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	日期	更新概述	修改人
2.1.0	2025/5/21	1. 增加标准OPUS格式和杰理OPUS格式的兼容 1.1 解码OPUS文件 1.2 解码OPUS数据流 1.3 编码OPUS文件 1.4 编码PCM数据流程	钟卓成
2.0.0	2025/5/19	1. 增加转码OGG文件的接口 2. 增加转码OGG数据流的接口	钟卓成
1.2.0	2024/1/16	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	2021/7/17	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
  1.5 转码OGG文件
  1.6 转码OGG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;

    /**
     * 编码文件
     * <p>
     * 格式: 单声道, 16K, PCM_16Bit, 不带协议头, 40Byte帧长
     * </p>
     *
     * @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);

    /**
     * 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 option      解码参数
     * @param callback    编码事件回调
     */
    public void encodeFile(String inFilePath, String outFilePath, OpusOption
option, OnStateCallback callback);

    /**
     * 解码OPUS文件
     * <p>
     * 格式: 单声道, 16K, PCM_16Bit, 不带协议头, 40Byte帧长
     * </p>
     *
     * @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
     */
    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);

    /**
     * 是否正在解码数据流
     *
     * @return 结果
     */
```

```
    public boolean isDecodeStream();

    /**
     * 开始解码流数据
     * <p>
     * 格式: 单声道，16K，PCM_16Bit，不带协议头，40Byte帧长
     * </p>
     *
     * @param callback 解码事件回调
     */
    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
     * <p>
     * 格式: 单声道，16K，PCM_16Bit，不带协议头，40Byte帧长
     * </p>
     *
     * @param callback 编码事件回调
     */
    public void startEncodeStream(OnEncodeStreamCallback callback);

    /**
     * 开始编码流数据
     *
     * @param option   OPUS编解码参数
     * @param callback 编码事件回调
     */
```

```
    public void startEncodeStream(OpusOption option, 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);

    /**
     * 是否正在转码
     *
     * @return boolean 结果
     */
    public boolean isTranscoding();

    /**
     * 转码OGG文件
     *
     * @param inFilePath String OPUS裸数据文件路径
     * @param outFilePath String OGG文件路径
     * @param callback OnStateCallback 状态回调器
     */
    public void transcodingOggFile(String inFilePath, String outFilePath,
OnStateCallback callback);

    /**
     * 转码OGG文件
     *
     * @param inFilePath String OPUS裸数据文件路径
     * @param outFilePath String OGG文件路径
     * @param configure OggConfigure OGG封装配置
     * @param callback OnStateCallback 状态回调器
     */
    public void transcodingOggFile(String inFilePath, String outFilePath,
OggConfigure configure, OnStateCallback callback);

    /**
     * 开始转码流式数据
     *
     * @param callback OnDecodeStreamCallback 解码事件回调
     */
    public void startTranscodingStream(OnDecodeStreamCallback callback);

    /**
     * 开始转码流式数据
     *
     * @param configure OggConfigure OGG封装配置
     * @param callback OnDecodeStreamCallback 解码事件回调
     */
```

```
    public void startTranscodingStream(OggConfigure configure,
OnDecodeStreamCallback callback);

    /**
     * 停止转码流数据
     */
    public void stopTranscodingStream();

    /**
     * 写入待转码的OPUS数据
     *
     * @param data opus数据
     */
    public void writeTransCodingStream(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

- 编解码类型

码值	常量	说明
1	TYPE_DECODE_FILE	解码文件
2	TYPE_ENCODE_FILE	编码文件
3	TYPE_DECODE_STREAM	解码数据流
4	TYPE_ENCODE_STREAM	编码数据流
5	TYPE_TRANSCODING_OGG_FILE	转码OGG文件
6	TYPE_TRANSCODING_OGG_STREAM	转码OGG数据流

