

AVD SDK

---

Android 版开发指南  
(版本 v3.1)

# 目 录

<b>1</b>	<b>修订记录 .....</b>	<b>4</b>
<b>2</b>	<b>概述 .....</b>	<b>4</b>
2.1	简介 .....	4
2.2	面向的读者 .....	4
2.3	获取 SDK DEMO .....	5
2.4	技术支持 .....	5
<b>3</b>	<b>编写说明 .....</b>	<b>5</b>
<b>4</b>	<b>环境准备 .....</b>	<b>5</b>
4.1.1	开发工具 ANDROID STUDIO .....	5
4.1.2	SDK DEMO 工程导入 .....	5
4.1.3	SDK DEMO 工程运行 .....	6
4.1.4	SDK DEMO 工程结构以及组织方式 .....	6
<b>5</b>	<b>音视频基本功能开发流程 .....</b>	<b>8</b>
5.1	鉴权 .....	8
5.2	运行时权限 .....	8
5.3	日志系统 .....	9
5.3.1	应用层获取日志统计 .....	9
5.4	配置引擎选项 .....	10
5.5	引擎初始化 .....	10
5.6	创建房间 .....	11
5.7	加入房间 .....	11
5.8	音频/视频处理 .....	12
5.9	共享屏幕 .....	16
5.10	参会者管理 .....	17
5.11	语音激励 .....	19
5.12	白板 .....	20
5.13	视频渲染窗口 .....	22
5.14	透明通道、聊天 .....	22
5.15	导入/导出音视频数据 .....	23
5.16	码流控制 .....	24
5.17	媒体统计 .....	25
5.18	回调通知 .....	25

5.19 扩展功能 .....	28
5.20 离开/关闭房间并释放资源 .....	30
<b>6 快速开发流程 .....</b>	<b>30</b>
<b>7 常见错误处理 .....</b>	<b>30</b>
<b>8 附录 .....</b>	<b>33</b>
8.1 名词解释 .....	33
8.2 错误码 .....	33

## 1 修订记录

修订日期	描述	作者	版本号
2019.6.5	初始文档	王龙渊	3.0.0
2021.1.15		王龙渊	3.1.0

## 2 概述

Open-AVD SDK 提供人与人实时沟通协作过程中需要用到的所有基本能力，涵盖了网络会议系统、IM 即时通讯系统及直播系统三大类终端产品音视频通讯的主要功能。

Open-AVD SDK 由业界资深工程师精心打造，稳定可靠，第三方团队拿来就能用，不必自己去造“轮子”，从而降低了第三方团队的技术风险，减少了项目的开发投入，尤其是能大幅缩短第三方团队开发具有多方音视频+数据协作能力的 App/Web 应用的时间。

Open-AVD SDK 可用于几乎所有行业，很多业务场景中需要用到人与人实时沟通与协作的能力，而类似 QQ，微信或会议系统这种通用沟通工具又不能直接使用或不能满足功能，这种情况下，Open-AVD SDK 就是您最好的选择。市场调研表明，Open-AVD SDK 在医疗、教育、金融、能源、交通等各个领域，都有巨大的市场需求。

### 2.1 简介

Open-AVD SDK 为移动、桌面和互联网应用提供一个完善的音视频及数据互动开发框架，屏蔽掉互动系统的复杂细节，对外提供较为简洁的 API 接口，方便第三方应用快速集成互动功能。

Open-AVD SDK Android 版提供如下功能：

- Android 全平台支持
- 实时高清语音通话
- 实时高清视频通话

### 2.2 面向的读者

本指南是提供给具有一定的 Android 编程经验和了解面向对象概念的产品经理及程序员使用，Open-AVD SDK 已很好的封装了音视频相关的底层技术细节，因而读者不需要具备音视频开发方面的经验。

## 2.3 获取 SDK Demo

公司的 github 网址(<https://github.com/3tee>)上会提供各类基于 Open-AVD SDK 的实例 Demo，包括基本音视频能力 Demo。

## 2.4 技术支持

您在使用本 SDK 的过程中，遇到任何困难，请与我们联系，我们将热忱为您提供帮助。你可以通过如下方式与我们取得联系。

- 技术支持工程师： 186XXXXXXXX
- 

## 3 编写说明

本指南编写目的是为了帮助使用 AVD SDK 的用户快速搭建 SDK 的开发环境、熟悉开发流程、掌握 SDK 开发功能接口而编写的。

本指南基于 Android studio IDE 开发，导入 baseVideo Demo 进行最简单的音视频能力进行编写，如果需要更多的功能，请参阅 SDK API 接口开发手册。

## 4 环境准备

### 4.1.1 开发工具 Android studio

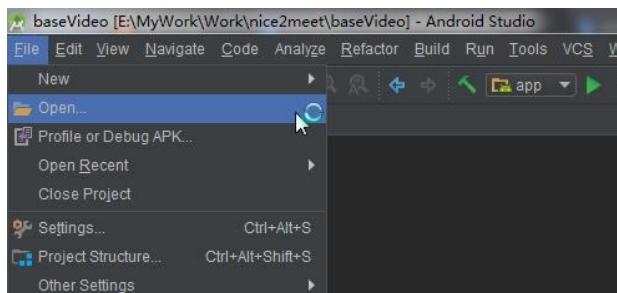
本指南开发工具采用 Android studio v3.1.2，build gradle 版本为 3.1.2，开发者可自行根据本地版本进行配置，抑或直接编译按照 Android studio 提示自动完成相应工具的安装；若使用 3.4.1 版本 Android studio 在界面上会有一定的出入，功能是类似的，开发者可自行查找处理方法。

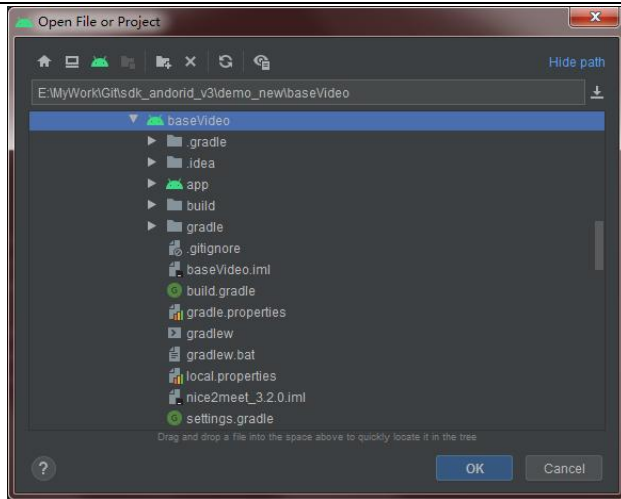
Android studio 官方下载地址：<https://developer.android.google.cn/studio/index.html>

### 4.1.2 SDK Demo 工程导入

2.3 中下载的 SDK Demo，通过下图所示导入工程：

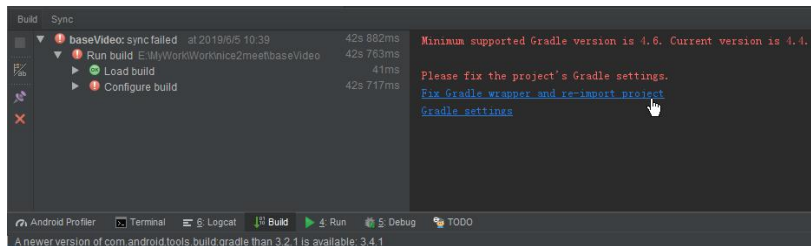
选择菜单 File -> Open 打开 Demo 所在目录，等待工程自动编译完成。



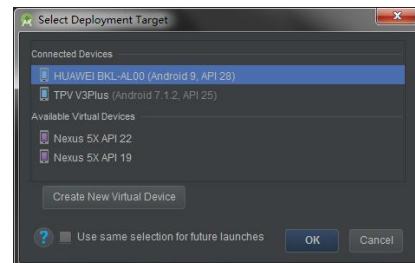
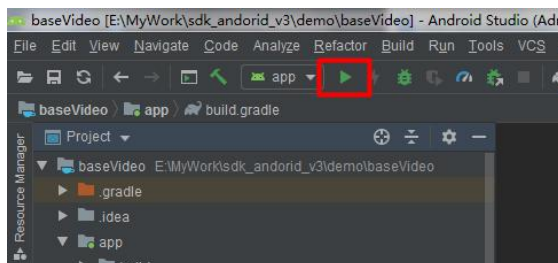


#### 4.1.3 SDK Demo 工程运行

等待 build gradle 自动编译完成即可运行工程，编译环境不一样时会提示安装相关工具（Android SDK、build tools、build gradle 等），按照提示点击链接自动安装即可。

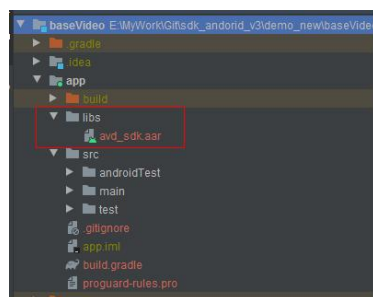


编译完成后 Shift+F10 或点击运行图标在真机上运行，注意：模拟器上运行时部分音视频功能可能会不可用！



#### 4.1.4 SDK Demo 工程结构以及组织方式

A. libs 目录导入 avd\_sdk.aar 包



## B. build.gradle 文件添加 avd\_sdk.aar 包依赖

```

1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 28
5      defaultConfig { ... }
6      buildTypes { ... }
7
8      compileOptions {
9          sourceCompatibility JavaVersion.VERSION_1_8
10         targetCompatibility JavaVersion.VERSION_1_8
11     }
12
13     lintOptions { disable 'GoogleAppIndexingWarning' }
14 }
15
16 repositories {
17     flatDir {
18         dirs 'libs'
19     }
20 }
21
22 dependencies {
23     implementation fileTree(include: ['*.jar'], dir: 'libs')
24     implementation 'com.android.support:appcompat-v7:28.0.0'
25     implementation 'com.android.support.constraint:constraint-layout:1.1.3'
26     testImplementation 'junit:junit:4.12'
27     androidTestImplementation 'com.android.support.test:runner:1.0.2'
28     androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
29
30     api(name: 'avd_sdk', ext: 'aar')
31
32     api 'com.tbruyelle.rxpermissions2:rxpermissions:0.9.5@aar'
33     api 'com.google.code.gson:gson:2.7'
34     api 'com.google.code.findbugs:jsr305:3.0.2'
35     api 'io.reactivex.rxjava2:rxandroid:2.0.1'
36     api 'io.reactivex.rxjava2:rxjava:2.1.16'
37     api 'com.android.support:recyclerview-v7:28.0.0'
38 }

