使用react-native-ffmpeg示例

笔记本:	我的第一个笔记本		
创建时间:	2021/3/22 19:26	**更新时间:**	2021/3/22 20:47
作者:	136iqdlb602		
URL:	file:///C:/Users/Lenovo/Desktop/使用react-native-ffmpeg示例.docx		

因为react-native-ffmpeg工作势必涉及到本地文件系统的操作，所以顺便把react-native-fs一起装了
首先配置fs
安装:

```
npm i react-native-fs
```

Settings.gradle最上面加上

```
include ':react-native-fs'
project(':react-native-
fs').projectDir=newFile(rootProject.projectDir,'../node_modules/react-native-
fs/android')
```

官方文档有一点老，很多提到的设置操作实际发现不需要进行（而且会报错），这样应该就可以运行了。

然后是ffmpeg
安装:

```
npm i react-native-ffmpeg
```

在android/build.gradle 文件buildscript.ext框下加上

```
buildscript {
    ...
    ext {
        ...
        reactNativeFFmpegPackage = "full"
    }
    ...
}
```

最后启用app读写文件的权限

在 main/res/AndroidManifest.xml内

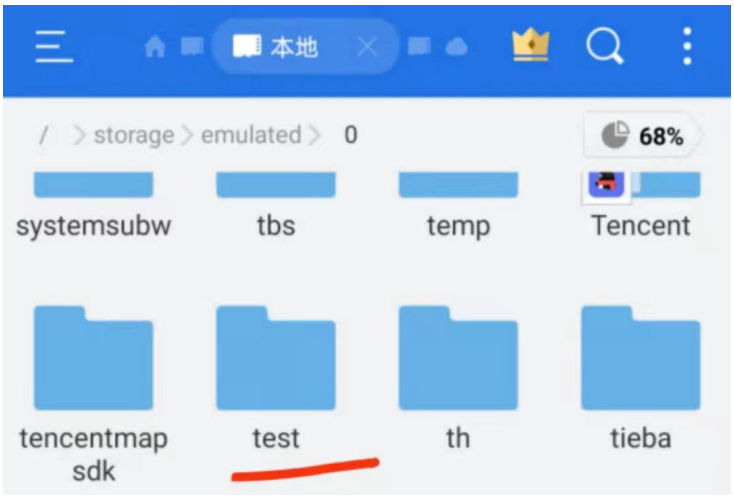
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.kgeapp">
    ...
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    ...
</manifest>
```

测试代码示例

```
import { LogLevel, RNFFmpeg, RNFFprobe } from 'react-native-ffmpeg';
import RNFS from 'react-native-fs';


const ffmpegCommand = '-i ' + RNFS.ExternalStorageDirectoryPath+
'/test/wx_camera_1582897957311.mp4 ';
RNFFmpeg.execute(ffmpegWatermarkCommand).then(result => console.log(`FFmpeg
process exited with rc=${result}.`));
```

RNFS.ExternalStorageDirectoryPath代表手机文件系统根目录，在我的手机上是storage/emulated/0



react-native-ffmpeg简单使用示例

```
import { LogLevel, RNFFmpeg } from 'react-native-ffmpeg';

//直接执行文本命令，这条-i：输入文件 -c：v 输出视频格式 -c：a 输出音频格式
RNFFmpeg.execute('-i file1.mp4 -c:v mpeg4 file2.mp4').then(result =>
console.log(`FFmpeg process exited with rc=${result}.`));

//上一条指令的参数化形式
RNFFmpeg.executeWithArguments(["-i", "file1.mp4", "-c:v", "mpeg4",
"file2.mp4"]).then(result => console.log(`FFmpeg process exited with
rc=${result}.`));

//执行异步指令
RNFFmpeg.executeAsync('-i file1.mp4 -c:v mpeg4 file2.mp4', completedExecution =>
{
    if (completedExecution.returnCode === 0) {
       console.log("FFmpeg process completed successfully");
    } else {
       console.log(`FFmpeg process failed with
rc=${completedExecution.returnCode}.`);
    }
}).then(executionId => console.log(`Async FFmpeg process started with executionId
${executionId}.`));

