

1、修订记录

2、概述

2.1、简介

2.2、面向读者

2.3、获取SDK Demo

2.4、技术支持

3、编写说明

4、环境准备

4.1.1、开发工具 DevEco Studio NEXT Developer Preview2

4.1.2、工程中导入RTC SDK进行开发

5、音视频基本功能开发流程

5.1、申请用户权限

5.2、RTC 引擎（AVDEngine）

5.2.1、日志系统

5.2.2、配置引擎选项

5.2.3、鉴权

5.2.4、启动RTC引擎

5.3、房间管理（Room）

5.3.1、获取房间管理对象

5.3.2、配置房间选项

5.3.3、设置房间事件监听

5.3.4、加入会议

5.3.5、离开会议

5.3.6、查询房间状态（网络状态）

5.3.7、房间网络状态事件通知

5.4、房间用户管理（MUserManager）

5.4.1、用户（User）

5.4.2、获取用户管理对象

5.4.3、设置用户事件监听

5.4.4、更改用户数据

5.4.5、用户异常离会

5.5、视频管理（MVideo）

5.5.1、获取视频管理对象

5.5.2、设置视频事件监听

5.5.3、发布视频

5.5.4、停止发布视频

5.5.5、订阅和取消订阅视频

5.5.6、渲染和停止渲染视频

1. RenderComponent 渲染组件：

2. 渲染/停止渲染：

5.5.7、读取视频状态

5.5.8、配置本地视频码流

5.6、音频管理（MAudio）

5.6.1、获取音频管理对象

5.6.2、设置音频事件监听

5.6.3、打开/关闭麦克风

5.6.4、查询麦克风状态

6、常见错误处理

1. 引擎报错：

2. 房间报错：

3. 离开会议：

4. 会议中网络异常处理：

5. 音频问题：

6. 视频问题：

7. 黑屏：

8. 模糊花屏：

9. 卡顿：

10. 延时：

7、附录

7.1、名词解释

7.2、错误码

# RTC SDK HarmonyOS版开发指南

(版本V3.1)

## 1、修订记录

*修订日期*	*描述*	*作者*	*版本号*
2024.07.17	初始文档	王龙渊	3.2.0
2025.07.09	版本修订	王龙渊	3.2.1

*修订日期*	*描述*	*作者*	*版本号*

## 2、概述

RTC SDK提供人与人实时沟通协作过程中需要用到所有基本能力，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。

RTC 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。

RTC 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，RTC SDK就是您最好的选择。市场调研表明，RTC 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

RTC 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。RTC SDK 采用动态共享包HSP本地化部署的方式，将 RTC\_SDK.tgz 包导入工程中进行应用开发，开发者可运行baseVideo示例工程来熟悉整个开发流程。

RTC SDK HarmonyOS版提供如下功能：

- 实时高清语音通话
- 实时高清视频通话

### 2.2、面向读者

本指南是提供给具有一定的HarmonyOS编程经验和了解面向对象概念的产品经理及程序员使用，RTC SDK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([Index of /3tee/sdk/harmony/](#))上会提供各类基于RTC 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联系。

Ø 技术支持工程师： 186XXXXXXX

## 3、编写说明

本指南编写目的是为了帮助使用RTC 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DevEco Studio NEXT Developer Preview2 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。

