音视频终端 SDK(腾讯云视立方)

实时互动

产品文档

腾讯云

文档目录

实时互动
直播连麦
连麦互动 RTC App

最近更新时间：2022-03-04 10:57:59

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	✓	✓	–	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

基于 RTC 协议的连麦方案

目前，在 **连麦互动 – RTMP方案** 中，腾讯云视立方 · 移动直播 SDK 提供连麦互动组件 MLVBLiveRoom 用来帮助开发者快速实现连麦需求，为了更好的满足开发者对连麦功能的需求，腾讯云新增了**基于实时音视频 TRTC 能力实现的新版连麦方案**，同时提供了更加简单灵活的全新接口：V2TXLivePusher （推流）、V2TXLivePlayer （拉流）。

移动直播全新接口同时支持通过 RTMP 协议及 RTC 协议进行推流/连麦，开发者可根据自身需求选择适合的方案，对比如下：

对比项	旧版连麦方案	新版连麦方案
协议	RTMP 基于 TCP 协议	RTC 基于 UDP 协议（更适合流媒体传输）
QoS	弱网抗性能力弱	50%丢包率可正常视频观看，70%丢包率可正常语音连麦
支持区域	仅支持中国内地（大陆）地区	全球覆盖
使用产品	需开通移动直播、云直播服务	需开通移动直播、云直播、实时音视频服务
价格	0.016元/分钟	阶梯价格，详情请参见 RTC 连麦方案怎么计算费用

RTC 连麦方案如何接入

RTC 连麦方案用来帮助客户实现更加灵活、更低延时、更多人数的直播互动场景。开播端可以利用全新接口提供的 RTC 推流能力，默认情况下，观众端观看则可使用 CDN 方式进行拉流。 CDN 观看费用较低。如果观众端有连麦需求，连麦观众上麦后，可以从 CDN 切换到 RTC 进行观看，这样延时更低，互动效果更好。以下是相关示例代码和具体的步骤说明，方便您快速接入：

示例工程

平台	源码地址	目标文件夹
Android	Github	LiveLink
iOS	Github	LiveLink

演示图示

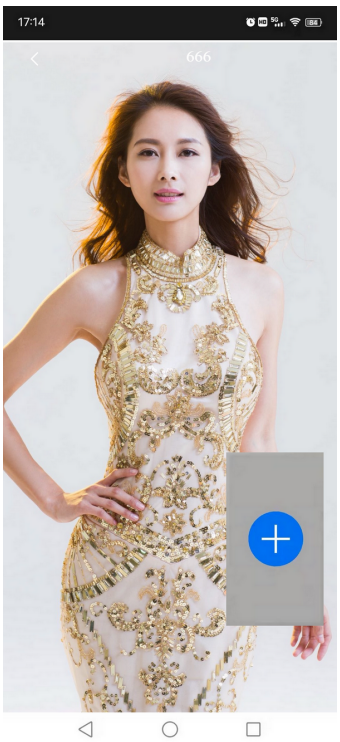
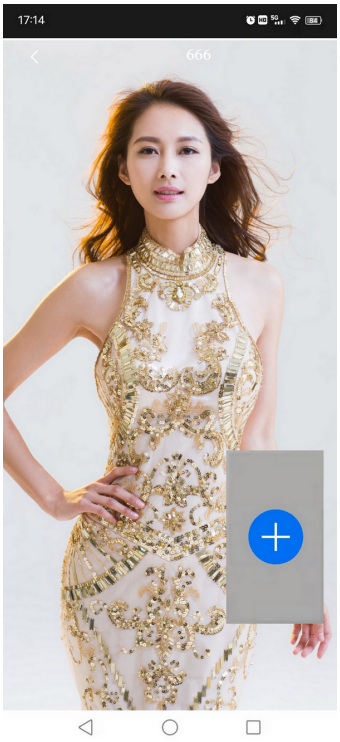
直播前

主播			

			
连麦观众和观众			

连麦操作

主播		

		
连麦观众		

连麦中

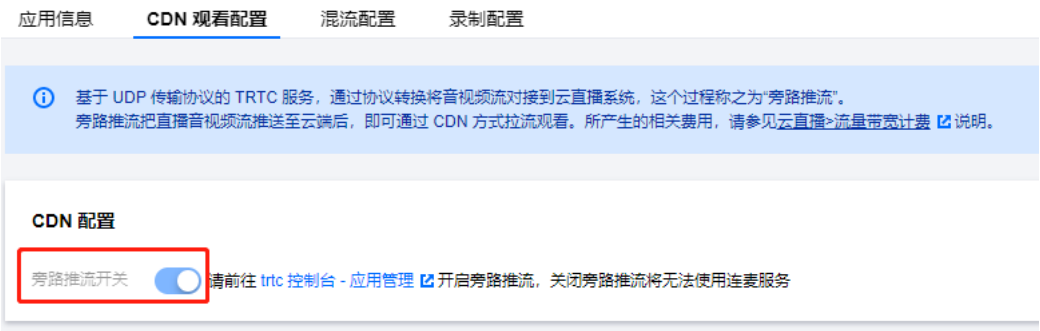
主播端（手机 A）	连麦观众（手机 B）	普通观众（手机 C）



步骤1：服务开通

1. 登录**云直播控制台**，选择 **连麦管理** > **连麦应用**。
2. 单击 **新建连麦应用** ，输入应用名称，例如 V2Demo ，单击 **确定** 。
3. 创建成功后，单击应用列表中 **应用名称** 为 V2Demo 这行右侧的 **管理** ，查看应用对应的 SDKAppID 和 秘钥 信息。

4. 若您的播放端需要进行 CDN 播放，则需要在 **连麦管理** > **连麦应用** 中选择 V2Demo 行右侧的 **管理**，选择 **CDN 观看配置** 页，开启 **旁路推流** 功能。



> **说明：**
>
> - 旁路推流的方式默认选择 指定流旁路 即可，对于 V2TXLivePusher 两种方式没有区别。
> - 在服务开通后，建议先可以编译和体验一下上述示例代码章节中腾讯云提供的移动直播的 API-Example 工程，配合上下文可以快速的了解相关 API 的使用。

步骤2：V2TXLivePusher RTC 推流

URL 拼接

具体的推流 URL字符串，需要开发者按照下方协议解析中的规则，在工程代码中自行拼接。URL 的示例如下：

推流 URL

```
trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx
```

在上述的 URL 中，存在一些关键字段，关于其中关键字段的含义信息，详见下表：

字段名称	字段含义
trtc://	互动直播推流 URL 的前缀字段
cloud.tencent.com	互动直播特定域名，**请勿修改**
push	标识位，表示推流
streamid	流 ID，需要由开发者自定义
sdkappid	对应 服务开通 一节中生成的 SDKAppID
userId	主播 ID，需要由开发者自定义
usersig	对应 服务开通 中获取的 UserSig 密钥

示例代码

```
// 创建□个 V2TXLivePusher 对象，并指定模式为 TXLiveMode_RTC;
V2TXLivePusher pusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_RTC);
pusher.setObserver(new MyPusherObserver());
pusher.setRenderView(mSurfaceView);
pusher.startCamera(true);
pusher.startMicrophone();
// 传□互动直播RTC推流协议地址，即可开始推流，其中streamid设置为自定义值，比如12345687；；
pusher.startPush("trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=finnguan&usersig=xxxxx");
```

步骤3：V2TXLivePlayer CDN 播放

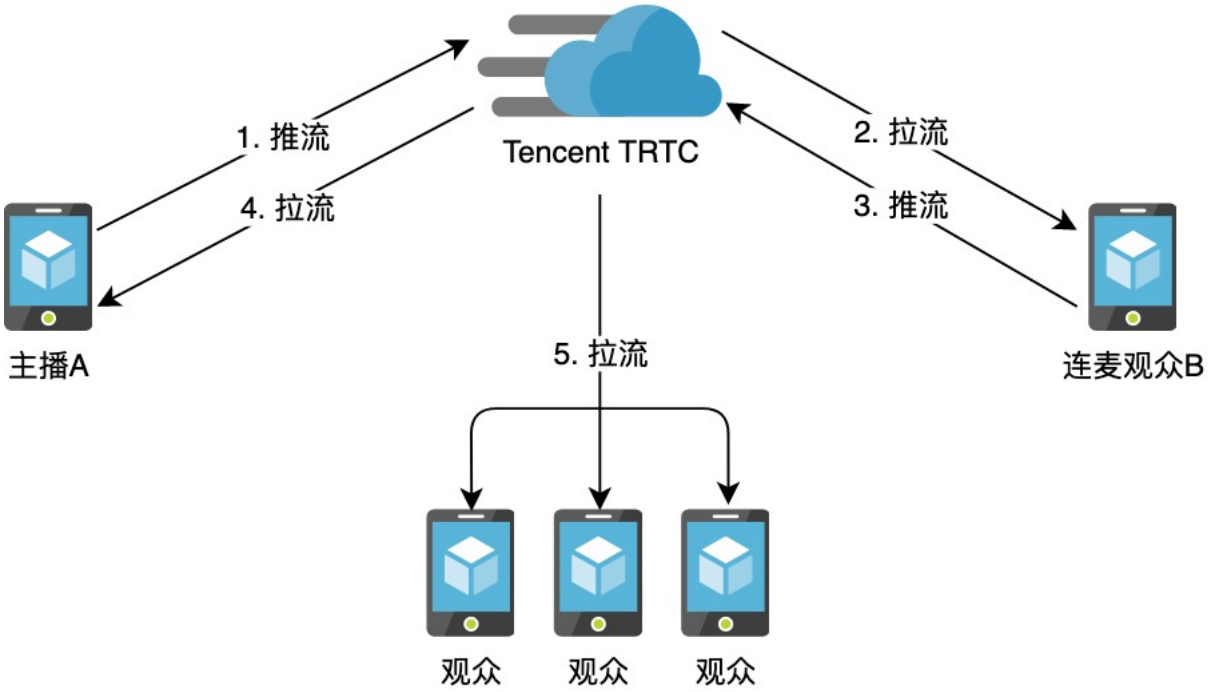
URL 拼接

具体的播放 URL 字符串，需要开发者按照 域名 + streamID ，在工程代码中自行拼接。

示例代码

```
// 创建□个 V2TXLivePlayer 对象;
V2TXLivePlayer player = new V2TXLivePlayerImpl(mContext);
player.setObserver(new MyPlayerObserver(playerView));
player.setRenderView(mSurfaceView);
// 传□互动直播播放协议地址，即可开始播放，streamid对应推流时设置的streamid，比如12345687;
player.startPlay("https://3891.liveplay.myqcloud.com/live/streamid.flv");
```

步骤4：实现观众连麦



- **主播 RTC 推流：**
  主播 A 开始推流，调用 V2TXLivePusher 组件开始主播 A 的推流。

```
V2TXLivePusher pusherA = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
/**
* pushURLA= "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxx";
*/
pusherA.startPush(pushURLA);
```

- **观众 CDN 拉流：**
  所有观众观看主播 A 直播，调用 V2TXLivePlayer 开始播放主播 A 的流。

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
/**
* 这里使用CDN拉流，支持FLV、HLS、WebRTC协议，任选一种协议。FLV、HLS等标准协议价格更合理，WebRTC快直播能够提供更低延迟的互动体验。
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidA.flv";
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidA.hls";
* playURLA= "webrtc://3891.liveplay.myqcloud.com/live/streamidA"
*/
playerA.startPlay(playURLA);
```

- **观众发起连麦：**
  其中观众 B 调用 V2TXLivePusher 发起推流（后续会称呼为连麦观众 B）。

```
V2TXLivePusher pusherB = new V2TXLivePusherImpl(this,V2TXLiveMode.TXLiveMode_RTC);
...
/*
* pushURLB= "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=B&usersig=xxx";
*/
pusherB.startPush(pushURLB);
```

- **进入连麦状态：**
  主播 A 调用 V2TXLivePlayer **使用 RTC 协议拉取放连麦观众 B 的流。**

```
V2TXLivePlayer playerB = new V2TXLivePlayerImpl(mContext);
...
/**
* playURLB= "trtc://cloud.tencent.com/play/streamid?sdkappid=1400188888&userId=A&usersig=xxx";
*/
playerB.startPlay(playURLB);
```

  同时，连麦观众 B 调用 V2TXLivePlayer **切换至 RTC 协议，开始播放主播 A 的流。**

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
/**
*playURLA= "trtc://cloud.tencent.com/play/streamid?sdkappid=1400188888&userId=B&usersig=xxx";
*/
playerA.startPlay(playURLA);
```

  此时主播 A 和连麦观众 B 即可进入超低延时的实时互动场景中。
- **连麦成功后，进行混流：**
  为了保证观众可以看到连麦观众 B 的画面，这里主播 A 需要发起一次混流操作。也就是将主播A自己和连麦观众 B，混合成一路流。观众可以在一路流上看到主播和连麦观众进行互动。 A 调用 setMixTranscodingConfig 接口启动云端混流，调用时需要设置音频相关的参数，例如 音频采样率 audioSampleRate、**音频码率** audioBitrate **和 声道数** audioChannels **等。**
  如果您的业务场景中也包含视频，需同时设置视频相关的参数，例如 视频宽度 videoWidth、**视频高度** videoHeight、**视频码率** videoBitrate、**视频帧率** videoFramerate **等。**
  示例代码：

```
V2TXLiveDef.V2TXLiveTranscodingConfig config = new V2TXLiveDef.V2TXLiveTranscodingConfig();
// 设置分辨率为 720 × 1280, 码率为 1500kbps，帧率为 20FPS
config.videoWidth = 720;
config.videoHeight = 1280;
config.videoBitrate = 1500;
config.videoFramerate = 20;
config.videoGOP = 2;
config.audioSampleRate = 48000;
config.audioBitrate = 64;
config.audioChannels = 2;
config.mixStreams = new ArrayList<>();

// 主播摄像头的画面位置
V2TXLiveDef.V2TXLiveMixStream local = new V2TXLiveDef.V2TXLiveMixStream();
local.userId = "localUserId";
local.streamId = null; // 本地画面不用填写 streamID，远程需要
local.x = 0;
local.y = 0;
local.width = videoWidth;
local.height = videoHeight;
local.zOrder = 0; // zOrder 为 0 代表主播画面位于最底层
config.mixStreams.add(local);

// 连麦者的画面位置
V2TXLiveDef.V2TXLiveMixStream remoteA = new V2TXLiveDef.V2TXLiveMixStream();
remoteA.userId = "remoteUserIdA";
remoteA.streamId = "remoteStreamIdA"; // 本地画面不用填写 streamID，远程需要
remoteA.x = 400; //仅供参考
remoteA.y = 800; //仅供参考
```

```
remoteA.width = 180; //仅供参考
remoteA.height = 240; //仅供参考
remoteA.zOrder = 1;
config.mixStreams.add(remoteA);

// 连麦者的画面位置
V2TXLiveDef.V2TXLiveMixStream remoteB = new V2TXLiveDef.V2TXLiveMixStream();
remoteB.userId = "remoteUserIdB";
remoteB.streamId = "remoteStreamIdB"; // 本地画面不用填写 streamID，远程需要
remoteB.x = 400; //仅供参考
remoteB.y = 800; //仅供参考
remoteB.width = 180; //仅供参考
remoteB.height = 240; //仅供参考
remoteB.zOrder = 1;
config.mixStreams.add(remoteB);

// 发起云端混流
pusher.setMixTranscodingConfig(config);
```

> ⚠ 注意：
> 发起云端混流后，默认混流 ID，是发起混流者的 ID，如果需要指定流 ID，需要进行传入。

这样其他观众在观看时，就可以看到 A，B 两个主播的连麦互动。

步骤5：实现主播 PK

1. 主播 A 开始推流，调用 V2TXLivePusher 组件开始主播 A 的推流。

```
V2TXLivePusher pusherA = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
/**
* pushURLA= "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxx";
*/
pusherA.startPush(pushURLA);
```

2. 主播 B 开始推流，调用 V2TXLivePusher 组件开始主播 B 的推流。

```
V2TXLivePusher pusherB = new V2TXLivePusherImpl(this, V2TXLiveMode.TXLiveMode_RTC);
...
/**
* pushURLB "trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=B&usersig=xxx";
*/
pusherB.startPush(pushURLB);
```

3. 开始 PK，主播 A 和主播 B 分别调用 V2TXLivePlayer 开始播放对方的流，此时主播 A 和主播 B 即进入 RTC 连麦互动直播场景中。

```
V2TXLivePlayer playerA = new V2TXLivePlayerImpl(mContext);
...
/**
* 这里使用CDN拉流，支持flv，hls，webrtc协议，任选一种协议。flv，hls等标准协议价格更合理，webrtc快直播能够提供更低延迟的互动体验。
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidB.flv";
* playURLA= "http://3891.liveplay.myqcloud.com/live/streamidB.hls";
* playURLA= "webrtc://3891.liveplay.myqcloud.com/live/streamidB"
*/
playerA.startPlay(playURLA);

V2TXLivePlayer playerB = new V2TXLivePlayerImpl(mContext);
...
```

```
/**
* 这里使用CDN拉流，支持FLV、HLS、WebRTC协议，任选一种协议。FLV、HLS等标准协议价格更合理，WebRTC快直播能够提供更低延迟的互动体验。
* playURLB= "http://3891.liveplay.myqcloud.com/live/streamidA.flv";
* playURLB= "http://3891.liveplay.myqcloud.com/live/streamidA.hls";
* playURLB= "webrtc://3891.liveplay.myqcloud.com/live/streamidA"
*/
playerB.startPlay(playURLB);
```

4. **PK 成功后**，主播 A 和主播 B 的观众各自调用 V2TXLivePlayer 开始播放另外一名主播的推流内容。

> 说明：
> **此处开发者可能会有疑问：貌似新的 RTC 连麦方案还需要我们自己维护一套房间和用户状态，这样不是更麻烦吗？是的，没有更好的方案，只有更适合自己的方案，我们也有考虑到这样的场景：**
> - **如果对时延和并发要求并不高的场景，可以继续使用连麦互动的旧方案。**
> - **如果既想用到新版连麦方案相关的接口，但是又不想维护一套单独的房间状态，可以尝试搭配 腾讯云 IM SDK，快速实现相关逻辑。**

RTC 连麦方案怎么计费用

新版直播连麦基于 TRTC 能力实现，连麦计费逻辑参考 TRTC 音视频时长计费，**新版直播连麦套餐包即 TRTC 音视频通用套餐包**。RTC 连麦服务费用按所有参与连麦的用户产生的 视频时长 和 语音时长 来统计连麦服务产生的用量。

> 注意：
> - **时长统计精度为秒，以当月累计秒数转换成分钟数后进行计费，不足一分钟按一分钟计。**
> - **如果您与商务经理约定了 TRTC 其他的计费方式，请以实际计费方式和账单中心输出账单为准，单击 查看费用账单。**
> - **直播连麦中的 CDN 拉流观看线路依然按照云直播 带宽 或 流量 计费，连麦服务仅对连麦通话部分进行计费。**

视频时长

视频时长是指所有用户观看其他参与连麦的用户视频流的累计时间。RTC 连麦会根据用户**实际接收到的视频分辨率**划分视频档位，然后分别对不同档位的视频时长进行计费。

视频档位与分辨率的对应关系如下表所示：

视频档位	实际接收分辨率
标清 SD	不高于640 × 480（含）
高清 HD	640 × 480 – 1280 × 720（含）
全高清 FHD	1280 × 720 – 1980 × 1080（含）

- 用户观看视频时，不管该视频里面有没有包含音频，都只统计一次视频时长，不会重复计算语音时长。
- 单个用户同时观看多路视频流时，其观看的每一路视频时长将分别统计后叠加计算。

语音时长

语音时长指所有用户收听其他参与连麦的用户音频流的时间。

- 只有当用户没有观看视频流时，才会统计语音时长。
- 当用户同时收听多个用户音频流时，统计语音时长时会去除掉重叠部分的时长。

服务定价

视频互动直播服务的刊例价如下表所示：

计费项	单价（元/千分钟）
语音	7.00
标清 SD	14.00
高清 HD	28.00

计费项	单价（元/千分钟）
全高清 FHD	105.00

计费方式

即支付方式，RTC 连麦互动直播支持**预付费套餐包**和**后付费**，默认采用预付费套餐包，后付费只能通过购买的套餐包消耗完或过期后自动开通，无法直接开通。

预付费套餐包

RTC连麦互动直播服务为您提供音视频通用套餐包，可按照**1:2:4:15**分别抵扣语音、标清 SD、高清 HD 和全高清 FHD 时长，例如1分钟高清视频时长扣除4分钟通用套餐包时长。
通用套餐包定价如下表所示：

套餐包类型	套餐包时长（千分钟）	刊例价（元/千分钟）	套餐内单价（元/千分钟）	套餐包价格（元）	折扣
固定套餐包	25	7.00	6.720	168.00	96%
	250	7.00	6.352	1588.00	91%
	1000	7.00	5.968	5968.00	85%
	3000	7.00	5.630	16888.00	80%
自定义套餐包	0 < X < 25	7.00	7.000	套餐内单价乘以套餐包时长 X	100%
	25 ≤ X < 250	7.00	6.720		96%
	250 ≤ X < 1000	7.00	6.352		91%
	1000 ≤ X < 3000	7.00	5.968		85%
	X ≥ 3000	7.00	5.630		80%

> 说明：
> 表格中套餐内单价按每千分钟单价向上取整精确到小数点后3位，实际计费按每分钟单价精确到小数点后8位。

通用套餐包说明：

- 通用套餐包的有效期为购买当日 – 次年当月最后一天。
  例如，2020年05月01日购买的套餐包，其有效时间为2020年05月01日 – 2021年05月31日。
- 通用套餐包可以叠加购买，根据各类型用量实际产生的时间实时从通用套餐包中扣除相应分钟数，优先使用先过期的套餐包进行抵扣。
- 新购套餐包支付成功后5分钟左右生效，**新购套餐包生效后会立即扣除购买新套餐包当日0点起产生的未被其他套餐包抵扣过的用量**。
- 为了不影响您线上业务的正常运行，**通用套餐包用完或过期后不会自动停服**，超出套餐包的用量将采用 后付费 的计费方式。
- 所有通用套餐包到期后未消耗的分钟数将自动清零且无法恢复。

后付费

当服务用量无套餐包可抵扣或超出套餐包余量时，将采用后付费的方式，按 刊例价 计费。
RTC 连麦互动直播服务后付费有 日结 和 月结 两种结算周期。

- **日结后付费：**
  2020年09月01日起首次在 实时音视频 控制台创建 应用 的用户，后付费生效后默认采用日结**方式结算。按日计费，每天上午10点扣除前一天产生的费用。
- **月结后付费：**
  2020年08月31日及之前首次在实时音视频控制台创建 应用 的用户，后付费生效后默认采用月结**方式结算。按月计费，每月01日 – 05日从您的账户余额中扣除前一月产生的费用，详情以 计费账单 为准。

> 注意：
> - 后付费只能通过先购买套餐包，待购买的所有套餐包都消耗完或过期后自动开通，无法直接开通。
> - 结算周期无法自主变更，若您希望将月结变更为日结，可以 提交工单 寻求帮助。
> - 若您的账户因余额不足而无法抵扣账单费用时，您使用的其他腾讯云服务也可能会因为账户欠费而自动停服。例如，云端录制依赖**云直播**和**云点播**，如果腾讯云账户欠费，将导致云端录制失败。

计费示例

> 注意：
> 本文计费示例采用刊例价计算，您可以通过 购买套餐包 的方式节省费用。

纯语音连麦示例

假设用户 A、B、C 三人使用RTC连麦互动直播服务连麦30分钟。A、B、C 三人始终没有接收视频画面。
则产生的语音时长**总费用为** 语音时长单价 × 所有用户语音时长之和 = 7.00元/千分钟 × (30分钟 + 30分钟 + 30分钟) / 1000 = 0.63元 。

纯视频连麦示例

假设用户 A、B 使用RTC连麦互动直播服务连麦45分钟。A、B 都始终观看对方的视频流。A 、B 实际观看到的视频分辨率如下表所示：

时间	A 接收 B 的分辨率	B 接收 A 的分辨率
前30分钟	1280 × 720（高清）	1920 × 1080（全高清）
后15分钟	640 × 360（标清）	640 × 360（标清）

分析：

- A 产生的用量及费用：
  - A 观看 B 的分辨率前30分钟位于高清档，后15分钟位于标清档。
  - A 产生的费用为 高清视频时长单价 × 高清视频时长 + 标清视频时长单价 × 标清视频时长 = 28元/千分钟 × (30分钟 / 1000) + 14元/千分钟 × (15分钟 / 1000）= 1.05元。
- B 产生的用量及费用：
  - B 观看 A 的分辨率前30分钟位于全高清档，后15分钟位于标清档。
  - B 产生的费用为 全高清视频时长单价 × 全高清视频时长 + 标清视频时长单价 × 标清视频时长 = 105元/千分钟 × (30分钟 / 1000) + 14元/千分钟 × (15分钟 / 1000）= 3.36元。

则产生的**总费用为** 用户 A 产生的费用 + 用户 B 产生的费用 = 4.41元 。

语音时长和视频时长混合示例

假设用户 A、B 使用RTC连麦互动直播服务连麦45分钟。A 始终观看 B 的视频流；B 前30分钟观看了 A 的视频流，后15分钟没有观看 A 的视频流 仅收听 A 的音频流。A 、B 实际接收到的视频分辨率如下表所示：

时间	A 接收 B 的分辨率	B 接收 A 的分辨率
前30分钟	1280 × 720（高清）	1920 × 1080（全高清）
后15分钟	640 × 360（标清）	语音（无视频）

分析：

- A 产生的用量及费用：
  - A 接收 B 的分辨率前30分钟位于高清档，后15分钟位于标清档。
  - A 产生的费用为 高清视频时长单价 × 高清视频时长 + 标清视频时长单价 × 标清视频时长 = 28元/千分钟 × (30分钟 / 1000） + 14元/千分钟 × (15分钟 / 1000)= 1.05元。
- B 产生的用量及费用：
  - B 接收 A 的分辨率前30分钟位于全高清档，后15分钟没有接收 A 的视频流。
  - B 产生的费用为 全高清视频时长单价 × 全高清视频时长 + 语音时长单价 × 语音时长 = 105元/千分钟 × (30分钟 / 1000) + 7元/千分钟 × (15分钟 / 1000）= 3.255元。

则产生的**总费用为** 用户 A 产生的费用 + 用户 B 产生的费用 = 4.305元 。

常见问题

1. 为什么使用 V2TXLivePusher&V2TXLivePlayer 接口时，同一台设备不支持使用相同 streamid 同时推流和拉流，而 TXLivePusher&TXLivePlayer 可以支持?

是的，目前 V2TXLivePusher&V2TXLivePlayer 是 腾讯云 TRTC 协议实现，其基于 UDP 的超低延时的私有协议暂时还不支持**同一台设备，使用相同的 streamid，一边推超低延时流，一边拉超低延时的流**，同时考虑到用户的使用场景，所以暂时并未支持，后续会酌情考虑此问题的优化。

2. 服务开通 章节中生成参数都是什么意思呢?

SDKAppID 用于标识您的应用，UserID 用于标识您的用户，而 UserSig 则是基于前两者计算出的安全签名，它由 HMAC SHA256 加密算法计算得出。只要攻击者不能伪造 UserSig，就无法盗用您的云服务流量。UserSig 的计算原理如下图所示，其本质就是对 SDKAppID、UserID、ExpireTime 等关键信息进行了一次哈希加密：

```
//UserSig 计算公式，其中 secretkey 为计算 usersig 用的加密密钥

usersig = hmacsha256(secretkey, (userid + sdkappid + currtime + expire +
base64(userid + sdkappid + currtime + expire)))
```

3. V2TXLivePusher&V2TXLivePlayer 如何设置音质或者画质呢?

我们有提供对应的音质和画质的设置接口，详情见 API 文件：设置推流音频质量 和 设置推流视频参数。

4. 收到一个错误码：-5，代表什么意思?