```

## C. build.gradle 添加 AVD SDK 所需第三方库依赖（lambda 语句支持、gson 库等）

Lambda 支持:

```

android {
    .....
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

```

导入 Gson 和 android 注解库:

```

dependencies {
    .....
    api 'com.google.code.gson:gson:2.7'
    api 'com.google.code.findbugs:jsr305:3.0.2'
}

```

```

android {
    compileSdkVersion 28
    defaultConfig { ... }
    buildTypes { ... }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    lintOptions { disable 'GoogleAppIndexingWarning' }
}

```

```

dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    api 'com.google.code.gson:gson:2.7'
    api 'com.google.code.findbugs:jsr305:3.0.2'
}

```

## 5. 音视频基本功能开发流程

本开发流程包括鉴权、运行时权限、日志配置、配置引擎选项、引擎初始化、创建房间、加入房间、音视频处理、共享屏幕、视频渲染、白板（标注）、扩展功能等，更多详细信息可以参阅 SDK API 接口开发手册或咨询我们技术支持工程师。

### 5.1 鉴权

针对正式客户及测试客户，会提供一对有效的 Access Key（默认值 demo\_access）和 Secret Key（默认值 demo\_secret），这对密钥可以产生访问令牌（24 小时有效），用于生成会议房间号，及加房间等操作。**注：初始化引擎鉴权失败会报 401 错误！**

A. 服务器地址，依据当前 web 服务协议(http,https)指定对应的端口：

```
serverURI = "http://dev.3tee.cn:81";
accessKey = "sdk2020_demo_access";
secretKey = "sdk2020_demo_secret";
```

B. 在初始化引擎时需要传入服务器地址和 key，引擎初始化将会在后续详细讲解如下所示：

```
AVDEngine.instance().init(context, listener, serverURI, accessKey, secretKey);
```

### 5.2 运行时权限

app 的 build.gradle 里把 targetSdkVersion 版本设为 23（包括 23）及其以上时，网络访问、SD 卡读写权限、摄像头、录音权限需要动态获取，在此强烈推荐将 targetSdkVersion 版本设置为 23 以上（Android 6.0+），避免 Android 某些功能受限和导入第三方库版本报错。AVD SDK 涉及到的权限如下表所示：

权限名称	说明
android.permission.INTERNET	网络权限
android.permission.CAMERA	摄像头权限（不开启视频通话将不可用）
android.permission.RECORD_AUDIO	录音权限（不开启语音通话将不可用）
android.permission.READ_PHONE_STATE	用于获取设备唯一的标识符 IMEI 用于设备绑定
android.permission.ACCESS_NETWORK_STATE	网络状态用于会议控制
android.permission.WRITE_EXTERNAL_STORAGE	日志写操作（不开启日志功能将不可用）
android.permission.READ_EXTERNAL_STORAGE	日志读操作



### 5.3 日志系统

在初始化引擎前配置日志，用于记录 SDK 运行状态，方便问题排查和运行流程追踪。包括的内容有日志路径配置、日志等级、日志时间标签格式、媒体统计等，日志参数 `params` 是一个拼接的字符串，参数之间用空格符号隔开，参数含义如下：

- **debug**: 调试模式，建议开启。
- **verbose**: 日志等级，支持的日志等级有 **verbose** 详细、**info** 信息、**warning** 报警、**error** 错误，日志等级由低到高，等级越低日志输出越详细，建议测试阶段使用 **verbose** 等级，正式发版时使用 **info** 或更高等级，且需要应用层定期清理日志，避免占用较多空间。
- **stats**: 将媒体统计信息（包含每路视频和音频的分辨率、帧率、码流等信息）输出到日志。
- **append**: 将此次加会的日志拼接到上次离会的日志后面打印，初始化引擎后均会生成一个新的日志（前提是 SDK 引擎已释放）。
- **realtimestamp**: 每条日志的时间戳使用当前时间格式 [月-日 小时: 分钟: 秒]。

代码示例如下：

```
@SuppressWarnings("SimpleDateFormat")
DateFormat format = new SimpleDateFormat("yyyyMMdd_HH:mm:ss");
String logPath = "doubleRecord/" + "doubleRecord"
                + format.format(new Date()) + ".log";
logPath = Environment.getExternalStorageDirectory().getAbsolutePath()
            + File.separator + logPath; // 日志路径
String params = "debug verbose stats append realtimestamp";
AVDEngine.instance().setLogParams(params, logPath);
```

#### 5.3.1 应用层获取日志统计

调用 `Tlog` 类的方法 `startLogTrace(String param, AVDEngine.ILogListener listener)` 用于获取打印的日志信息，`param` 配置过滤日志（等级、时间戳格式、调试信息等），`listener` 日志回调监听。

开始采集日志：

```
Tlog.startLogTrace("debug verbose stats append realtimestamp", new AVDEngine.ILogListener() {
    @Override
    public void onCallbackLog(String logmsg) {

    }
});
```

停止采集日志：

```
Tlog.stopLogTrace();
```

## 5.4 配置引擎选项

用于控制会议中音视频编解码格式、音频采集回音消除、优先使用的分辨率帧率格式、数据通道格式（UDP/TCP）等，使用 `AVDEngine.instance().setOption(AVDEngine.Option.[选项],[传入的值])` 配置引擎选项，示例如下：

```
AVDEngine.instance().setOption(AVDEngine.Option.[选项],[传入的值])

[选项] 说明:
eo_data_channel_tcp_priority      // 数据通道用的网络连接类型设置, true 优先使用 tcp,
                                   false 优先使用 udp (默认)
eo_video_swapwh_by_rotation:      // 不变换宽高严格按设置分辨率裁剪拉伸, 字符串"false"
eo_camera_mode_frontback:        // 使用前后置双摄像头模式, 字符串"true"
eo_video_resolution_16balign:     // 分辨率 16 位自动对齐, 字符串"true"
eo_video_codec_hw_priority:       // 优先使用硬件编解码, 字符串"true"
eo_audio_aec_Enable:              // 开启回音消除, 字符串"true"
eo_audio_aec_DAEcho_Enable:       // 启用延时消除算法, 字符串"true"
eo_audio_autoGainControl_Enable:  // 麦克风输入自动增益, 字符串"true"
eo_video_renderusecapture:        // 渲染视频时直接渲染方式不做拉伸变形, 字符串"true"
eo_video_codec_priority:          // 优先使用 H264 编码, 字符串"true"
eo_video_codec_hw_encode:         // 优先使用硬编码, 字符串"true"
eo_video_codec_hw_decode:         // 优先使用硬解码, 字符串"true"
eo_data_channel_tcp_priority:     // 房间数据通道优先使用 TCP("true")或 UDP ("false" 缺省值)
eo_camera_capability_default:     // 设置发布视频的分辨率和帧率 1280*720 30 帧, 传入 json 字符串,
                                   // 宽、高、最大帧率: "{\width\":1280,\height\":720,\maxFPS\":30}"
```

## 5.5 引擎初始化 (AVDEngine)

引擎初始化是一个异步操作，引擎初始化是否成功需要判断返回值，并在回调里对状态值做判断，为 0（`ErrorCode.AVD_OK`）代表初始化成功，回调为 0 后才能做后续の入会操作，初始化方法 `AVDEngine.instance().init(Context context, AVDEngine.Listener listener, String severuri, String appkey, String secretkey)`。

参数含义如下：

- context: 建议使用 application 的上下文
- listener: 初始化引擎回调
- severuri: 服务器地址
- appkey: 鉴权 accessKey
- secretkey: 鉴权 secretKey

初始化失败错误码如下：

- 1014: 网络错误，需要排查传入的服务器地址是否正确，设备网络是否正常连接，网络质量是否良好，服务器端部署是否正常。
- 401: 传入的密钥 Access Key 和 Secret Key 错误。
- 405: 服务器授权过期，请联系我方续期。

- 436: AVD SDK 版本和服务端版本不匹配，请联系我方对服务器做适配调整。
- 1008: 初始化引擎传入的上下文 context 无效（为空或未传）。

代码示例如下：

```
int result = AVDEngine.instance().init(context, listener, serverURI, accessKey, secretKey);
if (result != ErrorCode.AVD_OK) {
    // TODO: [初始化引擎失败]
}

AVDEngine.Listener listener = new AVDEngine.Listener() {
    .....

    if (ErrorCode.AVD_OK != result) {
        // TODO: [初始化引擎失败]
    } else {
        // TODO: [初始化引擎成功] 后才能做加会等操作，否则会报错！
    }
};
```

## 5.6 创建房间（Room）

在 5.5 中初始化引擎回调成功后才能执行创建房间操作，创建房间是异步回调过程，需要做返回值判断，返回值为 0 创建成功，返回值为 40005 或 40003 房间已存在。

```
if (AVDEngine.instance().isWorking()) {
    // roomId: 房间号; true: 开启语音激励; hostId: 房间所有者
    RoomInfo roomInfo = new RoomInfo(roomId, true, hostId);
    AVDEngine.instance().scheduleRoom(roomInfo);
}

AVDEngine.Listener listener = new AVDEngine.Listener() {
    .....

    @Override
    public void onScheduleRoomResult(int result, String roomId) {
        if (result == 0 || result == 40005 || result == 40003) {
            // TODO: 创建房间成功或已存在
        } else {
            // TODO: 创建房间失败
        }
    }
};
```

