

小溪音视频Android\_SDK开发手册

V 1.4.6

2017-04-05

# Android\_SDK接口(VoIP)文档（Android）

## 版本号：V1.4.6

本文档是针对音视频Android SDK为开发者和企业提供能力开放而编写的详细说明文档，包含SDK介绍、开发指南、接口说明、附录等几部分，旨在帮助开发者和企业快速集成小溪的音视频能力。

## 文档修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **版本** | **修订内容** | **修订时间** | **修订人** | **备注** |
| 1 | V1.0.0 | 新增：   1. 登录、退出 2. 1v1通话、电话状态监听；   3、 即时/预约会议 | 2015/01/29 |  |  |
| 2 | V1.0.0 | 优化：  1、1v1语音通话稳定性及音质  2、电话状态的监听 | 2015/03/31 |  |  |
| 3 | V1.1.0 | 新增：   1. 发起即时会议 2. 发起预约会议   优化：  1、用户登录、呼叫接口（传参由手机号改为用户名）  2、通话稳定性  修复：  被叫拒绝后，主叫进入错误状态的bug | 2015/05/08 |  |  |
| 4 | V1.2.0 | 优化：   1. 消息推拉结合的改造 2. 通话功能的稳定性 | 2015/05/28 |  |  |
| 5 | V1.2.0 | 优化：  来电监听里增加来电类型的字段 | 2015/06/02 |  |  |
| 6 | V1.2.0 | 新增   1. 发送DTMF指令 2. 初始化视频配置 3. 1V1视频通话 | 2015/06/15 |  |  |
| 7 | V1.3.0 | 新增   1. 邀请成员到会议 2. 剔除会议成员 | 2015/07/28 |  |  |
| 8 | V1.3.0 | 新增：   1. 切换前后置摄像头 2. 关闭或打开本地摄像头 | 2015/09/10 |  |  |
| 9 | V1.3.1 | 优化   1. 创建会议 2. 主动入会 | 2016/02/05 |  |  |
| 10 | V1.4 | 增加   1. 恢复当前通话 2. 获取本地预览视图 3. 获取远端渲染视频 4. 设置Token 5. 添加VoIP服务器状态监听   6、变声 | 2017/01/22 |  |  |
| 11 | V1.4.1 | 优化   1. 切换前后置摄像头；   增加   1. 获取远端渲染的ssrc; 2. 特效 3. 美颜 | 2017/02/06 |  |  |
| 12 | V1.4.2 | 更新：  去除会议状态的监听接口；增加会议信息接口（其中包含会议信息、会议成员列表、会议状态更新）；将ssrc放入sdk处理，会议信息接口列表中返回渲染的视频 | 2017/02/28 |  |  |
| 13 | V1.4.3 | 更新：  更新voip服务器配置 | 2017/03/21 |  |  |
| 14 | V1.4.4 | 更新：  会议控相关接口整合到sdk中 | 2017/03/24 |  |  |
| 15 | V1.4.5 | 更新  统一接口名 | 2017/04/05 |  |  |
| 16 | V1.4.6 | 更新：  增加直播、增加IMS登录 | 2017/11/15 |  |  |

## 目录

[小溪音视频Android\_SDK开发手册 1](#_Toc498706995)

[Android\_SDK接口(VoIP)文档（Android） 2](#_Toc498706996)

[版本号：V1.4.6 2](#_Toc498706997)

[文档修订记录 2](#_Toc498706998)

[目录 4](#_Toc498706999)

[一、SDK介绍 7](#_Toc498707000)

[1.1 能力说明 7](#_Toc498707001)

[二、前期准备 8](#_Toc498707002)

[2.1 环境要求 8](#_Toc498707003)

[2.2 添加SDK权限 8](#_Toc498707004)

[2.3 集成SDK能力 9](#_Toc498707005)

[三、接口说明 10](#_Toc498707006)

[3.1 公共模块 10](#_Toc498707007)

[3.1.1初始化 11](#_Toc498707008)

[3.1.2设置adapter服务器地址 11](#_Toc498707009)

[3.1.3设置VoIP服务器地址及conference地址 12](#_Toc498707010)

[3.1.4用户登录 13](#_Toc498707011)

[3.1.5注册来电监听 15](#_Toc498707012)

[3.1.6注册电话状态监听 17](#_Toc498707013)

[3.1.7注册服务器连接状态监听 19](#_Toc498707014)

[3.1.8注销电话状态监听 20](#_Toc498707015)

[3.1.9注销服务器连接状态监听 21](#_Toc498707016)

[3.1.10注册codec观察者 22](#_Toc498707017)

[3.1.11注销codec观察者 24](#_Toc498707018)

[3.1.12账号退出 24](#_Toc498707019)

[3.1.13 SDK销毁 25](#_Toc498707020)

[3.1.14检测摄像头是否可用 26](#_Toc498707021)

[3.1.15检测麦克风是否可用 27](#_Toc498707022)

[3.1.16设置视频图像分辨率 27](#_Toc498707023)

[3.1.17注册底层开启异常监听 29](#_Toc498707024)

[3.1.18注销底层开启异常监听 30](#_Toc498707025)

[3.1.19获取摄像头状态（通话过程中使用） 30](#_Toc498707026)

[3.2 1v1通话模块 31](#_Toc498707027)

[3.2.1 拨打音频/视频 31](#_Toc498707028)

[3.2.2 接听电话 33](#_Toc498707029)

[3.2.3 挂断电话 34](#_Toc498707030)

[3.2.4 设置所有会话输入静音 35](#_Toc498707031)

[3.2.5 设置单个会话输入静音 36](#_Toc498707032)

[3.2.6 设置所有会话输出静音 36](#_Toc498707033)

[3.2.7 设置单个会话输出静音 37](#_Toc498707034)

[3.2.8 获取1v1本地预览视频 38](#_Toc498707035)

[3.2.9 获取远端渲染视频 39](#_Toc498707036)

[3.2.10 切换前后摄像头 40](#_Toc498707037)

[3.2.11 特效 41](#_Toc498707038)

[3.2.12 美颜 45](#_Toc498707039)

[3.2.13 音频切换到视频 46](#_Toc498707040)

[3.2.14 视频切换到音频 47](#_Toc498707041)

[3.2.15 同意切换到视频 48](#_Toc498707042)

[3.2.16 拒绝切换到视频 48](#_Toc498707043)

[3.3 会议模块 49](#_Toc498707044)

[3.3.1 发起即时会议 49](#_Toc498707045)

[3.3.2 发起预约会议 52](#_Toc498707046)

[3.3.3 主动入会 53](#_Toc498707047)

[3.3.4 邀请会议成员 55](#_Toc498707048)

[3.3.5 剔除会议成员 56](#_Toc498707049)

[3.3.6 注册会议成员信息列表监听 58](#_Toc498707050)

[3.3.7 注销会议成员信息列表监听 60](#_Toc498707051)

[3.3.8 注册会议状态更新监听 60](#_Toc498707052)

[3.3.9 注销会议状态更新监听 62](#_Toc498707053)

[3.3.10 关闭会议 63](#_Toc498707054)

[3.3.11 通过会议号请求会议信息 64](#_Toc498707055)

[3.3.12 获取我的会议列表 66](#_Toc498707056)

[3.3.13 锁定会议 67](#_Toc498707057)

[3.3.14 会议成员禁言 69](#_Toc498707058)

[3.3.15 变声 70](#_Toc498707059)

[3.4 TV相关模块 71](#_Toc498707060)

[3.4.1 获取摄像头个数 71](#_Toc498707061)

[3.4.2 刷新摄像头信息 72](#_Toc498707062)

[3.4.3 关闭摄像头采集（TV 1v1视频使用） 73](#_Toc498707063)

[3.4.4 打开摄像头采集（TV 1v1视频使用） 73](#_Toc498707064)

[3.5 IMS相关模块 74](#_Toc498707065)

[3.5.1 注册IMS服务 74](#_Toc498707066)

[3.5.2 设置转换IMS服务 76](#_Toc498707067)

[四、状态说明 77](#_Toc498707068)

[4.1 电话接通状态码 77](#_Toc498707069)

[4.2 查询会议状态失败的错误码 77](#_Toc498707070)

[五、附录 78](#_Toc498707071)

[5.1 通话状态说明 78](#_Toc498707072)

[5.2 通话类型 78](#_Toc498707073)

[5.3 登录状态 79](#_Toc498707074)

[5.4 特效模式 79](#_Toc498707075)

[5.5 变声模式 79](#_Toc498707076)

一、SDK介绍

小溪VoIP SDK（Android）\_V1.4.6是一套基于Android 2.3及以上版本设备的应用程序接口，提供登录、来电监听、拨打、接听、挂断电话、语音会议等功能。通过集成本SDK能方便快速地帮您的Android应用实现1对1通话、召开语音会议、视频会议等多种功能。