-5表示由于许可证无效，因此无法调用API，对应的枚举值为：V2TXLIVE_ERROR_INVALID_LICENSE，更多错误码请参见 API 状态码。

5. RTC连麦方案的时延性有可以参考的数据吗?

新的 RTC 连麦方案中，主播连麦的延时 < 200ms，主播和观众的延时在 100ms - 1000ms。

6. RTC 推流成功后，使用 CDN 拉流一直提示404?

检查一下是否有开启实时音视频服务的旁路直播功能，基本原理是 RTC 协议推流后，如果需要使用 CDN 播放，RTC 会在后台服务中旁路流信息到 CDN 上。

Web

最近更新时间：2022-03-04 10:58:11

本文主要介绍以观众身份进入直播间，然后发起连麦互动的场景，以主播身份进入房间进行直播的场景跟实时音视频通话场景流程一样，请参见 实时音视频通话。

示例

您可以单击 Demo 前往体验音视频功能，也可以登录 GitHub 获取本文档相关的示例代码。

步骤1：创建 Client 对象

通过 TRTC.createClient() 方法创建 Client 对象，参数设置：

- mode：互动直播模式，设置为 live
- sdkAppId：您从腾讯云申请的 sdkAppId
- userId：用户 ID
- userSig：用户签名

```
const client = TRTC.createClient({
mode: 'live',
sdkAppId,
userId,
userSig
});
```

步骤2：以观众身份进入直播房间

调用 Client.join() 进入音视频通话房间。参数设置：

- roomId：房间 ID
- role：用户角色
  - anchor：主播，主播角色具有发布本地流和接收远端流的权限。默认是主播角色。
  - audience：观众，观众角色只有接收远端流的权限，没有发布本地流的权限。若观众想要跟主播连麦互动，需要通过 Client.switchRole() 切换角色至anchor主播角色后再发布本地流。

```
// 以观众角色进房收看
client
.join({ roomId, role: 'audience' })
.then(() => {
console.log('进房成功');
})
.catch(error => {
console.error('进房失败 ' + error);
});
```

步骤3：收看直播

1. 远端流通过监听事件 client.on('stream-added') 获取，收到该事件后，通过 Client.subscribe() 订阅远端音视频流。

> 说明：
> 请在 Client.join() 进房前注册 client.on('stream-added') 事件以确保您不会错过远端用户进房通知。

```
client.on('stream-added', event => {
const remoteStream = event.stream;
console.log('远端流增加: ' + remoteStream.getId());
```

```
//订阅远端流
client.subscribe(remoteStream);
});
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('远端流订阅成功：' + remoteStream.getId());
// 播放远端流
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

2. 在远端流订阅成功事件回调中，通过调用 Stream.play() 方法在网页中播放音视频。play 方法接受一个 div 元素 ID 作为参数，SDK 内部会在该 div 元素下自动创建相应的音视频标签并在其上播放音视频。

```
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('远端流订阅成功：' + remoteStream.getId());
// 播放远端流
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

步骤4：跟主播连麦互动

步骤4.1：切换角色

使用 Client.switchRole() 切换角色到 anchor 主播角色。

```
client
.switchRole('anchor')
.then(() => {
// 角色切换成功，现在是主播角色
})
.catch(error => {
console.error('角色切换失败 ' + error);
});
```

步骤4.2：连麦互动

1. 使用 TRTC.createStream() 方法创建本地音视频流。以下实例从摄像头及麦克风中采集音视频流，参数设置如下：
   - userId：本地流所属用户 ID
   - audio：是否开启音频
   - video：是否开启视频

```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
```

2. 调用 LocalStream.initialize() 初始化本地音视频流。

```
localStream
.initialize()
.then(() => {
console.log('初始化本地流成功');
})
.catch(error => {
console.error('初始化本地流失败 ' + error);
});
```

3. 初始化本地流成功时，播放本地流。

```
localStream
.initialize()
.then(() => {
console.log('初始化本地流成功');
localStream.play('local_stream');
})
.catch(error => {
console.error('初始化本地流失败 ' + error);
});
```

4. 在本地流初始化成功后，调用 Client.publish() 方法发布本地流，开启观众连麦互动。

```
client
.publish(localStream)
.then(() => {
console.log('本地流发布成功');
})
.catch(error => {
console.error('本地流发布失败 ' + error);
});
```

步骤5：退出直播房间

直播结束时调用 Client.leave() 方法退出直播房间，整个直播会话结束。

```
client
.leave()
.then(() => {
// 退房成功
})
.catch(error => {
console.error('退房失败 ' + error);
});
```

> 注意：
> 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

连麦互动 RTMP App

最近更新时间：2022-03-14 14:53:24

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	✓	✓	–	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

> 注意：
> 旧版移动直播连麦资源包已下线，仅限连麦老用户购买。

功能介绍

TXLivePusher 和 TXLivePlayer 这两个基础组件可以比较容易的实现推流和拉流功能，但如果想要实现复杂的直播连麦功能，就需要借助我们提供给您的 MLVBLiveRoom 组件，该组件基于云直播（CSS）和即时通信（IM）两个 PaaS 服务组合而成，支持：

- 主播创建新的直播间开播，观众进入直播间观看。
- 主播和观众进行视频连麦互动。
- 两个不同房间的主播 PK 互动。
- 每一个直播间都有一个不限制房间人数的聊天室，支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物。

> 说明：
> 连麦互动功能支持中国内地（大陆）地区使用，暂不支持中国港澳台/境外地区。

功能体验

我们提供了 iOS、Android 以及微信小程序三个平台上的直播连麦体验，它们均是使用 MLVBLiveRoom 组件实现的直播加连麦功能：

- **iOS**
  进入 App Store 安装应用“小直播”，注册一个账号即可开始体验。
- **Android**
  下载 apk 安装包，安装“小直播”，注册一个账号即可开始体验。
- **微信小程序**
  打开微信，选择**发现** > **小程序**，搜索“腾讯视频云”，单击“移动直播”功能即可体验。

代码对接

Step1. 下载 LiteAVSDK 和 MLVBLiveRoom 组件

移动直播提供的连麦能力需要依赖三个组件：

- LiteAVSDK：闭源，负责直播推流，直播拉流，以及连麦视频通话功能。
- TIMSDK：闭源，负责构建直播聊天室，以及聊天室中用户之间的消息传输功能。
- MLVBLiveRoom：开源，基于 LiteAVSDK 和 TIMSDK 搭建一个支持连麦互动和消息互动的“直播间”。

我们已将上述组件均托管在 Github 上，clone 下来便可使用，关键组件的具体获取路径如下表所示：

所属平台	LiteAVSDK	TIMSDK	MLVBLiveRoom 组件	示例代码
iOS	MLVBSDK	TIMSDK	MLVBLiveRoom	SimpleCode
Android	MLVBSDK	TIMSDK	MLVBLiveRoom	SimpleCode

Step2. 申请 License

下载 LiteAVSDK 后需要 License 授权才能使用，请阅读 License 申请 了解 License 的申请方法和使用方法。

- iOS
  建议在[AppDelegate application:didFinishLaunchingWithOptions:]中添加：

```
[TXLiveBase setLicenceURL:LicenceUrl key:Key];
```

- Android
  建议在 application 中添加：

```
TXLiveBase.getInstance().setLicence(context, LicenceUrl, Key);
```

Step3. 购买连麦套餐包

由于连麦功能会使用到高速专线来降低音视频传输延迟，这部分功能需要额外购买套餐包才能开通，否则移动直播的各端 SDK 只能使用云直播的普通服务（推流和拉流），并不能开启连麦功能。

- 仅限老用户前往 **云直播控制台 > 直播 SDK > 直播连麦**，单击**购买连麦包**
- 移动直播连麦计费说明

注意：

- **旧版移动直播连麦资源包已下线，仅限连麦老用户购买。**
- 默认开通连麦服务需要先购买正式的连麦套餐包（非体验包），开通后可选择继续购买套餐包进行消费抵扣，也可以按照后付费日结计费进行结算。
- 若您购买的套餐包为体验包，用尽后次日会自动停止连麦服务，超出部分仍会按照后付费计费。
- 使用连麦的**双方**是**按连麦时长计费**，普通观众观看会先通过直播混流，将连麦画面混合后通过 CDN 播放来降低播放成本。
  - 观众观看混流后产生的直播费用，按照直播流量/带宽计费。
  - 直播混流功能会产生标准转码费用，按照输出的分辨率和时长计费。

Step4. 在应用管理中添加一个新的应用

进入**云直播控制台 > 直播 SDK>**应用管理，单击**创建应用**。待应用创建完成后，记录其 SDKAPPID 信息。

说明：
该操作的目的是创建一个即时通信 IM 应用，并将当前直播账号和该即时通信 IM 应用绑定起来。即时通信 IM 应用能为小直播 App 提供聊天室和连麦互动的能力。

Step5. 登录房间服务

MLVBLiveRoom 单靠一个终端的组件无法独自运行，它依赖一个后台服务为其实现房间管理和状态协调，这个后台服务我们称之为**房间服务**（RoomService）。而要使用这个房间服务，MLVBLiveRoom 就需要先进行**登录**（login）。

MLVBLiveRoom 的 login 函数需要指定相关参数：

参数	类型	填写方案
sdkAppID	数字	当前应用的 AppID，在 Step4 中可以获取到。
userID	字符串	当前用户在您的帐号系统中的 ID。
userName	字符串	用户名（昵称）。
userAvatar	字符串	用户头像的 URL 地址。
userSig	字符串	登录签名，计算方法请参见 计算 UserSig。

说明：

- 由于 login 是一个需要跟后台服务器通讯的过程，建议等待 login 函数的异步回调后再调用其他函数。
- 后台接口限制并发为每秒100次请求，若您有高并发请求请提前 联系我们 处理，避免影响服务调用。

Step6. 获取房间列表（非必需）

说明：
如果您希望使用自己的房间列表，该步骤可跳过，但需要您在 Step7 中自行指定 roomID。为避免房间号重复，建议使用主播的 userID 作为 roomID。

不管是主播还是观众都需要有一个房间列表，调用 MLVBLiveRoom 的 **getRoomList** 接口可以获得一个简单的房间列表：

- 当主播通过 createRoom 创建一个新房间时，房间列表中会相应地增加一条新的房间信息。
- 当主播通过 exitRoom 退出房间时，房间列表中会移除该房间。

列表中每个房间都有对应的 roomInfo，由主播通过 createRoom 创建房间时传入，为提高扩展性，建议将 roomInfo 定义为 JSON 格式。

Step7. 主播开播

主播开播前，需要先调用 MLVBLiveRoom 中的 **startLocalPreview** 接口开启本地摄像头预览，该函数需要传入 view 对象用于显示摄像头的视频影像，这期间 MLVBLiveRoom 会申请摄像头使用权限。同时，主播也可以对着摄像头调整美颜和美白的具体效果。
然后调用 **createRoom** 接口，MLVBLiveRoom 会在后台的房间列表中新建一个直播间，同时主播端会进入直播状态。

说明：

- 为避免房间号重复，建议使用主播的 userID 作为 roomID。如果您不手动设置 roomID，后台将会自动为您分配一个 roomID。
- 如果您想要管理房间列表，可以先由您的服务器确定 roomID，再通过 createRoom、enterRoom 和 exitRoom 接口使用 MLVBLiveRoom 的连麦能力。

Step8. 观看直播

观众通过 MLVBLiveRoom 中的 **enterRoom** 接口可以进入直播间观看视频直播，enterRoom 函数需要传入 view 对象用于显示直播流的视频影像。

进入房间后，通过调用 **getAudienceList** 接口可以获取观众列表。如果少于30名观众，列表会展示全部观众信息。如果多于30名观众，列表仅展示新进入房间的30名观众的信息。

Step9. 弹幕消息

MLVBLiveRoom 包装了 TIMSDK 的消息发送接口，您可以通过 **sendRoomTextMsg** 函数发送普通的文本消息（用于弹幕），也可以通过 **sendRoomCustomMsg** 发送自定义消息（用于点赞，送花等等）。

> **注意：**
> 腾讯云 IM 的直播聊天室，每秒钟最多可以收取40条的消息。如果您以高于40条/次的速度刷新 UI 上的弹幕界面，很容易导致 CPU 100%，请注意控制刷新频率，避免高频刷新。

Step10. 观众与主播连麦

步骤	角色	详情
第一步	观众	观众调用 requestJoinAnchor() 向主播发起连麦请求。
第二步	主播	主播会收到 MLVBLiveRoomDelegate#onRequestJoinAnchor(AnchorInfo, String) 通知，之后可以展示一个 UI 提示，询问主播要不要接受连麦。
第三步	主播	主播调用 responseJoinAnchor() 确定是否接受观众的连麦请求。
第四步	观众	观众会收到 MLVBLiveRoomDelegate.RequestJoinAnchorCallback 回调通知，得知请求是否被同意。
第五步	观众	观众如果请求被同意，则调用 startLocalPreview() 开启本地摄像头，如果 App 还没有取得摄像头和麦克风权限，会触发 UI 提示用户授权摄像头和麦克风的使用权限。
第六步	观众	观众调用 joinAnchor() 正式进入连麦状态。
第七步	主播	当观众进入连麦状态后，主播就会收到 MLVBLiveRoomDelegate#onAnchorEnter(AnchorInfo) 通知。
第八步	主播	主播调用 startRemoteView() 就可以看到连麦观众的视频画面。
第九步	观众	如果直播间里已经有其他观众正在跟主播进行连麦，那么新加入的这位连麦观众也会收到 onAnchorJoin() 通知，用于展示（startRemoteView()）其他连麦者的视频画面。
第九步	主播或观众	主播或观众随时都可以通过 quitJoinAnchor() 接口退出连麦状态，同时，主播还可以通过 kickoutJoinAnchor() 接口移除连麦观众。

> **说明：**
> MLVBLiveRoom 在设计上最多支持10人同时连麦，但出于兼容低端 Android 终端和实际体验效果的考虑，建议将同时连麦人数控制在6人以下。

Step11. 主播间跨房间 PK

主播间跨房 PK 常被用于活跃直播平台的氛围，提升打赏频率，对平台的主播人数有一定要求。目前常见的主播 PK 方式是将所有愿意 PK 的主播“圈”在一起，再后台进行随机配对，每次 PK 都有一定时间要求，例如5分钟，超过后即结束 PK 状态。
由于我们暂时未在 MLVBLiveRoom 的房间服务里加入配对逻辑，因此目前仅提供了基于客户端 API 接口的简单 PK 流程，您可以通过即时通信 IM 服务的消息下发 REST API 接口，由您的配对服务器，将配对开始、配对结束等指令发送给指定的主播，从而实现服务器控制的目的。如果采用此种控制方式，下述步骤中的第三步实现为默认接受即可。

步骤	角色	详情
第一步	主播 A	主播 A 调用 requestRoomPK() 向主播 B 发起连麦请求。

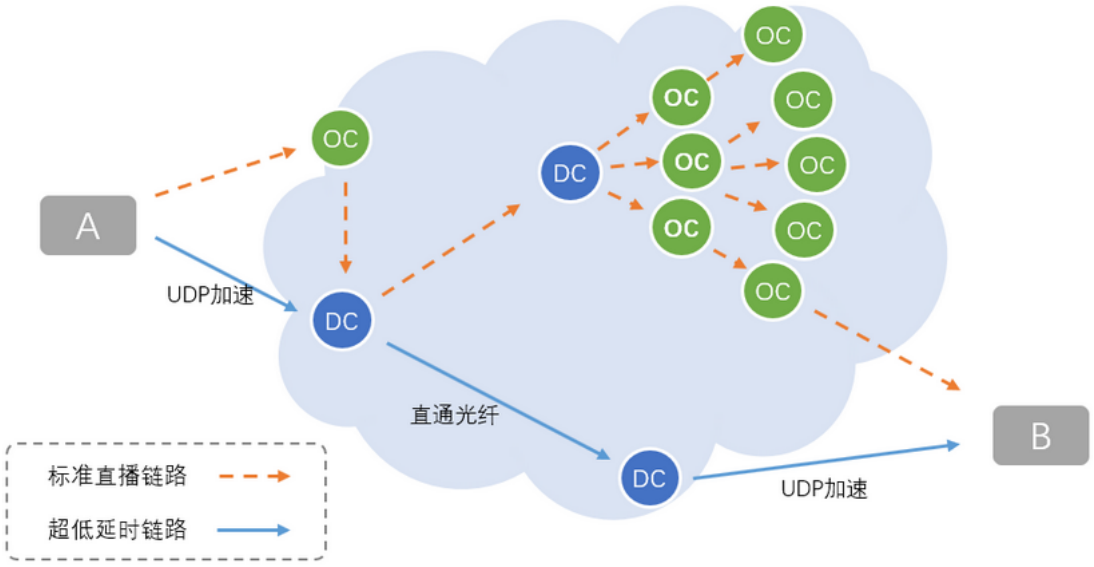
步骤	角色	详情
第二步	主播 B	主播 B 会收到 MLVBLiveRoomDelegate#onRequestRoomPK(AnchorInfo) 回调通知。
第三步	主播 B	主播 B 调用 responseRoomPK() 确定是否接受主播 A 的 PK 请求。如果采用服务器配对的 PK 方案，此处可以默认接受，不需要由主播 B 来决策。
第四步	主播 B	主播 B 在接受主播 A 的请求后，即可调用 startRemoteView() 来显示主播 A 的视频画面。
第五步	主播 A	主播 A 会收到 MLVBLiveRoomDelegate.RequestRoomPKCallback 回调通知，可以得知请求是否被同意，如果请求被同意，则可以调用 startRemoteView() 显示主播 B 的视频画面。
第六步	主播 A 或 B	主播 A 或 B 均可以通过调用quitRoomPK()接口结束 PK 状态。

常见问题

移动直播是不是使用 RTMP 协议进行连麦?

不是。腾讯云采用了两种传输通道才实现了直播 + 连麦功能：

- 直播采用标准的 HTTP-FLV 协议，使用标准 CDN 线路，没有并发观看人数的限制，且带宽成本很低，但延迟一般在3s以上。
- 连麦采用 UDP 协议，使用专用加速线路，延迟一般在500ms以内，但由于线路成本较高，因此采用连麦时长进行计费。



通道	直播通道	连麦通道
通讯延迟	⩾ 3s	⩽ 500ms
底层协议	HTTP-FLV 协议	UDP 协议
价格/费用	按带宽计费	按时长计费
最高并发	无上限	⩽ 10人
TXLivePusher	setVideoQuality 为 SD、HD、FHD	setVideoQuality 为 MAIN_PUBLISHER、SUB_PUBLISHER
TXLivePlayer	PLAY_TYPE_LIVE_FLV	PLAY_TYPE_LIVE_RTMP_ACC
播放 URL	普通的 FLV 地址	带防盗链签名的 RTMP-ACC 地址

小程序

最近更新时间：2022-03-14 14:52:48

功能介绍

<live-pusher> 和 <live-player> 这两个基础标签可以比较方便的实现推流和拉流功能，但如果想要实现复杂的直播连麦功能，就需要借助我们提供给您的 **<mlvb-live-room>** 小程序组件，该组件基于云直播和即时通信（IM）两款 PAAS 产品组合而成，支持：

- 主播创建新的直播间开播，观众进入直播间观看。
- 主播和观众进行视频连麦互动。
- 每一个直播间都有一个不限制房间人数的聊天室，支持发送各种文本消息和自定义消息，自定义消息可用于实现弹幕、点赞和礼物。

> **说明：**
> 连麦互动功能支持中国内地（大陆）地区使用，暂不支持中国港澳台/境外地区。

功能体验

我们提供了 iOS、Android 以及微信小程序三个平台上的直播连麦体验，它们都是使用 MLVBLiveRoom 组件实现的直播加连麦功能：

- **微信小程序**
  打开微信，选择**发现** > **小程序**，搜索“腾讯视频云”，单击“移动直播”功能即可体验。
- **iOS**
  进入 App Store 安装应用“小直播”，注册一个账号即可开始体验。
- **Android**
  下载 Apk 安装包，安装应用“小直播”，注册一个账号即可开始体验。

计费说明

小程序 <mlvb-live-room> 组件为开发者提供了云直播连麦的能力，具有低卡顿、低延时和易接入等特点。如果您希望用它来快速实现直播连麦应用，那么您需要购买直播流量资源包、移动直播 SDK License、移动直播连麦资源包和即时通信 IM 套餐包。详细计费说明请查看 云直播计费说明 和 即时通信 IM 定价。

> ⚠ 注意：
> 旧版移动直播连麦资源包已下线，仅限连麦老用户购买。

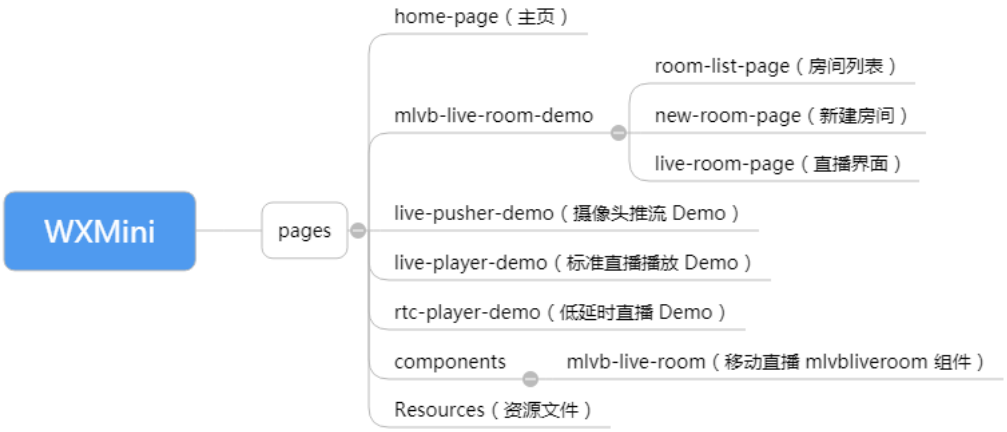
示例代码

- 示例代码：Github
- 组件 API：<mlvb-live-room>

接入指引

Step1. 下载 <mlvb-live-room> 小程序组件

单击 Github 下载 <mlvb-live-room> 组件代码，其目录结构如下：



Step2. 在工程中引入组件

在当前页面的 JSON 配置文件里必须引用 <mlvb-live-room> 组件，因为 <mlvb-live-room> 并非微信小程序的原生标签，需要用 usingComponents 指定加载路径。

```
"usingComponents": {
"mlvb-live-room": "/pages/components/mlvb-live-room/mlvbliveroomview"
}
```

Step3. 购买连麦套餐包

由于连麦功能会使用到高速专线来降低音视频传输延迟，这部分功能需要额外购买套餐包才能开通，否则移动直播的各端 SDK 只能使用云直播的普通服务（推流和拉流），并不能开启连麦功能。

- 仅限老用户前往 **云直播控制台 > 直播 SDK > 直播连麦**，单击**购买连麦包**
- 移动直播连麦计费说明

Step4. 在应用管理中添加一个新的应用

选择**云直播控制台 > 直播SDK> 应用管理**，单击**创建应用**开始创建一个新的应用。

应用管理

应用创建完成后要记录好 SDKAppID，这个 ID 会在 Step5 中使用到。

> **说明：**
> 这一步的目的是创建一个 TIM 即时通信 IM 应用，并将当前直播账号和即时通信 IM 应用绑定起来，即时通信 IM 应用主要提供聊天室和连麦互动的能力。

Step5. 登录房间服务（必需）

- 在使用 <mlvb-live-room> 标签前需要先调用 mlvbliveroomcore.js 中的 login() 方法进行登录。
- <mlvb-live-room> 单靠 Github 上的这些 javascript 代码是无法独自运行的，它依赖一个后台服务为其实现房间管理和主播间的状态同步，这个后台服务我们称之为**房间服务**（RoomService）。而要使用这个房间服务，<mlvb-live-room> 就需要先进行登录（login）。
- <mlvb-live-room> 的 login 函数需要指定一些参数，这些参数的填写方式如下：

参数	类型	填写方案
sdkAppID	数字	当前应用的 AppID，在 Step4 中可以获取到。
userID	字符串	当前用户在您的账号系统中的 ID。
userName	字符串	用户名（昵称）。
userAvatar	字符串	用户头像的 URL 地址。
userSig	字符串	登录签名，计算方法请参见 计算 UserSig。

```
//如下示例代码中的路径地址可能与您当前工程配置不符，请根据当前工程的具体情况进行相应地修改
var liveroom = require('components/mlvb-live-room/mlvbliveroomcore.js');
...
var loginInfo = {
sdkAppID: this.sdkAppID,
userID: this.userID,
userSig: this.userSig,
userName: this.userName,
userAvatar: this.userAvatar
}
liveroom.login({
data: loginInfo,
success: options.success,
fail: options.fail
});
```

> 注意：
> 后台接口限制并发为每秒100次请求，若您有高并发请求请提前 联系我们 处理，避免影响服务调用。

Step6. 获取房间列表（非必需）

您可以通过调用 mlvbliveroomcore.js 中的 getRoomList 方法来获取房间列表。

```
var liveroom = require('components/mlvb-live-room/mlvbliveroomcore.js');
...
liveroom.getRoomList({
data: {
index: 0, //列表索引号
cnt: 20 //要拉取的列表个数
},
success: function (ret) {
console.log('获取房间列表:', ret.rooms);
},
fail: function (ret) {
console.error('获取房间列表失败:', ret);
}
});
```

> 说明：
> 如果您希望使用自己的房间列表，这一步可以省略，因为您可以在 Step7 中自行指定 roomID，直播间列表的管理可以自行处理。

Step7. 在 UI 层添加视频标签

您需要在 page 目录下的 wxml 文件中添加 <mlvb-live-room> 标签。

```
<mlvb-live-room id="id_liveroom" wx:if="{{showLiveRoom}}"
roomid="{{roomID}}" role="{{role}}" roomname="{{roomName}}"
beauty="{{beauty}}" template="float"
bindRoomEvent="onRoomEvent"> // 绑定事件回调函数

// 此处可以添加<cover-view>
</mlvb-live-room>
```

> 说明：
> 示例代码请参见 room.wxml，Demo 中的直播主界面，主播和观众都共用这一个 xml 界面配置。

Step8. 主播开播和观众观看

Step7 仅仅是实现了 UI 布局相关工作，不管是主播开播，还是观众端开始观看，都需要调用 mlvbliveroomview.js 中的 start() 方法，并且在此之前还需要设置 <mlvb-live-room> 标签中的若干属性。

```
self.setData({
roomID: options.roomID,
roomName: options.roomName,
userName: options.userName,
pureAudio: JSON.parse(options.pureAudio),
role: role,
showLiveRoom: true
}, function () {
self.start(); // 启动组件进行开播或者播放
})
```

> 说明：
> 示例代码请参见 room.js，demo 中的主要逻辑代码，包含主播开播、观众观看以及主播和观众连麦的相关逻辑。

Step9. 连麦互动

观众可以通过调用 mlvbliveroomview.js 中的 requestJoinAnchor() 方法向主播发起连麦请求。

```
// 观众：可以向主播发起连麦请求
var liveroom = this.selectComponent("#id_liveroom");
liveroom.requestJoinAnchor();
```

此时主播可以通过在 Step7 中绑定的 onRoomEvent 函数接收到这一请求，并可以调用 mlvbliveroomview.js 中的 respondJoinAnchor() 函数进行“接受”或者“拒绝”的回应。

```
// 主播：可以接受或者拒绝连麦请求
var liveroom = this.selectComponent("#id_liveroom");
liveroom.respondJoinAnchor(true, aduience);
```

> 说明：
> 示例代码请参见 room.js，Demo 中的主要逻辑代码，包含主播开播、观众观看以及主播和观众连麦的相关逻辑。

Step10. 定制 UI 界面

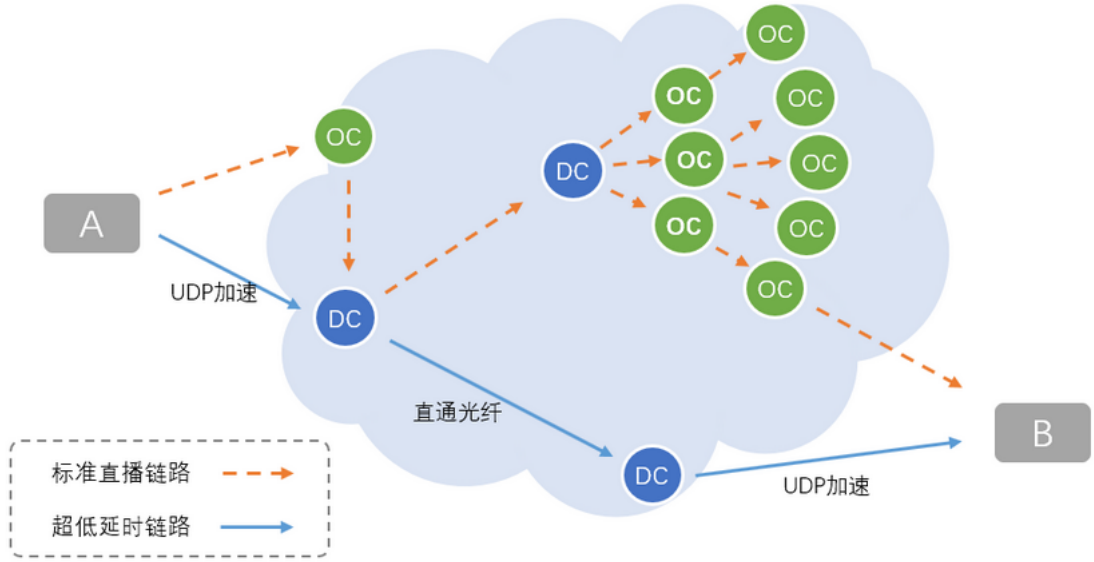
如果我们默认实现的界面布局不符合您的要求，您可以根据微信小程序的“界面模版”规范进行定制：

- 您可以直接修改 floattemplate.wxml 模版文件和 floattemplate.wxss 样式文件。
- 您也可以增加新的模版 wxml 和样式文件 wxss，但新加的文件需要在 mlvbliveroomview.wxml 和 mlvbliveroomview.wxss 两个文件中增加相关的配置。

常见问题

<mlvb-live-room> 是不是使用 RTMP 协议进行连麦?

