阿里巴巴集团

SDK参考手册

安卓媒体推流器SDK参考手册

文档修订历史

版本	作者	工作描述	修订历史	修改日期
1.0	德胜	安卓推流SDK文档		2016-06-21
1.1	德胜	安卓推流SDK文档		2016-07-02
1.2	江浩	安卓推流SDK文档		2016-10-31

参考文献

无

目录

文档说明

本文档面向所有使用该SDK的开发人员、测试人员以及对此感兴趣的用户,要求读者具有一定的Android开发能力。

1. 简介

安卓推流SDK是在安卓平台上使用的软件开发工具包(Soft Development Kit)，为Android开发者提供简单易用的接口，帮助开发者实现Android平台上的推流应用开发。

1.1功能说明

1. 方便快捷、低门槛实现媒体推流功能。用户无须关心内部实现细节，只需要自定义界面既可以实现专业级的推流应用。
2. 推流支持格式：rtmp
3. 编码目前为硬编

1.2安装包说明

推流器SDK的完整下载包中包含demo、doc、jar包、.so文件等：

- 1.demo：主要存放了调用SDK的示例工程，可以帮助用户了解如何使用该SDK。
- 2.jar：推流SDK　java库, SDK建议使用Android Studio进行集成，对于使用eclipse的开发者可以参考Google提供的迁移方法 https://developer.android.com/studio/intro/migrate.html (此地址需要翻墙，也可自行百度解决）,将原应用迁移到Android Studio后再集成SDK。
- 3.添加libs里面给出的j ar依赖，并且将jniLibs整个拷贝到moudle-name/src/main中，然后在moudle的build.gradle中加入以下配置（具体可参考Demo）：

```
splits {
        abi {
            enable true
            reset()
            include 'armeabi-v7a'
        }

    }
```

注意：如果使用其他的第三方库也有对.so的依赖，则需要选取其他第三方库的*armeabi-v7a*的.so，一并放入*moudle-name/src/main/jniLibs/armeabi-v7a*中，如果其他第三方库没有给出armeabi-v7a的.so，则可以使用其armeabi的.so替代

- 4.doc：存放SDK相关接入文档。

1.3推流器性能

- 1.目前推流SDK推流采用的是硬编。
- 2.推流采用FFMPEG推流
- 3.SDK的大小：去除ffmpeg动态库之后SDK对应用的大小增加在1M左右.加上ffmepg在2M左右

