**Android会议开发文档**

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订人** | **修订日期** | **修订内容** |
| 1.0.0 | 高翔 | 20180807 | * 初始版本 |
| 1.0.1 | 高翔 | 20181015 | * 修改插件依赖 |
| 2.0.13 | 李应(武汉UC事业部) | 20201022 | * 部分页面更改 |
| 5.6.1 | 李应 | 20201127 | * 更新视频页面部分 |

[1. 导入说明 2](#_Toc1706679174)

[1.1. 环境要求  2](#_Toc217804439)

[1.2. 导入sdk 3](#_Toc1327071785)

[1.3. 配置工程 4](#_Toc330332753)

[1.3.1. 权限配置 4](#_Toc657352176)

[1.3.2. SDK核心配置 6](#_Toc1462141864)

[2. 初始化SDK及登录 8](#_Toc562935627)

[2.1. 初始化SDK 8](#_Toc1593617954)

[2.2. 设置登录参数 10](#_Toc520907494)

[2.3. 设置通知回调监听 11](#_Toc1748906486)

[2.3.1. 设置登录回调监听 12](#_Toc1262633713)

[2.3.2. 设置音视频会议回调监听 13](#_Toc1798898384)

[2.4. 切换私有云环境 13](#_Toc1810357422)

[2.5. 登录SDK 14](#_Toc1128880858)

[3. 音视频会议开发说明 14](#_Toc82559161)

[3.1. 接口逻辑 14](#_Toc297382965)

[3.2. 业务流程 14](#_Toc921046186)

[3.3. 常见业务流程示意图 15](#_Toc961120526)

[3.3.1. 创建会议 15](#_Toc180687748)

[3.3.2. 加入会议 15](#_Toc277103778)

[3.3.3. 禁言成员 16](#_Toc1538650150)

[3.3.4. 关闭成员视频 16](#_Toc94993876)

[3.4. 创建视频视频会议准备工作 16](#_Toc981724211)

[3.4.1. 视频流显示 16](#_Toc721954376)

[3.5. 会议接口中常用对象 19](#_Toc604591882)

[3.5.1. ECAccountInfo(会议成员对象) 19](#_Toc1630626817)

[3.5.2. ECConferenceMemberInfo(成员信息对象) 20](#_Toc1906093952)

[3.5.3. ECConferenceInfo(会议信息对象) 21](#_Toc1707488965)

[3.6. 会议接口中常用枚举 23](#_Toc943059894)

[3.6.1. ECControlMediaAction 23](#_Toc1578323598)

[3.6.2. ECConferenceType 25](#_Toc1166703842)

[3.6.3. ECAccountType 25](#_Toc118291737)

[3.7. 会议主调业务接口 25](#_Toc1706850284)

[3.7.1. 创建会议或预约会议 25](#_Toc946166562)

[3.7.2. 邀请成员加入会议 27](#_Toc105001499)

[3.7.3. 加入会议 29](#_Toc1676119506)

[3.7.4. 拒绝加入 30](#_Toc1997539643)

[3.7.5. 发布语音 30](#_Toc1036926350)

[3.7.6. 发布视频 31](#_Toc791369045)

[3.7.7. 停止发布语音 32](#_Toc1173313444)

[3.7.8. 停止发布视频 32](#_Toc1684206554)

[3.7.9. 会议成员列表查询 33](#_Toc477601971)

[3.7.10. 请求成员视频（更新） 33](#_Toc1909937758)

[3.7.11. 取消成员视频（更新） 34](#_Toc1885826997)

[3.7.12. 重置成员视频（已弃用） 35](#_Toc383192506)

[3.7.13. 会议媒体控制 36](#_Toc12990989)

[3.7.14. 踢出成员 38](#_Toc1443703776)

[3.7.15. 锁定会议 39](#_Toc2059119426)

[3.7.16. 解散会议或取消会议 39](#_Toc921221377)

[3.7.17. 退出会议 40](#_Toc1758072016)

[3.7.18. 查询指定会议 41](#_Toc688873839)

[3.7.19. 查询进行中的会议以及预约会议列表 41](#_Toc818271096)

[3.7.20. 查询历史会议列表或指定历史会议 43](#_Toc197035084)

[3.7.21. 更新会议 44](#_Toc148873114)

[3.8. 会议回调业务接口 45](#_Toc291978243)

[3.8.1. 配置接口 45](#_Toc278196706)

[3.8.2. 接口中回调类型说明 46](#_Toc580138223)

[4. 登出及注销 50](#_Toc807356581)

[4.1. 登出 50](#_Toc1440375121)

[4.2. 释放 52](#_Toc1948989663)

[5. 日志打印及级别说明 52](#_Toc1101198350)

[5.1. 日志打印级别说明 52](#_Toc826598604)

[5.2. 开启日志打印 53](#_Toc571024985)

# 导入说明

## 环境要求

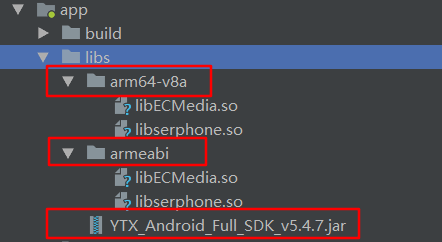
**在您集成容联云 SDK 前环境要求如下：**

* 建议使用Android Studio2.3及以上版本。
* JAVA 编译版本 JDK 1.7 及以上版本。

**SDK版本支持**

* 容联Android SDK最低支持Android **API 14**即安卓**4.0**以上版本。

## 导入sdk



* 如果需要集成会议功能，则需要导入全功能的SDK：YTX\_Android\_Full\_SDK.zip。
* 将解压后的文件导入您工程下的libs目录下。Jar版本号以SDK目录下放置的Jar包实际版本为准。

全能力版 SDK导入libs目录如图所示：

说明：

1).libserphone.so: IM消息和音视频以及音视频会议功能库，必须添加。

2).libECMedia.so: 使用音视频功能，必须添加。

3).libyuntx\_gl\_disp.so: 5.3.0以及之后版本添加了视频图像绘制库，使用ECOpenGLView控件需要添加。（5.6.1之后统一使用ECTextureViewRenderer，需要添加YTX\_Android\_Full\_SDK\_v5.x.xr.jar包）

4).libvoicechange.so: 5.3.0以及之后版本添加了语音变声接口库，用来将录制的语音文件进行音频数据处理转换成不同发声效果。

5).arm64-v8a目录中是64位的so库，适用于CPU是ARM64位的终端设备。

6).armeabi目录中是32位的so库，适用于CPU是通用ARM的终端设备。

7). YTX\_Android\_Full\_SDK\_v5.x.xr.jar是SDK对外提供的Jar包。

注意事项：

1).如果开发者项目需要添加armeabi-v7a目录，可将armeabi目录下的so拷贝至armeabi-v7a目录。

2).目前云通讯只支持arm系列的CPU。其他类型（X86，MIPS等）不支持，因此不能在libs目录下添加ARM类型以外的适配目录

## 配置工程

### 权限配置

//自定义权限配置

云通讯SDK需要用户声明接收IM消息和接收来电的自定义权限，用于接收SDK发出的广播通知。客户可根据需要使用的功能进行配置，如未设置对应权限将无法正常接收广播通知。

|  |
| --- |
| <!--接收IM消息的自定义权限-->  <permission android:name="com.yuntongxun.ecdemo.permission.RECEIVE\_MSG" android:protectionLevel="signature"/>  <!--使用接收IM消息权限-->  <uses-permission android:name="com.yuntongxun.ecdemo.permission.RECEIVE\_MSG" />  <!---接收来电的自定义权限-->  <permission android:name="com.yuntongxun.ecdemo.permission.INCOMING\_CALL" android:protectionLevel="signature"/>  <!--使用接收来电权限-->  <uses-permission android:name="com.yuntongxun.ecdemo.permission.INCOMING\_CALL" /> |

注意事项：

在开发自有项目APP时需要将以上配置中的com.yuntongxun.ecdemo替换为自有项目APP的包名。

//以下为必选配置

|  |
| --- |
| <!-- 允许程序打开网络sockets-->  <uses-permission android:name="android.permission.INTERNET" />  <!-- 允许程序访问有关的网络信息-->  <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  <!-- 允许程序访问Wi-Fi网络状态信息-->  <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />  <!-- 允许应用读写用户的外部存储器-->  <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  <!-- 允许读取电话的状态-->  <uses-permission android:name="android.permission.READ\_PHONE\_STATE" />  <!-- 允许一个程序获取信息有关当前或最近运行的任务，一个缩略的任务状态，是否活动等等-->  <uses-permission android:name="android.permission.GET\_TASKS" />  <!-- 允许程序录制音频-->  <uses-permission android:name="android.permission.RECORD\_AUDIO" />  <!-- 允许应用程序修改全局声音设置的权限-->  <uses-permission android:name="android.permission.MODIFY\_AUDIO\_SETTINGS" />  <!-- 允许使用PowerManager的WakeLocks在进程休眠时能够后台保持-->  <uses-permission android:name="android.permission.WAKE\_LOCK"/>  <!-- 允许请求访问使用照相设备(如果使用视频通话则必选配置) -->  <uses-permission android:name="android.permission.CAMERA" />  //以下列出几种可选配置，开发者可根据自己的需求进行选择配置  //如需使用蓝牙设备，可配置以下权限  <!--允许程序连接到已配对的蓝牙设备-->  <uses-permission android:name="android.permission.BLUETOOTH" />  <!--允许程序发现和配对蓝牙设备-->  <uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />  //如需GPS相关权限，可配置以下权限  <!-- GPS定位权限-->  <uses-permission  android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />  <!--允许一个程序访问CellID或WiFi热点来获取粗略的位置-->  <uses-permission  android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  <!--允许一个程序访问精良位置(如GPS)-->  <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  //其他  <!--允许程序读取用户联系人数据-->  <uses-permission android:name="android.permission.READ\_CONTACTS" />  <!-- 允许访问振动设备-->  <uses-permission android:name="android.permission.VIBRATE" />  <!-- 允许一个程序在系统启动完成时自定启动-->  <uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" /> |