不是的。
腾讯云采用了两种传输通道才实现了直播 + 连麦功能，其中直播采用标准的 HTTP-FLV 协议，走标准 CDN 线路，没有并发观看人数的限制，且带宽成本很低，但延迟一般在3s以上。
连麦则采用了 UDP 协议，走专用加速线路，延迟一般在500ms以内，但由于线路成本较高，因此采用连麦时长进行计费。



通道	直播通道	连麦通道
通讯延迟	⩾ 3s	⩽ 500ms

通道	直播通道	连麦通道
底层协议	HTTP-FLV 协议	UDP 协议
价格/费用	按带宽计费	按时长计费
最高并发	无上限	≤ 10人
<live-pusher> 的 mode 参数	HD	RTC
<live-player> 的 mode 参数	LIVE	RTC
播放 URL	普通的 FLV 地址	带防盗链签名的 RTMP-ACC 地址

多人通话
跑通通话模式（iOS&Mac）

最近更新时间：2022-03-04 10:56:01

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

适用场景

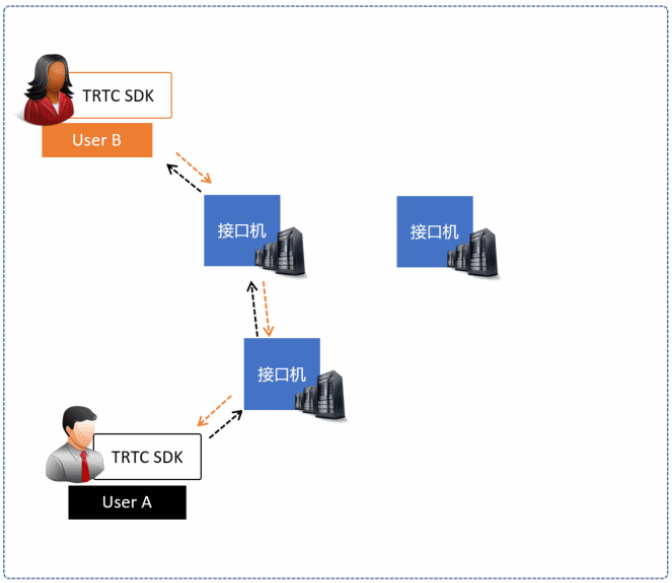
音视频通话TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的音视频通话 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

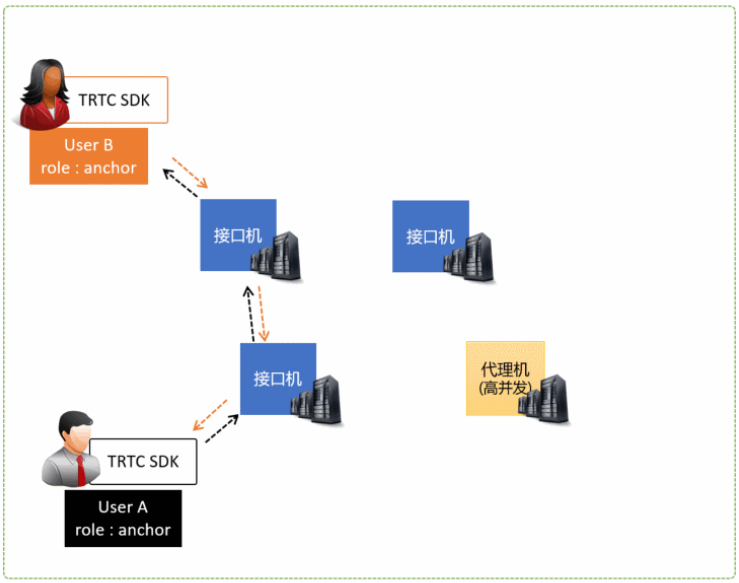
音视频通话TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **接口机**
  该类节点都采用最优质的线路和高性能的机器，善于处理端到端的低延时连麦通话。
- **代理机**
  该类节点都采用普通的线路和性能一般的机器，善于处理高并发的拉流观看需求。

在通话模式下，音视频通话TRTC 房间中的所有用户都会被分配到接口机上，相当于每个用户都是“主播”，每个用户随时都可以发言（最高的上行并发限制为50路），因此适合在线会议等场景，但单个房间的人数限制为300人。



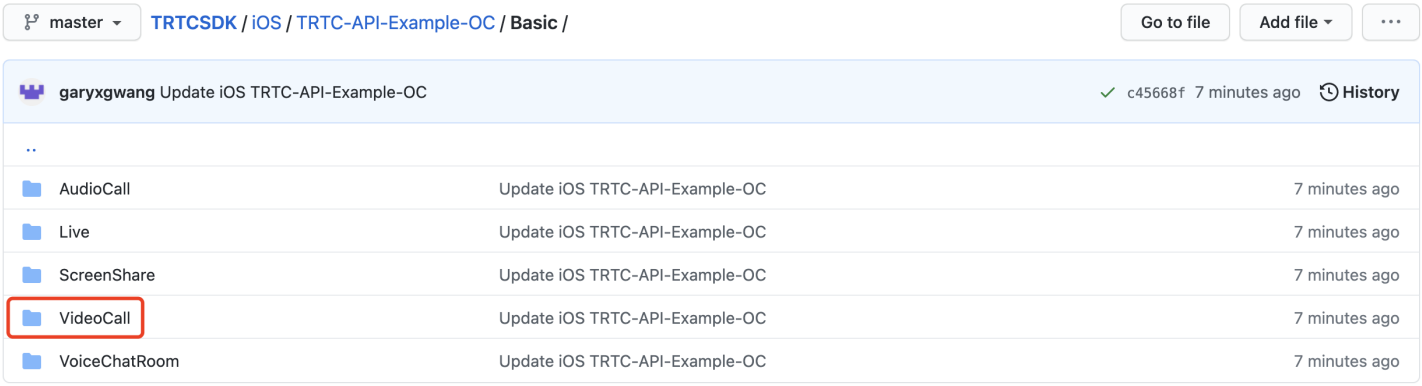
通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。



> 说明：
> 如果访问 Github 较慢，您也可以直接下载 TXLiteAVSDK_TRTC_iOS_latest.zip。

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 TRTC SDK 集成到项目中。

方式一：使用 CocoaPods 集成

1. 安装 CocoaPods，具体操作请参考 CocoaPods 官网安装说明。
2. 打开您当前项目根目录下的 Podfile 文件，添加以下内容：

> 说明：
> 如果该目录下没有 Podfile 文件，请先执行 pod init 命令新建文件再添加以下内容。

```
target 'Your Project' do
pod 'TXLiteAVSDK_TRTC'
end
```

3. 执行以下命令安装 TRTC SDK 。

```
pod install
```

安装成功后当前项目根目录下会生成一个 xcworkspace 文件。

4. 打开新生成的 xcworkspace 文件即可。

方式二：下载 ZIP 包手动集成

如果您暂时不想安装 CocoaPods 环境，或者已经安装但是访问 CocoaPods 仓库比较慢，您可以直接下载 ZIP 压缩包，并参考 快速集成(iOS) 将 SDK 集成到您的工程中。

步骤2：添加媒体设备权限

在 Info.plist 文件中添加摄像头和麦克风的申请权限：

Key	Value
Privacy – Camera Usage Description	描述使用摄像头权限的原因，例如，需要访问您的相机权限，开启后视频聊天才会有画面
Privacy – Microphone Usage Description	描述使用麦克风权限的原因，例如，需要访问您的麦克风权限，开启后聊天才会有声音

步骤3：初始化 SDK 实例并监听事件回调

1. 使用 sharedInstance() 接口创建 TRTCCloud 实例。

```
// 创建 trtcCloud 实例
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;
```

2. 设置delegate属性注册事件回调，并监听相关事件和错误通知。

```
// 错误通知是要监听的，需要捕获并通知用户
- (void)onError:(TXLiteAVError)errCode errMsg:(NSString *)errMsg extInfo:(NSDictionary *)extInfo {
if (ERR_ROOM_ENTER_FAIL == errCode) {
[self toastTip:@"进房失败"];
[self.trtcCloud exitRoom];
}
}
```

步骤4：组装进房参数 TRTCParams

在调用 enterRoom() 接口时需要填写一个关键参数 TRTCParams，该参数包含的必填字段如下表所示。

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1Ssll...
roomId	数字	默认不支持字符串类型的房间号，字符串类型的房间号会影响进房速度。如果您确实需要支持字符串类型的房间号，可以 提交工单 联系我们。	29834

> ⚠ 注意：
> TRTC 同一时间不支持两个相同的 userId 进入房间，否则会相互干扰。

步骤5：创建并进入房间

1. 调用 enterRoom() 即可加入 TRTCParams 参数中 roomId 代指的音视频房间。如果该房间不存在，SDK 会自动创建一个以字段 roomId 的值为房间号的新房间。
2. 请根据应用场景设置合适的 **appScene** 参数，使用错误可能会导致卡顿率或画面清晰度不达预期。
   - 视频通话，请设置为 TRTCAppScene.videoCall。
   - 语音通话，请设置为 TRTCAppScene.audioCall。
3. 进房成功后，SDK 会回调 onEnterRoom(result) 事件。其中，参数 result 大于0时表示进房成功，具体数值为加入房间所消耗的时间，单位为毫秒（ms）；当 result 小于0时表示进房失败，具体数值为进房失败的错误码。

```
- (void)enterRoom() {
TRTCParams *params = [TRTCParams new];
params.sdkAppId = SDKAppID;
params.roomId = _roomId;
params.userId = _userId;
params.role = TRTCRoleAnchor;
params.userSig = [GenerateTestUserSig genTestUserSig:params.userId];
[self.trtcCloud enterRoom:params appScene:TRTCAppSceneVideoCall];
}

- (void)onEnterRoom:(NSInteger)result {
if (result > 0) {
[self toastTip:@"进房成功"];
} else {
[self toastTip:@"进房失败"];
}
}
```

⚠ 注意：

- 如果进房失败，SDK 同时还会回调 onError 事件，并返回参数 errCode（错误码）、errMsg（错误原因）以及 extraInfo（保留参数）。
- 如果已在某一个房间中，则必须先调用 exitRoom() 退出当前房间，才能进入下一个房间。
- 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

步骤6：订阅远端的音视频流

SDK 支持自动订阅和手动订阅。

自动订阅模式（默认）

在自动订阅模式下，进入某个房间之后，SDK 会自动接收房间中其他用户的音频流，从而达到最佳的“秒开”效果：

1. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知，SDK 会自动播放这些远端用户的声音。
2. 您可以通过 muteRemoteAudio(userId, mute: true) 屏蔽某一个 userId 的音频数据，也可以通过 muteAllRemoteAudio(true) 屏蔽所有远端用户的音频数据，屏蔽后 SDK 不再继续拉取对应远端用户的音频数据。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知，但此时 SDK 未收到该如何展示视频数据的指令，因此不会自动处理视频数据。您需要通过调用 startRemoteView(userId, view: view) 方法将远端用户的视频数据和显示 view 关联起来。
4. 您可以通过 setRemoteViewFillMode() 指定视频画面的显示模式：
   - Fill 模式：表示填充，画面可能会等比放大和裁剪，但不会有黑边。
   - Fit 模式：表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。
5. 您可以通过 stopRemoteView(userId) 可以屏蔽某一个 userId 的视频数据，也可以通过 stopAllRemoteView() 屏蔽所有远端用户的视频数据，屏蔽后 SDK 不再继续拉取对应远端用户的视频数据。

```
// 实例代码：根据通知订阅（或取消订阅）远端用户的视频画面
- (void)onUserVideoAvailable:(NSString *)userId available:(BOOL)available {
UIView* remoteView = remoteViewDic[userId];
if (available) {
[_trtcCloud startRemoteView:userId streamType:TRTCVideoStreamTypeSmall view:remoteView];
} else {
[_trtcCloud stopRemoteView:userId streamType:TRTCVideoStreamTypeSmall];
}
}
```

ⓘ 说明：

如果您在收到 onUserVideoAvailable() 事件回调后没有立即调用 startRemoteView() 订阅视频流，SDK 将会在5s内停止接收来自远端的视频数据。

手动订阅模式

您可以通过 setDefaultStreamRecvMode() 接口将 SDK 指定为手动订阅模式。在手动订阅模式下，SDK 不会自动接收房间中其他用户的音视频数据，需要您手动通过 API 函数触发。

1. 在进房前调用 setDefaultStreamRecvMode(false, video: false) 接口将 SDK 设定为手动订阅模式。
2. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知。此时，您需要通过调用 muteRemoteAudio(userId, mute: false) 手动订阅该用户的音频数据，SDK 会在接收到该用户的音频数据后解码并播放。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知。此时，您需要通过调用 startRemoteView(userId, view: view) 方法手动订阅该用户的视频数据，SDK 会在接收到该用户的视频数据后解码并播放。

步骤7：发布本地的音视频流

1. 调用 startLocalAudio() 可以开启本地的麦克风采集，并将采集到的声音编码并发送出去。
2. 调用 startLocalPreview() 可以开启本地的摄像头，并将采集到的画面编码并发送出去。
3. 调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：
   - Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
   - Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。
4. 调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。

```
//示例代码：发布本地的音视频流
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

⚠ 注意：
Mac 版 SDK 默认会使用当前系统默认的摄像头和麦克风。您可以通过调用 setCurrentCameraDevice() 和 setCurrentMicDevice() 选择其他摄像头和麦克风。

步骤8：退出当前房间

调用 exitRoom() 方法退出房间，SDK 在退房时需要关闭和释放摄像头、麦克风等硬件设备，因此退房动作并非瞬间完成的，需收到 onExitRoom() 回调后才算真正完成退房操作。

```
// 调用退房后请等待 onExitRoom 事件回调
[self.trtcCloud exitRoom];

- (void)onExitRoom:(NSInteger)reason {
NSLog(@"离开房间: reason: %ld", reason)
}
```

⚠ 注意：
如果您的 App 中同时集成了多个音视频 SDK，请在收到 onExitRoom 回调后再启动其它音视频 SDK，否则可能会遇到硬件占用问题。

跑通通话模式(Android)

最近更新时间：2022-03-04 10:56:09

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

适用场景

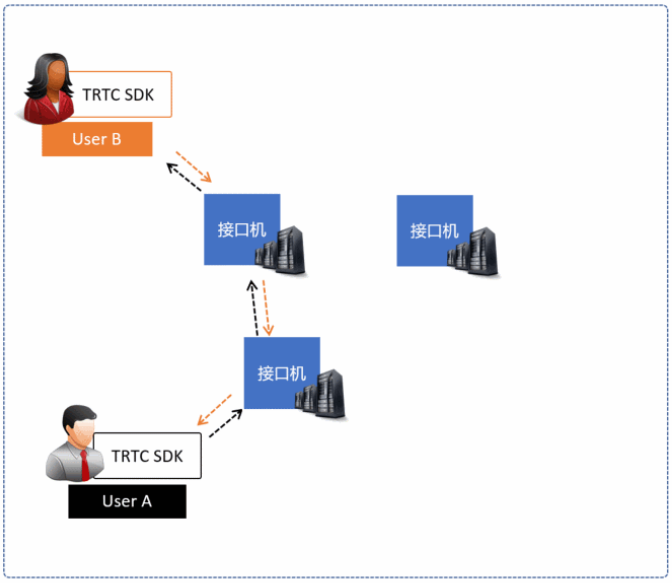
音视频通话TRTC 功能模块支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

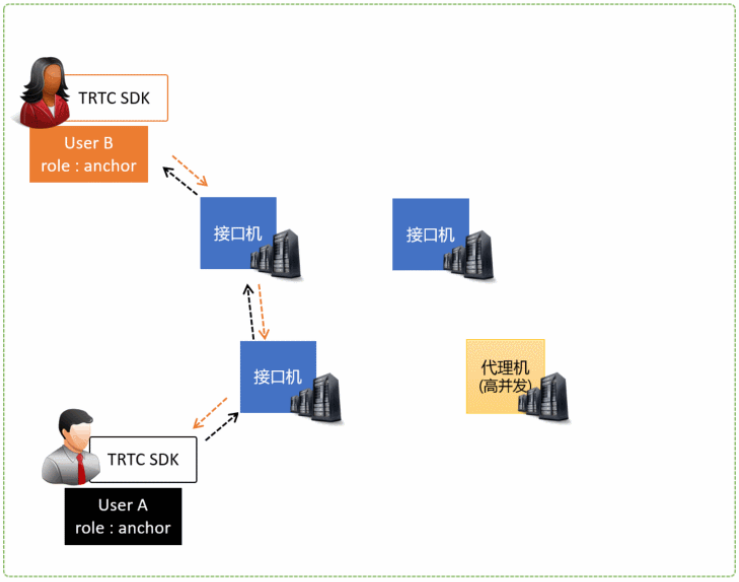
音视频通话TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **接口机**
  该类节点都采用最优质的线路和高性能的机器，善于处理端到端的低延时连麦通话。
- **代理机**
  该类节点都采用普通的线路和性能一般的机器，善于处理高并发的拉流观看需求。

在通话模式下，音视频通话TRTC 房间中的所有用户都会被分配到接口机上，相当于每个用户都是“主播”，每个用户随时都可以发言（最高的上行并发限制为50路），因此适合在线会议等场景，但单个房间的人数限制为300人。



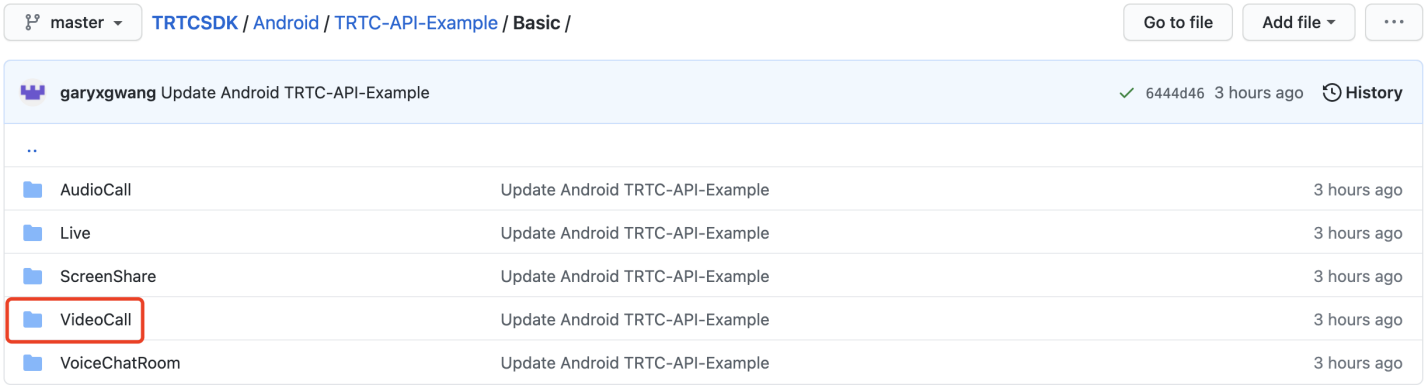
通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。



> 说明：
> 如果访问 Github 较慢，您也可以直接下载 TXLiteAVSDK_TRTC_Android_latest.zip。

操作步骤

步骤1：集成 SDK

您可以选择以下方式将**音视频通话 TRTC SDK** 集成到项目中。

方式一：自动加载（aar）

音视频通话 TRTC SDK 已发布到 jcenter 库，您可以通过配置 gradle 自动下载更新。
您只需用 Android Studio 打开待集成 SDK 的工程（TRTC-API-Example 已完成集成，示例代码可以供您参考），然后通过简单的步骤修改 app/build.gradle 文件，即可完成 SDK 集成：

1. 在 dependencies 中添加 TRTCSDK 的依赖。

```
dependencies {
compile 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release'
}
```

2. 在 defaultConfig 中，指定 App 使用的 CPU 架构。

> 说明：
> 目前 TRTC SDK 支持 armeabi， armeabi-v7a 和 arm64-v8a。

```
defaultConfig {
ndk {
abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
}
```

3. 单击**Sync Now**同步 SDK。
如果您的网络连接 jcenter 没有问题，SDK 会自动下载集成到工程中。

方式二：下载 ZIP 包手动集成

您可以直接下载 ZIP 压缩包，并参考 快速集成(Android) 将 SDK 集成到您的工程中。

步骤2：配置 App 权限

在 AndroidManifest.xml 文件中添加摄像头、麦克风以及网络的申请权限。

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />

<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

步骤3：初始化 SDK 实例并监听事件回调

1. 使用 sharedInstance() 接口创建 TRTCCloud 实例。

```
// 创建 trtcCloud 实例
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener(){
// 回调处理理
...
});
```

2. 设置 setListener 属性注册事件回调，并监听相关事件和错误通知。

```
// 错误通知监听，错误通知意味着 SDK 不能继续运行
@Override
public void onError(int errCode, String errMsg, Bundle extraInfo) {
Log.d(TAG, "sdk callback onError");
if (activity != null) {
Toast.makeText(activity, "onError: " + errMsg + "[" + errCode+ "]" , Toast.LENGTH_SHORT).show();
if (errCode == TXLiteAVCode.ERR_ROOM_ENTER_FAIL) {
activity.exitRoom();
}
}
}
```

步骤4：组装进房参数 TRTCParams

在调用 enterRoom() 接口时需要填写一个关键参数 TRTCParams，该参数包含的必填字段如下表所示。

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig。	eJyrVareCeYrSy1Ssll...
roomId	数字	默认不支持字符串类型的房间号，字符串类型的房间号会影响进房速度。如果您确实需要支持字符串类型的房间号，可以 提交工单 联系我们。	29834

> ⚠ 注意：
> TRTC 同一时间不支持两个相同的 userId 进入房间，否则会相互干扰。

步骤5：创建并进入房间

1. 调用 enterRoom() 即可加入 TRTCParams 参数中 roomId 代指的音视频房间。如果该房间不存在，SDK 会自动创建一个以字段 roomId 的值为房间号的新房间。

2. 请根据应用场景设置合适的 **appScene** 参数，使用错误可能会导致卡顿率或画面清晰度不达预期。
   - 视频通话，请设置为 TRTC_APP_SCENE_VIDEOCALL。
   - 语音通话，请设置为 TRTC_APP_SCENE_AUDIOCALL。
3. 进房成功后，SDK 会回调 onEnterRoom(result) 事件。其中，参数 result 大于0时表示进房成功，具体数值为加入房间所消耗的时间，单位为毫秒（ms）；当 result 小于0时表示进房失败，具体数值为进房失败的错误码。

```
public void enterRoom() {
TRTCCloudDef.TRTCParams trtcParams = new TRTCCloudDef.TRTCParams();
trtcParams.sdkAppId = sdkappid;
trtcParams.userId = userid;
trtcParams.roomId = 908;
trtcParams.userSig = usersig;
mTRTCCloud.enterRoom(trtcParams, TRTC_APP_SCENE_VIDEOCALL);
}

@Override
public void onEnterRoom(long result) {
if (result > 0) {
toastTip("进房成功，总计耗时[\(result)]ms")
} else {
toastTip("进房失败，错误码[\(result)]")
}
}
```

> ⚠ 注意：
>
> - 如果进房失败，SDK 同时还会回调 onError 事件，并返回参数 errCode（错误码）、errMsg（错误原因）以及 extraInfo（保留参数）。
> - 如果已在某一个房间中，则必须先调用 exitRoom()退出当前房间，才能进入下一个房间。
> - 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

步骤6：订阅远端的音视频流

SDK 支持自动订阅和手动订阅。

自动订阅模式（默认）

在自动订阅模式下，进入某个房间后，SDK 会自动接收房间中其他用户的音频流，从而达到最佳的“秒开”效果：

1. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知，SDK 会自动播放远端用户的声音。
2. 您可以通过 muteRemoteAudio(userId, true) 屏蔽某一个 userId 的音频数据，也可以通过 muteAllRemoteAudio(true) 屏蔽所有远端用户的音频数据，屏蔽后 SDK 不再继续拉取对应远端用户的音频数据。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知，但此时 SDK 未收到该如何展示视频数据的指令，因此不会自动处理视频数据。您需要通过调用 startRemoteView(userId, view) 方法将远端用户的视频数据和显示view关联起来。
4. 您可以通过 setRemoteViewFillMode() 指定视频画面的显示模式：
   - Fill 模式：表示填充，画面可能会等比放大和裁剪，但不会有黑边。
   - Fit 模式：表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。
5. 您可以通过 stopRemoteView(userId) 屏蔽某一个 userId 的视频数据，也可以通过 stopAllRemoteView() 屏蔽所有远端用户的视频数据，屏蔽后 SDK 不再继续拉取对应远端用户的视频数据。

```
@Override
public void onUserVideoAvailable(String userId, boolean available) {
TXCloudVideoView remoteView = remoteViewDic[userId];
if (available) {
mTRTCCloud.startRemoteView(userId, remoteView);
mTRTCCloud.setRemoteViewFillMode(userId, TRTC_VIDEO_RENDER_MODE_FIT);
} else {
mTRTCCloud.stopRemoteView(userId);
```

```
}
}
```

> 说明：
> 如果您在收到 onUserVideoAvailable() 事件回调后没有立即调用 startRemoteView() 订阅视频流，SDK 将在5s内停止接收来自远端的视频数据。

手动订阅模式

您可以通过 setDefaultStreamRecvMode() 接口将 SDK 指定为手动订阅模式。在手动订阅模式下，SDK 不会自动接收房间中其他用户的音视频数据，需要您手动通过 API 函数触发。

1. 在**进房前**调用 setDefaultStreamRecvMode(false, false) 接口将 SDK 设定为手动订阅模式。
2. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知。此时，您需要通过调用 muteRemoteAudio(userId, false) 手动订阅该用户的音频数据，SDK 会在接收到该用户的音频数据后解码并播放。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知。此时，您需要通过调用 startRemoteView(userId, remoteView) 方法手动订阅该用户的视频数据，SDK 会在接收到该用户的视频数据后解码并播放。

步骤7：发布本地的音视频流

1. 调用 startLocalAudio() 可以开启本地的麦克风采集，并将采集到的声音编码并发送出去。
2. 调用 startLocalPreview() 可以开启本地的摄像头，并将采集到的画面编码并发送出去。
3. 调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：
   - Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
   - Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。
4. 调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。

```
//示例代码：发布本地的音视频流
mTRTCCloud.setLocalViewFillMode(TRTC_VIDEO_RENDER_MODE_FIT);
mTRTCCloud.startLocalPreview(mIsFrontCamera, mLocalView);
mTRTCCloud.startLocalAudio();
```

步骤8：退出当前房间

调用 exitRoom() 方法退出房间，SDK 在退房时需要关闭和释放摄像头、麦克风等硬件设备，因此退房动作并非瞬间完成的，需收到 onExitRoom() 回调后才算真正完成退房操作。

```
// 调用退房后请等待 onExitRoom 事件回调
mTRTCCloud.exitRoom()

@Override
public void onExitRoom(int reason) {
Log.i(TAG, "onExitRoom: reason = " + reason);
}
```

> 注意：
> 如果您的 App 中同时集成了多个音视频 SDK，请在收到 onExitRoom 回调后再启动其它音视频 SDK，否则可能会遇到硬件占用问题。

跑通通话模式(Windows)

最近更新时间：2021-12-29 15:26:07

本文主要介绍如何基于音视频通话 TRTC SDK 实现一个简单的视频通话功能。本文仅罗列最常用的几个接口，如果您希望了解更多的接口函数，请参见 API 文档。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

示例代码

所属平台	示例代码
Windows（MFC）	TRTCMainViewController.cpp
Windows（Duilib）	TRTCMainViewController.cpp
Windows（C#）	TRTCMainForm.cs

视频通话

1. 初始化 SDK

使用音视频通话 TRTC SDK 的第一步，是先通过 getTRTCShareInstance 导出接口获取一个 TRTCCloud 单实例对象的指针 ITRTCCloud*，并注册监听 SDK 事件的回调。

- 继承 ITRTCCloudCallback 事件回调接口类，重写关键事件的回调接口，包括本地用户进房/退房事件、远端用户加入/退出事件、错误事件和警告事件等。
- 调用 addCallback 接口注册监听 SDK 事件。

> ⚠ 注意：
> 如果 addCallback 注册 N 次，同一个事件， SDK 就会触发 N 次回调，建议只调用一次 addCallback 。

C++

```
// TRTCMainViewController.h

// 继承 ITRTCCloudCallback 事件回调接口类
class TRTCMainViewController : public ITRTCCloudCallback
{
public:
TRTCMainViewController();
virtual ~TRTCMainViewController();

virtual void onError(TXLiteAVError errCode, const char* errMsg, void* arg);
virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void* arg);
virtual void onEnterRoom(int result);
virtual void onExitRoom(int reason);
virtual void onRemoteUserEnterRoom(const char* userId);
virtual void onRemoteUserLeaveRoom(const char* userId, int reason);
virtual void onUserVideoAvailable(const char* userId, bool available);
virtual void onUserAudioAvailable(const char* userId, bool available);
...
```

```
private:
ITRTCCloud * m_pTRTCSDK = NULL;
...
}

// TRTCMainViewController.cpp

TRTCMainViewController::TRTCMainViewController()
{
// 创建 TRTCCloud 实例
m_pTRTCSDK = getTRTCShareInstance();

// 注册 SDK 回调事件
m_pTRTCSDK->addCallback(this);
}

TRTCMainViewController::~TRTCMainViewController()
{
// 取消监听 SDK 事件
if(m_pTRTCSDK) {
m_pTRTCSDK->removeCallback(this);
}

// 释放 TRTCCloud 实例
if(m_pTRTCSDK != NULL) {
destroyTRTCShareInstance();
m_pTRTCSDK = null;
}
}

// 错误通知是需要监听的，错误通知意味着 SDK 无法继续运行
virtual void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
if (errCode == ERR_ROOM_ENTER_FAIL) {
LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
exitRoom();
}
}
```

C#

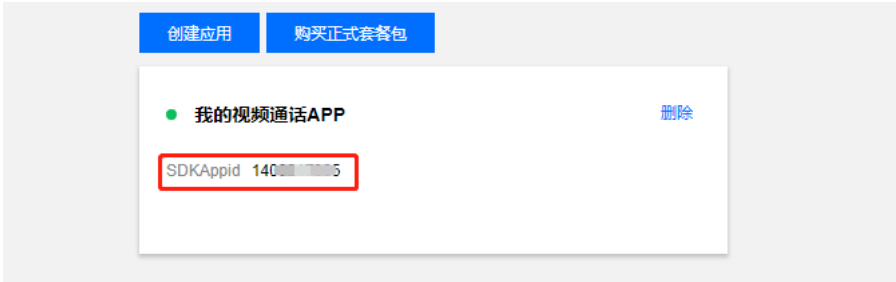
```
// TRTCMainForm.cs

// 继承 ITRTCCloudCallback 事件回调接口类
public partial class TRTCMainForm : Form, ITRTCCloudCallback, ITRTCLogCallback
{
...
private ITRTCCloud mTRTCCloud;
...

public TRTCMainForm(TRTCLoginForm loginForm)
{
InitializeComponent();
this.Disposed += new EventHandler(OnDisposed);
// 创建 TRTCCloud 实例
mTRTCCloud = ITRTCCloud.getTRTCShareInstance();
// 注册 SDK 回调事件
mTRTCCloud.addCallback(this);
...
```

```
}

private void OnDisposed(object sender, EventArgs e)
{
if (mTRTCCloud != null)
{
// 取消监听 SDK 事件
mTRTCCloud.removeCallback(this);
// 释放 TRTCCloud 实例
ITRTCCloud.destroyTRTCShareInstance();
mTRTCCloud = null;
}
...
}
...
// 错误通知是需要监听的，错误通知意味着 SDK 无法继续运行
public void onError(TXLiteAVError errCode, string errMsg, IntPtr arg)
{
if (errCode == TXLiteAVError.ERR_ROOM_ENTER_FAIL) {
exitRoom();
}
...
}
...
}
```

2. 组装 TRTCParams

TRTCParams 是 SDK 最关键的一个参数，它包含如下四个必填的字段：SDKAppID、userId、userSig 和 roomId。

- **SDKAppID**
  进入腾讯云实时音视频 控制台，如果您还没有应用，请创建一个，即可看到 SDKAppID。



- **userId**
  您可以随意指定，由于是字符串类型，可以直接跟您现有的账号体系保持一致，但请注意，**同一个音视频房间里不应该有两个同名的 userId**。
- **userSig**
  基于 SDKAppID 和 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig。
- **roomId**
  房间号是数字类型，您可以随意指定，但请注意，**同一个应用里的两个音视频房间不能分配同一个 roomId**。

3. 进入（或创建）房间

调用 enterRoom 可以加入 TRTCParams 参数中 roomId 所指定的音视频房间。如果该房间不存在，SDK 会自动创建一个以 roomId 为房间号的新房间。

appScene 参数指定 SDK 的应用场景，本文档中我们使用 TRTCAppSceneVideoCall（视频通话），该场景下 SDK 内部的编解码器和网络组件会更加侧重视频流畅性，降低通话延迟和卡顿率。

- 如进房成功，SDK 会回调 onEnterRoom 接口，参数：当 result 大于0时，进房成功，数值表示加入房间所消耗的时间，单位为毫秒（ms）；当 result 小于0时，进房失败，数值表示进房失败的错误码。
- 如进房失败，SDK 同时会回调 onError 接口，参数：errCode（错误码 ERR_ROOM_ENTER_FAIL，错误码可参考 TXLiteAVCode.h）、errMsg（错误原因）、extraInfo（保留参数）。

- 如果已在房间中，则必须调用 exitRoom 方法退出当前房间，才能进入下一个房间。

C++

```
// TRTCMainViewController.cpp

void TRTCMainViewController::enterRoom()
{
// TRTCParams 定义参考头文件 TRTCCloudDef.h
TRTCParams params;
params.sdkAppId = sdkappid;
params.userId = userid;
params.userSig = usersig;
params.roomId = 908; // 输入您想进入的房间
if(m_pTRTCSDK)
{
m_pTRTCSDK->enterRoom(params, TRTCAppSceneVideoCall);
}
}

...

void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
if(errCode == ERR_ROOM_ENTER_FAIL)
{
LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
// 检查 userSig 是否合法、网络是否正常等
}
}

...

void TRTCMainViewController::onEnterRoom(int result)
{
LOGI(L"onEnterRoom result[%d]", result);
if(result >= 0)
{
//进房成功
}
else
{
//进房失败，错误码 = result;
}
}
```

C#

```
// TRTCMainForm.cs

public void EnterRoom()
{
// TRTCParams 定义参考头文件 TRTCCloudDef.h
TRTCParams @params = new TRTCParams();
@params.sdkAppId = sdkappid;
@params.userId = userid;
@params.userSig = usersig;
@params.roomId = 908; // 输入您想进入的房间
if(mTRTCCloud != null)
```

```
{
mTRTCCloud.enterRoom(@params, TRTCAppSceneVideoCall);
}
}

...

public void onError(TXLiteAVError errCode, string errMsg, IntPtr arg)
{
if(errCode == TXLiteAVError.ERR_ROOM_ENTER_FAIL)
{
Log.E(String.Format("errCode : {0}, errMsg : {1}, arg = {2}", errCode, errMsg, arg));
// 检查 userSig 是否合法、网络是否正常等
}
}

...

public void onEnterRoom(int result)
{
if(result >= 0)
{
//进房成功
}
else
{
//进房失败，错误码 = result;
}
}
```

⚠ 注意：

- 请根据应用场景选择合适的 scene 参数，使用错误可能会导致卡顿率或画面清晰度不达预期。
- 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

4. 收听远端音频流

音视频通话TRTC SDK 会默认接收远端的音频流，您无需为此编写额外的代码。如果您不希望收听某一个 userid 的音频流，可以使用 muteRemoteAudio 将其静音。

5. 观看远端视频流

音视频通话TRTC SDK 并不会默认拉取远端的视频流，当房间里有用户上行视频数据时，房间里的其他用户可以通过 ITRTCCloudCallback 中的 onUserVideoAvailable 回调获知该用户的 userid。之后，即可调用 startRemoteView 方法来显示该用户的视频画面。

通过 setRemoteViewFillMode 可以指定视频显示模式为 Fill 或 Fit 模式。两种模式下视频尺寸都是等比缩放，区别在于：

- Fill 模式：优先保证视窗被填满。如果缩放后的视频尺寸与显示视窗尺寸不一致，多出的视频将被截掉。
- Fit 模式：优先保证视频内容全部显示。如果缩放后的视频尺寸与显示视窗尺寸不一致，未被填满的视窗区域将使用黑色填充。

C++

```
// TRTCMainViewController.cpp
void TRTCMainViewController::onUserVideoAvailable(const char* userId, bool available){
if (available) {
// 获取渲染窗口的句柄。
CWnd *pRemoteVideoView = GetDlgItem(IDC_REMOTE_VIDEO_VIEW);
HWND hwnd = pRemoteVideoView->GetSafeHwnd();

// 设置远端用户视频的渲染模式。
m_pTRTCSDK->setRemoteViewFillMode(TRTCVideoFillMode_Fill);
// 调用 SDK 接口播放远端用户流。
```

```
m_pTRTCSDK->startRemoteView(userId， hwnd);
} else {
m_pTRTCSDK->stopRemoteView(userId);
}
}
```

C#

```
// TRTCMainForm.cs
public void onUserVideoAvailable(string userId, bool available)
{
if (available)
{
// 获取窗口句柄
IntPtr ptr = GetHandleAndSetUserId(pos, userId, false);
SetVisableInfoView(pos, false);
// 设置远端用户视频的渲染模式。
mTRTCCloud.setRemoteViewFillMode(userId, TRTCVideoFillMode.TRTCVideoFillMode_Fit);
// 调用 SDK 接口播放远端用户流。
mTRTCCloud.startRemoteView(userId, ptr);
}
else
{
mTRTCCloud.stopRemoteView(userId);
...
}
}
```

6. 开关本地声音采集

音视频通话TRTC SDK 并不会默认打开本地的麦克风采集， startLocalAudio 可以开启本地的声音采集并将音视频数据广播出去， stopLocalAudio 则会关闭。

> **说明：**
> 您可以在 startLocalPreview 之后继续调用 startLocalAudio 。

7. 开关本地视频采集

音视频通话TRTC SDK 并不会默认打开本地的摄像头采集， startLocalPreview 可以开启本地的摄像头并显示预览画面， stopLocalPreview 则会关闭。

- 调用 startLocalPreview，指定本地视频渲染的窗口，**SDK 动态检测窗口大小，在 rendHwnd 表示的整个窗口进行渲染**。
- 调用 setLocalViewFillMode 接口，设置本地视频渲染的模式为 Fill 或者 Fit 。两种模式下视频尺寸都是等比缩放，区别在于：
  - Fill 模式：优先保证窗口被填满。如果缩放后的视频尺寸与窗口尺寸不一致，那么多出的部分将被裁剪掉。
  - Fit 模式：优先保证视频内容全部显示。如果缩放后的视频尺寸与窗口尺寸不一致，未被填满的窗口区域将使用黑色填充。

C++

```
// TRTCMainViewController.cpp

void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
...