## 5.7 加入房间（Room）

在 5.5 中初始化引擎回调成功后且房间存在才能执行加入房间操作，如果未初始化引擎加入房间则获取房间实例返回为空，如果房间不存在加入该房间会返回 404 房间不存在错误。加入房间前可以配置房间参数，加入房间是异步回调，需要在回调里判断是否成功（0 代表加入成功），必须要设置 DTLS 音视频加密选项才能加房间，`room.enableStats(true)`使能媒体

统计后 5.3 节配置的媒体统计日志 “stats” 才会生效，加入房间流程如下：

- A. 判断引擎是否初始化
- B. 获取房间并设置房间参数（非必选项，可按照需求设置）
- C. 创建用户并加入房间

```
if (!AVDEngine.instance().isWorking()) {
    return;
}

Room room = Room.obtain(roomId);    // 获取房间
if (room == null) {
    return;
}

room.setOption(Room.Option.ro_media_use_dtls, "true");    // 使用 DTLS 音视频加密算法（必须开启）
room.setOption(Room.Option.ro_room_rejoin_times, "-1");    // 设置网络波动断开连接后一直重连
room.setOption(Room.Option.ro_audio_option_codec, "opus");    // 音频编码格式 opus
room.enableStats(true);    // 使能房间内媒体统计（监测每路视频分辨率、帧率、码流，音频码流等信息）
User user = new User(userName, userName, 0, "", ""); // 创建用户（用户名：userName， 用户 id: userName）

// 加入房间
room.join(user, "", new Room.JoinResultListener() {
    @Override
    public void onJoinResult(int result) {
        if (result == ErrorCode.AVD_OK) {
            // TODO: 加入房间成功
        } else {
            // TODO: 加入房间失败
        }
    }
});
```

## 5.8 音频/视频处理（MVideo、MAudio）

本小节涉及音频和视频（麦克风和摄像头）开关操作，音视频的订阅，音视频的渲染，状态查询等功能。使用前需要先调用 `public static MVideo getVideo(Room room)` 获取 `MVideo` 类对象，这里需要传入 `Room` 类对象，`Room` 对象通过 `Room.obtain(roomId)` 获取。打开/关闭摄像头（或麦克风）后，其他用户会收到“发布/停止发布”回调通知，在回调通知里做“订阅/取消订阅”操作，而后会触发“订阅/取消订阅”回调通知，判断“订阅”成功后才能进一步“渲染视频”。

### A. 音视频的开关操作（发布/停止发布音视频）

开的状态对应着开启摄像头（或麦克风）采集并发布视频（或音频）这样一个连贯的动作，同样的关的状态对应着停止摄像头（或麦克风）采集并停止发布视频（或音频）动作，开关操作均是异步方式回调。其中预览本地视频功能只会开启摄像头采集并预览，而不会将视频发布出去。开启摄像头会触发 `MVideo.Listener` 回调 `onPublishLocalResult`，关闭摄像头会触发 `MVideo.Listener` 回调

onUnpublishLocalResult; 音频的开启/停止会相应触发 MAudio.Listener 回调。

开启摄像头:

```
public int publishLocalCamera(MVideo.CameraType type)
```

- MVideo.CameraType 摄像头类型, front 前置摄像头, back 后置摄像头。
- 返回值为 0 表示发布过程没有报错, 具体是否发布成功需要到回调里判断返回值。

关闭摄像头:

```
public int unpublishLocalCamera()
```

- 返回值为 0 表示停止发布过程没有报错, 具体是否发布成功需要到回调里判断返回值。

开启麦克风:

```
public int openMicrophone()
```

- 返回值为 0 没有报错, 具体是否发布成功需要到回调里判断返回值。

关闭麦克风:

```
public int closeMicrophone()
```

- 返回值为 0 没有报错, 具体是否发布成功需要到回调里判断返回值。

设置本地摄像头使用 Android 的 Camera 接口类型 (Camera1 或者 Camera2) :

```
public static void enableCamera2API(boolean enable)
```

- enable 为 true 代表使用 Camera2, false 使用 Camera1 接口, 默认使用 Camera2 (推荐)

查询本地摄像头是否已开启:

```
public boolean ispublishedLocalCamera()
```

- 返回值为 0 没有报错, 具体是否发布成功需要到回调里判断返回值。

查询指定类型摄像头是否已发布:

```
public boolean isCameraPublished(MVideo.CameraType type)
```

- MVideo.CameraType 摄像头类型, front 前置摄像头, back 后置摄像头。

查询指定设备 id 的摄像头是否已发布:

```
public boolean isCameraPublished(String deviceId)
```

- deviceId 设备 id, 发布的音视频均由设备 id 标识。

设置发布视频的码流大小:

```
public int setVideoBitrate(String deviceId, int minBitrateBps, int maxBitrateBps)
```

- deviceId 设备 id, 发布的音视频均由设备 id 标识。
- minBitrateBps 最小码流, 单位 (bps)
- maxBitrateBps 最大码流, 单位 (bps)

## B. 订阅/取消订阅音视频

只能订阅已发布且非自己的视频 (摄像头或共享屏幕视频), 可以在“发布/停止发布”视频回调中动态去订阅, 或者订阅已发布视频列表 (public List<MVideo.Camera> getPublishedCameras()接口获取) 中的设备。音频的订阅和取消订阅默认由 SDK 底层自动完成, 其他参会者开启音频时自动订阅, 其他参会者关闭音频时取消订阅, 如果需要手动操作需要关闭自动订阅音频 room.setOption(Room.Option.ro\_audio\_auto\_s

unsubscribe, "false")。

订阅视频会触发 MVideo.Listener 回调 onSubscribeResult，取消订阅视频会触发 MVideo.Listener 回调 onUnsubscribeResult；音频的订阅/取消订阅相应触发 MAudio.Listener 回调。

订阅视频：

```
public int subscribe(String deviceId)
```

- deviceId 要订阅的设备 id 号
- 返回值为 0 没有报错，具体是否发布成功需要到回调里判断返回值。

订阅视频并传入视频的期望分辨率等级，分辨率等级分为低中高 3 个等级，高等级代表的分辨率是发布的分辨率，中等级代表高分辨率宽高的一半，同理低等级是中等级的一半；需要注意的是分辨率宽高不是严格按照一半来计算，涉及到 16 位对齐以及弱网情况下服务器自动调低分辨率的情况；如果有多人使用该接口订阅视频，则最终拿到的视频分辨率按照所有订阅用户中等级最高的视频：

```
public int subscribeWithVideoQuality(String deviceId, VideoOptions.VideoQuality quality)
```

- deviceId 要订阅的设备 id 号
- quality 订阅该视频的分辨率等级：quality\_low 低、quality\_normal 中、quality\_high 高。

取消订阅视频：

```
public int unsubscribe(String deviceId)
```

- deviceId 要取消订阅的设备 id 号
- 返回值为 0 没有报错，具体是否发布成功需要到回调里判断返回值。

订阅音频：

```
public int subscribe(String userId)
```

- userId 要订阅的用户 id 号
- 返回值为 0 没有报错，具体是否发布成功需要到回调里判断返回值。

取消订阅音频：

```
public int unsubscribe(String userId)
```

- deviceId 要取消订阅的用户 id 号
- 返回值为 0 没有报错，具体是否发布成功需要到回调里判断返回值。

### C. 渲染视频

订阅视频异步回调成功后才能做视频渲染（视频显示到窗口）操作，同一路视频可以渲染到多个窗口显示，渲染的视频窗口可以基于 SDK tee3\web rtc\TextureViewRenderer.class 布局做自定义布局开发，TextureViewRenderer 布局需要绑定 \cn\tee3\avd\VideoRenderer.class 类对象，渲染的时候使用 VideoRenderer 作为载体来加载视频，且能通过 VideoRenderer 设置第一帧视频到达回调通知，自定义视频窗口可以参考 demo 里 T3VideoView 布局代码具体实现。

创建 VideoRenderer 并绑定 TextureViewRenderer 布局：

```
VideoRenderer mRenderer = new VideoRenderer(textureViewRenderer);
```

- textureViewRenderer 是 TextureViewRenderer 布局对象

判断是否窗口已渲染过视频，通过判空 videoid 判断是否正在显示视频：

```
String videoid = mRenderer.getVideoid()
```

渲染视频：

```
public int attachRender(String deviceId, VideoRenderer render)
```

- deviceId 要显示的视频 id 号（设备 id 号）
- render 视频窗口

取消渲染视频：

```
public int detachRender(VideoRenderer render)
```

- render 视频窗口

D. 预览本地摄像头视频不发布视频

```
public int previewLocalCamera(CameraType type, VideoRenderer render)
```

- type 摄像头类型，设置前置或后置。
- render 视频窗口

E. 已发布视频列表查询，已订阅视频列表查询

已发布视频列表：

```
public List<Camera> getPublishedCameras()
```

- 返回已发布的摄像头列表

已订阅视频列表：

```
public List<Camera> getSubscribedCameras()
```

- 返回已订阅视频列表

F. 将发布或预览的视频停顿在当前发布的这一帧画面：

停顿：

```
MVideo mVideo= MVideo.getVideo(room);
String deviceId = mVideo.getCurrentCameraId()
mVideo.muteLocalCamera(deviceId)
```

取消停顿：

```
MVideo mVideo= MVideo.getVideo(room);
String deviceId = mVideo.getCurrentCameraId()
mVideo.muteLocalCamera(deviceId)
```