1.1 能力说明

**公共模块**

**初始化**：在应用创建的时候，调用该接口初始化SDK，设置服务器地址

**用户登录：**用于账号的登录

**注册来电监听：**提供给已登录账号，当有来电时进入账号的回调函数中

**注册电话状态监听：**提供给已登录账号，当电话的状态发生改变时进行相应的处理

**注销电话状态监听：**提供给已登录账号，注销对电话状态改变的监听

**账号退出：**提供给已登录账号，用于注销通道，便于下一次登陆

**1V1通话模块**

**拨打电话：**提供给已登录账号，用于对在线的其他账号进行拨号

**接听电话：**提供给已登录账号，用于在来电的状态下，将电话接起来

**挂断电话：**提供给已登录账号，用于在来电或者正在通话时，将电话挂断，最终使通话状态回到空闲状态

**设置静音：**提供给已登录账号，用于在通话中，将自己的发音屏蔽，使对方听不到自己侧的声音

**恢复通话：**恢复当前通话，当视频界面关闭，需要恢复surfaceview的时候，必须先调用这个接口

**切换前后置摄像头**：视频通话中，切换前后置摄像头

**获取本地视频：**返回SurfaceView类型，即本地视频的view，将返回的SurfaceView加入到Layout中就可以显示本地视频。

**获取远端渲染视频：**当用户登录成功以后，在拨打1V1视频或者视频会议并进入通话状态后调用该接口获取远端渲染的。

**添加会议成员状态变更监听：**在创建即时会议、预约会议时，本成员可以实时对其它会议成员的状态变更进行监听。能监听到其它会议成员进入会议，退出会议。

**会议模块**

**设置Token:** 在计算机中作为身份认证

**发起即时会议：**提供给已登录账号，用于邀请其他在线账号立即召开会议

**发起预约会议：**提供给已登录账号，用于邀请其他在线账号发起预约会议

**主动入会**：主动进入一个会议

**邀请会议成员：**提供给已登录账号，在会议进行时，邀请指定的用户到指定的会议中

**剔除会议成员：**提供给已登录账号，在会议进行时，剔除指定的会议成员

**查询会议状态：**会议创建成功后，创建者或者其他成员需要查询本会议中所有成员的状态时调用该接口，非会议成员无权限查询。

**添加VoIP服务器状态监听：** 在登录成功后，添加对VoIP服务器状态的监听。如登录VoIP服务器成功，失败，断开与服务器连接，与服务器连接成功等状态

**添加会议状态监听：**在创建即时会议、预约会议时，本成员可以实时对其它会议成员的状态变更进行监听。能监听到其它会议成员进入会议，退出会议。

**变声：**在音频会议中，可以将自己的声音变为男声、女声、童声。默认为自己原声。

二、前期准备

通过以下几步，您就能快速地在您的Android应用中集成小溪的语音通讯能力。

2.1 环境要求

在您集成 SDK 前环境要求如下：

* Android SDK Build-tools 请升级到 23 及以上版本。
* JAVA 编译版本 JDK 1.7 及以上版本。

Android SDK 最低支持 Android API 9: Android 2.3(Gingerbread)。

2.2 添加SDK权限

#### a) 权限列表

INTERNET:允许程序联网和发送聊天数据的权限  
ACCESS\_NETWORK\_STATE:允许查看当前网络状态  
ACCESS\_WIFI\_STATE: 允许程序访问 Wi-Fi 网络状态信息  
WRITE\_EXTERNAL\_STORAGE:允许程序进行读写sdcard  
RECORD\_AUDIO:允许程序进行语音录制的权限  
CALL\_PHONE: 允许拨打电话的权限  
WAKE\_LOCK: 允许唤醒屏幕的权限  
VIBRATE: 允许振动的权限  
RECORD\_AUDIO：允许录制音频

#### b) 示例代码

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <manifest ……>  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="19"/>  <uses-permission android:name="android.permission.INTERNET" />  <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />  <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  <uses-permission android:name="android.permission.CALL\_PHONE " />  <uses-permission android:name="android.permission.WAKE\_LOCK " />  <uses-permission android:name="android.permission.VIBRATE " />  <uses-permission android:name="android.permission.RECORD\_AUDIO" />  <application ……>  <activity ……></activity>  </application>  </manifest> |

2.3 集成SDK能力

完成上述导入及配置后，您即可根据应用的业务流程集成小溪的SDK。下图为语音通讯的基本流程：



三、接口说明

3.1 公共模块

本模块的接口提供用户账号相关能力，包含用户登录、来电监听、注册电话状态监听、注销电话状态监听、账户退出等。

### 3.1.1初始化

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

初始化SDK

#### 调用场景

在application启动的onCreate()方法中调用该方法,只需初始化一次

#### 方法原型

CMVoIPManager.getInstance().init(Context mContext);

#### 调用实例

|  |
| --- |
| //初始化 CMVoIPManager.getInstance().init(mContext); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| mContext | 上下文 | 是 | Application上下文 |

#### 异常处理

1. mContext为空时，抛出空指针异常。

### 3.1.2设置adapter服务器地址

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

设置adapter服务器地址。(注：adapter是用于获取VoIP服务器地址及Conference地址)

#### 调用场景

在SDK初始化后设置服务器地址。(3.1.2 与 3.1.3 两者设置一个即可)

#### 方法原型

CMVoIPManager.getInstance().setAdapterServerAddress(String adapterServerAddress);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().init(Context mContext); *//配置adapter服务器地址* CMVoIPManager.getInstance().setAdapterServerAddress(String adapterServerAddress); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| adapterServerAddress | String | 是 | Sdk adapter地址 |

### 3.1.3设置VoIP服务器地址及conference地址

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

设置VoIP服务地址及Conference地址。

#### 调用场景

在SDK初始化后设置服务器地址。(2.1.2 与 2.1.3 两者设置一个即可)

#### 方法原型

CMVoIPManager.getInstance().setVoipServerAddress(String voipServerAddress, String conferenceServerAddress)

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().init(Context mContext); CMVoIPManager.getInstance().setVoipServerAddress("218.218.218.218:5061", "218.218.218.218:1080"); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| voipServerAddress | String | 是 | Voip服务器地址 |
| conferenceServerAddress | String | 是 | Conference服务器地址 |

### 3.1.4用户登录

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

用于voip账户登录

#### 调用场景

在app登录处调用

#### 方法原型

CMVoIPManager.getInstance().doLogin(Map<String, String> params, VoIPLoginCallBack callBack)

#### 调用实例

|  |
| --- |
| Map<String, String> map = new HashMap<String, String>(); map.put(VoIPConstant.ACCOUNT, userName); map.put(VoIPConstant.APPKEY, voip\_appKey); map.put(VoIPConstant.ACCOUNTTOKEN, voip\_token); CMVoIPManager.getInstance().doLogin(map, new VoIPLoginCallBack() {  @Override  public void onLoginSuccess() {  CMVoIPManager.getInstance().addInComingCallListener(voIPInComingCallListener);  CMVoIPManager.getInstance().addCallStateListener(LittlecService.this);  CMVoIPManager.getInstance().addServerConnectListener(serverConnectListener);  CMVoIPManager.getInstance().addConferenceStateListener(confStateCallback);  }  @Override  public void onLoginFailed(int code) {  switch (code) {  case VoIPConstant.VOIP\_GET\_SERVER\_CONFIG\_ERROR:*//无返回值*  break;  case VoIPConstant.VOIP\_LOGIN\_INVALID\_APPKY: *//appKey无效*  break;  case VoIPConstant.VOIP\_GET\_SERVER\_CONFIG\_ERROR:*//解析出错*  break;   case VoIPConstant.VOIP\_GET\_SERVER\_CONFIG\_ERROR: *//网络错误*  break;  default:  break;  }  }  }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| userName | String | 是 | 用户名 |
| Voip\_appKey | String | 是 | Appkey(应该为6位数字后加2位字母) |
| voip\_token | String | 是 | 登录token |
| voIPLoginCallBack | VoIPLoginCallBack | 是 | 登录结果回调 |

#### 异常处理

1. map或VoIPLoginCallBack为空时，会抛出空指针异常。
2. appkey格式不对时，会抛出异常。(应该为6位数字后加2位字母)
3. 回调的errorCode:

|  |  |  |
| --- | --- | --- |
| Error | Code | 描述 |
| VOIP\_GET\_SERVER\_CONFIG\_ERROR | 57 | 无返回值、解析出错、网络错误 |
| VOIP\_LOGIN\_INVALID\_APPKY | 56 | 登录失败，APPKEY无效 |
| VOIP\_LOGIN\_INVALID\_USER | 52 | 用户名不合法 |
| VOIP\_NET\_ERROR | 54 | 登录服务error |
| VOIP\_INIT\_FAIL | 51 | 初始化失败 |

