杰理音频编码库开发说明

声明

- 本项目所参考、使用技术必须全部来源于公知技术信息，或自主创新设计。
- 本项目不得使用任何未经授权的第三方知识产权的技术信息。
- 如个人使用未经授权的第三方知识产权的技术信息，造成的经济损失和法律后果由个人承担

历史版本

版本	更新概述	修改人	备注
1.0.1	1. 增加OpusOption解码参数 2. 新增OpusManager解码接口【decodeFile】和【startDecodeStream】	钟卓成	
1.0.0	1.增加OpusManager的重要接口说明 2.增加OpusManager的使用示例	钟卓成	

一、概述

本文档是为了方便用户接入杰理音频编解码库而创建，用户可以通过该文档快速接入杰理音频编解码库功能。
杰理音频编解码库已支持功能：
1.Opus编解码
  1.1 Opus解码文件
  1.2 Opus解码数据流
  1.3 Opus编码文件
  1.4 opus编码数据流
2.Speex编解码
  2.1 Speex解码文件
  2.2 Speex解码数据流
  2.3 Speex编码文件

二、接口说明

2.1 Opus编解码接口说明

OpusManager

```
public class OpusManager {
    public OpusManager() throws OpusException;
    /**
     * 是否正在解码数据流
     *
     * @return 结果
     */
    public boolean isDecodeStream();

    /**
     * 编码文件
     *
     * @param inFilePath  输入文件路径 (pcm文件)
     * @param outFilePath 输出文件路径 (opus文件)
     * @param callback    编码事件回调
     */
    public void encodeFile(String inFilePath, String outFilePath, OnStateCallback
callback);

    /**
     * 解码文件
     *
     * @param inFilePath  输入文件路径 (opus文件)
     * @param outFilePath 输出文件路径 (pcm文件)
     * @param callback    解码事件回调
     * @deprecated 已被 {@link #decodeFile(String, String, OpusOption,
OnStateCallback)} 代替
     */
    @Deprecated
```

```
    public void decodeFile(String inFilePath, String outFilePath, OnStateCallback
callback);

    /**
     * 解码OPUS文件
     *
     * @param inFilePath  输入OPUS编码文件路径
     * @param outFilePath 输出OPUS解码文件路径
     * @param option      解码参数
     * @param callback    状态回调
     */
    public void decodeFile(String inFilePath, String outFilePath, OpusOption
option, OnStateCallback callback);

    /**
     * 开始解码流数据
     *
     * @param callback 解码事件回调
     * @deprecated 已被 {@link #startDecodeStream(OpusOption,
OnDecodeStreamCallback)} 代替
     */
    @Deprecated
    public void startDecodeStream(OnDecodeStreamCallback callback);

    /**
     * 开始解码OPUS数据
     *
     * @param option   解码参数
     * @param callback 状态回调
     */
    public void startDecodeStream(OpusOption option, OnDecodeStreamCallback
callback);

    /**
     * 停止解码流数据
     */
    public void stopDecodeStream();

    /**
     * 写入需要解码的opus数据
     *
     * @param data opus数据
     */
    public void writeAudioStream(byte[] data);

    /**
     * 是否正在编码流数据
     *
     * @return 结果
     */
    public boolean isEncodeStream();

    /**
     * 开始编码流数据
     *
     * @param callback 编码事件回调
```

```
     */
    public void startEncodeStream(OnEncodeStreamCallback callback);

    /**
     * 停止编码流数据
     */
    public void stopEncodeStream();

    /**
     * 写入待编码的pcm数据
     *
     * @param data pcm数据
     */
    public void writeEncodeStream(byte[] data);

    /**
     * 释放资源
     */
    public void release();
}
```

2.2 Speex编解码接口说明

SpeexManager

```
public class SpeexManager {

    public SpeexManager() throws SpeexException;

    /**
     * 是否正在解码数据流
     *
     * @return 结果
     */
    public boolean isDecodeStream();

    /**
     * 编码文件
     *
     * @param inFilePath  输入文件路径 (pcm文件)
     * @param outFilePath 输出文件路径 (speex文件)
     * @param callback    编码事件回调
     */
    public void encodeFile(String inFilePath, String outFilePath, OnStateCallback
callback);

    /**
     * 解码文件
     *
     * @param inFilePath  输入文件路径 (speex文件)
     * @param outFilePath 输出文件路径 (pcm文件)
     * @param callback    解码事件回调
     */
```

```
    public void decodeFile(String inFilePath, String outFilePath, OnStateCallback
callback);

    /**
     * 开始解码流数据
     *
     * @param callback 解码事件回调
     */
    public void startDecodeStream(OnDecodeStreamCallback callback);

    /**
     * 停止解码流数据
     */
    public void stopDecodeStream();

    /**
     * 写入需要解码的speex数据
     *
     * @param data speex数据
     */
    public void writeAudioStream(byte[] data);

    /**
     * 释放资源
     */
    public void release();
}
```