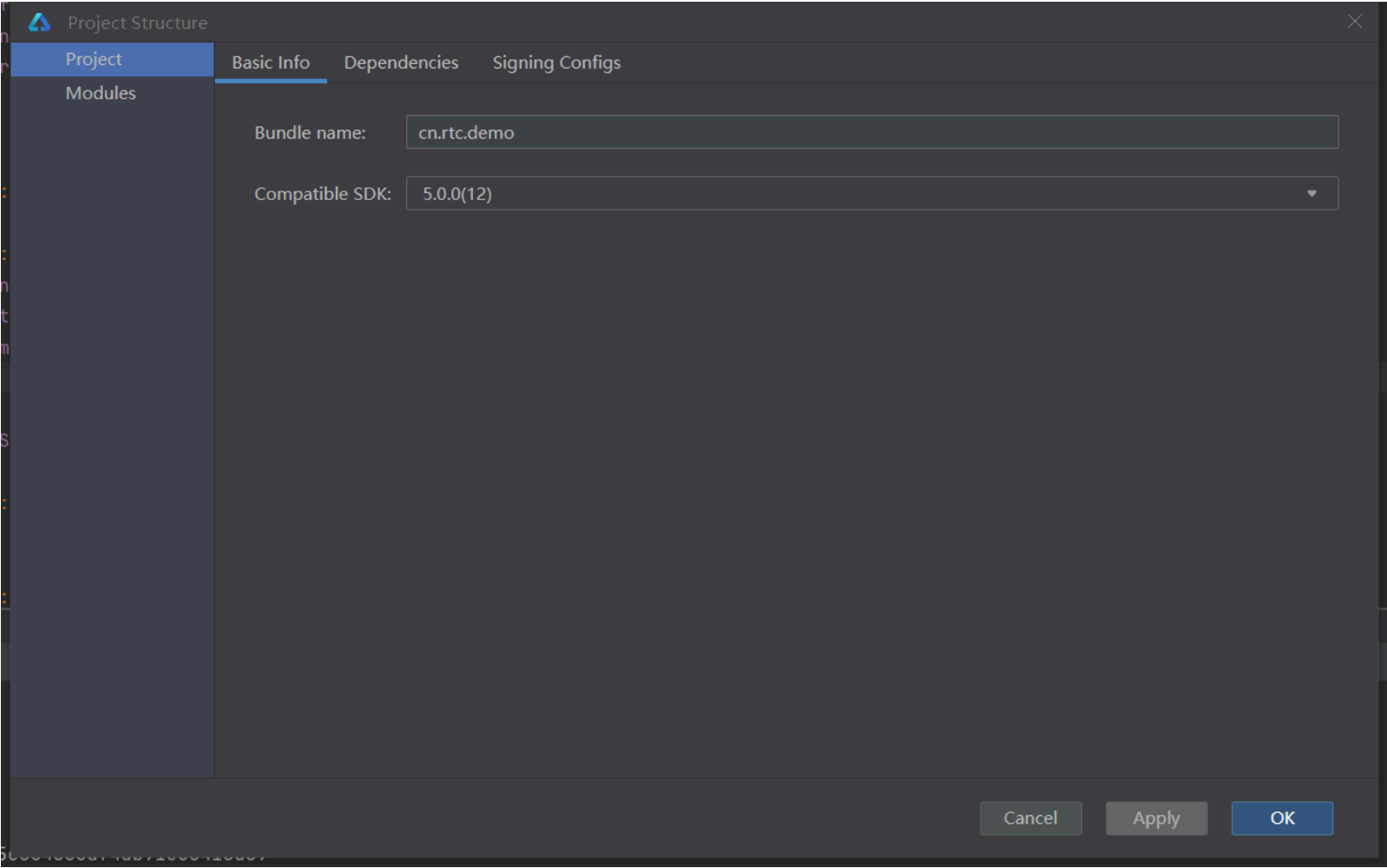
## 4、环境准备

### 4.1.1、开发工具 DevEco Studio NEXT Developer Preview2

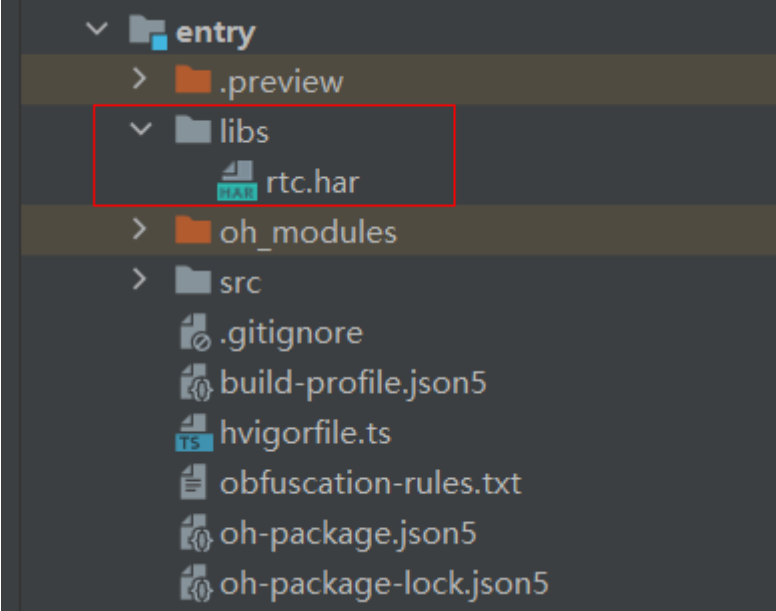
本指南采用 DevEco Studio NEXT Developer Preview2 作为工具开发，开发者可自行前往 [DevEco Studio 使用指南](#) 下载安装，并按照官方文档配置好开发环境。

### 4.1.2、工程中导入RTC SDK进行开发

在导入RTC SDK进行开发之前，开发者需要创建ArkTS应用（参考[ArkTS应用构建](#)），工程配置SDK版本选择5.0.0（API 12）。



接下来，在entry主模块下创建libs目录，并将rtc.har文件拷贝到此目录下。



配置entry下 oh-package.json5 文件引用 rtc.har 包

```
{
  "name": "entry",
  "version": "1.0.0",
  ...
  "dependencies": {
    "RTCSDK": "../libs/rtc.har"
  }
}
```

## 5、音视频基本功能开发流程

音视频开发流程：申请用户权限、日志系统配置（可选项）、引擎选项配置（可选项）、启动RTC引擎、加入会议、音视频管理（可选项）、用户管理（可选项）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。

**注：音视频功能需严格按照开发流程进行操作，未按照流程操作将影响使用效果。**

### 5.1、申请用户权限

RTC SDK 需要申请必要的权限来实现正常的视频会议功能，在未获得权限的情况下部分功能将受影响或不可用。

权限列表：

序号号	权限名称	用途
1	ohos.permission.INTERNET	网络请求
2	ohos.permission.CAMERA	相机功能开发
3	ohos.permission.MICROPHONE	麦克风功能开发
4	ohos.permission.READ_MEDIA	读取媒体文件

序号号	权限名称	用途
5	ohos.permission.WRITE_MEDIA	编辑媒体文件

权限申请流程如下：

- entry\src\main\module.json5 文件配置待申请权限

```
{
  "module": {
    "name": "entry",
    ... ..
    "requestPermissions": [
      {
        "name": "ohos.permission.INTERNET"
      },
      {
        "name": "ohos.permission.CAMERA",
        "reason": '$string:Camera_permission',
        "usedScene": {}
      },
      {
        "name": "ohos.permission.MICROPHONE",
        "reason": '$string:Camera_permission',
        "usedScene": {}
      },
      {
        "name": "ohos.permission.READ_MEDIA",
        "reason": '$string:Camera_permission',
        "usedScene": {}
      },
      {
        "name": "ohos.permission.WRITE_MEDIA",
        "reason": '$string:Camera_permission',
        "usedScene": {}
      }
    ]
  }
}
```

- 在配置好 module.json5 文件后，UIAbility中调用 abilityAccessCtrl.AtManager 的接口 requestPermissionsFromUser 动态向用户申请应用权限。动态申请权限时屏幕会弹出权限申请对话框，在点击确认后便获得相应权限， 向用户申请动态权限可参考 [官方文档“向用户申请授权”](#)

## 5.2、RTC 引擎（AVDEngine）

### 5.2.1、日志系统

日志系统用于记录RTC SDK运行状态，方便问题排查和运行流程监控。日志系统为可选项，在未配置日志系统情况下不会影响正常运行。日志系统建议在启动RTC引擎前进行配置，这样可以保留完整的调试信息。日志系统参数：生成路径配置、日志等级、日志时间标签格式、媒体统计等。日志参数由空格作为分割符，一个或多个参数关键字拼接而成的字符串。

- 接口定义：

```
/**
 * 配置RTC_SDK日志，建议采用配置参数
 * “debug verbose stats append realtstamp”
 * @param params 配置参数
 * @param filePath 日志文件路径
 * @returns 状态码（0代表成功）
public setLogParams(params: string, fileName: string): number
```

- params 配置参数：
  - 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）。
  - realtstamp：每条日志的时间戳使用当前时间格式 [月-日 小时：分钟：秒]。

filePath 日志文件路径

- 示例代码：

```
String params = "debug verbose stats append realtstamp";    // 日志参数
let applicationContext = this.context.getApplicationContext();

/* 读取应用文件目录（/data/storage/e12/base/files） */
let logPath = applicationContext.filesDir + "/日志名称.log";

/* 配置 RTC SDK 日志参数以及日志存放的位置 */
AVDEngine.instance().setLogParams(params, logPath);
```