### 3.1.5注册来电监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

对当前登录账号的电话状态进行监听，如果当前有来电则进入来电监听的回调方法中。

#### 调用场景

在voip登录成功的回调中注册

#### 方法原型

CMVoIPManager.getInstance().addInComingCallListener(VoIPInComingCallListener mListener);

#### 调用实例

|  |
| --- |
| voIPInComingCallListener = new VoIPInComingCallListener() {  @Override  public void onInComingCall(String phoneNumber, int callType, int session) {  //VoIPConstant.CallType.CALLTYPE\_1V1\_AUDIO 1v1音频  //VoIPConstant.CallType.CALLTYPE\_1V1\_VIDEO 1v1视频  //VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_AUDIO 音频会议  //VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO 视频会议  //todo  }  };   CMVoIPManager.getInstance().doLogin(map, new VoIPLoginCallBack() {  @Override  public void onLoginSuccess() {  //登录成功后注册  CMVoIPManager.getInstance().addInComingCallListener(voIPInComingCallListener);   }  @Override  public void onLoginFailed(int code) {    }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| mListener | VoIPInComingCallListener | 是 | 来电回调接口 |

### 3.1.6注册电话状态监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

监听电话的各种状态，拨打中、振铃中、拨打失败、接通、释放等。

#### 调用场景

登录成功后调用注册

#### 方法原型

CMVoIPManager.getInstance().addCallStateListener(VoIPCallStateCallBack callBack)

#### 调用实例

|  |
| --- |
| VoIPCallStateCallBack voIPCallStateCallBack = new VoIPCallStateCallBack() {  @Override  public void onCallProceeding(int session) {   }  @Override  public void onCallAlerting(int session) {  // 被叫如果有回应，就返回这个事件。当收到这个事件时，对方会响起振铃音，本地会响起回铃音。   }  @Override  public void onStopCallAlerting(int session) {  // 接收服务器推送的铃声之前,要停止本地铃声和震动   }  @Override  public void onCallAnswered(int session, int callType) {  // 外呼时，被叫接听，就返回这个事件。接收到这个事件，表明对方已经应答，进入通话状态。   }  @Override  public void onCallForward(int session) {  //OTT呼叫转IMS事件。收到转接后，app再调用呼叫IMS   }  @Override  public void onLogout() {  //登出事件。   }  @Override  public void onMakeCallFailed(int session, String status) {  // 如果呼叫失败，就返回这个事件。外呼失败的原因有很多，可以根据status来判断   }  @Override  public void onCallReleased(int session) {  // 呼叫被释放回调   }  @Override  public void onReceiveCallSwitch(int session) { // 对方将音频切换成视频时，就返回这个事件，然后应用层同意或者拒绝。  // 如果对方是将视频切换成音频，本方会直接主动切换成音频。  }  @Override  public void onCallReBuildResult(int session, int callType) {  // 同意或者拒绝切换后返回的事件，callType是返回的处理切换后的电话类型，0表示是语音通话(即语音切换至视频被拒绝了，或者视频切换成语音成功了)，1表示是视频通话（即语音切换成视频被同意了）  // 或者网络中断，然后重新建立成功了，0表示是语音通话；1表示是视频通话  }  }; CMVoIPManager.getInstance().addCallStateListener(voIPCallStateCallBack); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| voIPCallStateCallBack | VoIPCallStateCallBack | 是 | 监听电话状态改变回调接口 |

#### 异常处理

1. voIPCallStateCallBack为空会抛出异常；
2. onMakeCallFailed回调的status：

|  |  |
| --- | --- |
| Status | 描述 |
| “4” | 空号 |
| “403” | 没有加入会议的权限 |
| “Server Time-Out” | 断网超时 |

### 3.1.7注册服务器连接状态监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

监听voip登录服务器状态，包含登录成功、失败。重连等状态。

#### 调用场景

登录成功之后调用注册

#### 方法原型

CMVoIPManager.getInstance().addServerConnectListener(VoIPServerConnectListener serverConnectListener);

#### 调用实例

|  |
| --- |
| serverConnectListener = new VoIPServerConnectListener() {  @Override  public void onLoginSucceed(int sipFlag) {  //登录成功 sipFlag == 0为OTT登录成功，1为IMS  }  @Override  public void onLoginFailed(int failedReason, int sipFlag) {  //登录失败 sipFlag == 0为OTT登录成功，1为IMS  }  @Override  public void onImsLogging(int result, int sipFlag) {  //IMS登录状态  }  @Override  public void onDisConnected(int failedReason) {  //断开连接  }  @Override  public void onConnectSucceed() {  //连接成功  }  }; CMVoIPManager.getInstance().addServerConnectListener(serverConnectListener）; |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| serverConnectListener | VoIPServerConnectListener | 是 | 监听连接状态回调接口 |

### 3.1.8注销电话状态监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

注销电话状态监听

#### 调用场景

当要销毁监听状态时调用

#### 方法原型

CMVoIPManager.getInstance().removeCallStateListener(VoIPCallStateCallBack callBack);

#### 调用实例

|  |
| --- |
| @Override public void onDestroy() {  super.onDestroy();  CMVoIPManager.getInstance().removeCallStateListener(this); } |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callback | VoIPCallStateCallBack | 是 | 监听电话状态改变回调接口 |

### 3.1.9注销服务器连接状态监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

注销服务连接状态监听

#### 调用场景

当要销毁监听状态时调用

#### 方法原型

CMVoIPManager.getInstance().removeServerConnectListener(VoIPServerConnectListener serverConnectListener);

#### 调用实例

|  |
| --- |
| @Override public void onDestroy() {  super.onDestroy();  CMVoIPManager.getInstance().removeServerConnectListener(this); } |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| serverConnectListener | VoIPServerConnectListener | 是 | 监听连接状态回调接口 |

### 3.1.10注册codec观察者

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

观察解码器参数、视频流参数、解码器变化等。

#### 调用场景

需要显示解码器参数、视频流参数及解码器变化时调用。

#### 方法原型

CMVoIPManager.getInstance().registerVideoCodecObserver(VoIPCodecObserver observer);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().registerVideoCodecObserver(new VoIPCodecObserver() {  @Override  public void incomingRate(int session, int channel, int framerate, int bitrate) {   }   @Override  public void DecoderTiming(int decode\_ms, int max\_decode\_ms, int current\_delay\_ms, int target\_delay\_ms, int jitter\_buffer\_ms, int min\_playout\_delay\_ms, int render\_delay\_ms) {   }   @Override  public void incomingCodecChanged(int session, int Channel, int width, int height) {   }   @Override  public void requestNewKeyFrame(int session, int Channel) {   }   @Override  public void outgoingRate(int session, int channel, int framerate, int bitrate) {   }   @Override  public void suspendChange(int session, int channel, boolean is\_suspended) {   }   @Override  public void outgoingCodecChanged(int session, int channel, int width, int height) {   }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| Observer | VoIPCodecObserver | 是 | 观察者 |

#### 回调参数说明

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 描述 |
| session | int | 会话 |
| channel | int | 会话的通道 |
| framerate | int | 帧率 |
| bitrate | int | 码率 |
| decode\_ms | int | 解码时间 |
| max\_decode\_ms | int | 最大解码时间 |
| current\_delay\_ms | int | 当前延迟 |
| target\_delay\_ms | int | 目标延迟 |
| jitter\_buffer\_ms | int | 接收缓存延迟 |
| min\_playout\_delay\_ms | int | 最小输出延迟 |
| render\_delay\_ms | int | 渲染延迟 |
| width | int | 宽 |
| height | int | 高 |

### 3.1.11注销codec观察者

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

注销解码器、视频流、解码器变化的观察者。

#### 调用场景

使用完毕时调用

#### 方法原型

CMVoIPManager.getInstance().deregisterVideoCodecObserver();

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().deregisterVideoCodecObserver(); |

### 3.1.12账号退出

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

退出voip账号（主动登出，或者账号被挤出）

#### 调用场景

需要退出登录的时候调用

#### 方法原型

CMVoIPManager.getInstance().doLogout();

#### 调用实例

|  |
| --- |
| //需要退出的时候 CMVoIPManager.getInstance().doLogout(); |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| false | boolean | 登出失败，没初始化 |
| true | boolean | 登出成功 |

### 3.1.13 SDK销毁

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

销毁sdk，注销通道，停止监听

#### 调用场景

在application被销毁的时候调用

#### 方法原型

CMVoIPManager.getInstance().destroy();

#### 调用实例

|  |
| --- |
| //application @Override public void onTerminate() {  // TODO Auto-generated method stub  super.onTerminate();  CMVoIPManager.getInstance().destroy(); } |