G. 音频外放和听筒之间切换：

可以选择设备扬声器或者听筒作为会议音频播放设备，`mAudio.setHandFree(true)`使用扬声器外放播放声音，同理设置为 `false` 使用听筒播放声音。

H. 麦克风静音功能：

将麦克风静音，但音频流还是会发布到房间里。

静音：`mAudio.muteMicrophone()`

取消静音：`mAudio.unmuteMicrophone()`

I. 扬声器静音功能：

将扬声器静音，但音频流还是会发布到房间里。

静音：`mAudio.muteSpeaker()`

取消静音：`mAudio.unmuteSpeaker()`

J. 设置麦克风采集数据源：

部分机顶盒、TV、华为 Pad 等 Android 定制设备不支持通话音频 `AudioSource.VOICE_COMMUNICATION`（SDK 默认为该音频源）功能，会出现会议里听不到声音问题，此时需要先于打开麦克风之前调整麦克风数据源为非通话音频（`AudioSource.MIC`），示例代码如下：

```
WebRtcAudioRecord.setDefaultAudioSource(MediaRecorder.AudioSource.MIC);
```

## 5.9 共享屏幕（MScreen）

共享屏幕使用 `MScreen` 类进行操作，包括有共享屏幕的发布、订阅/取消订阅、渲染等内容。

### A. 发布共享屏幕视频

发布共享屏幕需要初始化 Android 共享屏幕接口，重写 `Activity` 方法 `onActivityResult`，发送 `intent` 启动共享屏幕服务，示例代码如下所示：

```
@TargetApi(21)
private void startScreenCapture() {
    ScreenCapturerAndroid.configureContext(this); // 配置当前 activity 上下文，用于共享屏幕自适应屏幕方向
    MediaProjectionManager mediaProjectionManager = (MediaProjectionManager)
        getSystemService(MEDIA_PROJECTION_SERVICE);

    // 启动共享屏幕 Intent
    startActivityForResult(mediaProjectionManager.createScreenCaptureIntent(), 1);
}

@TargetApi(21)
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    mVideo.sendScreenIntent(data); // 设置共享屏幕 Intent
    // 发布 720P 20 帧共享屏幕视频
    mScreen.publishedScreens(MScreen.ScreenResolution.SCREEN_720P, 20, true);
}
```

### B. 订阅/取消订阅

房间里发布共享屏幕会触发 `MScreen.Listener` 类回调方法 `onPublishScreenNotify`，在此方法中判断非自己的共享屏幕时可做订阅操作。同 `MVideo` 类一样，订阅后会触发 `onSubscribeResult` 回调，取消订阅会触发 `onUnSubscribeResult` 回调。

订阅共享屏幕视频：

```
public int subscribe(String deviceId)

- deviceId 要订阅的设备 id 号
- 返回值为 0 没有报错，具体是否发布成功需要到回调里判断返回值。
```

取消订阅共享屏幕视频：

```
public int unsubscribe(String deviceId)

- deviceId 要订阅的设备 id 号
- 返回值为 0 没有报错，具体是否发布成功需要到回调里判断返回值。
```

### C. 渲染



同摄像头一样共享屏幕使用 `MVideo` 类进行渲染操作，具体可查阅 5.8 小节部分文档。

### 5.10 参会者管理（`MUserManager`、`Room`、`User`）

用户管理类 `MUserManager` 用于监听房间中用户加会操作、离会离会、用户调用 `updateUser` 更新用户名称/用户标签数据、用户摄像头/麦克风开关状态变更。`MUserManager` 类对外提供 `updateUser` 更新用户数据接口，获取用户对象 `User` 实例，获取房间参会者列表，关联用户 `User` 和设备等作用。

房间管理类 `Room` 提供将会议中参会者踢出房间功能，以下对相应部分分别加以说明。

用户对象由 `User` 类实例加以描述，包含的成员变量有：用户 ID 号 `userId`、用户名 `userName`、用户数据（`String` 类型自定义数据，用于描述标记等等）`userData`、用户是有数据 `userAgent`、用户状态（用于记录该用户摄像头麦克风设备以及设备状态）`status`。

#### A. 用户加会、离会回调通知：

当用户调用 `Room` 类 `join(User user, String password, Room.JoinResultListener joinresult)` 加入房间和 `destoryRoom(Room room)` 离开房间成功时，其他参会者会分别触发 `void onUserJoinNotify(User user)` 用户加入房间和 `void onUserLeaveNotify(User user)`、`public void onUserLeaveNotify(int reason, User user)` 用户离开房间回调通知。

其中用户离会 `reason` 状态 0 代表正常离会，非 0 状态代表异常离会，状态码如下表所示：

状态码 code	离会原因
0	正常离会，用户自己调用 <code>destoryRoom</code> 离会
804	被相同 <code>userId</code> 用户入会挤下线
807	UDP 媒体通道建立超时，常见于 <code>join</code> 加会时。
808	该用户被其他用户踢出房间
809	服务没有授权用户入会
810	用户关闭房间时，所有用户被踢出房间。
811	服务器维护是关闭房间
812	和服务器之间心跳超时，异常离会。例如：网络中断、APP 被系统或后台杀掉。

回调接口：

```
/** 用户加入房间通知
 * @param[in] user 用户信息数据。
 * @note 某用户调用房间中的 join 加入房间后，房间内所有用户会接收到此通知
 * @sa join
 */
public void onUserJoinNotify(User user);

/** 用户离开房间通知
 * @param[in] user 用户信息数据。
 * @note 某用户调用房间中的 leave 离开房间后，房间内所有用户会接收到此通知
 * @sa leave
 */
```

```
public void onUserLeaveNotify(User user);

/**
 * 用户异常离开房间通知
 * @param reason 用户离会原因，0 代表正常离会，非 0 异常离会。
 * @param user 离会用户 user 实例
 */
public void onUserLeaveNotify(int reason, User user);
```

#### B. 更新用户数据：

调用接口更新用户数据时，其他参会者会收到用户数据更新通知。

接口定义：public int updateUser(User user)，返回 0 代表成功。

```
/** 用户加入房间通知
 * @param[in] user 用户信息数据。
 * @note 某用户调用房间中的 join 加入房间后，房间内所有用户会接收到此通知
 * @sa join
 */
public void onUserJoinNotify(User user);

/** 用户离开房间通知
 * @param[in] user 用户信息数据。
 * @note 某用户调用房间中的 leave 离开房间后，房间内所有用户会接收到此通知
 * @sa leave
 */
public void onUserLeaveNotify(User user);
```

#### C. 摄像头/麦克风状态变更：

当用户摄像头或麦克风状态改变时（打开、关闭）会收到状态变更回调通知。

```
/** 用户状态更改通知
 *
 * @param[in] status 当前用户状态。
 * @param[in] fromId 关联的用户 Id。
 *
 * @note 某用户调用 updateUserStatus 更改自己状态后，房间内所有用户会接收到此通知；
 * 此通知从 onUserUpdateNotify 中分离出来，因用户状态更改较为频繁
 * @sa updateUserStatus
 * @sa onUserUpdateNotify
 */
public void onUserStatusNotify(int status, String fromId);
```

#### D. 参会者列表管理：

注：获取参会者列表和参会者人数均不包含自己

获取参会者列表： public List<User> getParticipants(int beginindex, int ncount)

- beginindex: 获取的起始位置

- ncount: 获取的长度

获取参会者人数: `public int getParticipantsCount()`

示例:

```
MUserManager mUserManager = MUserManager.getUserManager(room); // 获取 MUserManager 实例
mUserManager.getParticipants(0, mUserManager.getParticipantsCount()); // 获取房间参会者列表, 不包含自己。
```

E. 用户实例 `User` 获取和 `User` 相关操作接口介绍:

- 获取用户 `Id` 对应的该用户 `User` 实例: `public User getUser(String userId)`
- 获取自己的用户 `User` 实例: `public User getSelfUser()`
- 获取自己的用户 `id`: `public String getSelfUserId()`

F. `Room` 将用户踢出房间:

将 `userId` 用户踢出房间, 如果正常返回 `0` 时, 其他用户会收到该用户离会通知, 离会原因 `reason` 为 `808`。

踢出用户接口: `public int kickoutUser(int reason, String userId)`

- `reason`: 踢出原因
- `userId`: 用户 `id` 号
- 返回 `int`: `0` 代表踢出成功, 非 `0` 踢出失败。

## 5.11 语音激励 (MAudio)

语音激励用于获取房间里所有参会者语音能量值, 排列出最大语音能量值用户, 常用于将语音能量值最大的用户标记为发言者, 将该用户视频投影到大屏上显示。语音激励功能需要房间 `Room` 开启语音激励功能, 创建房间时房间信息 `RoomInfo` 变量 `enableVoiceActivated` 置位或在后台 `box` 配置当前房间开启语音激励功能。语音能量开启后会每隔 `1` 秒时间将房间用户语音能量列表回调 (`public void onAudioLevelMonitorNotify(AudioInfo info)`) 给用户, 语音能量值 `0~9`, 数值越大音量越高。

**注:** 语音激励需要用户自己做缓存消除抖动处理, 避免计算出来的最大语音能量用户 (当前发言者) 频繁抖动切换, 在大屏上体现出来的闪屏问题 (切换用户视频导致)。

A. 接口:

查询是否已打开语音激励功能 (本地非服务器): `public boolean ismonitorAudioLevel()`

- 返回 `boolean`: `true` 语音激励已开启, `false` 已关闭。

开启语音激励功能: `public int monitorAudioLevel()`

- 返回 `int`: `0` 代表成功, 非 `0` 代表失败。

关闭语音激励功能: `public int unmonitorAudioLevel()`

- 返回 `int`: `0` 代表成功, 非 `0` 代表失败。

B. 回调:

```
/** 语音激励通知
 * @param[in] info 语音激励信息。
 * @note 语音激励通知, 只有启用语音激励后才会有语音激励通知。启用语音激励接口为: monitorAudioLevel。
 * @sa monitorAudioLevel
 */
public void onAudioLevelMonitorNotify(AudioInfo info);
```