2.3 状态事件回调器

OnStateCallback

```
public interface OnStateCallback {
    /**
     * 操作开始
     */
    void onStart();

    /**
     * 操作结束
     *
     * @param outFilePath 输出路径 （如果是流式回调，则是成功信息）
     */
    void onComplete(String outFilePath);

    /**
     * 操作异常
     *
     * @param code    错误码
     * @param message 错误描述
     */
    void onError(int code, String message);
```

```
}
```

2.4 解码流事件回调器

OnDecodeStreamCallback

```
public interface OnDecodeStreamCallback extends OnStateCallback {
    /**
     * 解码数据流回调
     *
     * @param data 解码数据
     */
    void onDecodeStream(byte[] data);
}
```

2.5 编码流事件回调器

OnEncodeStreamCallback

```
public interface OnEncodeStreamCallback extends OnStateCallback {
    /**
     * 编码数据流回调
     *
     * @param data 编码数据
     */
    void onEncodeStream(byte[] data);
}
```

2.6 常量定义

DecodeConstant

```
public class DecodeConstant {

    //decode type
    public final static int TYPE_DECODE_FILE = 1;
    public final static int TYPE_ENCODE_FILE = 2;
    public final static int TYPE_DECODE_STREAM = 3;
    public final static int TYPE_ENCODE_STREAM = 4;

    //decode state
    public final static int STATE_IDLE = 0;
    public final static int STATE_WORKING = 1;
    public final static int STATE_ERROR = 2;

    //data type
    public final static int DATA_TYPE_PCM = 0;
    public final static int DATA_TYPE_SPX = 1;
```

```
    public final static int DATA_TYPE_OPUS = 2;

}
```

2.7 错误码

ErrorCode

```
public class ErrorCode {
    /**
     * 参数错误
     */
    public final static int ERR_BAD_ARGS = -1;
    /**
     * 文件不存在
     */
    public final static int ERR_FILE_NOT_EXIST = -2;
    /**
     * 不支持功能
     */
    public final static int ERR_NOT_SUPPORT_FUNCTION = -3;
    /**
     * 正在处理中
     */
    public final static int ERR_IN_PROCESS = -4;

    /**
     * 未知异常
     */
    public final static int ERR_UNKNOWN_EXCEPTION = -999;

}
```

三、使用说明

3.1 Opus编解码使用说明

使用步骤

1. 初始化OpusManager对象
2. 使用编解码接口

3.1.1 Opus解码参数

```
public class OpusOption {
    /**
     * 是否具有协议头
     */
    private boolean hasHead;
    /**
     * 通道数。取值范围:[1, 2]
     */
    private int channel = 1;
    /**
     * 采样率。参考值:{8000, 16000, 32000, 48000}
     */
    private int sampleRate = 16000;
    /**
     * 数据包长度。仅hasHead = false时生效。
     */
    private int packetSize = 40;
    ...
}
```

3.1.2 解码Opus文件

```
/*
 *Step0. 初始化OpusManager对象
 * */
try {
     OpusManager mOpusManager = new OpusManager();
} catch (OpusException e) {
     e.printStackTrace();
     Log.e("zzc", "初始化 opus库失败！！！");
     mOpusManager = null;
}
//Step1.调用解码文件接口
decodeOpusFile(mOpusManager, "test.opus", "dec_res.pcm", new OpusOption());

...

//StepN.当不需要使用OpusManager,记得释放资源
mOpusManager.release();


private void decodeOpusFile(OpusManager manager, String inName, String outName,
OpusOption option) {
        String inFilePath =
getApplicationContext().getExternalCacheDir().getPath() + "/" + inName;
        String outFilePath =
getApplicationContext().getExternalCacheDir().getPath() + "/" + outName;
```

```
        Log.i("zzc", "-decodeOpusFile- inFilePath : " + inFilePath + ",
outFilePath : " + outFilePath + ", " + option);
        manager.decodeFile(inFilePath, outFilePath, option, new OnStateCallback()
{
            @Override
            public void onStart() {
                Log.d("zzc", "解码开始>>> stream");
            }

            @Override
            public void onComplete(String outFilePath) {
                Log.d("zzc", "解码成功>>> " + outFilePath);
            }

            @Override
            public void onError(int code, String message) {
                Log.e("zzc", "解码错误>>> " + code + ", " + message);
            }
        });
}
```