### 3.1.14检测摄像头是否可用

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

检测摄像头是否可用

#### 调用场景

在需要检测的时候调用

#### 方法原型

CMVoIPManager.getInstance().isCameraAvailable();

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().isCameraAvailable(); |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| false | boolean | 不可用 |
| true | boolean | 可用 |

### 3.1.15检测麦克风是否可用

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

检测麦克风是否可用

#### 调用场景

在需要检测的时候调用

#### 方法原型

CMVoIPManager.getInstance().isMicAvailable();

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().isMicAvailable(); |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| false | boolean | 不可用 |
| true | boolean | 可用 |

### 3.1.16设置视频图像分辨率

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

设置视频图像分辨率

#### 调用场景

在需要设置视频图像分辨率的时候调用

#### 方法原型

CMVoIPManager.getInstance().setVideoResolutionType(int nIndex, int callType);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setVideoResolutionType(1, VoIPConstant.CallType.CALLTYPE\_1V1\_VIDEO); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| nIndex | int | 是 | 要切换清晰度的索引，0,1,2,3 |
| callType | int | 是 | 1v1视频：  VoIPConstant.CallType.CALLTYPE*1V1*VIDE  视频会议：VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| -2 | int | 可用 |
| -1 | int | 不可用 |
| 0 | int | 流畅 |
| 1 | int | 标清 |
| 2 | int | 高清 |
| 3 | int | 超清 |

### 3.1.17注册底层开启异常监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

监听视频通话摄像头状态

#### 调用场景

在通话界面开启时增加监听

#### 方法原型

CMVoIPManager.getInstance().registerCameraStatusListener

(VoIPCameraStatusCallBack mCameraStatusListener);

#### 调用实例

|  |
| --- |
| CMVoIPManager.*getInstance*().registerCameraStatusListener(new VoIPCameraStatusCallBack() {  @Override  public void onOpenCameraError() {  //开启失败处理  } }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| mCameraStatusListener | VoIPCameraStatusCallBack | 是 | 开启状态回调接口 |

### 3.1.18注销底层开启异常监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

注销视频通话摄像头状态监听

#### 调用场景

在通话界面销毁时调用

#### 方法原型

CMVoIPManager.getInstance().unregisterCameraStatusListener

();

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().unregisterCameraStatusListener(); |

### 3.1.19获取摄像头状态（通话过程中使用）

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

获取视频通话摄像头状态

#### 调用场景

在通话中获取

#### 方法原型

CMVoIPManager.getInstance().getCameraStatus(int session);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().getCameraStatus(0); |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| 0 | int | 成功 |
| 其它 | int | 失败 |

3.2 1v1通话模块

### 3.2.1 拨打音频/视频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

当用户登录成功以后，需要拨打电话时调用该接口。拨打电话后，电话状态发生改变会走addCallStateListener中注册状态监听的回调，需先加入电话状态的监听才能得到状态的回调通知。电话状态监听状态里onMakeCallFailed(int status)会返回所有拨打失败的原因。

1v1音频中设置入参callType为0 VoIP.CallType.CALLTYPE\_1V1\_AUDIO

1v1视频中设置入参callType为1 VoIP.CallType.CALLTYPE\_1V1\_VIDEO

#### 调用场景

当登录成功后，需发起呼叫时调用

#### 方法原型

CMVoIPManager.getInstance().callOut(String call\_number, int callType, VoIPDialCallBack mDialCallBack);

#### 调用实例

|  |
| --- |
| VoIPDialCallBack mDialCallBack; mDialCallBack = new VoIPDialCallBack() {  @Override  public void onHandleDialSuccess() {   }   @Override  public void onHandleDialError(int errorCode) {  switch(errorCode){  case VoIPDialCallBack.VOIP\_DIAL\_NUMBER\_ERROR:  break;  case VoIPDialCallBack.VOIP\_DIAL\_FAIL:  break;  }  }  }; CMVoIPManager.getInstance().callOut(call\_number,callType,mDialCallBack); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| call\_number | String | 是 | 拨打的账号名 |
| callType | int | 是 | 通话类型（0：音频， 1：视频） |
| mDialCallBack | VoIPDialCallBack | 是 | 拨号结果的回调，不能为null |

#### 异常处理

1. onHandleDialError回调errorCode：

|  |  |  |
| --- | --- | --- |
| Error | Code | 描述 |
| VOIP\_DIAL\_NUMBER\_ERROR | 1 | 号码有误 |
| VOIP\_DIAL\_FAIL | 2 | 拨号失败 |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| 0 | int | 拨打成功 |
| -2 | int | 未登录 |
| -3 | int | 底层注册失败 |
| -1 | int | 号码有误或者初始化失败 |

### 3.2.2 接听电话

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在登录成功后有电话呼入的情况下，用户点击接听按键调用该接口

#### 调用场景

在用户登录成功后有电话呼入的情况下调用该接口，进入通话

#### 方法原型

CMVoIPManager.getInstance().pickUpCall(int callSession);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().pickUpCall(0); //传0即可 |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callSession | int | 是 | 一个session代表一个会话，默认session为0 |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| 0 | int | 成功 |
| -1 | int | 初始化失败 |
| 其它 | int | 失败 |

### 3.2.3 挂断电话

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

挂断电话

#### 调用场景

当用户点击挂断电话按钮或者需要对电话状态进行复位的时候使用，当挂断之后需要对电话状态进行查询来判断挂断是否成功

#### 方法原型

CMVoIPManager.getInstance().hangUpCall(int callSession);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().hangUpCall(0); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callSession | int | 是 | 一个session代表一个会话，默认session为0 |

### 3.2.4 设置所有会话输入静音

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在通话中，将自己的发音屏蔽，使对方听不到自己的语音。

#### 调用场景

在正在通话界面，设置静音功能。

#### 方法原型

CMVoIPManager.getInstance().setInputMute(boolean callMute);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setInputMute(true); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callMute | boolean | 是 | True为静音，false为正常 |

### 3.2.5 设置单个会话输入静音

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在通话中，将某个会话发音屏蔽，使对方听不到自己的语音。

#### 调用场景

在正在通话界面，设置静音功能。

#### 方法原型

CMVoIPManager.getInstance().setInputMute(int session, boolean isMute);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setInputMute(1, true); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callMute | boolean | 是 | true为静音，false为正常 |
| session | int | 是 | 一个session代表一个会话 |

### 3.2.6 设置所有会话输出静音

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在通话中，将远端的发音屏蔽，本地不会发出声音

#### 调用场景

在正在通话界面，设置输出静音功能.

#### 方法原型

CMVoIPManager.getInstance().setOutputMute(boolean callMute);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setOutputMute(true); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callMute | boolean | 是 | true为静音，false为正常 |

### 3.2.7 设置单个会话输出静音

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在通话中，将远端的某个会话发音屏蔽，本地不会发出声音

#### 调用场景

在正在通话界面，设置输出静音功能。

#### 方法原型

CMVoIPManager.getInstance().setOutputMute(int session, boolean isMute);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setOutputMute(0, true); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callMute | boolean | 是 | true为静音，false为正常 |
| session | int | 是 | 一个session代表一个会话 |

### 3.2.8 获取1v1本地预览视频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

当用户登录成功以后，在拨打1V1视频或者视频会议并进入通话状态后，调用此方法获取本地

#### 调用场景

当已经拨打1v1视频或者视频会议进入通话后，可调用该接口获取本地视频。

#### 方法原型

CMVoIPManager.getInstance().getLocalPreviewSurfaceView(int session)

#### 调用实例

|  |
| --- |
| //获得本地视频 SurfaceView sv\_local = CMVoIPManager.getInstance().getLocalPreviewSurfaceView(0); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |

#### 异常处理

1. 接口返回的view可能为null，需要处理。

### 3.2.9 获取远端渲染视频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

当用户登录成功以后，在拨打1V1视频或者视频会议并进入通话状态后调用该接口获取远端渲染的。在调用此接口前，先了解session，channel的概念：

session：会话。比如一个1V1通话，一个会议都是一个session。

channel：会话中的通道，一个session中可以有多个channel。比如召开一个6方会议，是一个session，但其中有6路视频，即有6个channel。

#### 调用场景

已经拨打1v1视频 或者 视频会议有其他与会人时，可调用该接口获取远端渲染视频。

#### 方法原型

CMVoIPManager.getInstance().getRemoteRenderSurfaceView(int session);

#### 调用实例

SurfaceView sv\_remote = CMVoIPManager.getInstance().getRemoteRenderSurfaceView(0);