### 5.2.2、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、发布到会议里的视频分辨率/帧率配置、数据通道格式（UDP/TCP）等。

- 接口定义：

```
/**
 * 配置引擎选项
 * @param optionName 选项名称
 * @param value 参数值
 * @returns 状态码（0代表成功）
 */
public setOption(optionName: EngineOption, value: string): number
```

- EngineOption 枚举定义：

```
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:     // 数据通道选择: TCP(true)或UDP(false)，默认UDP
eo_camera_capability_default:     // 设置发布视频的分辨率和帧率
```

- 示例代码：

```
// 以配置本地发布到会议中的视频分辨率为例：

/* 视频分辨率 1080P: 宽-1920、高-1080、帧率-30帧 */
let resolution: string = "{\"width\":1920,\"height\":1080,\"maxFPS\":30}"

/* 配置打开相机后发布的视频分辨率为1080P */
AVDEngine.setOption(EngineOption.eo_camera_capability_default, resolution)
```

### 5.2.3、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）用于鉴权认证。该密钥可生成访问令牌（24小时有效），用于加入会议、端口访问等操作。**注：启动引擎时鉴权失败会报401错误！**

- 服务器地址，根据当前web服务协议(http/https)指定对应的端口：

```
serverURI = "https://cd.nice2meet.cn:7820";
accessKey = "demo_access";
secretKey = "demo_secret";
```

- 在初始化引擎时需要传入服务器地址和key，启动引擎会在后续做详细说明：

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

### 5.2.4、启动RTC引擎

启动RTC引擎属于异步操作，启动引擎是否成功需要判断返回值，并在异步回调接口中对返回值做判断，返回值为0代表初始化成功，启动引擎成功后才能做后续加入房间操作。

- 接口定义：

```
/**
 * 初始化RTC_SDK引擎
 * @param context UIAbility的上下文
 * @param serverUrl 服务器地址
 * @param accessKey 鉴权 accessKey
 * @param secretKey 鉴权 secretKey
 * @param listener 回调接口
 * @returns 状态码（0代表成功）
 */
init(context: common.BaseContext, serverUrl: string, accessKey: string, secretKey: string, listener:
IAVDEngineListener)
```

- 启动失败错误码：

错误码	说明
401	传入的密匙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	启动引擎传入的上下文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。

- 异步回调接口：

```
export interface IAVDEngineListener {
  /**
   * 初始化引擎操作异步返回
   * @param result 状态码（0代表成功）
   */
  onInitResult: (result: number) => void

  /**
   * 释放引擎操作异步返回
   * @param result 状态码（0代表成功）
   */
  onUninitResult: (result: number) => void
}
```

- 示例代码：

```
let path = "https://cd.nice2meet.cn:7820";
let accessKey = "demo_access";
let secretKey = "demo_secret";

int result = AVDEngine.instance().init(this.context, path, accessKey, secretKey, this.engineListener);

if (result != ErrorCode.AVD_OK) {
  // 【启动引擎失败】
}

/* RTC SDK 引擎回调接口实现 */
private engineListener: IAVDEngineListener = {
  onInitResult: (result: number) => {
    if (result === 0) {
      // 【启动引擎成功】 后才能做加会等操作，否则会提示错误！
    } else {
      // 【启动引擎失败】
    }
  },
  onUninitResult: (result: number) => {
    // 【关闭引擎回调通知】
  }
}
```

### 5.3、房间管理（Room）

房间管理用于 本地用户信息管理（用户id、用户名、用户数据）、加入/离开会议、会议状态监听（和服务器连接状态）、网络诊断、用户数据互联等。除此之外，还用于各子模块的生命周期管理（用户管理模块MUserManager、视频管理模块MVideo、音频管理模块MAudio、屏幕共享管理模块MScreen、消息管理模块MChat）。

### 5.3.1、获取房间管理对象

在 5.2.4 节启动RTC引擎成功后，通过传入会议号的方式获取房间管理对象实例。

- 接口定义：

```
/**
 * 获取房间实例对象
 * @param roomId 房间号
 * @returns 房间句柄
 */
public static obtain(roomId: string): Room
```

- 示例代码：

```
let roomId = "10000000000" // 会议号
let room = Room.obtain(roomId)
if (room != null) {
    .....
    room.join(user, "") // 加入会议
}
```

### 5.3.2、配置房间选项

可配置 与服务器之间数据传输DTLS加密、与服务器之间网络中断重连机制（重连时间、次数、或一直重连）、配置音频编码格式（isac、opus、pcmu、pcma 等）、是否自动订阅音频等。

- 接口定义：

```
/**
 * 配置房间选项
 * @param optionName 选项名称
 * @param value 参数值
 * @returns 状态码（0代表成功）
 */
public setOption(optionName: RoomOption, value: string): number
```

- 示例代码：

```
let roomId = "10000000000" // 会议号
let room = Room.obtain(roomId)
if (room != null) {
    .....
    /* 断网机制设置为：中断后一直尝试连接服务 */
    room.setOption(RoomOption.ro_room_rejoin_times, "-1")
}
```

### 5.3.3、设置房间事件监听

本用户加入会议异步通知、本用户被踢出房间通知、公共和私有广播消息接收通知、更改应用层用户数据通知、网络连接状态改变通知

- 接口定义：

```
/**
 * 设置房间事件回调
 * @param listener 回调接口
 * @returns 状态码（0代表成功）
 */
public setListener(listener: IRoomListener): number
```

- 房间事件接口定义：

```
export interface IRoomListener {
    /**
     * 加入房间异步回调通知
     * @param result 状态码（0代表成功）
     */
    onJoinResult: (result: number) => void

    /**
     * 当前用户被邀请离开房间事件通知
     * @param reason 被邀请离开房间错误码（查阅ErrorCode）
     * @param userId 指示发起用户Id
     */
    onLeaveIndication: (reason: number, userId: string) => void

    /**
```



```

    * 房间类公共广播
    * @param data 广播数据载体
    * @param length 广播数据的长度
    * @param userId 发起人
    */
onPublicData: (data: number[], length: number, userId: string) => void

/**
    * 房间类私有广播
    * @param data 广播数据载体
    * @param length 广播数据的长度
    * @param userId 发起人
    */
onPrivateData: (data: number[], length: number, userId: string) => void

/**
    * 房间应用层数据更改通知
    * @param key 应用层数据关键字
    * @param value 应用层数据关键字相关数据内容
    */
onAppDataNotify: (key: string, value: string) => void

/**
    * 房间网络状态通知
    * @param status 网络状态
    */
onConnectionStatus: (status: ConnectionStatus) => void
}
```

