互动直播

常见问题

产品文档

腾讯云

文档目录

常见问题
如何切换分辨率码率

最近更新时间：2018-07-19 16:42:34

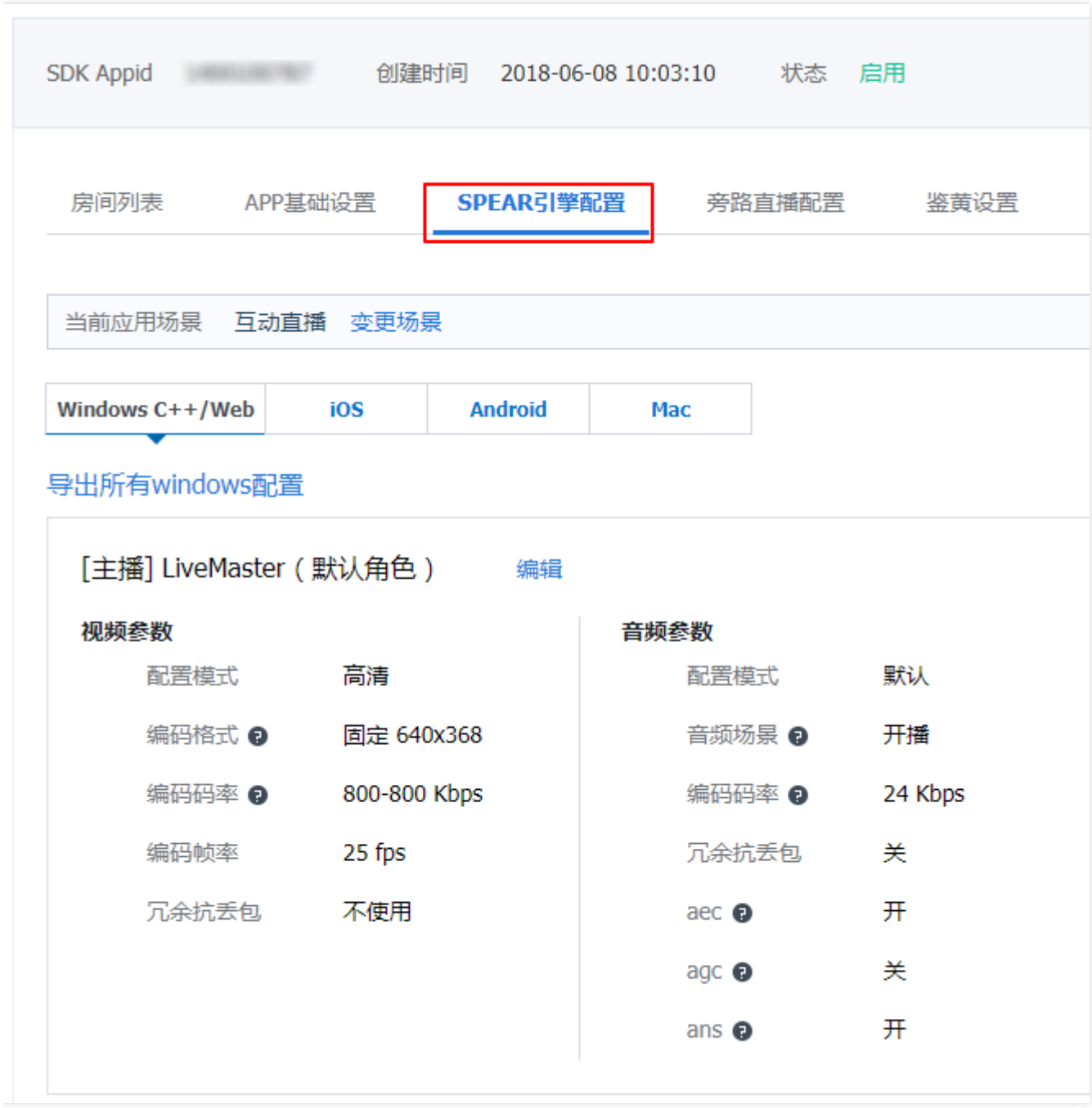
SPEAR 引擎流控配置简介

在进行实际开发前，首先要在腾讯云互动直播配置流控参数，才能获得更好的视频效果。

入口

登录腾讯云，进入 互动直播控制台 配置页面。选择对应的应用名称。

单击【SPEAR 引擎配置】，进入参数配置。



配置说明

场景

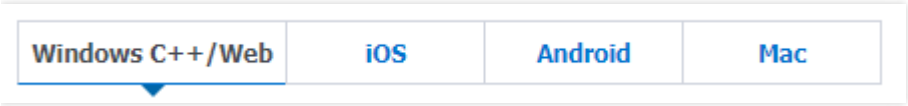
互动直播的配置首先分为不同的场景，每个场景保存自己的配置参数。一个 appid 可以设置一种对应的场景，当切换场景时，所有的音视频参数都会切换。

目前互动直播主要支持两种场景的业务：互动直播和实时通信。针对不同的场景，提供更适合业务需要的参数配置。

- **互动直播**：主要针对一个或少量几个主播在直播，其他大多数的观众观看的场景。只有一个或几个用户有上行数据，其他大多数用户（几百、几千或更多）只有下行数据，没有上行数据。
- **实时通信**：主要针对多人聊天或多人会议的场景。参与的用户都有上下行数据的需要，同时需要较高的实时性，延时要小。

平台

每个场景中包含不同平台的配置，支持根据不同的平台设置平台相关的不同参数。目前支持 Windows、Web、iOS、Android、Mac 五个平台，其中 Windows 和 Web 的参数配置是统一的。这样设置好后，Windows 客户端会自动使用 Windows 页面的配置，iOS 客户端会自动使用 iOS 页面的配置，以此类推。



角色

具体的某一平台的配置中，可以添加多个角色。角色实现了在单一平台上也可以配置不同的参数，即同一客户端可以通过切换角色来实现改变音视频配置的策略。当客户端没有选择角色时，会使用默认角色的配置。

可以在页面的最底部添加角色，角色名称设置角色名。角色可以编辑，自己添加的角色可以删除。角色可以根据业务需要添加多个。

场景、平台、角色的不同

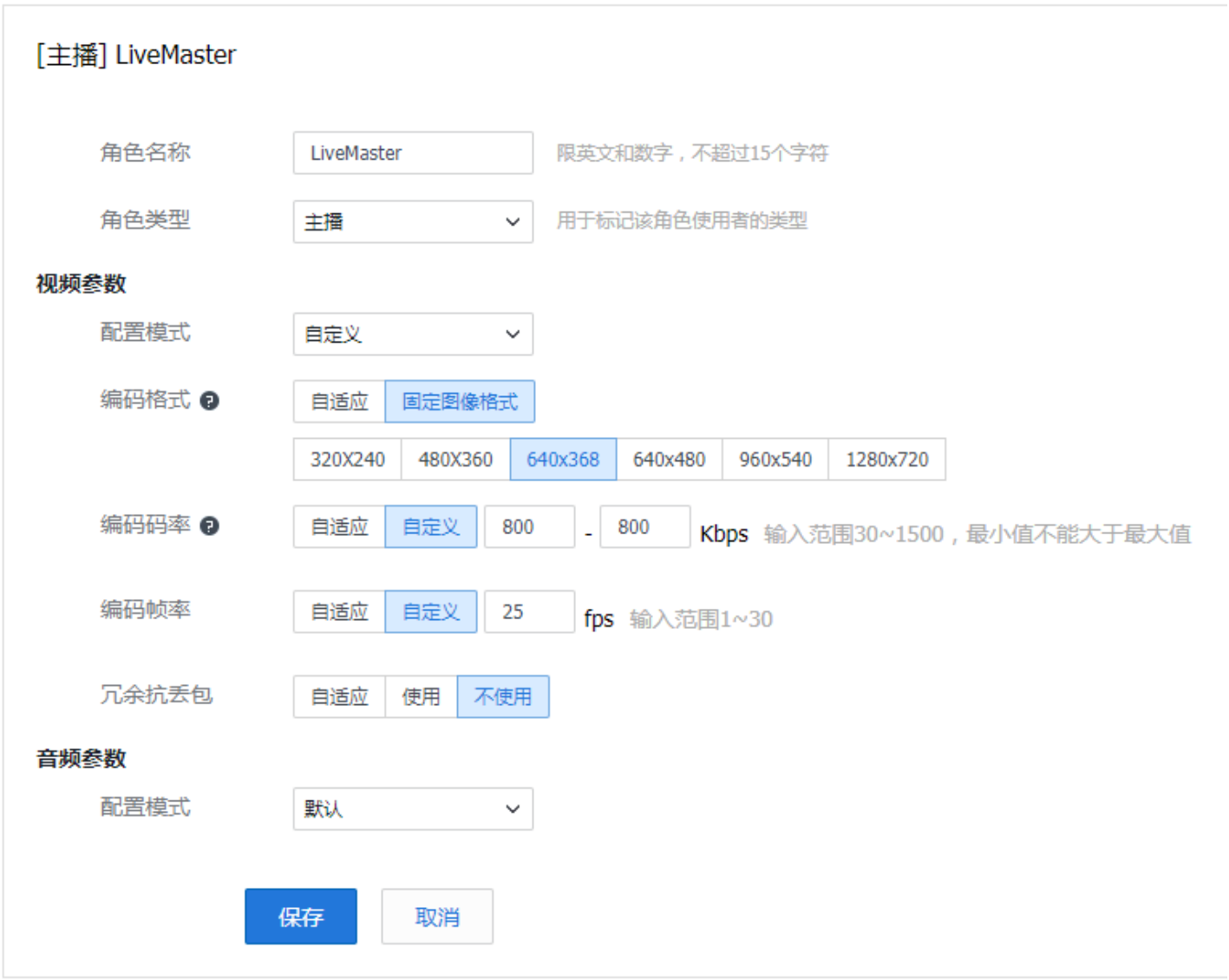
场景、平台、角色都可以用来设置不同的配置方案，他们之间有什么区别？

- **场景：** 是应用层面的不同，即一个 App 是主要用于直播还是实时通话。直播对清晰度流畅度要求高、延时要求低，实时通话对延时要求高；所以互动直播和实时通信的很多配置项都是不同的。一般一个 App 选择一个场景后，后续不会再改变。
- **平台：**主要处理平台相关的不同配置，比如 PC 的性能一般比手机高，手机上的 App 会有权限的要求的等等。
- **角色：**角色的区分主要是为了业务层面的支持。不同角色可设置的配置项和取值范围都是一样的，配置的不同完全由业务决定。比如，作为主播时使用可以上行数据的配置，作为观众时使用只有下行数据的配置；性能高的机器上使用分辨率更高的配置等等。

参数配置

参数配置分为视频参数、音频参数、网络参数三部分。在互动直播场景下，视频参数和大部分音频参数（下图中的参数）主要应用于上行数据，即只对主播起作用，对观众不起作用；网络参数应用于上下行数据，对主播、观众都起作用。实时通信场景，所有用户都有上下行数据，所以三部分参数对所有用户都起作用。

视频参数：



- **配置模式**：提供预设的【标清】、【高清】、【超清】打包模式，选择后不需要再配置其他参数。建议选择【高清】。选择【自定义】模式，可以自己配置其他参数
- **编码格式**：【自适应】由 SDK 根据设备性能和网络情况自行调整分辨率；【固定图像格式】选择一个固定的分辨率，提供 4:3 和 16:9 两种类型的分辨率。
- **编码码率**：【自适应】由 SDK 根据设备性能和网络情况自行调整码率；【自定义】指定 SDK 可以调整的范围，如果需要固定码率可以把最大最小值设置成一样的
- **编码帧率**：【自适应】由 SDK 根据设备性能和网络情况自行调整帧率；【自定义】指定一个期望的帧率。帧率和码率、清晰度相关，如果 MinQP、MaxQP 设置的不当，可能达不到期望的帧率。
- **冗余抗丢包**：通过 FEC 等方式增加冗余度来抵消网络丢包，冗余度一般为丢包率的两倍。一般情况下建议关闭。

帧率、码率、分辨率有一定的相关性，QP 设置的不合理也有可能会降低帧率。简单的用法可以直接选择预设的【标清】、【高清】、【超清】打包模式；自定义情况下，帧率、码率、分辨率的对应建议如下：

分辨率	码率	帧率
640 x 368	800Kbps	25fps
960 x 540	1200Kbps	15fps

音频参数：

配置模式：【默认】模式不需要再设置其他参数，【自定义】模式可设置更多参数。
音频场景：【开播】模式可以采集、编码、发送音频，适用于主播；【观看】模式不会打开音频设备也不会发送音频数据，适用于观众。
编码码率：设置音频码率，范围 0~64。
冗余抗丢包：通过 FEC 等方式增加冗余度来抵消网络丢包。互动直播场景建议关闭，实时通信场景建议打开。
aec：回声消除。互动直播不连麦的场景建议关闭，互动直播可能连麦场景和实时通信场景建议打开。
agc：自动增益。互动直播不连麦的场景建议关闭，互动直播可能连麦场景和实时通信场景建议打开。
ans：噪声抑制。互动直播不连麦的场景建议关闭，互动直播可能连麦场景和实时通信场景建议打开。

参数差异

不同平台、不同场景下的参数配置略有不同，根据页面显示的可配置选项配置即可。如实时通信场景，音频参数和网络参数是不能配置的。可配置参数可能会随着版本升级发生变化，以配置网页上显示的配置项为准。

客户端逻辑

配置生效时机

在腾讯云上配置好后，客户端不会立即生效。客户端需要重新登录时，去后台拉取最新配置。拉取成功后会在本地保存。

默认配置

SDK 自带一套保底默认配置，用户可以调用的 API 修改保底默认配置，设置的参数可以从云端配置网站获得，参数值为明文 JSON 字符串。SDK 使用配置时，如果云端配置第一次无法获取，则使用保底默认配置。其他情况仍走之前云端配置逻辑。

实时查看直播质量

最近更新时间：2018-06-19 12:00:57

AV_Monitor 概述

AV_Monitor 是帮助用户定位排除网络问题的实时监控工具。单击打开 AV_Monitor。

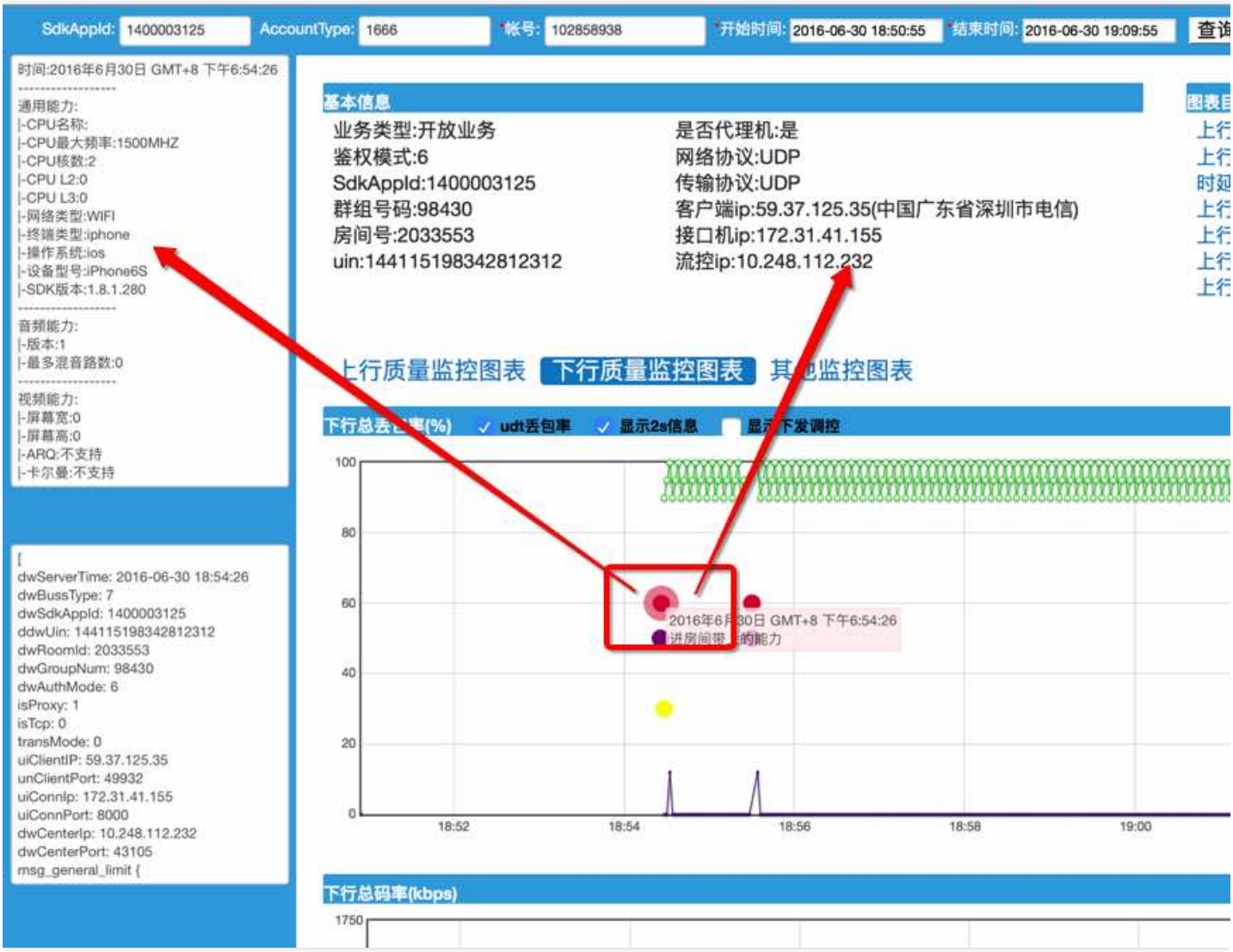
基本功能

界面最上方是查询条件录入文本框，SdkAppid、账号(Identifier)、查询时间段，都是必填项目。界面左边竖排的两个文本框，主要用于显示某些上报具体的取值，方便开发同事定位问题。例如，在下图中单击进入房间的红色点就可以在左侧文本框里看到一些基本信息，如 CPU、操作系统、网络类型、机型、SDK 版本等。