C. 示例代码:

```
AVDManager.getInstance().setMAudioListener(new MAudio.Listener() {
    @Override
    public void onAudioLevelMonitorNotify(MAudio.AudioInfo audioInfo) {
        int myLevel = info.getInputLevel(); // 自己的语音能量值
        final Map<String, Integer> inputLevel = info.getActiveStreams(); // 房间里其他用户能量值
        inputLevel.put(mUserManager.getSelfUserId(), MyLevel); // 将自己的语音能量值一并存入 Map 中
    }
    .....
});
MeetingManager.getInstance().getMAudio().monitorAudioLevel();
```

## 5.12 白板（标注 MWhiteboard、WhiteboardView、WhiteboardDrawView、WhiteboardToolView）

白板布局由“白板底层相对布局容器”和“白板画布”组成，画线时是画在画布上进行展示。白板支持多个用户同时画线，且同一时间只能显示一个白板，如果 A 先于 B 用户共享白板，此后 B 用户共享白板时 A 用户共享的白板会替换为 B 用户白板的内容，新用户白板会替换旧用户白板。白板支持缩放（隐藏工具栏后两个手指做缩放操作）操作、横竖屏显示（跟随界面横竖屏动态计算宽高值）；白板支持两种显示格式，第一种方式显示的白板宽高比维持原发布白板的宽高比，第二种方式显示的白板铺满整个 WhiteboardView 布局大小；同时白板还支持背景色透明显示，即“白板底层相对布局容器”和“白板画布”均可设置为透明色，只有白板上画的图形不透明，这种方式方便直接在视频上或视频某一帧图片上作画，适应会议多种场景需求。MWhiteboard 类处理会议相关白板逻辑，WhiteboardView 是白板布局添加有子布局 WhiteboardDrawView 和 WhiteboardToolView，WhiteboardDrawView 处理白板具体画图逻辑，WhiteboardToolView 处理工具栏业务逻辑。

A. 白板布局及其初始化:

在 xml 文件中定义该白板布局，并初始化白板。

```
<cn.tee3.avd.WhiteboardView
    android:id="@+id/whiteboardView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

```
WhiteboardView whiteboardView = findViewById(R.id.whiteboardView);
whiteboardView.setWhiteboardColor(Color.WHITE); // 设置白板画布颜色
whiteboardView.setBackgroundColor(Color.BLACK); // 设置白板布局背景色

MWhiteboard mWhiteboard = MWhiteboard.getWhiteboard(room); // 获取白板实例
mWhiteboard.showToolbar(true) // 设置显示白板工具栏
    .onAttachView(whiteboardView) // 绑定白板布局
    .setListener(whiteboardListener); // 设置监听
```

## B. 发布和停止发布白板：

查询本地白板是否已发布：`public boolean hasLocalWhiteboardShared()`

- 返回 **boolean**：true 代表已发布，false 代表未发布

发布本地白板：`public String createLocalWhiteboard(final String title, final String desc, final int color)`

- **title**：指定要发布白板的标题

- **desc**：设置要发布白板的描述

- **color**：设置发布白板的画布背景色，其他端会按照该颜色显示白板画布背景色。

- 返回 **String**：返回创建的白板 id 号

停止本地已发布的白板：`public void closeLocalBoard()`

## C. 白板工具栏：

用于选择白板操作的工具类型，包括画实线、画半透明线、设置画线颜色、设置画线的宽度、画单箭头、橡皮擦、激光笔。点击左边圆形图标控制工具栏的展开和收缩，工具栏展开时才能画线；工具栏收缩情况下不能画线，可以做白板画布的缩放操作。注：当工具栏不符合业务需求时，需要用户自己对 `WhiteboardToolView` 自定义布局做修改。

D. 白板的两种显示模式（`WhiteboardView` 类管理）：

`public WhiteboardView setDrawType(DrawType type)`

- **type**：显示模式，`KEEP_WIDTH_HEIGHT_RATIO` 原始宽高比显示，`FOLLOW_SETTING_SIZE` 铺满显示。

## E. 白板的透明色配置：

`whiteboardView.setWhiteboardColor(Color.TRANSPARENT);` // 设置白板画布颜色为透明色

`whiteboardView.setBackgroundColor(Color.TRANSPARENT);` // 设置白板布局的背景为透明色

## F. 设置白板背景图片：

动态设置方式：

可以将图片作为白板背景色上传到服务器（`MWhiteboard` 类的方法 `public int setBackground(String id, String filePath)`），其他端均会收到 `void onUpdateBoardNotify(MWhiteboard.Whiteboard board)` 回调通知，白板背景图片 `http url` 地址可以通过 `board._remoteBgUrl` 来获取，此时就可以把图片显示到 `WhiteboardView` 里的 `ImageView` 布局上了，或者将 `WhiteboardView` 设置为透明色，用户自己定义 `ImageView` 置于 `WhiteboardView` 布局底层显示该网页图片。

发布白板时指定白板背景图片方式：

在发布白板时指定本地 SD 卡上图片的路径地址，SDK 会自动将该图片上传服务器，其他端在收到发布白板回调通知（`public void onShareBoardNotify(MWhiteboard.Whiteboard board)`）后通过 `board._remoteBgUrl` 来获取白板背景图片 `url`，创建白板接口如下：

`public String createLocalWhiteboard(String title, String desc, String remoteBgUrl, int color)`

- **remoteBgUrl**：存于设备 SD 卡上图片存储绝对路径地址

显示背景图片到 `Whiteboard` 布局：

```
if (whiteboardView.getBgmView() == null) {
    whiteboardView.setOnViewCreatedListener(new WhiteboardView.OnViewCreatedListener() {
        @Override
```

```

public void onCreate() {
    whiteboardView.getBgmView().setImageBitmap(bitmap); // 设置背景图片
}
});
} else {
    whiteboardView.getBgmView().setImageBitmap(bitmap); // 设置背景图片
}
}

```

### 5.13 视频渲染窗口（VideoRenderer、TextureViewRenderer）

视频渲染、取消渲染均需要使用 VideoRenderer 作为载体，实际呈现视频效果是由 TextureViewRenderer 布局来完成，这里需要 VideoRenderer 内部绑定 TextureViewRenderer（传入 TextureViewRenderer 布局）。开发过程中用户可以对 TextureViewRenderer 做进一步封装，自定义布局。除了用于呈现视频功能，还提供第一帧视频到达回调通知和截取视频一帧画面的功能，以下将对这些功能一一说明。

A. VideoRenderer 的创建和绑定 TextureViewRenderer 窗口：

获取布局：TextureViewRenderer textureView = view.findViewById(R.id.textureView);

创建渲染对象并绑定窗口：VideoRenderer mRenderer = new VideoRenderer(textureView);

注：以上两步需要在 AVDEngine 初始化之后去执行（查询 AVDEngine.instance().isWorking()），通常情况下窗口 TextureViewRenderer 是在 xml 布局里定义的，TextureViewRenderer 的构造方法在某些情况下会先于 AVDEngine 初始化运行。

B. 第一帧视频到达通知：mRenderer.setFirstFrameCallback(callback)

C. 截取视频一帧画面：

调用 VideoRenderer 类接口，从视频窗口中提取正在显示的视频一帧画面（Bitmap 类对象），每调用一次提取接口均会回调一帧画面，需要确保窗口当前渲染过视频且第一帧已到达，否则提取会失败：

```
public boolean frameSampleCapture(EglRenderer.FrameListener listener)
```

### 5.14 透明通道、聊天（Room、MChat）

可以向会议中其他参会者发送透明通道数据（byte 序列化数据）或聊天信息（String 类型数据），消息类型分为公共广播消息（房间内所有人都能收到）和私有消息（只有指定用户能收到）。

透明通道（Room）：

```

byte[] data = new byte[1024];
room.setListener(new Room.Listener() {
    .....

    @Override /* 公共透明通道数据回调: bytes 消息内容, i 消息数组长度, s 发送消息的用户 id */
    public void onPublicData(byte[] bytes, int i, String s) { ..... }

    @Override /* 私有透明通道数据回调: bytes 消息内容, i 消息数组长度, s 发送消息的用户 id */
    public void onPrivateData(byte[] bytes, int i, String s) { ..... }
});
room.sendPublicData(data, data.length); // 发送公共透明通道数据
room.sendPrivateData(data, data.length, userId); // 发送私有透明通道数据

```

聊天消息（MChat）：

```
MChat mChat = MChat.getChat(room); // 获取 MChat 实例
mChat.setListener(new MChat.Listener() {
    @Override /* 公共消息 */
    public void onPublicMessage(MChat.Message message) {
        String userId = message.getFromId(); // 发送消息的用户 id
        String userName = message.getFromName(); // 发送消息的用户名称
        String messageBody = message.getMessage(); // 消息体
    }
    @Override /* 私密消息 */
    public void onPrivateMessage(MChat.Message message) {
        String userId = message.getFromId(); // 发送消息的用户 id
        String userName = message.getFromName(); // 发送消息的用户名称
        String messageBody = message.getMessage(); // 消息体
    }
});
mChat.sendPrivateMessage(mssage, userId); // 发送私有消息 message 给用户
mChat.sendPublicMessage(mssage); // 发送公共消息给房间里所有人
```

## 5.15 导入/导出音视频数据

只有有导入 H264 视频编码格式功能（作为单独的一路视频发布出去，等同于摄像头视频），导出包括 Camera1 接口采集的原始数据 NV21 格式数据和麦克风采集的原始音频 PCM 格式数据，以下对相关功能加以说明。

A. 导入 H264 视频（FakeVideoCapturer 类）：

导入步骤包括创建虚拟摄像头并配置摄像头参数、创建 FakeVideoCapturer 对象并设置回调、发布该虚拟摄像头、传入 H264 每一帧数据。

创建 FakeVideoCapturer 对象接口：