- 示例代码：

```
let roomId = "10000000000" // 会议号
let room = Room.obtain(roomId)
if (room != null) {
    .....
    room.setListener(this.roomListener)
}
```

5.3.4、加入会议

在 5.2.4 中启动引擎回调成功后才能做加入会议操作，当会议号不存在时，加入该会议会异步回调（onJoinResult）404错误。加入会议前可配置房间参数用于控制房间行为（如服务断线重连等），加入会议后会收到异步回调 onJoinResult 通知，在收到通知后判断是否加入成功（0代表成功）。

- 接口定义：

```
/**
    * 加入房间
    * @param user 用户信息
    * @param password 房间密码（若会议未配置密码，传入空字符串"" ）
    * @returns 状态码（0代表成功）
    */
public join(user: User, password: string): number
```

- user 用户信息：

```
status: UserStatus // 用户状态集合
userId: string // 用户Id，应用层可设置
userName: string // 用户名称，应用层设置
userAgent: string // 携带应用数据 如 iOS Android web pc
userData: string // 用户数据，应用层设置
```

- 加会议失败错误码：

错误码	说明
401	服务器认证失败
404	会议号不存在
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。

- 加入会议流程如下：

- 启动RTC引擎
- 配置房间参数（非必选项）



- 创建用户并加入会议

- 示例代码：

```
let room = Room.obtain(roomId) // 获取房间管理对象
if (room != null) {
    let ret: number = room.setListener(this.roomListener) // 设置房间事件监听
    if (ret === 0) {
        /* 断线重连机制设置为网络断线后一直尝试重新连接服务 */
        room.setOption(RoomOption.ro_room_rejoin_times, "-1")
        /* 创建加入会议的用户，用户号设置为随机UUID */
        let user = new User(util.generateRandomUUID(), userName, "")
        room.join(user, "") // 加入会议
    }
}

/* 房间事件监听 */
private roomListener: IRoomListener = {
    .....
    onJoinResult: (result: number) => {
        if (result === 0) {
            /* 加入会议成功 */
        } else {
            /* 加入会议失败 */
        }
    }
}
```

5.3.5、离开会议

离开会议并释放Room及各个子模块资源（用户管理模块MUserManager、视频管理模块MVideo、音频管理模块MAudio、屏幕共享管理模块MScreen、消息管理模块MChat）。

- 接口定义：

```
/**
 * 离开会议
 * @returns 状态码（0代表成功）
 */
public leave(): number
```

- 示例代码：

```
let roomId = "10000000000" // 会议号
let room = Room.obtain(roomId)
if (room != null) {
    .....
    room.leave()
}
```

5.3.6、查询房间状态（网络状态）

查询房间是否正常与服务器建立连接，在 5.3.4 小结中加入会议并收到“onJoinResult: (result: number) => void” 加入会议成功通知后，房间状态会变更为已连接状态。

- 接口定义：

```
/**
 * 查询房间状态是否正常（与服务器连接正常）
 * @returns
 */
public isworking(): boolean
```

- 示例代码：

```
let isworking = room.isworking()
```

5.3.7、房间网络状态事件通知

加入会议后，RTC SDK 与服务端网络中断以及重新与服务端建立连接的事件，均会通过 IRoomListener 的 onConnectionStatus: (status: ConnectionStatus) 接口下发给监听端，其中 ConnectionStatus 为网络状态枚举对象，ConnectionStatus.CONNECTING 正在建立网络连接，ConnectionStatus.CONNECTED 连接已恢复。

- ConnectionStatus 网络状态：

```
export enum ConnectionStatus {
    READY = 0,           // 初始状态
    CONNECTING = 1,       // 正在连接服务器
    CONNECTED = 2,        // 已经连接上服务器
    CONNECT_FAILED = 3,    // 连接服务器失败
    DATA_CONNECTING = 4,  // 媒体数据通道正在连接中
    DATA_CONNECTED = 5    // 媒体数据通道已连接成功
}
```

- 示例代码：

```
private roomListener: IRoomListener = {
    .....
    onConnectionStatus: (status: ConnectionStatus) => {
        switch (status) {
            /* 由于网络等原因，与服务器的连接已断开，等待重连服务。 */
            case ConnectionStatus.CONNECTING:
                promptAction.showToast({
                    message: "媒体服务已断开，重连中...",
                    duration: 2000,
                    bottom: 100
                })
                break
            /* 恢复与服务器的连接 */
            case ConnectionStatus.CONNECTED:
                promptAction.showToast({
                    message: "媒体服务已恢复！",
                    duration: 2000,
                    bottom: 100
                })
                break
        }
    }
}
```

## 5.4、房间用户管理（MUserManager）

用户管理类 MUserManager 用于监听会议内用户加入会议、离开会议、用户更改用户名称/用户标签数据、用户摄像头/麦克风开关状态变更等事件。MUserManager 类对外提供 updateUser 接口用于更改本地用户数据、获取用户实例对象（User）、获取会议参会者列表、关联用户 User 和设备（相机/麦克风）等功能。

### 5.4.1、用户（User）

用户数据使用 User 类定义，在加入会议过程中使用 User 对参会者加以描述。User 类包含的成员变量有：用户号 userId、用户名 userName、用户数据 userData（string 用户自定义数据）、用户设备信息 userAgent（iOS 平台、Android 平台、PC 平台等，用户无需关心该数据）、用户状态（摄像头/麦克风设备开光状态）status。

```
enum UserStatus {
    MICROPHONE_HAS = 0x80000000, /*拥有麦克风设置位*/
    MICROPHONE_ON = 0x40000000, /*麦克风打开设置位*/
    CAMERA_HAS = 0x20000000, /*拥有摄像头设置位*/
    CAMERA_ON = 0x10000000, /*摄像头打开设置位*/
    SCREEN_ON = 0x2000000, /*屏幕窗口共享设置位*/
    IDLE = 0x00
}

class User {
    status: UserStatus // 用户状态集合
    userId: string // 用户Id，应用层可设置
    userName: string // 用户名称，应用层设置
    userAgent: string // 用户设备信息：harmony、iOS、Android
    userData: string // 用户数据，应用层设置
}
```

### 5.4.2、获取用户管理对象

在 5.3.4 节加入会议成功后，通过传入 Room 对象的方式获取用户管理对象实例。

- 接口定义：

```
public static getUserManager(room: Room): MUserManager
```

- 示例代码：

```
let room = Room.obtain(roomId)
if (room != null) {
    let mUserManager = MUserManager.getUserManager(room)
}
```