同样是这张图，在曲线上方显示地列出了所查询用户的一些常用的基本信息，其中需要关注以下几项：

- Tinyid，即基本信息左侧最下面的 uin:xxxxxx，这个有时候开发查问题需要用。
- 是否代理机，在 DC 上的用户，这里显示【否】；在 OC 上的用户，这里显示【是】。
- 传输协议，通常主播类型都是 UDT。
- 客户端 IP，可以看到用户侧的省份运营商。
- 接口机 IP，可以查看分配的接入机 IP 和客户端是否匹配，但目前只能显示内网 IP，需要腾讯开发在内部网站 `tnm2.oa.com` 查询。

最后，上图中最右边的图表目录是各种曲线图表的快捷入口，单击即达想要的位置。



下行质量监控图表

顾名思义，主要用于展示观众端的各种动作和状态，下面注意列举各图表展示的内容和使用场景。

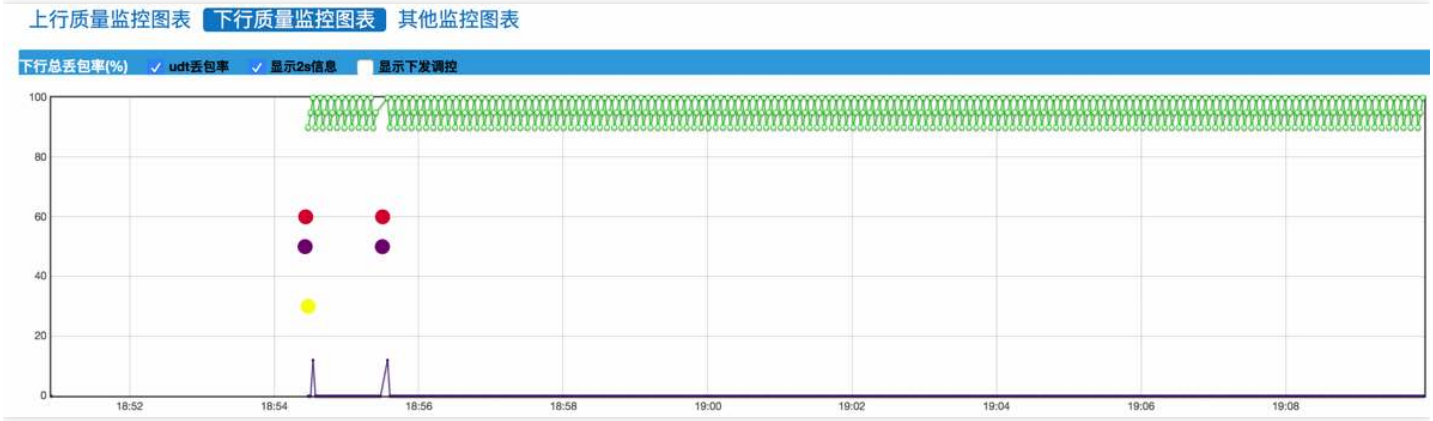
下行总丢包率

- **丢包率曲线：**最下面的紫色曲线是丢包率曲线，每 2s 一个点，反应的是用户当前的网络状况。高丢包率会导致音视频卡顿，而出现高丢包通常说明用户可能网络质量较差。例如，上图中丢包率曲线显示，在用户开播的时候有短时的丢包情况，而其他时间丢包率为 0，说明其整体网络状况良好

- **红色点**：红色点代表用户一次进房间的动作。红点是可以用鼠标点的，点了之后图表上方的基本信息和左侧的文本框里会刷新出该用户这次进房间时所携带的基本信息

- **蓝色点**：蓝色点代表用户一次退出房间的动作。由于用户可能不退出房间直接杀进程，所以蓝色点可能会比真实情况晚 90s（后台的超时时间）。鼠标放在蓝色点上会弹出提示信息，可以看到是正常退出还是超时退出

- **黄色点**：黄色点是说明用户在使用过程中 IP 发生了改变，极有可能发生了网络切换但又没退出房间



下行总码率

下行总码率=下行视频码率+下行音频码率（包含各自的 FEC）。单击下行总码率的任意一个点，都可以得到主播的 TinyID，而通过这个 TinyID 就可以查看主播的状态信息，不用再去找业务开发要主播 ID 了，事实证明这个功能非常的有用。

下行音频码率

顾名思义，需要注意的是，由于音频最高会有 100% 的 FEC，网络不好的时候和平时码率可能会相差一倍。

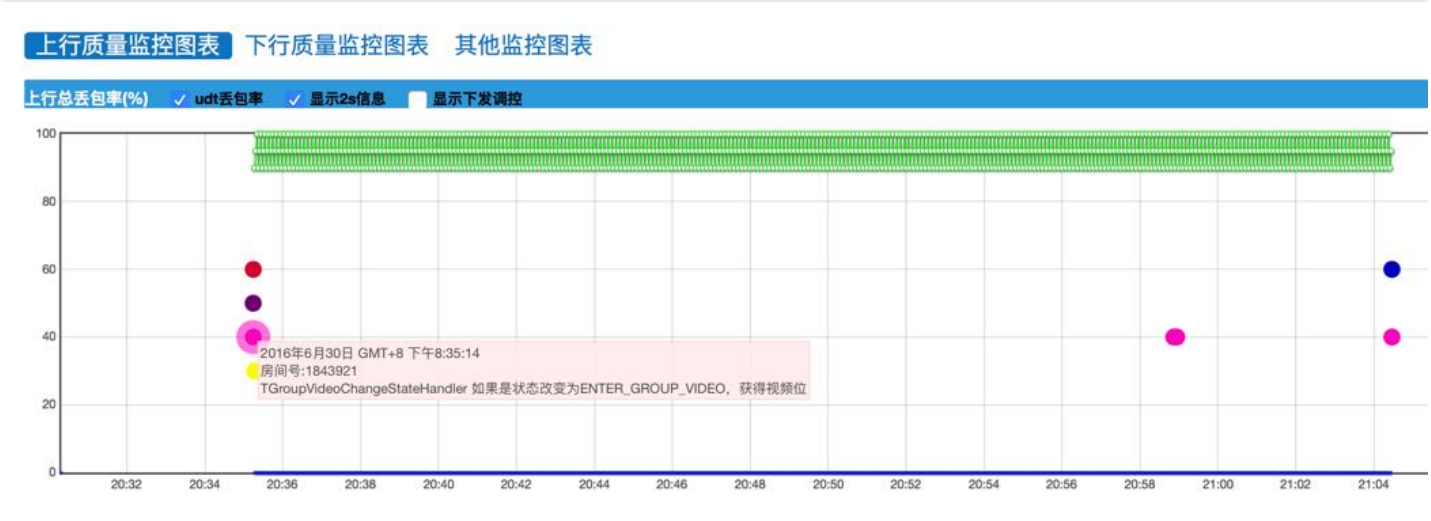
其他下行图标

- 下行大画面总帧率，顾名思义。
- 下行大画面总帧率，顾名思义，帧率越低，画面越不流畅。
- 下行辅路总帧率/总码率，屏幕分享的帧率码率。

上行质量监控图表

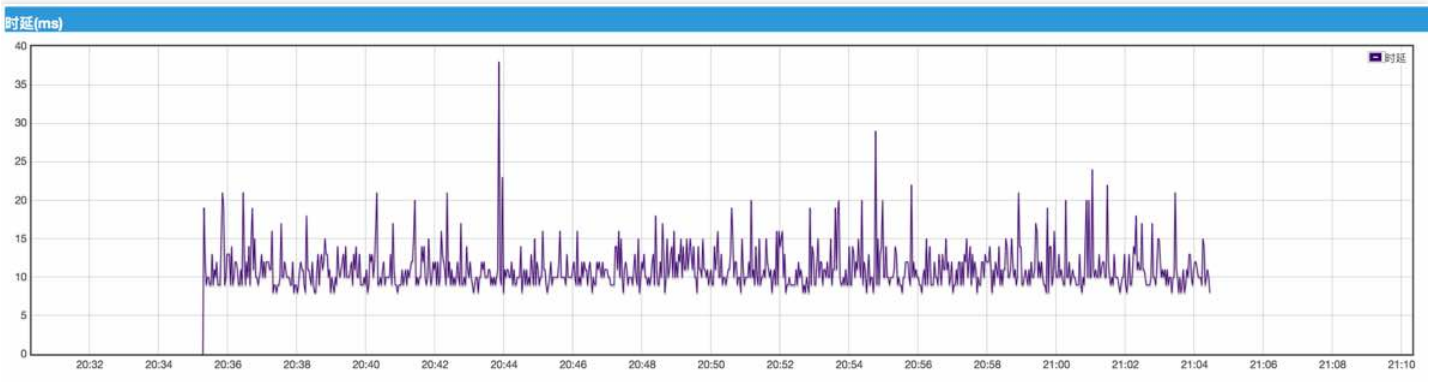
上行总丢包率

与下行丢包率视图相比，这里只多了一个粉色点，代表获取/清除视频位。这是主播独有的动作，即开播前需要向后台申请视频位成功，后台才会将该主播的视频流转给观众。若该主播 30s 内没有任何视频数据上行，那么后台将会清理视频位；如果主播后来仍要上行视频流，则需要重新申请视频位。



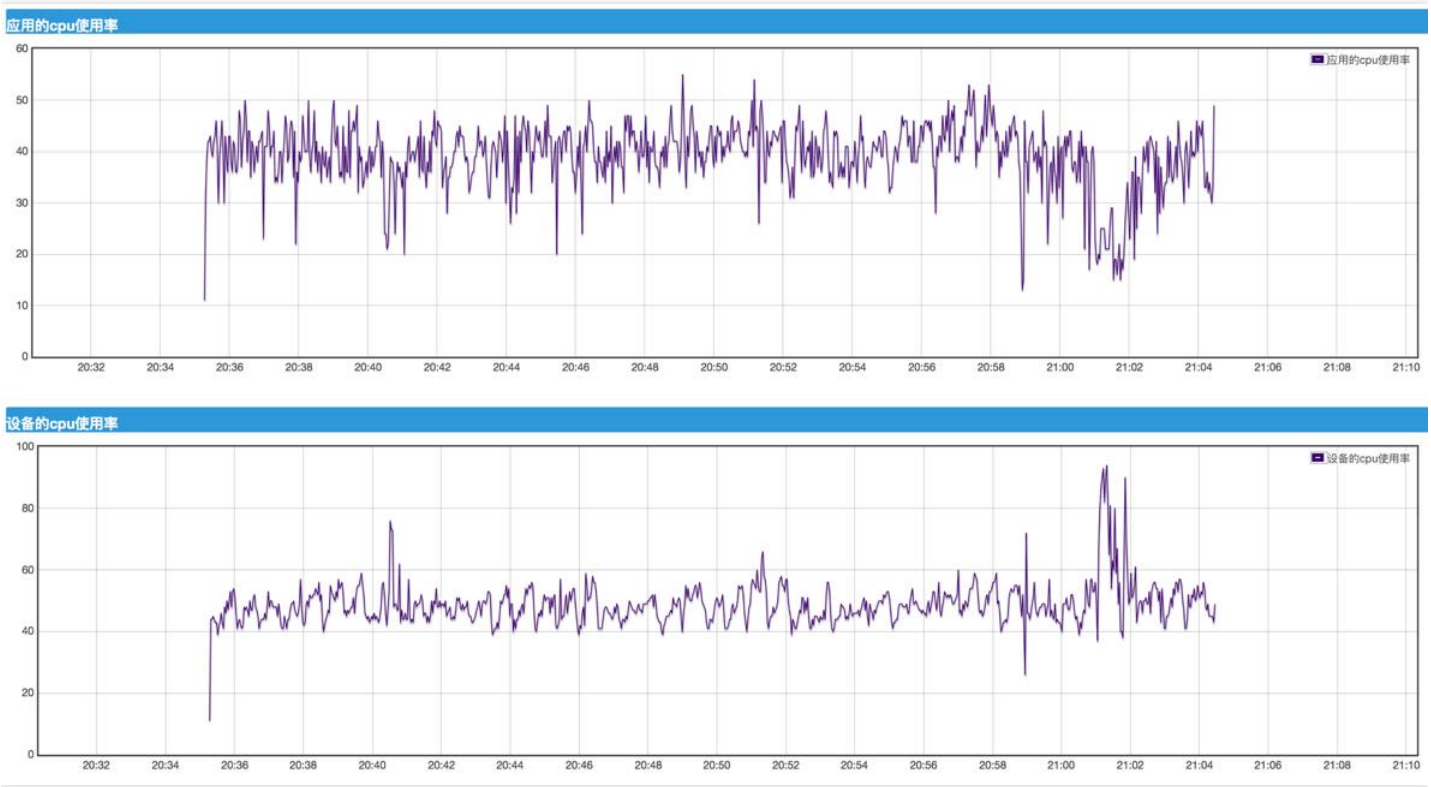
时延（ms）

时延，是指客户端到接口机 Hello 包的 RTT。通常网络变差时延时就会变大，造成视频卡顿。需要注意的是，有时候 CPU 高也会引起延时变大，这是由于 CPU 高占用使得 Hello 包的处理进程无法正常处理回包导致的。



其他监控图表

应用的 CPU 使用率是指 SDK 的 CPU 占用情况，设备的 CPU 使用率是指 App 整体占用情况。需要注意的是，最近 iPhone 遇到某些场景下 CPU 降频导致 CPU 整体占用过高的问题，需要具体情况具体分析。



互动直播音视频后台 Q&A

关于“房间”的认知

对于互动直播音视频后台而言，房间主要用于实现用户群体的隔离，用户在不同的房间可以看到不同主播的视频流，这个和现实中秀场的概念是想通的。房间是用 房间 ID 标识的，类型为 32 位整形数字，由**业务方生成和管理**，不同的房间应该使用不同的 房间 ID 。

> **注意：**
> 音视频的 房间 ID 与 IM 群组 ID 是没有任何关联的，但通常业务在使用过程中其实是将**音视频房间**和 **IM 群组**做了一一对应的，即可以用 IM 群组 ID 作为音视频 房间 ID 或在二者之间建立映射，这个都是业务方自己根据实际需要来决策