// 获取渲染窗口的句柄。
CWnd *pLocalVideoView = GetDlgItem(IDC_LOCAL_VIDEO_VIEW);
HWND hwnd = pLocalVideoView->GetSafeHwnd();

if(m_pTRTCSDK)
{
// 调用 SDK 接口设置渲染模式和渲染窗口。
m_pTRTCSDK->setLocalViewFillMode(TRTCVideoFillMode_Fit);
```

```
m_pTRTCSDK->startLocalPreview(hwnd);
}

...
}
```

C#

```
// TRTCMainForm.cs

public void onEnterRoom(int result)
{
...
// 获取渲染窗口的句柄。
IntPtr ptr = GetHandle();
if (mTRTCCloud != null)
{
// 调用 SDK 接口设置渲染模式和渲染窗口。
mTRTCCloud.setLocalViewFillMode(TRTCVideoFillMode_Fit);
mTRTCCloud.startLocalPreview(ptr);
}
...
}
```

8. 屏蔽音视频数据流

- **屏蔽本地视频数据**
  如果用户在通话过程中，出于隐私目的希望屏蔽本地的视频数据，让房间里的其他用户暂时无法看到您的画面，可以调用 muteLocalVideo 。
- **屏蔽本地音频数据**
  如果用户在通话过程中，出于隐私目的希望屏蔽本地的音频数据，让房间里的其他用户暂时无法听到您的声音，可以调用 muteLocalAudio 。
- **屏蔽远程视频数据**
  通过 stopRemoteView 可以屏蔽某一个 userid 的视频数据。
  通过 stopAllRemoteView 可以屏蔽所有远端用户的视频数据。
- **屏蔽远程音频数据**
  通过 muteRemoteAudio 可以屏蔽某一个 userid 的音频数据。
  通过 muteAllRemoteAudio 可以屏蔽所有远端用户的音频数据。

9. 退出房间

调用 exitRoom 方法退出房间。不论当前是否还在通话中，调用该方法会把视频通话相关的所有资源释放掉。

> **说明：**
> 在您调用 exitRoom 之后，SDK 会进入一个复杂的退房握手流程，当 SDK 回调 onExitRoom 方法时才算真正完成资源的释放。

C++

```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
if(m_pTRTCSDK)
{
m_pTRTCSDK->exitRoom();
}
}
....
void TRTCMainViewController::onExitRoom(int reason)
```

```
{
// 退房成功，reason 参数保留，暂未使用。

...
}
```

C#

```
// TRTCMainForm.cs

public void OnExit()
{
if(mTRTCCloud != null)
{
mTRTCCloud.exitRoom();
}
}
...
public void onExitRoom(int reason)
{
// 退房成功
...
}
```

跑通通话模式(Electron)

最近更新时间：2022-03-04 10:56:25

适用场景

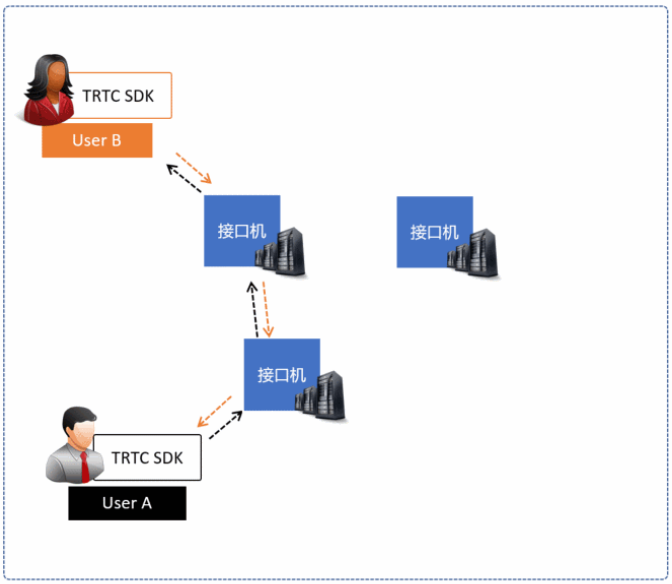
腾讯云视立方音视频通话 TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

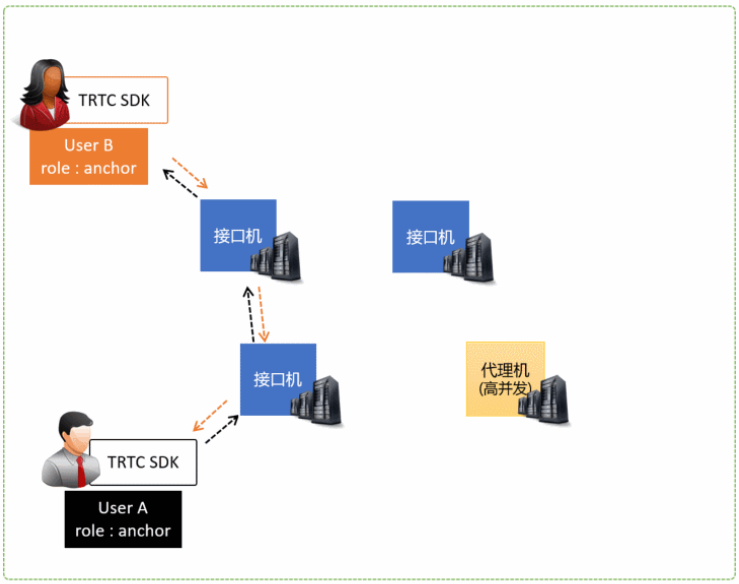
腾讯云视立方音视频通话 TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **接口机**
  该类节点都采用最优质的线路和高性能的机器，善于处理端到端的低延时连麦通话。
- **代理机**
  该类节点都采用普通的线路和性能一般的机器，善于处理高并发的拉流观看需求。

在通话模式下，音视频通话 TRTC 房间中的所有用户都会被分配到接口机上，相当于每个用户都是“主播”，每个用户随时都可以发言（最高的上行并发限制为50路），因此适合在线会议等场景，但单个房间的人数限制为300人。



通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。

操作步骤

步骤1：尝试跑通官网 SimpleDemo

建议您先阅读文档 跑通 SimpleDemo(Electron)，并按照文档的指引，跑通我们为您提供的官方 SimpleDemo。

- 如果 SimpleDemo 能顺利运行，说明您已经掌握了在项目中安装 Electron 的方法。
- 反之，如果运行 SimpleDemo 遇到问题，您大概率遭遇了 Electron 的下载、安装问题，此时您可以参考我们总结的 Electron 常见问题收录，也可以参考 Electron 官方的 安装指引。

步骤2：为您的项目集成 trtc-electron-sdk

如果 步骤1 正常执行并且效果符合预期，说明您已经掌握了 Electron 环境的安装方法。

- 您可以在我们的官方 Demo 的基础上进行二次开发，项目的起步阶段会比较顺利。

- 您也可以执行以下指令，把 trtc-electron-sdk 安装到您现有的项目中：

```
npm install trtc-electron-sdk --save
```

步骤3：初始化 SDK 实例并监听事件回调

1. 创建 trtc-electron-sdk 实例：

```
import TRTCCloud from 'trtc-electron-sdk';
let trtcCloud = new TRTCCloud();
```

2. 监听 onError 事件：

```
// 错误通知是要监听的，需要捕获并通知用户
let onError = function(err) {
console.error(err);
};
trtcCloud.on('onError',onError);
```

步骤4： 组装进房参数 TRTCParams

在调用 enterRoom() 接口时需要填写一个关键参数 TRTCParams，该参数包含的必填字段如下表所示。

参数	类型	说明	示例
sdkAppId	数字	应用 ID，您可以在 控制台 >应用管理 > 应用信息中查找到。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1SsII...
roomId	数字	默认不支持字符串类型的房间号，字符串类型的房间号会影响进房速度。如果您确实需要支持字符串类型的房间号，可以 提交工单 联系我们。	29834

```
import {
TRTCParams,
TRTCRoleType
} from "trtc-electron-sdk/liteav/trtc_define";

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SsII...';
```

> 注意：
>
> - TRTC 同一时间不支持两个相同的 userId 进入房间，否则会相互干扰。
> - 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

步骤5：创建并进入房间

1. 调用 enterRoom() 即可加入 TRTCParams 参数中 roomId 代指的音视频房间。如果该房间不存在，SDK 会自动创建一个以字段 roomId 的值为房间号的新房间。
2. 请根据应用场景设置合适的 appScene 参数，使用错误可能会导致卡顿率或画面清晰度不达预期。
   - 视频通话，请设置为 TRTCAppScene.TRTCAppSceneVideoCall。
   - 语音通话，请设置为 TRTCAppScene.TRTCAppSceneAudioCall。

> 说明：

> 关于 TRTCAppScene 的详细介绍，请参见 TRTCAppScene。

3. 进房成功后，SDK 会回调 onEnterRoom(result) 事件。其中，参数 result 大于0时表示进房成功，具体数值为加入房间所消耗的时间，单位为毫秒（ms）；当 result 小于0时表示进房失败，具体数值为进房失败的错误码。

```
import TRTCCloud from 'trtc-electron-sdk';
import { TRTCParams, TRTCAppScene } from "trtc-electron-sdk/liteav/trtc_define";
import TRTCCloud from 'trtc-electron-sdk';
let trtcCloud = new TRTCCloud();

let onEnterRoom = function (result) {
if (result > 0) {
console.log(`onEnterRoom，进房成功，使用了 ${result} 秒`);
} else {
console.warn(`onEnterRoom: 进房失败 ${result}`);
}
};

// 订阅进房成功事件
trtcCloud.on('onEnterRoom', onEnterRoom);

// 进房，如果房间不存在，TRTC 后台会自动创建一个新房间
let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SsII...';
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneVideoCall);
```

步骤6: 订阅远端的音视频流

SDK 支持自动订阅和手动订阅两种模式，自动订阅追求秒开速度，适合于人数少的通话场景；手动订阅追求流量节约，适合人数较多的会议场景。

自动订阅（推荐）

进入某个房间之后，SDK 会自动接收房间中其他用户的音频流，从而达到最佳的“秒开”效果：

1. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知，SDK 会自动播放这些远端用户的声音。
2. 您可以通过 muteRemoteAudio(userId, true) 屏蔽某一个 userId 的音频数据，也可以通过 muteAllRemoteAudio(true) 屏蔽所有远端用户的音频数据，屏蔽后 SDK 不再继续拉取对应远端用户的音频数据。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知，但此时 SDK 未收到该如何展示视频数据的指令，因此不会自动处理视频数据。您需要通过调用 startRemoteView(userId, view) 方法将远端用户的视频数据和显示 view 关联起来。
4. 您可以通过 setLocalViewFillMode() 指定视频画面的显示模式：
   - TRTCVideoFillMode.TRTCVideoFillMode_Fill 模式：表示填充，画面可能会等比放大和裁剪，但不会有黑边。
   - TRTCVideoFillMode.TRTCVideoFillMode_Fit 模式：表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。
5. 您可以通过 stopRemoteView(userId) 可以屏蔽某一个 userId 的视频数据，也可以通过 stopAllRemoteView() 屏蔽所有远端用户的视频数据，屏蔽后 SDK 不再继续拉取对应远端用户的视频数据。

```
<div id="video-container"></div>

<script>
import TRTCCloud from 'trtc-electron-sdk';
const trtcCloud = new TRTCCloud();
const videoContainer = document.querySelector('#video-container');
const roomId = 29834;

/**
* 用户是否开启摄像头视频
* @param {number} uid - 用户标识
* @param {boolean} available - 画面是否开启
```

```
**/

let onUserVideoAvailable = function (uid, available) {

console.log(`onUserVideoAvailable: uid: ${uid}, available: ${available}`);
if (available === 1) {
let id = `${uid}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (!view) {
view = document.createElement('div');
view.id = id;
videoContainer.appendChild(view);
}
trtcCloud.startRemoteView(uid, view);
trtcCloud.setRemoteViewFillMode(uid, TRTCVideoFillMode.TRTCVideoFillMode_Fill);
} else {
let id = `${uid}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (view) {
videoContainer.removeChild(view);
}
}
};

// 实例代码：根据通知订阅（或取消订阅）远端用户的视频画面
trtcCloud.on('onUserVideoAvailable', onUserVideoAvailable);

</script>
```

> **说明：**
> 如果您在收到 onUserVideoAvailable() 事件回调后没有立即调用 startRemoteView() 订阅视频流，SDK 将会在5s内停止接收来自远端的视频数据。

手动订阅

您可以通过 setDefaultStreamRecvMode(autoRecvAudio, autoRecvVideo) 接口将 SDK 指定为手动订阅模式。在手动订阅模式下，SDK 不会自动接收房间中其他用户的音视频数据，需要您手动通过 API 函数触发。

1. 在**进房前**调用 setDefaultStreamRecvMode(false, false) 接口将 SDK 设定为手动订阅模式。
2. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知。此时，您需要通过调用 muteRemoteAudio(userId, false) 手动订阅该用户的音频数据，SDK 会在接收到该用户的音频数据后解码并播放。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable(userId, available) 事件通知。此时，您需要通过调用 startRemoteView(userId, view) 方法手动订阅该用户的视频数据，SDK 会在接收到该用户的视频数据后解码并播放。

步骤7：发布本地的音视频流

1. 调用 startLocalAudio() 可以开启本地的麦克风采集，并将采集到的声音编码并发送出去。
2. 调用 startLocalPreview() 可以开启本地的摄像头，并将采集到的画面编码并发送出去。
3. 调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：
   - TRTCVideoFillMode.TRTCVideoFillMode_Fill：模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
   - TRTCVideoFillMode.TRTCVideoFillMode_Fit：模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。
4. 调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。

```
//示例代码：发布本地的音视频流
trtcCloud.startLocalPreview(view);
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);
trtcCloud.startLocalAudio();
//设置本地视频编码参数
let encParam = new TRTCVideoEncParam();
encParam.videoResolution = TRTCVideoResolution.TRTCVideoResolution_640_360;
```

```
encParam.resMode = TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape;
encParam.videoFps = 25;
encParam.videoBitrate = 600;
encParam.enableAdjustRes = true;
trtcCloud.setVideoEncoderParam(encParam);
```

⚠ 注意：
SDK 默认会使用当前系统默认的摄像头和麦克风。您可以通过调用 setCurrentCameraDevice() 和 setCurrentMicDevice() 选择其他摄像头和麦克风。

步骤8：退出当前房间

调用 exitRoom() 方法退出房间，SDK 在退房时需要关闭和释放摄像头、麦克风等硬件设备，因此退房动作并非瞬间完成的，需收到 onExitRoom() 回调后才算真正完成退房操作。

```
// 调用退房后请等待 onExitRoom 事件回调
let onExitRoom = function (reason) {
console.log(`onExitRoom, reason: ${reason}`);
};
trtcCloud.exitRoom();
trtcCloud.on('onExitRoom', onExitRoom);
```

⚠ 注意：
如果您的 Electron 程序中同时集成了多个音视频 SDK，请在收到 onExitRoom 回调后再启动其它音视频 SDK，否则可能会遇到硬件占用问题。

跑通通话模式(Web)

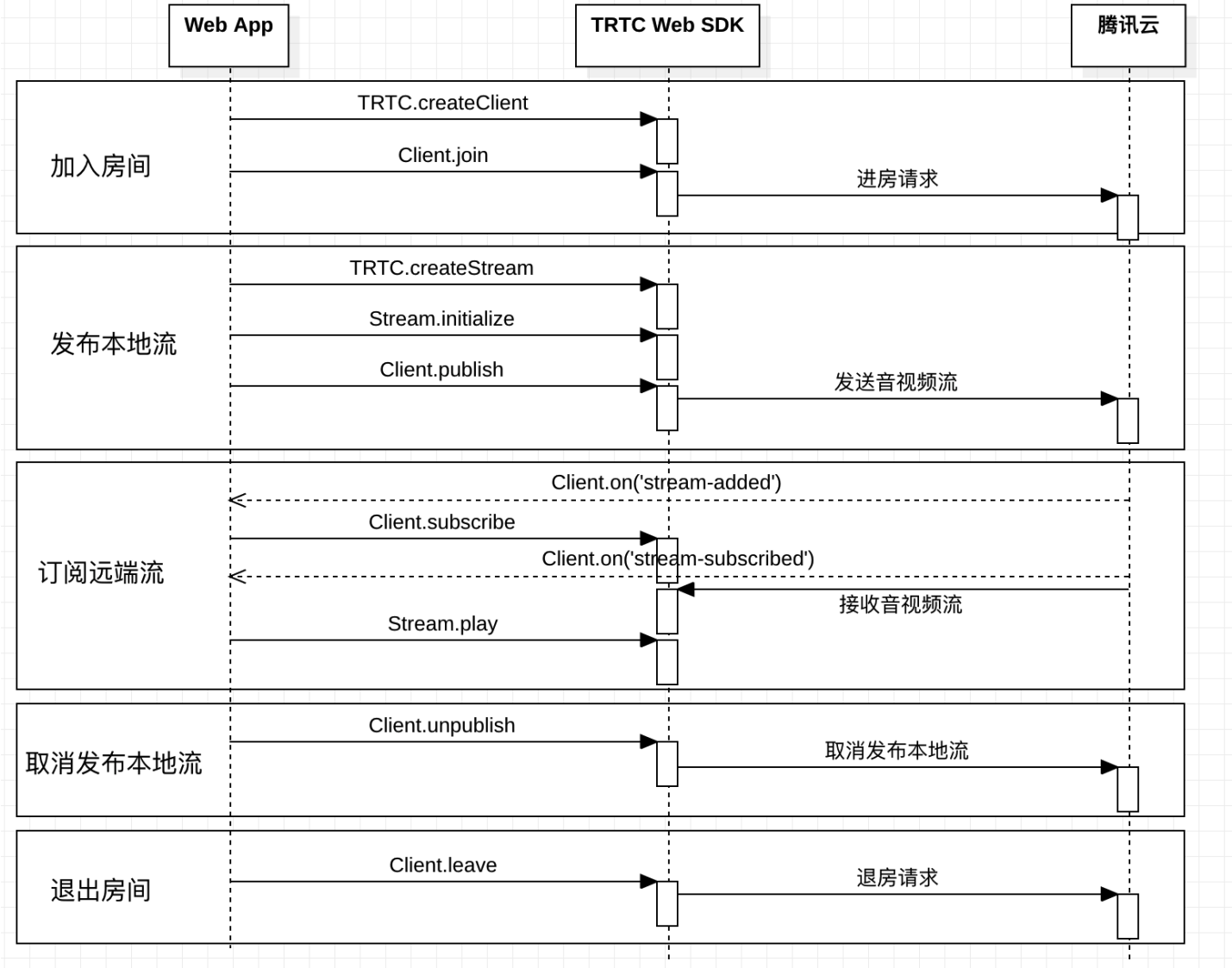
最近更新时间：2021-12-29 15:26:19

本文主要介绍腾讯云视立方音视频通话 TRTC Web SDK 的基本工作流程以及如何实现一个实时音视频通话功能。

在使用音视频通话 TRTC Web SDK 中，经常会接触到以下对象：

- Client 对象，代表一个本地客户端。Client 类的方法提供了加入通话房间、发布本地流、订阅远端流等功能。
- Stream 对象，代表一个音视频流对象，包括本地音视频流对象 LocalStream 和远端音视频流对象 RemoteStream。Stream 类的方法主要提供音视频流对象的行为，包括音频和视频的播放控制。

基本音视频通话的 API 调用流程如下图所示：



步骤1：创建 Client 对象

通过 TRTC.createClient() 方法创建 Client 对象，参数设置如下：

- mode：实时音视频通话模式，设置为rtc
- sdkAppId：您从腾讯云申请的 sdkAppId
- userId：用户 ID
- userSig：用户签名，计算方式请参见 如何计算及使用 UserSig

```
const client = TRTC.createClient({
mode: 'rtc',
```

```
sdkAppId,
userId,
userSig
});
```

步骤2：进入音视频通话房间

调用 Client.join() 进入音视频通话房间。
参数 roomId ： 房间 ID

```
client
.join({ roomId })
.then(() => {
console.log('进房成功');
})
.catch(error => {
console.error('进房失败 ' + error);
});
```

步骤3：发布本地流和订阅远端流

1. 使用 TRTC.createStream() 方法创建本地音视频流。
   以下实例从摄像头及麦克风中采集音视频流，参数设置如下：
   - userId：本地流所属的用户 ID
   - audio：是否开启音频
   - video：是否开启视频

```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
```

2. 调用 LocalStream.initialize() 初始化本地音视频流。

```
localStream
.initialize()
.then(() => {
console.log('初始化本地流成功');
})
.catch(error => {
console.error('初始化本地流失败 ' + error);
});
```

3. 在本地流初始化成功后，调用 Client.publish() 方法发布本地流。

```
client
.publish(localStream)
.then(() => {
console.log('本地流发布成功');
})
.catch(error => {
console.error('本地流发布失败 ' + error);
});
```

4. 远端流通过监听事件Client.on('stream-added')获取，收到该事件后，通过 Client.subscribe() 订阅远端音视频流。

> 说明：

- 请在 Client.join() 进房前注册 Client.on('stream-added') 事件以确保您不会错过远端用户进房通知。
- 远端流离开等其他事件可以在 API 详细文档 中查看。

```
client.on('stream-added', event => {
const remoteStream = event.stream;
console.log('远端流增加: ' + remoteStream.getId());
//订阅远端流
client.subscribe(remoteStream);
});
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('远端流订阅成功：' + remoteStream.getId());
// 播放远端流
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

5. 在本地流初始化成功的回调中，或远端流订阅成功事件回调中，通过调用 Stream.play() 方法在网页中播放音视频。play 方法接受一个 div 元素 ID 作为参数，SDK 内部会在该 div 元素下自动创建相应的音视频标签并在其上播放音视频。
   - 初始化本地流成功时播放本地流

```
localStream
.initialize()
.then(() => {
console.log('初始化本地流成功');
localStream.play('local_stream');
})
.catch(error => {
console.error('初始化本地流失败 ' + error);
});
```

   - 订阅远端流成功时播放远端流

```
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('远端流订阅成功：' + remoteStream.getId());
// 播放远端流
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

步骤4：退出音视频通话房间

通话结束时调用 Client.leave() 方法退出音视频通话房间，整个音视频通话会话结束。

```
client
.leave()
.then(() => {
// 退房成功，可再次调用client.join重新进房开启新的通话。
})
.catch(error => {
console.error('退房失败 ' + error);
// 错误不可恢复，需要刷新页面。
});
```

⚠ 注意：

每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

跑通通话模式(小程序)

最近更新时间：2021-12-29 15:26:27

本文主要介绍腾讯云视立方音视频通话 TRTC SDK 的几个最基本功能的使用方法，阅读此文档有助于您对 TRTC 的基本使用流程有一个简单的认识。

环境要求

- 微信 App iOS 最低版本要求：7.0.9。
- 微信 App Android 最低版本要求：7.0.8。
- 小程序基础库最低版本要求：2.10.0。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在真机上进行运行体验。
- 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，需要申请常规小程序账号进行开发。
- 不支持 uniapp 开发环境，请使用原生小程序开发环境。

前提条件

1. 您已 注册腾讯云 账号，并完成 实名认证。
2. **开通小程序类目与推拉流标签权限（如不开通则无法正常使用）。**
   出于政策和合规的考虑，微信暂未放开所有小程序对实时音视频功能（即 <live-pusher> 和 <live-player> 标签）的支持：
   - 小程序推拉流标签不支持个人小程序，只支持企业类小程序。
   - 小程序推拉流标签使用权限暂时只开放给有限 类目。
   - 符合类目要求的小程序，需要在 **微信公众平台 > 开发 > 开发管理 > 接口设置**中自助开通该组件权限，如下图所示：

准备工作

在使用基本功能前，请确保您已完成以下步骤，详情请参见 跑通Demo(小程序)，快速集成(小程序)。

- 创建了腾讯云实时音视频应用，购买了相应的套餐，并获取到 SDKAppID 和密钥信息。
- 开通小程序类目与推拉流标签权限。
- 小程序服务器域名配置。
- 在您的小程序项目中集成 <trtc-calling> 组件。

部署签名服务

在初始化组件时需要签名服务进行签发 UserSig，详情请参见 如何计算及使用 UserSig。

组装参数

参数准备

要使用 <trtc-calling> 组件，必须准备好如下参数：

- **SDKAppID**
  进入腾讯云实时音视频 控制台 创建一个新的应用，获得 SDKAppID。


- **UserID**
  UserID为字符串类型，您可以自定义指定。例如，直接与您现有的账号体系保持一致，或直接使用小程序的 openid。

> ⚠ 注意：
> **同一个音视频房间里不能存在两个相同的 userID。**

- **UserSig**
  基于 SDKAppID 和 UserID 可以计算出 UserSig，计算方法请参见 如何计算及使用 UserSig。

TRTCCalling 属性

属性	类型	默认值	必填	说明
id	String	-	是	绑定 TRTCCalling 的 DOM ID，可通过 this.selectComponent(ID) 获取实例
config	Object	-	是	TRTCCalling 初始化配置
backgroundMute	Boolean	false	否	进入后台时是否保持音频通话，true 保持、false 挂断

config 参数

参数	类型	必填	说明
sdkAppID	Number	是	开通实时音视频服务创建应用后分配的 SDKAppID
userID	String	是	用户 ID，可以由您的帐号体系指定
userSig	String	是	身份签名（即相当于登录密码），由 userID 计算得出，具体计算方法请参见 如何计算及使用 UserSig
type	Number	是	指定通话类型。1：语音通话，2：视频通话

集成组件

1. 在需要引入组件的页面目录下，配置相应页面的 xxx.json 文件。

```
// index.json
"usingComponents": {
"TRTCCalling": "../../components/TRTCCalling/TRTCCalling",
}
```

2. 在相应页面的 xxx.wxml 文件中使用标签。

```
// index.wxml
<TRTCCalling
id="TRTCCalling-component"
config="{{trtcConfig}}"
backgroundMute="{{true}}"
></TRTCCalling>
```

登录

设置 <trtc-calling> 组件的属性后，获取 <trtc-calling> 组件的对象实例，并调用对象实例的 login() 方法即可完成初始化。