注意事项

1)SDK 的权限声明与application同级，因此要放到AndroidManifest.xml 清单文件的application节点外。

2)文档仅提供了几种可选权限，开发者如需使用其他权限，可以参考Android相关文档，根据自己的应用添加所需权限。

3)部分必须权限在Android6.0及以上机型需要动态权限申请。

### SDK核心配置

清单文件配置项

|  |
| --- |
| // SDK核心服务  <service android:name="com.yuntongxun.ecsdk.ECClientService"  android:enabled="true" android:process=":push" />  //提高sdk服务存活率（v5.2.2r以后版本提供）  <service android:name="com.yuntongxun.ecsdk.ECClientService$InnerService"  android:enabled="true" android:process=":push" />  // 以下两项为SDK内部处理项，其中booter.Alarm在v5.1.9r以后版本提供  <receiver android:name="com.yuntongxun.ecsdk.booter.CCPReceivers$AlarmReceiver"  android:process=":push" />  <receiver android:name="com.yuntongxun.ecsdk.booter.Alarm"  android:exported="false" android:process=":push"/>  // SDK网络改变广播  <receiver  android:name="com.yuntongxun.ecsdk.booter.CCPReceivers$ConnectionReceiver"  android:process=":push">  <intent-filter>  <action android:name="android.net.conn.CONNECTIVITY\_CHANGE" />  </intent-filter>  </receiver>  // SDK广播通知配置（开机广播）  // 可选广播、如果应用想做开机重启，可以实现该广播通知即可  <receiver  android:name="com.yuntongxun.ecsdk.booter.CCPReceivers$BootReceiver"  android:process=":push">  <intent-filter>  <action android:name="android.intent.action.BOOT\_COMPLETED" />  </intent-filter>  </receiver>  //声明自定义通知的广播接收器（需要替换包名，详见注意事项）  <receiver  android:name=".service.YuntxNotifyReceiver"  android:exported="false">  <intent-filter>  // 接收push消息  // action匹配规则{packageName + ".action" + "功能"}  <action android:name="com.yuntongxun.ecdemo.action.VOIP\_CALL" />  <action android:name="com.yuntongxun.ecdemo.action.MESSAGE" />  <action android:name="com.yuntongxun.ecdemo.action.NOTIFICATION\_CLICK" />  <action android:name="com.yuntongxun.ecdemo.action.PERSON\_VERSION" />  </intent-filter>  </receiver> |

注意事项：

1).SDK核心配置是属于application的属性，需要放到AndroidManifest 清单文件的application节点内。

2).声明自定义通知的广播接收器，开发自有项目APP时，action 中的 com.yuntongxun.ecdemo 请替换为自有项目APP的包名。

# 初始化SDK及登录

* 在程序入口初始化SDK并设置回调接口——初始化要在程序完全启动（即Android Application 创建完成）之后才能调用。
* 初始化及登陆过程分由以下四步组成：

1) 初始化SDK

2) 设置登录参数

3) 设置通知回调监听

4) 验证参数是否正确，登陆SDK

**特别提醒：**设置登录参数、设置通知回调监听和验证参数是否正确，登陆SDK，需要在SDK初始化成功的方法（onInitialized回调）后执行。

## 初始化SDK

初始化SDK 是调用其他功能（IM，实时音视频，音视频会议等）的前提，如果初始化不成功，后续的接口都无法调用。

初始化示例代码如下：

|  |
| --- |
| //判断SDK是否已经初始化   * if(!ECDevice.isInitialized()) { * / \*\* initial: ECSDK 初始化接口 * \* 参数： * \* inContext - Android应用上下文对象 * \* inListener - SDK初始化结果回调接口，ECDevice.InitListener * \* * \* 说明：示例在应用程序创建时初始化 SDK引用的是Application的上下文， * \* 开发者可根据开发需要调整。 * \*/ * ECDevice.initial(getApplicationContext(), new ECDevice.InitListener() { * @Override * public void onInitialized() { * // SDK已经初始化成功 * Log.i("","初始化SDK成功");   //设置登录参数，可分为自定义方式和VoIP验密方式  //设置通知回调监听包含登录状态监听，接收消息监听，VoIP呼叫事件回调监听和设置接收VoIP来电事件通知Intent等  //验证参数是否正确，登陆SDK   * } * @Override * public void onError(Exception exception) {   //在初始化错误的方法中打印错误原因   * Log.i("","初始化SDK失败"+exception.getMessage()); * } * }); * }   // 已经初始化成功，后续开发业务代码。   * Log.i(TAG, "初始化SDK及登陆代码完成"); |

注意事项：

1）初始化不要在application的onCreate方法中进行调用，因为如果android的application没有创建完成时执行初始化方法，将会导致初始化无效，在后续的登录和其他接口的调用中出现问题。

2）如果调用初始化触发初始化失败onError回调。可能是由下列原因造成：

a、可能SDK已经处于初始化状态(未做ECDevice.isInitialized判断时出现)。

b、SDK所声明必要的权限未在清单文件（AndroidManifest.xml）里配置、

或者未配置服务属性android:exported="false";

c、当前手机设备系统版本低于SDK所支持的最低版本（当前SDK支持Android

Build.VERSION\_CODES.FROYO 以及以上版本）

## 设置登录参数

初始化成功之后我们就可以进行登录。

1）登陆方式：任意帐号体系(即自定义登录方式)

2）账号登陆限制：不能包含中文，特殊符号，不能以g开头，建议在5-20位之间。

3）示例代码如下:

任意帐号体系(即自定义登录方式)

|  |
| --- |
| * ECInitParams params = ECInitParams.createParams(); * params.setUserid("推荐使用客户项目APP的登录帐号，测试阶段Userid可以填写手机号"); * params.setAppKey("APP ID"); * params.setToken("APP TOKEN");   // 设置登陆验证模式：自定义登录方式   * params.setAuthType(ECInitParams.LoginAuthType.NORMAL\_AUTH); * // LoginMode（强制上线：FORCE\_LOGIN 默认登录：AUTO。使用方式详见注意事项） * params.setMode(ECInitParams.LoginMode.FORCE\_LOGIN); |

注意事项：

1）相同终端之间不可以登录同一账号，移动设备之间不可以登录同一账号。移动设备、平板、电脑设备、Web、PC之间可以登录相同账号。

2）自定义方式（NORMAL\_AUTH）登录时Userid不可填VoIP帐号，否则会导致VoIP密码失效。

3）两种登录模式的使用：

FORCE\_LOGIN ：强制登录，可以强制上线，踢掉已经在线的同终端设备。

AUTO：自动重连

建议在首次登陆时使用FORCE\_LOGIN强制上线登录模式，之后再次启动APP时使用AUTO自动重连模式。

4）SDK三种种情况后会自动重连

1. 登录方式为FORCE\_LOGIN类型并且登陆成功
2. 客户端使用AUTO登录方式并登陆成功
3. 手机网络状态改变

5）自定义登录方式和VoIP验密登录方式在开发者正式项目中只可选用一种。

## 设置通知回调监听

登录参数设置完成后，开发者可根据业务需要设置回调监听。如下表所示：

|  |  |  |
| --- | --- | --- |
| **监听** | **说明** | **约束** |
| OnECDeviceConnectListener | 登录回调监听 | 必须设置 |
| OnConferenceListener | 音视频会议回调监听 | 使用音视频会议时设置 |

说明：

1)登录回调监听将SDK登录状态反馈给APP层，是所有通讯功能的基础，因此必须设置。

### 设置登录回调监听

|  |
| --- |
| * ECDevice.setOnDeviceConnectListener(new ECDevice.OnECDeviceConnectListener() { * public void onConnect() { * //兼容旧版本的方法，不必处理 * } * @Override * public void onDisconnect(ECError error) { * //兼容旧版本的方法，不必处理 * } * @Override * public void onConnectState(ECDevice.ECConnectState state, ECError error) { * if(state == ECDevice.ECConnectState.CONNECT\_FAILED ){ * if(error.errorCode == SdkErrorCode.SDK\_KICKED\_OFF) { * Log.i("","==帐号异地登陆"); * }else * { * Log.i("","==其他登录失败,错误码："+ error.errorCode); * } * return ; * } * else if(state == ECDevice.ECConnectState.CONNECT\_SUCCESS) { * Log.i("","==登陆成功"); * } * } * }); |