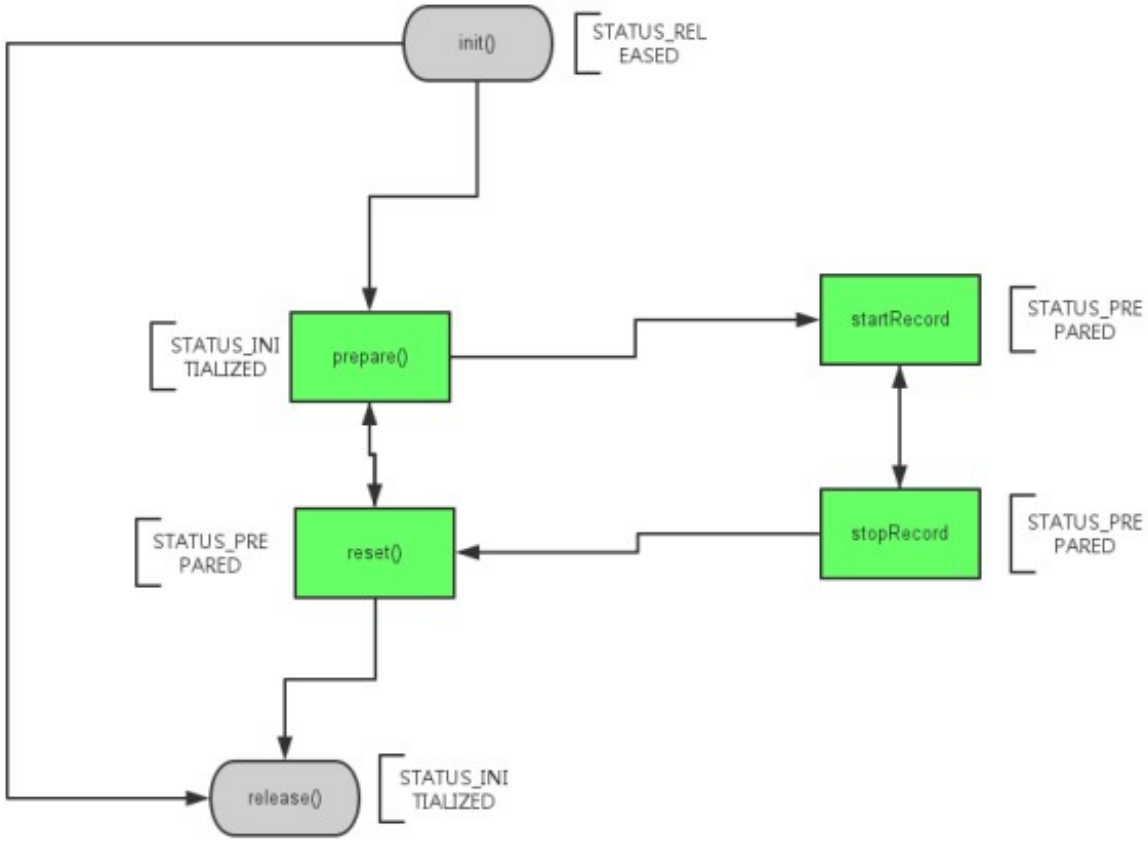
1.4注意事项

- 1.推流器SDK目前只支持单实例。不能够同时开2个推流实例，同时只能存在一个实例，需要另开实例的时候，需要关闭之前存在的实例。
- 2.操作系统版本要求android4.3以上。

2. 系统框架

2.1系统框架图

2.2类框架图



3. 使用说明

3.1 开发环境配置

- 1.需要准备Android运行环境,以及硬件CPU支持ARMv7或ARMv7s的安卓设备。
- 2.权限开通，在阿里云上申请推流SDK开发权限。

3.2 开发步骤

首先在安卓应用程序中，需要声明以下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" /
>
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_SETTINGS" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

使用用媒体推流器SDK的调用顺序为：

- 1.调用AlivcMediaRecorderFactory.createMediaRecorder获取实例
- 2.初始化mMediaRecorder.init(this);
- 3.调用mMediaRecorder.prepare(mConfigure, mPreviewSurface);
- 4.调用mMediaRecorder.startRecord(pushUrl);开始推流
- 5.调用mMediaRecorder.stopRecord();停止推流
- 6.调用mMediaRecorder.reset();释放预览资源,对应的是prepare
- 7.调用mMediaRecorder.release();释放资源,对应的是init

3.3 Demo示例

在SDK中提供了Demo，此Demo是用推流SDK开发了一个完整的推流器，用户可以参考Demo进行推流的开发。

下面给出了部分重要的Demo中调用SDK的代码。

1）创建SurfaceView和SurfaceView的Callbac

```
_CameraSurface = (SurfaceView) findViewById(R.id.camera_surface);
_CameraSurface.getHolder().addCallback(_CameraSurfaceCallback);
private final SurfaceHolder.Callback _CameraSurfaceCallback = new
SurfaceHolder.Callback() {
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        holder.setKeepScreenOn(true);
        mPreviewSurface = holder.getSurface();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int wi
dth, int height) {
        mMediaRecorder.setPreviewSize(width, height);
        mPreviewWidth = width;
        mPreviewHeight = height;
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        mPreviewSurface = null;
        mMediaRecorder.stopRecord();
        mMediaRecorder.reset();
    }
};
mPreviewSurface = holder.getSurface();
```

2）、 创建推流器，准备推流：

- 1.创建实例初始化

```
mMediaRecorder = AlivcMediaRecorderFactory.createMediaRecorder();
mMediaRecorder.init(this);
mMediaRecorder.setOnRecordStatusListener(mRecordStatusListener);
```

```
mMediaRecorder.setOnNetworkStatusListener(mOnNetworkStatusListener);
mMediaRecorder.setOnRecordErrorListener(mOnErrorListener);
```