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |

#### 异常处理

1. 接口返回的view可能为null，需要处理

### 3.2.10 切换前后摄像头

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

当用户登录成功以后，在拨打1V1视频并进入通话状态后调用该接口可切换前置和后置的摄像头。

#### 调用场景

当已经拨打1v1视频进入通话后，可调用该接口切换前后置摄像头。

#### 方法原型

CMVoIPManager.getInstance().switchCamera(int seesion, int cameraIndex);

#### 调用实例

|  |
| --- |
| //切换前置 CMVoIPManager.getInstance().switchCamera(0, 1); //切换完成后，需要重新获取本地预览视频(即调用3.2.8接口)  //切换后置 CMVoIPManager.getInstance().switchCamera(0, 0); //切换完成后，需要重新获取本地预览视频(即调用3.2.8接口) |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |
| cameraIndex | int | 是 | 0表示后置，1表示前置 |

#### 异常处理

1. cameraIndex不为0或者1时，会抛出异常

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| -1 | int | 初始化失败或不存在操作 |
| -2 | int | 摄像头不可用 |
| -3 | int | cameraIndex不为0或1 |
| 0 | int | 切换成功 |

### 3.2.11 特效

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

用户登录成功后，发起1v1视频可以选择特效。

#### 调用场景

当已经拨打1v1视频进入通话后，可调用该接口切换前后置摄像头。

#### 方法原型

CMVoIPManager.getInstance().setFacialSpecialEffectsMode(int session, int mode);

#### 调用实例

|  |
| --- |
| //无特效 CMVoIPManager.getInstance().setFacialSpecialEffectsMode(0, VoIPConstant.FacialEffect.MODE\_NONE); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |
| mode | int | 是 | VoIPConstant.FacialEffect.MODE\_GLASSES：眼镜特效；VoIPConstant.FacialEffect.MODE\_RABBIT：兔子耳朵特效；VoIPConstant.FacialEffect.MODE\_MASK：面具特效；VoIPConstant.FacialEffect.MODE\_NONE：无特效。 |

#### 使用说明

1. 把assets文件中的图片复制到工程中的assets文件夹中。
2. 在app启动后，加入以下代码，将文件写入路径

|  |
| --- |
| * private Context context;  public int fileAddToRootOfApp(){  context = getApplicationContext();  try {  if(fileIsExists(context.getFilesDir()+"/glass.png") && fileIsExists(context.getFilesDir()+"/rabbit.png") && fileIsExists(context.getFilesDir()+"/mask.png")) {  return -1;  }  InputStream glassInput = context.getResources().getAssets().open("glass.png");  InputStream rabbitInput = context.getResources().getAssets().open("rabbit.png");  InputStream maskInput = context.getResources().getAssets().open("mask.png");  FileOutputStream glassOutput = new FileOutputStream(context.getFilesDir()+"/glass.png");  FileOutputStream rabbitOutput = new FileOutputStream(context.getFilesDir()+"/rabbit.png");  FileOutputStream maskOutput = new FileOutputStream(context.getFilesDir()+"/mask.png");  byte[] buffer = new byte[1024];  int count = 0;  while((count = glassInput.read(buffer))>0){  glassOutput.write(buffer,0,count);  }  count = 0;  while((count = rabbitInput.read(buffer))>0){  rabbitOutput.write(buffer,0,count);  }  count = 0;  while((count = maskInput.read(buffer))>0){  maskOutput.write(buffer,0,count);  }  glassOutput.flush();  glassOutput.close();  glassInput.close();  rabbitOutput.flush();  rabbitOutput.close();  rabbitInput.close();  maskOutput.flush();  maskOutput.close();  maskInput.close();  buffer = null;   } catch (IOException e) {  e.printStackTrace();  }  return 0;  }  public boolean fileIsExists(String filePath){  File f=new File(filePath);  if(f.exists()){  return true;  }  return false; } * private Context context;  public int fileAddToRootOfApp(){  context = getApplicationContext();  try {  if(fileIsExists(context.getFilesDir()+"/glass.png") && fileIsExists(context.getFilesDir()+"/rabbit.png") && fileIsExists(context.getFilesDir()+"/mask.png")) {  return -1;  }  InputStream glassInput = context.getResources().getAssets().open("glass.png");  InputStream rabbitInput = context.getResources().getAssets().open("rabbit.png");  InputStream maskInput = context.getResources().getAssets().open("mask.png");  FileOutputStream glassOutput = new FileOutputStream(context.getFilesDir()+"/glass.png");  FileOutputStream rabbitOutput = new FileOutputStream(context.getFilesDir()+"/rabbit.png");  FileOutputStream maskOutput = new FileOutputStream(context.getFilesDir()+"/mask.png");  byte[] buffer = new byte[1024];  int count = 0;  while((count = glassInput.read(buffer))>0){  glassOutput.write(buffer,0,count);  }  count = 0;  while((count = rabbitInput.read(buffer))>0){  rabbitOutput.write(buffer,0,count);  }  count = 0;  while((count = maskInput.read(buffer))>0){  maskOutput.write(buffer,0,count);  }  glassOutput.flush();  glassOutput.close();  glassInput.close();  rabbitOutput.flush();  rabbitOutput.close();  rabbitInput.close();  maskOutput.flush();  maskOutput.close();  maskInput.close();  buffer = null;   } catch (IOException e) {  e.printStackTrace();  }  return 0;  }  public boolean fileIsExists(String filePath){  File f=new File(filePath);  if(f.exists()){  return true;  }  return false; } |

1. 在合适位置调用fileAddToRootOfApp()即可。

### 3.2.12 美颜

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

发起1v1视频 或者 视频会议中可以选择开启美颜效果。

#### 调用场景

用户登录成功后，发起1v1视频 或者 视频会议中可以选择开启美颜效果。

#### 方法原型

CMVoIPManager.getInstance().setFaceEnhanceParams(int session, boolean isStartSmoothFace, float degreeOfFaceSmooth);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setFaceEnhanceParams(0, true, 1); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |
| isStartSmoothFace | boolean | 是 | True开启，false关闭 |
| degreeOfFaceSmooth | Float | 是 | 美颜级别（目前可以随便设置） |

### 3.2.13 音频切换到视频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

发起1v1音频通话，切换到视频通话。

#### 调用场景

用户登录成功后，发起1v1音频，可以切换成视频通话。发起请求后，对方会在电话状态的回调onReceiveCallSwitch接收到切换的消息(3.1.6)，此时用户可以自己处理是否同意切换操作。当对方同意后会收到onCallReBuildResult消息(3.1.6)，其中带有通话类型。

#### 方法原型

CMVoIPManager.getInstance().switchToVideo(int session);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().switchToVideo(0); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |

### 3.2.14 视频切换到音频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

1v1视频通话中，切换到音频通话。

#### 调用场景

用户登录成功后，发起1v1视频，可以切换成音频通话。发起请求后，无需对方同意，会直接收到onCallReBuildResult消息(3.1.6)，其中带有通话类型。

#### 方法原型

CMVoIPManager.getInstance().switchToAudio(int session);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().switchToAudio(0); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |

### 3.2.15 同意切换到视频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

1v1通话中，对方发送切换视频的请求，同意切换时，调用此接口。

#### 调用场景

用户登录成功后，发起1v1音频，对方发起切换视频请求，用户会在onReceiveCallSwitch收到消息(3.1.6)，用户如果同意切换就调用此接口。切换成功后会在onCallReBuildResult收到切换的结果，此时app更新上层逻辑。如果是视频切换成音频无需用户确认，直接会在onCallReBuildResult收到切换成音频的消息。

#### 方法原型

CMVoIPManager.getInstance().agreeToVideo(int session);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().agreeToVideo(0); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |

### 3.2.16 拒绝切换到视频

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

1v1通话中，对方发送切换视频的请求，拒绝切换调用此接口。

#### 调用场景

用户登录成功后，发起1v1音频，对方发起切换视频请求，用户会在onReceiveCallSwitch收到消息(3.1.6)，用户如果拒绝切换就调用此接口。

#### 方法原型

CMVoIPManager.getInstance().refuseToVideo(int session);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().refuseToVideo(0); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| session | int | 是 | 一个session代表一个会话 |

3.3 会议模块

### 3.3.1 发起即时会议

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

调用此接口可以创建即时语音、视频会议，创建成功后，在线用户会收到会议的来电。

#### 调用场景

在登录之后，当需要发起即时会议的时候进行调用

#### 方法原型

CMVoIPManager.getInstance().createConference(VoIPInstantConferenceCallBack callBacke, int callType, String subject, List<String> memberList);