自己采集音视频输入

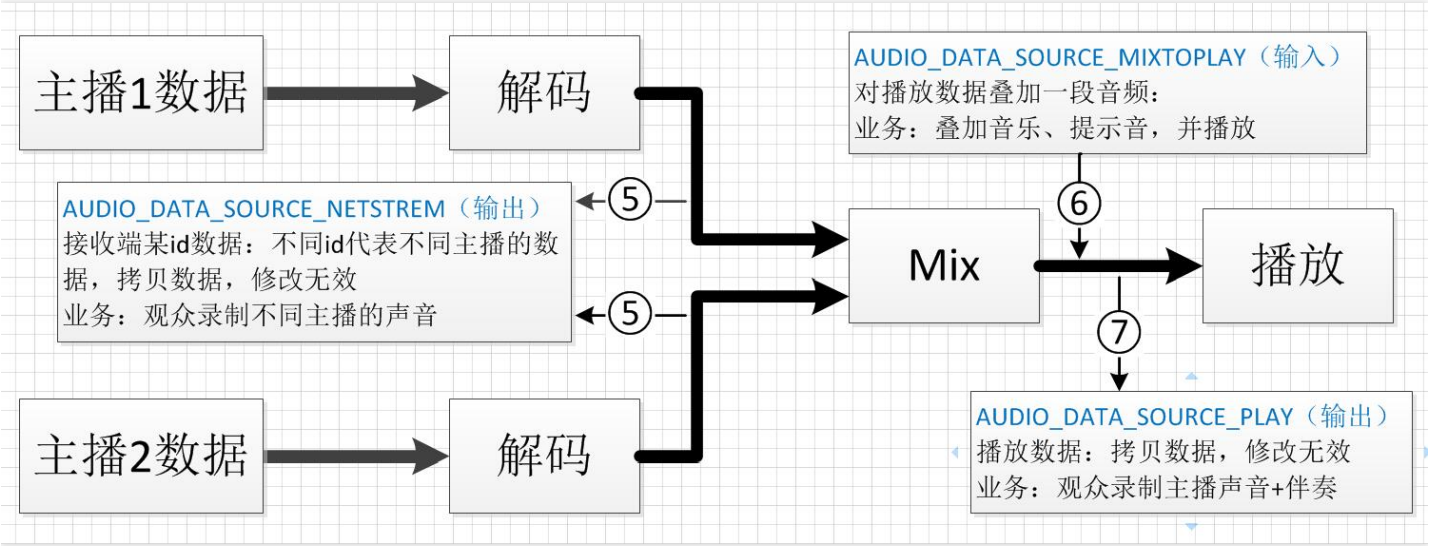
最近更新时间：2018-06-15 18:35:46

自定义音频数据

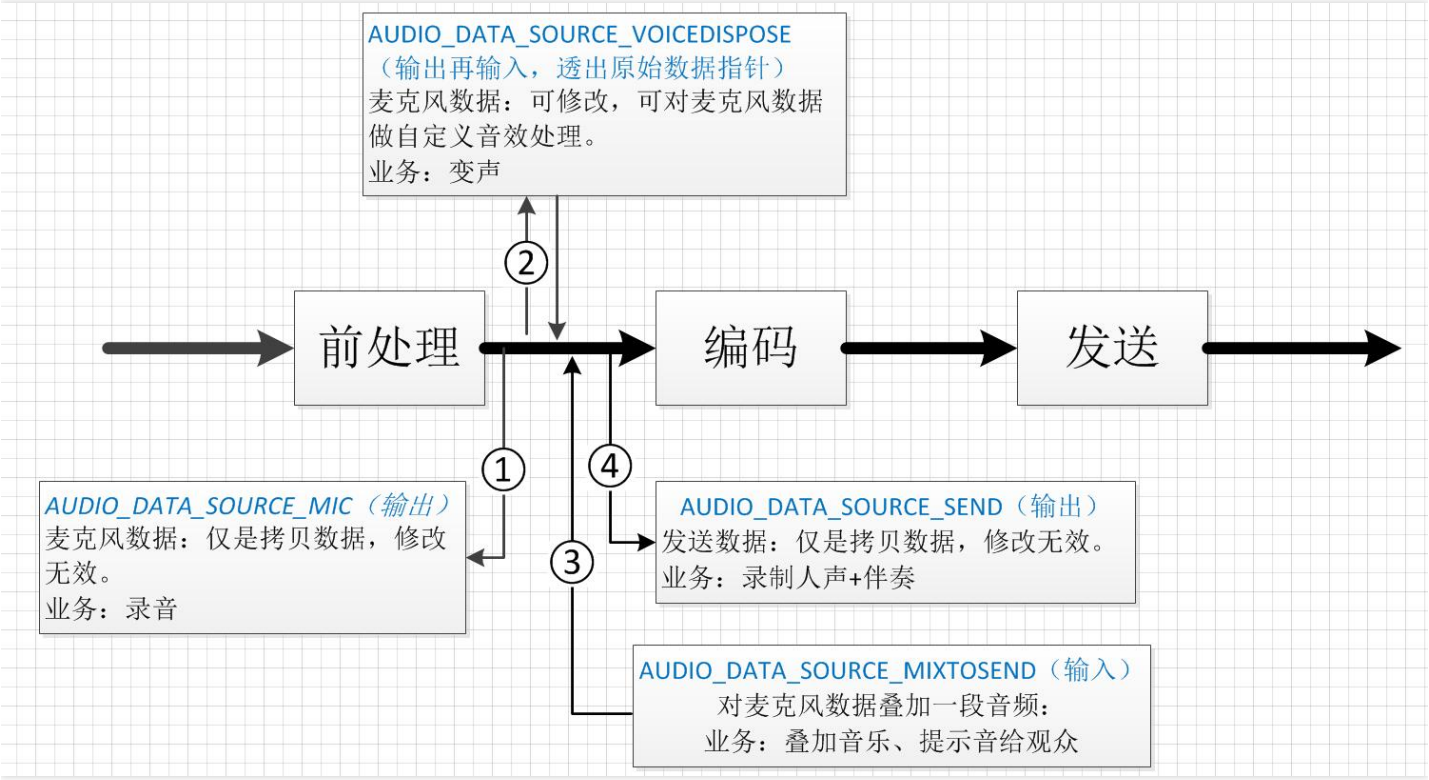
音频本地处理流程

音频本地处理流程图，用户可以其中任意环节对数据进行拦截并作相应的处理。

观众侧：



主播侧：



自定义音频采集流程

1. 主播或上麦观众腾讯云后台 spear 角色配置里音频场景要设置为【开播】（开播场景会占用本地录音权限），也可以设置为【观看】这样就不会占用本地音频设备了；
2. 进房间时 mic 和 speaker 都要打开；
3. 进房间成功后调用接口 AVAudioCtrl.changeAudioCategory 切换至观看场景（第一步如果设置为【观看】场景此步省略）；
4. 调 enableExternalAudioDataInput 开启自定义采集音频；
5. 调 fillExternalAudioFrame 将外部采集的音频塞给 AVSDK；
6. 以上接口都需在主线程调用。

注：avsdk1.8.4 后增加了此功能，目前的做法还需这样。略显繁琐，今后会进一步优化。

Android 混音

注册回调：

接口名	接口描述
registAudioDataCallbackWithByteBuffer	注册具体数据类型的回调函数

参数类型	说明
int	音频数据类型(参考上图)
RegistAudioDataCompleteCallbackWithByteBuffer	指向 App 定义的音频数据回调函数

```
ILiveSDK.getInstance().getAvAudioCtrl().registAudioDataCallbackWithByteBuffer(
    AVAudioCtrl.AudioDataSourceType.AUDIO_DATA_SOURCE_MIXTOSEND, mAudioDataCompleteCallb
```

添加需要混入的音频数据：

```
private AVAudioCtrl.RegistAudioDataCompleteCallbackWithByteBuffer mAudioDataCompleteCallbackW
    new AVAudioCtrl.RegistAudioDataCompleteCallbackWithByteBuffer() {
      @Override
      public int onComplete(AVAudioCtrl.AudioFrameWithByteBuffer audioFrameWithByteBuffer, int srcT
        if (srcType==AudioDataSourceType.AUDIO_DATA_SOURCE_MIXTOSEND) {
          synchronized (obj){
            /***********************************************
             将要混入的音频数据写入 audioFrameWithByteBuffer 中
            ***********************************************/
          }
        }
        return AVError.AV_OK;
      }
    };
```

iOS 混音

音频透传，主要用于在直播中对 Mic 采集到的数据作再加工处理，一般用于在直播间内添加背景音等，其对透传的音频数据有格式要求，默认使用的音频格式为 `QAVAudioFrameDesc = {48000, 2, 16}` 。通常的有下面两种使用方法：麦克风透传、扬声器透传数据。

- **麦克风透传：**开麦克风端（有上行音频能力端）能听到，其他人可听到：以下代码为设置麦克风透传。

```
// 设置音频处理回调
[[[ILiveSDK getInstance] getAVContext].audioCtrl setAudioDataEventDelegate:self];
// 注意为 QAVAudioDataSource_MixToSend
[[[ILiveSDK getInstance] getAVContext].audioCtrl registerAudioDataCallback:QAVAudioDataSource_M
[[[ILiveSDK getInstance] getAVContext].audioCtrl setAudioDataFormat:QAVAudioDataSource_MixToS
```

- **扬声器透传数据：**开扬声器端配置，只有自己听到，其他人听不到：以下代码为设置扬声器透传。

```
// 设置音频处理回调
[[[ILiveSDK getInstance] getAVContext].audioCtrl setAudioDataEventDelegate:self];
// 注意为 QAVAudioDataSource_MixToPlay
[[[ILiveSDK getInstance] getAVContext].audioCtrl registerAudioDataCallback:QAVAudioDataSource_M
[[[ILiveSDK getInstance] getAVContext].audioCtrl setAudioDataFormat:QAVAudioDataSource_MixTo
```

回调处理：注意三个回调中的注释。

```
- (QAVResult)audioDataComes:(QAVAudioFrame *)audioFrame type:(QAVAudioDataSourceType)type
{
    // 主要用于保存直播中的音频数据
    return QAV_OK;
}
- (void)handle:(QAVAudioFrame **)frameRef withPCM:(NSData *)data offset:(NSInteger *)offset
{
    // 演示如何将透传的数据添加到 QAVAudioFrame
    const QAVAudioFrame *aFrame = *frameRef;
    NSInteger off = *offset;
    NSInteger length = [aFrame.buffer length];
    if (length)
    {
        NSMutableData *pdata = [NSMutableData data];
        const Byte *btyes = [data bytes];
        while (pdata.length < length)
        {
            if (off + length > data.length)
            {
                const Byte *byteOff = btyes + off;
                [pdata appendBytes:byteOff length:data.length - off];
                off = 0;
            }
            else
            {
                const Byte *byteOff = btyes + off;
                [pdata appendBytes:byteOff length:length];
                off += length;
            }
        }
        if (pdata.length == length)
        {
            *offset = off;
```

```
            const void *abbytes = [aFrame.buffer bytes];
            memcpy((void *)abbytes, [pdata bytes], length);
        }
    }
}
- (QAVResult)audioDataShouInput:(QAVAudioFrame *)audioFrame type:(QAVAudioDataSourceType)typ
{
    // 混音输入（Mic 和 Speaker）的主要回调

    // 麦克风透传处理
    if (type == QAVAudioDataSource_MixToSend)
    {
        // self.micAudioTransmissionData 为要透传的音频数据，默认使用 QAVAudioFrameDesc = {48000, 2, 1
        if (self.micAudioTransmissionData)
        {
            NSInteger off = self.micAudioOffset;
            [self handle:&audioFrame withPCM:self.micAudioTransmissionData offset:&off];
            self.micAudioOffset = off;
        }
    }
    // 扬声明器透传处理
    else if (type == QAVAudioDataSource_MixToPlay)
    {
    // self.speakerAudioTransmissionData 为要透传的音频数据，默认同样使用 QAVAudioFrameDesc = {480
        if (self.speakerAudioTransmissionData)
        {
            NSInteger off = self.speakerAudioOffset;
            [self handle:&audioFrame withPCM:self.speakerAudioTransmissionData offset:&off];
            self.speakerAudioOffset = off;
        }
    }
//    NSLog(@"%@", audioFrame.buffer);
    return QAV_OK;
}

- (QAVResult)audioDataDispose:(QAVAudioFrame *)audioFrame type:(QAVAudioDataSourceType)type
{
    // 主要用作作变声处理
    return QAV_OK;
}
```

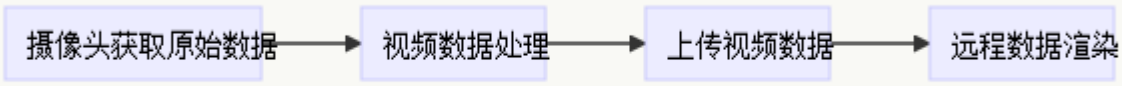
退出直播间，取消透传回调：

```
// 取消所有音频透传处理
[[[ILiveSDK getInstance] getAVContext].audioCtrl unregisterAudioDataCallbackAll];
```

```
[[[ILiveSDK getInstance] getAVContext].audioCtrl setAudioDataEventDelegate:nil];
// 或调用 AVSDK 接口取消不同类型的透传
// 方法详见 QAVSDK.framework 中的 QAVAudioCtrl
/*!
 @abstract     反注册音频数据类型的回调
 @discussion   要注册监听的音频数据源类型，具体参考 QAVAudioDataSourceType。
 @param        type          要反注册监听的音频数据源类型，具体参考 QAVAudioDataSourceType
 @return       成功返回 QAV_OK, 其他情况请参照 QAVResult。
 @see          QAVAudioDataSourceType QAVResult
 */
- (QAVResult)unregisterAudioDataCallback:(QAVAudioDataSourceType)type;
```

自定义视频采集

自定义视频流程图



Android

- 打开摄像头，同时需要调用 enableExternalCapture 做一些准备。
- 获取原始视频数据，加工处理。接收到系统回吐出的原始数据，用户就可以对其做预处理；比如美白，美颜，人脸识别等；预处理之后的画面需要用户自己完成渲染，与 ILiveSDK 无直接联系。

接口名	接口描述
enableExternalCapture	开启/关闭外部视频捕获设备

参数类型	说明
boolean	true 表示开启，false 表示关闭
boolean	true 表示开启本地渲染，false 表示关闭
EnableExternalCaptureCompleteCallback	指向 App 定义的回调函数

```
ILiveSDK.getInstance().getAvVideoCtrl().enableExternalCapture(false, true
    new AVVideoCtrl.EnableExternalCaptureCompleteCallback(){
@Override
protected void onComplete(boolean enable, int result) {
super.onComplete(enable, result);
}
});
```

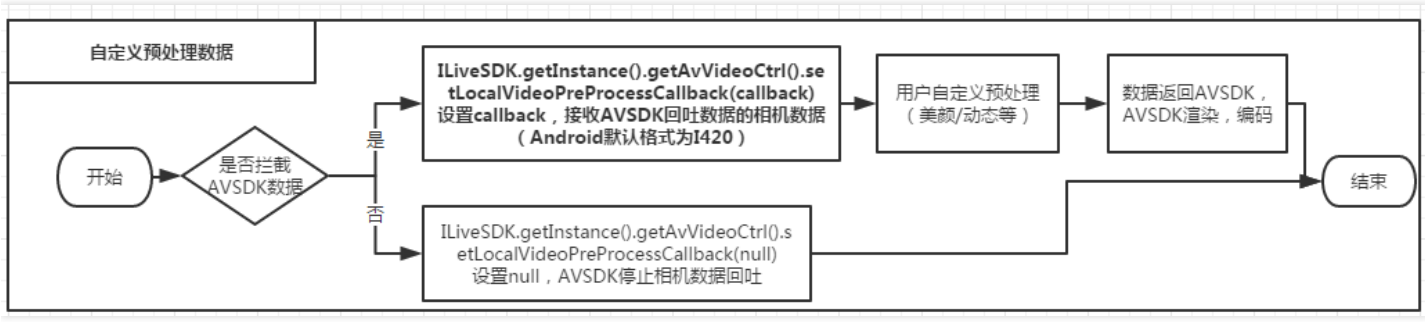
- 获取原始视频数据，加工处理
- 上传视频数据

接口名	接口描述
fillExternalCaptureFrame	输入从外部视频捕获设备获取的视频图像到 SDK

参数类型	说明
byte 数组	图像数据
int	图像数据长度
int	图像的 byteperRow。RGB32 图像专用，一般为宽度的 4 倍，特殊分辨率图像需要注意
int	图像宽度
int	图像高度
int	图像渲染角度。角度可以是 0、1、2、3,含义分别为图像需要顺时针旋转 0、90、180、270 度才能正立
int	图像颜色格式，具体值参考 EXTERNAL_FORMAT_I420、EXTERNAL_FORMAT_RGBA 等值
int	视频源类型。当前仅支持 VIDEO_SRC_TYPE_CAMERA

```
// 图像需要旋转 90 度
ILiveSDK.getInstance().getAvVideoCtrl().fillExternalCaptureFrame(data, data.length, 0,
    mCameraSize.width, mCameraSize.height, 3, AVVideoCtrl.COLOR_FORMAT_I420, AVView.VIDEO_SRC_T
```