```
Page({
// ...
data: {
trtcConfig: {
sdkAppID: 0,
userID: 0,
userSig: "",
type: 1
},
},
/**
* 生命周期函数--监听页面加载
*/
onLoad: function (options) {
// 将初始化后到TRTCCalling实例注册到this.TRTCCalling中，this.TRTCCalling 可使用TRTCCalling所以方法功能。
this.TRTCCalling = this.selectComponent('#TRTCCalling-component');
// 绑定需要监听到事件
this.bindTRTCCallingRoomEvent();
// 登录TRTCCalling
this.TRTCCalling.login();
},
/**
* 生命周期函数--监听页面卸载
*/
onUnload: function () {
// 取消监听事件
this.unbindTRTCCallingRoomEvent();
// 退出登录
this.TRTCCalling.logout();
},
invitedEvent() {
console.log('收到邀请')
},

hangupEvent() {
console.log('挂断')
},

rejectEvent() {
console.log('对方拒绝')
},

userLeaveEvent() {
console.log('用户离开房间')
},

onRespEvent() {
console.log('对方无应答')
},

callingTimeoutEvent() {
console.log('无应答超时')
},
```

```
lineBusyEvent() {
console.log('对方忙线')
},

callingCancelEvent() {
console.log('取消通话')
},

userEnterEvent() {
console.log('用户进入房间')
},

callEndEvent() {
console.log('通话结束')
},

// 绑定 TRTCCalling
bindTRTCCallingRoomEvent: function() {
const TRTCCallingEvent = this.TRTCCalling.EVENT
this.TRTCCalling.on(TRTCCallingEvent.INVITED, this.invitedEvent)
// 处理挂断的事件回调
this.TRTCCalling.on(TRTCCallingEvent.HANG_UP, this.hangupEvent)
this.TRTCCalling.on(TRTCCallingEvent.REJECT, this.rejectEvent)
this.TRTCCalling.on(TRTCCallingEvent.USER_LEAVE, this.userLeaveEvent)
this.TRTCCalling.on(TRTCCallingEvent.NO_RESP, this.onRespEvent)
this.TRTCCalling.on(TRTCCallingEvent.CALLING_TIMEOUT, this.callingTimeoutEvent)
this.TRTCCalling.on(TRTCCallingEvent.LINE_BUSY, this.lineBusyEvent)
this.TRTCCalling.on(TRTCCallingEvent.CALLING_CANCEL, this.callingCancelEvent)
this.TRTCCalling.on(TRTCCallingEvent.USER_ENTER, this.userEnterEvent)
this.TRTCCalling.on(TRTCCallingEvent.CALL_END, this.callEndEvent)
},
// 取消 TRTCCalling 监听事件
unbindTRTCCallingRoomEvent() {
const TRTCCallingEvent = this.TRTCCalling.EVENT
this.TRTCCalling.off(TRTCCallingEvent.INVITED, this.invitedEvent)
this.TRTCCalling.off(TRTCCallingEvent.HANG_UP, this.hangupEvent)
this.TRTCCalling.off(TRTCCallingEvent.REJECT, this.rejectEvent)
this.TRTCCalling.off(TRTCCallingEvent.USER_LEAVE, this.userLeaveEvent)
this.TRTCCalling.off(TRTCCallingEvent.NO_RESP, this.onRespEvent)
this.TRTCCalling.off(TRTCCallingEvent.CALLING_TIMEOUT, this.callingTimeoutEvent)
this.TRTCCalling.off(TRTCCallingEvent.LINE_BUSY, this.lineBusyEvent)
this.TRTCCalling.off(TRTCCallingEvent.CALLING_CANCEL, this.callingCancelEvent)
this.TRTCCalling.off(TRTCCallingEvent.USER_ENTER, this.userEnterEvent)
this.TRTCCalling.off(TRTCCallingEvent.CALL_END, this.callEndEvent)
},
// ...
})
```

进行通话

通过组件实例的 API call() 即可开始通话。type：1—语音通话，2—视频通话。

```
Page({
// ...
call: function() {
this.TRTCCalling.call({ userID: this.data.userId, type:2})
},
```

```
// ...
})
```

进阶功能
实现 CDN 直播观看

最近更新时间：2021-12-23 17:05:39

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	✓	–	–	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

适用场景

CDN 直播观看，也叫 “CDN 旁路直播”，由于 TRTC 采用 UDP 协议进行传输音视频数据，而标准直播 CDN 则采用的 RTMP\HLS\FLV 等协议进行数据传输，所以需要将 TRTC 中的音视频数据**旁路**到直播 CDN 中，才能在让观众通过直播 CDN 进行观看。

给 TRTC 对接 CDN 观看，一般被用于解决如下两类问题：

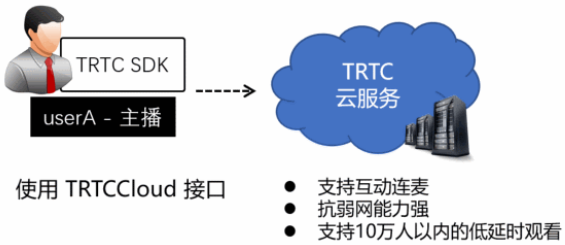
- **问题一：超高并发观看**
  TRTC 的低延时观看能力，单房间支持的最大人数上限为10万人。CDN 观看虽然延迟要高一些，但支持10万人以上的并发观看，且 CDN 的计费价格更加便宜。
- **问题二：移动端网页播放**
  TRTC 虽然支持 WebRTC 协议接入，但主要用于 Chrome 桌面版浏览器，移动端浏览器的兼容性非常不理想，尤其是 Android 手机浏览器对 WebRTC 的支持普遍都很差。所以如果希望通过 Web 页面在移动端分享直播内容，还是推荐使用 HLS(m3u8) 播放协议，这也就需要借助直播 CDN 的能力来支持 HLS 协议。

原理解析

腾讯云会使用一批旁路转码集群，将音视频通话 TRTC 中的音视频数据旁路到直播 CDN 系统中，该集群负责将音视频通话 TRTC 所使用的 UDP 协议转换为标准的直播 RTMP 协议。

单路画面的旁路直播

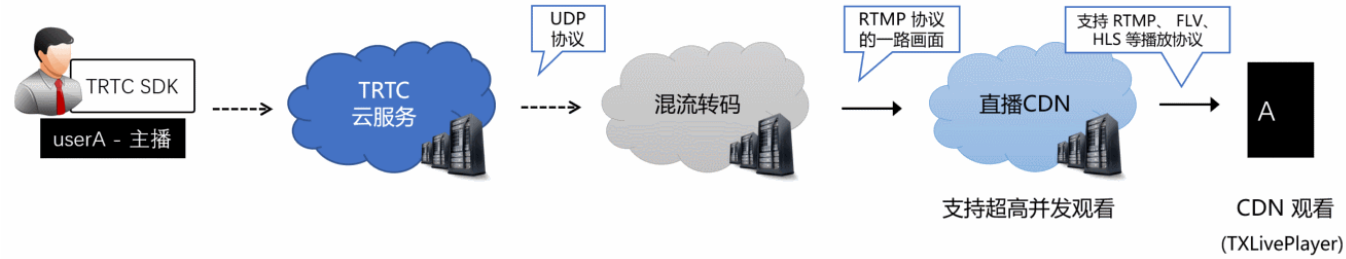
当音视频通话 TRTC 房间中只有一个主播时，TRTC 的旁路推流跟标准的 RTMP 协议直推功能相同，不过 TRTC 的 UDP 相比于 RTMP 有更强大的弱网络抗性。



混合画面的旁路直播

音视频通话TRTC 最擅长的领域就是音视频互动连麦，如果一个房间里同时有多个主播，而 CDN 观看端只希望拉取一路音视频画面，就需要使用 云端混流服务 将多路画面

合并成一路，其原理如下图所示：



为什么不直接播放多路 CDN 画面?
播放多路 CDN 画面很难解决多路画面的延迟对齐问题，同时拉取多路画面所消耗的下载流量也比单独画面要多，所以业内普遍采用云端混流方案。

前提条件

已开通腾讯 云直播 服务。应国家相关部门的要求，直播播放必须配置播放域名，具体操作请参考 添加自有域名。

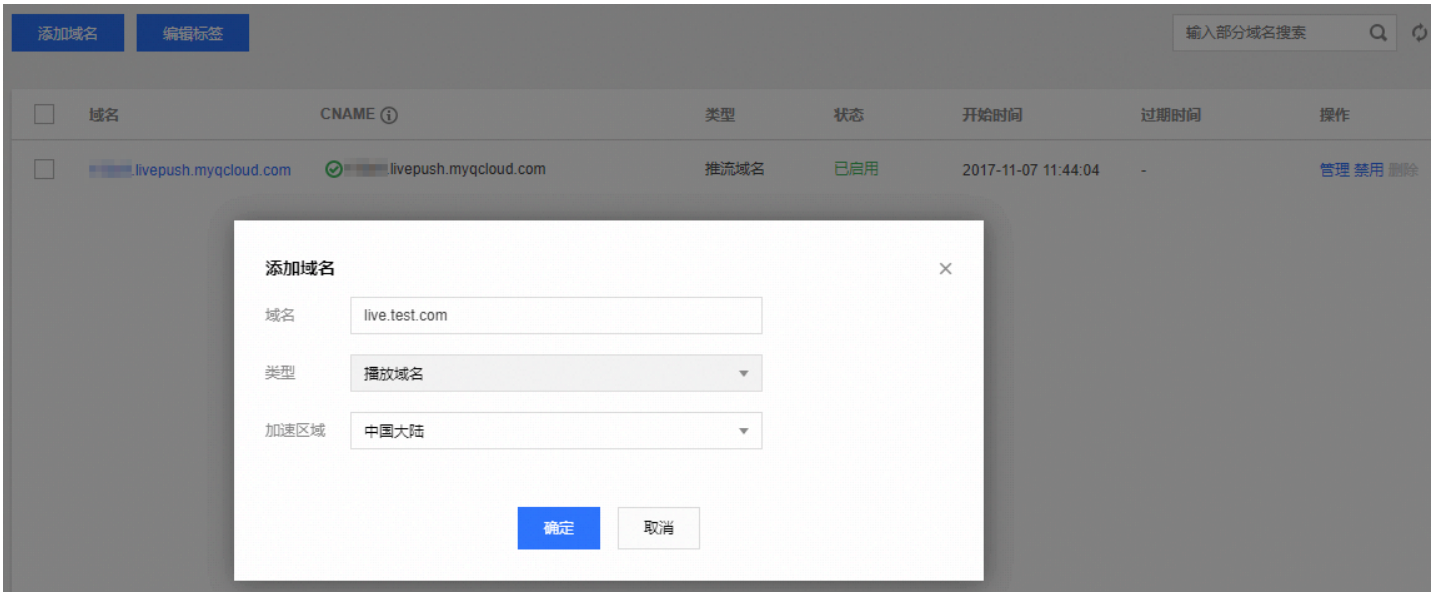
使用步骤

步骤1：开启旁路推流功能

1. 登录 实时音视频控制台。
2. 在左侧导航栏选择**应用管理**，单击目标应用所在行的**功能配置**。
3. 在**旁路推流配置**中，单击**启用旁路推流**右侧的，在弹出的**开启旁路推流功能**对话框中，单击**开启旁路推流功能**即可开通。

步骤2：配置播放域名并完成 CNAME

1. 登录 云直播控制台。
2. 在左侧导航栏选择**域名管理**，您会看到在您的域名列表新增了一个推流域名，格式为 xxxxx.livepush.myqcloud.com，其中 xxxxx 是一个数字，叫做 bizid，您可以在实时音视频控制台 > **应用管理** > **应用信息**中查找到 bizid 信息。
3. 单击**添加域名**，输入您已经备案过的播放域名，选择域名类型为**播放域名**，选择加速区域（默认为**中国大陆**），单击**确定**即可。
4. 域名添加成功后，系统会为您自动分配一个 CNAME 域名（以.liveplay.myqcloud.com为后缀）。CNAME 域名不能直接访问，您需要在域名服务提供商处完成 CNAME 配置，配置生效后，即可享受云直播服务。具体操作请参见 CNAME 配置。



> 注意：
> **不需要添加推流域名，在 步骤1 中开启旁路直播功能后，腾讯云会默认在您的云直播控制台中增加一个格式为 xxxxx.livepush.myqcloud.com 的推流域名，该域名为腾讯云直播服务和 TRTC 服务之间约定的一个默认推流域名，暂时不支持修改。**

步骤3：关联 TRTC 的音视频流到直播 streamId

开启旁路推流功能后， TRTC 房间里的每一路画面都配备一路对应的播放地址，该地址的格式如下：

```
http://播放域名/live/[streamId].flv
```

地址中的 streamId 可以在直播中唯一标识一条直播流，您可以自己指定 streamId，也可以使用系统默认生成的。

方式一：自定义指定 streamId

您可以在调用 TRTCCloud 的 enterRoom 函数时，通过其参数 TRTCParams 中的 streamId 参数指定直播流 ID。
以 iOS 端的 Objective-C 代码为例：

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
TRTCParams *param = [[TRTCParams alloc] init];
param.sdkAppId = 1400000123; // TRTC 的 SDKAppID，创建应用后可获得
param.roomId = 1001; // 房间号
param.userId = @"rexchang"; // 用户名
param.userSig = @"xxxxxxxx"; // 登录签名
param.role = TRTCRoleAnchor; // 角色：主播
param.streamId = @"stream1001"; // 流 ID
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE]; // 请使用 LIVE 模式
```

userSig 的计算方法请参见 如何计算 UserSig。

方式二：系统指定 streamId

开启自动旁路推流后，如果您没有自定义指定 streamId，系统会默认为您生成一个缺省的 streamId，生成规则如下：

- **拼装 streamId 用到的字段**
  - SDKAppID：您可以在 控制台 >**应用管理** > **应用信息**中查找到。
  - bizid：您可以在 控制台 >**应用管理** > **应用信息**中查找到。
  - roomId：由您在 enterRoom 函数的参数 TRTCParams 中指定。
  - userId：由您在 enterRoom 函数的参数 TRTCParams 中指定。
  - streamType：摄像头画面为 main，屏幕分享为 aux（WebRTC 由于同时只支持一路上行，因此 WebRTC 上屏幕分享的流类型是 main）。

- **拼装 streamId 的计算规则**

拼装	2020年01月09日及此后新建的应用	2020年01月09日前创建且使用过的应用
拼装规则	streamId = urlencode(sdkAppId_roomId_userId_streamType)	StreamId = bizid_MD5(roomId_userId_streamType)
计算样例	例如：sdkAppId = 12345678，roomId = 12345，userId = userA，用户当前使用了摄像头。 那么：streamId = 12345678_12345_userA_main	例如：bizid = 1234，roomId = 12345，userId = userA，用户当前使用了摄像头。 那么：streamId = 1234_MD5(12345_userA_main) = 1234_8D0261436C375BB0DEA901D86D7D70E8

步骤4：控制多路画面的混合方案

如果您想要获得混合后的直播画面，需要调用 TRTCCloud 的 setMixTranscodingConfig 接口启动云端混流转码，该接口的参数 TRTCTranscodingConfig 可用于配置：

- 各个子画面的摆放位置和大小。
- 混合画面的画面质量和编码参数。

画面布局的详细配置方法请参考 云端混流转码，整个流程所涉及的各模块关系可以参考 原理解析。

> ⚠ 注意：
> setMixTranscodingConfig 并不是在终端进行混流，而是将混流配置发送到云端，并在云端服务器进行混流和转码。由于混流和转码都需要对原来的音视频数据进行解码和二次编码，所以需要更长的处理时间。因此，混合画面的实际观看时延要比独立画面的多出1s - 2s。

步骤5：获取播放地址并对接播放

当您通过 步骤2 配置完播放域名和 步骤3 完成 streamId 的映射后，即可得到直播的播放地址。播放地址的标准格式为：

```
http://播放域名/live/[streamId].flv
```

例如，您的播放域名为 live.myhost.com ，您将房间（1001）中的用户 userA 的直播流 ID 通过进房参数指定为 streamId = "streamd1001" 。
则您可以得到三路播放地址：

```
rtmp 协议的播放地址：rtmp://live.myhost.com/live/streamd1001
flv 协议的播放地址：http://live.myhost.com/live/streamd1001.flv
hls 协议的播放地址：http://live.myhost.com/live/streamd1001.m3u8
```

我们推荐以 http 为前缀且以 .flv 为后缀的 **http - flv** 地址，该地址的播放具有时延低、秒开效果好且稳定可靠的特点。
播放器选择方面推荐参考如下表格中的指引的方案：

所属平台	对接文档	API 概览	支持的格式
iOS App	接入指引	TXLivePlayer(iOS)	推荐 FLV
Android App	接入指引	TXLivePlayer(Android)	推荐 FLV
Web 浏览器	接入指引	TXLivePusher	桌面端 Chrome 浏览器支持 FLV Mac 端 Safari和移动端手机浏览器仅支持 HLS
微信小程序	接入指引	<live-player> 标签	推荐 FLV

步骤6：优化播放延时

开启旁路直播后的 http - flv 地址，由于经过了直播 CDN 的扩散和分发，观看时延肯定要比直接在 TRTC 直播间里的通话时延要高。
按照目前腾讯云的直播 CDN 技术，如果配合 TXLivePlayer 播放器，可以达到下表中的延时标准：

旁路流类型	TXLivePlayer 的播放模式	平均延时	实测效果
独立画面	极速模式（推荐）	2s - 3s	下图中左侧对比图（橙色）
混合画面	极速模式（推荐）	4s - 5s	下图中右侧对比图（蓝色）

下图中的实测效果，采用了同样的一组手机，左侧 iPhone 6s 使用了 TRTC SDK 进行直播，右侧的小米6 使用 TXLivePlayer 播放器播放 FLV 协议的直播流。

如果您在实测中延时比上表中的更大，可以按照如下指引优化延时：

- **使用 TRTC SDK 自带的 TXLivePlayer**
  普通的 ijkplayer 或者 ffmpeg 基于 ffmpeg 的内核包装出的播放器，缺乏延时调控的能力，如果使用该类播放器播放上述直播流地址，时延一般不可控。
  TXLivePlayer 有一个自研的播放引擎，具备延时调控的能力。
- **设置 TXLivePlayer 的播放模式为极速模式**
  可以通过设置 TXLivePlayerConfig 的三个参数来实现极速模式，以 iOS 为例。
  以 iOS 端的 Objective-C 代码为例：

```
// 设置 TXLivePlayer 的播放模式为极速模式
TXLivePlayerConfig * config = [[TXLivePlayerConfig alloc] init];
config.bAutoAdjustCacheTime = YES;
config.minAutoAdjustCacheTime = 1; // 最小缓冲1s
config.maxAutoAdjustCacheTime = 1; // 最大缓冲1s
[player setConfig:config];
// 启动直播播放
```

相关费用

实现 CDN 直播观看的费用包括**观看费用**和**转码费用**，观看费用为基础费用，转码费用仅在启用 多路画面混合 时才会收取。

> ⚠ 注意：
> 本文中的价格为示例，仅供参考。若价格与实际不符，请以 云直播 > 标准直播 的计费说明为准。

观看费用：通过直播 CDN 观看时产生的费用

通过直播 CDN 观看时，**云直播**将向您收取因观看产生的下行流量/带宽费用，可以根据实际需要选择适合自己的计费方式，默认采用流量计费，详情请参见 云直播 > 标准直播 > 流量带宽 计费说明。

转码费用：启用多路画面混合时收取

如果您启用了 多路画面混合 ，混流需要进行解码和编码，因此会产生额外的混流转码费用。混流转码根据分辨率大小和转码时长进行计费，主播用的分辨率越高，连麦时间（通常在连麦场景才需要混流转码）越长，费用越高，详情请参见 云直播 > 直播转码 计费说明。

示例：
您通过 setVideoEncodrParam() 设置主播的码率（videoBitrate）为1500kbps，分辨率为720P。如果有一位主播跟观众连麦了1个小时，连麦期间开启了多路画面混合，那么产生的转码费用为 0.0325元/分钟 × 60分钟 = 1.95元 。

常见问题

为什么房间里只有一个人时画面又卡又模糊？

请将 enterRoom 中 TRTCAppScene 参数指定为 **TRTCAppSceneLIVE**。

VideoCall 模式针对视频通话做了优化，所以在房间中只有一个用户时，画面会保持较低的码率和帧率以节省用户的网络流量，看起来会感觉又卡又模糊。

实时屏幕分享
iOS

最近更新时间：2022-03-04 10:59:22

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

功能说明

腾讯云视立方音视频通话 TRTC 在 iOS 平台下支持两种不同的屏幕分享方案：

- **应用内分享**
  即只能分享当前 App 的画面，该特性需要 iOS 13 及以上版本的操作系统才能支持。由于无法分享当前 App 之外的屏幕内容，因此适用于对隐私保护要求高的场景。
- **跨应用分享**
  基于苹果的 Replaykit 方案，能够分享整个系统的屏幕内容，但需要当前 App 额外提供一个 Extension 扩展组件，因此对接步骤也相对应用内分享要多一点。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

应用内分享

应用内分享的方案非常简单，只需要调用音视频通话 TRTC SDK 提供的接口 startScreenCaptureInApp 并传入编码参数 TRTCVideoEncParam 即可。该参数可以设置为 nil，此时 SDK 会沿用开始屏幕分享之前的编码参数。

我们推荐的用于 iOS 屏幕分享的编码参数是：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

跨应用分享

示例代码

我们在 Github 中的 **ScreenShare** 目录下放置了一份跨应用分享的示例代码，其包含如下一些文件：

```
├─ TRTC-API-Example-OC // TRTC API Example
| ├─ Basic // 演示跨应用屏幕分享功能
| | ├─ ScreenShare // 演示跨应用屏幕分享功能
```

```
| | | ├── ScreenAnchorViewController.h
| | | ├── ScreenAnchorViewController.m // 主播录屏状态显示界面
| | | ├── ScreenAnchorViewController.xib
| | | ├── ScreenAudienceViewController.h
| | | ├── ScreenAudienceViewController.m // 观众观看录播界面
| | | ├── ScreenAudienceViewController.xib
| | | ├── ScreenEntranceViewController.h
| | | ├── ScreenEntranceViewController.m // 功能入口界面
| | | ├── ScreenEntranceViewController.xib
| | | ├── TRTCBroadcastExtensionLauncher.h
| | | ├── TRTCBroadcastExtensionLauncher.m // 用于唤起系统录屏的辅助代码
| | | ├── TXReplayKit_Screen // 录屏进程 Broadcast Upload Extension 代码详见步骤2
| | | | ├── Info.plist
| | | | ├── SampleHandler.h
| | | | └── SampleHandler.m // 用于接收来自系统的录屏数据
```

您可以通过 README 中的指引跑通该示例 Demo。

对接步骤

iOS 系统上的跨应用屏幕分享，需要增加 Extension 录屏进程以配合主 App 进程进行推流。Extension 录屏进程由系统在需要录屏的时候创建，并负责接收系统采集到屏幕图像。因此需要：

1. 创建 App Group，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 Broadcast Upload Extension 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 TXLiteAVSDK_ReplayKitExt.framework。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 Broadcast Upload Extension 的录屏数据。
4. 使用 Demo 中预先实现的一个 helper class（RPSystemBroadcastPickerView），实现类似腾讯会议 iOS 版中点击一个按钮即可唤起屏幕分享的效果（可选）。

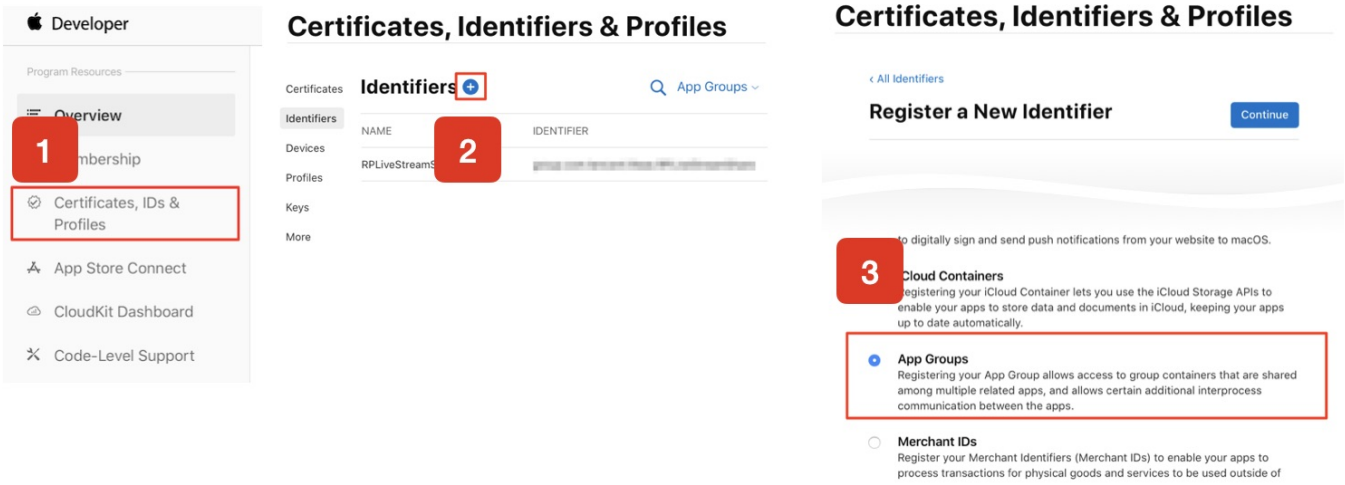
> ⚠ 注意：
> 如果跳过步骤1，也就是不配置 App Group（接口传 nil），屏幕分享依然可以运行，但稳定性要打折扣，故虽然步骤较多，但请尽量配置正确的 App Group 以保障屏幕分享功能的稳定性。

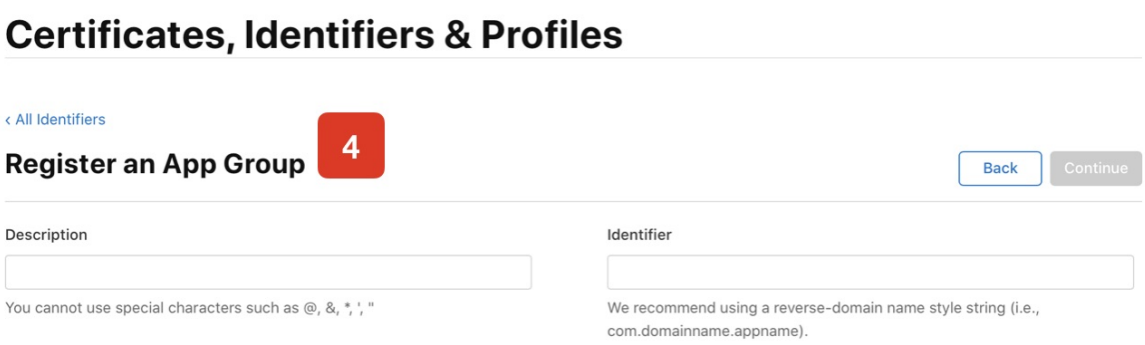
步骤1：创建 App Group

使用您的帐号登录 https://developer.apple.com/，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

1. 单击**Certificates, IDs & Profiles**。
2. 在右侧的界面中单击加号。
3. 选择**App Groups**，单击**Continue**。

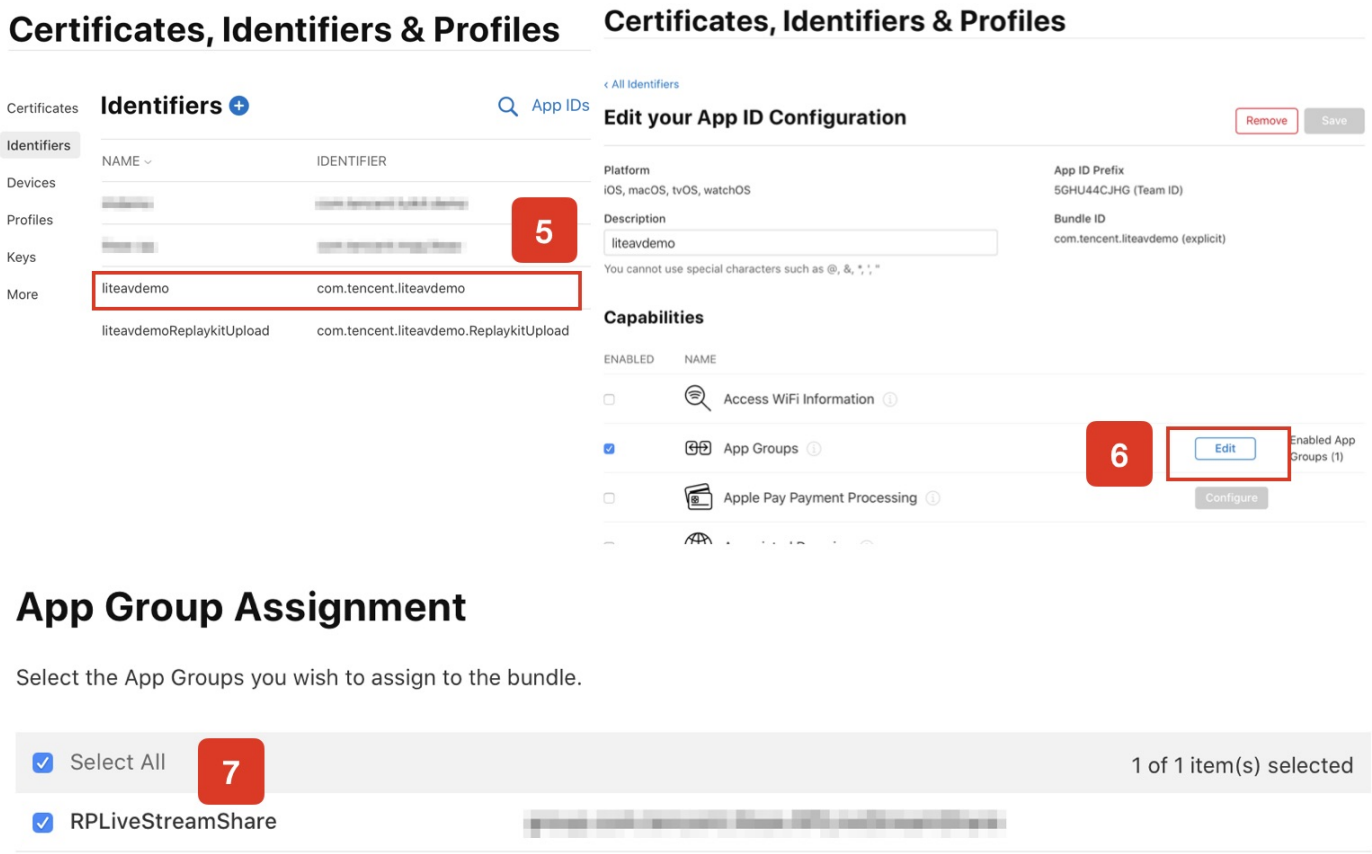
4. 在弹出的表单中填写 Description 和 Identifier，其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击**Continue**。





5. 回到 Identifier 页面，左上边的菜单中选择**App IDs**，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中**App Groups**并单击**Edit**。

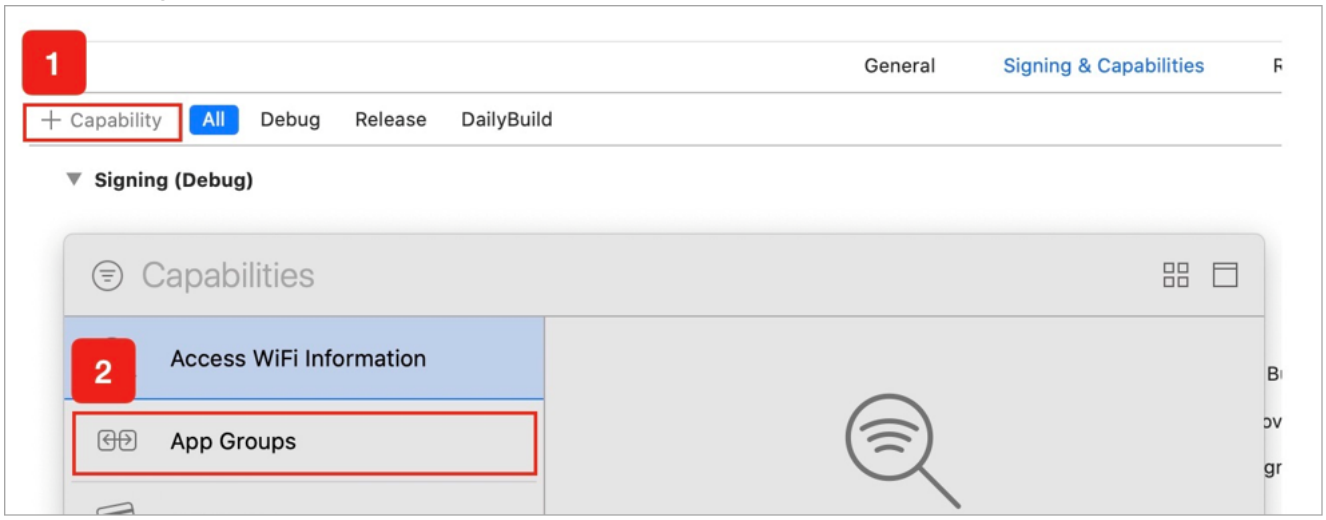
7. 在弹出的表单中选择您之前创建的 App Group，单击**Continue**返回编辑页，单击**Save**保存。