### 5.4.3、设置用户事件监听

用户事件包括以下内容：会议中用户加入/离开会议（会议中其他用户，不包含自己）、用户信息（User）更改、用户状态更改、用户应用层数据更改

5.3.4 和 5.3.5 小节中，用户加会/离会操作，会议中其他用户会触发 onUserJoinNotify 和 onUserLeaveNotify 事件通知。

用户调用“mUserManager.updateUser(...)、mUserManager.updateUserName(...)、mUserManager.updateSelfUserStatus(...)”接口修改用户数据User和更改用户状态，会议中其他用户会触发 onUserUpdateNotify 和 onUserStatusNotify 事件通知。

- 接口定义：

```
/**
 * 设置视频事件回调接口
 * @param listener 回调接口
 * @returns 状态码（0代表成功）
 */
public setListener(listener: IMUserManagerListener)
```

- 用户事件接口定义：

```
export interface IMUserManagerListener {
  /**
   * 其他用户加入房间通知
   * @param user 用户对象
   */
  onUserJoinNotify: (user: User) => void
  /**
   * 其他用户离开房间通知
   * @param reason 离会原因，0 -> 正常离会，非0 -> 异常离会
   * @param user 用户对象
   */
  onUserLeaveNotify: (reason: number, user: User) => void
  /**
   * 用户信息更改通知（某用户调用 updateUser 更改自己信息后，
   * 房间内所有用户会接收到此通知）
   * @param user 用户对象
   */
  onUserUpdateNotify: (user: User) => void
  /**
   * 用户状态更改通知
   * @param status 用户状态
   * @param userId 用户id
   */
  onUserStatusNotify: (status: number, userId: string) => void
  /**
   * 用户应用层数据更改通知
   * @param userData 当前用户应用层数据
   * @param userId 用户id
   */
  onUserDataNotify: (userData: string, userId: string) => void
}
```

- 示例代码：

```
let mUserManager = MUserManager.getUserManager(room)
mUserManager.setListener(this.mUserManagerListener)
```

### 5.4.4、更改用户数据

修改当前用户自定义数据（User.userData）赋能多种应用场景的扩展，通过 onUserUpdateNotify 事件通知会议中其他用户，同步用户自定义数据。

- 接口定义：

```
/**
 * 获取当前用户信息
 * @returns 返回本用户信息
 */
public getSelfUser(): User

/**
 * 更新当前用户信息
 * @param user 用户信息
 * @returns 状态码（0代表成功）
 */
public updateUser(user: User): number
```

- 示例代码：

```
let mUserManager = UserManager.getUserManager(room)
let user = mUserManager.getSelfUser()
user.userData = "user data"
mUserManager.updateUser(user)
```

### 5.4.5、用户异常离会

5.3.5 小节中，用户离会操作，会议中其他用户会触发 onUserLeaveNotify: (reason: number, user: User) 事件通知。

reason 参数代表用户离开会议原因，reason 值为0代表正常离会，非0为异常离会，reason 错误码如下表所示：

状态码code	离会原因
0	正常离会
804	被相同userId用户入会挤下线
807	UDP媒体通道建立超时，常见于join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。

## 5.5、视频管理（MVideo）

本小节有以下内容：当前用户视频发布/停止发布（开关相机）、视频的订阅、视频的渲染、会议中用户视频状态查询等功能。

在 5.3.4 小节成功加入会议后，才能调用视频相关操作。 用户开关相机会通知会议内所有用户，接收到开关相机事件（发布/停止发布视频）后需要先做视频订阅（取消订阅），而后才可以进行视频渲染（停止渲染）操作，这里渲染的含义是指界面上展示该视频。

### 5.5.1、获取视频管理对象

在 5.3.4 节加入会议成功后，通过传入Room对象的方式获取视频管理对象实例。

- 接口定义：

```
let mVideo = MVideo.getVideo(room)
```

- 示例代码：

```
let room = Room.obtain(roomId)
if (room != null) {
    let mVideo = MVideo.getVideo(room)
}
```

### 5.5.2、设置视频事件监听

当前用户发布本地视频 publishLocalCamera（打开相机），自己会收到 onPublishLocalResult 事件通知，会议中其他用户会收到 onPublishCameraNotify 事件通知（如果发布失败则不会收到该通知）；当前用户停止发布本地视频 unpublishLocalCamera（关闭相机），自己会收到 onUnpublishLocalResult 事件通知，其他用户会收到 onUnpublishCameraNotify 事件通知（如果停止发布失败则不会收到该通知）。

在接收到他人发布视频通知后（onPublishCameraNotify），才能做视频订阅的操作（自己不能订阅自己发布的视频），订阅视频 subscribe 后，会收到订阅事件通知 onSubscribeResult，并判断是否订阅成功（result 为 0）；同理，在接收到他人停止发布视频通知后（onUnpublishCameraNotify），需要做取消视频订阅的操作，取消订阅视频 unsubscribe 后，会收到取消订阅事件通知 onUnsubscribeResult，并判断是否取消订阅成功（result 为 0）。注：在发布视频后才能做订阅操作，否则会订阅失败；每订阅一路视频，会相应增加一路

- 接口定义：

```
/**
 * 设置视频事件回调接口
 * @param listener 回调接口
 * @returns 状态码（0代表成功）
 */
public setListener(listener: IMVideoListener)
```

- 视频事件接口定义：

```
export interface IMVideoListener {
    /**
     * 相机状态更改通知
     * @param status 就绪、发布
```

```

    * @param deviceId 视频id（设备id）
    */
onCameraStatusNotify: (status: number, deviceId: string) => void
/**
    * 相机数据更改通知
    * @param level 相机数字数据
    * @param description 相机字符数据
    * @param deviceId 视频id（设备id）
    */
onCameraDataNotify: (level: number, description: string, deviceId: string) => void
/**
    * 视频发布通知
    * @param camera 相机参数
    */
onPublishCameraNotify: (camera: Camera) => void
/**
    * 停止发布视频通知
    * @param camera 相机参数
    */
onUnpublishCameraNotify: (camera: Camera) => void
/**
    * 订阅视频异步返回通知
    * @param result 状态码（0代表成功）
    * @param deviceId 视频id
    */
onSubscribeResult: (result: number, deviceId: string) => void
/**
    * 取消订阅视频异步返回通知
    * @param result 状态码（0代表成功）
    * @param deviceId 视频id
    */
onUnsubscribeResult: (result: number, deviceId: string) => void
/**
    * 发布本地视频异步返回通知
    * @param result 状态码（0代表成功）
    * @param deviceId 视频id
    */
onPublishLocalResult: (result: number, deviceId: string) => void
/**
    * 停止发布本地视频异步返回通知
    * @param result 状态码（0代表成功）
    * @param deviceId 视频id
    */
onUnpublishLocalResult: (result: number, deviceId: string) => void
}