自定义预处理视频数据：



- 拦截 AVSDK 相机数据

接口名	接口描述
setLocalVideoPreProcessCallback	设置本地摄像头视频的前处理函数

实现代码：

```
/*
函数说明：
a,预处理相机数据后，一般只需要将处理后的数据重新写回 VideoFrame.data ，并且需要保持视频数据格式和大小
b,其他的参数一般不需要修改

返回值：true： 成功    false: 失败
*/
boolean bRet = ILiveSDK.getInstance().getAvVideoCtrl().setLocalVideoPreProcessCallback(new AVVideoC

            @Override
            public void onFrameReceive(AVVideoCtrl.VideoFrame var1) {
                Log.d("SdkJni", "base class SetLocalPreProcessCallback.onFrameReceive");
            }
}
```

回调数据类型 AVVideoCtrl.VideoFrame 参数说明：

参数	含义
byte[] data	视频数据（a,目前 Android 只支持 I420 数据输出;b,对数据预处理后，数据仍然塞回 data；只要视频格式保持一致，AVSDK 就可以正常的预览和编码）
int dataLen	视频数据长度
int width	视频宽

参数	含义
int height	视频高
int rotate	视频图像角度。角度可以是 0、1、2、3,含义分别为图像需要顺时针旋转 0、90、180、270 度才能正立
int videoFormat	视频格式，目前支持 I420、NV21、NV12、RGB565、RGB24、ARGB；默认为 0
String identifier	房间成员 identifie
int srcType	视频采集来源 NONE = 0 CAMERA = 1 SCREEN = 2 MEDIA = 3
long timeStamp	视频帧的时间戳，SDK 内部会自动填写好，utc 时间，0 为无效值

- 预处理视频数据：拦截到 AVSDK 回吐的相机数据，用户就可以对其做预处理；比如美白，美颜，人脸识别等；预处理之后的数据，通过 AVVideoCtrl.VideoFrame.data 塞回 AVSDK；AVSDK 会自动渲染，编码。

- 取消拦截 AVSDK 相机数据

实现代码：

```
// 返回 true： 成功  false: 失败
boolean bRet = ILiveSDK.getInstance().getAvVideoCtrl().setLocalVideoPreProcessCallback(null);
```

iOS

自定义采集画面 主要用于预处理原始数据，比如用户需要人脸识别，画面美化，动效处理等，如下是通过自定义采集画面后，增加动效效果图。

> 注：动效效果是用户自己增加的，和本文档无关，用户可以对原始数据做任何预处理（不仅是动效）。

- **流程说明：**首先请记住，若要使用自定义采集画面，则采集画面的过程与 ILiveSDK 没有任何关系，完全不依赖 ILiveSDK，自定义采集画面的流程中，ILiveSDK 的作用是透传数据以及渲染远程数据。而本文介绍的流程是：**【自定义采集画面】>【画面传入 ILiveSDK】>【远程端收到画面帧渲染】** 的整个过程，流程图如下：

- **采集前准备：**进入房间之后，采集画面之前。

接口	描述
enableExternalCapture:	打开(关闭)外部视频捕获设备,自定义采集时，需要打开.返回 QAV_OK 表示执行成功。用了此方法之后，**不能**再调用 ILiveSDK 的打开摄像头接口，且 ILiveSDK 的美白，美颜将失效

参数类型	参数名	说明
BOOL	isEnableExternalCapture	是否打开外部视频捕获设备，自定义采集时传 YES

参数类型	参数名	说明
BOOL	shouldRender	SDK 是否渲染输入流视频数据，YES 表示会，NO 表示不会

示例：

```
QAVContext *context = [[ILiveSDK getInstance] getAVContext];
[context.videoCtrl enableExternalCapture:YES shouldRender:NO];
```

- **自定义采集：**自定义采集使用的是系统层级接口，和 ILiveSDK 没有直接联系，不做过多赘述，简单列下需要用到的系统类和方法。

系统类或方法	描述
AVCaptureSession	采集画面
AVCaptureDeviceInput	画面输入流
AVCaptureVideoDataOutput	画面输出流
captureOutput: didOutputSampleBuffer: fromConnection:	采集画面回调函数，采集到的画面将从这里回吐，用户可在这个接口作预处理

- **采集后处理：**接收到系统回吐出的原始数据（ `CMSampleBufferRef` 类型数据），用户就可以对其做预处理，比如美白，美颜，人脸识别等，预处理之后的画面需要用户自己完成渲染，与 ILiveSDK 无直接联系。

- **ILiveSDK 透传：**

接口	描述
fillExternalCaptureFrame:	向音视频 SDK 传入捕获的视频帧，返回 QAV_OK 表示执行成功

参数类型	参数名	说明
QAVVideoFrame	frame	AVSDK 画面帧类型，用户需将自定义采集的画面转换成 QAVVideoFrame 类型

示例：

```
QAVVideoCtrl *videoCtrl = [[ILiveSDK getInstance] getAvVideoCtrl].videoCtrl;
QAVResult result = [videoCtrl fillExternalCaptureFrame:frame];
```

- **远端渲染：**

接口	描述
OnVideoPreview:	远程画面回调，接收远程帧数据，再使用 ILiveSDK 的渲染接口渲染，用户只需要添加一个渲染区域即可。渲染详情请参考新版随心播

参数类型	参数名	说明
QAVVideoFrame	frameData	AVSDK 画面帧类型

注：

- 如果渲染自定义采集的画面使用了 OpenGL，则不能使用 ILiveSDK 中的渲染，否则会 Crash。也就是说，此时界面上应该有两个 view，一个渲染自定义采集的画面，另一个渲染 QAVVideoFrame 对象。
- 转换成 QAVVideoFrame 时，属性 color_format 必需填写 AVCOLOR_FORMAT_NV12，srcType 属性必须填写 QAVVIDEO_SRC_TYPE_CAMERA。

如何旋转和裁剪画面

最近更新时间：2018-06-15 18:35:51

问题

由于观众和主播的屏幕方向和大小都可能不一致，所以需要在观众端，按照观众的屏幕大小和方向对主播的画面进行选择，缩放或裁剪。

方案选择

从 1.2.0 版本起，iLiveSDK 已经封装了常见显示方案，开发者只需按自己的业务需要选择即可。

Android SDK：

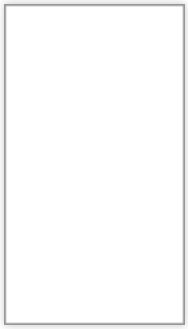
参数名/函数名	说明	默认值
setRotate	主播画面是否旋转	true（旋转）
setSameDirectionRenderMode	方向一致渲染模式	BaseVideoView.BaseRenderMode.SCALE_TO_FIT（全屏适应）
setDiffDirectionRenderMode	方向不一致渲染模式	BaseVideoView.BaseRenderMode.BLACK_TO_FILL（黑边）

iOS SDK：

参数名/函数名	说明	默认值
isRotate	主播画面是否旋转	YES（旋转）
sameDirectionRenderMode	方向一致渲染模式	ILIVERENDERMODE_SCALEASPECTFILL（全屏适应）
diffDirectionRenderMode	方向不一致渲染模式	ILIVERENDERMODE_SCALEASPECTFIT（黑边）

方案一 旋转主播画面

效果如下：

主播画面	观众屏幕	铺满屏幕， 不留黑边	画面大小一致	尽量显示， 可以留黑边
				
				

Android SDK 接口：

```
//设定需要旋转画面
AVVideoView.setRotate(true);
//设定是铺满屏幕还是留黑边(下为黑边)
AVVideoView.setSameDirectionRenderMode(BaseVideoView.BaseRenderMode.BLACK_TO_FILL);
```

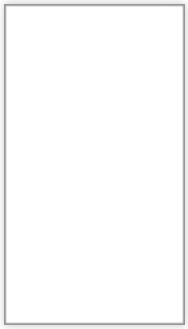
iOS SDK 接口：

```
//设定需要旋转画面
iLiveRenderView.isRotate = YES;
//设定是铺满屏幕还是留黑边
iLiveRenderView.sameDirectionRenderMode = ILiveRenderMode;
```

方案二 不旋转主播画面

效果如下：

主播画面	观众屏幕	铺满屏幕， 不留黑边	画面大小一致	尽量显示， 可以留黑边

主播画面	观众屏幕	铺满屏幕， 不留黑边	画面大小一致	尽量显示， 可以留黑边
				
				

Android SDK 接口：

```
//设定不需要旋转画面
AVVideoView.setRotate(false);
//设定在方向不一致情况下，是铺满屏幕还是留黑边(下为黑边)
AVVideoView.setDiffDirectionRenderMode(BaseVideoView.BaseRenderMode.BLACK_TO_FILL);
//设定在方向一致情况下，是铺满屏幕还是留黑边(下为全屏)
AVVideoView.setSameDirectionRenderMode(BaseVideoView.BaseRenderMode.SCALE_TO_FIT);
```

iOS SDK 接口：

```
//设定不需要旋转画面
iLiveRenderView.isRotate = NO;
//设定在方向不一致情况下，是铺满屏幕还是留黑边
iLiveRenderView.diffDirectionRenderMode = ILiveRenderMode;
//设定在方向一致情况下，是铺满屏幕还是留黑边
iLiveRenderView.sameDirectionRenderMode = ILiveRenderMode;
```

直播中断事件的处理

最近更新时间：2018-06-15 18:35:55

概述

在直播过程中，经常会遇到突发事件导致直播被中断。这里列举一些常见事件的处理方式供开发者参考：

- 来电话
- 音频中断
- 切后台
- 锁屏
- 断网
- crash
- 被踢

来电话

无需做任何处理。来电话时，来电界面会覆盖当前直播界面，参考切后台处理。

音频中断

无需做任何处理。发产生音频中断（如闹钟事件）时，分两种情况：

- 无新界面，不影响当前播放(录制)视频及语音
- 有新界面，参考切后台处理

切后台

Android

Android 应用在切后台时，ILiveSDK 中为用户提供了三种模式：

模式名称	模式说明
VIDEOMODE_BSUPPORT	支持后台模式,应用切到后台时依然默默地播放(录制)视频和语音
VIDEOMODE_NORMAL	普通模式(*默认*)，应用切到后台时，暂停播放(录制)视频和语音，回到前台时恢复
VIDEOMODE_BMUTE	后台静默模式，应用切到后台，关闭所有音视频上下行数据，回到前台时恢复

iOS

在直播房间时，如果 90 秒无上行数据，音视频房间会被后台收回。iOS 分两种情况：

- 界面被覆盖，此时会暂停播放(录制)视频和语音，回到前面时恢复
- 切到后台，短时间只会暂停(同上)，长时间会被回收（系统机制）

锁屏

Android

同切后台，用户可以在应用中自行设置保活模式，避免锁屏。

iOS

ILiveSDK 内部设置保活模式，不会锁屏（进入房间保活，退出房间不保活）。

断网

在网络中断时，SDK 内部会尝试重连，用户可自行监控系统网络状态。如需了解 SDK 内部网络状态，可参考 ILiveQualityData。如房间超过 90 秒没有上行数据，音视频房间会被回收，会上抛 onRoomDisconnect 事件。

App crash

ILiveSDK 内置了 bugly 上报，可以方便用户迅速定位到问题。同时用户只需记录当前直播房间（roomid）及直播状态，在应用重启后，可以回到原直播房间，恢复直播。如时间较短（未超过 90 秒），观众端只会感到画面卡顿

（此时可以根据相应事件做出友好提示）。用户可以在监听 onEndpointsUpdateInfo 事件中的关摄像头事件做出友好提示。

被踢

多终端登录时，会收到被踢事件，此时建议用户重新登录（如果恢复需自己记录状态）。

Android

可以在 ILiveLoginManager 类调用 setUserStatusListener 监听强制下线事件通知。

```
public interface TILVBStatusListener {
    void onForceOffline(int error, String message);
}
```

iOS

设置用户状态监听：

```
//设置用户状态监听
[[ILiveSDK getInstance] setUserStatusListener:[[LiveUserStatusListener alloc] init]];
```

```
//用户状态监听类
@interface LiveUserStatusListener : NSObject <TIMUserStatusListener>
@end

@implementation LiveUserStatusListener
//被踢
- (void)onForceOffline{
}
//票据过期
- (void)onUserSigExpired{
}
```

回调中处理业务逻辑：

```
- (void)onForceOffline{
    //被踢下线（以下处理逻辑可做参考，具体可由自身业务决定）
    //1、如果未处于直播间，可跳转到用户登录界面
    //2、如果处于直播间
    //（1）保存直播状态信息，如房间 ID，房间标题等并退出直播间
```

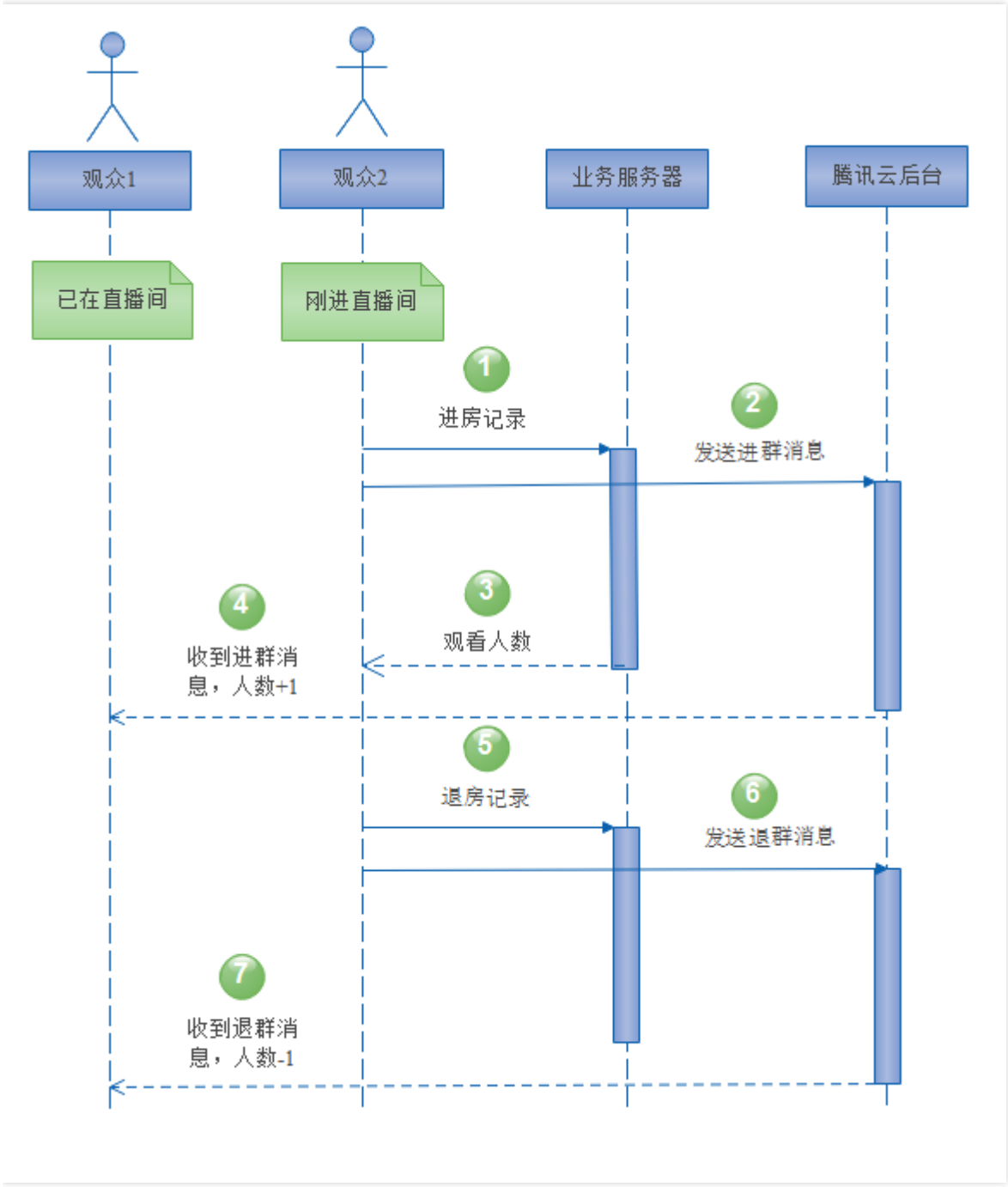
```
    //（2）重新登录后，根据直播状态信息恢复（进入）直播间
}
```