### 设置音视频会议回调监听

|  |
| --- |
| ECDevice.*getECConferenceManager*().setConferenceListener(new OnConferenceListener() {  @Override  public void onReceivedConferenceNotification(ECConferenceNotification ecConferenceNotification) {  //会有不同的通知，下面文档详细介绍   * }   }); |

注意事项：

回调监听要放在初始化成功的onInitialized回调方法里面，这样保证登录成功之后就可以接收到他人的来电、消息、会议通知等。

## 切换私有云环境

**1.在登录sdk之前调用ECDevice.initServer((arg0, arg1)**

|  |
| --- |
| public static boolean initServer(android.content.Context ctx, java.lang.String xml)  将服务器地址对应的XML文件初始化到SDK访问目录 login(ECInitParams)接口之前调用 参数: ctx - Android应用上下文 xml - 服务器地址对应的XML文件 |

例如：   
Connector ip：123.57.33.80 端口：8085  
lvs ip：123.57.33.80 端口：8888  
Fileserver ip：123.57.33.80 端口：8090

**示例文件：**

****

## 登录SDK

设置完成登录参数和通知回调监听，接下来调用验证参数并登录SDK。如果返回是true，则说明参数正确。

|  |
| --- |
| * //验证参数是否正确 * if(params.validate()) { * // 登录函数 * ECDevice.login(params); * } |

接下来我们可以测试是否登录成功。

调试登录回调监听（ECDevice.OnECDeviceConnectListener）来判断连接状态。登录成功的errorcode为200.

# 音视频会议开发说明

## 接口逻辑

接口调用是采取“请求回调”和“通知回调”的方式和服务端交互。用户发出邀请、踢出、解散等操作时，在对应的sdk的方法参数中有“请求回调”方法，请求的结果直接在参数的回调方法中处理。会议中的成员收到服务端发送过来的消息通过“通知回调”接口OnConferenceListener中的onReceivedConferenceNotification接收并进行处理。

## 业务流程

1) 用户A创建会议，创建成功后加入会议，然后发布语音和视频

2) 用户A邀请用户B或者用户B主动加入会议，加入成功后，发布语音和视频

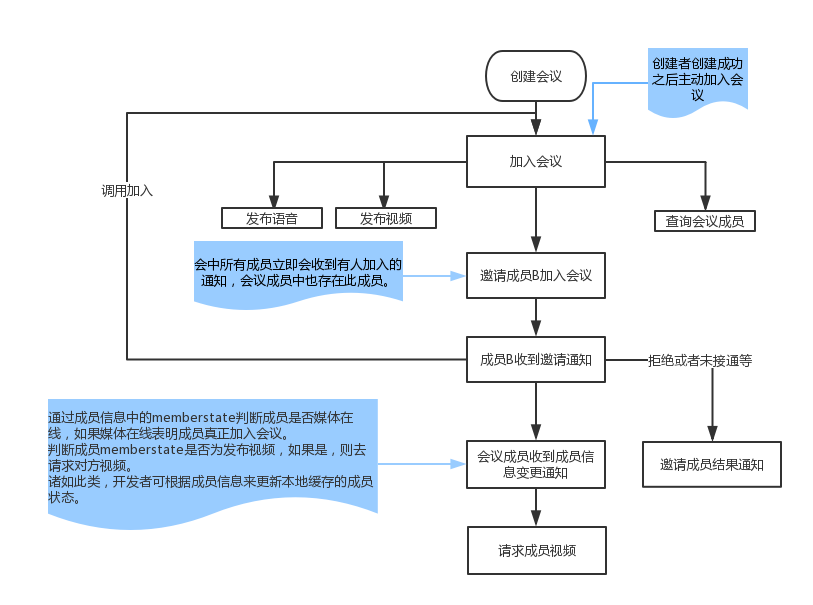
3) 用户A作为主持人可以对所有会议成员静音、踢出等会控操作

4) 用户A作为主持人即可解散会议，也可退出会议，如果退出会议，第二个加入会议的会变更为临时主持人，以此类推，如果用户A再次加入会议，身份依然是主持人,临时主持人变为普通成员。

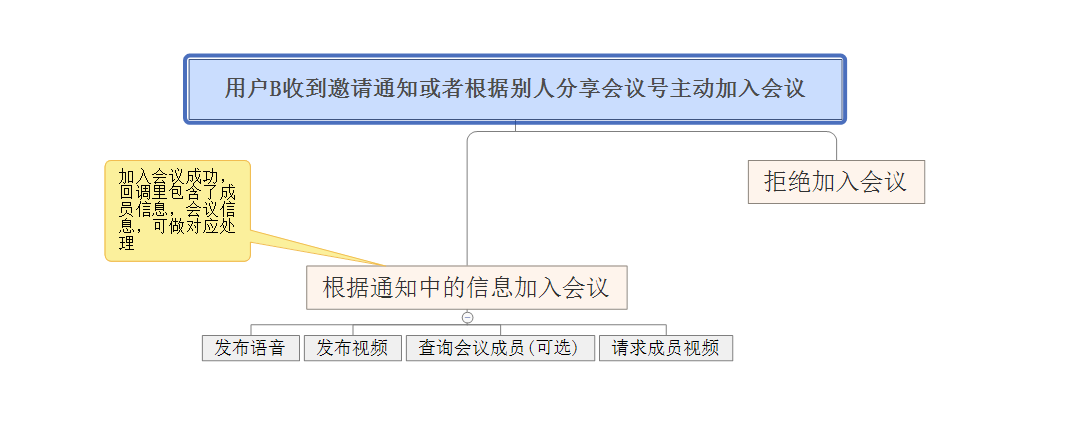
5) 普通成员只能退出会议。

## 常见业务流程示意图

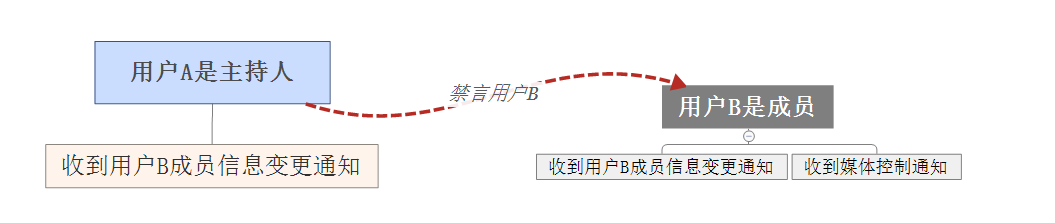
### 创建会议



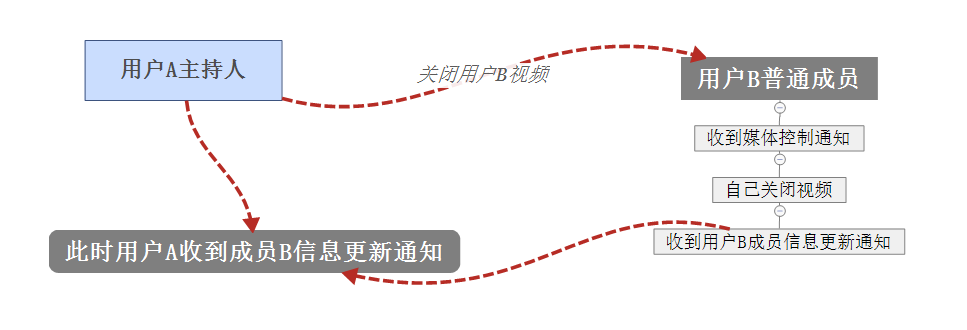
### 加入会议



### 禁言成员



### 关闭成员视频



## 创建视频视频会议准备工作

注意事项：如果是电话会议以及语音会议可忽略此条。

### 视频流显示

注意事项：视频渲染，最新版SDK中统一使用ECTextureViewRenderer。

开发者使用封装好的视频控件AVDSurfaceFrameLayout，对应方法参看接口说明IAVDVidyoFrame。

|  |
| --- |
| * public interface IAVDVidyoFrame { * /\*\* * \* 当前控件显示的成员 * \* @return 显示的成员 * \*/ * NetMeetingMember getEmployee() ; * /\*\* * \* 当前控件显示的成员 * \* @param employee 成员 * \*/ * void setEmployee(NetMeetingMember employee); * /\*\* * \* 判断是否可以绘制 * \* @return 是否可以绘制 * \*/ * boolean isRender(); * /\*\* * \* 初始化界面 * \*/ * void init(); * /\*\* * \* 绘制远端视频数据 * \* @param data 远端视频数据 * \* @param width 视频数据-宽 * \* @param height 视频预览数据-高 * \* @param length 视频预览数据-大小 * \*/ * void renderFrame(byte[] data, int width, int height, int length);  * void renderCapture(byte[] data, int width, int height, int length, int size); * void renderFrame(VideoFrame frame , boolean isSelf); * /\*\* * \* 视频会议成员图像操作结果回调接口 * \* @param isRequest 是否请求成员图像 * \* @param success 是否成功 * \* @param account 成员账号 * \*/ * void obtainVideoFrameChange(boolean isRequest, boolean success, String account); * /\*\* * \* 发布视频会议图像 * \*/ * void requestFrame(); * /\*\* * \* 取消视频会议图像 * \*/ * void cancelFrame(); * /\*\* * \* 发布视频会议图像 * \*/ * void resetFrame(); * /\*\* * \* 当前视频会议图像是否成功请求 * \* @return 是否成功请求 * \*/ * boolean isFrameActivate(); * /\*\* * \* 刷新当前的界面 * \*/ * void invalidateFrame() ; * } |