//执行FFprobe命令（计算音视频信息的ffmpeg组件），
import { LogLevel, RNFFprobe } from 'react-native-ffmpeg';
RNFFprobe.execute('-i file1.mp4').then(result => console.log(`FFprobe process
exited with rc=${result}.`));

//检查上一条指令返回的信息
RNFFmpegConfig.getLastReturnCode().then(returnCode => {
    console.log(`Last return code: ${returnCode}`);
});

RNFFmpegConfig.getLastCommandOutput().then(output => {
    console.log(`Last command output: ${output}`);
```

```
});

//停止所有指令
RNFFmpeg.cancel();
//停止特定指令
RNFFmpeg.cancelExecution(executionId);
//注意以上两条指令不会等到真的停止了才返回，而是立即返回。

//获取文件信息
//获取全部信息
RNFFprobe.getMediaInformation('<file path or uri>').then(information => {
    console.log('Result: ' + JSON.stringify(information));
});
//获取指定信息
RNFFprobe.getMediaInformation('<file path or uri>').then(information => {
    if (information.getMediaProperties() !== undefined) {
        if (information.getMediaProperties().filename !== undefined) {
            console.log(`Path: ${information.getMediaProperties().filename}`);
        }
        if (information.getMediaProperties().format_name !== undefined) {
            console.log(`Format:
${information.getMediaProperties().format_name}`);
        }
        if (information.getMediaProperties().bit_rate !== undefined) {
            console.log(`Bitrate: ${information.getMediaProperties().bit_rate}`);
        }
        if (information.getMediaProperties().duration !== undefined) {
            console.log(`Duration:
${information.getMediaProperties().duration}`);
        }
        if (information.getMediaProperties().start_time !== undefined) {
            console.log(`Start time:
${information.getMediaProperties().start_time}`);
        }
        if (information.getMediaProperties().nb_streams !== undefined) {
            console.log(`Number of streams:
${information.getMediaProperties().nb_streams.toString()}`);
        }
        let tags = information.getMediaProperties().tags;
        if (tags !== undefined) {
            Object.keys(tags).forEach((key) => {
                console.log(`Tag: ${key}:${tags[key]}`);
            });
        }


        let streams = information.getStreams();
        if (streams !== undefined) {
            for (let i = 0; i < streams.length; ++i) {
               let stream = streams[i];
               console.log("---");
               if (stream.getAllProperties().index !== undefined) {
                   console.log(`Stream index:
${stream.getAllProperties().index.toString()}`);
               }
               if (stream.getAllProperties().codec_type !== undefined) {
                   console.log(`Stream type:
${stream.getAllProperties().codec_type}`);
               }
               if (stream.getAllProperties().codec_name !== undefined) {
                   console.log(`Stream codec:
${stream.getAllProperties().codec_name}`);
               }
            }
        }
    }
});

//重定向log信息到指定地方
logCallback = (log) => {
```

```
    this.appendLog(`${log.executionId}:${log.message}`);
};
...
RNFFmpegConfig.enableLogCallback(this.logCallback);

//返回音视频的实时统计信息
statisticsCallback = (statistics) => {
    console.log(`Statistics; executionId: ${statistics.executionId}, video frame
number: ${statistics.videoFrameNumber}, video fps: ${statistics.videoFps}, video
quality: ${statistics.videoQuality}, size: ${statistics.size}, time:
${statistics.time}, bitrate: ${statistics.bitrate}, speed: ${statistics.speed}`);
};
...
RNFFmpegConfig.enableStatisticsCallback(this.statisticsCallback);

//返回上一次的统计信息
RNFFmpegConfig.getLastReceivedStatistics().then(statistics => console.log('Stats:
' + JSON.stringify(statistics)));

//列出正在运行的操作
RNFFmpeg.listExecutions().then(executionList => {
    executionList.forEach(execution => {
        console.log(`Execution id is ${execution.executionId}`);
        console.log(`Execution start time is ` + new Date(execution.startTime));
        console.log(`Execution command is ${execution.command}`);
    });
});

