Android音频开发

笔记本:	我的第一个笔记本		
创建时间:	2021/3/8 22:21	**更新时间:**	2021/3/9 0:20
作者:	136iqdlb602		
URL:	https://www.jianshu.com/p/a72deab95b4c		

[Android音频开发（1）：音频基础知识 - 简书 (jianshu.com)](https://www.jianshu.com/p/a72deab95b4c)

https://github.com/zhaolewei/ZlwAudioRecorder

使用AudioRecord录制pcm格式音频

一、AudioRecord类的介绍

AudioRecord构造函数:

```
/**
* @param audioSource : 录音源
* 这里选择使用麦克风：MediaRecorder.AudioSource.MIC
* @param sampleRateInHz:  采样率
* @param channelConfig：声道数
* @param audioFormat:  采样位数.
*   See {@link AudioFormat#ENCODING_PCM_8BIT}, {@link
AudioFormat#ENCODING_PCM_16BIT},
*   and {@link AudioFormat#ENCODING_PCM_FLOAT}.
* @param bufferSizeInBytes:  音频录制的缓冲区大小
*   See {@link #getMinBufferSize(int, int, int)}
*/
public AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int
audioFormat,  int bufferSizeInBytes)
```

getMinBufferSize()

```
/**
* 获取AudioRecord所需的最小缓冲区大小
* @param sampleRateInHz:  采样率
* @param channelConfig：声道数
* @param audioFormat:  采样位数.
*/
public static int getMinBufferSize (int sampleRateInHz, int channelConfig,  int
audioFormat)
```

getRecordingState()

```
/**
* 获取AudioRecord当前的录音状态
*   @see AudioRecord#RECORDSTATE_STOPPED
*   @see AudioRecord#RECORDSTATE_RECORDING
*/
public int getRecordingState()
```

startRecording()

```
/**
* 开始录制
*/
public int startRecording()
```

stop()

```
/**
* 停止录制
*/
public int stop()
```

read()

```
/**
* 从录音设备中读取音频数据
* @param audioData 音频数据写入的byte[]缓冲区
* @param offsetInBytes 偏移量
* @param sizeInBytes 读取大小
* @return 返回负数则表示读取失败
*      see {@link #ERROR_INVALID_OPERATION} -3 :  初始化错误
        {@link #ERROR_BAD_VALUE}  -3:  参数错误
        {@link #ERROR_DEAD_OBJECT} -6:
        {@link #ERROR}
*/
public int read(@NonNull byte[] audioData, int offsetInBytes, int sizeInBytes)
```

使用AudioRecord实现录音的暂停和恢复

现在可以实现音频的文件录制和停止，并生成pcm文件，那么暂停时将这次文件先保存下来，恢复播放后开始新一轮的录制，那么最后会生成多个pcm音频，再将这些pcm文件进行合并，这样就实现了暂停/恢复的功能了。