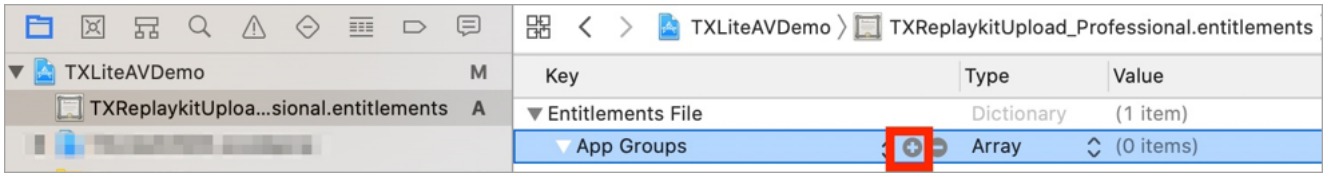
8. 重新下载 Provisioning Profile 并配置到 XCode 中。

步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击**File**、**New** 、**Target...**，选择**Broadcast Upload Extension**。
2. 在弹出的对话框中填写相关信息，**不用**勾选"**Include UI Extension**，单击**Finish**完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。
4. 选中新增加的 Target，依次单击**+ Capability**，双击**App Groups**，如下图：



操作完成后，会在文件列表中生成一个名为 Target名.entitlements 的文件，如下图所示，选中该文件并单击 + 号填写上述步骤中的 App Group 即可。



5. 选中主 App 的 Target，**并按照上述步骤对主 App 的 Target 做同样的处理。**
6. 在新创建的 Target 中，Xcode 会自动创建一个名为 "SampleHandler.m" 的文件，用如下代码进行替换。**需将代码中的 APPGROUP 改为上文中的创建的 App Group Identifier**。

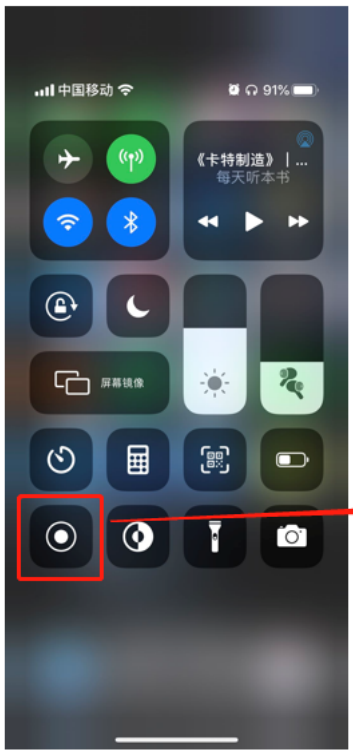
```
#import "SampleHandler.h"
@import TXLiteAVSDK_ReplayKitExt;

#define APPGROUP @"group.com.tencent.liteav.RPLiveStreamShare"

@interface SampleHandler() <TXReplayKitExtDelegate>
@end

@implementation SampleHandler
// 注意：此处的 APPGROUP 需要改成上文中的创建的 App Group Identifier。
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupInfo {
[[TXReplayKitExt sharedInstance] setupWithAppGroup:APPGROUP delegate:self];
}

- (void)broadcastPaused {
// User has requested to pause the broadcast. Samples will stop being delivered.
}

- (void)broadcastResumed {
// User has requested to resume the broadcast. Samples delivery will resume.
}

- (void)broadcastFinished {
[[TXReplayKitExt sharedInstance] finishBroadcast];
// User has requested to finish the broadcast.
}

#pragma mark - TXReplayKitExtDelegate
- (void)broadcastFinished:(TXReplayKitExt *)broadcast reason:(TXReplayKitExtReason)reason
{
NSString *tip = @"";
switch (reason) {
case TXReplayKitExtReasonRequestedByMain:
tip = @"屏幕共享已结束";
break;
case TXReplayKitExtReasonDisconnected:
tip = @"应用断开";
break;
case TXReplayKitExtReasonVersionMismatch:
tip = @"集成错误（SDK 版本号不相符合）";
break;
}

NSError *error = [NSError errorWithDomain:NSStringFromClass(self.class)
code:0
userInfo:@{
NSLocalizedFailureReasonErrorKey:tip
}];
[self finishBroadcastWithError:error];
}

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampleBufferType {
switch (sampleBufferType) {
case RPSampleBufferTypeVideo:
[[TXReplayKitExt sharedInstance] sendVideoSampleBuffer:sampleBuffer];
break;
```

```
case RPSampleBufferTypeAudioApp:
// Handle audio sample buffer for app audio
break;
case RPSampleBufferTypeAudioMic:
// Handle audio sample buffer for mic audio
break;

default:
break;
}
}
@end
```

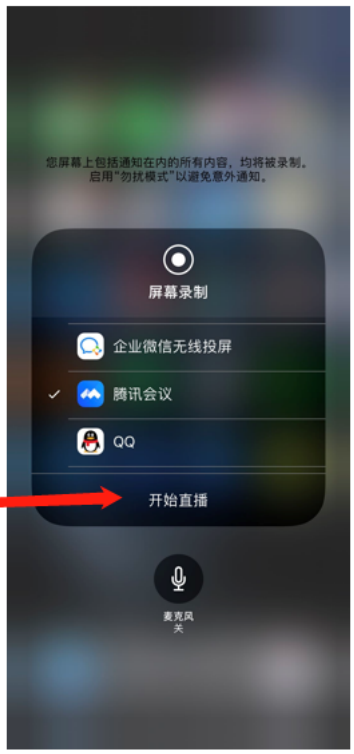
步骤3：对接主 App 端的接收逻辑

按照如下步骤，对接主 App 端的接收逻辑。也就是在用户触发屏幕分享之前，要让主 App 处于“等待”状态，以便随时接收来自 Broadcast Upload Extension 进程的录屏数据。

1. 确保 TRTCCloud 已经关闭了摄像头采集，如果尚未关闭，请调用 stopLocalPreview 关闭摄像头采集。
2. 调用 startScreenCaptureByReplaykit:appGroup: 方法，并传入 步骤1 中设置的 AppGroup，让 SDK 进入“等待”状态。
3. 等待用户触发屏幕分享。如果不实现 步骤4 中的“触发按钮”，屏幕分享就需要用户在 iOS 系统的控制中心，通过长按录屏按钮来触发，这一操作步骤如下图所示：



长按“录屏” 按钮



点击“开始直播”

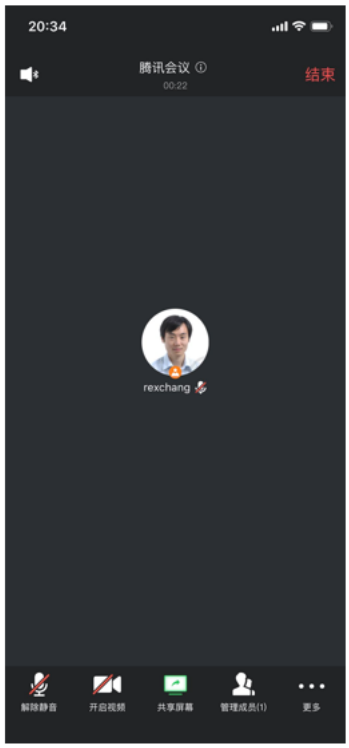
4. 通过调用 stopScreenCapture 接口可以随时中止屏幕分享。

```
// 开始屏幕分享，需要将 APPGROUP 替换为上述步骤中创建的 App Group Identifier。
- (void)startScreenCapture {
TRTCVideoEncParam *videoEncConfig = [[TRTCVideoEncParam alloc] init];
videoEncConfig.videoResolution = TRTCVideoResolution_1280_720;
videoEncConfig.videoFps = 10;
videoEncConfig.videoBitrate = 2000;
//需要将 APPGROUP 替换为上述步骤中创建的 App Group Identifier:
[[TRTCCloud sharedInstance] startScreenCaptureByReplaykit:videoEncConfig
appGroup:APPGROUP];
}
```

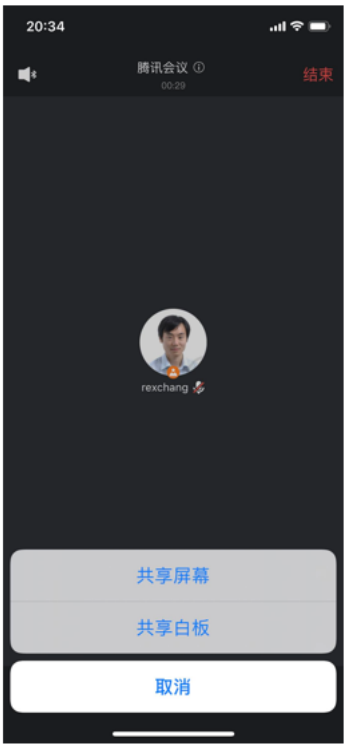
```
// 停止屏幕分享
- (void)stopScreenCapture {
[[TRTCCloud sharedInstance] stopScreenCapture];
}

// 屏幕分享的启动事件通知，可以通过 TRTCCloudDelegate 进行接收
- (void)onScreenCaptureStarted {
[self showTip:@"屏幕分享开始"];
}
```

步骤4：增加屏幕分享的触发按钮（可选）

截止到 步骤3，我们的屏幕分享还必须要用户从控制中心中长按录屏按钮来手动启动。您可通过下述方法实现类似腾讯会议的单击按钮即可触发的效果：



点击分享按钮



触发屏幕分享

开始直播

1. 在 Demo 中寻找 TRTCBroadcastExtensionLauncher 这个类，并将其加入到您的工程中。
2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 TRTCBroadcastExtensionLauncher 中的 launch 函数，就可以唤起屏幕分享功能了。

```
// 自定义按钮响应方法
- (IBAction)onScreenButtonTapped:(id)sender {
[TRTCBroadcastExtensionLauncher launch];
}
```

⚠ 注意：

- 苹果在 iOS 12.0 中增加了 RPSystemBroadcastPickerView 可以从应用中弹出启动器供用户确认启动屏幕分享，到目前为止，RPSystemBroadcastPickerView 尚不支持自定义界面，也没有官方的唤起方法。
- TRTCBroadcastExtensionLauncher 的原理就是遍历 RPSystemBroadcastPickerView 的子 View 寻找 UIButton 并触发了其点击事件。
- 但该方案不被苹果官方推荐，并可能在新一轮的系统更新中失效，因此 步骤4 只是一个可选方案，您需要自行承担风险来选用此方案。

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的

onUserSubStreamAvailable 事件获得这个通知。
希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。

- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

Android

最近更新时间：2022-03-04 10:59:35

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

注意事项

腾讯云视立方音视频通话 TRTC 在 Android 系统上支持屏幕分享，即将当前系统的屏幕内容通过 TRTC SDK 分享给房间里的其他用户。关于此功能，有两点需要注意：

- 音视频通话 TRTC Android 8.6 之前的版本屏幕分享并不像桌面端版本一样支持“辅路分享”，因此在启动屏幕分享时，摄像头的采集需要先被停止，否则会相互冲突；8.6 及之后的版本支持“辅路分享”，则不需要停止摄像头的采集。
- 当一个 Android 系统上的后台 App 在持续使用 CPU 时，很容易会被系统强行杀掉，而且屏幕分享本身又必然会消耗 CPU。要解决这个看似矛盾的冲突，我们需要在 App 启动屏幕分享的同时，在 Android 系统上弹出悬浮窗。由于 Android 不会强杀包含前台 UI 的 App 进程，因此该种方案可以让您的 App 可以持续进行屏幕分享而不被系统自动回收。如下图所示：

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

启动屏幕分享

要开启 Android 端的屏幕分享，只需调用 TRTCCloud 中的 startScreenCapture() 接口即可。但如果要达到稳定和清晰的分享效果，您需要关注如下三个问题：

添加 Activity

在 manifest 文件中粘贴如下 activity（若项目代码中存在则不需要添加）。

```
<activity
android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity"
android:theme="@android:style/Theme.Translucent"/>
```

设定视频编码参数

通过设置 startScreenCapture() 中的首个参数 encParams ，您可以指定屏幕分享的编码质量。如果您指定 encParams 为 null，SDK 会自动使用之前设定的编码参数，我们推荐的参数设定如下：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

弹出悬浮窗以避免被强杀

从 Android 7.0 系统开始，切入到后台运行的普通 App 进程，但凡有 CPU 活动，都很容易会被系统强杀掉。所以当 App 在切入到后台默默进行屏幕分享时，通过弹出悬浮窗的方案，可以避免被系统强杀掉。同时，在手机屏幕上显示悬浮窗也有利于告知用户当前正在做屏幕分享，避免用户泄漏个人隐私。

- **方案1：弹出普通的悬浮窗**
  要弹出类似“腾讯会议”的迷你悬浮窗，您只需要参考示例代码 FloatingView.java 中的实现即可：

```
public void showView(View view, int width, int height) {
mWindowManager = (WindowManager) mContext.getSystemService(Context.WINDOW_SERVICE);
//TYPE_TOAST仅适用于4.4+系统，假如要支持更低版本使用TYPE_SYSTEM_ALERT(需要在manifest中声明权限)
//7.1（包含）及以上系统对TYPE_TOAST做了限制
int type = WindowManager.LayoutParams.TYPE_TOAST;
if (Build.VERSION.SDK_INT > Build.VERSION_CODES.N) {
type = WindowManager.LayoutParams.TYPE_PHONE;
}
mLayoutParams = new WindowManager.LayoutParams(type);
mLayoutParams.flags = WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE;
mLayoutParams.flags |= WindowManager.LayoutParams.FLAG_WATCH_OUTSIDE_TOUCH;
mLayoutParams.width = width;
mLayoutParams.height = height;
mLayoutParams.format = PixelFormat.TRANSLUCENT;
mWindowManager.addView(view, mLayoutParams);
}
```

- **方案2：弹出摄像头预览窗**
  由于音视频通话 TRTC Android 8.6之前的版本屏幕分享并不像桌面端版本一样支持“辅路分享”，因此在启动屏幕分享时，摄像头这一路的视频数据无法上行，否则会相互冲突。8.6及之后的版本支持“辅路分享”，则无此冲突，一样可以使用以下方法。
  那要如何才能做到同时分享屏幕和摄像头画面呢?
  答案很简单：只需要在屏幕上悬浮一个摄像头画面即可，这样一来，TRTC 在采集屏幕画面的同时也会将摄像头画面一并分享出去。

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserSubStreamAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。

- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudListener 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

```
//示例代码：观看屏幕分享的画面
@Override
public void onUserSubStreamAvailable(String userId, boolean available) {
startRemoteSubStreamView(userId);
}
```

常见问题

一个房间里可以同时有多路屏幕分享吗?
目前一个 TRTC 音视频房间只能有一路屏幕分享。

Windows

最近更新时间：2021-12-23 17:06:16

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

功能说明

腾讯云视立方音视频通话 TRTC 支持屏幕分享功能，Windows 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在音视频通话 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（**substream**）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 startScreenCapture 接口时，通过将 TRTCVideoStreamType 参数指定为 TRTCVideoStreamTypeSub 来启用该模式。观看该路画面需要使用专门的 startRemoteSubStreamView 接口。
- **主路分享**
  在音视频通话 TRTC 中，我们一般把摄像头走的通道叫做“主路（**bigstream**）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 startScreenCapture 接口时，通过将 TRTCVideoStreamType 参数指定为 TRTCVideoStreamTypeBig 来启用该模式。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

依赖的 API

API 功能	C++ 版本	C# 版本	Electron 版本
选择分享目标	selectScreenCaptureTarget	selectScreenCaptureTarget	selectScreenCaptureTarget
开始屏幕分享	startScreenCapture	startScreenCapture	startScreenCapture
暂停屏幕分享	pauseScreenCapture	pauseScreenCapture	pauseScreenCapture
恢复屏幕分享	resumeScreenCapture	resumeScreenCapture	resumeScreenCapture
结束屏幕分享	stopScreenCapture	stopScreenCapture	stopScreenCapture

获取分享目标

通过 getScreenCaptureSources 可以枚举可共享的窗口列表，列表通过出参 sourceInfoList 返回。

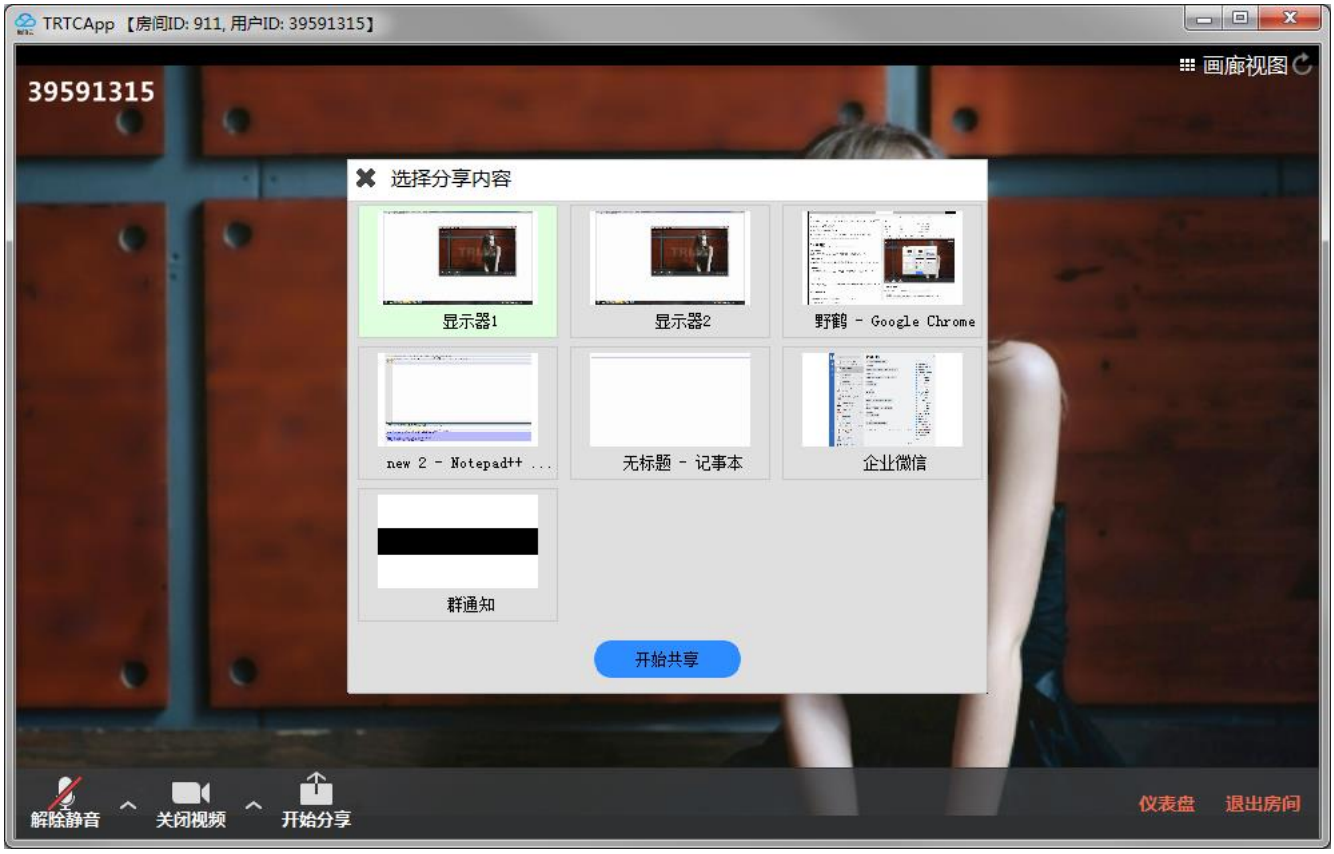
> 说明：
> Windows 里的桌面屏幕也是一个窗口，叫桌面窗口（Desktop），有两台显示器时，每一台显示器都有一个对应的桌面窗口。所以，getScreenCaptureSources 返回的窗口列表里也会有 Desktop 窗口。

sourceInfoList 中每一个 sourceInfo 可以分享的目标，它由如下字段描述。

字段	类型	含义

字段	类型	含义
type	TRTCScreenCaptureSourceType	采集源类型，指定类型为窗口或屏幕
sourceId	HWND	采集源 ID • 对于窗口，该字段指示窗口句柄 • 对于屏幕，该字段指示屏幕 ID
sourceName	string	窗口名字，如果是屏幕则返回 Screen0 Screen1...
thumbWidth	int32	窗口缩略图宽度
thumbHeight	int32	窗口缩略图高度
thumbBGRA	buffer	窗口缩略图的二进制 buffer
iconWidth	int32	窗口图标的宽度
iconHeight	int32	窗口图标的高度
iconBGRA	buffer	窗口图标的二进制 buffer

根据上述信息，您可以实现一个简单的列表页面，将可以分享的目标罗列出来供用户选择，如下图：



选择分享目标

音视频通话TRTC SDK 支持三种分享模式，您可以通过 selectScreenCaptureTarget 来指定：

- **整个屏幕分享**：
  即分享整个屏幕窗口，支持多显示器分屏的情况。需要指定一个 sourceInfoList 中 type 为 TRTCScreenCaptureSourceTypeScreen 的 source 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。
- **指定区域分享**：
  即分享屏幕的某个区域，需要用户圈定区域的位置坐标。需要指定一个 sourceInfoList 中 type 为 TRTCScreenCaptureSourceTypeScreen 的 source 参数，并将 captureRect 设为非 NULL，例如 { 100, 100, 300, 300 }。

- **指定窗口分享：**
  即分享目标窗口的内容，需要用户选择要分享的窗口。需要指定一个 sourceInfoList 中 type 为 TRTCScreenCaptureSourceTypeWindow 的 source 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

> 说明：
> 两个额外参数：
> - 参数 captureMouse 用于指定是否捕获鼠标指针。
> - 参数 highlightWindow 用于指定是否高亮正在共享的窗口，以及当捕获图像被遮挡时提示用户移走遮挡。这部分的 UI 特效是由 SDK 内部实现的。

开始屏幕分享

- 选取分享目标后，使用 startScreenCapture 接口可以启动屏幕分享。
- 分享过程中，您依然可以通过调用 selectScreenCaptureTarget 更换分享目标。
- pauseScreenCapture 和 stopScreenCapture 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resume。

```
/**
* \brief 7.5 **屏幕共享**启动屏幕分享
* \param：rendHwnd - 承载预览画面的 HWND
*/
void startScreenCapture(HWND rendHwnd);

/**
* \brief 7.6 **屏幕共享**暂停屏幕分享
*/
void pauseScreenCapture();

/**
* \brief 7.7 **屏幕共享**恢复屏幕分享
*/
void resumeScreenCapture();

/**
* \brief 7.8 **屏幕共享**关闭屏幕分享
*/
void stopScreenCapture();
```

设定画面质量

您可以通过 setSubStreamEncoderParam 接口设定屏幕分享的画面质量，包括分辨率、码率和帧率，我们提供如下建议参考值：

清晰度级别	分辨率	帧率	码率
超高清（HD+）	1920 × 1080	10	800kbps
高清（HD）	1280 × 720	10	600kbps
标清（SD）	960 × 720	10	400kbps

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserSubStreamAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。

- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

```
//示例代码：观看屏幕分享的画面
void CTRTCCloudSDK::onUserSubStreamAvailable(const char * userId, bool available)
{
LINFO(L"onUserSubStreamAvailable userId[%s] available[%d]\n", UTF82Wide(userId).c_str(), available);
if (available) {
startRemoteSubStreamView(userId, hWnd);
} else {
stopRemoteSubStreamView(userId);
}
}
```

常见问题

一个房间里可以同时有多路屏幕分享吗?
目前一个 TRTC 音视频房间只能有一路屏幕分享。

指定窗口分享（SourceTypeWindow），当窗口大小变化时，视频流的分辨率会不会也跟着变化?
默认情况下，SDK 内部会自动根据分享的窗口大小进行编码参数的调整。
如需固定分辨率，需调用 setSubStreamEncoderParam 接口设置屏幕分享的编码参数，或在调用 startScreenCapture 时指定对应的编码参数。

Mac

最近更新时间：2021-12-23 17:06:22

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

功能说明

腾讯云视立方音视频通话 TRTC 支持屏幕分享功能，Mac 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在音视频通话 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（**substream**）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 startScreenCapture 接口时，通过将 TRTCVideoStreamType 参数指定为 TRTCVideoStreamTypeSub 来启用该模式。观看该路画面需要使用专门的 startRemoteSubStreamView 接口。
- **主路分享**
  在音视频通话 TRTC 中，我们一般把摄像头走的通道叫做“主路（**bigstream**）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 startScreenCapture 接口时，通过将 TRTCVideoStreamType 参数指定为 TRTCVideoStreamTypeBig 来启用该模式。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

获取分享目标

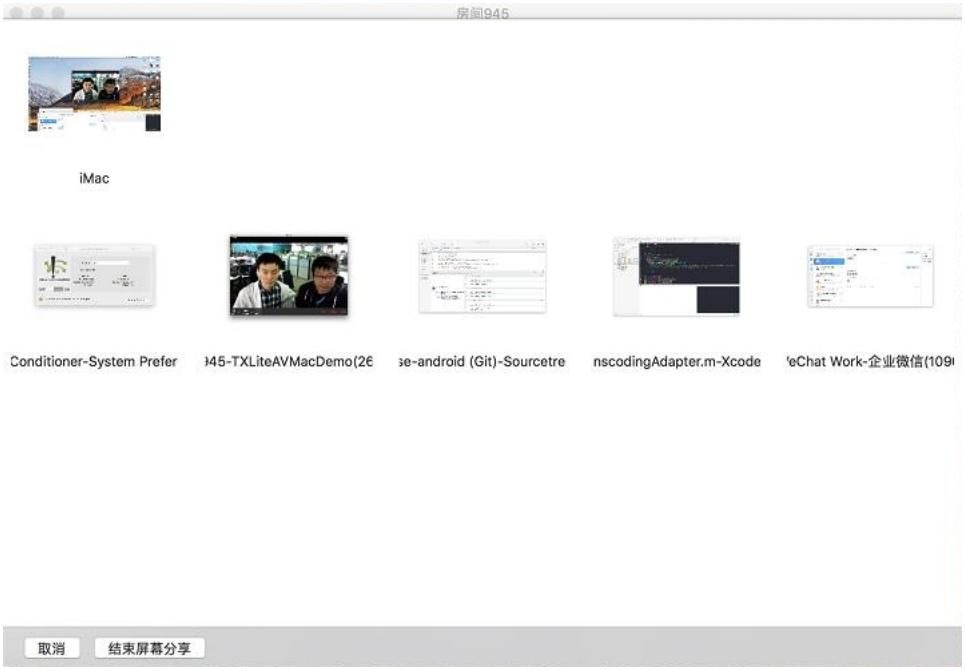
通过 getScreenCaptureSourcesWithThumbnailSize 可以枚举可共享的窗口列表，每一个可共享的目标都是一个 TRTCScreenCaptureSourceInfo 对象。

Mac OS 里的桌面屏幕也是一个可共享目标，普通的 Mac 窗口的 type 为 TRTCScreenCaptureSourceTypeWindow，桌面屏幕的 type 为 TRTCScreenCaptureSourceTypeScreen。

除了 type，每一个 TRTCScreenCaptureSourceInfo 还有如下字段信息：

字段	类型	含义
type	TRTCScreenCaptureSourceType	采集源类型：指定类型为窗口或屏幕
sourceId	NSString	采集源 ID：对于窗口，该字段指示窗口句柄； 对于屏幕，该字段指示屏幕 ID
sourceName	NSString	窗口名字，如果是屏幕则返回 Screen0 Screen1...
extInfo	NSDictionary	共享窗口的附加信息
thumbnail	NSImage	窗口缩略图
icon	NSImage	窗口图标

有了上面这些信息，您就可以实现一个简单的列表页面，将可以分享的目标罗列出来供用户选择，如下图：



选择分享目标

音视频通话 TRTC SDK 支持三种分享模式，您可以通过 selectScreenCaptureTarget 来指定：

- **整个屏幕分享**：
  即把整个屏幕窗口分享出去，支持多显示器分屏的情况。需要指定一个 type 为 TRTCScreenCaptureSourceTypeScreen 的 screenSource 参数，并将 rect 设为 { 0, 0, 0, 0 }。
- **指定区域分享**：
  即把屏幕的某个区域分享出去，需要用户圈定区域的位置坐标。需要指定一个 type 为 TRTCScreenCaptureSourceTypeScreen 的 screenSource 参数，并将 captureRect 设为非 NULL，例如 { 100, 100, 300, 300 }。
- **指定窗口分享**：
  即把目标窗口的内容分享出去，需要用户选择要分享的是哪一个窗口。需要指定一个 type 为 TRTCScreenCaptureSourceTypeWindow 的 screenSource 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

> 说明：
> 两个额外参数：
> - 参数 capturesCursor 用于指定是否捕获鼠标指针。
> - 参数 highlight 用于指定是否高亮正在共享的窗口，以及当捕获图像被遮挡时提示用户移走遮挡。（这一分部的 UI 特效是 SDK 内部实现的）

开始屏幕分享

- 选取分享目标之后，使用 startScreenCapture 接口可以启动屏幕分享。
- 两个函数 pauseScreenCapture 和 stopScreenCapture 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resumeScreenCapture。

```
/**
* 7.6 **屏幕共享**启动屏幕分享
* @param view 渲染控件所在的父控件
*/
- (void)startScreenCapture:(NSView *)view;

/**
```

```
* 7.7 **屏幕共享**停止屏幕采集
* @return 0：成功 <0:失败
*/
- (int)stopScreenCapture;

/**
* 7.8 **屏幕共享**暂停屏幕分享
* @return 0：成功 <0:失败
*/
- (int)pauseScreenCapture;

/**
* 7.9 **屏幕共享**恢复屏幕分享
*
* @return 0：成功 <0:失败
*/
- (int)resumeScreenCapture;
```

设定画面质量

您可以通过 setSubStreamEncoderParam 接口设定屏幕分享的画面质量，包括分辨率、码率和帧率，我们提供如下建议参考值：

清晰度级别	分辨率	帧率	码率
超高清（HD+）	1920 × 1080	10	800kbps
高清（HD）	1280 × 720	10	600kbps
标清（SD）	960 × 720	10	400kbps

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserSubStreamAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。

- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

```
//示例代码：观看屏幕分享的画面

- (void)onUserSubStreamAvailable:(NSString *)userId available:(BOOL)available {
if (available) {
[self.trtcCloud startRemoteSubStreamView:userId view:self.capturePreviewWindow.contentView];
} else {
[self.trtcCloud stopRemoteSubStreamView:userId];
}
}
```

常见问题

一个房间里可以同时有多个人共享屏幕吗?
目前一个 TRTC 音视频房间只能有一路屏幕分享。

指定窗口分享（SourceTypeWindow），当窗口大小变化时，视频流的分辨率会不会也跟着变化?

