# https://blog.csdn.net/fangdawei\_www/article/details/65039470

# Android中实现多段wav音频文件拼接

2017年03月22日 23:11:33 [猿某某](https://me.csdn.net/fangdawei_www" \t "/Users/wjw\\x/_blank) 阅读数：787

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/fangdawei\_www/article/details/65039470

WAV为微软公司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。   
由于项目中需要接入讯飞的语音听写进行快速录入，并且同时保存语音文件。讯飞语音听写的SDK只支持保存语音文件为pcm或者wav这两种格式。讯飞的语音听写服务有很多限制，比如前后端点允许静音最长10秒、一次听写连续不能超过60秒。项目中需要支持长时间不间断语音听写，和产品怼了很久，经过不懈的抗争，最后还是我妥协了。讯飞语音听写的SDK提供了一些回调，在超时中断时，会回调onEndOfSpeech方法，这样我们就可以在这里马上重新开始启动听写。但是这会引起另一个问题，录制的音频文件最后是一段一段的，最后还得把他们进行拼接。第一次使用讯飞的语音听写SDK，不是很熟，不知道有没有哪位大神有更好的解决办法，求赐教啊啊啊啊。。。   
寻找了很久，在Android的API中没找到可以实现wav拼接的方法，只能自己去实现了。万幸的是wav格式的结构还比较简单。

### **WAV文件格式**



（本来是使用table编辑的表格，简书上竟然不支持，没办法只能截了个图放上来了）   
可以看出，WAV文件主要是以四种chunk组成，这里我们分别称呼为riff chunk、fmt chunk、fact chunk和data chunk，其中fact chunk不是必须的，大部分时候的没有。所以我在查阅资料的额时候，发现很多解析WAV文件的代码都直接认为其只有固定的44字节的头部。   
此格式来源于百度百科，奇怪的是维基百科中也认为WAV具有一个44字节的固定头部，如果哪位大神知道的，可以告诉我一下。

### **WAV拼接实现方法**

由于这里采集的音频相关参数一致，做我们去其中一段的头部作为拼接后的音频的头部。但是也不是这样就可以了。从上面WAV的格式中可以看出，头部中两个位置的数据需要修改。1、riff chunk中的size值；2、data chunk的size值。因此可以先将其他数据的data chunk部分的数据追加到结果文件中，最后写入这两个地方的值。   
好了，是时候上代码了。。。

### **实现代码**

public class WavMergeUtil {

public static void mergeWav(List<File> inputs, File output) throws IOException {

if (inputs.size() < 1) {

return;

}

FileInputStream fis = new FileInputStream(inputs.get(0));

FileOutputStream fos = new FileOutputStream(output);

byte[] buffer = new byte[2048];

int total = 0;

int count;

while ((count = fis.read(buffer)) > -1) {

fos.write(buffer, 0, count);

total += count;

}

fis.close();

for (int i = 1; i < inputs.size(); i++) {

File file = inputs.get(i);

Header header = resolveHeader(file);

FileInputStream dataInputStream = header.dataInputStream;

while ((count = dataInputStream.read(buffer)) > -1) {

fos.write(buffer, 0, count);

total += count;

}

dataInputStream.close();

}

fos.flush();

fos.close();

Header outputHeader = resolveHeader(output);

outputHeader.dataInputStream.close();

RandomAccessFile res = new RandomAccessFile(output, "rw");

res.seek(4);

byte[] fileLen = intToByteArray(total + outputHeader.dataOffset - 8);

res.write(fileLen, 0, 4);

res.seek(outputHeader.dataSizeOffset);

byte[] dataLen = intToByteArray(total);

res.write(dataLen, 0, 4);

res.close();

}

/\*\*

\* 解析头部，并获得文件指针指向数据开始位置的InputStreram，记得使用后需要关闭

\*/

private static Header resolveHeader(File wavFile) throws IOException {

FileInputStream fis = new FileInputStream(wavFile);

byte[] byte4 = new byte[4];

byte[] buffer = new byte[2048];

int readCount = 0;

Header header = new Header();

fis.read(byte4);*//RIFF*

fis.read(byte4);

readCount += 8;

header.fileSizeOffset = 4;

header.fileSize = byteArrayToInt(byte4);

fis.read(byte4);*//WAVE*

fis.read(byte4);*//fmt*

fis.read(byte4);

readCount += 12;

int fmtLen = byteArrayToInt(byte4);

fis.read(buffer, 0, fmtLen);

readCount += fmtLen;

fis.read(byte4);*//data or fact*

readCount += 4;

if (isFmt(byte4, 0)) {*//包含fmt段*

fis.read(byte4);

int factLen = byteArrayToInt(byte4);

fis.read(buffer, 0, factLen);

fis.read(byte4);*//data*

readCount += 8 + factLen;

}

fis.read(byte4);*// data size*

int dataLen = byteArrayToInt(byte4);

header.dataSize = dataLen;

header.dataSizeOffset = readCount;

readCount += 4;

header.dataOffset = readCount;

header.dataInputStream = fis;

return header;

}

private static boolean isRiff(byte[] bytes, int start) {

if (bytes[start + 0] == 'R' && bytes[start + 1] == 'I' && bytes[start + 2] == 'F' && bytes[start + 3] == 'F') {

return true;

} else {

return false;

}

}

private static boolean isFmt(byte[] bytes, int start) {

if (bytes[start + 0] == 'f' && bytes[start + 1] == 'm' && bytes[start + 2] == 't' && bytes[start + 3] == ' ') {

return true;

} else {

return false;

}

}

private static boolean isData(byte[] bytes, int start) {

if (bytes[start + 0] == 'd' && bytes[start + 1] == 'a' && bytes[start + 2] == 't' && bytes[start + 3] == 'a') {

return true;

} else {

return false;

}

}

/\*\*

\* 将int转化为byte[]

\*/

private static byte[] intToByteArray(int data) {

return ByteBuffer.allocate(4).order(ByteOrder.LITTLE\_ENDIAN).putInt(data).array();

}

/\*\*

\* 将short转化为byte[]

\*/

private static byte[] shortToByteArray(short data) {

return ByteBuffer.allocate(2).order(ByteOrder.LITTLE\_ENDIAN).putShort(data).array();

}

/\*\*

\* 将byte[]转化为short

\*/

private static short byteArrayToShort(byte[] b) {

return ByteBuffer.wrap(b).order(ByteOrder.LITTLE\_ENDIAN).getShort();

}

/\*\*

\* 将byte[]转化为int

\*/

private static int byteArrayToInt(byte[] b) {

return ByteBuffer.wrap(b).order(ByteOrder.LITTLE\_ENDIAN).getInt();

}

/\*\*

\* 头部部分信息

\*/

static class Header {

public int fileSize;

public int fileSizeOffset;

public int dataSize;

public int dataSizeOffset;

public int dataOffset;

public FileInputStream dataInputStream;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144

这里int、short相互转化的时候需要考虑大小端的问题。   
免责声明：（晕。。。。。。）   
此文章内容仅作为参考，能力有限，难免会有一些不足之处，欢迎大家指正，相互学习，共同进步。。。哈哈