- 编解码状态

码值	常量	说明
0	STATE_IDLE	空闲状态
1	STATE_WORKING	工作状态
2	STATE_ERROR	错误状态

- 音频类型

码值	常量	说明
0	DATA_TYPE_PCM	PCM数据
1	DATA_TYPE_SPX	SPEEX数据
2	DATA_TYPE_OPUS	OPUS数据

- 版本信息
  - getLibVersionName --- 返回库的版本名称
  - getLibVersionCode --- 返回库的版本号

2.7 错误码

ErrorCode

码值	常量	说明
0	ERR_NONE	没有错误/操作成功
-1	ERR_BAD_ARGS	参数错误
-2	ERR_BUFFER_TOO_SMALL	BUFFER空间不足
-3	ERR_INTERNAL_ERROR	内部错误
-4	ERR_INVALID_PACKET	错误数据帧
-5	ERR_UNIMPLEMENTED	不支持的请求号
-6	ERR_INVALID_STATE	编码器或解析器结构无效或已释放
-7	ERR_ALLOC_FAIL	分配空间失败
-128	ERR_OUTPUT_EXCEPTION	输出数据异常
-129	ERR_ENC_FAILURE	编码失败

码值	常量	说明
-130	ERR_OPEN_FILE	打开文件失败
-1000	ERR_NONE_INIT	没有初始化
-1001	ERR_NOT_SUPPORT_FUNCTION	不支持功能
-1002	ERR_FILE_NOT_EXIST	文件不存在
-1003	ERR_IN_PROCESS	正在处理中
-1004	ERR_OP_FAIL	操作失败
-2000	ERR_IO_EXCEPTION	IO异常

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

3.1.2.1 解码杰理OPUS文件

```
String inFilePath = "OPUS文件路径";
String outFilePath = "PCM输出文件路径";
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isDecodeStream()) return; //正在解码中
    //执行解码文件接口
    opusManager.decodeFile(inFilePath, outFilePath, new OnStateCallback() {
        @Override
        public void onStart() {
            //开始解码
        }

        @Override
        public void onComplete(String outFilePath) {
            //解码成功
            //outFilePath --- 输出文件路径
        }

        @Override
        public void onError(int code, String message) {
            //解码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //销毁OPUS管理器
    //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失
}
```

3.1.2.2 解码标准OPUS文件

```
String inFilePath = "OPUS文件路径";
String outFilePath = "PCM输出文件路径";
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isDecodeStream()) return; //正在解码中
    OpusOption option = new OpusOption()
            .setHasHead(true)
            .setSampleRate(16000)
            .setChannel(1);
    //执行解码文件接口
```

```
    opusManager.decodeFile(inFilePath, outFilePath, option, new OnStateCallback()
{
        @Override
        public void onStart() {
            //开始解码
        }

        @Override
        public void onComplete(String outFilePath) {
            //解码成功
            //outFilePath --- 输出文件路径
        }

        @Override
        public void onError(int code, String message) {
            //解码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //销毁OPUS管理器
    //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.3 解码Opus数据流

3.1.3.1 解码杰理OPUS数据流

```
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isDecodeStream()) return; //正在解码中
    //执行解码流数据接口
    opusManager.startDecodeStream(new OnDecodeStreamCallback() {
        @Override
        public void onDecodeStream(byte[] data) {
            //回调解码数据
        }

        @Override
        public void onStart() {
            //开始解码
        }

        @Override
        public void onComplete(String outFilePath) {
             //解码成功
             //outFilePath --- 结束信息
        }
```

```
        @Override
        public void onError(int code, String message) {
             //解码失败
             //code --- 错误码
             //message --- 错误描述
        }
    });
    //添加解码数据
    byte[] opusData = new byte[0]; //需要解码的OPUS压缩数据
    opusManager.writeAudioStream(opusData);
    //停止流式解码
    opusManager.stopDecodeStream();
    //销毁OPUS管理器
    //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.3.2 解码标准OPUS数据流

```
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isDecodeStream()) return; //正在解码中
    OpusOption option = new OpusOption()
            .setHasHead(true)
            .setSampleRate(16000)
            .setChannel(1);
    //执行解码流数据接口
    opusManager.startDecodeStream(option, new OnDecodeStreamCallback() {
        @Override
        public void onDecodeStream(byte[] data) {
            //回调解码数据
        }

        @Override
        public void onStart() {
            //开始解码
        }

        @Override
        public void onComplete(String outFilePath) {
            //解码成功
            //outFilePath --- 结束信息
        }

        @Override
        public void onError(int code, String message) {
            //解码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //添加解码数据
```

```
    byte[] opusData = new byte[0]; //需要解码的OPUS压缩数据
    opusManager.writeAudioStream(opusData);
    //停止流式解码
    opusManager.stopDecodeStream();
    //销毁OPUS管理器
    //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.4 编码Opus文件

3.1.4.1 编码杰理OPUS文件