默认情况下，SDK 内部会自动根据分享的窗口大小进行编码参数的调整。

如需固定分辨率，需调用 setSubStreamEncoderParam 接口设置屏幕分享的编码参数，或在调用 startScreenCapture 时指定对应的编码参数。

Web

最近更新时间：2022-03-04 16:32:45

TRTC Web SDK 屏幕分享支持度请查看 浏览器支持情况。同时 SDK 提供 TRTC.isScreenShareSupported 接口判断当前浏览器是否支持屏幕分享。

本文通过以下不同的场景介绍实现过程。

> 注意：
>
> - Web 端暂不支持发布辅流，发布屏幕分享是通过发布主流实现的。远端屏幕分享流来自于 Web 用户时，RemoteStream.getType() 返回值为 'main'，通常会通过 userId 来标识这是来自 Web 端屏幕分享流。
> - Native（iOS、Android、Mac、Windows 等）端支持发布辅流，并且发布屏幕分享是通过发布辅流实现的。远端屏幕分享流来自于 Native 用户时，RemoteStream.getType() 返回值为 'auxiliary'。

创建和发布屏幕分享流

> 注意：
>
> 请严格按照以下顺序执行创建和发布屏幕分享流的代码。

步骤1：创建负责进行屏幕分享的客户端对象

通常情况下，建议给 userId 加上前缀 share_，用来标识这是一个屏幕分享的客户端对象。

```
const shareClient = TRTC.createClient({
mode: 'rtc',
sdkAppId,
userId, // 例如：‘share_teacher’
userSig
});
// 客户端对象进入房间
try {
await shareClient.join({ roomId });
// ShareClient join room success
} catch (error) {
// ShareClient join room failed
}
```

步骤2：创建屏幕分享流

屏幕分享流包含视频流和音频流。其中音频流分为麦克风音频或者系统音频。

```
// Good 正确用法
// 仅采集屏幕视频流
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// or 采集麦克风音频及屏幕视频流
const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// or 采集系统音频及屏幕视频流
const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });

// Bad 错误用法
const shareStream = TRTC.createStream({ camera: true, screen: true });
// or
const shareStream = TRTC.createStream({ camera: true, screenAudio: true });
```

> 注意：
>
> - audio 与 screenAudio 属性不能同时设为true，camera 与 screenAudio 属性不能同时设为true。关于 screenAudio 更多信息会在本文第五部分介绍。
> - camera 与 screen 属性不能同时设为true。

步骤3：初始化屏幕分享流

初始化时浏览器会向用户请求屏幕共享的内容和权限，如果用户拒绝授权或者系统未授予浏览器屏幕分享的权限，代码会捕获到 NotReadableError 或者 NotAllowedError 错误，这时需要引导用户进行浏览器设置或者系统设置开启屏幕共享权限。

```
try {
await shareStream.initialize();
} catch (error) {
// 当屏幕分享流初始化失败时，提醒用户并停止后续进房发布流程
switch (error.name) {
case 'NotReadableError':
// 提醒用户确保系统允许当前浏览器获取屏幕内容
return;
case 'NotAllowedError':
if (error.message.includes('Permission denied by system')) {
// 提醒用户确保系统允许当前浏览器获取屏幕内容
} else {
// 用户拒绝/取消屏幕分享
}
return;
default:
// 初始化屏幕分享流时遇到了未知错误，提醒用户重试
return;
}
}
```

步骤4：发布屏幕分享流

通过第一步创建的客户端对象进行发布。发布成功后，远端就能收到屏幕分享流。

```
try {
await shareClient.publish(shareStream);
} catch (error) {
// ShareClient failed to publish local stream
}
```

完整代码

```
const shareClient = TRTC.createClient({
mode: 'rtc',
sdkAppId,
userId, // 例如：‘share_teacher’
userSig
});
// 客户端对象进入房间
try {
await shareClient.join({ roomId });
// ShareClient join room success
} catch (error) {
// ShareClient join room failed
}
// 仅采集屏幕视频流
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// or 采集麦克风音频及屏幕视频流
// const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// or 采集系统音频及屏幕视频流
// const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
try {
await shareStream.initialize();
} catch (error) {
// 当屏幕分享流初始化失败时，提醒用户并停止后续进房发布流程
switch (error.name) {
case 'NotReadableError':
```

```
// 提醒用户确保系统允许当前浏览器获取屏幕内容
return;
case 'NotAllowedError':
if (error.message.includes('Permission denied by system')) {
// 提醒用户确保系统允许当前浏览器获取屏幕内容
} else {
// 用户拒绝/取消屏幕分享
}
return;
default:
// 初始化屏幕分享流时遇到了未知错误，提醒用户重试
return;
}
}
try {
await shareClient.publish(shareStream);
} catch (error) {
// ShareClient failed to publish local stream
}
```

屏幕分享参数配置

可设置的参数包括分辨率、帧率和码率，如果有需要可以通过 setScreenProfile() 接口指定 profile，每个 profile 对应着一组分辨率、帧率和码率。SDK 默认使用 '1080p' 的配置。

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// setScreenProfile() 必须在 initialize() 之前调用。
shareStream.setScreenProfile('1080p');
await shareStream.initialize();
```

或者使用自定义分辨率、帧率和码率：

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// setScreenProfile() 必须在 initialize() 之前调用。
shareStream.setScreenProfile({ width: 1920, height: 1080, frameRate: 5, bitrate: 1600 /* kbps */});
await shareStream.initialize();
```

屏幕分享属性推荐列表：

profile	分辨率（宽 x 高）	帧率（fps）	码率 (kbps)
480p	640 x 480	5	900
480p_2	640 x 480	30	1000
720p	1280 x 720	5	1200
720p_2	1280 x 720	30	3000
1080p	1920 x 1080	5	1600
1080p_2	1920 x 1080	30	4000

> ⚠ 注意：
> 建议按照推荐的参数进行配置，避免设置过高的参数，引发不可预料的问题。

停止屏幕分享

```
// 屏幕分享客户端取消发布流
await shareClient.unpublish(shareStream);
// 关闭屏幕分享流
shareStream.close();
// 屏幕分享客户端退房
await shareClient.leave();

// 以上三个步骤非必须，按照场景需求执行需要的代码即可，通常需要添加是否已进房，是否已经发布流的判断，更详细的代码示例请参考 [demo 源码](https://github.com/LiteAVSDK/TRTC_Web/blob/main/base-js/js/share-client.js)。
```

另外用户还可能会通过浏览器自带的按钮停止屏幕分享，因此屏幕分享流需要监听屏幕分享停止事件，并进行相应的处理。


```
// 屏幕分享流监听屏幕分享停止事件
shareStream.on('screen-sharing-stopped', event => {
// 屏幕分享客户端停止推流
await shareClient.unpublish(shareStream);
// 关闭屏幕分享流
shareStream.close();
// 屏幕分享客户端退房
await shareClient.leave();
});
```

同时发布摄像头视频和屏幕分享

一个 Client 至多只能同时发布一路音频和一路视频。同时发布摄像头视频和屏幕分享，需要创建两个 Client，让它们“各司其职”。
例如创建两个 Client 分别为：

- **client**：负责发布本地音视频流，并订阅除了 shareClient 之外的所有远端流。
- **shareClient**：负责发布屏幕分享流，且不订阅任何远端流。

> ⚠ 注意：
>
> - 负责屏幕分享的 shareClient 需要关闭自动订阅，否则会出现重复订阅远端流的情况。请参见 API说明。
> - 负责本地音视频流发布的 client 需要取消订阅 shareClient 发布的流。否则会出现自己订阅自己的情况。

示例代码：

```
const client = TRTC.createClient({ mode: 'rtc', sdkAppId, userId, userSig });
// 需要设置 shareClient 关闭自动订阅远端流，即：autoSubscribe: false
const shareClient = TRTC.createClient({ mode: 'rtc', sdkAppId, `share_${userId}`, userSig, autoSubscribe: false,});

// 负责本地音视频流发布的 client 需要取消订阅 shareClient 发布的流。
client.on('stream-added', event => {
const remoteStream = event.stream;
const remoteUserId = remoteStream.getUserId();
if (remoteUserId === `share_${userId}`) {
// 取消订阅自己的屏幕分享流
client.unsubscribe(remoteStream);
} else {
// 订阅其他一般远端流
client.subscribe(remoteStream);
}
});
```

```
await client.join({ roomId });
await shareClient.join({ roomId });

const localStream = TRTC.createStream({ audio: true, video: true, userId });
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });

// ... 省略初始化和发布相关代码，按需发布流即可。
```

屏幕分享采集系统声音

采集系统声音只支持 Chrome M74+ ，在 Windows 和 Chrome OS 上，可以捕获整个系统的音频，在 Linux 和 Mac 上，只能捕获选项卡的音频。其它 Chrome 版本、其它系统、其它浏览器均不支持。

```
// 创建屏幕分享流 screenAudio 请设置为 true， 不支持同时采集系统和麦克风音量，请勿同时设置 audio 属性为 true
const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
await shareStream.initialize();
...
```

在弹出的对话框中勾选 分享音频 ，发布的 stream 将会带上系统声音。

Flutter

最近更新时间：2021-12-23 17:06:36

基于 Android 平台

腾讯云视立方音视频通话 TRTC 在 Android 系统上支持屏幕分享，即将当前系统的屏幕内容通过音视频通话 TRTC SDK 分享给房间里的其他用户。关于此功能，有两点需要注意：

- 音视频通话 TRTC Android 8.6 之前的版本屏幕分享并不像桌面端版本一样支持“辅路分享”，因此在启动屏幕分享时，摄像头的采集需要先被停止，否则会相互冲突；8.6 及之后的版本支持“辅路分享”，则不需要停止摄像头的采集。
- 当一个 Android 系统上的后台 App 在持续使用 CPU 时，很容易会被系统强行杀掉，而且屏幕分享本身又必然会消耗 CPU。要解决这个看似矛盾的冲突，我们需要在 App 启动屏幕分享的同时，在 Android 系统上弹出悬浮窗。由于 Android 不会强杀包含前台 UI 的 App 进程，因此该种方案可以让您的 App 可以持续进行屏幕分享而不被系统自动回收。如下图所示：

启动屏幕分享

要开启 Android 端的屏幕分享，只需调用 TRTCCloud 中的 startScreenCapture() 接口即可。但如果要达到稳定和清晰的分享效果，您需要关注如下三个问题：

- **添加 Activity**
  在 manifest 文件中粘贴如下 activity（若项目代码中存在则不需要添加）。

```
<activity
android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity"
android:theme="@android:style/Theme.Translucent"/>
```

- **设定视频编码参数**
  通过设置 startScreenCapture() 中的首个参数 encParams，您可以指定屏幕分享的编码质量。如果您指定 encParams 为 null，SDK 会自动使用之前设定的编码参数，我们推荐的参数设定如下：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

说明：

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

- **弹出悬浮窗以避免被强杀**
  从 Android 7.0 系统开始，切入到后台运行的普通 App 进程，但凡有 CPU 活动，都很容易会被系统强杀掉。所以当 App 在切入到后台默默进行屏幕分享时，通过弹出悬浮窗的方案，可以避免被系统强杀掉。同时，在手机屏幕上显示悬浮窗也有利于告知用户当前正在做屏幕分享，避免用户泄漏个人隐私。

  处理方案：弹出普通的悬浮窗
  要弹出类似“腾讯会议”的迷你悬浮窗，您只需要参考示例代码 tool.dart 中的实现即可：

```
//屏幕分享时弹出小浮窗，防止切换到后台应用被杀死
static void showOverlayWindow() {
SystemWindowHeader header = SystemWindowHeader(
title: SystemWindowText(
text: "屏幕分享中", fontSize: 14, textColor: Colors.black45),
decoration: SystemWindowDecoration(startColor: Colors.grey[100]),
);
SystemAlertWindow.showSystemWindow(
width: 18,
height: 95,
header: header,
margin: SystemWindowMargin(top: 200),
gravity: SystemWindowGravity.TOP,
);
}
```

基于 iOS 平台

- **应用内分享**
  即只能分享当前 App 的画面，该特性需要 iOS 13 及以上版本的操作系统才能支持。由于无法分享当前 App 之外的屏幕内容，因此适用于对隐私保护要求高的场景。
- **跨应用分享**
  基于苹果的 Replaykit 方案，能够分享整个系统的屏幕内容，但需要当前 App 额外提供一个 Extension 扩展组件，因此对接步骤也相对应用内分享要多一点。

方案1：iOS 平台应用内分享

应用内分享的方案非常简单，只需要调用音视频通话 TRTC SDK 提供的接口 startScreenCapture 并传入编码参数 TRTCVideoEncParam 和 参数 appGroup 设置为 '' 。TRTCVideoEncParam 参数可以设置为 null，此时 SDK 会沿用开始屏幕分享之前的编码参数。

我们推荐的用于 iOS 屏幕分享的编码参数是：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

说明：

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

方案2：iOS 平台跨应用分享

示例代码

我们在 Github 中的 ** trtc_demo/ios ** 目录下放置了一份跨应用分享的示例代码，其包含如下一些文件：

```
├── Broadcast.Upload //录屏进程 Broadcast Upload Extension 代码详见步骤2
│ ├── Broadcast.Upload.entitlements //用于设置进程间通信的 AppGroup 信息
│ ├── Broadcast.UploadDebug.entitlements //用于设置进程间通信的 AppGroup 信息（debug环境）
│ ├── Info.plist
│ └── SampleHandler.swift // 用于接收来自系统的录屏数据
├── Resource // 资源文件
├── Runner // TRTC 精简化 Demo
├── TXLiteAVSDK_ReplayKitExt.framework //TXLiteAVSDK_ReplayKitExt SDK
```

您可以通过 README 中的指引跑通该示例 Demo。

对接步骤

iOS 系统上的跨应用屏幕分享，需要增加 Extension 录屏进程以配合主 App 进程进行推流。Extension 录屏进程由系统在需要录屏的时候创建，并负责接收系统采集到屏幕图像。因此需要：

1. 创建 App Group，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 Broadcast Upload Extension 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 TXLiteAVSDK_ReplayKitExt.framework。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 Broadcast Upload Extension 的录屏数据。
4. 编辑 pubspec.yaml 文件引入 replay_kit_launcher 插件，实现类似 TRTC Demo Screen 中点击一个按钮即可唤起屏幕分享的效果（可选）。

```
# 引入 trtc sdk和replay_kit_launcher
dependencies:
tencent_trtc_cloud: ^0.2.1
replay_kit_launcher: ^0.2.0+1
```

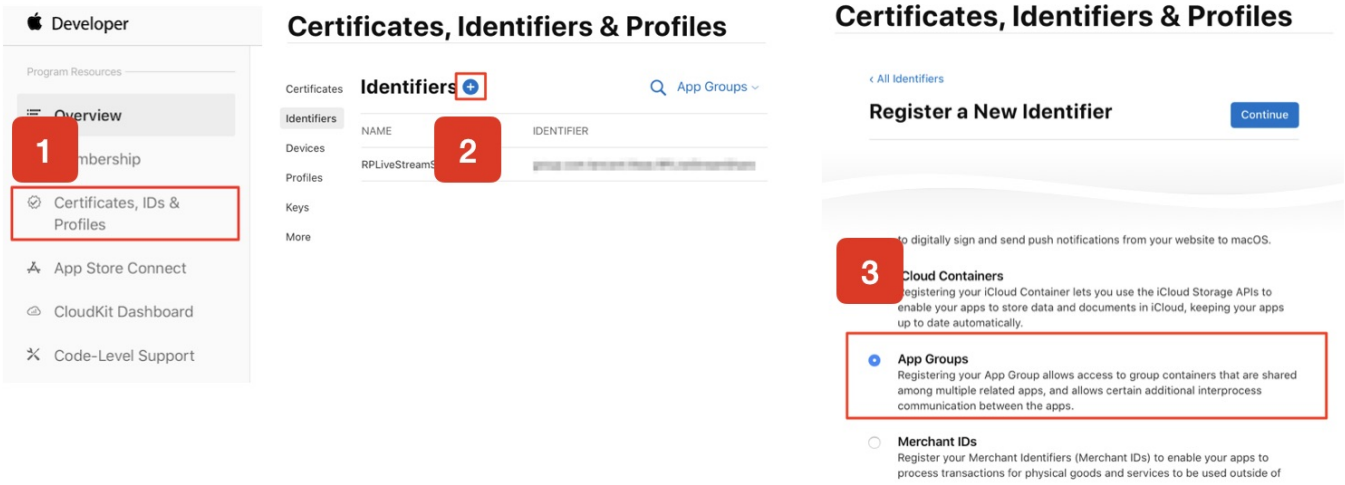
> ⚠ 注意：
> 如果跳过 步骤1，也就是不配置 App Group（接口传 null），屏幕分享依然可以运行，但稳定性要打折扣，故虽然步骤较多，但请尽量配置正确的 App Group 以保障屏幕分享功能的稳定性。

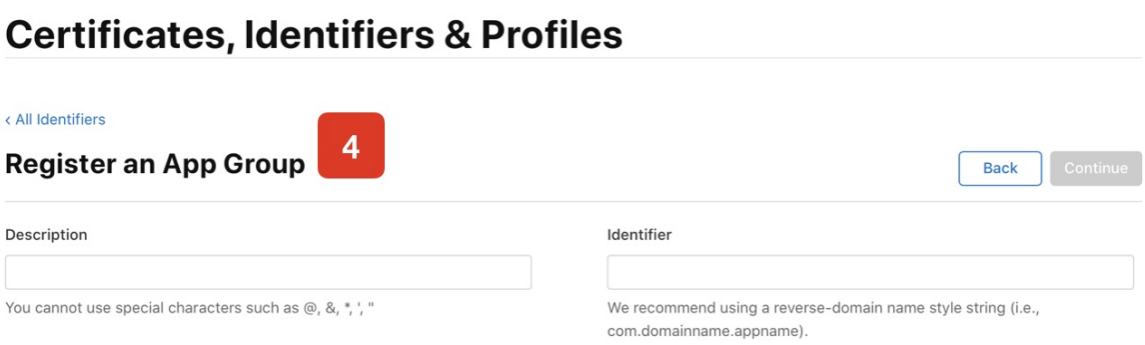
步骤1：创建 App Group

使用您的帐号登录 https://developer.apple.com/ ，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

1. 单击**Certificates, IDs & Profiles**。
2. 在右侧的界面中单击加号。
3. 选择**App Groups**，单击**Continue**。

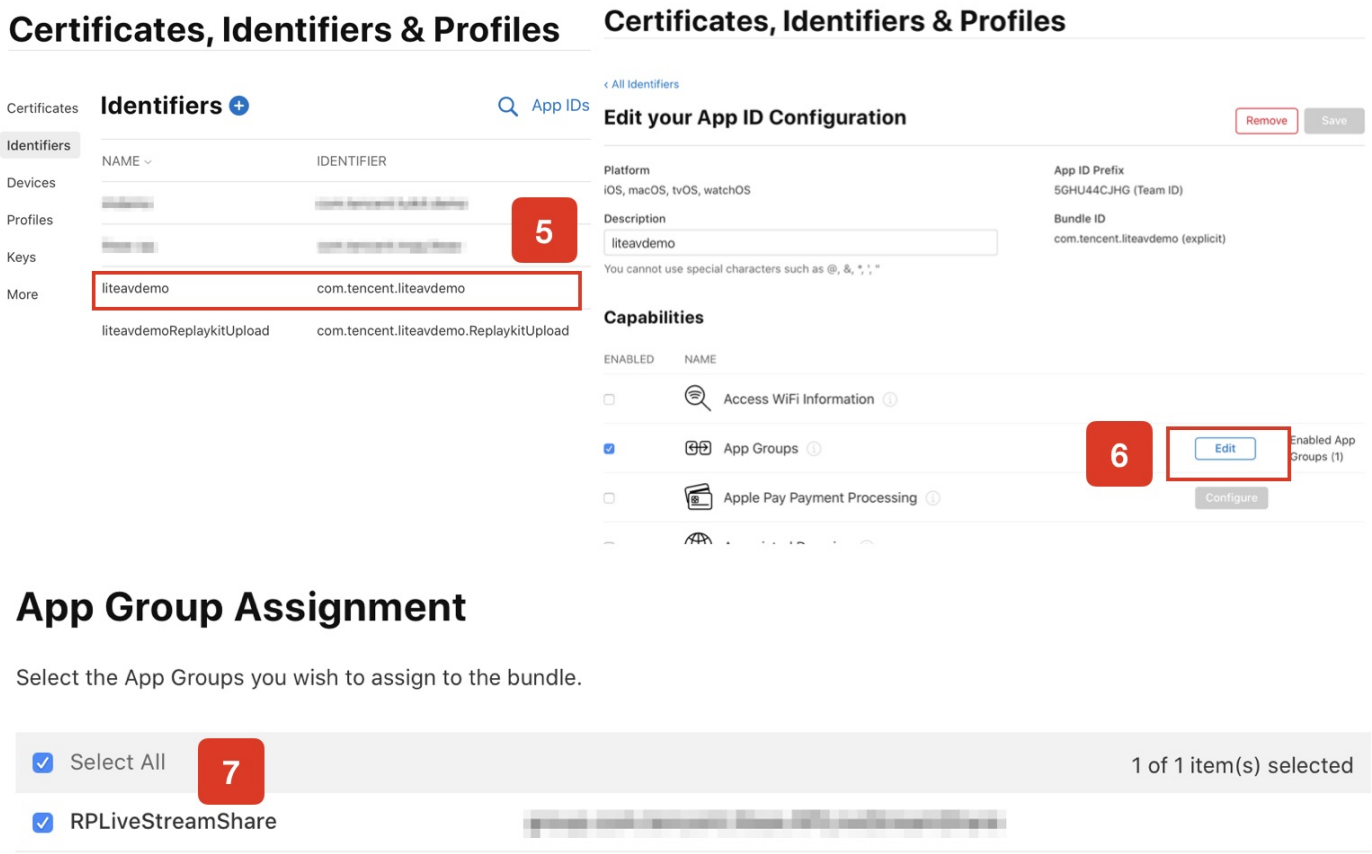
4. 在弹出的表单中填写 Description 和 Identifier，其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击**Continue**。





5. 回到 Identifier 页面，左上边的菜单中选择**App IDs**，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中**App Groups**并单击**Edit**。

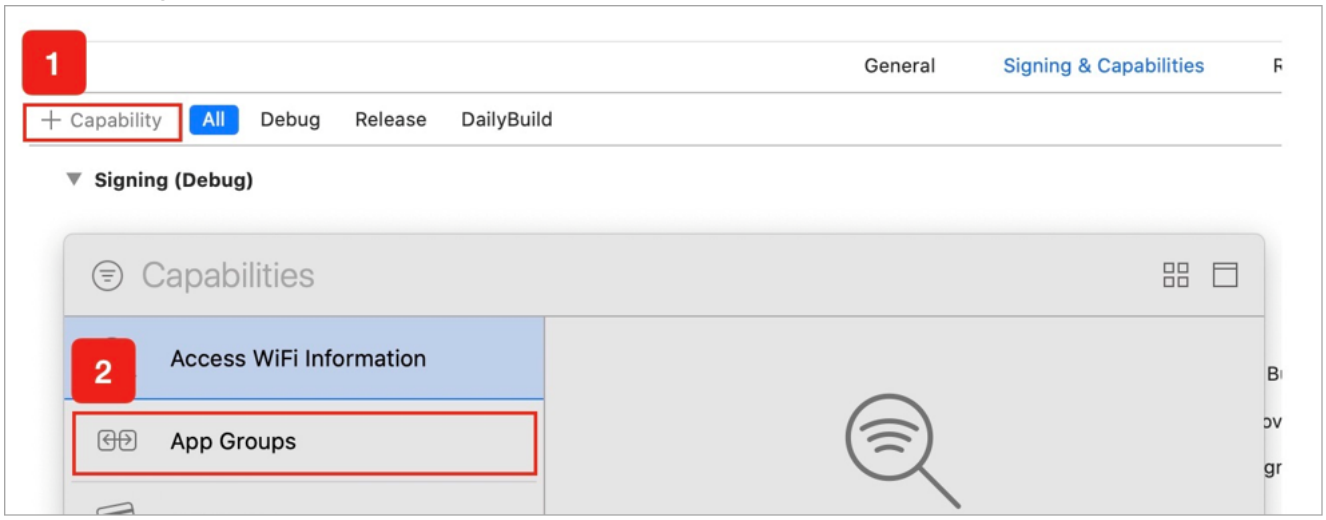
7. 在弹出的表单中选择您之前创建的 App Group，单击**Continue**返回编辑页，单击**Save**保存。



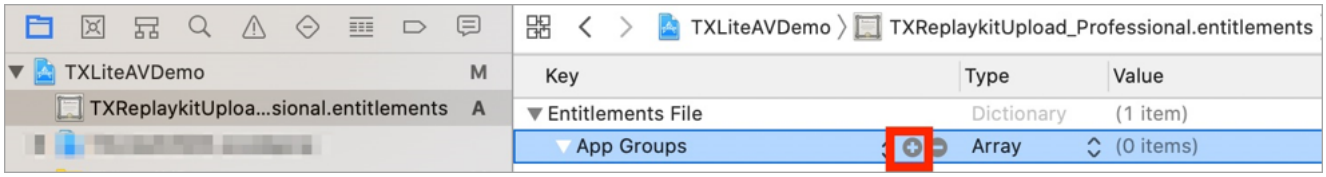
8. 重新下载 Provisioning Profile 并配置到 XCode 中。

步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击**File > New >Target...**，选择**Broadcast Upload Extension**。
2. 在弹出的对话框中填写相关信息，**不用**勾选"**Include UI Extension**，单击**Finish**完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。
4. 选中新增加的 Target，依次单击**+ Capability**，双击**App Groups**，如下图：



操作完成后，会在文件列表中生成一个名为 Target名.entitlements 的文件，如下图所示，选中该文件并单击 + 号填写上述步骤中的 App Group 即可。



5. 选中主 App 的 Target，**并按照上述步骤对主 App 的 Target 做同样的处理。**
6. 在新创建的 Target 中，Xcode 会自动创建一个名为 "SampleHandler.swift" 的文件，用如下代码进行替换。**需将代码中的 APPGROUP 改为上文中的创建的 App Group Identifier。**

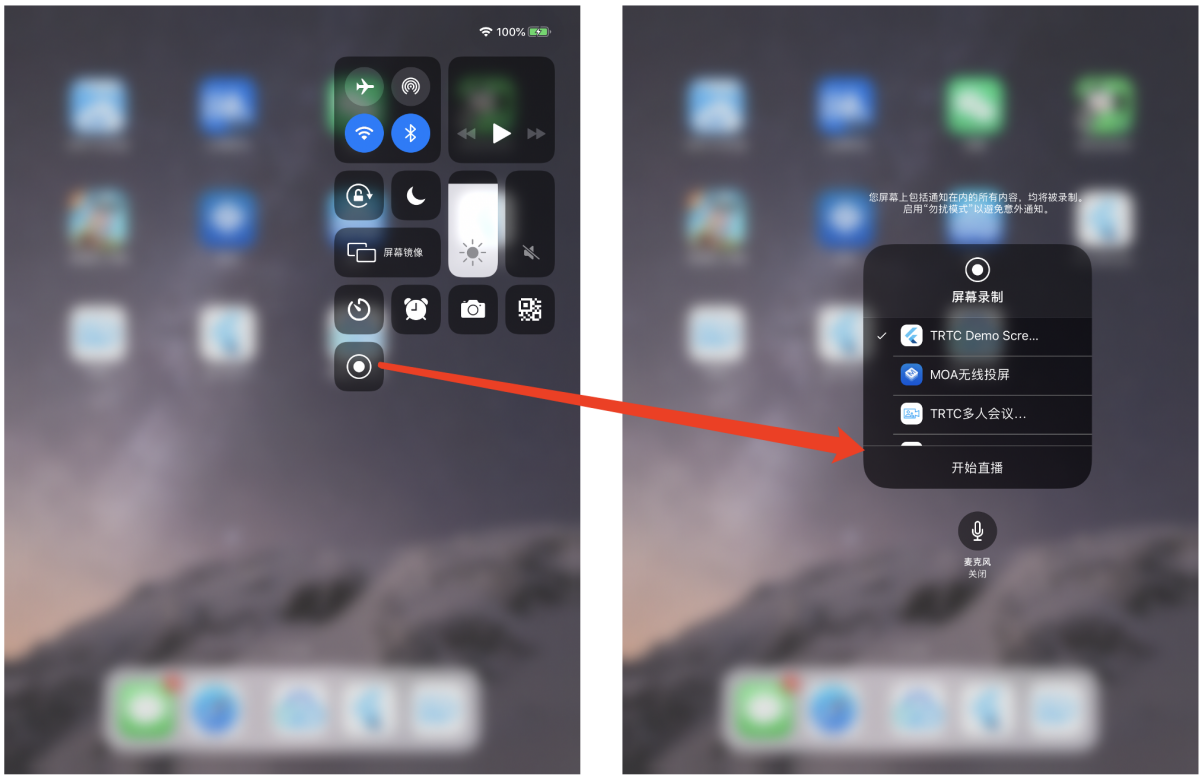
```
import ReplayKit
import TXLiteAVSDK_ReplayKitExt


let APPGROUP = "group.com.tencent.comm.trtc.demo"

class SampleHandler: RPBroadcastSampleHandler, TXReplayKitExtDelegate {

let recordScreenKey = Notification.Name.init("TRTCRecordScreenKey")

override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
// User has requested to start the broadcast. Setup info from the UI extension can be supplied but optional.
TXReplayKitExt.sharedInstance().setup(withAppGroup: APPGROUP, delegate: self)
}

override func broadcastPaused() {
// User has requested to pause the broadcast. Samples will stop being delivered.
}

override func broadcastResumed() {
// User has requested to resume the broadcast. Samples delivery will resume.
}

override func broadcastFinished() {
// User has requested to finish the broadcast.
TXReplayKitExt.sharedInstance() .finishBroadcast()
}

func broadcastFinished(_ broadcast: TXReplayKitExt, reason: TXReplayKitExtReason) {
var tip = ""
switch reason {
case TXReplayKitExtReason.requestedByMain:
tip = "屏幕共享已结束"
break
case TXReplayKitExtReason.disconnected:
tip = "应用断开"
break
case TXReplayKitExtReason.versionMismatch:
tip = "集成错误（SDK 版本号不相符合）"
break
default:
break
}

let error = NSError(domain: NSStringFromClass(self.classForCoder), code: 0, userInfo: [NSLocalizedFailureReasonErrorKey:tip])
finishBroadcastWithError(error)
}

override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType: RPSampleBufferType) {
switch sampleBufferType {
case RPSampleBufferType.video:
// Handle video sample buffer
TXReplayKitExt.sharedInstance() .sendVideoSampleBuffer(sampleBuffer)
break
case RPSampleBufferType.audioApp:
// Handle audio sample buffer for app audio
```

```
break
case RPSampleBufferType.audioMic:
// Handle audio sample buffer for mic audio
break
@unknown default:
// Handle other sample buffer types
fatalError("Unknown type of sample buffer")
}
}
}
```

步骤3：对接主 App 端的接收逻辑

按照如下步骤，对接主 App 端的接收逻辑。也就是在用户触发屏幕分享之前，要让主 App 处于“等待”状态，以便随时接收来自 Broadcast Upload Extension 进程的录屏数据。

1. 确保 TRTCCloud 已经关闭了摄像头采集，如果尚未关闭，请调用 stopLocalPreview 关闭摄像头采集。
2. 调用 startScreenCapture 方法，并传入 步骤1 中设置的 AppGroup，让 SDK 进入“等待”状态。
3. 等待用户触发屏幕分享。如果不实现 步骤4 中的“触发按钮”，屏幕分享就需要用户在 iOS 系统的控制中心，通过长按录屏按钮来触发，这一操作步骤如下图所示：



长按“录制”按钮　　点击开始直播

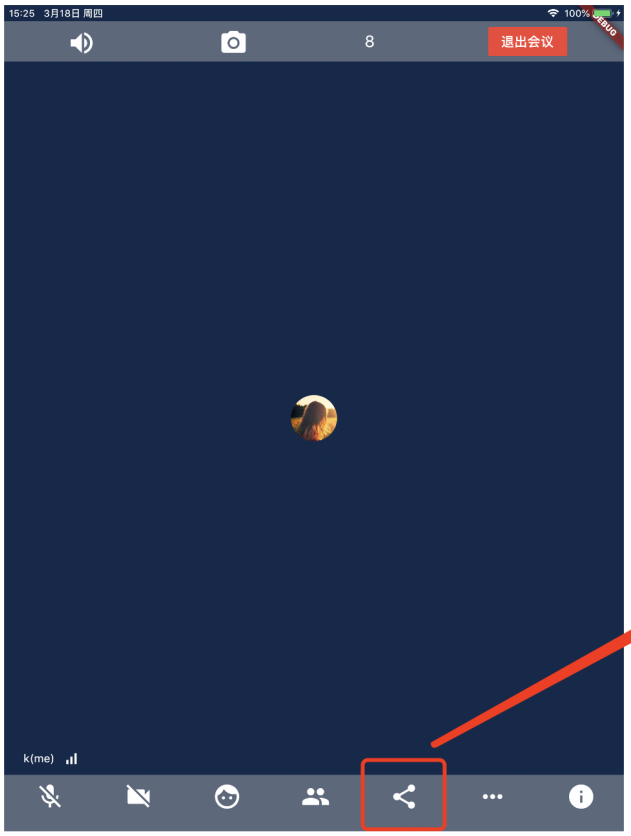
4. 通过调用 stopScreenCapture 接口可以随时中止屏幕分享。

```
// 开始屏幕分享，需要将 APPGROUP 替换为上述步骤中创建的 App Group
trtcCloud.startScreenCapture(
TRTCVideoEncParam(
videoFps: 10,
videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720,
videoBitrate: 1600,
videoResolutionMode: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT,
),
iosAppGroup,
```

```
);

// 停止屏幕分享
await trtcCloud.stopScreenCapture();


// 屏幕分享的启动事件通知，可以通过 TRTCCloudListener 进行接收
onRtcListener(type, param){
if (type == TRTCCloudListener.onScreenCaptureStarted) {
//屏幕分享开始
}
}
```

步骤4：增加屏幕分享的触发按钮（可选）

截止到 步骤3，我们的屏幕分享还必须要用户从控制中心中长按录屏按钮来手动启动。您可通过下述方法实现类似 TRTC Demo Screen 的单击按钮即可触发的效果：



点击“分享”按钮

开始直播

1. 将 replay_kit_launcher 插件加入到您的工程中。
2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName); 函数，就可以唤起屏幕分享功能了。

```
// 自定义按钮响应方法
onShareClick() async {
if (Platform.isAndroid) {
if (await SystemAlertWindow.requestPermissions) {
MeetingTool.showOverlayWindow();
}
} else {
//屏幕分享功能只能在真机测试
```

```
ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);
}
}
```

观看屏幕分享

观看 Android / iOS 屏幕分享

若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudListener 中的 onUserVideoAvailable 事件获得这个通知。
希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

常见问题

一个房间里可以同时有多路屏幕分享吗?