## 会议接口中常用对象

### ECAccountInfo(会议成员对象)

|  |
| --- |
| * //成员的id * private String accountId; * //成员的类型  ECAccountType\_PhoneNumber, ECAccountType\_AppNumber * private ECConferenceEnums.ECAccountType ecAccountType; * //电话号 * public String phoneNumber; * //名字 * public String userName; * //邀请结果 * private String inviteResult; * //角色id （1主持人 2管理员 3普通成员） * private int roleId; * //设备类型 IPHONE、ANDROID\_PHONE、PC、WEB等 * private ECConferenceEnums.ECDeviceType ecDeviceType; |

### ECConferenceMemberInfo(成员信息对象)

|  |
| --- |
| * //名字 * private String userName; * //扩展字段 * private String appData; * //成员的状态 * public int state; * //成员对象 * private ECAccountInfo member; |

注意：判断会议成员状态通过成员信息里面的state与以下值进行&运算，如果运算结果大于0则表示当前状态，当前状态可以为多个。

// 离线

public static final int *OFFLINE\_STATE* = 0x000000;

// 在线

public static final int *ONLINE\_STATE* = 0x000001;

// 正在说话

public static final int *SPEAKING\_STATE* = 0x000002;

// 打开摄像头

public static final int *VIDEO\_STATE* = 0b000100;

// 在白板房间

public static final int *WBSSSHARE\_STATE* = 0x000010;

// 可讲状态

public static final int *AllOW\_SPEAKING\_STATE* = 0x000040;

// 可听

public static final int *LISTEN\_STATE* = 0x000080;

// 媒体在线

public static final int *MEDIA\_ONLINE\_STATE* = 0x000100;

// 拒绝

public static final int *REFUSE\_STATE* = 0x000200;

// 正在邀请

public static final int *INVITE\_ING* = 0x000400;

// 挂断

public static final int *HANG\_UP* = 0x000800;

// 视频共享

public static final int *SCREEN\_SHARING* = 0x000008;

### ECConferenceInfo(会议信息对象)

|  |
| --- |
| * //会议中成员人数 * private int memberCount; * //会议id * private String conferenceId; * //会议的创建者 * private ECAccountInfo creator; * //会议的名字 * private String confName; * //会议的创建者密码 * private String ownerPassword; * //会议的密码 * private String password; * //会议的类型 * ECConferenceType\_Temporary(临时), * ECConferenceType\_Permanent(永久), * private ECConferenceEnums.ECConferenceType confType; * //会议成员的信息集合   /\*格式：members={"members":[{"memberId":"252525","idType":2,"userName":"会议1"},{"memberId":"112211","idType":2,"userName":"会议2"}]}  \*/   * private List<ECConferenceMemberInfo> memberInfos; * //会议的最大人数 * private int maxMember; * //会议的状态 * public int state;   //0：自由模式，1：主持人模式，  private static int confMode = 1;  /\*\*主持人离开会议后，会议是否自动结束，confMode=1时，此字段生效  0:否，1：是,默认值1\*/  private static int autoClose = 0;  /\*\*0 是纯落地电话会议  1是多路视频流 2是视频混屏 3是多路 + 混屏 4是实时对讲\*/   * private static int mediaType = 1; * //是否是预约会议 0 否 1 是 * public int reserveEnable; * //会议创建的时间 * public String createTime; * //会议更新的时间 * public String updateTime; * //会议的时长 单位 分钟 * public long duration; * //会议的预约时间 * public String reserveStartTime; * //会议的媒体类型 * public long mediaType; * //会议的结束时间 * public String endTime; * //会议的开始时间 * public String startTime; |

注意：判断会议状态通过会议信息里面的state与以下的值进行&运算，如果运算结果大于0，则表示当前状态。

//会议未开始  
public static final int *NOT\_START\_STATE* = 0b00000000;  
//会议已开始  
public static final int *UNDER\_WAY\_STATE* = 0b00000001;  
//会议被锁定  
public static final int *LOCK\_STATE* = 0b00000010;  
//会议得白板已锁定  
public static final int *WHITE\_SHARE\_LOCK* = 0b00000100;  
//会议得白板标注已锁定  
public static final int *WHITE\_OPERATE\_LOCK* = 0b00001000;  
//当前会议中正在共享白板  
public static final int *WHITE\_SHARING* = 0b00010000‬;  
//当前会议中正在屏幕共享  
public static final int *SCREEN\_SHARING* = 0b00100000‬;

‬‬‬‬

## 会议接口中常用枚举

### ECControlMediaAction

|  |
| --- |
| * /\*\*多个改变\*/ * ECControlMediaAction\_MoreChange(-1), * /\*\* 禁听 \*/ * ECControlMediaAction\_CloseListen(0), * /\*\* 可听 \*/ * ECControlMediaAction\_OpenListen(1), * /\*\* 禁言 \*/ * ECControlMediaAction\_CloseSpeak(2), * /\*\* 可说 \*/ * ECControlMediaAction\_OpenSpeak(3), * /\*\* 停止观看视频 \*/ * ECControlMediaAction\_CloseLookVideo(10), * /\*\* 观看视频 \*/ * ECControlMediaAction\_OpenLookVideo(11), * /\*\* 停止发布视频 \*/ * ECControlMediaAction\_StopPublish(12), * /\*\* 发布视频 \*/ * ECControlMediaAction\_PublishVideo(13), * /\*\* 停止观看屏幕共享\*/   ECControlMediaAction\_CloseLookScreen(20),   * /\*\* 观看屏幕共享\*/   ECControlMediaAction\_OpenLookScreen(21),   * /\*\* 关闭屏幕共享 \*/   ECControlMediaAction\_CloseScreen(22),   * /\*\* 打开屏幕共享 \*/   ECControlMediaAction\_OpenScreen(23),   * /\*\* 设置会议成员角色 \*/   ECControlMediaAction\_SetRole(50),  ECControlMediaAction\_UserInfo(51),  ECControlMediaAction\_MultiMediaRemove(66),  ECControlMediaAction\_MemberQuit(67);   * /\*\* 直播相关，普通会议插件可忽略 \*/   ECControlAction\_ConfLiveStart(69),  ECControlAction\_ConfLiveStop(70),  ECControlAction\_ConfLiveSwitch(71),  ECControlAction\_ConfLiveGetPlayUrl(72),  ECControlAction\_ConfFileChanged(75),  ECControlAction\_ConfAbstractChanged(76),  ECControlAction\_ApplyPublishVoiceByModerator(77),  ECControlAction\_AcceptPublishVoiceByModerator(78),  ECControlAction\_RejectPublishVoiceByModerator(79),  ECControlAction\_StopPublishVoiceByModerator(80),  ECControlAction\_ApplyPublishVoiceBySelf(81),  ECControlAction\_AcceptPublishVoiceBySelf(82),  ECControlAction\_RejectPublishVoiceBySelf(83),  ECControlAction\_StopPublishVoiceBySelf(84),  ECControlAction\_ControlMemberDisconnect(121),  ECControlAction\_ControlSetConfLayout(92),  ECControlAction\_CancelRaiseHandBySelf(126); |

### ECConferenceType

|  |
| --- |
| * /\*\*临时会议\*/ * ECConferenceType\_Temporary , * /\*\*永久会议\*/ * ECConferenceType\_Permanent , * /\*\*如果是群内通话则需要\*/ * ECConferenceType\_Sudoku， |

### ECAccountType

|  |
| --- |
| * /\*\*电话参会\*/ * ECAccountType\_PhoneNumber , * /\*\*app参会\*/ * ECAccountType\_AppNumber , |