查看在线人数

最近更新时间：2018-06-15 18:35:58

方案一：业务服务器管理

流程图



步骤描述

1. 观众 2 调用业务服务器进房接口生成进房记录。
2. 观众 2 调用 imsdk 接口发送进群自定义消息，可与步骤 1 同步进行。
3. 观众 2 生成进房记录同时，业务服务器返回当前房间人数。
4. 观众 1 收到观众 2 进群自定义消息，在线人数 + 1。
5. 观众 2 调用业务服务器退房接口生成退房记录。
6. 观众 2 调用 imsdk 接口发送退群自定义消息，可与步骤 5 同步进行。
7. 观众 1 收到观众 2 退群自定义消息，在线人数 - 1。

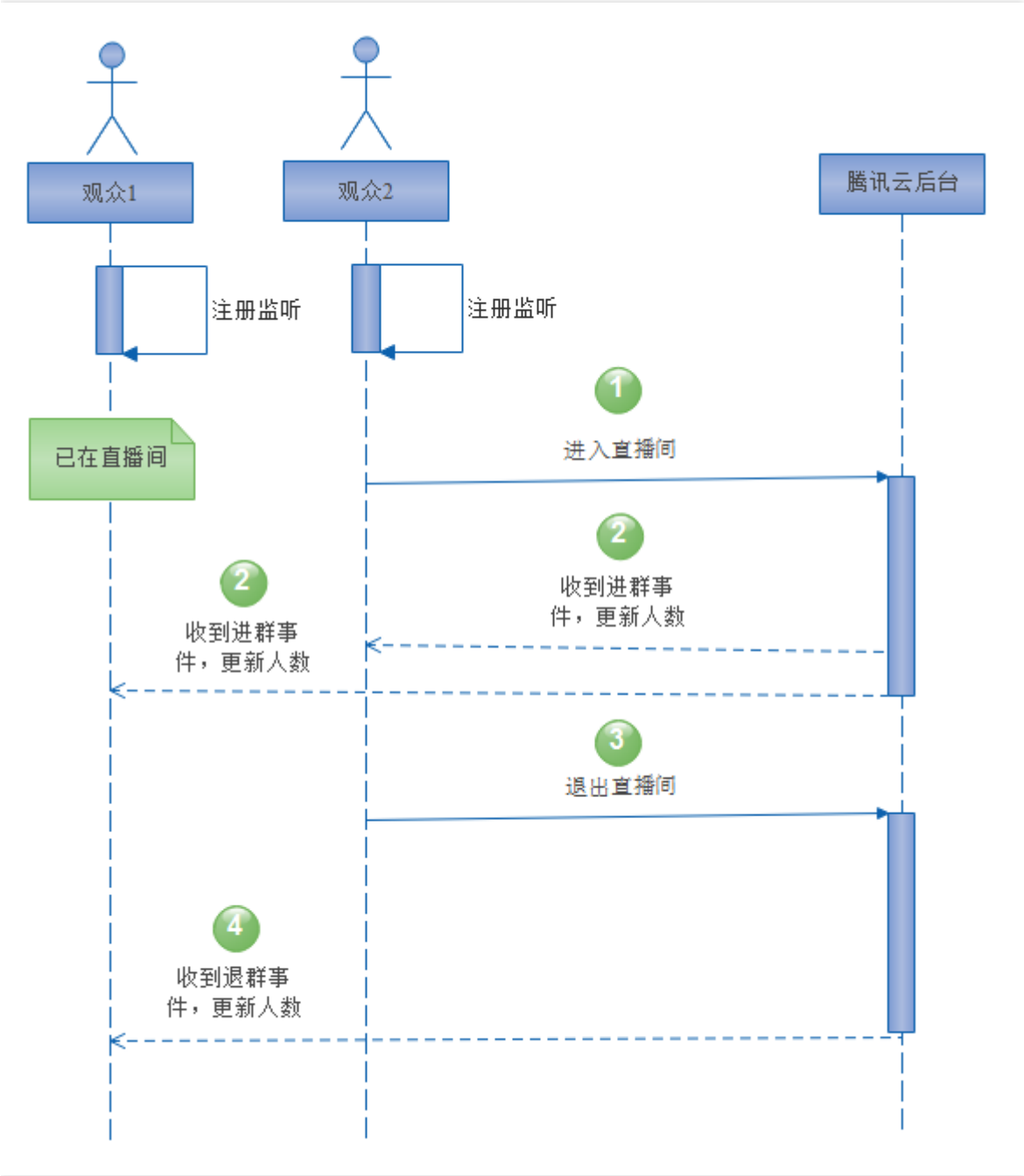
方案二：使用 IMSDK 接口

> **注意：**
> ILiveSDK 暂未提供该方案，推荐使用方案一。

通过 IMSDK 接口获取直播间在线人数主要有两种方式，详情参阅 获取观看直播的人数。

- AVChatRoom 通过群组资料获取人数
  - 不是实时的，有 1 分钟左右的时延
  - 只返回人数
- AVChatRoom 注册 TIMGroupEventListener 接口收到进群、退群事件时获取人数
  - imsdk2.4 支持
  - 统计结果存在时延
  - 只返回人数
  - 人数较少场景（比如只有几十人），统计结果偏差明显

流程图



步骤描述

1. 观众 2 进入直播。
2. 观众 1 和观众 2 收到观众 2 的进群事件通知，使用附带的人数字段 memberNum 更新在线人数。
3. 观众 2 退出直播。
4. 观众 1 收到观众 2 的退群事件通知，使用附带的人数字段 memberNum 更新在线人数。

播放背景音乐

最近更新时间：2018-06-15 18:36:03

需求背景

主播使用第三方音乐 App 播放背景音乐，自己收听的同时也希望观众一起畅听。

iOS

只需要**主播端**开启高音质即可，观众和上麦者的配置无需改变。

```
//进入房间时开启高音质配置
ILiveRoomOption *option = [ILiveRoomOption defaultHostLiveOption];
option.avOption.autoHdAudio = YES;
```

Android

Android 无需配置即可支持。

注意：

- 必须先进入房间然后才打开音乐 App 开启背景音乐，否则直播间会中断背景音乐。
- 主播和上麦观众的后台 spear 配置需要开启 aec（回声消除）。

画面对焦

最近更新时间：2018-06-15 18:36:07

自动对焦

iOS 自动对焦

iLiveSDK 在 iOS 中已提供自动对焦功能，用户无需任何配置。

Android 自动对焦

iLiveSDK 在 Android 中也提供了自动对焦功能，需要用户手动开启，在进入房间（发起呼叫）时的 option 配置。Android 的对焦原理是，基于运动传感器事件，在手机发生移动时，对手机中心进行对焦操作。

```
ILiveRoomOption option = new ILiveRoomOption(strHostId)
        .autoFocus(true);      // 开启自动对焦
```

手动对焦

> **注：当前只支持后置摄像头手动对焦。**

如果需要对自带对焦效果不满意，或有高级应用场景需求，用户可以自己完成对焦功能，这里以实现手动对焦为例。

iOS 手动对焦

流程如下：

- **单击事件：**因为交互界面在最顶层，渲染界面在最底层，所以单击事件添加到交互界面上。
- **获取单击点坐标：**获取单击手势在视图上的坐标，此坐标是相对于交互视图的坐标。

- **将单击手势坐标转换为 layer 坐标：**【获取单击点坐标】获取的是相对于交互视图的坐标，要转换为画面渲染视图的坐标，将交互视图和渲染视图想对的屏幕的坐标同时计算出来，即可将交互视图坐标映射到渲染视图。

转换函数：

```
//功能：将交互视图上的点映射成渲染视图的点
//本 Demo 只实现了全屏下的聚焦和缩放功能，所以在转换时使用的 liveViewController.livePreview.imageVie
- (CGPoint)layerPointOfInterestForPoint:(CGPoint)point
{
    FocusDemoViewController *liveViewController = (FocusDemoViewController *)_liveController;
    CGRect rect = [liveViewController.livePreview.imageView relativePositionTo:[UIApplication sharedAppl
    BOOL isContain = CGRectContainsPoint(rect, point);
    if (isContain)
    {
        CGFloat x = (point.x - rect.origin.x)/rect.size.width;
        CGFloat y = (point.y - rect.origin.y)/rect.size.height;
        CGPoint layerPoint = CGPointMake(x, y);
        return layerPoint;
    }
    return CGPointMake(0, 0);
}
```

- **获取 AVCaptureSession 并设置焦点：**通过 AVSDK 接口获取相机 session，通过此 session 设置相机焦点，见 Demo 中 onSingleTap 函数

iOS 手动缩放

```
//响应双击事件
- (void)onDoubleTap:(UITapGestureRecognizer *)tapGesture
{
    CGPoint point = [tapGesture locationInView:self.view];
    [_focusView.layer removeAllAnimations];
    __weak FocusDemoUIViewController *ws = self;
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(0.05 * NSEC_PER_SEC)), dispatch_get_ma
        [ws layoutFoucsView:point];
    });
    static BOOL isscale = YES;
    CGFloat rate = isscale ? 1.0 : -2.0;
    [ self zoomPreview:rate];
    isscale = !isscale;
}
```

```
//缩放
-(void)zoomPreview:(float)rate
{
    // 以下是获取 AVCaptureSession 演示摄像头缩放的。iphone4s 暂时不支持。
    if ([FocusDemoUIViewController isIphone4S:self])
    {
      return;
    }
    //to do
    QAVVideoCtrl *videoCtrl = [_roomEngine getVideoCtrl];
    AVCaptureSession *session = [videoCtrl getCaptureSession];
    if (session)
    {
       for( AVCaptureDeviceInput *input in session.inputs)
       {
          NSError* error = nil;
          AVCaptureDevice*device = input.device;
          if ( ![device hasMediaType:AVMediaTypeVideo] )
          continue;
          BOOL ret = [device lockForConfiguration:&error];
          if (error)
          {
             DebugLog(@"ret = %d",ret);
          }
          if (device.videoZoomFactor == 1.0)
          {
            CGFloat current = 2.0;
            if (current < device.activeFormat.videoMaxZoomFactor)
            {
              [device rampToVideoZoomFactor:current withRate:10];
            }
          }
          else
          {
             [device rampToVideoZoomFactor:1.0 withRate:10];
          }
          [device unlockForConfiguration];
          break;
       }
    }
}
```

Android 手动对焦

流程如下：

- 添加单击事件回调，如 setOnTouchListener。
- 获取单击坐标，如 MotionEvent。
- 获取 Camera 对象：

```
Camera camera = ILiveSDK.getInstance().getAvVideoCtrl().getCamera();
```

- 根据焦点来对焦

示例：

```
protected boolean onFocus(Point point, Camera.AutoFocusCallback callback) {
    if (camera == null) {
        return false;
    }
    Camera.Parameters parameters = null;
    try {
        parameters = camera.getParameters();
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    //不支持设置自定义聚焦，则使用自动聚焦，返回
    if(Build.VERSION.SDK_INT >= 14) {
        if (parameters.getMaxNumFocusAreas() <= 0) {
            return focus(camera, callback);
        }
        Log.i(TAG, "onCameraFocus:" + point.x + "," + point.y);
        List<Camera.Area> areas = new ArrayList<Camera.Area>();
        int left = point.x - 300;
        int top = point.y - 300;
        int right = point.x + 300;
        int bottom = point.y + 300;
```

```
            left = left < -1000 ? -1000 : left;
            top = top < -1000 ? -1000 : top;
            right = right > 1000 ? 1000 : right;
            bottom = bottom > 1000 ? 1000 : bottom;
            areas.add(new Camera.Area(new Rect(left, top, right, bottom), 100));
            parameters.setFocusAreas(areas);
            try {
                //兼容部分定制机型，在此捕捉异常，对实际聚焦效果没影响
                camera.setParameters(parameters);
            } catch (Exception e) {
                // TODO: handle exception
                e.printStackTrace();
                return false;
            }
        }
        return focus(camera, callback);
    }
    private boolean focus(Camera camera, Camera.AutoFocusCallback callback) {
        try {
            camera.autoFocus(callback);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

日志

最近更新时间：2018-06-25 09:56:28

日志位置

日志在开发控制台输出的同时，也会写入终端的存储中。开发者可以在这些地方找到日志。

Android

SDK	目录
iLiveSDK	tencent/imsdklogs/包名/ilivesdk_YYYYMMDD.log
IMSDK	tencent/imsdklogs/包名/imsdk_YYYYMMDD.log
AVSDK	tencent/imsdklogs/包名/QAVSDK_YYYYMMDD.log

iOS

SDK	目录
iLiveSDK	Library/Caches/ilivesdk_YYYMMDD.log
IMSDK	Library/Caches/imsdk_YYYMMDD.log
AVSDK	Library/Caches/QAVSDK_YYYMMDD.log

Mac

SDK	目录
iLiveSDK	/Users/**user**/Library/Caches/ilivesdk_YYYMMDD.log
IMSDK	/Users/**user**/Library/Caches/imsdk_YYYMMDD.log
AVSDK	/Users/**user**/Library/Caches/QAVSDK_YYYMMDD.log

- 注：user 是你自己电脑的登录账户，比如当前电脑的登录账户是 zhangsan，那么缓存路径是 /Users/zhangsan/Library/Caches

PC

SDK	目录
iLiveSDK	程序运行目录/txsdklog/ilivesdk_YYYMMDD.log
IMSDK	程序运行目录/txsdklog/imsdk_YYYMMDD.log
AVSDK	程序运行目录/txsdklog/QAVSDK_YYYMMDD.log

Web

SDK	目录
iLiveSDK	%appdata%/Tencent/iLiveSDK/txsdklog/ilivesdk_YYYMMDD.log （在开始菜单的运行中执行%appdata%可打开 appdata 目录）
IMSDK	%appdata%/Tencent/iLiveSDK/txsdklog/imsdk_YYYMMDD.log
AVSDK	%appdata%/Tencent/iLiveSDK/txsdklog/QAVSDK_YYYMMDD.log

日志收集接口

适用场景

任何需要从手机上收集互动直播日志的场景。

使用方法

集成 iLive SDK：**1.4.0 以上。**在需要用户上报日志的时候，调用日志上报接口。iLive SDK 会直接把日志传到腾讯云后台。

Android 接口：

```
/**
 * 上报日志
 *
 * @param desc 描述
 * @param data 0 表示当天 1-昨天 2-前天 类推
 * @param callBack 回调
 *
```

```
 */
ILiveSDK.getInstance().uploadLog(String desc, int data ILiveCallBack callback);
```

iOS 接口：