3.1.3 解码Opus数据流

```
/*
 *Step0. 初始化OpusManager对象
 * */
try {
     OpusManager mOpusManager = new OpusManager();
} catch (OpusException e) {
     e.printStackTrace();
     Log.e("zzc", "初始化 opus库失败！！！");
     mOpusManager = null;
}
//Step1.调用解码数据流接口
decodeOpusStream(mOpusManager, option);

//Step2.等待开始回调，开始写入待解码数据
mOpusManager.writeAudioStream(readData);

...

//Step3.当不需要解码数据时，可以停止解码
mOpusManager.stopDecodeStream();

//StepN.当不需要使用OpusManager,记得释放资源
mOpusManager.release();

private void decodeOpusStream(OpusManager manager, OpusOption option) {
        /*
         * 1. 开始解码
         */
        manager.startDecodeStream(option, new OnDecodeStreamCallback() {
            @Override
```

```
            public void onDecodeStream(byte[] data) {
                /*
                 * 3. 获得解码后PCM数据
                 */
                Log.d("zzc", "解码后的PCM流数据 >> ");
            }

            @Override
            public void onStart() {
                Log.d("zzc", "解码开始 >> ");
                //2. 传入待解码数据
                //此时应该等待写入数据
                //比如：蓝牙有数据过来，应该调用
                //manager.writeAudioStream(readData); //写入待解码数据

            }

            @Override
            public void onComplete(String outFilePath) {
                Log.d("zzc", "解码结束 >> " + outFilePath);
            }

            @Override
            public void onError(int code, String message) {
                Log.e("zzc", "解码失败 >> " + code + ", " + message);
            }
        });
    }
```

3.1.4 编码Opus文件

```
/*
 *Step0. 初始化OpusManager对象
 * */
try {
     OpusManager mOpusManager = new OpusManager();
} catch (OpusException e) {
     e.printStackTrace();
     Log.e("zzc", "初始化 opus库失败！！！");
     mOpusManager = null;
}
//Step1.调用解码文件接口
decodeOpusFile(mOpusManager, "test.pcm", "enc_res.opus");

...

//StepN.当不需要使用OpusManager,记得释放资源
mOpusManager.release();

 private void encodeOpusFile(OpusManager manager, String inName, String outName)
{
```

```
        String inFilePath =
getApplicationContext().getExternalCacheDir().getPath() + "/" + inName;
        String outFilePath =
getApplicationContext().getExternalCacheDir().getPath() + "/" + outName;
        Log.i("zzc", "-encodeOpusFile- inFilePath : " + inFilePath + ",
outFilePath : " + outFilePath);
        manager.encodeFile(inFilePath, outFilePath, new OnStateCallback() {
            @Override
            public void onStart() {
                Log.d("zzc", "-encodeOpusFile- 编码开始>>> stream");
            }

            @Override
            public void onComplete(String outFilePath) {
                File file = new File(outFilePath);
                Log.i("zzc", "编码结束>>> " + outFilePath + ", file length = " +
file.length());
            }

            @Override
            public void onError(int code, String message) {
                Log.e("zzc", "编码错误>>> " + code + ", " + message);
            }
        });
    }
```

3.1.5 编码Pcm数据流

```
/*
 *Step0. 初始化OpusManager对象
 * */
try {
     OpusManager mOpusManager = new OpusManager();
} catch (OpusException e) {
     e.printStackTrace();
     Log.e("zzc", "初始化 opus库失败！！！");
     mOpusManager = null;
}
//Step1.调用编码数据流接口
encodePcmStream(mOpusManager);

//Step2.等待开始回调，开始写入待编码数据
mOpusManager.writeEncodeStream(readData);

...

//Step3.当不需要编码数据时，可以停止编码
mOpusManager.stopEncodeStream();

//StepN.当不需要使用OpusManager,记得释放资源
mOpusManager.release();

private void encodePcmStream(OpusManager manager) {
```

```
        /*
         * 1. 开始编码
         */
        manager.startEncodeStream(new OnEncodeStreamCallback() {
            @Override
            public void onEncodeStream(byte[] data) {
                /*
                 * 3. 获得编码后Opus数据
                 */
                Log.d("zzc", "编码后的OPUS流数据 >> " + data.length);
            }

            @Override
            public void onStart() {
                Log.d("zzc", "编码开始 >> ");
                //2.传入待编码数据
                //此时应该等待写入数据
                //比如：蓝牙有数据过来，应该调用
                //manager.writeEncodeStream(readData); //写入待编码数据

            }

            @Override
            public void onComplete(String outFilePath) {
                Log.d("zzc", "编码结束 >> " + outFilePath);
            }

            @Override
            public void onError(int code, String message) {
                Log.e("zzc", "编码失败 >> " + code + ", " + message);
            }
        });
    }
```

3.2 Speex编解码使用说明

类似Opus的使用，省略