#### 调用实例

|  |
| --- |
| VoIPInstantConferenceCallBack callBack = new VoIPInstantConferenceCallBack() {  @Override  public void onSuccess(String conferenceNumber, String conferenceId, int csTime) {  //conferenceNumber 会议号  //会议id  //cs剩余时间  }   @Override  public void onError(String errorString) {  if (errorString.contains("-1003")) {  //无法创建会议，请先结束上一次会再创建新的议  } else if (errorString.contains("-1004")) {  //会议成员达到最大数  } else {  //创建会议失败  }  }  };  CMVoIPManager.getInstance().createConference(callBack, callType, subject, mMemberList);  //音频 //VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_AUDIO //视频 //VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPInstantConferenceCallBack | 是 | 创建会议的回调 |
| callType | Int | 是 | VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_AUDIO：音频会议；VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO：视频会议 |
| subject | String | 是 | 会议主题（必填） |
| mMemberList | List | 是 | 会议成员号码列表 |

#### 异常处理

1. callBack为空会抛出空指针异常。
2. subject会议主题为null会抛出异常。
3. mMemberList列表为空时创建失败。
4. 会议成员号码不正常时会抛出异常。
5. onError回调接口返回描述：

|  |  |
| --- | --- |
| ErrorString | 描述 |
| 字符串包含“-1003“ | 无法创建会议，请先结束上一次会再创建新的会议 |
| 字符串包含“-1004” | 会议成员达到最大数 |
| 创建会议人输入有误 | 创建会议人输入有误 |
| 会议参会人空，创建失败 | 为会议参会人空，创建失败 |
| 请填写会议主题，长度不超过40个字 | 请填写会议主题，长度不超过40个字 |
| 会议创建失败，无返回值 | 会议创建失败，无返回值 |
| 会议创建失败,接口调用失败 | 会议创建失败,接口调用失败 |
| 会议创建失败，没有创建者 | 会议创建失败，没有创建者 |
| 会议创建失败，字符串解析出错 | 会议创建失败，字符串解析出错 |

### 3.3.2 发起预约会议

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

调用此接口可以创建预约语音会议，创建成功后，在线用户会收到会议的来电。

#### 调用场景

在登录之后，当需要发起预约会议的时候进行调用。

#### 方法原型

CMVoIPManager.getInstance().createReservationConference(VoIPReservationConferenceCallBack callBack, int callType, String subject, List<String> memberList,  
 Map<String, Integer> attributes, int minutes)

#### 调用实例

|  |
| --- |
| VoIPReservationConferenceCallBack callback = new VoIPReservationConferenceCallBack() {  @Override  public void onSuccess(int confNum, String confId) {    }   @Override  public void onError(String errorString) {   }  };  Map<String, Integer> attributes = new HashMap<>(); attributes.put(VoIPConstant.CONFERENCE\_START\_YEAR, 2010); attributes.put(VoIPConstant.CONFERENCE\_START\_MONTH, 12); attributes.put(VoIPConstant.CONFERENCE\_START\_DAY, 12); attributes.put(VoIPConstant.CONFERENCE\_START\_HOUR, 12); attributes.put(VoIPConstant.CONFERENCE\_START\_MINUTE, 12);  VoIPManag-er.getInstance().createReservationConference(callBack, callType, subject, memberList, attributes, minutes) |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPReservationConferenceCallBack | 是 | 预约会议的回调 |
| callType | Int | 是 | VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_AUDIO：音频会议；VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO：视频会议 |
| subject | String | 是 | 会议主题（必填） |
| mMemberList | List | 是 | 会议成员号码列表 |
| attributes | Map | 是 | 预约时间 |
| minutes | int | 是 | 会议持续分钟 |

#### 异常处理

1. callBack为空会抛出空指针异常
2. subject会议主题为null会抛出异常
3. mMemberList列表为空时创建失败
4. 会议成员号码不正常时会抛出异常。
5. attributes为null或size为1或年大于2037，则回调onError()错误
6. minutes传递的值小于等于０或大于300则回调onError()错误

### 3.3.3 主动入会

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

当用户登录成功以后，需要根据会议号码主动进入该会议时调用该接口。

#### 调用场景

当登录成功后，需主动入会时调用该接口。

#### 方法原型

CMVoIPManager.getInstance().joinConference(String number, int callType, VoIPDialCallBack VoIPDialCallBack);

#### 调用实例

|  |
| --- |
| VoIPDialCallBack mDialCallBack = new VoIPDialCallBack() {  @Override  public void onHandleDialSuccess() {  //主动入会成功  }   @Override  public void onHandleDialError(int errorCode) {  //主动入会失败  }  };  CMVoIPManager.getInstance().joinConference(call\_number, callType, mDialCallBack); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| call\_number | String | 是 | 会议号 |
| callType | Int | 是 | VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_AUDIO：音频会议；VoIPConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO：视频会议 |
| mDialCallBack | VoIPDialCallBack | 是 | 拨打结果的回调 |

#### 异常处理

1. 同拨打电话

### 3.3.4 邀请会议成员

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在会议通话过程中调用该接口邀请指定的成员到该指定的会议中

#### 调用场景

会议进行时，需要邀请指定的成员入会，此时调用该接口

#### 方法原型

CMVoIPManager.getInstance().inviteConferenceMembers(long conferenceNumber, List<String> memberList, CallBack callBack)

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().inviteConferenceMembers(Long.parseLong(conferenceNum), memberList, new VoIP.CallBack() {  @Override  public void onSuccess() {    }   @Override  public void onFailed(int code, String errorString) {   }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIP.CallBack | 是 | 常规回调 |
| memberList | List | 是 | 邀请的成员 |
| conferenceNum | long | 是 | 会议号 |

#### 异常处理

1. callBack为空会抛出异常
2. onFailed回调返回描述:

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -1000 |  | 参数有误 |
| -1002 | 邀请会议成员失败,指定的会议不存在 | 邀请会议成员失败,指定的会议不存在 |
| -1004 | 邀请会议成员失败,会议成员人数达到最大值 | 邀请会议成员失败,会议成员人数达到最大值 |

### 3.3.5 剔除会议成员

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在语音会议通话过程中调用该接口剔除指定的成员到该会议中，只有会议创建者调用才有效。

#### 调用场景

会议进行时，需要剔除本会议中指定的成员，此时调用该接口

#### 方法原型

CMVoIPManager.getInstance().kickOutConferenceMember(long conferenceNumber, String member, CallBack callBack)

#### 调用实例

|  |
| --- |
| long num = Long.parseLong(conferNum); CMVoIPManager.getInstance().kickOutConferenceMember(num, memberId, new VoIP.CallBack() {  @Override  public void onSuccess() {  //此人从与会人列表中删除，不能再主动入会；    }   @Override  public void onFailed(int code, String errorString) {    }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIP.CallBack | 是 | 常规回调 |
| memberId | String | 是 | 要剔除会议成员的账户名 |
| num | long | 是 | 会议号 |

#### 异常处理

1. callBack为空时会抛出异常。
2. memberId用户名不合法会抛出异常。
3. onFailed回调返回描述：

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -1000 | 请输入要剔除会议成员的用户名 | 请输入要剔除会议成员的用户名 |
| 1 | 剔除会议成员失败，无返回值 | 剔除会议成员失败，无返回值 |
| 1000 | 剔除会议成员失败，参数校检失败 | 剔除会议成员失败，参数校检失败 |
| -1003 | 剔除会议成员失败，只有会议创建者才有权限 | 剔除会议成员失败，只有会议创建者才有权限 |

### 3.3.6 注册会议成员信息列表监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

会议发起或者加入成功时，接口会返回会场信息及成员信息数据；当数据有更新时也会返回新的成员列表信息。

#### 调用场景

在会议界面开启时，加入监听。

#### 方法原型

CMVoIPManager.getInstace().addConferenceInfoListener(String userName, int callType, VoIPConferenceInfoCallBack callback);

#### 调用实例

|  |
| --- |
| VoIPConferenceInfoCallBack callBack = new VoIPConferenceInfoCallBack() {  @Override  public void onResponseConferenceData(String conferNum, String creator, String conferName, int isLock, int callType, String time) {  //conferNum 会议号  //creator 创建者id  //conferName 会议名  //isLock 会议锁定状态  //callType 电话类型  //time 开启时间  }   @Override  public void onResponseMemberList(List<MediaMember> mediaMembers) {  //会议成员信息列表  for (int i = 0; i < mediaMembers.size(); i++) {  mediaMembers.get(i).getUserName());//成员名  mediaMembers.get(i).getCallStatus());//接通状态 (见3.状态说明)  mediaMembers.get(i).getMute());//禁言状态 (0, 1)  mediaMembers.get(i).getCsStatus());//cs接入状态 (0, 1)  mediaMembers.get(i).getUserID());//成员id(用户名 + AppKey)  mediaMembers.get(i).getVideoView());//成员的视频view(音频无)  }    }   @Override  public void onResponseError(String errorString) {   }  };   CMVoIPManager.getInstance().addConferenceInfoListener(userName, callType, callBack); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPConferenceInfoCallBack | 是 | 会议成员信息回调 |
| callType | int | 是 | 会议类型 |
| userName | String | 是 | 用户登录名 |