二、实现

- 实现过程就是调用上面的API的方法，构造AudioRecord实例后再调用startRecording()，开始录音，并通过read()方法不断获取录音数据记录下来，生成PCM文件。涉及耗时操作，所以最好在子线程中进行。

```
/**
* @author zhaolewei on 2018/7/10.
*/public class RecordHelper {
    private volatile RecordState state = RecordState.IDLE;
    private AudioRecordThread audioRecordThread;

    private File recordFile = null;
    private File tmpFile = null;
    private List<File> files = new ArrayList<>();

    public void start(String filePath, RecordConfig config) {
        this.currentConfig = config;
        if (state != RecordState.IDLE) {
            Logger.e(TAG, "状态异常当前状态： %s", state.name());
            return;
        }
```

```
        recordFile = new File(filePath);
        String tempFilePath = getTempFilePath();
        Logger.i(TAG, "tmpPCM File: %s", tempFilePath);
        tmpFile = new File(tempFilePath);
        audioRecordThread = new AudioRecordThread();
        audioRecordThread.start();
    }

    public void stop() {
        if (state == RecordState.IDLE) {
            Logger.e(TAG, "状态异常当前状态： %s", state.name());
            return;
        }

        //若在暂停中直接停止，则直接合并文件即可
        if (state == RecordState.PAUSE) {
            makeFile();
            state = RecordState.IDLE;
        } else {
            state = RecordState.STOP;
        }
    }

    public void pause() {
        if (state != RecordState.RECORDING) {
            Logger.e(TAG, "状态异常当前状态： %s", state.name());
            return;
        }
        state = RecordState.PAUSE;
    }

    public void resume() {
        if (state != RecordState.PAUSE) {
            Logger.e(TAG, "状态异常当前状态： %s", state.name());
            return;
        }
        String tempFilePath = getTempFilePath();
        Logger.i(TAG, "tmpPCM File: %s", tempFilePath);
        tmpFile = new File(tempFilePath);
        audioRecordThread = new AudioRecordThread();
        audioRecordThread.start();
    }

    private class AudioRecordThread extends Thread {
        private AudioRecord audioRecord;
        private int bufferSize;

        AudioRecordThread() {
            bufferSize =
AudioRecord.getMinBufferSize(currentConfig.getFrequency(),
                    currentConfig.getChannel(), currentConfig.getEncoding()) *
RECORD_AUDIO_BUFFER_TIMES;
            audioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC,
currentConfig.getFrequency(),
                    currentConfig.getChannel(), currentConfig.getEncoding(),
bufferSize);
        }

        @Override
        public void run() {
            super.run();
            state = RecordState.RECORDING;
            notifyState();
            Logger.d(TAG, "开始录制");
            FileOutputStream fos = null;
            try {
                fos = new FileOutputStream(tmpFile);
                audioRecord.startRecording();
                byte[] byteBuffer = new byte[bufferSize];
```

```
                while (state == RecordState.RECORDING) {
                        int end = audioRecord.read(byteBuffer, 0,
byteBuffer.length);
                    fos.write(byteBuffer, 0, end);
                    fos.flush();
                }
                audioRecord.stop();
                //1. 将本次录音的文件暂存下来，用于合并
                files.add(tmpFile);
                //2. 再此判断终止循环的状态是暂停还是停止，并做相应处理
                if (state == RecordState.STOP) {
                    makeFile();
                } else {
                    Logger.i(TAG, "暂停！");
                }
            } catch (Exception e) {
                Logger.e(e, TAG, e.getMessage());
            } finally {
                try {
                    if (fos != null) {
                        fos.close();
                    }
                } catch (IOException e) {
                    Logger.e(e, TAG, e.getMessage());
                }
            }
            if (state != RecordState.PAUSE) {
                state = RecordState.IDLE;
                notifyState();
                Logger.d(TAG, "录音结束");
            }
        }
    }

    private void makeFile() {
        //合并文件
        boolean mergeSuccess = mergePcmFiles(recordFile, files);

        //TODO:转换wav
        Logger.i(TAG, "录音完成！ path: %s ； 大小：%s",
recordFile.getAbsoluteFile(), recordFile.length());
    }

    /**
     * 合并Pcm文件
     *
     * @param recordFile 输出文件
     * @param files      多个文件源
     * @return 是否成功
     */
    private boolean mergePcmFiles(File recordFile, List<File> files) {
        if (recordFile == null || files == null || files.size() <= 0) {
            return false;
        }

        FileOutputStream fos = null;
        BufferedOutputStream outputStream = null;
        byte[] buffer = new byte[1024];
        try {
            fos = new FileOutputStream(recordFile);
            outputStream = new BufferedOutputStream(fos);

            for (int i = 0; i < files.size(); i++) {
                BufferedInputStream inputStream = new BufferedInputStream(new
FileInputStream(files.get(i)));
                int readCount;
                while ((readCount = inputStream.read(buffer)) > 0) {
                    outputStream.write(buffer, 0, readCount);
                }
                inputStream.close();
```

```
            }
        } catch (Exception e) {
            Logger.e(e, TAG, e.getMessage());
            return false;
        } finally {
            try {
                if (fos != null) {
                    fos.close();
                }
                if (outputStream != null) {
                    outputStream.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        //3. 合并后记得删除缓存文件并清除list
        for (int i = 0; i < files.size(); i++) {
            files.get(i).delete();
        }
        files.clear();
        return true;
    }

}
```

PCM转WAV格式音频

笔记本:	我的第一个笔记本		
创建时间:	2021/3/8 23:38	**更新时间:**	2021/3/9 0:13
作者:	136iqdlb602		
URL:	https://www.jianshu.com/p/90c77197f1d4		