## 会议主调业务接口

接口使用前注意事项：

如果接口中参数是可选，如果不需要该参数，int类型可传为-1，String类型可传为空。

### 创建会议或预约会议

|  |
| --- |
| * //获取confmgr * ECConferenceManager conferenceManager = ECDevice.getECConferenceManager(); * //创建ECConferenceInfo 对象 * ECConferenceInfo conferenceInfo = new ECConferenceInfo(); * //设置会议名字 * conferenceInfo.setConfName("会议name"); * //设置会议密码 * conferenceInfo.setPassword("password"); * //设置会议最大人数 * conferenceInfo.setMaxMember(getMaxNum()); * //设置会议的背景音模式 推荐设置ECConferenceVoiceMode\_All * conferenceInfo.setVoiceMode(ECConferenceVoiceMode\_All); * //设置会议的类型 * //ECConferenceType\_Temporary, 临时会议 * //ECConferenceType\_Permanent, 永久会议 * //ECConferenceType\_Reservation; 弃用 * conferenceInfo.setConfType(ECConferenceType\_Temporary); * /\*\* * \* 创建会议 * \* 第一个参数为必选，其他均为可选 * \* @param ECConferenceInfo 会议相关信息 * \* @param int confMode 会议模式 * \* @param int autoClose 主持人离开会议后是否自动关闭会议(0:否，1:是) * \* @param int moderator 指定主持人ID * \* @param string 开始时间 * \* @param string 邀请的成员json字符串 例如members:[{"memberId":"xxx","memberIdType":1}, {...}, ...] * \* @param int duration 会议持续时间，单位：分钟 默认值60 * \* @param int sendInvitation 到时间之后是否发送入会邀请（对于应用账号发送邀请通知，对于手机号发起邀请呼叫）0：否，1：是;默认值 1 * \* @param int sendReserveNote 启用预约的会议中使用这个字段，是否给会议成员发送提醒通知（提醒成员被列入预约会议成员） 0：否，1：是 默认值 0 * \* @param int remindBeforeStart 预约的会议中使用这个字段，会议开始前多久发送提醒通知，单位：分钟.默认值10 * \* @param int remindBeforeEnd  :预约的会议中使用这个字段，会议结束前多久发送提醒通知，单位：分钟默认值10 * \* @param int mediaType: 0:落地电话会议 1:多路视频会议 2:混屏视频会议 3:多路+混屏视频会议 4 实时对讲 * \* @param int reserveEnable: 是否启用预约功能,0:不启用，1：启用;预约会议必须传1 * \* @param OnCreateConferenceListener 创建结果回调 200成功 * \*/ * conferenceManager.createConference(conferenceInfo, confMode, autoClose, moderator, "", members.toString(),30, -1, 0, 3, -1, mediaType, 0, new ECConferenceManager.OnCreateConferenceListener() {   @Override  public void onCreateConference(ECError ecError, ECConferenceInfo ecConferenceInfo) {   * if (ecError.errorCode == SdkErrorCode.REQUEST\_SUCCESS) { * LogUtil.e(TAG, "create success"); * } * } * } * ); |

### 邀请成员加入会议

注意：

1. 如果进行中的会议邀请别人，虽然被邀请人还未同意加入会议，但是会议中成员会立即收到有人加入的通知，会议成员中也存在此成员。但此时被邀请者可能并未真正加入会议，判断被邀请者是否真正加入，是根据成员信息变更通知来判断，如果成员信息变更通知中携带的state表明该成员媒体在线，那么说明成员已加入会议，这时候再去请求对方视频等。

2.会议中邀请其他人加入会议时不要传phoneNumber，带了 phoneNumber 参数，触发CCM电话会议相关逻辑，然后被邀请人调用 MemberJoin 入会时，会再次收到邀请通知。所以不使用电话会议相关功能时，就不要传phoneNumber参数

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //组装成员对象 * List<ECConferenceMemberInfo> memberList = new ArrayList<>();       //创建成员   * ECAccountInfo accountInfo = new ECAccountInfo(); * //设置被邀请者的名字 * accountInfo.setAccountId("被邀请者Id"); * //设置被邀请者的类型 ECAccountType\_PhoneNumber:电话参会类型 * //ECAccountType\_AppNumber:app参会类型 * accountInfo.setEcAccountType(*ECAccountType\_AppNumber*); * //设置被邀请者的名字 * accountInfo.setUserName("被邀请者名字"); * //设置成员信息对象 * ECConferenceMemberInfo memberInfo = new ECConferenceMemberInfo(); * memberInfo.setMember(accountInfo); * memberList.add(memberInfo); * /\*\* * \* 邀请成员 * \* @param List<ECConferenceMemberInfo>  必选 成员信息集合 * \* @param string 必选 会议iD * \* @param int 可选 callImmediately : 是否立即发起呼叫 1表示立即给邀请者发起呼叫,0表示仅在会议中增加成员（一般用户预约会议开始前增加成员） * \* @param string 可选 appdata 扩展字段 * \* @param OnInviteMemberListener 邀请回调 200成功 * \*/ * conferenceManager.inviteMembers(memberList, "会议号", 1, "", * new ECConferenceManager.OnInviteMemberListener() { * @Override * public void onInviteMembers(ECError reason) { * LogUtil.*e*(*TAG*, "doInviteMobileMember Net" + reason.errorCode); * } * }); |

### 加入会议

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //创建加入信息对象 * ECConferenceJoinInfo conferenceJoinInfo = new ECConferenceJoinInfo(); * //设置加入会议的Id * conferenceJoinInfo.setConferenceId(mMeetingNo); * //设置加入会议的密码 * conferenceJoinInfo.setPassword(""); * //设置自己加入会议的名字 * conferenceJoinInfo.setUserName("名字"); * /\*\* * \* 加入会议 * \* @param ECConferenceJoinInfo 必选 加入信息 * \* @param int 可选 inviteType 邀请者ID类型：1，电话号码；2，平台账户 * \* @param int 可选 mediaType 0:落地电话会议 1:多路视频会议 2:混屏视频会议 3:多路+混屏视频会议 * \* @param OnJoinOrQuitConferenceListener 加入回调 200成功 * \*/ * conferenceManager.joinConference(conferenceJoinInfo, inviteType, mediaType, new ECConferenceManager.OnJoinOrQuitConferenceListener() { * @Override * public void onJoinOrQuitConference(ECError reason, ECConferenceInfo info) { * LogUtil.*e*(*TAG*, "joinconference " + reason.errorCode); * } * }); |

### 拒绝加入

|  |
| --- |
| * //获取confMgr * ECConferenceManager manager = ECDevice.*getECConferenceManager*(); * /\*\* * \* 拒绝加入会议 * \* @param string 必选 会议Id * \* @param string 可选 cause 邀请结果,可以自定义透传错误码 * \*/   manager.rejectConferenceInvitation(meetingNo, "邀请结果", new ECConferenceManager.OnRejectInvitationListener() {   * @Override * public void onRejectResult(ECError ecError) { * LogUtil.*e*(*TAG*, "onRejectResult " + ecError.errorCode); * } * }); |

### 发布语音

调用此接口发布自己的语音，也可以用于会议中对自己解除静音，这个可以应用层来控制。

|  |
| --- |
| * /\*\* * \* 发布语音 * \* @param string 必选 会议Id * \* @param int 可选 exclusively : 仅用于实时对讲功能。 * 1 表示控麦，此时仅允许会中有一个人发布语音，如果已经有人发布语音，此接口调用会返回错误 * 0 表示发布语音，不考虑其他人语音发布状态 * 默认为0 * \* @param OnPublishVoiceInConferenceListener 发布语音的回调 * \*/ * conferenceManager.publishVoiceInConference("会议ID", 0, new ECConferenceManager.OnPublishVoiceInConferenceListener() { * @Override * public void onPublishVoiceInConference(ECError ecError) { * LogUtil.*e*(*TAG*, "publishVoiceInConference  code is " + ecError.errorCode); * } * }); |

### 发布视频

用户关闭的视频可重新发布打开视频。

注意：成员加入视频会议后需要调用发布视频接口，默认是关闭的。对于音频，服务器的默认设置是每个人进入都是可听状态，可以不用发布音频。如果遇到听不见声音的情况，可以确认是否发布了音频。

|  |
| --- |
| * /\*\* * \* 发布视频 * \* @param string 必选 会议Id * \* @param OnPublishVideoInConferenceListener 发布视频的回调 * \*/ * conferenceManager.publishVideoInConference("会议ID", new ECConferenceManager.OnPublishVideoInConferenceListener() { * @Override * public void onOnPublishVideoInConference(ECError ecError) { * LogUtil.*e*(*TAG*, "publishVideoInConference  code is " + ecError.errorCode); * } * }); |

### 停止发布语音

用户想要静音自己可以调用停止发布语音的接口。

|  |
| --- |
| * /\*\* * \* 停止发布语音 * \* @param string 必选 会议id * \* @param int 可选 exclusively: 仅用于实时对讲功能。 * \*            1 表示 放麦，放麦成功后其他人才可以控麦 * 0 表示发布语音，不考虑其他人语音发布状态 * 默认为0 * \* @param OnCancelPublishVoiceInConferenceListener 200表示成功 * \*/ * conferenceManager.cancelVoiceInConference("会议ID", 0, new ECConferenceManager.OnCancelPublishVoiceInConferenceListener() { * @Override * public void onCancelPublishVoiceInConference(ECError ecError) { * LogUtil.*e*(*TAG*, "cancelPublishVoice  code is " + ecError.errorCode); * } * }); |