- 2.开始预览

```
private Map<String, Object> mConfigure = new HashMap<>();
mConfigure.put(AlivcMediaFormat.KEY_CAMERA_FACING, cameraFrontFacing);
mConfigure.put(AlivcMediaFormat.KEY_MAX_ZOOM_LEVEL, 3);
mConfigure.put(AlivcMediaFormat.KEY_OUTPUT_RESOLUTION, resolution);
mMediaRecorder.prepare(mConfigure, mPreviewSurface);
```

- 3.开始推流

```
mMediaRecorder.startRecord(pushUrl);
```

- 4.开启关闭美颜滤镜

```
mMediaRecorder.addFlag(AlivcMediaFormat.FLAG_BEAUTY_ON);//开启美颜
mMediaRecorder.removeFlag(AlivcMediaFormat.FLAG_BEAUTY_ON);//关闭美颜
```

- 5.切换摄像头

```
mMediaRecorder.switchCamera();
```

- 6.对焦

```
mMediaRecorder.focusing(x, y);// demo提供了首次对焦+手动对焦
```

- 7.缩放

```
mMediaRecorder.setZoom(scaleGestureDetector.getScaleFactor());
```

- 8.结束推流

```
mMediaRecorder.stopRecord();
```

- 9.释放资源

```
mMediaRecorder.reset();//释放预览资源
```

```
mMediaRecorder.release();//释放推流资源
```

- 10.添加水印

```
//创建水印信息对象
mWatermark = new AlivcWatermark.Builder()
                    .watermarkUrl(bundle.getString(WATERMARK_PATH)) //水印图片地址
                    .paddingX(bundle.getInt(WATERMARK_DX))
//水印图片在x轴的偏移
                    .paddingY(bundle.getInt(WATERMARK_DY))
//水印图片在y轴的偏移
                    .site(bundle.getInt(WATERMARK_SITE))
//水印图片位置
                    .build();

mConfigure.put(AlivcMediaFormat.KEY_WATERMARK, mWatermark); //配置水印信息
```

3 ） 网络状态事件通知-- OnNetworkStatusListener

OnNetworkStatusListener：

```
/**
 * 网络较差时的回调，此时推流buffer为满的状态，会执行丢包，此时数据流不能正常推送
 */
void onNetworkBusy();

/**
 * 网络空闲状态，此时本地推流buffer不满，数据流可正常发送
 */
void onNetworkFree();

/**
 * @param        status
 */
void onConnectionStatusChange(int status);

/**
 * 网络重连
 * @return  false:停止重连  true:允许重连
 */
boolean onNetworkReconnect();
```

4）错误事件通知中：

```
private OnLiveRecordErrorListener mOnErrorListener = new
OnLiveRecordErrorListener() {
    @Override
    public void onError(int errorCode) {
        switch (errorCode) {
            case AlivcStatusCode.ERROR_SERVER_CLOSED_CONNECTION:
            case AlivcStatusCode.ERORR_OUT_OF_MEMORY:
            case AlivcStatusCode.ERROR_CONNECTION_TIMEOUT:
            case AlivcStatusCode.ERROR_BROKEN_PIPE:
            case AlivcStatusCode.ERROR_ILLEGAL_ARGUMENT:
            case AlivcStatusCode.ERROR_IO:
            case AlivcStatusCode.ERROR_NETWORK_UNREACHABLE:
                Log.i(TAG, "Live stream connection error-->" +
errorCode);
                ToastUtils.showToast(LiveCameraActivity.this,"Live stre
am connection error-->" + errorCode);
                 break;
             default:
        }
    }
};
```

4.接口说明

SDK中提供了两个类AlivcMediaRecorder和AlivcMediaRecorderFactory，其中AlivcMediaRecorder是推流SDK使用接口类，AlivcMediaRecorderFactory用来创建推流器器AlivcMediaRecorder。同时提供了多个事件通知接口，用来监听推流器的各种状态。

类名	功能
AlivcMediaRecorderFactory	创建推流器接口
AlivcMediaRecorder	推流功能接口类
AlivcMediaFormat	推流器可配参数类
OnNetworkStatusListener	推流状态监听接口
OnRecordStatusListener	预览状态监听接口