```

- 示例代码：

```
let mVideo = MVideo.getVideo(room)
mVideo.setListener(this.mVideoListener)
```

### 5.5.3、发布视频

通过 MVideo 的 getVideo(room) 接口获取MVideo实例对象， setListener 设置视频相关事件监听。当前用户调用接口 publishLocalCamera 发布本地视频（打开相机）， 随后当前用户会触发 onPublishLocalResult 异步事件通知发布视频结果（发布成功/失败）， 同时会议中其他用户会收到 onPublishCameraNotify 异步事件通知（如发布失败则不会收到该事件）。

**视频发布流程：**发布视频 -> 接收发布视频事件 -> 订阅该视频 -> 订阅异步事件 -> 订阅成功 -> 渲染（显示）该视频

- 接口定义：

```
/**
    * 发布本地视频：打开制定的相机（前置/后置相机），并将该相机视频流发布到当前会议里。
    * 房间里其他用户在收到 “onPublishCameraNotify” 回调后方可订阅和渲染该视频。
    * @param cameraType 相机类型（前置/后置）
    * @returns 状态码（0代表成功）
    */
public publishLocalCamera(cameraType: CameraType): number

```

- 相机类型 CameraType：

```
export enum CameraType {
    UNKNOWN = 0,    /* 未知类型 */
    FRONT = 1,      /* 前置摄像头 */
    BACK = 2         /* 后置摄像头 */
}
```

- 示例代码：

```
let room = Room.obtain(roomId) // 获取房间管理对象
let mVideo: MVideo = null
if (room != null) {
    let ret: number = room.setListener(this.roomListener) // 设置房间事件监听
    if (ret === 0) {
        let user = new User(util.generateRandomUUID(), userName, "")
        let mVideo = MVideo.getVideo(room)
        mVideo.setListener(this.mVideoListener)
        room.join(user, "") // 加入会议
    }
}

/* 房间事件监听 */
private roomListener: IRoomListener = {
    .....
    onJoinResult: (result: number) => {
        if (result === 0) {
            /* 加入会议成功，发布本地视频 */
            mVideo?.publishLocalCamera(cameraType)
        } else {
            /* 加入会议失败 */
        }
    }
}

/* 视频事件监听 */
private mVideoListener: IMVideoListener = {
    .....
    /* 发布视频回调 */
    onPublishCameraNotify: (camera: Camera) => {
        /* 订阅视频，会触发异步回调 onSubscribeResult */
        mVideo?.subscribe(camera.id)
    },
    /* 关闭视频回调 */
    onUnpublishCameraNotify: (camera: Camera) => {
        /* 取消订阅视频，会触发异步回调 onUnsubscribeResult */
        mVideo?.unsubscribe(camera.id)
    },
    /* 发布本地视频（打开本地相机）回调 */
    onPublishLocalResult: (result: number, deviceId: string) => {
        if (result === 0) {
            /* 设置本地发布的视频码流 1500kbps ~ 2000kbps */
            mVideo?.setVideoBitrate(deviceId, 1500 * 1000, 2000 * 1000)
            /* 渲染视频（显示视频） */
            mVideo?.attachRender(deviceId, "XComponentId01")
        }
    },
    .....
}
```

### 5.5.4、停止发布视频

调用接口 `unpublishLocalCamera` 停止发布本地视频（关闭相机），随后当前用户会触发 `onUnpublishLocalResult` 异步事件通知停止发布视频结果（停止发布成功/失败），同时会议中其他用户会收到 `onUnpublishLocalResult` 异步事件通知（如停止发布失败则不会收到该事件）。发布到会议里的视频会有相应的设备号 `deviceId` 与之对应，后续 订阅/取消订阅 视频使用设备号对视频进行识别。

**视频停止流程：** 停止发布视频 -> 接收停止发布视频事件 -> 取消订阅该视频 -> 取消订阅异步事件 -> 取消订阅成功 -> 停止渲染（显示）该视频

- 接口定义：

```
/**
 * 停止发布本地视频
 * @returns 状态码（0代表成功）
 */
public unpublishLocalCamera(): number
```

- 示例代码：

```
let room = Room.obtain(roomId) // 获取房间管理对象
let mVideo: MVideo = null
if (room != null) {
    let ret: number = room.setListener(this.roomListener) // 设置房间事件监听
    if (ret === 0) {
        let user = new User(util.generateRandomUUID(), userName, "")
        let mVideo = MVideo.getVideo(room)
        mVideo.setListener(this.mVideoListener)
        room.join(user, "") // 加入会议
    }
}
```



```
/* 房间事件监听 */
private roomListener: IRoomListener = {
    .....
    onJoinResult: (result: number) => {
        if (result === 0) {
            /* 加入会议成功，停止发布本地视频 */
            mVideo?.unpublishLocalCamera()
        } else {
            /* 加入会议失败 */
        }
    }
}

/* 视频事件监听 */
private mVideoListener: IMVideoListener = {
    .....
    /* 发布视频回调 */
    onPublishCameraNotify: (camera: Camera) => {
        /* 订阅视频，会触发异步回调 onSubscribeResult */
        mVideo?.subscribe(camera.id)
    },
    /* 关闭视频回调 */
    onUnpublishCameraNotify: (camera: Camera) => {
        /* 取消订阅视频，会触发异步回调 onUnsubscribeResult */
        mVideo?.unsubscribe(camera.id)
    },
    /* 停止发布本地视频（关闭本地相机）回调 */
    onUnpublishLocalResult: (result: number, deviceId: string) => {
        /* 取消渲染视频（关闭显示视频） */
        mVideo?.detachRender("xComponentId01")
    }
    .....
}
```

### 5.5.5、订阅和取消订阅视频

在收到视频发布事件通知后，且该视频非自己发布的视频才可订阅，而自己发布的视频无需订阅即可做渲染显示。使用接口 subscribe 订阅视频会触发 MVideo.IMVideoListener 回调 onSubscribeResult；使用接口 unsubscribe 取消订阅视频会触发 MVideo.IMVideoListener 回调 onUnsubscribeResult。