```
String inFilePath = "PCM文件路径";
String outFilePath = "OPUS输出文件路径";
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isEncodeStream()) return; //正在编码中
    //执行编码文件接口
    opusManager.encodeFile(inFilePath, outFilePath, new OnStateCallback() {
        @Override
        public void onStart() {
            //开始编码
        }

        @Override
        public void onComplete(String outFilePath) {
            //编码成功
            //outFilePath --- 输出文件路径
        }

        @Override
        public void onError(int code, String message) {
            //编码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //销毁OPUS管理器
    //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.4.2 编码标准OPUS文件

```
String inFilePath = "PCM文件路径";
String outFilePath = "OPUS输出文件路径";
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isEncodeStream()) return; //正在编码中
    OpusOption option = new OpusOption()
            .setHasHead(true)
            .setSampleRate(16000)
            .setChannel(1);
    //执行编码文件接口
    opusManager.encodeFile(inFilePath, outFilePath, option, new OnStateCallback()
{
        @Override
        public void onStart() {
            //开始编码
        }

        @Override
        public void onComplete(String outFilePath) {
            //编码成功
            //outFilePath --- 输出文件路径
        }

        @Override
        public void onError(int code, String message) {
            //编码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //销毁OPUS管理器
    //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.5 编码Pcm数据流

3.1.5.1 编码PCM数据流成为杰理OPUS数据

```
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isEncodeStream()) return; //正在编码中
    //执行编码流数据接口
    opusManager.startEncodeStream(new OnEncodeStreamCallback() {

        @Override
```

```
        public void onEncodeStream(byte[] data) {
            //回调编码数据
        }

        @Override
        public void onStart() {
            //开始编码
        }

        @Override
        public void onComplete(String outFilePath) {
            //编码成功
            //outFilePath --- 结束信息
        }

        @Override
        public void onError(int code, String message) {
            //编码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //添加编码数据
    byte[] pcmData = new byte[0]; //需要编码的PCM数据
    opusManager.writeEncodeStream(pcmData);
    //停止流式编码
    opusManager.stopEncodeStream();
    //销毁OPUS管理器
    // opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.5.2 编码PCM数据流成为标准OPUS数据

```
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isEncodeStream()) return; //正在编码中
    OpusOption option = new OpusOption()
            .setHasHead(true)
            .setSampleRate(16000)
            .setChannel(1);
    //执行编码流数据接口
    opusManager.startEncodeStream(option, new OnEncodeStreamCallback() {

        @Override
        public void onEncodeStream(byte[] data) {
            //回调编码数据
        }

        @Override
        public void onStart() {
            //开始编码
```

```
        }

         @Override
         public void onComplete(String outFilePath) {
             //编码成功
             //outFilePath --- 结束信息
         }

         @Override
         public void onError(int code, String message) {
             //编码失败
             //code --- 错误码
             //message --- 错误描述
         }
     });
     //添加编码数据
     byte[] pcmData = new byte[0]; //需要编码的PCM数据
     opusManager.writeEncodeStream(pcmData);
     //停止流式编码
     opusManager.stopEncodeStream();
     //销毁OPUS管理器
     //opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.1.6 转码OGG文件

将杰理OPUS数据转换成标准OGG_OPUS文件

```
String inFilePath = "OPUS文件路径";
String outFilePath = "OGG输出文件路径";
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isTranscoding()) return; //正在转码中
    OggConfigure configure = new OggConfigure(); //OGG封装配置
    //执行转码OGG文件接口
    opusManager.transcodingOggFile(inFilePath, outFilePath, configure, new
OnStateCallback() {
        @Override
        public void onStart() {
            //开始转码
        }

        @Override
        public void onComplete(String outFilePath) {
            //转码成功
            //outFilePath --- 输出文件路径
        }
```

```
        @Override
        public void onError(int code, String message) {
            //转码失败
            //code --- 错误码
            //message --- 错误描述
         }
     });
     //销毁OPUS管理器
//     opusManager.release();
} catch (OpusException e) {
     e.printStackTrace(); //初始化失败
}
```

3.1.6.1 OggConfigure

OGG配置参数