类名	功能
OnLiveRecordErrorListener	推流错误信息监听接口

4.1 AlivcMediaRecorderFactory

类名：AlivcMediaRecorderFactory
功能：创建推流器接口AlivcMediaRecorder
成员：

成员	功能
createMediaRecorder	创建推流器AlivcMediaRecorder

详细说明：

AlivcMediaRecorderFactory.createMediaRecorder()

createMediaRecorder用来创建推流器，返回AlivcMediaRecorder类。

参数:

null

返回值：返回空为错误，正确则为有效的AlivcMediaRecorder值。

4.2 AlivcMediaRecorder

类名：AlivcMediaRecorder
功能：推流器接口类AlivcMediaRecorder，提供推流控制
成员：

成员	功能
init	初始化推流器
prepare	开始预览
startRecord	开始推流

成员	功能
switchCamera	切换摄像头
stopRecord	结束推流
reset	释放预览资源,对应的是prepare
focusing	对焦
setZoom	缩放
setPreviewSize	设置预览大小
addFlag	添加美颜
removeFlag	移除美颜
release	释放资源对应: init
setOnRecordErrorListener	设置推流错误回调
setOnRecordStatusListener	设置推流的状态回调监听
setOnNetworkStatusListener	设置网络状态的回调监听
getVersionName	获取SDK版本名称

下面详细介绍一下各个成员函数的具体使用：

- 1.初始化推流器

```
/**
 * @param context：Android上下文
 */
void init(Context context);
```

- 2.开始预览

```
/**
 * 开始预览
 * @param  params: 推流过程中不可动态改变的参数.
 * @param  surface: 预览窗口
 */
 void prepare(Map<String, Object> params, Surface surface);
```

- 3.开始推流

```
/**
 * @param outputUrl 推流地址URL
 */
 void startRecord(String outputUrl);
```

> 备注：在prepare完成之后调用startRecord进行推流。

- 4.结束推流

```
void stopRecord();
```

- 5.切换摄像头

```
void switchCamera();
```

- 6.释放资源

```
void reset();
```

> 备注：*释放预览资源,对应的是prepare*

- 1. 手动对焦

```
/**
 * @param xRatio 横向坐标点所占的比例
 * @param yRatio 纵向坐标点所占的比例
 */
void focusing(float xRatio, float yRatio);
```

- 8.缩放

```
/**
 * 这是个新增加的接口，老接口为下面的接口，已经废弃，不建议使用
 *
 * @param scaleFactor参数为缩放比例
 */
void setZoom(float scaleFactor);
```

```
/**
 * 这个接口已经废弃了，建议使用上面的接口
 *
 * @param scaleFactor参数为缩放比例
 */
@Deceperated
void setZoom(float
scaleFactor,CaptureRequest.OnCaptureRequestResultListener listener);
```

- 9.设置预览大小

```
/**
 * @param width 预览宽
 * @param height 预览高
 */
void setPreviewSize(int width, int height);
```

- 10.开启美颜 / 开启手动对焦 / 开启闪光灯 / 开启静音推流

```
/**
 * 增加效果 如：美颜，对焦，闪光灯，静音等
 * @see AlivcMediaFormat#FLAG_BEAUTY_ON
 * @see AlivcMediaFormat#FLAG_AUTO_FOCUS_ON
 * @see AlivcMediaFormat#FLAG_FLASH_MODE_ON
 * @see AlivcMediaFormat#FLAG_MUTE_ON
 * @see AlivcMediaRecorder#removeFlag(int)
 * @param flag
 */
void addFlag(int flag);
```

- 11.关闭美颜 / 关闭手动对焦 / 关闭闪光灯 / 关闭静音推流

```
/**
 * 移除效果 如：美颜，对焦，闪光灯，静音等
 * @see AlivcMediaFormat#FLAG_BEAUTY_ON
```

```
     * @see AlivcMediaFormat#FLAG_AUTO_FOCUS_ON
     * @see AlivcMediaFormat#FLAG_FLASH_MODE_ON
     * @see AlivcMediaFormat#FLAG_MUTE_ON
     * @param flag
     */
    void removeFlag(int flag);
```