//设置log等级
RNFFmpegConfig.setLogLevel(LogLevel.AV_LOG_WARNING);
```

附：官方示例中对于ffmpeg的api封装

```
//react-native-ffmpeg-test-master\src\react-native-ffmpeg-api-wrapper.js
import React from 'react';
import {RNFFmpeg, RNFFmpegConfig, RNFFprobe} from 'react-native-ffmpeg';
import {ffprint} from './util';

export function ffprint(text) {
    console.log(text.endsWith('\n') ? text.replace('\n', '') : text);
}

export async function executeFFmpeg(command) {
    return await RNFFmpeg.execute(command)
}


export function executeFFmpegWithArguments(commandArguments) {
    RNFFmpeg.executeWithArguments(commandArguments).then(rc => ffprint(`FFmpeg
process exited with rc ${rc}`));
}


export async function executeFFmpegAsync(command, callback) {
    return await RNFFmpeg.executeAsync(command, callback);
}


export function executeFFmpegAsyncWithArguments(commandArguments, callback) {
    RNFFmpeg.executeAsyncWithArguments(commandArguments,
callback).then(executionId => ffprint(`FFmpeg process started with executionId
${executionId}.`));
}


export function executeFFmpegCancel() {
    RNFFmpeg.cancel();
```

```
}


export function executeFFmpegCancelExecution(executionId) {
    RNFFmpeg.cancelExecution(executionId);
}


export function getLastCommandOutput() {
    return RNFFmpegConfig.getLastCommandOutput();
}


export function getLogLevel() {
    return RNFFmpegConfig.getLogLevel();
}


export function setLogLevel(logLevel) {
    return RNFFmpegConfig.setLogLevel(logLevel);
}


export function enableLogCallback(logCallback) {
    RNFFmpegConfig.enableLogCallback(logCallback);
}


export function enableStatisticsCallback(statisticsCallback) {
    RNFFmpegConfig.enableStatisticsCallback(statisticsCallback);
}


export function listExecutions() {
    RNFFmpeg.listExecutions().then(executionList => {
        executionList.forEach(execution => {
            ffprint(`Execution id is ${execution.executionId}`);
            ffprint(`Execution start time is ` + new Date(execution.startTime));
            ffprint(`Execution command is ${execution.command}`);
        });
    });
}


export async function executeFFprobe(command) {
    return await RNFFprobe.execute(command);
}


export function executeFFprobeWithArguments(commandArguments) {
    RNFFprobe.executeWithArguments(commandArguments).then(rc => ffprint(`FFprobe
process exited with rc ${rc}`));
}


export function resetStatistics() {
    RNFFmpegConfig.resetStatistics();
}


export function parseArguments(command) {
    return RNFFmpeg.parseArguments(command);
}


export function getFFmpegVersion() {
    return RNFFmpegConfig.getFFmpegVersion();
}
```

```
export function getPlatform() {
    return RNFFmpegConfig.getPlatform();
}


export function getPackageName() {
    return RNFFmpegConfig.getPackageName();
}


export function getExternalLibraries() {
    return RNFFmpegConfig.getExternalLibraries();
}


export function getLastReturnCode() {
    return RNFFmpegConfig.getLastReturnCode();
}


export function getLastReceivedStatistics() {
    return RNFFmpegConfig.getLastReceivedStatistics();
}


export function setFontDirectory(path, mapping) {
    RNFFmpegConfig.setFontDirectory(path, mapping);
}


export function setFontconfigConfigurationPath(path) {
    RNFFmpegConfig.setFontconfigConfigurationPath(path);
}


export function setEnvironmentVariable(name, value) {
    RNFFmpegConfig.setEnvironmentVariable(name, value);
}


export function getMediaInformation(path) {
    return RNFFprobe.getMediaInformation(path);
}


export function registerNewFFmpegPipe() {
    return RNFFmpegConfig.registerNewFFmpegPipe();
}


export function writeToPipe(inputPath, pipePath) {
    return RNFFmpegConfig.writeToPipe(inputPath, pipePath);
}
```