```
public static FakeVideoCapturer Create(FourccType fourFormat, boolean isScreen,
    Listener listener)
```

- fourFormat: 导入的视频格式，这里选择 H264（FakeVideoCapturer.FourccType.ft\_H264）。
- isScreen: 是否是共享屏幕视频，推荐传 false。
- listener: 回调通知

传入 H264H264 每一帧数据接口：

```
public int inputEncodedFrame(long timestamp_ns, int w, int h, byte[] data, int len)
```

- timestamp\_ns: ns 时间戳
- w: 视频宽
- h: 视频高
- data: 视频数据
- len: 视频数据长度

示例代码：

```
MVideo.Camera camera = new MVideo.Camera(deviceId, deviceName); // 创建虚拟摄像头设备，传入设备 id 和设备名称
// 按照要发布的视频实际参数创建要发布摄像头数据：1920*1080 分辨率，30 帧，H264 编码格式。
PublishVideoOptions option = new PublishVideoOptions(new MVideo.CameraCapability(1920, 1080, 30),
    VideoOptions.VideoCodec.codec_h264);
camera.setPublishedQualities(option);
// 创建导入视频采集器
FakeVideoCapturer capturer = FakeVideoCapturer.Create(FakeVideoCapturer.FourccType.ft_H264,
    false, listener);
MVideo mVideo = MVideo.getVideo(room);
// 发布要导入的视频流，ret 返回值为 0 代表成功，发布视频会触发 onPublishCameraNotify 通知
int ret = mVideo.publishLocalCamera(camera, capturer);

// 循环发送 H264 视频帧数据
for( ; ; ) {
    capturer.inputEncodedFrame(timestamp_ns, 1920, 1080, importData, importData.length);
}
```

### 5.16 码流控制

针对使用默认的码流配置视频效果或带宽限制等不符合实际需求的情况，可手动配置码流大小。目前发布的所有视频（本地摄像头、共享屏幕、虚拟摄像头视频）共用一个码流参数，设置的码流大小需要动态根据发布的视频分辨率和路数情况来调整，否则存在码流偏大或偏小问题，会导致视频模糊马赛克等问题。通常情况下  $640 \times 480$  分辨率对应码流为 500kbps~1000kbps， $1280 \times 720$  分辨率对应码流为 1000kbps~1500kbps， $1920 \times 1080$  分辨率对应码流为 2000kbps~4000kbps。

**注：**发布本地视频（摄像头或共享屏幕）回调后再配置码流才会生效。

接口：

```
public int setVideoBitrate(String deviceId, int minBitrateBps, int maxBitrateBps)
```

- deviceId：设备 id，这里可传空字符串 ""。
- minBitrateBps：最小码流（bps）
- maxBitrateBps：最大码流（bps）

示例代码如下：

```
Room room = getRoom();
MVideo mVideo = MVideo.getVideo(room);
mVideo.setVideoBitrate("", 1000*1000, 2000*1000); // 码流大小 1000kbps~2000kbps
```



### 5.17 媒体统计 (Room、NetworkStats.MediaStats)

用于获取每一路订阅或自己发布的音频和视频信息，分为两种：一种是获取当前已订阅视频或自己已发布的视频分辨率、帧率、码流、丢包率等信息；另一种是将媒体统计信息打印到日志文件里，方便异常情况（视频黑屏、视频卡顿等）排查错误。需要注意的是媒体统计的支持需要 Room 开启媒体统计功能 `room.enableStats(true)`，以下对这两种方式分别加以说明。

A. 获取视频媒体信息：

```
NetworkStats.MediaStats stats = room.getMediaStats(videoId); // 获取设备 id 对应视频的媒体统计信息
if (stats != null) {
    int bpsSent = stats.getBps_sent() / 1000;           // 上行码流，bps 转 kbps
    int bpsReceived = stats.getBps_received() / 1000;  // 下行码流，bps 转 kbps
    int frameRate = stats.getFrame_rate();             // 帧率
    int frameWidth = stats.getFrame_width();           // 视频宽
    int frameHeight = stats.getFrame_height();         // 视频高
    int lostPercentSend = stats.getLostpercent_sent();  // 发送丢包率
    int lostPercentReceived = stats.getLostpercent_received(); // 接收丢包率
}
```

B. 将媒体信息打印到日志里：

除了需要房间开启媒体统计功能外还需要日志等级设置为 `verbose` 等级并设置日志 `stats` 状态才会每隔一秒的时间往日志里打印媒体统计信息，媒体统计信息日志如下图所示：

```
OnComplete: stream stats: item:14_85257a769961adb6aa533327eb62fbeaaeb2a4e0e4b68
bf53c65ea8e2f566798 video[ H264 dsp:1280x720 15] net[ bps:1072704 0 0 0 0 byte
s:1295959 0 pkt:1175 0 6 0 0 ]
```

- 14\_85257a769961adb6aa533327eb62fbeaaeb2a4e0e4b68bf53c65ea8e2f566798: 设备 id
- video[ H264 dsp:1280x720 15]: 代表是视频，H264 编码格式，分辨率 1280\*720，帧率 15 帧。
- bps:1072704 0 0 0 0: 5 个数据分别代表下行码流、上行码流、上下行总丢包率、下行丢包率、上行丢包率，如果是本地视频则下行码流为 0，如果是他人视频则上行为 0。

### 5.18 回调通知

SDK 涉及到的回调接口有引擎 `AVDEngine` 回调、房间管理 `Room` 回调、视频管理 `MVideo` 回调、共享屏幕 `MScreen` 回调、用户管理 `MUserManager` 回调、白板（标注）`MWhiteboard` 回调、音频管理 `MAudio` 回调、聊天功能 `MChat` 回调、视频渲染 `VideoRenderer` 第一帧视频到达通知。添加回调通过类实例对象方法 `setListener` 设置，例如：`room.setListener`、`mVideo.setListener` 等。回调方法参数列表含义可以查阅 SDK API 文档。

A. `AVDEngine` 引擎回调：

包括 `AVDEngine.Listener` 回调和音频采集 PCM 原始数据回调

设置 `AVDEngine.Listener` 回调：`AVDEngine.instance().init(context, listener, serverU`

```
RI, accessKey, secretKey)
```

设置音频采集 PCM 原始数据回调: `AVDEngine.instance().setAudioFrameCallback(JavaAudioDeviceModule.SamplesReadyCallback callback)`

```
AVDEngine.Listener:
    void onInitResult(int result);    // 初始化引擎异步返回
    void onUninitResult(int reason); // 反初始化引擎异步返回
    void onGetRoomResult(int result, RoomInfo room);    // 获取房间信息操作异步返回
    void onFindRoomsResult(int result, List<RoomInfo> rooms); // 查询房间信息操作异步返回
    void onScheduleRoomResult(int result, String roomId);    // 创建房间操作异步返回
    void onCancelRoomResult(int result, String roomId);    // 取消房间操作异步返回
    void onCallOutgoingDeviceResult(int result, String userId); // 呼叫外部设备回调

JavaAudioDeviceModule.SamplesReadyCallback 音频 PCM 原始数据回调:
    // AudioSamples 类包含音频数据信息: 音频格式、通道数、采样率、数据内容 byte 数组
    public void onWebRtcAudioRecordSamplesReady(AudioSamples samples)
```

B. 房间管理 Room 回调:

Room 回调分为加入房间回调通知 (JoinResultListener) 和房间信息类回调 (Listener) 两种。

- 设置JoinResultListener回调: `room.join(user, "", joinResultListener)`
- 设置Listener回调: `room.setListener(listener)`

```
JoinResultListener 回调:
    void onJoinResult(int result); // 加入房间操作异步返回

Listener 回调:
    void onJoinResult(int result); // 加入房间操作异步返回
    void onLeaveIndication(int reason, String fromId);    // 指示用户离开房间
    void onPublicData(byte[] data, int len, String fromId); // 透明通道, 接收到广播数据通知
    void onPrivateData(byte[] data, int len, String fromId); // 透明通道, 接收到私有数据通知
    void onAppDataNotify(String key, String value);    // 房间应用层数据更改通知
    void onRoomStatusNotify(RoomInfo.RoomStatus status); // 房间状态通知
    void onConnectionStatus(ConnectionStatus status);    // 房间网络状态通知
    void onOutgoingInviteStatusNotify(int type, String roomId,
                                      String addr, int status, String amsg); // sip 邀请状态通知
```

C. 视频管理 MVideo 回调:

分为视频状态回调, 本地摄像头事件监听, 本地摄像头使用 Camera1 接口时摄像头原始数据回调。

- 视频状态: `public boolean setListener(Listener listener)`
- 本地摄像头事件: `public static void setCameraEventListener(CameraVideoCapturer.CameraEventsHandler listener)`
- 本地摄像头使用Camera1接口时摄像头原始数据: `public void setPreviewCallback(NativeCapturerObserver.PreviewCallback callback)`

视频状态 Listener 回调：

```
void onCameraStatusNotify(DeviceStatus status, String fromId); // 摄像头状态更改通知
void onCameraDataNotify(int level, String description, String fromId); // 摄像头数据更改通知
void onPublishCameraNotify(Camera camera); // 摄像头视频发布通知
void onUnpublishCameraNotify(Camera camera); // 摄像头视频取消发布通知
void onSubscribeResult(int result, String fromId); // 本用户订阅视频异步返回
void onUnsubscribeResult(int result, String fromId); // 本用户取消订阅视频异步返回
void onPublishLocalResult(int result, String fromId); // 本用户发布摄像头视频异步返回
void onUnpublishLocalResult(int result, String fromId); // 本用户取消发布摄像头视频异步返回
```

本地摄像头事件监听 CameraVideoCapturer.CameraEventsHandler:

```
void onCameraError(String errorDescription); // 摄像头打开失败或运行异常
void onCameraDisconnected(); // 摄像头断开连接，被系统强制断开或被占用。
void onCameraFreezed(String errorDescription);
void onCameraOpening(String cameraName); // 摄像头打开回调
void onFirstFrameAvailable(); // 摄像头第一帧到达回调
void onCameraClosed(); // 摄像头关闭回调
```

本地摄像头使用 Camera1 接口时摄像头原始数据 NativeCapturerObserver.PreviewCallback:

```
void onPreviewFrame(byte[] data, int width, int height); // 摄像头 NV21 原始数据回调，data 数组
```

#### D. 共享屏幕 MScreen 回调：

共享屏幕状态更新，发布/停止发布共享屏幕，订阅/取消订阅共享屏幕回调通知。设置回调 mScreen.setListener(listener)，listener 即 MScreen.Listener 对象。

```
public void onScreenStatusNotify(Device.DeviceStatus status, String fromId); // 共享屏幕状态更改通知
public void onScreenDataNotify(int level, String description, String fromId); // 共享屏幕数据更改通知
public void onPublishScreenNotify(ScreenWindow screen); // 发布共享屏幕通知
public void onUnpublishScreenNotify(ScreenWindow screen); // 取消发布共享屏幕通知
public void onSubscribeResult(int result, String fromId); // 本用户订阅共享屏幕异步返回
public void onUnsubscribeResult(int result, String fromId); // 本用户取消订阅共享屏幕异步返回
```

#### E. 用户管理 MUserManager 回调：

示例：mUserManager.setListener(listaner)

```
public void onUserJoinNotify(User user); // 用户加入房间通知
public void onUserLeaveNotify(User user); // 用户离开房间通知
public void onUserLeaveNotify(int reason, User user); // 用户异常离开房间通知
public void onUserUpdateNotify(User user); // 用户信息更改通知
public void onUserStatusNotify(int status, String fromId); // 用户状态更改通知
public void onUserDataNotify(String userData, String fromId); // 用户应用层数据更改通知
```

#### F. 白板（标注）MWhiteboard 回调：

示例：mWhiteboard.setListener(listaner)

```
void onCreateBoardNotify(MWhiteboard.Whiteboard var1); // 创建白板通知（包括自己）
void onRemoveBoardNotify(String var1); // 清除白板通知
void onUpdateBoardNotify(MWhiteboard.Whiteboard var1); // 白板状态更新通知
void onShareBoardNotify(MWhiteboard.Whiteboard var1); // 共享白板通知
void onCloseBoardNotify(String var1); // 关闭白板通知
```

G. 音频管理 MAudio 回调：

麦克风状态变更（开关、静音等），参会用户开关麦克风，用户语音激励（语音激励功能打开），会议里播放视频流 TTS 均会触发该回调接口。

示例：mAudio.setListener(listener)

```
public void onMicrophoneStatusNotify(DeviceStatus status, String fromUserId); // 麦克风状态更改通知
public void onAudioLevelMonitorNotify(AudioInfo info); // 语音激励通知
public void onOpenMicrophoneResult(int result); // 本用户打开麦克风异步返回
public void onCloseMicrophoneResult(int result); // 本用户关闭麦克风异步返回
public void onAudioData(String userId, long timestamp, int sampleRate, // 音频数据回调接口
                        int channels, byte[] data, int len);
public void onMediaPlayNotify(String roomId, String playingMediaId, // TTS 媒体播放/停止回调通知
                              String userId, int playEvent);
```

H. 聊天功能 MChat 回调：

当会议里其他用户向当前用户发送共有或私有消息时会触发以下回调

示例：mChat.setListener(listener)

```
public void onPublicMessage(Message message); // 接收到公聊消息通知
public void onPrivateMessage(Message message); // 接收到私聊消息通知
```

I. 视频渲染 VideoRenderer 第一帧视频到达通知

监听渲染视频帧到达通知，用于界面上窗口状态展示，例如：没有渲染视频时显示占位图片或用户头像，视频第一帧到达后隐藏占位图片显示视频内容等。

示例：public void setFirstFrameCallback(FirstFrameCallback value)

```
public void onFirstFrameArrived(VideoRenderer render); // 只在第一帧视频达到时回调一次，取消渲染再次渲染时会再次回调。
```

## 5.19 扩展功能

扩展部分包括一些小的功能，包括：闪光灯开关控制（后置摄像头）、后置摄像头变焦（拉近拉远）、摄像头角度错误的矫正（图像是歪的未处于正方向）、设置代理等等。

A. 闪光灯控制：

Android 手机目前只有后置摄像头才有闪光灯功能，即如果当前使用 MVideo 打开的前置摄像头则没有闪光灯功能，需要切换至后置摄像头，打开摄像头功能请查阅 5.8 小节内容。SDK 支持基于 Camera1 接口和基于 Camera2 接口两种，可以通过 mVideo.enableCamera2API(true)来选择使用 Camera2 接口，false 使用 Camera1 接口，以下分别对两种接口控制闪光灯加以说明。

- 使用 Camera1 接口闪光灯控制：

```
if (Camera1Session.getInstance() != null) {
    Camera1Session.getInstance().turnOnFlash(); // 打开闪光灯
    Camera1Session.getInstance().turnOffFlash(); // 关闭闪光灯
}
```

- 使用 Camera2 接口闪光灯控制:

```
if (Camera2Session.getInstance() != null) {
    Camera2Session.getInstance().turnOnFlash();    // 打开闪光灯
    Camera2Session.getInstance().turnOffFlash();   // 关闭闪光灯
}
```

B. 摄像头变焦控制（拉近拉远）:

只有开启的是后置摄像头才有变焦功能，在变焦之前需要判断是否是后置摄像头。

```
if (Camera2Session.getInstance() != null && mVideo.isCameraPublished(MVideo.CameraType.back)) {
    float maxZoom = Camera2Session.getInstance().getMaxZoom(); // 最大的变倍值，设置的变倍数必须要
                                                                小于最大值。
    Camera2Session.getInstance().setCameraZoom(2.0f); // zoom 取值范围 [0~maxZoom]
}
```

C. 摄像头角度错误的矫正:

只有在部分盒子、TV、外接摄像头等定制设备上才会出现摄像头画面角度错误画面颠倒的情况，此类情况是因为设备问题，读回来的摄像头角度和实际画面角度不匹配导致。此时可以通过 SDK 预设角度偏移量对摄像头视频做一个角度的补偿旋转到正方向（只支持 Camera1 接口），具体偏移量角度设置多大需要根据实测来设置，偏移量角度范围 0~360 度。实例代码:

```
mVideo.setCameraOrientationOffset(90) // 逆时针旋转 90 度
```

D. AVDEngine 代理功能:

代理接口:

public void setProxy(ProxyType type, String ip, int port, String userName, String password)

- type: 代理的类型. 目前支持 Https 代理(音视频数据以 TCP 发送), Socks5 代理（音视频数据以 UDP 发送，推荐使用该代理方式）。
- ip: 代理的 IP 地址
- port: 代理的端口
- userName: 代理的认证账户(如果不认证,可传递空字符串)
- password: 代理的认证密码(如果不认证,可传递空字符串)

代码示例:

```
AVDEngine.getInstance().setProxy(AVDEngine.ProxyType.PROXY_HTTPS, "192.168.3.116", 808, "116",
                                "123456");

AVDEngine.getInstance().setNetworkGatewayAdapter(new AVDEngine.NetworkGatewayAdapter() {
    @Override
    public String onGetGatewayAddress(String ip, int port) { // 待转换 ip, 待转换 port
        return "192.168.3.100:8080"; // 主动返回代理的 ip 和 port
    }
});
```

## 5.20 离开/关闭房间并释放占用资源

离开房间而不释放 SDK 引擎：