### 停止发布视频

用户想关闭自己的视频但可以看到会议室中其他成员视频可调用此方法。

|  |
| --- |
| * /\*\* * \* 停止发布视频 * \* @param string 必选 会议id * \* @param OnCancelPublishVideoInConferenceListener 200表示成功 * \*/ * conferenceManager.cancelVideoInConference("会议ID", * new ECConferenceManager.OnCancelPublishVideoInConferenceListener() { * @Override * public void onCancelPublishVideoInConference(ECError ecError) { * LogUtil.*e*(*TAG*, "cancelPublishVideo  code is " + ecError.errorCode); * } * }); |

### 会议成员列表查询

用户加入会议后，需要获取当前会议中的所有成员。

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //设置获取过滤规则对象 * ECConferenceFilterInfo info = new ECConferenceFilterInfo(); * //设置获取第几页 可选参数 * info.setNumber(1); * //设置每页返回的人数 默认20 最大100 可选参数 * info.setSize(20); * /\*\* * \* 获取成员列表 * \* @param string 必选 会议Id * \* @param ECConferenceFilterInfo 必选 过滤规则对象 * \* @param OnGetMembersListener 回调 200成功 * \*/ * conferenceManager.getMemberListOfConference(meetingNo, info,new ECConferenceManager.OnGetMembersListener() { * @Override   public void onGetMember(ECError ecError, List<ECConferenceMemberInfo> list) {   * LogUtil.*e*(*TAG*,"onGetMember code is " + ecError.errorCode); * } * }); |

### 请求成员视频（更新）

注意：进入会议，请求成员视频的前提是，此成员需要先调用发布视频接口，不然请求成员视频会报未找到。

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //设定action * ECConferenceEnums.ECControlMediaAction action = ECConferenceEnums.ECControlMediaAction.ECControlMediaAction\_PublishVideo; * } * //参数说明参见 ：*[会议媒体控制](#_会议媒体控制)* * conferenceManager.conferenceMediaControl(action, meetingNo, isAll, 0, * memberInfoList, new ECConferenceManager.OnConferenceMediaControlListener() { * @Override * public void onConferenceMediaControl(ECError ecError) { * LogUtil.*e*(*TAG*, "onConferenceMediaControl isall = " + isAll + " ecerror " + ecError.errorCode); * }           }); |

### 取消成员视频（更新）

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //设定action * ECConferenceEnums.ECControlMediaAction action = ECConferenceEnums.ECControlMediaAction.ECControlMediaAction\_StopPublish; * } * //参数说明参见 ：*[会议媒体控制](#_会议媒体控制)* * conferenceManager.conferenceMediaControl(action, meetingNo, isAll, 0, * memberInfoList, new ECConferenceManager.OnConferenceMediaControlListener() { * @Override * public void onConferenceMediaControl(ECError ecError) { * LogUtil.*e*(*TAG*, "onConferenceMediaControl isall = " + isAll + " ecerror " + ecError.errorCode); * }           }); |

### 重置成员视频（已弃用）

重置成员视频的方法一般是用于切换大小流的。传的view的宽高影响请求的视频分辨率。sdk会根据view的宽高来设置分辨率。宽高积跟哪些分辨率最接近就设置哪种分辨率。

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //创建videoinfo对象 * ECConferenceVideoInfo videoInfo = new ECConferenceVideoInfo(); * //必选 设置会议id * videoInfo.setConferenceId("会议Id"); * //必选 设置请求的view view的宽高影响视频的分辨率 * videoInfo.setView(glView); * //创建成员对象 * ECAccountInfo accountInfo = new ECAccountInfo(); * //设置成员id * accountInfo.setAccountId("成员Id"); * videoInfo.setMember(accountInfo); * /\*\* * \* 重置成员视频 * \* @param ECConferenceVideoInfo videoinfo对象 * \* 返回值 0 代表成功 * \*/ * int resetCode = conferenceManager.resetMemberVideoSSRC(videoInfo); * if (resetCode == 0) { * LogUtil.*e*(*TAG*, resetCode + "  resetMemberVideoSSRC success"); * } |

### 会议媒体控制

成员的静音，解除静音，停止发布视频等。

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //设置成员信息列表 * List<ECConferenceMemberInfo> memberInfoList = new ArrayList<>(); * //创建成员信息对象 * ECConferenceMemberInfo conferenceMemberInfo = new ECConferenceMemberInfo(); * //创建成员 * ECAccountInfo accountInfo = new ECAccountInfo(); * //设置成员id * accountInfo.setAccountId("被控制成员ID"); * //设置成员的类型 ECAccountType\_PhoneNumber, ECAccountType\_AppNumber;   accountInfo.setEcAccountType(ECConferenceEnums.ECAccountType.*ECAccountType\_AppNumber*);   * //设置成员的名字 * conferenceMemberInfo.setUserName("被控制成员名字"); * conferenceMemberInfo.setMember(accountInfo); * memberInfoList.add(conferenceMemberInfo); * /\*\* * \* 媒体控制接口 * \* @param ECConferenceEnums.ECControlMediaAction 必选 action * \*      // 禁听 * ECControlMediaAction\_CloseListen(0) * //可听 * ECControlMediaAction\_OpenListen(1) * //禁言 * ECControlMediaAction\_CloseSpeak(2) * //可说 * ECControlMediaAction\_OpenSpeak(3) * // 停止观看视频 * ECControlMediaAction\_CloseLookVideo(10) * // 观看视频 * ECControlMediaAction\_OpenLookVideo(11) * // 停止发布视频 * ECControlMediaAction\_StopPublish(12) * // 发布视频 * ECControlMediaAction\_PublishVideo(13)等 * 参考文档[ECControlMediaAction](#_ECControlMediaAction) * \* @param string 必选 会议id * \* @param int 可选 allMember 0: 作用于members参数指定的成员 * \*1: 作用于会议中的所有成员（主持人除外）2: 作用于会议中所有成员（主持人除外），及后续加入会议的成员，取值为2时，仅影响 禁讲、可讲（action=2 action=3）功能 * \* @param int 可选 unchangable 是否不可被成员更改 0：否，1：是 * \* @param List<ECConferenceMemberInfo> 必选 成员信息列表 * \* @param OnConferenceMediaControlListener 200成功 * \*/ * conferenceManager.conferenceMediaControl(action, meetingNo, isAll, 0, * memberInfoList, new ECConferenceManager.OnConferenceMediaControlListener() { * @Override * public void onConferenceMediaControl(ECError ecError) { * LogUtil.*e*(*TAG*, "onConferenceMediaControl isall = " + isAll + " ecerror " + ecError.errorCode); * } * }); |

### 踢出成员

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //设置成员信息集合 * List<ECConferenceMemberInfo> memberInfoList = new ArrayList<>(); * //创建成员信息对象 * ECConferenceMemberInfo conferenceMemberInfo = new ECConferenceMemberInfo(); * //创建成员对象 * ECAccountInfo accountInfo = new ECAccountInfo(); * //设置被踢出成员的ID * accountInfo.setAccountId("被踢出成员的ID"); * //设置成员类型   ECAccountType\_PhoneNumber,ECAccountType\_AppNumber;   accountInfo.setEcAccountType(ECConferenceEnums.ECAccountType.*ECAccountType\_AppNumber*);   * conferenceMemberInfo.setMember(accountInfo); * memberInfoList.add(conferenceMemberInfo); * /\*\* * \* 踢出成员 * \* @param List<ECConferenceMemberInfo> 必选 成员信息集合 * \* @param string 必选 会议id * \* @param string 可选 appdata 扩展字段 * \*/ * conferenceManager.kickMembers(memberInfoList, "会议ID", "", "", new ECConferenceManager.OnKickMemberListener() { * @Override * public void onKickMembers(ECError ecError) { * LogUtil.*e*(*TAG*, "onKickMembers code is " + ecError.errorCode); * } * }); |

### 锁定会议

|  |
| --- |
| * //获取confMgr * ECConferenceManager manager = ECDevice.*getECConferenceManager*(); * /\*\* * \* 锁定会议 * \* @param string 必选 会议ID * \* @param int 0锁定 1解锁 * \*/   manager.lockConference("会议号", lockAction, new ECConferenceManager.OnLockConferenceListener() {   * @Override * public void onLockConference(ECError ecError) { * LogUtil.*e*(*TAG*, "onLockConference code is " + ecError.errorCode); * } * }); |

### 解散会议或取消会议

创建者或主持人解散会议时，会中所有成员收到会议解散的通知。

|  |
| --- |
| * //获取confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //创建会议信息对象 * ECConferenceInfo conferenceInfo = new ECConferenceInfo(); * //设置会议id 必选 * conferenceInfo.setConferenceId("会议id"); * /\*\* * \* 解散会议 * \* @param ECConferenceInfo 必选会议信息 * \* @param OnDeleteConferenceListener 200 成功 * \*/ * conferenceManager.deleteConference(conferenceInfo, new ECConferenceManager.OnDeleteConferenceListener() { * @Override * public void onDeleteConference(ECError ecError) { * LogUtil.*e*(*TAG*, "ecerror onDeleteConference " + ecError.errorCode); * } * }); |