#### 回调参数说明

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 描述 |
| conferNum | String | 会议号 |
| Creator | String | 创建者Id |
| conferName | String | 会议名 |
| isLock | int | 会议锁定状态(0, 1) |
| callType | int | 电话类型 |

#### 异常处理

1. callBack为空时会抛出异常

### 3.3.7 注销会议成员信息列表监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

移除会议成员信息的监听

#### 调用场景

在Activity销毁时，移除对会议成员信息的监听

#### 方法原型

CMVoIPManager.getInstance().removeConferenceInfoListener();

#### 调用实例

|  |
| --- |
| @Override protected void onDestroy() {  // TODO Auto-generated method stub  super.onDestroy();  CMVoIPManager.getInstance().removeConferenceInfoListener(); } |

### 3.3.8 注册会议状态更新监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

监听会议状态变化，会议结束，会议中踢除与会人，会议对某个与会人禁言。

#### 调用场景

可以在服务中voip登录成功时加入监听，也可在需要的界面加入监听。

#### 方法原型

CMVoIPManager.getInstance().addConferenceStateListener(VoIPConferenceStateCallBack callback);

#### 调用实例

|  |
| --- |
| VoIPConferenceStateCallBack confStateCallback = new VoIPConferenceStateCallBack() {  @Override  public void onConferenceClosed(String conferNum) {  //会议关闭  }  @Override  public void onConferenceKicked(String conferNum, String kicked) {  //会议剔除  //kicked被剔除的成员名  }  @Override  public void onConferenceMuted(String conferNum, boolean mute) {   //会议禁言(当自己被禁言时会收到)  }  @Override  public void onConferenceUpdated(String conferNum) {    }  };  CMVoIPManager.getInstance().doLogin(map, new VoIPLoginCallBack() {  @Override  public void onLoginSuccess() {  CMVoIPManager.getInstance().addConferenceStateListener(confStateCallback);   }  @Override  public void onLoginFailed(int code) {    }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPConferenceStateCallBack | 是 | 会议状态更新回调 |

### 3.3.9 注销会议状态更新监听

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

注销会议状态变化监听。

#### 调用场景

可以在服务或者界面的onDestroy中注销监听。

#### 方法原型

CMVoIPManager.getInstance().removeConferenceStateListener(VoIPConferenceStateCallBack callback);

#### 调用实例

|  |
| --- |
| @Override public void onDestroy() {  super.onDestroy();  CMVoIPManager.getInstance().removeConferenceStateListener(confStateCallback); } |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPConferenceStateCallBack | 是 | 会议状态更新回调 |

### 3.3.10 关闭会议

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

用于结束音视频会议接口。(仅会议创建者调用)

#### 调用场景

在音视频会议中，主持人挂机时，需要调用些接口来结束会议。结束成功后，其它与会人会接收到电话状态回调中的onCallReleased，以结束会议界面

#### 方法原型

CMVoIPManager.getInstance().closeConference(String creatorUserId, String accessCode, CallBack callBack)

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().closeConference(userName + VoIPConfig.appkey, call\_number, new VoIP.CallBack() {  @Override  public void onSuccess() {    }   @Override  public void onFailed(int code, String errorString) {    }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIP.CallBack | 是 | 常规回调 |
| accessCode | String | 是 | 会议号 |
| creatorUserId | String | 是 | 创建者登录名+appKey |

#### 异常处理

1. callBack为空时会抛出异常
2. onFailed回调返回描述：

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -2001 | 会议号为空 | 会议号为空 |
| -2004 | 创建者id为空 | 创建者id为空 |
| -2005 | 底层的token为空 | 底层的token为空 |
| 100 | 会议结束失败，无返回值 | 会议结束失败，无返回值 |
| -2 | Error | Error |

### 3.3.11 通过会议号请求会议信息

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

用于输入会议号，来获取会议的基本信息

#### 调用场景

加入会议可调用请求出会议类型和其他信息加入会议。

#### 方法原型

CMVoIPManager.getInstance().requestMemberConferenceInfo(String userId, String creatorUserId, VoIPConferenceDataCallBack callBack)

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().requestMemberConferenceInfo(userName + VoIPConfig.appkey, call\_number + VoIPConfig.appkey, new VoIPConferenceDataCallBack() {  @Override  public void onSuccess(ConferenceData data) {  //data.getConferenceNum() 会议号  //data.getConferenceName() 会议名  //data.getCreator() 会议创建者id  //data.getStartTime() 会议开始时间  //data.getLock() 会议锁状态 0未锁，1锁定  //data.getConfType() 会议类型 0音频会议，1视频会议  }   @Override  public void onFailed(int code, String error) {    }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPConferenceDataCallBack | 是 | 会议基本信息的回调 |
| creatorUserId | String | 是 | 创建者id |
| userId | String | 是 | 登录名id |

#### 异常处理

1. callBack为空时会抛出异常。
2. onFailed回调返回描述:

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -2003 | 用户名id为空 | 用户名id为空 |
| -2004 | 创建者id为空 | 创建者id为空 |
| -2005 | 底层的token为空 | 底层的token为空 |
| 100 | 查询无返回值 | 查询无返回值 |
| -1002 | 会议不存在 | 会议不存在 |

### 3.3.12 获取我的会议列表

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

用于请求自己的会议列表。

#### 调用场景

在我的会议列表界面中调用。

#### 方法原型

CMVoIPManager.getInstance().requestConferenceList(String userId, VoIPConferenceDataListCallBack callBack);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().requestConferenceList(username + VoIPConfig.appkey, new VoIPConferenceDataListCallBack() {  @Override  public void onSuccess(List<ConferenceData> mList) {  //mList带有会议基本信息的列表  }   @Override  public void onFailed(int code, String error) {   }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIPConferenceDataListCallBack | 是 | 会议基本信息列表的回调 |
| userId | String | 是 | 登录名id |

#### 异常处理

1. callBack为空时会抛出异常
2. onFailed回调返回描述：

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -2003 | 用户名id为空 | 用户名id为空 |
| -2005 | 底层的token为空 | 底层的token为空 |
| 100 | 获取会议列表失败，无返回值 | 获取会议列表失败，无返回值 |
| 其它 | Error | Error |

### 3.3.13 锁定会议

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

锁定会议，会议以外的成员无法主动加入会议。(仅创建者调用)

#### 调用场景

当需要阻止会议以外的成员主动入会时，可调用此接口

#### 方法原型

CMVoIPManager.getInstance().lockConference(String creatorUserId, String accessCode, String lock, CallBack callBack)

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().lockConference(creatorUserId, conferNum, isLock, new VoIP.CallBack() {  @Override  public void onSuccess() {    }   @Override  public void onFailed(int code, String errorString) {   }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIP.CallBack | 是 | 常规回调 |
| creatorUserId | String | 是 | 创建者用户名id |
| accessCode | String | 是 | 会议号 |
| lock | String | 是 | 0：解锁，1：锁 |

#### 异常处理

1. callBack为空时会抛出异常
2. onFailed回调返回描述：

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -2001 | 会议号为空 | 会议号为空 |
| -2004 | 创建者Id为空 | 创建者Id为空 |
| -2005 | 底层的token为空 | 底层的token为空 |
| 100 | 会议锁定失败，无返回值 | 会议锁定失败，无返回值 |
| 其它 | Error | Error |

### 3.3.14 会议成员禁言

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

主持人对会议中的成员禁言。禁言后无法听到该成员的讲话。(仅创建者调用)

#### 调用场景

主持人需要对某成员禁言时，可调用此接口。

#### 方法原型

CMVoIPManager.getInstance().setConferenceMemberMute(String memberUserId, String accessCode, String creatorUserId, String mute, CallBack callBack);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setConferenceMemberMute(memberUserId, conferNum, creatorUserId, mute, new VoIP.CallBack() {  @Override  public void onSuccess() {    }   @Override  public void onFailed(int code, String errorString) {  }  }); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| callBack | VoIP.CallBack | 是 | 常规回调 |
| creatorUserId | String | 是 | 创建者用户名id |
| accessCode | String | 是 | 会议号 |
| memberUserId | String | 是 | 要禁言的成员id |
| Mute | String | 是 | 1：禁言，0：未禁言 |

#### 异常处理

1. callBack为空时会抛出异常
2. onFailed回调返回描述

|  |  |  |
| --- | --- | --- |
| code | ErrorString | 描述 |
| -2001 | 会议号为空 | 会议号为空 |
| -2002 | 成员id为空 | 成员id为空 |
| -2004 | 创建者Id为空 | 创建者Id为空 |
| -2005 | 底层的token为空 | 底层的token为空 |
| 100 | 禁言失败 | 禁言失败 |
| 其它 | Error | Error |