```
/**
 日志上报

 @param logDesc     日志描述
 @param dayOffset   日期，0-当天，1-昨天，2-前天，以此类推
 @param uploadResult 上报回调
 */
(void)uploadLog:(NSString *)logDesc logDayOffset:(int)dayOffset uploadResult:(ILiveLogUploadResultBl
```

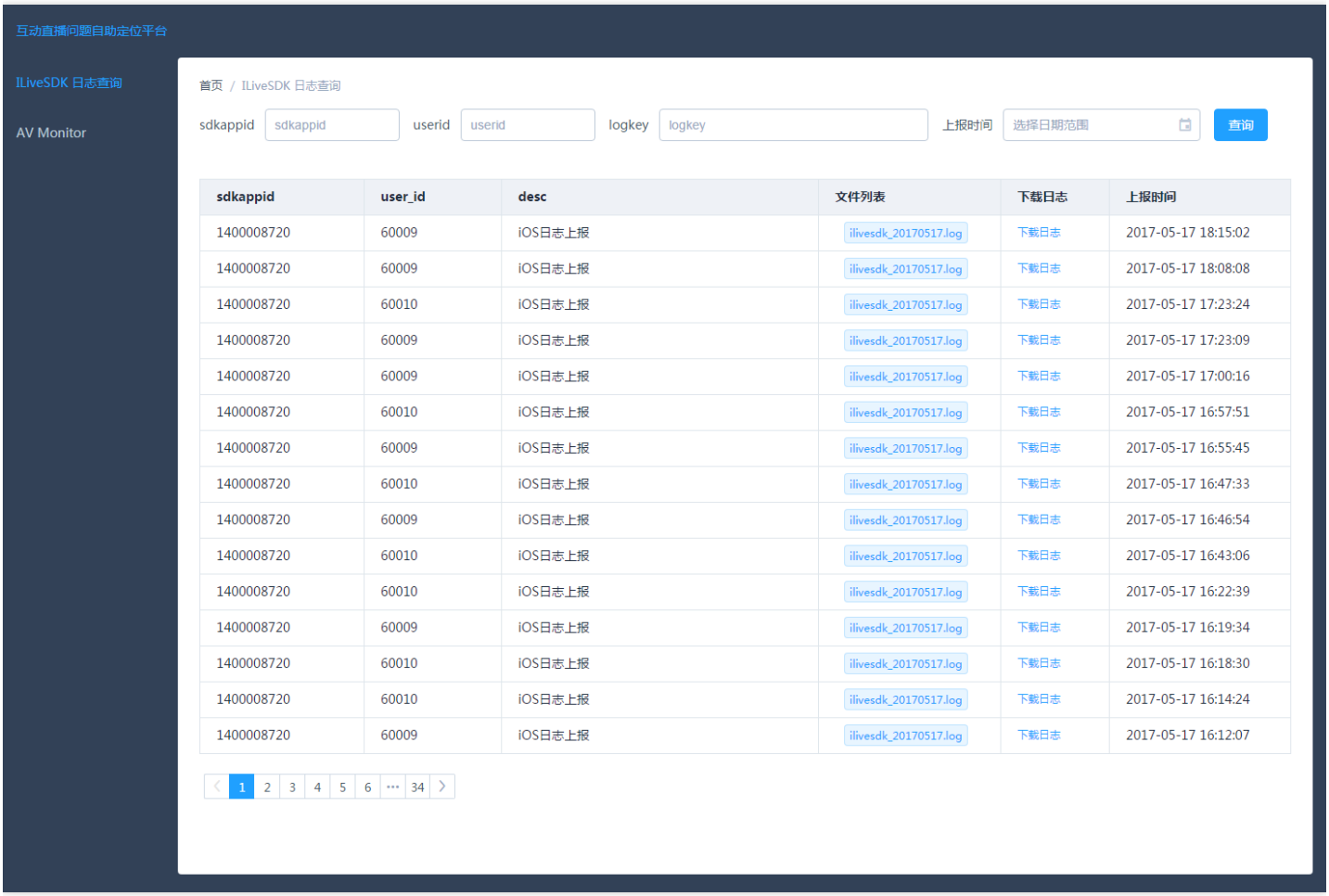
PC 接口：

```
//暂未暴露接口
```

Web 接口：

```
//暂未暴露接口
```

开发者可以在 日志查询平台 查询用户和下载用户上传的日志。当需要进一步分析问题的时候，可以提供日志 ID 给腾讯云技术支持人员。



常见问题

都会上报哪些日志？

iLiveSDK 流程日志，IMSDK 消息系统日志，AVSDK 日志后期会加上 Crash 信息日志。

如何指定上报哪天的日志？

0 表示当天 1-昨天 2-前天 类推。

上报过程中如果网络中断了会怎么样？

需要重新传送。

上报的日志是否有删除策略？

超过 15 天， 日志超过 1000 条自动清理。

错误码：

错误码	说明
0	日志上报成功

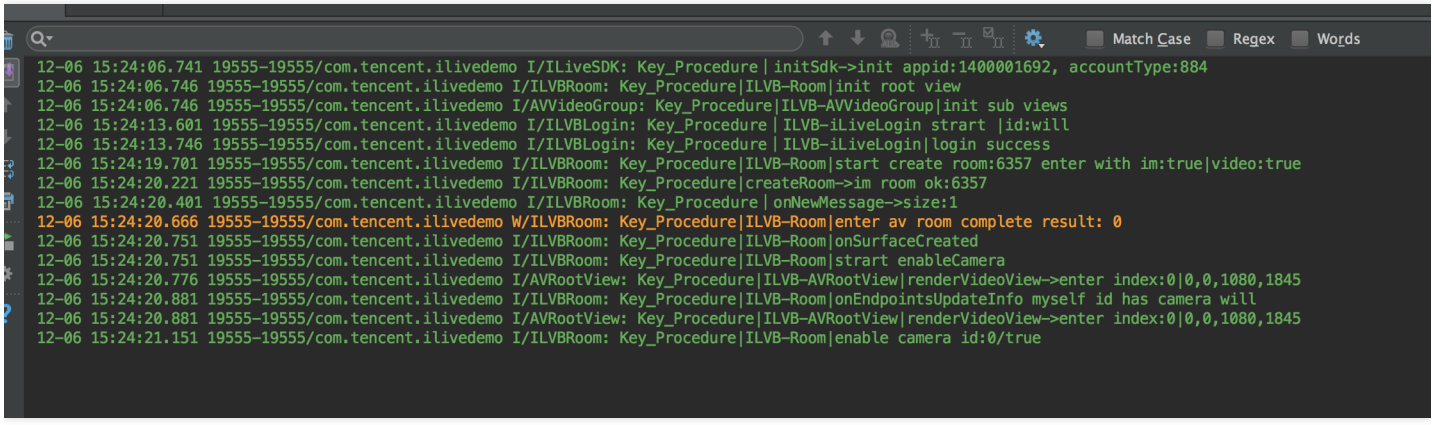
错误码	说明
8101	参数错误
8102	文件不存在
8103	压缩失败
8104	获取签名失败
8105	协议解析失败
8106	上传失败
8107	上报结果失败

iLive SDK 关键路径的 LOG

开发者在遇到问题时，可以根据这些日志，判断哪个流程执行出错，有助于定位问题。在 iLiveSDK 1.0.3 以后版本过滤关键字 Key_Procedure 会搜索到创建房间或加入房间的关键路径。

Android

创建房间流程正确 LOG 如下：



示例：

1. 初始化步骤
ILiveSDK: Key_Procedure | initSdk->init appid:1400001692, accountType:884

2. 设置渲染层
ILVBRoom: Key_Procedure|ILVB-Room|init root view

AVVideoGroup: Key_Procedure|ILVB-AVVideoGroup|init sub views

3. iLive 登录
ILVBLogin: Key_Procedure | ILVB-iLiveLogin strart |id:will
ILVBLogin: Key_Procedure | ILVB-iLiveLogin|login success

4. 创建房间
ILVBRoom: Key_Procedure|ILVB-Room|start create room:6357 enter with im:true|video:true

5. 直播聊天室创建完毕
ILVBRoom: Key_Procedure|createRoom->im room ok:6357

6. AV 房间创建完毕
ILVBRoom: Key_Procedure|ILVB-Room|enter av room complete result: 0

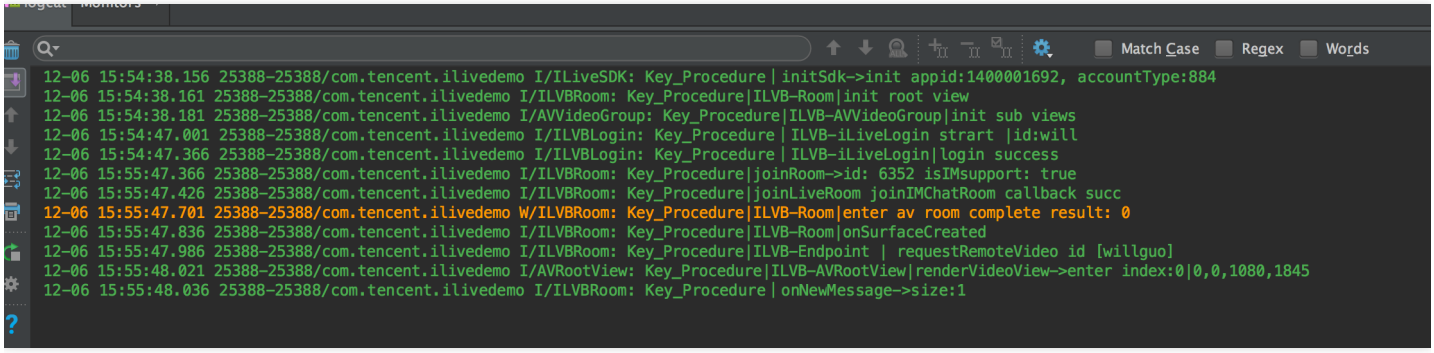
7. 打开摄像头
ILVBRoom: Key_Procedure|ILVB-Room|strart enableCamera

8. server 回调用户上线
ILVBRoom: Key_Procedure|ILVB-Room|onEndpointsUpdateInfo myself ID has camera will

9. 渲染
AVRootView: Key_Procedure|ILVB-AVRootView|renderVideoView->enter index:0|0,0,1080,1845

10. 摄像头上报成功回调
ILVBRoom: Key_Procedure|ILVB-Room|enable camera id:0/true

加入房间流程正确 LOG 如下：



示例：

1. 初始化

ILiveSDK: Key_Procedure | initSdk->init appid:1400001692, accountType:884

2. 设置渲染层
ILVBRoom: Key_Procedure|ILVB-Room|init root view
AVVideoGroup: Key_Procedure|ILVB-AVVideoGroup|init sub views

3. iLive 登录
ILVBLogin: Key_Procedure | ILVB-iLiveLogin strart |id:will
ILVBLogin: Key_Procedure | ILVB-iLiveLogin|login success

4. 加入房间
ILVBRoom: Key_Procedure|joinRoom->id: 6352 isIMsupport: true

5. 直播聊天室加入成功
ILVBRoom: Key_Procedure|joinLiveRoom joinIMChatRoom callback succ

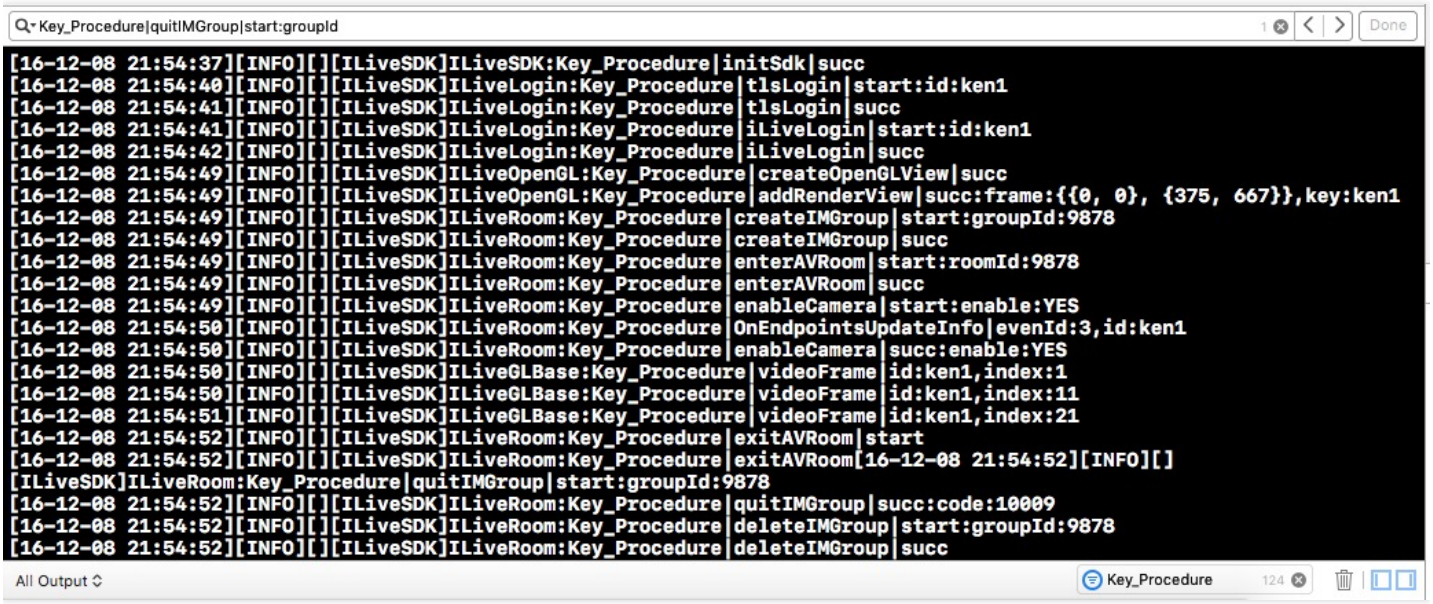
6. AV 房间加入成功
ILVBRoom: Key_Procedure|ILVB-Room|enter av room complete result: 0

7. 获取 server 成员上线回调
ILVBRoom: Key_Procedure|ILVB-Endpoint | requestRemoteVideo ID [willguo]

8. 渲染界面
AVRootView: Key_Procedure|ILVB-AVRootView|renderVideoView->enter index:0| 0,0,1080,1845

iOS

创建房间流程正确 LOG 如下：



示例：

1. 初始化 SDK
ILiveSDK:Key_Procedure|initSdk|succ

2. 登录 SDK（托管模式，如果是独立模式，无 tlsLogin 的 log）
ILiveLogin:Key_Procedure|tlsLogin|start:id:ken1
ILiveLogin:Key_Procedure|tlsLogin|succ
ILiveLogin:Key_Procedure|iLiveLogin|start:id:ken1
ILiveLogin:Key_Procedure|iLiveLogin|succ

3. 创建渲染根视图和主播渲染视图
ILiveOpenGL:Key_Procedure|createOpenGLView|succ
ILiveOpenGL:Key_Procedure|addRenderView|succ:frame:{{0, 0}, {375, 667}},key:ken1

4. 创建聊天群组
ILiveRoom:Key_Procedure|createIMGroup|start:groupId:9878
ILiveRoom:Key_Procedure|createIMGroup|succ

5. 创建直播房间
ILiveRoom:Key_Procedure|enterAVRoom|start:roomId:9878
ILiveRoom:Key_Procedure|enterAVRoom|succ

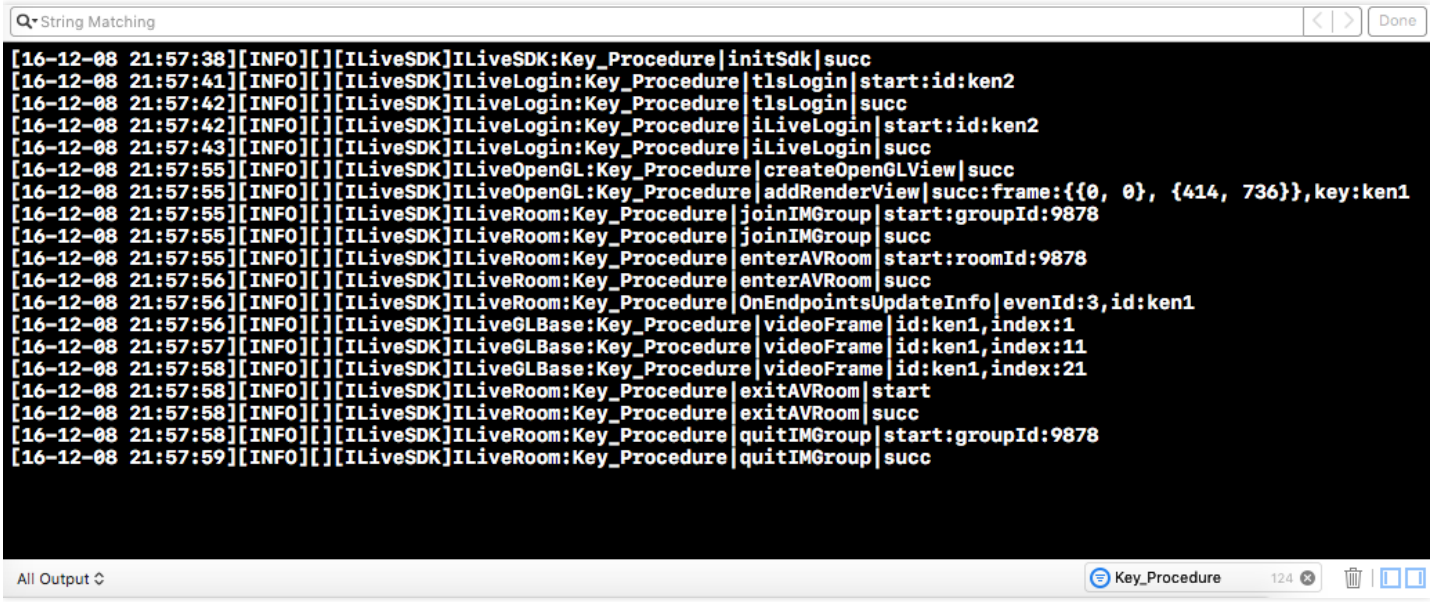
6. 打开摄像头并收到 server 事件（此处是摄像头开启事件）回调
ILiveRoom:Key_Procedure|enableCamera|start:enable:YES
ILiveRoom:Key_Procedure|OnEndpointsUpdateInfo|evenId:3,id:ken1
ILiveRoom:Key_Procedure|enableCamera|succ:enable:YES

7. 收到视频帧（间隔打印）
ILiveGLBase:Key_Procedure|videoFrame|id:ken1,index:1
ILiveGLBase:Key_Procedure|videoFrame|id:ken1,index:11
ILiveGLBase:Key_Procedure|videoFrame|id:ken1,index:21

8. 退出直播房间
ILiveRoom:Key_Procedure|exitAVRoom|start
ILiveRoom:Key_Procedure|exitAVRoom|succ

9 退出聊天群组（群主解散群组）
ILiveRoom:Key_Procedure|quitIMGroup|start:groupId:9878
ILiveRoom:Key_Procedure|quitIMGroup|succ:code:10009
ILiveRoom:Key_Procedure|deleteIMGroup|start:groupId:9878
ILiveRoom:Key_Procedure|deleteIMGroup|succ

加入房间流程正确 LOG 如下：



示例：

1. 初始化 SDK
ILiveSDK:Key_Procedure|initSdk|succ

2. 登录 SDK（托管模式，如果是独立模式，无 tlsLogin 的 log）
ILiveLogin:Key_Procedure|tlsLogin|start:id:ken2
ILiveLogin:Key_Procedure|tlsLogin|succ
ILiveLogin:Key_Procedure|iLiveLogin|start:id:ken2
ILiveLogin:Key_Procedure|iLiveLogin|succ

3. 创建渲染根视图和主播渲染视图
ILiveOpenGL:Key_Procedure|createOpenGLView|succ
ILiveOpenGL:Key_Procedure|addRenderView|succ:frame:{{0, 0}, {414, 736}},key:ken1

4. 加入聊天群组
ILiveRoom:Key_Procedure|joinIMGroup|start:groupId:9878
ILiveRoom:Key_Procedure|joinIMGroup|succ

5. 进入直播房间
ILiveRoom:Key_Procedure|enterAVRoom|start:roomId:9878
ILiveRoom:Key_Procedure|enterAVRoom|succ

6. server 事件（此处是摄像头开启事件）回调
ILiveRoom:Key_Procedure|OnEndpointsUpdateInfo|evenId:3,id:ken1

7. 收到主播视频帧（间隔打印）
ILiveGLBase:Key_Procedure|videoFrame|id:ken1,index:1

ILiveGLBase:Key_Procedure|videoFrame|id:ken1,index:11
ILiveGLBase:Key_Procedure|videoFrame|id:ken1,index:21

8. 退出直播房间
ILiveRoom:Key_Procedure|exitAVRoom|start
ILiveRoom:Key_Procedure|exitAVRoom|succ

9 退出聊天群组
ILiveRoom:Key_Procedure|quitIMGroup|start:groupId:9878
ILiveRoom:Key_Procedure|quitIMGroup|succ

PC

创建房间流程正确 LOG 如下：