### 退出会议

|  |
| --- |
| * //设置confMgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * /\*\* * \* 退出会议 * \* @param string 会议Id * \* @param OnJoinOrQuitConferenceListener 200成功 * \*/ * conferenceManager.quitConference(meetingNo, new ECConferenceManager.OnJoinOrQuitConferenceListener() { * @Override * public void onJoinOrQuitConference(ECError ecError, ECConferenceInfo info) { * LogUtil.*e*(*TAG*, " onJoinOrQuitConference " + ecError.errorCode); * } * }); |

注意：如果是误操作退出或者断线等非正常退出，服务器会有5分钟的检测时间，5分钟之内如果检测到此会议还存在，依然是可以再次进入会议的；如果5分钟检测不到，才会把这个会议删除。

### 查询指定会议

|  |
| --- |
| * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * /\*\* * \* 查询指定会议  只能查询进行中的会议 或者 预约会议 * 历史会议 使用 getHistoryConfListByAccount 查询 * \* @param string 必选 会议id * \* @param OnGetConferenceListener 200 成功 * \*/ * conferenceManager.getConference(mMeetingNo, new ECConferenceManager.OnGetConferenceListener() { * @Override * public void onGetConference(ECError ecError, ECConferenceInfo ecConferenceInfo) { * LogUtil.*e*(*TAG*, "queryConference result is " + ecError.errorCode); * } * }); |

### 查询进行中的会议以及预约会议列表

|  |
| --- |
| * ECConferenceManager ecConferenceManager = ECDevice.*getECConferenceManager*(); * //创建条件对象 * ECConferenceCondition ecconferenceCondition = new ECConferenceCondition(); * //必选 设置查询成员id * ecconferenceCondition.setMemberId("成员Id"); * //必选 设置成员类型  1: 电话号码，2: 应用账号 * ecconferenceCondition.setIdType(2); * //可选 设置查询会议的类型ECConferenceType\_Temporary(临时会议),ECConferenceType\_Permanent(永久会议), * ecconferenceCondition.setConfType(ECConferenceEnums.ECConferenceType.*ECConferenceType\_Temporary*); * //可选 设置查询与自己相关的会议类型 1: 用户创建的会议，2:用户参与的会议，3:用户相关的会议（创建或者参与） * ecconferenceCondition.setSearchType(ECConferenceEnums.ECSearchType.*ABOUT*); * //设置查询过滤对象 * ECConferenceFilterInfo ecConferenceFilterInfo = new ECConferenceFilterInfo(); * //设置获取第几页的内容 * ecConferenceFilterInfo.setNumber(number); * //设置每页返回的会议个数 * ecConferenceFilterInfo.setSize(size); * /\*\* * \* 查询进行中的会议以及预约会议 * \* @param ECConferenceCondition 必选 查询条件对象 * \* @param ECConferenceFilterInfo 必选 查询过滤对象 * \* @param OnGetConferenceListWithCondition 200 成功 * \*/ * ecConferenceManager.getConferenceListWithCondition(ecconferenceCondition, ecConferenceFilterInfo, new ECConferenceManager.OnGetConferenceListWithCondition() { * @Override * public void OnGetConferenceListWithCondition(ECError ecError, List<ECConferenceInfo> list) { * LogUtil.*e*(*TAG*,"OnGetConferenceListWithCondition code is " + ecError.errorCode); * } * }); |

### 查询历史会议列表或指定历史会议

|  |
| --- |
| * ECConferenceManager ecConferenceManager = ECDevice.*getECConferenceManager*(); * //创建过滤规则 * ECConferenceFilterInfo ecConferenceFilterInfo = new ECConferenceFilterInfo(); * //设置获取第几页的内容 * ecConferenceFilterInfo.setNumber(number); * //设置每页返回的会议个数 * ecConferenceFilterInfo.setSize(size); * /\*\* * \* 获取历史会议 其中包括 已结束的会议和已取消的会议 * \* @param string 可选 createTimeBegin 会议创建时间起始点 * \* @param string 可选 createTimeEnd   会议创建时间终点 * \* @param int 可选 confType 0：临时会议；1：永久会议。默认值：0 * \* @param int 可选 searchByMember 1: 用户创建的会议，2:用户参与的会议，3:用户相关的会议（创建或者参与） * \* @param ECConferenceFilterInfo 必选 过滤条件 * \* @param string 可选 会议id   如果设置这个参数 说明 要查询指定的会议 * \* @param OnGetConferenceListWithCondition 200 成功 * \*/ * ecConferenceManager.getHistoryConferenceListWithCondition("", "", 0, 3, ecConferenceFilterInfo, confId, new ECConferenceManager.OnGetConferenceListWithCondition() { * @Override * public void OnGetConferenceListWithCondition(ECError ecError, List<ECConferenceInfo> list) { * LogUtil.*e*(*TAG*,"OnGetConferenceListWithCondition code is " + ecError.errorCode); * } * }); |

### 更新会议

|  |
| --- |
| * //获取confmgr * ECConferenceManager conferenceManager = ECDevice.*getECConferenceManager*(); * //创建ECConferenceInfo 对象 * ECConferenceInfo conferenceInfo = new ECConferenceInfo(); * //设置会议名字 * conferenceInfo.setConfName("会议name"); * //设置会议密码 * conferenceInfo.setPassword("password"); * //设置会议最大人数 * conferenceInfo.setMaxMember(*getMaxNum*()); * //设置会议的背景音模式 推荐设置ECConferenceVoiceMode\_All * conferenceInfo.setVoiceMode(*ECConferenceVoiceMode\_All*); * //设置会议的类型 * // ECConferenceType\_Temporary, 临时会议 * // ECConferenceType\_Permanent, 永久会议 * conferenceInfo.setConfType(*ECConferenceType\_Temporary*); * /\*\* * \* 更新会议 * \* 第一个参数为必选，其他均为可选 * \* @param ECConferenceInfo 会议相关信息 * \* @param int confMode 会议模式 * \* @param int autoClose 主持人离开会议后是否自动关闭会议 * \* @param int moderator 指定主持人ID * \* @param string startTime 开始时间 * \* @param string 邀请的成员json字符串 例如members:[{"memberId":"xxx","memberIdType":1}, {...}, ...] * \* @param int duration 会议持续时间，单位：分钟 默认值60 * \* @param int sendInvitation 到时间之后是否发送入会邀请（对于应用账号发送邀请通知，对于手机号发起邀请呼叫）0：否，1：是;默认值 1 * \* @param int sendReserveNote 启用预约的会议中使用这个字段，是否给会议成员发送提醒通知（提醒成员被列入预约会议成员） 0：否，1：是 默认值 0 * \* @param int remindBeforeStart 预约的会议中使用这个字段，会议开始前多久发送提醒通知，单位：分钟.默认值10 * \* @param int remindBeforeEnd  :预约的会议中使用这个字段，会议结束前多久发送提醒通知，单位：分钟默认值10 * \* @param int mediaType: 0:落地电话会议 1:多路视频会议 2:混屏视频会议 3:多路+混屏视频会议 * \* @param int reserveEnable: 是否启用预约功能,0:不启用，1：启用 * \* @param onUpdateConference 创建结果回调 200成功 * \*/ * conferenceManager.updateConference(conferenceInfo, confMode, autoClose, moderator, startTime,members.toString(), duration, -1, -1, -1, -1, -1, new ECConferenceManager.OnUpdateConferenceListener() { * @Override * public void onUpdateConference(ECError ecError) { * LogUtil.*e*(*TAG*,"onUpdateConference code is " + ecError.errorCode); * } * }); |

## 会议回调业务接口

### 配置接口

注意事项：sdk初始化成功后，就配置好接收会议相关通知的操作。

参考：[设置音视频会议回调监听](#_设置音视频会议回调监听)

### 接口中回调类型说明

#### 邀请通知

|  |
| --- |
| if (ecConferenceNotification instanceof ECConferenceInviteNotification) {  //类型转换操作  ECConferenceInviteNotification notification =  (ECConferenceInviteNotification) ecConferenceNotification;   * if (notification.callImmediately == 0) { * //一般是预约会议中邀请别人，收到邀请后 建议不立即加入会议，只发送通知 * } else{ * String confId = notification.getConferenceId(); * String pwd = notification.getPassword(); * //...... * //一般是立即会议中邀请，收到邀请后可以加入会议 * } * } |

#### 成员加入通知

注意事项：如果进行中的会议邀请别人，会立即收到此回调，但此时对象可能并未真正加入，判断被邀请者是否真正加入，是根据成员信息变更通知来判断，如果成员信息变更通知中携带的state表明该成员媒体在线，那么说明成员已加入会议，这时候再请求对方视频等操作。

|  |
| --- |
| if (ecConferenceNotification instanceof ECConferenceJoinNotification) {  //类型转换操作  ECConferenceJoinNotification joinNotification = (ECConferenceJoinNotification) ecConferenceNotification;  List<ECConferenceMemberInfo> membersList = joinNotification.getMembersList();   * } |

#### 成员退出通知

|  |
| --- |
| if (ecConferenceNotification instanceof ECConferenceQuitNotification) {  //类型转换操作  ECConferenceQuitNotification quitNotification = (ECConferenceQuitNotification) ecConferenceNotification;  ECAccountInfo member = quitNotification.member;} |