- 12.释放资源

```
    void release();
```

- 13.设置推流错误回调

```
    /**
     * @param listener 直播错误回调
     */
    void setOnRecordErrorListener(OnLiveRecordErrorListener listener);
```

> 备注：*具体的错误信息会在下面介绍*

- 14.设置推流的状态回调监听

```
    /**
     * @param listener 直播状态回调
     */
    void setOnRecordStatusListener(OnRecordStatusListener listener);
```

> 备注：*回调具体的信息在下面介绍*

- 15.设置网络状态的回调监听

```
    /**
     * @param listener 直播网络状态回调
     */
    void setOnNetworkStatusListener(OnNetworkStatusListener listener);
```

> 备注：*回调具体的信息在下面介绍*

- 16.添加水印图片

首先需要创建一个水印信息对像——AlivcWatermark

```
AlivcWatermark mWatermark = new AlivcWatermark.Builder()
                        .watermarkUrl(bundle.getString(WATERMARK_PATH)) //水印图片地址
                        .paddingX(bundle.getInt(WATERMARK_DX))
//水印图片在x轴的偏移
                        .paddingY(bundle.getInt(WATERMARK_DY))
//水印图片在y轴的偏移
                        .site(bundle.getInt(WATERMARK_SITE))
//水印图片位置
                        .build();
```

水印位置有四个常量表示：

常量值	含义
AlivcWatermark.SITE_TOP_LEFT	左上角
AlivcWatermark.SITE_TOP_RIGHT	右上角
AlivcWatermark.SITE_BOTTOM_LEFT	左下角
AlivcWatermark.SITE_BOTTOM_RIGHT	右下角

然后需要将该水印信息描述对像通过AlivcMediaRecorder#prepare接口的map参数传入进去

```
mConfigure.put(AlivcMediaFormat.KEY_WATERMARK, mWatermark); //配置水印信息
......
mMediaRecorder.prepare(mConfigure, mPreviewSurface);
```

- 17.获取SDK版本名称

```
String getVersionName()
```

4.3 OnNetworkStatusListener

当调用startRecord()后，推流器开始推流，当前的网络连接状态会返回，用户需要注册该事件,以便获取到该时间通知，在结束和开始的状态都会给予返回.

```
public interface OnNetworkStatusListener {

    /**
     * 网络较差时的回调，此时推流buffer为满的状态，会执行丢包，此时数据流不能正常推送
     */
    void onNetworkBusy();

    /**
     * 网络空闲状态，此时本地推流buffer不满，数据流可正常发送
     */
    void onNetworkFree();

    /**
     * @param   status
     */
    void onConnectionStatusChange(int status);

    /**
     * 重连失败
     * @return false:停止重连 true:继续重连
     * 说明： ｓｄｋ检测到检测到需要重连时将会自动执行重连，直到重连成功或者重连尝试超时，
     * 超时时间可以通过{@link AlivcMediaFormat#KEY_RECONNECT_TIMEOUT}来设置，
     * 默认为５ｓ，超时后将触发此回调，若返回true表示继续开始新的一轮尝试，返回false，
     * 表示不再尝试
     */
    boolean onNetworkReconnectFailed();
}
```

注意：这里 *onNetworkReconnect()*接口已经移除掉，现在网络重连为ｓｄｋ自动执行，在重连指定时间后弱依然不能重连成功，则会执行

*onNetworkReconnectFailed()*的回调，表示重连失败

4.4 OnRecordStatusListener

当prepare完成后，会发出该事件通知消息，用户需要注册该事件，采集开始得到摄像头的状态以便做一些事情。比如进入对焦一次.

```
public interface OnRecordStatusListener {
    /*** 摄像头打开成功    */
    void onDeviceAttach();
    /*** 开启预览成功    */
    void onSessionAttach();
    /**    * 停止预览    */
    void onSessionDetach();
    /*** 关闭摄像头    */
    void onDeviceDetach();
    /**摄像头打开失败**/
    void onDeviceAttachFailed(int facing)
}
```