```
public class OggConfigure {
    /**
     * 每页帧数量
     */
    private int pageFrameCount = 30;
    /**
     * 帧长
     */
    private int frameLen = 40;
}
```

3.1.7 转码OGG数据流

将杰理OPUS数据转换成标准OGG_OPUS数据流程

```
try {
    OpusManager opusManager = new OpusManager();
    if (opusManager.isTranscoding()) return; //正在转吗中
    OggConfigure configure = new OggConfigure(); //OGG封装配置
    //执行转码流数据接口
    opusManager.startTranscodingStream(configure, new OnDecodeStreamCallback() {
        @Override
        public void onDecodeStream(byte[] data) {
            //回调转码数据
        }

        @Override
        public void onStart() {
            //开始转码
```

```
        }

        @Override
        public void onComplete(String outFilePath) {
            //转码成功
            //outFilePath --- 结束信息
        }

        @Override
        public void onError(int code, String message) {
            //转码失败
            //code --- 错误码
            //message --- 错误描述
        }
    });
    //添加转码数据
    byte[] opusData = new byte[0]; //需要转码的OPUS压缩数据
    opusManager.writeTransCodingStream(opusData);
    //停止流式转码
    opusManager.stopTranscodingStream();
    //销毁OPUS管理器
//    opusManager.release();
} catch (OpusException e) {
    e.printStackTrace(); //初始化失败
}
```

3.2 Speex编解码使用说明

3.2.1 解码文件

```
String inFilePath = "SPEEX文件路径";
String outFilePath = "PCM输出文件路径";
try {
    SpeexManager speexManager = new SpeexManager();
    if (speexManager.isDecodeStream()) return; //正在解码中
    //执行解码文件接口
    speexManager.decodeFile(inFilePath, outFilePath, new OnStateCallback() {
        @Override
        public void onStart() {
            //开始解码
        }

        @Override
        public void onComplete(String outFilePath) {
            //解码成功
            //outFilePath --- 输出文件路径
        }

        @Override
```

```
        public void onError(int code, String message) {
            //解码失败
            //code --- 错误码
            //message --- 错误信息
        }
    });
    //不再使用，需要销毁对象
//    speexManager.release();
} catch (SpeexException e) {
    e.printStackTrace(); //初始化失败
}
```

3.2.2 解码流数据

```
try {
    SpeexManager speexManager = new SpeexManager();
    if (speexManager.isDecodeStream()) return; //正在解码中
    //执行流式解码接口
    speexManager.startDecodeStream(new OnDecodeStreamCallback() {
        @Override
        public void onDecodeStream(byte[] data) {
            //回调解码数据
        }

        @Override
        public void onStart() {
            //开始解码
        }

        @Override
        public void onComplete(String outFilePath) {
            //解码成功
            //outFilePath --- 输出文件路径
        }

        @Override
        public void onError(int code, String message) {
            //解码失败
            //code --- 错误码
            //message --- 错误信息
        }
    });
    //添加解码数据
    byte[] speexData = new byte[0]; //待解码的SPEEX数据
    speexManager.writeAudioStream(speexData);
    //停止流式解码
    speexManager.stopDecodeStream();
    //不再使用，需要销毁对象
//    speexManager.release();
} catch (SpeexException e) {
    e.printStackTrace(); //初始化失败
```

```
}
```

3.2.3 编码文件

```
String inFilePath = "PCM文件路径";
String outFilePath = "SPEEX输出文件路径";
try {
    SpeexManager speexManager = new SpeexManager();
    //执行编码文件接口
    speexManager.encodeFile(inFilePath, outFilePath, new OnStateCallback() {
        @Override
        public void onStart() {
            //开始编码
        }

        @Override
        public void onComplete(String outFilePath) {
            //编码成功
            //outFilePath --- 输出文件路径
        }

        @Override
        public void onError(int code, String message) {
            //编码失败
            //code --- 错误码
            //message --- 错误信息
        }
    });
    //不再使用，需要销毁对象
//    speexManager.release();
} catch (SpeexException e) {
    e.printStackTrace(); //初始化失败
}
```