#### 会议解散通知

|  |
| --- |
| * if (ecConferenceNotification instanceof ECConferenceDeleteNotification) { * //会议被解散 * } |

#### 成员被移除通知

|  |
| --- |
| if (ecConferenceNotification instanceof ECConferenceKickOutNotification) {  // 成员被移除  //类型转换操作  ECConferenceKickOutNotification kickOutNotification = (ECConferenceKickOutNotification) ecConferenceNotification;   * List<ECConferenceMemberInfo> kickedMembers = kickOutNotification.kickedMembers; * } |

#### 成员信息变更通知

|  |
| --- |
| if (ecConferenceNotification instanceof ECConferenceMemberInfoNotification) {   * // 成员信息变更通知   //类型转换  ECConferenceMemberInfoNotification memberInfoNotification = (ECConferenceMemberInfoNotification) ecConferenceNotification;   * List<ECConferenceMemberInfo> members = memberInfoNotification.getMembers();   ECConferenceEnums.ECControlMediaAction action = memberInfoNotification.getAction();  *//通过获取通知里的action判断成员当前状态，参考[ECControlMediaAction](#_ECControlMediaAction)*   * *//通过成员的state来判断成员状态，参考[ECConferenceMemberInfo(成员信息对象)](#_ECConferenceMemberInfo(成员信息对象))* * } |

#### 媒体控制通知

被静音，禁言禁听等，会收到此通知。

|  |
| --- |
| if (ecConferenceNotification instance of ECConferenceMediaControlNotification) {   * ECConferenceMediaControlNotification info = (ECConferenceMediaControlNotification) ecConferenceNotification; * ECConferenceEnums.ECControlMediaAction action = info.action; * } |

#### 邀请成员结果通知

|  |
| --- |
| if (ecConferenceNotification instanceof ECConferenceInviteResultNotification) {   * // 成员拒绝邀请加入会议请求 * ECConferenceInviteResultNotification inviteResultNotification = (ECConferenceInviteResultNotification) ecConferenceNotification; * String result = inviteResultNotification.getResult();   } |

拒绝邀请或者未接通的情况下会收到此通知。具体可以看代码：

|  |
| --- |
| * private static String switchMemberRejectType(NetMeetingMember member, String result) { * if (TextUtil.isEmpty(result)) return null;  String tip;  switch (result) {  case "107770":  member.type = NetMeetingMember.Type.POWER\_OFF;  tip = "已关机";  break;  case "107771":  member.type = NetMeetingMember.Type.HALT;  tip = "已停机";  break;  case "107773":  member.type = NetMeetingMember.Type.BUSY;  tip = "正在通话中";  break;  case "107781":  member.type = NetMeetingMember.Type.REJECT\_INVITE;  tip = "拒绝了邀请";  break;  case "107775":  member.type = NetMeetingMember.Type.MISS\_CALLS;  tip = "未接听电话";  break;  case "107774":  member.type = NetMeetingMember.Type.ERROR\_NUM;  tip = "无效号码";  break;  default:  member.type = NetMeetingMember.Type.UNABLE\_CONNECT;  tip = "无法接通";  break;  } |

#### 会议信息更新通知

会议信息发生变化时会收到此通知。比如会议锁定状态发生改变lockConference、会议议题发生变化、会议全员静音状态发生改变。

|  |
| --- |
| if (ecConferenceNotification instance of ECConferenceUpdateNotification) {  ECConferenceUpdateNotification notification = (ECConferenceUpdateNotification) ecConferenceNotification;   * int action = notification.getAction(); * } |

# 登出及注销

## 登出

|  |
| --- |
| * ECDevice.logout(new ECDevice.OnLogoutListener() { * @Override * public void onLogout() { * // SDK 回调通知当前登出成功 * // 这里可以做一些（与云通讯IM相关的）应用资源的释放工作 * // 如（关闭数据库，释放界面资源和跳转等） * }}) |

|  |
| --- |
| * // V5.1.8r版本开始，增加退出登录新消息提醒接口 * /\*\* * \* 根据设置的{@link NotifyMode} 类型注销SDK，SDK会根据应用设置的是否接受消息提醒属性来提醒接收到的Push消息。 * \* 1、如果应用设置了{@link NotifyMode#NOT\_NOTIFY} 即表示应用注销SDK，不再接收新的消息提醒， * \* 调用此接口后SDK处于离线状态，此时可以再次调用注册接口{@link #login(ECInitParams)} * \* 进行登录，也可以调用{@link #unInitial()}接口释放SDK。 * \* 2、如果应用设置了{@link NotifyMode#IN\_NOTIFY} 即表示应用注销SDK，但是继续接收新的消息提醒， * \* 此时SDK处于在线状态，并且SDK的所有新消息不会再直接PUSH给应用，而是SDK直接采用状态栏通知的方式进行提醒， * \* 此时可以再次调用注册接口{@link #login(ECInitParams)} 进行登录，也可以调用{@link #unInitial()}接口释放SDK， * \* 并不会对SDK正常接收消息有任何影响 * \* 注意：如果应用调用了{@link #unInitial()}接口释放SDK,需要调用{@link #initial(Context, InitListener)}接口 * \* 重新初始化SDK，才能调用注册接口注册SDK，否则会提示SDK未初始化异常{@link SdkErrorCode#SDK\_NOT\_INIT} * \* @param notifyMode 消息提醒类型（是否接受消息Push） * \* @param listener SDK注销回调 * \* * \* @see #login(ECInitParams) * \* @see OnLogoutListener * \*/ * public static void logout(NotifyMode notifyMode , OnLogoutListener listener) * // 消息提醒规则可以通过如下接口设置： * ECNotifyOptions mOptions = new ECNotifyOptions(); * // 设置新消息是否提醒 * mOptions.setNewMsgNotify(true); * // 设置状态栏通知图标 * mOptions.setIcon(R.drawable.ic\_launcher); * // 设置是否启用勿扰模式（不会声音/震动提醒） * mOptions.setSilenceEnable(false); * // 设置勿扰模式时间段（开始小时/开始分钟-结束小时/结束分钟）// 小时采用24小时制// 如果设置勿扰模式不启用，则设置勿扰时间段无效// 当前设置晚上11点到第二天早上8点之间不提醒 * mOptions.setSilenceTime(23, 0, 8, 0); * // 设置是否震动提醒(如果处于免打扰模式则设置无效，没有震动) * mOptions.enableShake(true); * // 设置是否声音提醒(如果处于免打扰模式则设置无效，没有声音) * mOptions.enableSound(true); * // 设置退出登录后接收消息提醒规则 * ECDevice.setNotifyOptions(mOptions); * 说明: 注销SDk，调用此接口后SDK处于离线状态，此时可以再次调用注册接口void * login(ECInitParams inInitParams)进行登录，也可以调用unInitial()接口释放SDK。 |

## 释放

|  |
| --- |
| * ECDevice.unInitial(); * 说明:通过该方法释放云通讯SDK,方法释放后，SDK处于资源被释放状态、需要调用initial(Context, InitListener)重新初始化SDK。 |

# 日志打印及级别说明

## 日志打印级别说明

LOG\_LEVEL\_ERR    =10 ,//10以前预留给中间层日志

LOG\_LEVEL\_WARNING=11,// 潜在错误的情形日志

LOG\_LEVEL\_INFO=12, //普通信息日志

LOG\_LEVEL\_DEBUG=13,//普通级别信息日志

LOG\_LEVEL\_DEBUG\_WBSS=14,//白板日志开启级别

LOG\_LEVEL\_MEDIA\_ERR=20,//媒体库日志,打印ECmedia调用日志和媒体库Error日志；

LOG\_LEVEL\_MEDIA\_WARNING=21,//添加严重错误的日志；

LOG\_LEVEL\_MEDIA\_INFO=22,//添加警告日志；

LOG\_LEVEL\_MEDIA\_DEFAULT=23,//媒体库默认日志，一般打开这个；

LOG\_LEVEL\_MEDIA\_DEBUG=24,//添加调试信息；

LOG\_LEVEL\_MEDIA\_ALL=25,//大于24： 所有日志都打开，级别最好，把媒体流的信息都打印出来

LOG\_LEVEL\_END=99，//打印所有的，不建议开这个级别。

## 开启日志打印

Android中开启SDK日志和媒体库日志，在清单文件（AndroidManifest.xml）中的application节点内添加：

<meta-data

android:name="DEBUG\_LOG"

android:value="true" />

<meta-data

android:name="TRACE\_LOG"//打开媒体库日志

android:value="true" />

<meta-data

android:name="LOG\_LEVEL"

android:value="60"//最高99

/>

保存位置：

SDK日志：yuntongxun/log/ECSDK.log。媒体库日志：yunytongxun/log/ecmedia.log

java层日志:yuntongxun/log/[创建时间]...577335.log

注：对于新版本日志的打印如果一直开启状态可能会占用资源，为了避免影响客户正常业务，不建议客户一直将日志开启状态。