFormat.CHANNEL_OUT_STEREO)
                            .build())
                    .setBufferSizeInBytes(BufferSize)
                    .build();

        }

        @Override
        public void run() {

            Log.d(TAG, "播放开始");
            try {
                FileInputStream fis = new FileInputStream(autoFile);
                mAudioTrack.play();
                byte[] bytes = new byte[BufferSize];

                while(true == isRecord) {
                    int read = fis.read(bytes);
                    //若读取有错则跳过
                    if (AudioTrack.ERROR_INVALID_OPERATION == read
                            || AudioTrack.ERROR_BAD_VALUE == read) {
                        continue;
                    }

                    if (read != 0 && read != -1) {
                        mAudioTrack.write(bytes, 0, BufferSize);
                    }
                }
                mAudioTrack.stop();
                mAudioTrack.release();//释放资源
                fis.close();//关流

            } catch (Exception e) {
                e.printStackTrace();
            }

            isRecord = false;
            Log.d(TAG, "播放停止");
        }
    }


    /**
     * 开始录制
     */
    public synchronized void startRecord() {
        //可防止重复点击录制
        if (true == isRecord) {
            Log.d(TAG, "无法开始录制，当前状态为：" + isRecord);
            return;
        }
        isRecord = true;
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("yyMMdd_HHmmss", Locale.CHINA);
        //源pcm数据文件
        PcmFile = new File(AudioFolderFile + File.separator + sdf.format(new Date())+"
        //wav文件
        WavFile = new File(PcmFile.getPath().replace(".pcm",".wav"));

        Log.d(TAG, "PcmFile:"+ PcmFile.getName()+"WavFile:"+WavFile.getName());

        if (null != mAudioRecordThead) {
            //若线程不为空,则中断线程
            mAudioRecordThead.interrupt();
            mAudioRecordThead = null;
        }
        mAudioRecordThead = new AudioRecordThread();
        mAudioRecordThead.start();
    }

    /**
     * 停止录制
     */
    public synchronized void stopRecord() {
        if (null != mAudioRecordThead) {
            mAudioRecordThead.interrupt();
            mAudioRecordThead = null;
        }

        isRecord = false;
    }

    /**
     * 录制线程
     */
    private class AudioRecordThread extends Thread {
        AudioRecord mAudioRecord;
        int BufferSize = 10240;

        AudioRecordThread() {
            /**
             * 获取音频缓冲最小的大小
             */
            BufferSize = AudioRecord.getMinBufferSize(SAMPLE_RATE_HERTZ,
                    CHANNEL_CONFIG, AUDIO_FORMAT);
            /**
             * 参数1: 音频源
             * 参数2: 采样率 主流是44100
             * 参数3: 声道设置 MONO单声道 STEREO立体声
             * 参数4: 编码格式和采样大小 编码格式为PCM,主流大小为16BIT
             * 参数5: 采集数据需要的缓冲区大小
             */
            mAudioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC,
                    SAMPLE_RATE_HERTZ, CHANNEL_CONFIG, AUDIO_FORMAT, BufferSize);
        }

        @Override
        public void run() {
            //将状态置为录制

            Log.d(TAG, "录制开始");
            try {
                byte[] bytes = new byte[BufferSize];

                FileOutputStream PcmFos = new FileOutputStream(PcmFile);

                //开始录制
                mAudioRecord.startRecording();

                while (true == isRecord && !isInterrupted()) {
                    int read = mAudioRecord.read(bytes, 0, bytes.length);
                    //若读取数据没有出现错误，将数据写入文件
                    if (AudioRecord.ERROR_INVALID_OPERATION != read) {
                        PcmFos.write(bytes, 0, read);
                        PcmFos.flush();
                    }
                }
                mAudioRecord.stop();//停止录制
                PcmFos.close();//关流

            } catch (Exception e) {
                e.printStackTrace();

            }
            isRecord = false;
            //当录制完成就将Pcm编码数据转化为wav文件，也可以直接生成.wav
```

```
            PcmtoWav(PcmFile.getPath(),WavFile.getPath(),new byte[BufferSize]);
            Log.d(TAG, "录制结束");
        }

    }


    /**
     * 初始化目录
     */
    public static void init() {
        //文件目录
        AudioFolderFile = Environment.getExternalStoragePublicDirectory(Environment.DI
                + File.separator + DIR_NAME;
        File WavDir = new File(AudioFolderFile);
        if (!WavDir.exists()) {
            boolean flag = WavDir.mkdirs();
            Log.d(TAG,"文件路径:"+AudioFolderFile+"创建结果:"+flag);
        } else {
            Log.d(TAG,"文件路径:"+AudioFolderFile+"创建结果: 已存在");
        }
    }