4.5 OnLiveRecordErrorListener

当开始推流，返回给开发者的错误信息。

```
public interface OnLiveRecordErrorListener {
    /**
     * @param  errorCode 获取到的错误类型Code
     * @see AlivcStatusCode#ERROR_**
     */
    void onError(int errorCode);
}
```

> 备注：具体ErrorCode详情可参考下面的ErrorCode说明

5. ErrorCode说明

ErrorCode常量	含义	可能出现的场景
AlivcStatusCode.ERROR_BROKEN_PIPE	管道中断	推流时进行了违法操作，比如同时推流同一个地址，或者重复推流，服务器端会主动关闭socket，引起broken pipe
AlivcStatusCode.ERROR_OUT_OF_MEMORY	内存不足	手机内存不足时导致底层某些内存开辟失败引起该错误
AlivcStatusCode.ERROR_IO	I/O错误	导致该错误的情况比较多，比如网络环境较差或者推流域名错误等导致DNS解析失败等
AlivcStatusCode.ERROR_iLLIGAL_ARGUMENT	参数非法	该错误通常发生在帧数据错误的情况下
AlivcStatusCode.ERROR_NETWORK_UNREACHABLE	网络不可达	该错误通常发生在网络无法传输数据的情况，或者推流过程网络中断等情况
AlivcStatusCode.ERROR_SERVER_CLOSED_CONNECTION	服务器关闭链接	发生违法操作时，服务器会主动断开链接

ErrorCode常量	含义	可能出现的场景
AlivcStatusCode.ERROR_CONNECTION_TIMEOUT	网络链接超时	网络较差时导致链接超时或者数据发送超时
AlivcStatusCode.ERROR_AUTH_FAILED	鉴权失败	推流地址开启鉴权时，鉴权失败
AlivcStatusCode.ERROR_OPERATION_NOT_PERMITTED	操作不允许	发生违法操作时，服务器端主动断开链接导致
AlivcStatusCode.ERROR_CONNECTION_REFUSED	服务器拒绝链接	域名解析错误，或者其他异常导致无法链接服务器时

6.打点日志

- 1. 获取性能日志

所有的性能日志都通过AlivcRecordReporter来获取,通过AlivcMediaRecorder#getRecordReporter()获取AlivcRecordReporter的对象
注意：AlivcMediaRecorder#getRecordReporter()必须在AlivcMediaRecordReporter#init()方法之后，AlivcMediaRecordReporter#release()方法之前调用，否则将返回一个null对象

/** 获取int类型的性能属性 **/
AlivcRecordReporter#getInt(int key);

/** 获取Double类型的性能属性 **/
AlivcRecordReporter#getDouble(int key);

/** 获取Long类型的性能属性 **/
AlivcRecordReporter#getLong(int key);

/** 获取Float类型的性能属性 **/
AlivcRecordReporter#getFloat(int key);

/** 获取boolean类型的性能属性 **/
AlivcRecordReporter#getBoolean(int key);

/** 获取String类型的性能属性 **/
AlivcRecordReporter#getString(int key);

相关属性的Key值都可通过AlivcRecordReporter拿到，比如视频采集帧率的Key值——AlivcRecordReporter#VIDEO_CAPTURE_FPS,其对应的属性值类型可在相应的类文档中查看。

- 1. 事件通知

打点事件是通过订阅的形式获取监听，通过AlivcMediaRecorder#subscribeEvent()订阅一个事件，通过AlivcMediaRecorder#unSubscribeEvent()取消订阅。

订阅事件：
需要创建一个AlivcEventSubscriber对象，该对象中包含要订阅的事件类型以及对应的事件响应，事件类型可以通过AlivcEvent
EventType类中定义的常量来表示。
事件响应需要实现AlivcEventResponse接口，其中的onEvent表示事件发生，其参数AlivcEvent代表一个事件，AlivcEvent#getBundle()方法获取一个Bundle对象，该对象存储了事件的一些相关信息，具体每个事件对应有哪些信息可以查看相关类文档，获取事件信息的Key值可以通过AlivcEvent
EventBundleKey中定义的常量表示

7. 注意事项

无

8. 版权声明