一、wav 和 pcm

一般通过麦克风采集的录音数据都是PCM格式的，即不包含头部信息，播放器无法知道音频采样率、位宽等参数，导致无法播放，显然是非常不方便的。pcm转换成wav，我们只需要在pcm的文件起始位置加上至少44个字节的WAV头信息即可。

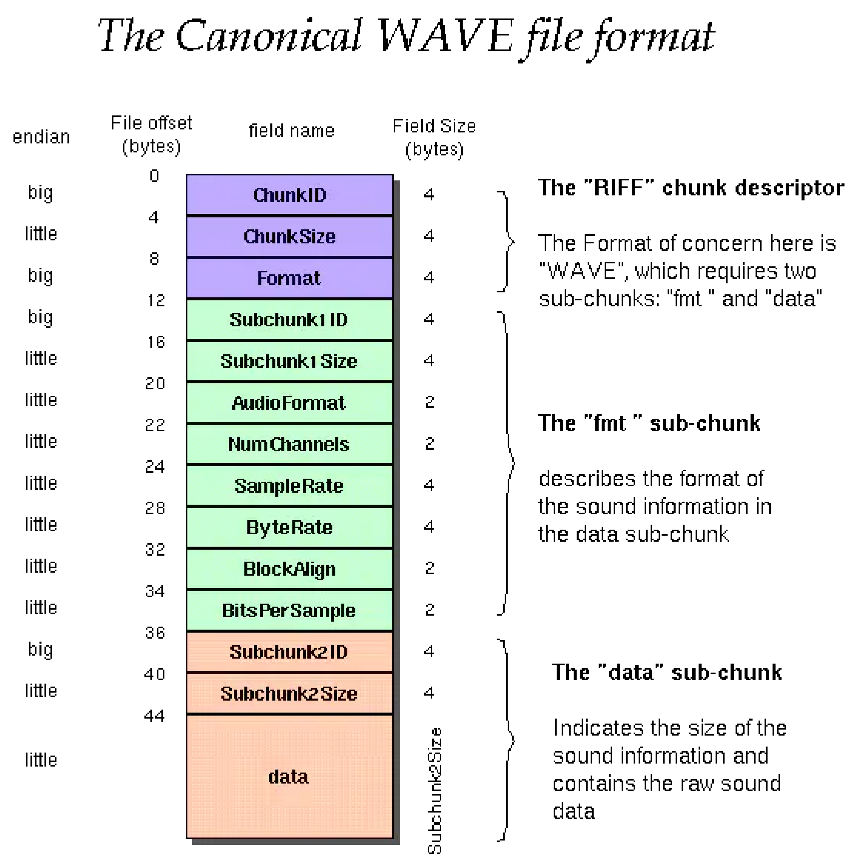
RIFF

- WAVE文件是以RIFF(Resource Interchange File Format, "资源交互文件格式")格式来组织内部结构的
  RIFF文件结构可以看作是树状结构,其基本构成是称为"块"（Chunk）的单元.
- WAVE文件是由若干个Chunk组成的。按照在文件中的出现位置包括：RIFF WAVE Chunk, Format Chunk, Fact Chunk(可选), Data Chunk。

  Fact Chunk 在压缩后或在非PCM编码时存在

二、WAV头文件

所有的WAV都有一个文件头，这个文件头记录着音频流的编码参数。数据块的记录方式是little-endian字节顺序。



偏移地址	命名	内容
00-03	ChunkId	"RIFF"
04-07	ChunkSize	下个地址开始到文件尾的总字节数(此Chunk的数据大小)
08-11	fccType	"WAVE"

12-15	SubChunkId1	"fmt ",最后一位空格。
16-19	SubChunkSize1	一般为16，表示fmt Chunk的数据块大小为16字节，即20-35
20-21	FormatTag	1：表示是PCM 编码
22-23	Channels	声道数，单声道为1，双声道为2
24-27	SamplesPerSec	采样率
28-31	BytesPerSec	码率： 采样率 * 采样位数 * 声道个数，bytePerSecond = sampleRate * (bitsPerSample / 8) * channels
32-33	BlockAlign	每次采样的大小：位宽*声道数/8
34-35	BitsPerSample	位宽
36-39	SubChunkId2	"data"
40-43	SubChunkSize2	音频数据的长度
44-...	data	音频数据