    /**
     * 将pcm文件转化为可点击播放的wav文件
     * @param inputPath pcm路径
     * @param outPath wav存放路径
     * @param data
     */
    private void PcmtoWav(String inputPath ,String outPath ,byte[] data){
        FileInputStream in;
        FileOutputStream out;
        try{
        in = new FileInputStream(inputPath);
        out = new FileOutputStream(outPath);
        //添加头部信息
        writeWavFileHeader(out,in.getChannel().size(),SAMPLE_RATE_HERTZ,CHANNEL_CONFIG
        while(in.read(data)!= -1){
          out.write(data);
        }
        in.close();
        out.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    /**
     * @param out            wav音频文件流
     * @param totalAudioLen  不包括header的音频数据总长度
     * @param longSampleRate 采样率,也就是录制时使用的频率
     * @param channels       audioRecord的频道数量
     * @throws IOException 写文件错误
     */
    private void writeWavFileHeader(FileOutputStream out, long totalAudioLen, long lon
                                    int channels) throws IOException {
        byte[] header = generateWavFileHeader(totalAudioLen, longSampleRate, channels)
        out.write(header, 0, header.length);

    }

    /**
     * 任何一种文件在头部添加相应的头文件才能够确定的表示这种文件的格式,
     * wave是RIFF文件结构，每一部分为一个chunk，其中有RIFF WAVE chunk,
     * FMT Chunk，Fact chunk,Data chunk,其中Fact chunk是可以选择的
     *
     * @param totalAudioLen  不包括header的音频数据总长度
     * @param longSampleRate 采样率,也就是录制时使用的频率
     * @param channels       audioRecord的频道数量
     */
    private byte[] generateWavFileHeader(long totalAudioLen, long longSampleRate, int
        long totalDataLen = totalAudioLen + 36;
        long byteRate = longSampleRate * 2 * channels;
        byte[] header = new byte[44];
        header[0] = 'R'; // RIFF
        header[1] = 'I';
        header[2] = 'F';
        header[3] = 'F';
        header[4] = (byte) (totalDataLen & 0xff);//数据大小
        header[5] = (byte) ((totalDataLen >> 8) & 0xff);
        header[6] = (byte) ((totalDataLen >> 16) & 0xff);
        header[7] = (byte) ((totalDataLen >> 24) & 0xff);
        header[8] = 'W';//WAVE
```

```
        header[9] = 'A';
        header[10] = 'V';
        header[11] = 'E';
        //FMT Chunk
        header[12] = 'f'; // 'fmt '
        header[13] = 'm';
        header[14] = 't';
        header[15] = ' ';//过渡字节
        //数据大小
        header[16] = 16; // 4 bytes: size of 'fmt ' chunk
        header[17] = 0;
        header[18] = 0;
        header[19] = 0;
        //编码方式 10H为PCM编码格式
        header[20] = 1; // format = 1
        header[21] = 0;
        //通道数
        header[22] = (byte) channels;
        header[23] = 0;
        //采样率，每个通道的播放速度
        header[24] = (byte) (longSampleRate & 0xff);
        header[25] = (byte) ((longSampleRate >> 8) & 0xff);
        header[26] = (byte) ((longSampleRate >> 16) & 0xff);
        header[27] = (byte) ((longSampleRate >> 24) & 0xff);
        //音频数据传送速率,采样率*通道数*采样深度/8
        header[28] = (byte) (byteRate & 0xff);
        header[29] = (byte) ((byteRate >> 8) & 0xff);
        header[30] = (byte) ((byteRate >> 16) & 0xff);
        header[31] = (byte) ((byteRate >> 24) & 0xff);
        // 确定系统一次要处理多少个这样字节的数据，确定缓冲区，通道数*采样位数
        header[32] = (byte) (2 * channels);
        header[33] = 0;
        //每个样本的数据位数
        header[34] = 16;
        header[35] = 0;
        //Data chunk
        header[36] = 'd';//data
        header[37] = 'a';
        header[38] = 't';
        header[39] = 'a';
        header[40] = (byte) (totalAudioLen & 0xff);
        header[41] = (byte) ((totalAudioLen >> 8) & 0xff);
        header[42] = (byte) ((totalAudioLen >> 16) & 0xff);
        header[43] = (byte) ((totalAudioLen >> 24) & 0xff);
        return header;
    }
}
```

> 使用AudioRecordActivity （由于布局简单就四个按钮，这里就不再给出）

注意在AndroidManifest.xml 添加相关权限：