- 接口定义：

```
/**
 * 订阅他人发布的视频：会议其他参会者发布视频后，本参会者才可以
 * 订阅该视频，订阅后会收到该视频下行视频流（会占用部分下行带宽），
 * 而后可自行做视频渲染展示等操作。
 * @param deviceId 待订阅的视频id
 * @returns 状态码（0代表成功）
 */
public subscribe(deviceId: string): number

/**
 * 取消订阅他人发布的视频：取消订阅房间里已发布的视频，
 * 此操作将中断该视频的下行视频流，下次渲染展示该视频需要
 * 重新调用“subscribe”订阅
 * @param deviceId 待停止订阅的视频id
 * @returns 状态码（0代表成功）
 */
public unsubscribe(deviceId: string): number {
```

- 示例代码：

```
/* 视频事件监听 */
private mVideoListener: IMVideoListener = {
    .....
    /* 发布视频回调 */
    onPublishCameraNotify: (camera: Camera) => {
        /* 订阅视频，会触发异步回调 onSubscribeResult */
        mVideo?.subscribe(camera.id)
    },
    /* 关闭视频回调 */
    onUnpublishCameraNotify: (camera: Camera) => {
        /* 取消订阅视频，会触发异步回调 onUnsubscribeResult */
        mVideo?.unsubscribe(camera.id)
    },

    /* 订阅视频回调 */
    onSubscribeResult: (result: number, deviceId: string) => {
```



```
        /* 获取空闲的渲染组件（未渲染视频） */
        let render: RenderComponent | null = RenderComponent.getFreeRender()
        if (render && result === 0) {
            /* 渲染视频（显示视频） */
            mVideo?.attachRender(deviceId, render.getRenderId())
            /* 渲染组件绑定设备id（视频id） */
            render.bindDeviceId(deviceId)
        }
    },
    /* 取消订阅视频回调 */
    onUnsubscribeResult: (result: number, deviceId: string) => {
        /* 通过设备id获取绑定的渲染组件 */
        let render: RenderComponent | null = RenderComponent.getRenderByDeviceId(deviceId)
        if (render) {
            /* 取消渲染视频（关闭显示视频） */
            mVideo?.detachRender(render.getRenderId())
            render.unbindDeviceId()
        }
    },
    .....
}
```

5.5.6、渲染和停止渲染视频

在 5.5.5 小节订阅视频（取消订阅）成功后才能对视频进行渲染（取消渲染）操作，渲染视频时需要提前在应用UI界面添加RTC SDK渲染组件 RenderComponent，屏幕上需要同时显示多少路视频就需要添加对应数目的 RenderComponent 组件。使用 attachRender 接口进行视频渲染，使用 detachRender 接口停止渲染。

1. RenderComponent 渲染组件：

该自定义组件用于视频展示，同时具有组件管理功能。在该组件未设置 enableIndependent 属性为 false 时，所有 RenderComponent 组件均会放入列表进行维护；enableIndependent 属性设为 true 时，该组件不会添加到列表进行管理。

列表管理的 RenderComponent 组件可通过以下接口快捷获取：可通过 RenderComponent 接口 getFreeRender 获取空闲的组件（未展示视频的组件），通过 getRenderByRenderId 接口传入组件id来获取对应的渲染组件，通过 getRenderByDeviceId 传入设备id来获取展示该视频的渲染组件。

- 接口定义：

```
/**
 * 获取正在渲染的视频id
 * @returns 视频id
 */
public getDeviceId(): string

/**
 * 将视频绑定到渲染组件实例对象
 * @param deviceId 视频id
 */
public bindDeviceId(deviceId: string): void

/**
 * 解绑视频
 */
public unbindDeviceId(): void

/**
 * 通过视频id获取对应的渲染组件实例对象
 * @param deviceId 视频id
 * @returns 渲染控件实例
 */
public static getRenderByDeviceId(deviceId: string): RenderComponent | null

/**
 * 获取渲染组件id
 * @returns 组件id
 */
public getRenderId(): string

/**
 * 通过组件id获取渲染组件实例对象
 * @param renderId
 * @returns
 */
public static getRenderByRenderId(renderId: string): RenderComponent | null

/**
 * 获取处于空闲状态（未渲染视频）的渲染组件
 * @returns 渲染组件
 */
```

```
public static getFreeRender(): RenderComponent | null

/**
 * 获取渲染组件列表
 * @returns 组件列表
 */
public getRenderList(): Array<RenderComponent>

/**
 * 清空渲染组件列表
 */
public cleanRenderList(): void
```

- 示例代码：

```
@Entry()
@Component
struct Meeting {
    build() {
        column() {
            column() {
                /* enableIndependent: 作为独立的组件即不添加到列表中进行管理 */
                RenderComponent({renderId: "XComponentId01", enableIndependent: true})
                    .margin(2)
                    .layoutWeight(1)
                    .width('100%')
                RenderComponent({renderId: "XComponentId02"})
                    .margin(2)
                    .layoutWeight(1)
                    .width('100%')
                RenderComponent({renderId: "XComponentId03"})
                    .margin(2)
                    .layoutWeight(1)
                    .width('100%')
            }
            .height('90%')
            .width('100%')
        }
        .backgroundColor('#bbbbbb')
        .height('100%')
        .width('100%')
    }
}
```

2. 渲染/停止渲染：

- 接口定义：

```
/**
 * 渲染视频：将 “deviceId” 对应的视频渲染到 “renderId” 绑定的控件显示，
 * 这里的 “renderId” 是由 RenderComponent 控件创建时由应用层指定。
 * @param deviceId 设备（视频）id
 * @param renderId RenderComponent 控件 surfaceId
 */
public attachRender(deviceId: string, renderId: string)

/**
 * 停止渲染视频：停止 RenderComponent 控件正在渲染的视频
 * @param renderId RenderComponent 控件 surfaceId
 */
public detachRender(renderId: string)
```

- 代码示例：

```
/* 渲染视频（显示视频） */
mVideo?.attachRender(deviceId, render.getRenderId())

/* 取消渲染视频（关闭显示视频） */
mVideo?.detachRender(render.getRenderId())
```

5.5.7、读取视频状态

提供一些辅助判断视频状态的接口：获取当前本地发布的视频id、查询视频是否发布到会议里、查询本地视频是否已发布

- 接口定义：

```
/**
 * 获取当前本地发布的视频id（相机的id）
 * @returns 视频id（设备id）
```

```
 */
public getCurrentCameraId(): string | null

/**
 * 查询视频是否发布到会议里
 * @param deviceId 视频id（设备id）
 * @returns true: 已发布, false: 未发布
 */
public isCameraPublished(deviceId: String): boolean

/**
 * 查询本地视频是否已发布（是否打开了本地相机）
 * @returns true: 已发布, false: 未发布
 */
public isPublishedLocalCamera(): boolean
```

