/\*\*

\* Created by ZhouMeng on 2018/8/31.

\* 用于实现录音、暂停、继续、停止、播放

\* 最近看了下pcm和wav，内容真多，要是有一些参数不理解的，可以查阅资料

\* PCM BufferSize = 采样率 \* 采样时间 \* 采样位深 / 8 \* 通道数（Bytes）

\*/

public class AudioRecorder {

private final static int AUDIO\_INPUT = MediaRecorder.AudioSource.DEFAULT;

//音频输入-麦克风

// private final static int AUDIO\_INPUT = MediaRecorder.AudioSource.MIC;

//为语音通信（如VoIP）调谐的麦克风音频源

// private final static int AUDIO\_INPUT = MediaRecorder.AudioSource.VOICE\_COMMUNICATION;

//为语音识别而调谐的麦克风音频源

// private final static int AUDIO\_INPUT = MediaRecorder.AudioSource.VOICE\_RECOGNITION;

/\*\*

\* 采样率即采样频率，采样频率越高，能表现的频率范围就越大

\* 设置音频采样率，44100是目前的标准，但是某些设备仍然支持22050，16000，11025

\*/

private final static int AUDIO\_SAMPLE\_RATE = 16000;

// 设置音频的录制的声道CHANNEL\_IN\_STEREO为双声道，CHANNEL\_IN\_MONO为单声道

// private final static int AUDIO\_CHANNEL = AudioFormat.CHANNEL\_IN\_MONO;

private final static int AUDIO\_CHANNEL = AudioFormat.CHANNEL\_IN\_MONO;

/\*\*

\* 位深度也叫采样位深，音频的位深度决定动态范围

\* 音频数据格式:PCM 16位每个样本。保证设备支持。PCM 8位每个样本。不一定能得到设备支持。

\*/

private final static int AUDIO\_ENCODING = AudioFormat.ENCODING\_PCM\_16BIT;

// 缓冲区字节大小

private int bufferSizeInBytes = 0;

//录音对象

private AudioRecord audioRecord;

//用来回调，转码后的文件绝对路径

private final Listeners<IRecordParser> parsers = Listeners.listeners(IRecordParser.class);

/\*\*

\* 创建带有缓存的线程池

\* 当执行第二个任务时第一个任务已经完成，会复用执行第一个任务的线程，而不用每次新建线程。

\* 如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

\* 一开始选择错误，选用newSingleThreadExecutor，导致停止后在录制，出现一堆问题

\*/

private final ExecutorService cachedThreadPool = Executors.newCachedThreadPool();

/\*\*

\* 重置，删除所有的pcm文件

\*/

private Recorder recorder = null;

private boolean isRecording = false;

public void addRecordParser(IRecordParser parser) {

parsers.add(parser);

}

public void removeRecordParser(IRecordParser parser) {

parsers.remove(parser);

}

public boolean isRecording() {

return isRecording;

}

public void startRecord() {

createDefaultAudio();

start();

isRecording = true;

recorder = new Recorder(audioRecord);

cachedThreadPool.execute(recorder);

}

public void stopRecord() {

try {

isRecording = false;

if (recorder != null) {

recorder.stop();

recorder = null;

}

if (audioRecord != null) {

audioRecord.stop();

audioRecord.release();

audioRecord = null;

}

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 创建默认的录音对象

\*/

@RequiresApi(api = Build.VERSION\_CODES.LOLLIPOP)

private void createDefaultAudio() {

// 获得缓冲区字节大小

bufferSizeInBytes = AudioRecord.getMinBufferSize(AUDIO\_SAMPLE\_RATE, AUDIO\_CHANNEL, AUDIO\_ENCODING);

audioRecord = new AudioRecord(AUDIO\_INPUT, AUDIO\_SAMPLE\_RATE, AUDIO\_CHANNEL, AUDIO\_ENCODING, bufferSizeInBytes);

}

/\*\*

\* 开始录音

\*/

public void start() {

if (audioRecord != null) {

audioRecord.startRecording();

}

}

/\*\*

\* 暂停录音

\*

\* @deprecated 目前暂停功能暂时不可用

\*/

@Deprecated

public void pauseRecord() {

// TODO: 2021/6/16 录音暂停

if (audioRecord != null) {

audioRecord.stop();

}

}

private class Recorder implements Runnable {

private AudioRecord record;

private boolean stop = false;

public Recorder(AudioRecord record) {

this.record = record;

}

void stop() {

stop = true;

}

@Override

public void run() {

Lig.d("开始录音");

// new一个byte数组用来存一些字节数据，大小为缓冲区大小

parsers.listen().onStart();

int readSize;

byte[] audioData = new byte[bufferSizeInBytes];

while (!stop) {

try {

readSize = record.read(audioData, 0, bufferSizeInBytes);

if (readSize != bufferSizeInBytes) {

Lig.d("readSize", readSize, bufferSizeInBytes);

} else {

if (readSize > 0) {

if (!stop) {

parsers.listen().parse(audioData, readSize);

}

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

parsers.listen().close();

record = null;

Lig.d("录音结束");

}

}

}