```
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <!-- 录音权限 -->
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
```

AudioRecordActivity：

```
package com.example.medialearn.test2;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Build;
import android.os.Environment;
import android.support.annotation.NonNull;
import android.support.annotation.RequiresApi;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import com.example.medialearn.R;
```

```
18
19  import java.io.File;
20  import java.util.ArrayList;
21  import java.util.List;
22
23  /**
24   * @author vveng
25   * @version version 1.0.0
26   * @date 2018/7/24 16:03.
27   * @email vvengstuggle@163.com
28   * @instructions 说明
29   * @describe 描述
30   * @features 功能
31   */
32  public class AudioRecordActivity extends AppCompatActivity
33          implements View.OnClickListener {
34      private String TAG = "AudioRecordActivity";
35      private Button btn_start, btn_stop, btn_play, btn_onplay;
36      private AudioRecordManager mManager;
37      //申请权限列表
38      private int REQUEST_CODE = 1001;
39      private String[] permissions = new String[]{
40              Manifest.permission.WRITE_EXTERNAL_STORAGE,
41              Manifest.permission.READ_EXTERNAL_STORAGE,
42              Manifest.permission.RECORD_AUDIO
43      };
44      //拒绝权限列表
45      private List<String> refusePermissions = new ArrayList<>();
46
47      @Override
48      protected void onCreate(Bundle savedInstanceState) {
49          super.onCreate(savedInstanceState);
50          setContentView(R.layout.activity_audio_record);
51          AudioRecordManager.init();//初始化目录
52          initView();
53          checkPermission();
54      }
55
56      /**
57       * 6.0以上要动态申请权限
58       */
59      private void checkPermission() {
60          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
61              for (int i = 0; i < permissions.length; i++) {
62                  if (ContextCompat.checkSelfPermission(this,
63                          permissions[i]) != PackageManager.PERMISSION_GRANTED) {
64                      refusePermissions.add(permissions[i]);
65                  }
66              }
67              if (!refusePermissions.isEmpty()) {
68                  String[] permissions = refusePermissions.toArray(new String[refusePerm
69                  ActivityCompat.requestPermissions(this, permissions, REQUEST_CODE);
70              }
71          }
72
73      }
74
75      /**
76       * 权限结果回调
77       * @param requestCode
78       * @param permissions
79       * @param grantResults
80       */
81      @Override
82      public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissi
83          if (requestCode == REQUEST_CODE) {
84              if (grantResults.length > 0 && grantResults != null) {
85                  for(int i = 0 ; i<permissions.length;i++){
86                      if(grantResults[i]!=PackageManager.PERMISSION_GRANTED){
87                          Log.d(TAG,permissions[i]+"   被禁用");
88                      }
89                  }
90              }
91          }
92      }
93
94      private void initView() {
95          btn_start = findViewById(R.id.record_start);
96          btn_stop = findViewById(R.id.record_stop);
97          btn_play = findViewById(R.id.record_play);
98          btn_onplay = findViewById(R.id.record_noplay);
99          btn_start.setOnClickListener(this);
100         btn_stop.setOnClickListener(this);
```

```
        btn_play.setOnClickListener(this);
        btn_onplay.setOnClickListener(this);
        //初始化
        mManager = AudioRecordManager.NewInstance();
    }

    @RequiresApi(api = Build.VERSION_CODES.M)
    @Override
    public void onClick(View view) {

        switch (view.getId()) {
            case R.id.record_start:
            //录音
                mManager.startRecord();
                break;
            case R.id.record_stop:
            //停止
                mManager.stopRecord();
                break;
            case R.id.record_play:
            //播放
                mManager.playRecord();
                break;
            case R.id.record_noplay:
            //停止
                mManager.stopPlayRecord();
                break;
            default:
                break;
        }
    }
}

```

参考链接：

https://developer.android.com/reference/android/media/AudioRecord#AudioRecord(int,%20int,%20int,%20int,%20int)

https://developer.android.com/reference/android/media/AudioTrack#play()

https://blog.csdn.net/ameyume/article/details/7618820

https://blog.csdn.net/zyuanyun/article/details/60890534

http://www.qingpingshan.com/rjbc/az/376811.html