### 5.5.8、配置本地视频码流

可自定义本地发布到会议中的视频码流范围，适应不同场景对画质的要求。

**注意：**配置视频码流需要接收到发布本地视频事件通知后才可操作。

- 接口定义：

```
/**
 * 设置视频推流码流取值范围（单位bps）
 * @param deviceId 本地发布的设备id（视频id）
 * @param minBitrateBps 最小码流
 * @param maxBitrateBps 最大码流
 * @returns 状态码（0代表成功）
 */
public setVideoBitrate( deviceId: string, minBitrateBps: number, maxBitrateBps: number): number
```

- 示例代码：

```
/* 设置本地发布的视频码流 1500kbps ~ 2000kbps */
mVideo?.setVideoBitrate(deviceId, 1500 * 1000, 2000 * 1000)
```

## 5.6、音频管理（MAudio）

音频管理提供本地麦克风开关功能，用于监听会议内参会用户音频状态。

### 5.6.1、获取音频管理对象

在 5.3.4 节加入会议成功后，通过传入Room对象的方式获取音频管理对象实例。

- 接口定义：

```
/**
 * 获取MAudio实例
 * @param room 房间管理实例
 * @returns MAudio实例
 */
public static getAudio(room: Room): MAudio
```

- 示例代码：

```
let room = Room.obtain(roomId)
if (room != null) {
    let mAudio = MAudio.getAudio(room)
}
```

### 5.6.2、设置音频事件监听

用于监听会议内用户音频状态改变（打开/关闭麦克风、静音/取消静音）事件，语音激励事件通知，当前用户开关麦克风异步事件通知等。

- 接口定义：

```
/**
 * 设置音频事件回调接口
 * @param listener 回调接口
 * @returns 状态码（0代表成功）
 */
public setListener(listener: IMAudioListener)
```

- 音频事件接口定义：

```
export interface IMAudioListener {
```

```
/**
 * 麦克风状态更改通知：房间用户麦克风状态（打开/关闭/静音）更改后，
 * 房间内所有用户接收到此通知。
 * @param status 麦克风状态
 * @param fromUserId 麦克风状态改变的用户id
 */
onMicrophoneStatusNotify: (status: number, fromUserId: string) => void

/**
 * 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
 */
onAudioLevelMonitorNotify: (/*AudioInfo info*/) => void

/**
 * 打开本地麦克风异步回调通知
 * @param result 状态码（0代表成功）
 */
onOpenMicrophoneResult: (result: number) => void

/**
 * 关闭本地麦克风异步回调通知
 * @param result 状态码（0代表成功）
 */
onCloseMicrophoneResult: (result: number) => void
}
```

- 示例代码：

```
let mAudio = MAudio.getAudio(room)
mAudio.setListener(this.mAudioListener)
```

### 5.6.3、打开/关闭麦克风

当前用户打开麦克风 openMicrophone 会触发 onOpenMicrophoneResult 事件通知，通过 result 是否为0来判断打开是否成功；关闭麦克风 closeMicrophone 会触发 onCloseMicrophoneResult 事件通知，通过 result 是否为0来判断关闭是否成功。

- 接口定义：

```
/**
 * 打开麦克风
 */
public openMicrophone(): void

/**
 * 关闭麦克风
 */
public closeMicrophone(): void
```

- 示例代码：

```
let room = Room.obtain(roomId) // 获取房间管理对象
let mAudio: MAudio = null
if (room != null) {
    let ret: number = room.setListener(this.roomListener) // 设置房间事件监听
    if (ret === 0) {
        let user = new User(util.generateRandomUUID(), userName, "")
        let mAudio = MAudio.getAudio(room)
        mAudio.setListener(this.mVideoListener)
        room.join(user, "") // 加入会议
    }
}

/* 房间事件监听 */
private roomListener: IRoomListener = {
    .....
    onJoinResult: (result: number) => {
        if (result === 0) {
            mAudio.openMicrophone() // 打开麦克风
            mAudio.closeMicrophone() // 关闭麦克风
        } else {
            /* 加入会议失败 */
        }
    }
}

/* 麦克风事件监听 */
private mAudioListener: IMAudioListener = {
    .....
    /* 打开本地麦克风异步结果返回 */
}
```

```
onOpenMicrophoneResult: (result: number): void => {
    if (result === 0) {
        /* 打开成功 */
    }
},

/* 关闭本地麦克风异步结果返回 */
onCloseMicrophoneResult: (result: number): void => {
    if (result === 0) {
        /* 关闭成功 */
    }
},
.....
}
```

### 5.6.4、查询麦克风状态

- 接口定义：

```
/**
 * 获取麦克风状态  — 1：关闭状态，2：打开状态
 */
public getMicrophoneStatus(): number
```

## 6、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。

### 1. 引擎报错：

错误码	说明
401	传入的密匙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。

### 2. 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。

### 3. 离开会议：

特征码	离会原因
0	正常离会
804	被相同userId用户入会挤下线
807	UDP媒体通道建立超时，常见于join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。

特征码	离会原因
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。

## 4. 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 onConnectionStatus，需要用户对状态做判断。ConnectionStatus.connecting 代表正在重连状态，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 RoomOption.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。

## 5. 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。

## 6. 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：

## 7. 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，视频帧率恢复正常后再去做渲染操作。

## 8. 模糊花屏：

视频模糊（晃动镜头的时候会更明显）一般由于码流设置偏低导致，需要将码流适当增加。通常情况下640×480分辨率对应码流为500kbps1000kbps，1280×720分辨率对应码流为1000kbps1500kbps，1920×1080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1280×720摄像头视频，码流设置为1200kbps，帧率20帧。此时再发布一路1280×720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。

## 9. 卡顿：

卡顿通常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。

## 10. 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。

# 7、附录

## 7.1、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Id：唯一标示一个房间用户的Id，由应用层来设置。	

## 7.2 、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	



<b>*错误码*</b>	<b>*描述*</b>	<b>*出现原因*</b>	<b>*处理方法*</b>	<b>*备注*</b>
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去调用发布视频导致错误2或者初始化引擎报错	1等媒体通道连接上后再做音视频的操作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	
<b>*错误码*</b>	<b>*描述*</b>	<b>*出现原因*</b>	<b>*处理方法*</b>	<b>*备注*</b>



*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	

*错误码*	*描述*	*出现原因*	*处理方法*	*备注*
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