### 3.3.15 变声

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

在音频会议中，可以将自己的声音变为男声、女声、童声。默认为自己原声。

#### 调用场景

成员需要变声时调用

#### 方法原型

CMVoIPManager.getInstance().setVoiceChangeStatus(boolean enable, int mode);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setVoiceChangeStatus(true, VoIPConstant.VoiceMode.CHILD); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| Enable | boolean | 是 | True:变声，false:不变声 |
| mode | int | 是 | VoIPConstant.VoiceMode.NORMAL：原声；VoIPConstant.VoiceMode.MAN：男声；VoIPConstant.VoiceMode.WOMAN：女声；VoIPConstant.VoiceMode.CHILD：童声 |

#### 接口返回值

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| -2 | int | 参数无效 |
| -1 | int | 初始化失败 |
| 其它 | int | 成功 |

3.4 TV相关模块

### 3.4.1 获取摄像头个数

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

获取摄像头个数。

#### 调用场景

在TV需要获取个数时调用。

#### 方法原型

CMVoIPManager.getInstance().getCameraNum();

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().getCameraNum(); |

#### 接口返回

返回摄像头个数

### 3.4.2 刷新摄像头信息

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

刷新摄像头信息

#### 调用场景

摄像头异常或插拔时需要刷新信息。

#### 方法原型

CMVoIPManager.getInstance().refreshCameraInfo(int session);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().refreshCameraInfo(0); |

### 3.4.3 关闭摄像头采集（TV 1v1视频使用）

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

1v1视频通话关闭采集。

#### 调用场景

需要关闭摄像头采集时调用。

#### 方法原型

CMVoIPManager.getInstance().StopTVLocalVideoAndInfoRemote()

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().StopTvLocalVideoAndInfoRemote(); |

### 3.4.4 打开摄像头采集（TV 1v1视频使用）

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

打开摄像头采集。

#### 调用场景

关闭后，再次打开摄像头采集调用。

#### 方法原型

CMVoIPManager.getInstance().StarTVLocalVideoAndInfoRemote();

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().StartTVLocalVideoAndInfoRemote(); |

3.5 IMS相关模块

### 3.5.1 注册IMS服务

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

注册IMS服务。

#### 调用场景

登录IMS账号时使用。

#### 方法原型

CMVoIPManager.getInstance().register2Ims (String serverUrl,

String serverAddr,

Int port,

String username,

String displayName,

String authName,

String password);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().register2Ims (domain, serverAddr, port, username, imsNum, imsAccount, password**)** ; |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| domain | String | 是 | 服务器url |
| serverAddr | String | 是 | 服务器地址 |
| port | String | 是 | 服务器端口 |
| userName | String | 是 | 真实用户名 |
| imsNum | String | 是 | 显示用户名 |
| imsAccount | String | 是 | 鉴权用户名（username@serverUrl） |
| password | String | 是 | 注册密码 |

#### 接口返回

|  |  |  |
| --- | --- | --- |
| 返回值 | 类型 | 描述 |
| 0 | int | 成功 |
| 其它 | int | 失败 |

### 3.5.2 设置转换IMS服务

#### 所属类

com.mobile.voip.sdk.api.CMVoIPManager;

#### 接口说明

设置电话是否转接到IMS服务

#### 调用场景

需要IMS功能转接时调用

#### 方法原型

CMVoIPManager.getInstance().setCallForwardingFlag(boolean enable);

#### 调用实例

|  |
| --- |
| CMVoIPManager.getInstance().setCallForwardingFlag(true); |

#### 入参说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 类型 | 必选 | 描述 |
| true | boolean | 是 | 转接 |
| False | boolean | 是 | 不转接 |

四、状态说明

4.1 电话接通状态码

1 E\_CALL\_INVITE, 呼叫中 （呼叫中）

2 E\_CALL\_RING, 振铃中 （呼叫中）

3 E\_CALL\_ANSWER, 应答、在线 （会议中）

4 E\_CALL\_FAIL, 呼叫失败 （未接听）

5 E\_CALL\_USER\_BUSY, 被叫忙 （未接听）

6 E\_CALL\_404\_NOUSER, 未注册 （未注册）

7 E\_CALL\_NO\_ANSWER, 无应答 （未接听）

8 E\_CALL\_NOT\_LOGIN, 未登录 （未登录）

9 E\_CALL\_REJECTED, 拒接 （挂断）

10 E\_CALL\_HANGUP, 挂机、离线 （挂断）

4.2 查询会议状态失败的错误码

// 0 ServerReasonNone,

// -1000 ServerReason, /\*\*失败通用错误码，参数校验失败 \*/

// -1001 ServerReason, /\*\*无空闲接入号\*/

// -1002 ServerReason, /\*\*指定的会议不存在\*/

// -1003 ServerReason, /\*\*没有权限\*/

// -1004 ServerReason, /\*\*会议成员达到最大值\*/

// -1005 ServerReason, /\*\*会议开始时间格式不正确\*/

// -1006 ServerReason, /\*\*预约会议起始时间大于当前时间\*/

// -1007 ClientReason, /\*\* 网络请求调用失败 \*/

// -1008 ServerReason, /\*\*接入号已存在\*/

// -1009 ClientReason, /\*\* Token错误\*/

五、附录

VoipSDK底层对会议部分进行了数据的同步，采取多播模式，减少了上层的数据处理。上层只需要实现多播模式的代理。

5.1 通话状态说明

VoIpConstant.CALL\_STATE\_RELEASED= 0; /\*通道空闲，仅该状态下能呼入呼出\*/  
 VoIpConstant.CALL\_STATE\_PROCEEDING= 1; /\* 主叫呼出等待 \*/  
 VoIpConstant.CALL\_STATE\_INCOMING= 2;/\* 来电\*/  
 VoIpConstant.CALL\_STATE\_ANSWERED = 3;/\* 通话中 \*/  
 VoIpConstant.CALL\_STATE\_MAKE\_FAILED = 4;/\*通话错误, 挂断 \*/  
 VoIpConstant.CALL\_STATE\_ALERTING = 5;/\* 等待被叫摘机 \*/  
 VoIpConstant.CALL\_STATE\_STOP\_ALERTING = 6;/\* 取消振铃 \*/

5.2 通话类型

VoIpConstant.CallType.CALLTYPE\_1V1\_AUDIO = 0 /\*1v1音频\*/  
 VoIpConstant.CallType.CALLTYPE\_1V1\_VIDEO = 1 /\*1v1视频\*/  
 VoIpConstant.CallType.CALLTYPE\_CONFERENCE\_AUDIO = 2 /\*音频会议\*/  
 VoIpConstant.CallType.CALLTYPE\_CONFERENCE\_VIDEO = 3 /\*视频会议\*/

VoIpConstant.CallType.CALLTYPE\_1V1\_AUDIO\_IMS = 20 /\*IMS 1v1音频\*/ VoIpConstant.CallType.CALLTYPE\_1V1\_VIDEO\_IMS = 21 /\*IMS 1v1视频\*/

5.3 登录状态

VoIpConstant.VOIP\_LOGIN\_SUCCESS = 50;/\*登陆成功\*/  
 VoIpConstant.VOIP\_INIT\_FAIL = 51;/\*初始化失败\*/  
 VoIpConstant.VOIP\_LOGIN\_INVALID\_USER = 52;/\*错误的用户名或密码\*/  
 VoIpConstant.VOIP\_SERVER\_ERROR = 53; /\*服务器错误\*/  
 VoIpConstant.VOIP\_NET\_ERROR = 54; /\*网络错误\*/  
 VoIpConstant.VOIP\_LOGIN\_TIMEOUT = 55; /\*登录超时\*/  
 VoIpConstant.OIP\_LOGIN\_INVALID\_APPKY = 56; /\*appkey 无效 \*/  
 VoIpConstant.VOIP\_GET\_SERVER\_CONFIG\_ERROR = 57; /\*服务器配置信息出

5.4 特效模式

VoIpConstant.FacialEffect.MODE\_NONE = -1 /\*无\*/  
 VoIpConstant.FacialEffect.MODE\_GLASSES = 0 /\*眼镜\*/  
 VoIpConstant.FacialEffect.MODE\_RABBIT = 1 /\*兔耳朵\*/  
 VoIpConstant.FacialEffect.MODE\_MASK = 2 /\*面具\*/

5.5 变声模式

VoIpConstant.VoiceMode.NORMAL = 0 /\*正常\*/  
 VoIpConstant.VoiceMode.MAN = 1 /\*男声\*/  
 VoIpConstant.VoiceMode.WOMAN = 2 /\*女声\*/  
 VoIpConstant.VoiceMode.CHILD = 3 /\*童声\*/