```
Room.destoryRoom(room)
```

释放 SDK：

```
if (AVDEngine.instance().isWorking()) {
    AVDEngine.instance().uninit();
}
```

## 6 快速开发流程

预先在 Application 里初始化好 AVDEngine，这样可以避免加入会议时初始化引擎占用较多时间导致加房间慢的问题（多增加 1 秒左右时间）。而后依次执行获取动态权限、设置日志路径、设置引擎选项、初始化引擎（如有必要）、设置房间参数并加入房间、发布音视频、渲染视频等。具体操作步骤可以参考上一章节或参考 demo 代码具体实现。

## 7 常见错误处理

常见错误包括：引擎报错、房间报错、网络异常处理、音频问题、视频问题。

A. 引擎报错：

权限报错：

权限	说明
android.permission.INTERNET	没有网络权限初始化报 1014 错误
android.permission.ACCESS_NETWORK_STATE	没有网络状态访问权限初始化时会报异常崩溃

初始化报错：

错误码	说明
401	传入的密钥 Access Key 和 Secret Key 错误。
405	服务器授权过期，请联系续期。
436	AVD SDK 版本和服务器版本不适配，请联系我方对服务器做适配调整。
1008	初始化引擎传入的上下文 context 无效（为空或未传）。
1014	网络错误，需要排查传入的服务器地址是否正确，设备网络是否正常连接，网络质量是否良好，服务器端部署是否正常。

**B. 房间报错：**

创建房间报错：

错误码	说明
40003/40005	重复创建房间，房间已存在。
其他非 0 错误码	非 0 则是创建失败，这类错误码相对较少，出现时和我方确认解决。

加入房间：

错误码	说明
401	服务器认证失败
404	房间不存在
405	服务器授权过期，请联系续期。
1014	网络异常，检查设备和服务器整个网络链路是否正常。

离会回调通知：

在以下情况下会收到 **Room** 类离会回调通知（**void onLeaveIndication(int reason, String fromId)**）：和服务器中断连接（网络或其他连接异常）、房间内用户关闭房间、被其他用户踢出房间等。

收到离会回调后需要用户做离会操作，退出会议界面并释放房间（**destroyRoom**），具体被踢出房间原因可以根据回调 **reason** 值进行判断并在界面上给予提示，可对照下表排查原因：

特征码	离会原因
0	正常离会，用户自己调用 <b>destoryRoom</b> 离会
804	被相同 <b>userId</b> 用户入会挤下线
807	UDP 媒体通道建立超时，常见于 <b>join</b> 加会时。
808	该用户被其他用户踢出房间
809	服务没有授权用户入会
810	用户关闭房间时，所有用户被踢出房间。
811	服务器维护是关闭房间
812	和服务器之间心跳超时，异常离会。例如：网络中断、APP 被系统或后台杀掉。

会议中网络异常处理：

在会议过程中网络波动会导致设备和服务器心跳包中断，此时设备会做断线重连操作，会触发 **Room** 回调方法 **void onConnectionStatus(ConnectionStatus status)**，需要用户对这里的 **ConnectionStatus** 状态做判断。**Room.ConnectionStatus.connecting** 代表正在重连状态，**Room.ConnectionStatus.connectFailed** 代表重连失败，如果是后者重连失败情况下需要用户做离开房间操作，关闭会议界面；重连时间（单位 **ms**）可以

设置 Room 选项 `Room.Option.ro_room_rejoin_times` 来设置，不设置默认是重连 3 次（5 秒），"-1"代表一直重连。收到网络正在重连状态时最好在页面给用户一个提示，避免音视频中断影响使用效果。

### C. 音频问题：

音频问题常见于 Android 定制设备（机顶盒、一体机、TV）和部分 Pad（华为），现象包括麦克风采集没有声音，此时需要调整麦克风采集数据源，将语音通话音频源修改为麦克风音频源，具体可参考 5.8-J 章节；杂音、丢字、音色不理想情况可能由于音频采样率不匹配或由于使用软件回音消除不彻底导致，此类问题可以咨询排查。

### D. 视频问题：

常见视频问题包括视频黑屏、视频模糊花屏、视频卡顿、视频延时，出现黑屏或卡顿时可加入多个设备验证是发布（编码）问题还是接收视频（解码）这端有问题，例如：如果 A 发布视频，B 看 A 发布的视频黑屏，C 看 A 视频没有黑屏，则可能是 B 这端解码有问题导致；同理如果 B、C 看 A 发布视频均是黑屏，则可能是 A 端发布有问题。以下分别对几种情况加以说明：

#### 黑屏：

如果是本地打开视频黑屏则需要排查编码和打开摄像头是否有报错（发布本地摄像头回调 `result` 值是否为 0、查看日志分析是否有异常）；显示其他端的视频黑屏，排查是否视频有正常发布出来，可以加入其他设备进来查看是否黑屏，如果发布端没有问题，则需要排查本地解码视频是否有报错或者当前是否是弱网情况。在网络比较差的情况下，渲染多路视频较大概率出现视频显示黑屏问题，此时需要实时监测视频的媒体统计信息（详见 5.15-A），视频帧率恢复正常后再去做渲染操作。

#### 模糊花屏：

视频模糊（晃动镜头的时候会明显）一般由于码流设置偏低导致，需要将码流适当增加（详见 5.15 节）。通常情况下  $640 \times 480$  分辨率对应码流为 500kbps~1000kbps， $1280 \times 720$  分辨率对应码流为 1000kbps~1500kbps， $1920 \times 1080$  分辨率对应码流为 2000kbps~4000kbps。注：SDK 所有发布的视频（摄像头、共享屏幕、虚拟摄像头）共用一个码流配置。例如：发布一路  $1280 \times 720$  摄像头视频，码流设置为 1200kbps，帧率 20 帧。此时再发布一路  $1280 \times 720$  共享桌面视频，两路 720P 视频总码流是 1200kbps，可能会出现共享屏幕模糊问题，则需要根据发布的视频路数动态设置码流大小。

#### 卡顿：

卡顿通常情况是编解码帧率输出不稳定（编码或解码丢帧）和网络比较差丢包引起，需要查看 SDK 日志分析丢帧和丢包情况，结合设备 CPU 占用（CPU 占用过大）情况，路由器带宽占用情况，服务器交互日志具体分析。

#### 延时：

码流设置过大导致设备编解码压力增加或者由于网络环境差存在丢包均会出现视频延时问题，此类情况需要结合终端日志和服务器交互日志具体分析。



## 8 附录

### 8.1 名词解释

名词	解释	备注
room	房间对象，是实时沟通功能的一个管理单元，房间中会有多个沟通参与者即用户，房间有各种沟通功能，如文字聊天、语音视频等，沟通是基于房间的。不同房间沟通是隔离的	
user	用户对象，每个加入到房间的客户端作为一个房间用户，用户将会根据权限和设备情况执行房间中各种沟通功能。 用户 <b>Id</b> ：唯一标示一个房间用户的 <b>Id</b> ，由应用层来设置。	

### 8.2 错误码

错误码	描述	出现原因	处理方法	备注
1000	基本错误	一些基本的无法细分错误	提供 sdk 日志排查	
1001	参数错误	参数是否填错	检查传参	
1002	没有初始化	没有初始化引擎	检查是否初始化	
1003	已经初始化	sdk 内部逻辑通知	无	对应用层无影响
1004	未实现	该方法未实现	反馈客户端开发排查	
1005	空指针	代码逻辑	反馈客户端开发排查	
1006	未知异常	代码逻辑	反馈客户端开发排查	
1007	内存越界	代码逻辑	反馈客户端开发排查	
1008	非法参数	代码逻辑	反馈客户端开发排查	
1009	操作无效	代码逻辑	反馈客户端开发排查	
1011	对象没找到	代码逻辑	反馈客户端开发排查	
1014	超时	可能网络异常，或者服务器访问不了	先检查网络或端口是否有问题	
1015	对象错误状态	代码逻辑	反馈客户端开发排查	
1016	网络错误	1 在媒体通道还未连接上时	1 等媒体通道连接上后	

		去调用发布视频导致错误2 或者初始化引擎报错	再做音视频的操作2 检 查是否认证失败	
1017	没有 token	无	无	未用
1018	图像转换失败	主要是导入视频时报错	检测导入数据是否有问 题	
1019	缓存不够	使用 C 接口返回错误	c++到 c 转换数据时长度 不够，反馈 c++开发	
1020	设备被占用	比如摄像头已经被其他应 用正在使用，当前 sdk 无法 使用。	先检查是哪个应用正常 使用摄像头，先关闭掉 后 sdk 再使用	
1021	操作以及完成	无	无	sdk 内部通 知，不影响 应用层
1025	函数未认证	服务端验证是否有使用权 限	找客户端排查	
1026		无	无	未用
1027	mcu 服务器连接失败	由于媒体服务未连上产生 的发布视频错误	检查下媒体链接是否有 问题	
1028	视频不支持的分辨率			
1029	房间已经关闭	无	无	未用
1030	媒体流连接超时	媒体服务连接超时	检查下媒体链接是否有 问题	
1031	集群模式下获取 mcu 失败	初始化引擎返回的错误		
1032	房间信令连接失败	连接服务端的信令通道失 败	检查自己网络或者服务 器是否可连	
1033	房间数据连接失败	无	无	未用
1034	等待数据	抓图时，数据还没有	重复调用即可	

错误码	描述	出现原因	处理方法	备注
401	初始化引擎时未认证	认证的 token 或者 key 有问 题	检查是否 token 或 key 使用错误，如果无误则 需要找服务端排查	
402	客户端重复加入房间	内部逻辑，不会返回到应用 层	无	不影响应用 层

404	房间号不存在	房间号在服务器是不存在的	客户端先确认房间号，如果房间号正确，需要服务端排查	
405	license 不够	用户数超过最大 license 个数	找我们技术支持申请 license	
406-422				未用到，可以不用关心
434		无		未出现过，可以不用关心
445	主持人不在房间	主要使用 token 初始化引擎的用户	可以让主持人先加入房间或者修改后台设置	
503-508		无		未出现过，可以不用关心
601		无		未出现过，可以不用关心
602		无		未出现过，可以不用关心
612	该 mcu 服务器没有找到指定的房间	服务端的配置或者逻辑问题	需要服务端排查	
613		无		未出现过，可以不用关心
700		无		未出现过，可以不用关心
701		无		未出现过，可以不用关心
800		无		未用到，可以不用关心
801		无		未用到，可

				以不用关心
802	发布视频或音频时重复的设备 id	没有出现过	如果出现反馈日志，需要检查 sdk 逻辑	
803	房间错误的 token	没有出现过	如果出现，需要 sdk 和服务端排查下	
804	使用相同 id 加入房间，已经在房间的那个相同用户 id 会被踢出房间。	多个用户使用重复 id 的，则会被服务器踢出重复的。	保证每个加入房间用户 id 唯一	
805		无	无	暂未用
806		无	无	暂未用
807	sdk 无法链接上服务端的媒体通道，而被服务器踢出房间	可能网络或者服务端端口配置问题，sdk 无法连上服务端的媒体通道	检查网络或者检查服务器端口	
808	被其他用户踢出房间	某个用户通过 sdk 接口把自己踢出房间了	无	
809				
810	房间被关闭	某个用户通过 sdk 接口关闭了房间	无	
811	服务器关闭	可能服务器正在重启	等服务器重启成功后重连	
812	自己收到其他用户连接超时后被踢出房间的通知	某个用户可能网络异常导致无法链接服务器而被服务器踢出房间了	无	忽略
815	房间被关闭了	可能是其他用户通过 rest 接口关闭了房间	无	这个是正常逻辑，不影响应用层