```
2017-07-26 11:11:14|10772|Key_Procedure|iLiveRoomMgr|setMessageCallBack()
2017-07-26 11:11:14|10772|Key_Procedure|iLiveLoginMgr|setForceOfflineCallback()
2017-07-26 11:11:14|10772|Key_Procedure|iLiveRoomMgr|setLocalVideoCallBack()
2017-07-26 11:11:14|10772|Key_Procedure|iLiveRoomMgr|setRemoteVideoCallBack()
2017-07-26 11:11:14|10772|Key_Procedure|iLiveRoomMgr|setDeviceOperationCallback()
2017-07-26 11:11:14|10772|Key_Procedure|iLiveSDK|initSdk(). version: 1.5.0.0 appid: 1400027849 accoutType: 11656
2017-07-26 11:11:19|10772|Key_Procedure|iLiveLoginMgr|iLiveLogin() succeed. userId: yjp
2017-07-26 11:11:21|10244|Key_Procedure|iLiveRoomMgr|createRoom(). roomid: 51544
2017-07-26 11:11:45|10772|Key_Procedure|iLiveRoomMgr|MemStatusChange. identifier: yjp evnet_id: EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO
2017-07-26 11:11:45|10772|Key_Procedure|iLiveRoomMgr|openCamera(\\?\usb#vid_046d&pid_082d&mi_00#6&3b5ccbaa&0&0000#{65e8773d-8f56-11d0-a3b9-00a0c9223196}\global)
2017-07-26 11:11:46|10772|Key_Procedure|iLiveRoomMgr|requestViewList(yjp)
2017-07-26 11:11:50|10772|Key_Procedure|iLiveRoomMgr|MemStatusChange. identifier: yjp evnet_id: EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO
2017-07-26 11:11:50|10772|Key_Procedure|iLiveRoomMgr|closeCamera()
2017-07-26 11:11:52|11728|Key_Procedure|iLiveRoomMgr|quitRoom()
2017-07-26 11:11:53|11728|Key_Procedure|iLiveLoginMgr|iLiveLogout()
2017-07-26 11:11:54|10772|Key_Procedure|iLiveSDK|destroy()
```

示例：

1. 设置各个回调接口
Key_Procedure|iLiveRoomMgr|setMessageCallBack()　　//设置接收消息回调
Key_Procedure|iLiveLoginMgr|setForceOfflineCallback()　//设置被踢下线的回调
Key_Procedure|iLiveRoomMgr|setLocalVideoCallBack()　　//设置本地视频处理函数
Key_Procedure|iLiveRoomMgr|setRemoteVideoCallBack()　　//设置远程视频处理函数
Key_Procedure|iLiveRoomMgr|setDeviceOperationCallback() //设置设备操作回调

2. 初始化 SDK
Key_Procedure|iLiveSDK|initSdk(). version: 1.5.0.0 appid: 1400027849 accoutType: 11656

3. 登录
Key_Procedure|iLiveLoginMgr|iLiveLogin() succeed. userId: yjp

4. 创建直播间
Key_Procedure|iLiveRoomMgr|createRoom(). roomid: 51544

5. 打开摄像头
Key_Procedure|iLiveRoomMgr|openCamera(\\摄像头 id)

6. 退出直播间
Key_Procedure|iLiveRoomMgr|quitRoom()

7. 登出
Key_Procedure|iLiveLoginMgr|iLiveLogout()

8. 清理
Key_Procedure|iLiveSDK|destroy()

加入房间流程正确 LOG 如下：