三、java 生成头文件

1. WavHeader.class

```
public static class WavHeader {
    /**
     * RIFF数据块
     */
    final String riffChunkId = "RIFF";
    int riffChunkSize;
    final String riffType = "WAVE";


    /**
     * FORMAT 数据块
     */
    final String formatChunkId = "fmt ";
    final int formatChunkSize = 16;
    final short audioFormat = 1;
    short channels;
    int sampleRate;
    int byteRate;
    short blockAlign;
    short sampleBits;


    /**
     * FORMAT 数据块
     */
    final String dataChunkId = "data";
    int dataChunkSize;


    WavHeader(int totalAudioLen, int sampleRate, short channels, short sampleBits) {
        this.riffChunkSize = totalAudioLen;
        this.channels = channels;
        this.sampleRate = sampleRate;
        this.byteRate = sampleRate * sampleBits / 8 * channels;
        this.blockAlign = (short) (channels * sampleBits / 8);
        this.sampleBits = sampleBits;
        this.dataChunkSize = totalAudioLen - 44;
    }


    public byte[] getHeader() {
        byte[] result;
        result = ByteUtils.merger(ByteUtils.toBytes(riffChunkId), ByteUtils.toBytes(riffChunkSize));
        result = ByteUtils.merger(result, ByteUtils.toBytes(riffType));
        result = ByteUtils.merger(result, ByteUtils.toBytes(formatChunkId));
        result = ByteUtils.merger(result, ByteUtils.toBytes(formatChunkSize));
        result = ByteUtils.merger(result, ByteUtils.toBytes(audioFormat));
        result = ByteUtils.merger(result, ByteUtils.toBytes(channels));
        result = ByteUtils.merger(result, ByteUtils.toBytes(sampleRate));
        result = ByteUtils.merger(result, ByteUtils.toBytes(byteRate));
        result = ByteUtils.merger(result, ByteUtils.toBytes(blockAlign));
        result = ByteUtils.merger(result, ByteUtils.toBytes(sampleBits));
        result = ByteUtils.merger(result, ByteUtils.toBytes(dataChunkId));
        result = ByteUtils.merger(result, ByteUtils.toBytes(dataChunkSize));
        return result;
```

```
    }
}
```

ByteUtils: https://github.com/zhaolewei/ZlwAudioRecorder/blob/master/recorderlib/src/main/java/com/zlw/main/recorderlib/utils/ByteUtils.java

四、PCM转Wav

1. WavUtils.java

```
public class WavUtils {
    private static final String TAG = WavUtils.class.getSimpleName();
        /**
         * 生成wav格式的Header
         * 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（可选）,Data chunk
         *
         * @param totalAudioLen 不包括header的音频数据总长度
         * @param sampleRate    采样率,也就是录制时使用的频率
         * @param channels      audioRecord的频道数量
         * @param sampleBits    位宽
         */
        public static byte[] generateWavFileHeader(int totalAudioLen, int sampleRate, int channels, int sampleBits)
{
            WavHeader wavHeader = new WavHeader(totalAudioLen, sampleRate, (short) channels, (short) sampleBits);
            return wavHeader.getHeader();
        }
    }


    /**
     * 将header写入到pcm文件中 不修改文件名
     *
     * @param file   写入的pcm文件
     * @param header wav头数据
     */
    public static void writeHeader(File file, byte[] header) {
        if (!FileUtils.isFile(file)) {
            return;
        }


        RandomAccessFile wavRaf = null;
        try {
            wavRaf = new RandomAccessFile(file, "rw");
            wavRaf.seek(0);
            wavRaf.write(header);
            wavRaf.close();
        } catch (Exception e) {
            Logger.e(e, TAG, e.getMessage());
        } finally {
            try {
                if (wavRaf != null) {
                    wavRaf.close();
                }
            } catch (IOException e) {
                Logger.e(e, TAG, e.getMessage());
            }
        }
```

2. RecordHelper.java

```
private void makeFile() {
    mergePcmFiles(recordFile, files);


    //这里实现上一篇未完成的工作
    byte[] header = WavUtils.generateWavFileHeader((int) resultFile.length(), currentConfig.getSampleRate(),
currentConfig.getChannelCount(), currentConfig.getEncoding());
    WavUtils.writeHeader(resultFile, header);

    Logger.i(TAG, "录音完成! path: %s ； 大小：%s", recordFile.getAbsoluteFile(), recordFile.length());
}
```