```
2017-07-26 11:22:07|10404|Key_Procedure|iLiveRoomMgr|setMessageCallBack()
2017-07-26 11:22:07|10404|Key_Procedure|iLiveLoginMgr|setForceOfflineCallback()
2017-07-26 11:22:07|10404|Key_Procedure|iLiveRoomMgr|setLocalVideoCallBack()
2017-07-26 11:22:07|10404|Key_Procedure|iLiveRoomMgr|setRemoteVideoCallBack()
2017-07-26 11:22:07|10404|Key_Procedure|iLiveRoomMgr|setDeviceOperationCallback()
2017-07-26 11:22:07|10404|Key_Procedure|iLiveSDK|initSdk(). version: 1.5.0.0 appid: 1400027849 accoutType: 11656
2017-07-26 11:22:12|10404|Key_Procedure|iLiveLoginMgr|iLiveLogin() succeed. userId: yjp
2017-07-26 11:22:15|10404|Key_Procedure|iLiveRoomMgr|MemStatusChange. identifier: yjp1 evnet_id: EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO
2017-07-26 11:22:15|10404|Key_Procedure|iLiveRoomMgr|requestViewList(yjp1)
2017-07-26 11:22:15|11868|Key_Procedure|iLiveRoomMgr|joinRoom(). roomid: 51552
2017-07-26 11:22:19|11868|Key_Procedure|iLiveRoomMgr|quitRoom()
2017-07-26 11:22:20|11868|Key_Procedure|iLiveLoginMgr|iLiveLogout()
2017-07-26 11:22:21|10404|Key_Procedure|iLiveSDK|destroy()
```

示例：

1. 设置各个回调接口
Key_Procedure|iLiveRoomMgr|setMessageCallBack()　　//设置接收消息回调
Key_Procedure|iLiveLoginMgr|setForceOfflineCallback()　//设置被踢下线的回调
Key_Procedure|iLiveRoomMgr|setLocalVideoCallBack()　　//设置本地视频处理函数
Key_Procedure|iLiveRoomMgr|setRemoteVideoCallBack()　　//设置远程视频处理函数
Key_Procedure|iLiveRoomMgr|setDeviceOperationCallback() //设置设备操作回调

2. 初始化 SDK
Key_Procedure|iLiveSDK|initSdk(). version: 1.5.0.0 appid: 1400027849 accoutType: 11656

3. 登录
Key_Procedure|iLiveLoginMgr|iLiveLogin() succeed. userId: yjp

4. 加入直播间
Key_Procedure|iLiveRoomMgr|joinRoom(). roomid: 51552

5. 请求画面
Key_Procedure|iLiveRoomMgr|requestViewList(yjp1)

6. 退出直播间
Key_Procedure|iLiveRoomMgr|quitRoom()

7. 登出

Key_Procedure|iLiveLoginMgr|iLiveLogout()

8. 清理
Key_Procedure|iLiveSDK|destroy()

Web

与 PC 端一致。

错误码表

最近更新时间：2018-06-15 18:36:16

IMSDK 错误码

因互动直播 SDK 包含了 IMSDK，有些错误码为 IMSDK 返回，详情参阅 错误码。

AVSDK 客户端错误

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_FAILED	1	一般错误	具体原因需要通过分析日志等来定位。	分析日志。
AV_ERR_REPEATED_OPERATION	1001	重复操作	"已经在进行某种操作，再次去做同样的操作，则会产生这个错误。如已经在进入房间过程中，再去做进入房间的操作，就会产生这个错误。操作类别主要有：AVContext 类、房间类、设备类、成员类。 AVContext 类型的操作 ：StartContext/StopContext。 房间类型的操作 ：EnterRoom/ExitRoom。 设备类型的操作 ：打开/关闭某个设备。 成员类型的操作 ：请求画面/取消画面。"	等待上一个操作完成后再进行下一个操作。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_EXCLUSIVE_OPERATION	1002	互斥操作	已经在进行某种操作，再次去做同类型的其他操作，则会产生这个错误。如在进入房间过程中，去做退出房间的操作，就会产生这个错误。	等待上一个操作完成后再进行下一个操作。
AV_ERR_HAS_IN_THE_STATE	1003	已处于所要状态	对象已经处于某种状态，再去做使得它进入这种状态的操作时，则会产生这个错误。如已经在房间中，再去做进入房间的操作，就会产生这个错误。	由于已经处于所要状态，可以认为该操作已经成功，当作成功来处理。
AV_ERR_INVALID_ARGUMENT	1004	错误参数	调用 SDK 接口时，传入错误的参数，则会产生这个错误。如进入房间时，传入的房间类型不等于 AVRoom::ROOM_TYPE_PAIR 或 AVRoom::ROOM_TYPE_MULTI，就会产生这个错误。	详细阅读 API 文档，搞清楚每个接口的每个参数的有效取值范围，确认哪些参数没有按照规范来取值，保证传入参数的正确性并进行相应的预防处理。
AV_ERR_TIMEOUT	1005	超时	进行某个操作，在规定的时间内，还未返回操作结果，则会产生这个错误。多数情况下，涉及到信令传输的、且网络出问题的情况下，才容易产生这个错误。如执行进入房间操作时，30s 后还没有返回进入房间操作完成的结果的话，就会产生这个错误。	确认网络是否有问题，并尝试重试。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_NOT_IMPLEMENTED	1006	未实现	调用 SDK 接口时，如果相应的功能还未支持，则会产生这个错误。	暂不支持该功能，找其他替代方案。
AV_ERR_NOT_IN_MAIN_THREAD	1007	不在主线程	SDK 对外接口要求在主线程执行，如果业务侧调用 SDK 接口时，没有在主线程调用，则会产生这个错误。	修改业务侧逻辑，确保在主线程调用 SDK 接口。
AV_ERR_RESOURCE_IS_OCCUPIED	1008	资源被占用	当需要用到某种资源，如摄像头、屏幕等，而这些资源已经被占用了，则会产生这个错误。	确认具体要用到哪些资源，及这样资源被谁占用了，确保在正确的时机使用 SDK 相关功能以保证资源使用不冲突。
AV_ERR_CONTEXT_NOT_EXIST	1101	AVContext 对象未处于 CONTEXT_STATE_STARTED 状态	当 AVContext 对象未处于 CONTEXT_STATE_STARTED 状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。
AV_ERR_CONTEXT_NOT_STOPPED	1102	AVContext 对象未处于 CONTEXT_STATE_STOPPED 状态	当 AVContext 对象未处于 CONTEXT_STATE_STOPPED 状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。如不在这种状态下，去调用 AVContext::DestroyContext 时，就会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_ROOM_NOT_EXIST	1201	AVRoom 对象未处于 ROOM_STATE_ENTERED 状态	当 AVRoom 对象未处于 ROOM_STATE_ENTERED 状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。
AV_ERR_ROOM_NOT_EXITED	1202	AVRoom 对象未处于 ROOM_STATE_EXITED 状态	当 AVRoom 对象未处于 ROOM_STATE_EXITED 状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。如不在这种状态下，去调用 AVContext::StopContext 时，就会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。
AV_ERR_DEVICE_NOT_EXIST	1301	设备不存在	当设备不存在或者设备初始化未完成时，去使用设备，则会产生这个错误。	确认设备是否真的存在，确保设备 ID 填写的正确性，确保在设备初始化成功后再去使用设备。
AV_ERR_ENDPOINT_NOT_EXIST	1401	AVEndpoint 对象不存在	当成员没有在发语音或视频时，去获取它的 AVEndpoint 对象时，就可能产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。
AV_ERR_ENDPOINT_HAS_NOT_VIDEO	1402	成员没有发视频	当成员没有在发视频时，去做需要成员发视频的相关操作时，就可能产生这个错误。如当某个成员没有发送视频时，去请求他的画面，就会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_TINYID_TO_OPENID_FAILED	1501	tinyid 转 identifier 失败	当收到某个成员信息更新的信令时，需要去把 tinyid 转成 identifier，如果 IMSDK 库相关逻辑存在问题、网络存在问题等，则会产生这个错误。	确认网络是否存在问题，查看日志确认 IMSDK 相关逻辑是否存在问题。
AV_ERR_OPENID_TO_TINYID_FAILED	1502	identifier 转 tinyid 失败	当调用 StartContext 接口时，需要去把 identifier 转成 tinyid，如果 IMSDK 库相关逻辑存在问题、网络存在问题、没有处于登录态时等，则会产生这个错误。	确认网络是否存在问题，查看日志确认 IMSDK 相关逻辑是否存在问题，确认已经成功调用 IMSDK 的登录接口。
AV_ERR_DEVICE_TEST_NOT_EXIST	1601	AVDeviceTest 对象未处于 DEVICE_TEST_STATE_STARTED 状态 (Windows 特有)	当 AVDeviceTest 对象未处于 DEVICE_TEST_STATE_STARTED 状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。
AV_ERR_DEVICE_TEST_NOT_STOPPED	1602	AVDeviceTest 对象未处于 DEVICE_TEST_STATE_STOPPED 状态（Windows 特有）	当 AVDeviceTest 对象未处于 DEVICE_TEST_STATE_STOPPED 状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。如不在这种状态下，去调用 AVContext::StopContext 时，就会产生这个错误。	修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_INVITE_FAILED	1801	发送邀请失败	发送邀请时产生的失败。	邀请模块只是用于 DEMO 演示方便用，对外暂不支持邀请功能，所以业务侧不需要处理这些错误码。
AV_ERR_ACCEPT_FAILED	1802	接受邀请失败	接受邀请时产生的失败。	邀请模块只是用于 DEMO 演示方便用，对外暂不支持邀请功能，所以业务侧不需要处理这些错误码。
AV_ERR_REFUSE_FAILED	1803	拒绝邀请失败	拒绝邀请时产生的失败。	邀请模块只是用于 DEMO 演示方便用，对外暂不支持邀请功能，所以业务侧不需要处理这些错误码。
QAV_ERR_NOT_TRY_NEW_ROOM	2001	没有尝试进入新房间，将停留在旧房间。	切换到新房间失败，但仍然在当前房间	在当前房间仍可以正常使用。
QAV_ERR_TRY_NEW_ROOM_FAILED	2002	尝试进入新房间，但失败了，旧房间也将关闭。	切换到新房间失败，同时已退出原来的房间	退出房间，重新进入。

AVSDK 服务端错误

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_SERVER_FAILED	10001	一般错误	具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_SERVER_INVALID_ARGUMENT	10002	错误参数	调用 SDK 接口时，或 SDK 内部发送信令给后台时，传了错误的参数，具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	确保调用 SDK 接口时所传的参数的正确性。分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_NO_PERMISSION	10003	没有权限	没有权限使用某个功能，具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。如进入房间时所带的签名错误或过期，就会产生这个错误。	确保在设置正确的权限参数后才去使用相应的功能。分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_TIMEOUT	10004	超时	具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_ALLOC_RESOURCE_FAILED	10005	资源不够	执行某些操作时，需要分配更多的资源(如内存)失败了，或者超过最大的资源限制(如超过最大的房间成员人数)，则会产生这个错误。具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_ID_NOT_IN_ROOM	10006	不在房间内	在不在房间内时，去执行某些操作，则会产生这个错误。具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_NOT_IMPLEMENT	10007	未实现	调用 SDK 接口时，如果相应的功能还未支持，则会产生这个错误。	暂不支持该功能，找其他替代方案。

错误码名称	错误码值	含义	原因	建议方案
AV_ERR_SERVER_REPEATED_OPERATION	10008	重复操作	具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_ROOM_NOT_EXIST	10009	房间不存在	房间不存在时，去执行某些操作，则会产生这个错误。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_ENDPOINT_NOT_EXIST	10010	成员不存在	某个成员不存在时，去执行该成员相关的操作，则会产生这个错误。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
AV_ERR_SERVER_INVALID_ABILITY	10011	错误能力	具体原因需要通过分析日志确认后台返回给客户端的真正错误码才能知道。	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。

录制错误码

错误码名称	错误说明	处理建议
30000000	SDK 请求解析失败	【录制请求字段填写是否完整】
30000001	SDK 请求解析失败-没有录制请求包体	【录制请求字段填写是否完整】
30000002	SDK 请求解析失败-没有录制文件名字段	【录制请求字段填写是否完整】
30000003	SDK 请求解析失败-没有录制请求操作字段	【录制请求字段填写是否完整】
30000004	SDK 请求解析失败-视频源类型错误（摄像头/桌面等）	【录制请求字段填写是否完整】
30000201	请求服务器内部数据打包错误	【反馈腾讯客服】
30000202	请求服务器内部数据打包错误	【反馈腾讯客服】
30000203	请求服务器内部数据打包错误	【反馈腾讯客服】

错误码名称	错误说明	处理建议
30000207	请求录制服务器通讯错误-拉取录制服务器地址失败	【反馈腾讯客服】
30000208	请求录制服务器通讯错误-请求录制服务器超时	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
30000301	解析录制服务器回包错误-数据包解析失败	【反馈腾讯客服】
30000302	解析录制服务器回包错误-数据包解析失败	【反馈腾讯客服】
30000303	解析录制服务器回包错误-没有返回 IP	【反馈腾讯客服】
30000304	解析录制服务器回包错误-没有返回端口	【反馈腾讯客服】
30000305	解析录制服务器回包错误-没有返回结果	【反馈腾讯客服】
30000401	查询房间获取 grocery 服务 IP 错误	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
30000402	查询房间拉取 grocery 数据错误	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
30000403	查询房间拉取 grocery 不存在（房间不存在）	【检查是否成功开房，录制的用户 ID，groupid 是否填写正确】
30000404	查询房间流控服务器超时	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
30000405	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
30000406	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
30000407	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
30000408	查询房间回包错误-没有返回结果	【反馈腾讯客服】
30000409	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
30000410	录制的房间不存在	【检查是否成功开房，录制的用户 ID，groupid 是否填写正确,或者用户是否已经退出房间】

错误码名称	错误说明	处理建议
30000411	录制的房间不存在，或发起录制的用户不存在	【检查是否成功开房，录制的用户 ID，groupid 是否填写正确,或者用户是否已经退出房间】
30000412	停止录制重复发送，用户已经停止录制	【如果是录制停止操作说明已经停止，检查是否多次发送停止操作，无需处理】
30000413	停止录制重复发送，用户已经停止录制	【如果是录制停止操作说明已经停止，检查是否多次发送停止操作，无需处理】
30000414	查询房间-服务器内部操作类型错误	【反馈腾讯客服】
30000415	启动录制重复发送，用户正在录制；或者发起录制的用户不存在	【检查是否成功开房，录制的用户 ID，groupid 是否填写正确】

旁路直播错误码

错误码名称	错误说明	处理建议
40000000	SDK 请求解析失败	【推流请求字段填写是否完整】
40000001	SDK 请求解析失败-没有推流请求包体	【推流请求字段填写是否完整】
40000002	SDK 请求解析失败-没有推流请求操作字段	【推流请求字段填写是否完整】
40000003	SDK 请求解析失败-缺少推流请求的输出编码（HLS/RTMP 等）	【推流请求字段填写是否完整】
40000004	SDK 请求解析失败-视频源类型错误（摄像头/桌面等）	【推流请求字段填写是否完整】
40000005	SDK 请求解析失败-请求操作错误（请求推流、停止推流）	【推流请求字段填写是否完整】
40000006	请求推流的时候检查用户 ID 不正确	【推流请求字段填写是否正确】
40000007	推流房间 ID 填写成 0	【请检查推流的房间 ID 的填写】
40000201	请求服务器内部数据打包错误	【反馈腾讯客服】

错误码名称	错误说明	处理建议
40000202	请求服务器内部数据打包错误	【反馈腾讯客服】
40000203	请求服务器内部数据打包错误	【反馈腾讯客服】
40000207	请求推流服务器通讯错误-拉取推流服务器地址失败	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
40000208	请求推流服务器通讯错误-请求推流服务器超时	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
40000301	解析推流服务器回包错误-数据包解析失败	【反馈腾讯客服】
40000302	解析推流服务器回包错误-数据包解析失败	【反馈腾讯客服】
40000303	解析推流服务器回包错误-没有返回 IP	【反馈腾讯客服】
40000304	解析推流服务器回包错误-没有返回端口	【反馈腾讯客服】
40000305	解析推流服务器回包错误-没有返回结果	【反馈腾讯客服】
40000306	解析推流服务器回包错误-返回 URL 长度溢出	【反馈腾讯客服】
40000401	查询房间获取 grocery 服务 IP 错误	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
40000402	查询房间拉取 grocery 数据错误	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
40000403	查询房间拉取 grocery 不存在（请求推流的房间不存在）	【检查是否成功开房，推流的用户 ID，groupid 是否填写正确】
40000404	查询房间流控服务器超时	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
40000405	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
40000406	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
40000407	查询房间回包错误-数据包解析失败	【反馈腾讯客服】
40000408	查询房间回包错误-没有返回结果	【反馈腾讯客服】
40000409	查询房间回包错误-数据包解析失败	【反馈腾讯客服】

错误码名称	错误说明	处理建议
40000410	请求推流的房间不存在	【检查是否成功开房，推流的用户 ID，groupid 是否填写正确,或者用户是否已经退出房间】
40000411	发起推流用户不在房间内	【检查是否成功开房，推流的用户 ID，groupid 是否填写正确,或者用户是否已经退出房间】
40000412	停止推流重复发送，用户已经停止推流	【如果是推流停止操作说明已经停止，重复停止操作，无需处理】
40000413	停止推流重复发送，用户已经停止推流	【如果是推流停止操作说明已经停止，无需处理】
40000414	查询房间-服务器内部操作类型错误	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
40000415	启动推流重复发送，用户正在推流	【如果是推流启动操作说明已经是在推流状态，无需处理】
40000500	启动推流频率控制	【非异常，同一个用户在 3 秒内重复请求推流会返回次错误，重试请求需要在上一次请求发起 3 秒后】
1001	权限错误	【一般是 sdkappid 填写错误导致】
20101	通道数超过上限	【推流通道数存在上限，在推流控制台检查并删除无用的通道】
20406	用户欠费	【检查是否已欠费】
50002	输入参数检验错误	检查用户 ID 是否填写错误，sdkappid 是否填写错误
50003	后端没有拉取到拉流的 url	【反馈腾讯客服】
50004	推流请求的推流类型错误	检查推流类型字段填写是否正确
50005	连接后端控制台超时	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
50006	连接后端控制台超时	【可能是网络问题，重试处理，重试失败反馈腾讯客服】
50007	后端返回参数为空	【反馈腾讯客服】

功能定制

最近更新时间：2018-06-15 18:36:23

功能定制

如果您的业务场景非常特殊，翻遍完我们的文档和 Demo 代码，都不能满足需求。您可以自己拿到 IM SDK 和 AVSDK 的接口，自由操作信令、群组、音视频接口进行定制。

Android

> **注意：**
> 请在 ILiveSDK 初始化并登录后再使用以上接口。

获取 IM 管理器实例：详情请参阅 Android IMSDK 接口。

```
TIMManager imManager = ILiveSDK.getTIMManager();
```

获取 AV 上下文实例：详情请参阅 Android AVSDK 接口。

```
AVContext avContext = ILiveSDK.getAVContext();
```

iOS

> **注意：**
> 请在 ILiveSDK 初始化并登录后再使用以上接口。

获取 IM 管理器实例：详情请参阅 iOS IMSDK 接口。

```
TIMManager *imManager = [[ILiveSDK getInstance] getTIMManager];
```

获取 AV 上下文实例：详情请参阅 iOS AVSDK 接口。

```
QAVContext *avContext = [[ILiveSDK getInstance] getAVContext];
```

联系我们

如果您希望腾讯云提供开发服务并尽快上线产品，请提 工单。

音视频密钥使用说明

最近更新时间：2018-06-15 18:36:28

密钥及加密算法

密钥： appId 对应音视频密钥，长度 16 字节。互动直播控制台 上可查。
加密算法：Tea 加密。
加密库及例子：附件《tea.zip》。

使用场景一：开房间权限加密串

密文内容

字段描述	类型/长度	值定义/备注
cVer	unsigned char/1	版本号，填 0
wAccountLen	unsigned short /2	第三方自己的帐号长度

字段描述	类型/长度	值定义/备注
buffAccount	wAccountLen	第三方自己的帐号字符
dwSdkAppid	unsigned int/4	sdkappid
dwAuthId	unsigned int/4	群组号码
dwExpTime	unsigned int/4	过期时间（当前时间 + 有效期（单位：秒，建议 300 秒））
dwPrivilegeMap	unsigned int/4	权限位
dwAccountType	unsigned int/4	第三方帐号类型

加密方式

- 密文中的数字转换成网络字节序（大端字节序）。
- 把密文拼接成一段字符串。
- 用 tea 加密对字符串加密，symmetry_encrypt 函数输出的字符串即为权限加密串(注意：不用要把二进制串转成 16 进制)。

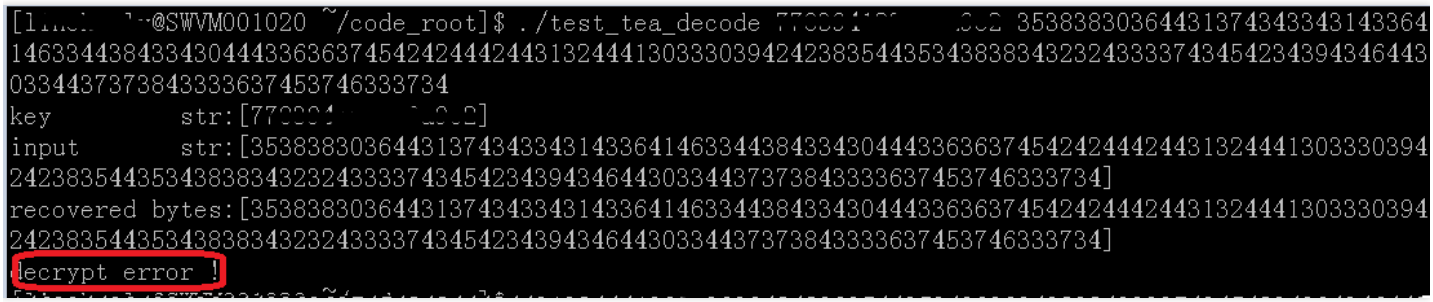
密文验证

用工具 test_tea_decode（和 lib 一起打包提供）可以初步验证密文是否可以解密。将密文转成 16 进制的可读字符串并使用工具校验。

能正确解密的场景：

```
[i...  ...@SWVM001U20 ~/code_root]$ ./test_tea_decode 77...  0909F0336E8C7AE71572C7D33997
76A03965451A2B8D2F2FBC13316947D24065C63C3277C2F11CC7395EB4E604AB09C3
key          str:[...]
input        str:[0909F0336E8C7AE71572C7D3399776A03965451A2B8D2F2FBC13316947D24065C63C3277C2F11CC7395
EB4E604AB09C3]
recovered bytes:[0909f0336e8c7ae71572c7d3399776a03965451a2b8d2f2fbc13316947d24065c63c3277c2f11cc7395
eb4e604ab09c3]
decrypt = [0](len:32)
3000093134343333303434375372a8a10023.,
```

不能正确解密的场景：



使用场景二：跨房间连麦加密串

密文内容

附件：《conn_room_sig.proto》

```
package tencent.im.groupvideo.conn_room;
message ConnRoomSig
{
    optional uint32      uint32_groupcode          = 1; //发起方群号
    optional string      str_third_account         = 2; //发起方第三方帐号

    optional uint32      uint32_conned_groupcode   = 3; //被连方群号
    optional string      str_conned_third_account  = 4; //被连方第三方帐号

    optional uint32      uint32_create_time        = 5; //签名创建时间
    optional uint32      uint32_expire_time        = 6; //签名过期时间
}
```

字段描述	是否必须	值定义/备注
uint32_groupcode	是	发起方群号/群组 Id
str_third_account	是	发起方帐号
uint32_conned_groupcode	是	被连麦群号/群组 Id
str_conned_third_account	是	被连方帐号
uint32_create_time	是	签名创建时间
uint32_expire_time	是	签名过期时间，建议设置为创建时间+300秒

加密方式

- 使用 google protobuf 序列化 ConnRoomSig 对象，输出二进制字符串。
- 用 tea 加密对二进制字符串加密，symmetry_encrypt 函数输出的字符串即为加密串。
- 把加密串转换成 16 进制字符串，大小写均可。

```
bytes_conn_room_sig:
"00A47C7E5FD0471A6D90CBE6FAAC2D862A114EFC6337D8F0BD1161BB53250F4EE46DB0244E8515D58B
```

详情请参阅 google protobuf 官网，单击下载 加密代码。