目前一个音视频通话 TRTC 音视频房间只能有一路屏幕分享。

云端混流转码

最近更新时间：2021-12-23 17:06:45

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

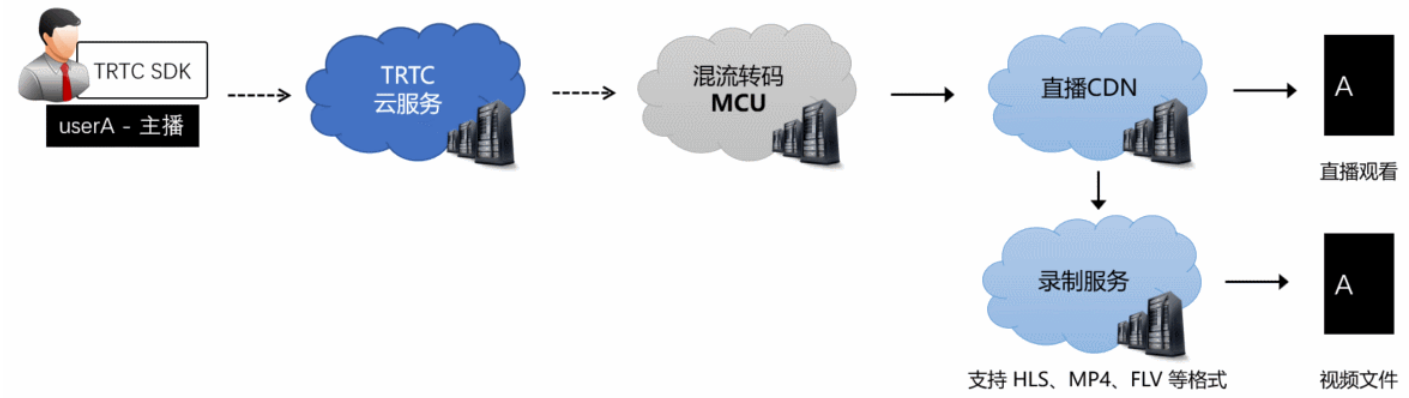
不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

适用场景

在 CDN 直播观看 和 云端录制回放 等应用场景中，常需要将 TRTC 房间里的多路音视频流混合成一路，您可以使用腾讯云服务端的 MCU 的混流转码集群完成该项工作。MCU 集群能将多路音视频流进行按需混合，并将最终生成的视频流分发给直播 CDN 和云端录制系统。

云端混流有两种控制方式：

- **方案一**：使用服务端 REST 接口 StartMCUMixTranscode（数字房间号版本 / 字符串房间号版本）和 StopMCUMixTranscode（数字房间号版本 / 字符串房间号版本）进行控制，该接口还可以同时支持启动 CDN 观看和云端录制。
- **方案二**：使用客户端 TRTC SDK 的 setMixTranscodingConfig 接口进行控制，其原理如下图：

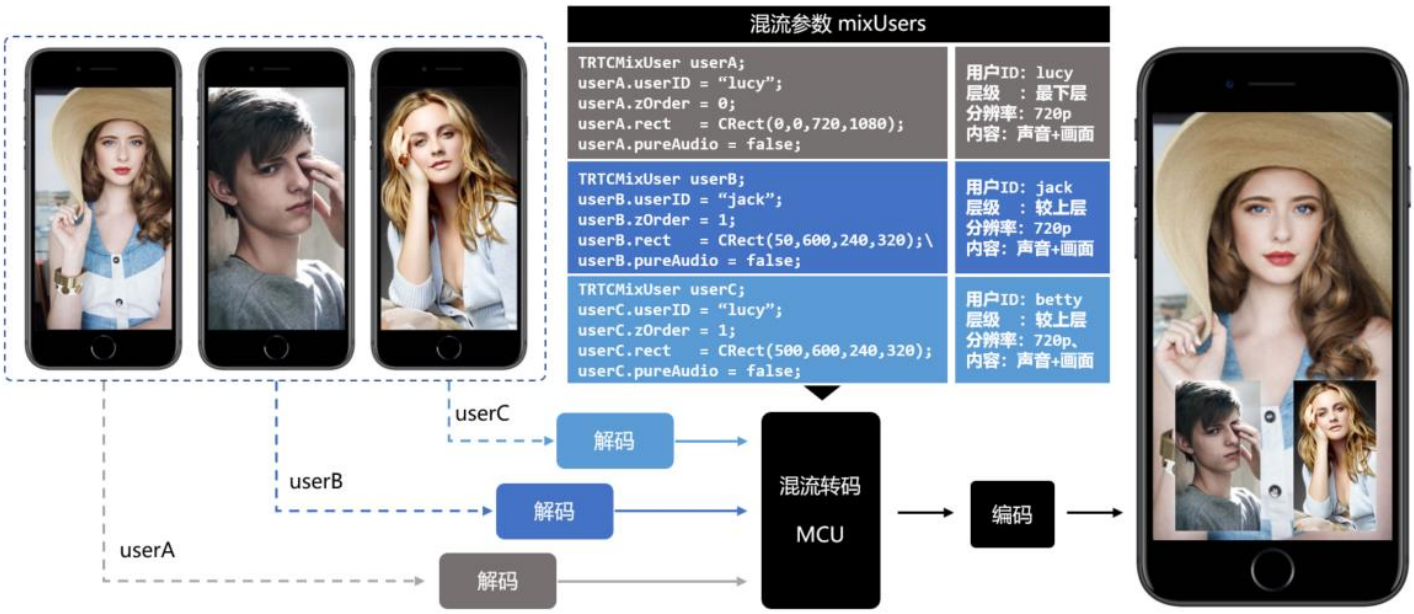


> ⚠ 注意：
> 方案二支持 iOS、Android、Windows、Mac、Electron、Flutter 和桌面浏览器平台的 SDK，如果您希望在微信小程序上也实现混流功能，请使用方案一。

原理解析

云端混流包含解码、混合和再编码三个过程：

- **解码**：MCU 需要将多路音视频流进行解码，包括视频解码和音频解码。
- **混合**：MCU 需要将多路画面混合在一起，并根据来自 SDK 的混流指令实现具体 的排版方案。同时，MCU 也需要将解码后的多路音频信号进行混音处理。
- **编码**：MCU 需要将混合后的画面和声音进行二次编码，并封装成一路音视频流，交给下游系统（例如直播和录制）。



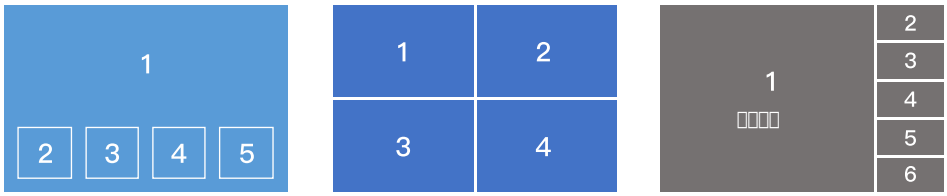
方案一：服务端 REST API 混流方案

启动混流

由您的服务器调用 REST API StartMCUMixTranscode 可以启动云端混流，对于此 API 您需关注如下细节：

步骤1：设置画面排版模式（必需）

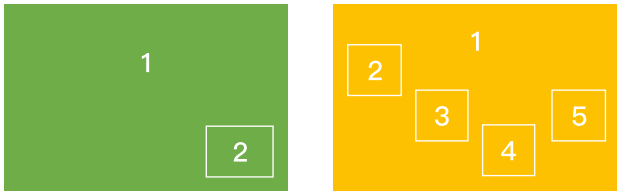
通过 StartMCUMixTranscode 中的 LayoutParams 参数，可以设置如下几种排版模式：



悬浮模板
LayoutParams.Template=0

九宫格模板
LayoutParams.Template=1

屏幕分享模板
LayoutParams.Template=2



画中画模板
LayoutParams.Template=3

自定义模板
LayoutParams.Template=4

悬浮模板（LayoutParams.Template = 0）

- 第一个进入房间的用户的视频画面会铺满整个屏幕，其他用户的视频画面从左下角依次水平排列，显示为小画面。
- 最多4行，每行最多4个，小画面悬浮于大画面之上。
- 最多支持1个大画面和15个小画面。
- 如果用户只发送音频，仍然会占用画面位置。

九宫格模板（LayoutParams.Template = 1）

- 所有用户的视频画面大小一致，平分整个屏幕，人数越多，每个画面的尺寸越小。
- 最多支持16个画面，如果用户只发送音频，仍然会占用画面位置。

屏幕分享模板（LayoutParams.Template = 2）

- 适合视频会议和在线教育场景的布局。
- 屏幕分享（或者主讲的摄像头）始终占据屏幕左侧的大画面位置，其他用户依次垂直排列于右侧。
- 需要通过 LayoutParams.MainVideoUserId 和 LayoutParams.MainVideoStreamType 这两个参数来指定左侧主画面的内容。
- 最多两列，每列最多8个小画面。最多支持1个大画面和15个小画面。
- 如果用户只发送音频，仍然会占用画面位置。

画中画模板（LayoutParams.Template = 3）

- 该模板以“一大一小，一前一后”的模式混合房间中的两路画面，即将房间中一路画面铺满整个屏幕，再将一路画面作为小画面悬浮在大画面之上，小画面的位置可以通过参数指定。
- 您可以通过 LayoutParams 中的 MainVideoUserId 和 MainVideoStreamType 参数指定大画面这一路的用户 ID 和流类型。
- 您可以通过 LayoutParams 中的 SmallVideoLayoutParams 参数指定小画面这一路的用户 ID、流类型和布局位置等信息。
- 场景示例1：在线教育场景中，混合老师端摄像头（通常作为小画面）和老师端屏幕（通常作为大画面）两路画面，并混合课堂中学生的声音。
- 场景示例2：1v1 视频通话场景中，混合远端用户画面（通常作为大画面）和本地用户画面（通常作为小画面）。

自定义模板（LayoutParams.Template = 4）

- 适用于需要自定义排布各路画面位置的场景，您可以通过 LayoutParams 中的 PresetLayoutConfig 参数（这是一个数组），预先设置各路画面的位置。
- 您可以不指定 PresetLayoutConfig 参数中的 UserId 参数，排版引擎会根据进房的先后顺序，将进房的用户依次分配到 PresetLayoutConfig 数组中指定的各个位置上。
- 如果 PresetLayoutConfig 数组中的某一个被指定了 UserId 参数，则排版引擎会预先给指定的用户预留好他/她在画面中的位置。
- 如果用户只上行音频，不上行视频，该用户依然会占用画面位置。
- 当 PresetLayoutConfig 数组中预设的位置被用完后，排版引擎将不再混合其它用户的画面和声音。

> ⚠ 注意：
> 云端混流服务最多支持同时混合16路音视频流，如果用户只有音频也会被算作一路。

步骤2：设置混流编码参数（必需）

通过 StartMCUMixTranscode 中的 EncodeParams 参数，可以设置混流编码参数：

名称	描述	推荐值
AudioSampleRate	混流-输出流音频采样率	48000
AudioBitrate	混流-输出流音频码率，单位 kbps	64
AudioChannels	混流-输出流音频声道数	2
VideoWidth	混流-输出流宽，音视频输出时必填	自定义
VideoHeight	混流-输出流高，音视频输出时必填	自定义
VideoBitrate	混流-输出流码率，单位 kbps，音视频输出时必填	自定义
VideoFramerate	混流-输出流帧率，音视频输出时必填	15
VideoGop	混流-输出流 GOP，音视频输出时必填	3
BackgroundColor	混流-输出流背景色	自定义

步骤3：指定混合后的 streamID（必需）

- **OutputParams.StreamId**
  通过该参数您可以指定混合后的音视频流在直播 CDN 上的 streamID。不过只有在您已经开通了直播服务并配置了播放域名的情况下，才能通过 CDN 正常观看这条直播流。
- **OutputParams.PureAudioStream**
  如果您只希望做纯音频直播，可以设置 OutputParams.PureAudioStream 参数为 1，代表仅把混音后的音频数据流转发到 CDN 上。

步骤4：设置是否开启云端录制（可选）

- **OutputParams.RecordId**
  该参数用于指定是否启动 云端录制，如果您指定此参数，那么混流后的音视频流会被录制成文件并存储到 云点播 中。录制下来的文件会按照 OutputParams.RecordId_

开始时间_结束时间 的格式命名，例如：file001_2020-02-16-12-12-12_2020-02-16-13-13-13。

- **OutputParams.RecordAudioOnly**
  如果您只希望录制音频而不需要视频内容，可以设置 OutputParams.RecordAudioOnly 参数为1，表示仅录制 MP3 格式的文件。

结束混流

由您的服务器调用 REST API StopMCUMixTranscode 即可结束混流。

方案二：客户端 SDK API 混流方案

使用音视频通话 TRTC SDK 发起混流指令非常简单，只需调用各个平台的 setMixTranscodingConfig() API 即可，目前 SDK 提供了 4 种常用的混流方案：

参数项	纯音频模式（PureAudio）	预排版模式（PresetLayout）	屏幕分享模式（ScreenSharing）	全手动模式（Manual）
调用次数	只需调用一次接口	只需调用一次接口	只需调用一次接口	以下场景需调用混流接口： • 有连麦者加入时 • 有连麦者离开时 • 连麦者开关摄像头时 • 连麦者开关麦克风时
混合内容	只混合音频	自定义设置各路内容	不混合学生端的画面	自定义设置各路内容
audioSampleRate	推荐48000	推荐48000	推荐48000	推荐48000
audioBitrate	推荐64	推荐64	推荐64	推荐64
audioChannels	推荐2	推荐2	推荐2	推荐2
videoWidth	无需设置	不能为0	推荐0	不能为0
videoHeight	无需设置	不能为0	推荐0	不能为0
videoBitrate	无需设置	不能为0	推荐0	不能为0
videoFramerate	无需设置	推荐15	推荐15	推荐15
videoGOP	无需设置	推荐3	推荐3	推荐3
mixUsers 数组	无需设置	使用占位符设置	无需设置	使用真实 userId 设置

纯音频模式（PureAudio）

适用场景

纯音频模式适用于语音通话（AudioCall）和语音聊天室（VoiceChatRoom）等纯音频应用场景，该类场景下您可以在调用 SDK 的 enterRoom 接口时进行设定。纯音频模式下，SDK 会自动将房间里的多路音频流混合成一路。

使用步骤

1. 在调用 enterRoom() 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 TRTCAppSceneAudioCall 或 TRTCAppSceneVoiceChatRoom，明确当前房间中没有视频且只有音频。
2. 开启 旁路直播，并设定 TRTCParams 中的 streamId 参数，指定 MCU 输出的混合音频流的去处。
3. 调用 startLocalAudio() 开启本地音频采集和音频上行。

> 说明：
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音频上行才能构成混流的前提条件。

4. 调用 setMixTranscodingConfig() 接口启动云端混流，需要您在调用时将 TRTCTranscodingConfig 中的 mode 参数设定为 **TRTCTranscodingConfigMode_Template_PureAudio**，并指定 audioSampleRate、audioBitrate 和 audioChannels 等关乎音频输出质量的参数。
5. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。

> 注意：
> 纯音频模式下 setMixTranscodingConfig() 接口无需多次调用，在进房成功并开启本地音频上行后调用一次即可。

预排版模式（PresetLayout）

适用场景

预排版模式适用于视频通话（VideoCall）和互动直播（LIVE）等音频和视频皆有的应用场景，该类场景下您可以在调用 SDK 的 enterRoom 接口时进行设定。
预排版模式下，SDK 会自动按照您预先设定各路画面的排版规则将房间里的多路音频流混合成一路。

使用步骤

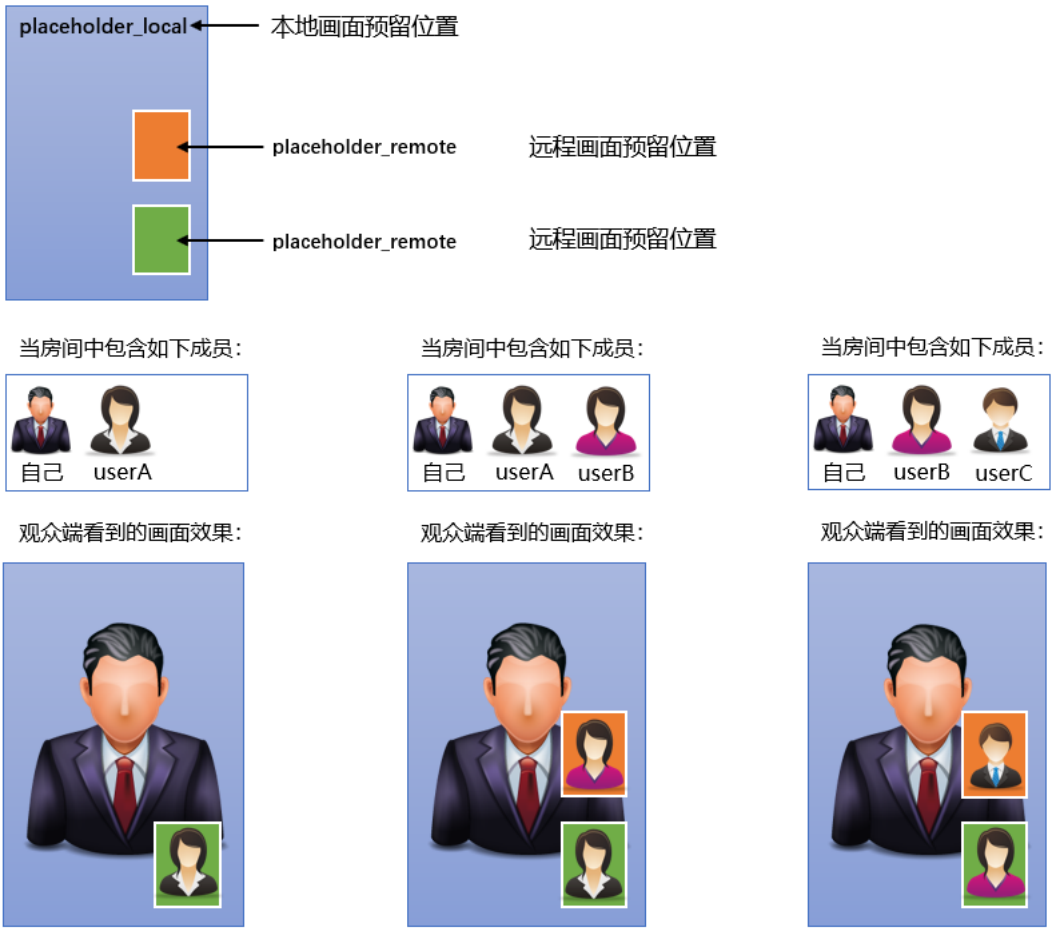
1. 在调用 enterRoom() 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 TRTCAppSceneVideoCall 或 TRTCAppSceneLIVE。
2. 开启 旁路直播，并设定 TRTCParams 中的 streamId 参数，指定 MCU 输出的混合音频流的去处。
3. 调用 startLocalPreview() 和 startLocalAudio() 开启本地的音视频上行。

> 说明：
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音视频上行才能构成混流的前提条件。

4. 调用 setMixTranscodingConfig() 接口启动云端混流，需要您在调用时将 TRTCTranscodingConfig 中的 mode 参数设定为 **TRTCTranscodingConfigMode_Template_PresetLayout**，并指定 audioSampleRate、audioBitrate 和 audioChannels 等关乎音频输出质量的参数，以及 videoWidth、videoHeight、videoBitrate、videoFramerate 等关乎视频输出质量的参数。
5. 组装 mixUser 参数，预排版模式下 mixUser 中的 userId 参数请使用 **$PLACE_HOLDER_REMOTE$**、**$PLACE_HOLDER_LOCAL_MAIN$** 以及 **$PLACE_HOLDER_LOCAL_SUB$** 这三个占位字符串，其含义如下表所示：

占位符	含义	是否支持多个
$PLACE_HOLDER_LOCAL_MAIN$	指代本地摄像头这一路	不支持
$PLACE_HOLDER_LOCAL_SUB$	指代本地屏幕分享这一路（只有画面）	不支持
$PLACE_HOLDER_REMOTE$	指代远端连麦者，可以同时设置多个	支持

6. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。



示例代码

您可根据下面的示例代码实现“一大二小，上下叠加”的混合效果：

iOS

```
TRTCTranscodingConfig *config = [[TRTCTranscodingConfig alloc] init];
// 设置分辨率为720 × 1280, 码率为1500kbps，帧率为20FPS
config.videoWidth = 720;
config.videoHeight = 1280;
config.videoBitrate = 1500;
config.videoFramerate = 20;
config.videoGOP = 2;
config.audioSampleRate = 48000;
config.audioBitrate = 64;
config.audioChannels = 2;
// 采用预排版模式
config.mode = TRTCTranscodingConfigMode_Template_PresetLayout;

NSMutableArray *mixUsers = [NSMutableArray new];
// 主播摄像头的画面位置
TRTCMixUser* local = [TRTCMixUser new];
local.userId = @"$PLACE_HOLDER_LOCAL_MAIN$";
local.zOrder = 0; // zOrder 为0代表主播画面位于最底层
local.rect = CGRectMake(0, 0, videoWidth, videoHeight);
local.roomID = nil; // 本地用户不用填写 roomID，远程需要
[mixUsers addObject:local];

// 连麦者的画面位置
TRTCMixUser* remote1 = [TRTCMixUser new];
remote1.userId = @"$PLACE_HOLDER_REMOTE$";
```

```
remote1.zOrder = 1;
remote1.rect = CGRectMake(400, 800, 180, 240); //仅供参考
remote1.roomID = @"97392"; // 本地用户不用填写 roomID，远程需要
[mixUsers addObject:remote1];

// 连麦者的画面位置
TRTCMixUser* remote2 = [TRTCMixUser new];
remote2.userId = @"$PLACE_HOLDER_REMOTE$";
remote2.zOrder = 1;
remote2.rect = CGRectMake(400, 500, 180, 240); //仅供参考
remote2.roomID = @"97392"; // 本地用户不用填写 roomID，远程需要
[mixUsers addObject:remote2];

config.mixUsers = mixUsers;
// 发起云端混流
[_trtc setMixTranscodingConfig:config];
```

Android

```
TRTCCloudDef.TRTCTranscodingConfig config = new TRTCCloudDef.TRTCTranscodingConfig();
// 设置分辨率为720 × 1280, 码率为1500kbps，帧率为20FPS
config.videoWidth = 720;
config.videoHeight = 1280;
config.videoBitrate = 1500;
config.videoFramerate = 20;
config.videoGOP = 2;
config.audioSampleRate = 48000;
config.audioBitrate = 64;
config.audioChannels = 2;

// 采用预排版模式
config.mode = TRTCCloudDef.TRTC_TranscodingConfigMode_Template_PresetLayout;
config.mixUsers = new ArrayList<>();

// 主播摄像头的画面位置
TRTCCloudDef.TRTCMixUser local = new TRTCCloudDef.TRTCMixUser();
local.userId = "$PLACE_HOLDER_LOCAL_MAIN$";
local.zOrder = 0; // zOrder 为0代表主播画面位于最底层
local.x = 0;
local.y = 0;
local.width = videoWidth;
local.height = videoHeight;
local.roomId = null; // 本地用户不用填写 roomID，远程需要
config.mixUsers.add(local);

// 连麦者的画面位置
TRTCCloudDef.TRTCMixUser remote1 = new TRTCCloudDef.TRTCMixUser();
remote1.userId = "$PLACE_HOLDER_REMOTE$";
remote1.zOrder = 1;
remote1.x = 400; //仅供参考
remote1.y = 800; //仅供参考
remote1.width = 180; //仅供参考
remote1.height = 240; //仅供参考
remote1.roomId = "97392"; // 本地用户不用填写 roomID，远程需要
config.mixUsers.add(remote1);

// 连麦者的画面位置
TRTCCloudDef.TRTCMixUser remote2 = new TRTCCloudDef.TRTCMixUser();
remote2.userId = "$PLACE_HOLDER_REMOTE$";
```

```
remote2.zOrder = 1;
remote1.x = 400; //仅供参考
remote1.y = 500; //仅供参考
remote1.width = 180; //仅供参考
remote1.height = 240; //仅供参考
remote1.roomId = "97393"; // 本地用户不用填写 roomID，远程需要
config.mixUsers.add(remote2);

// 发起云端混流
trtc.setMixTranscodingConfig(config);
```

C++

```
TRTCTranscodingConfig config;
// 设置分辨率为1280 × 720, 码率为1500kbps，帧率为20fps
config.videoWidth = 1280;
config.videoHeight = 720;
config.videoBitrate = 1500;
config.videoFramerate = 20;
config.videoGOP = 2;
config.audioSampleRate = 48000;
config.audioBitrate = 64;
config.audioChannels = 2;

// 采用预排版模式
config.mode == TRTCTranscodingConfigMode_Template_PresetLayout
TRTCMixUser* mixUsersArray = new TRTCMixUser[3];
mixUsersArray[0].userId = "$PLACE_HOLDER_LOCAL_MAIN$";
mixUsersArray[0].zOrder = 0; // zOrder 为0代表主播画面位于最底层
mixUsersArray[0].rect.left = 0;
mixUsersArray[0].rect.top = 0;
mixUsersArray[0].rect.right = videoWidth;
mixUsersArray[0].rect.bottom = videoHeight;
mixUsersArray[0].roomId = nullptr; // 本地用户不用填写 roomID，远程需要

mixUsersArray[1].userId = "$PLACE_HOLDER_REMOTE$";
mixUsersArray[1].zOrder = 1;
mixUsersArray[1].rect.left = 400; //仅供参考
mixUsersArray[1].rect.top = 800; //仅供参考
mixUsersArray[1].rect.right = 180; //仅供参考
mixUsersArray[1].rect.bottom = 240; //仅供参考
mixUsersArray[1].roomId = "97392"; // 本地用户不用填写 roomID，远程需要

mixUsersArray[2].userId = "$PLACE_HOLDER_REMOTE$";
mixUsersArray[2].zOrder = 1;
mixUsersArray[2].rect.left = 400; //仅供参考
mixUsersArray[2].rect.top = 500; //仅供参考
mixUsersArray[2].rect.right = 180; //仅供参考
mixUsersArray[2].rect.bottom = 240; //仅供参考
mixUsersArray[2].roomId = "97393"; // 本地用户不用填写 roomID，远程需要
config.mixUsersArray = mixUsersArray;

// 发起云端混流
trtc->setMixTranscodingConfig(&config);
```

C#

```
TRTCTranscodingConfig config = new TRTCTranscodingConfig();
// 设置分辨率为1280 × 720, 码率为1500kbps，帧率为20fps
```

```
config.videoWidth = 1280;
config.videoHeight = 720;
config.videoBitrate = 1500;
config.videoFramerate = 20;
config.videoGOP = 2;
config.audioSampleRate = 48000;
config.audioBitrate = 64;
config.audioChannels = 2;
config.mode = RTCTranscodingConfigMode.TRTCTranscodingConfigMode_Template_PresetLayout;
TRTCMixUser[] mixUsersArray = new TRTCMixUser[3];

// 主播摄像头的画面位置
TRTCMixUser local = new TRTCMixUser();
local.userId = "$PLACE_HOLDER_LOCAL_MAIN$";
local.zOrder = 0; // zOrder 为0代表主播画面位于最底层
local.roomId = null; // 本地用户不用填写 roomID，远程需要
RECT rtLocal = new RECT() {
left = 0,
top = 0,
right = videoWidth,
bottom = videoHeight
};
local.rect = rtLocal;
mixUsersArray[0] = local;

// 连麦者的画面位置
TRTCMixUser remote1 = new TRTCMixUser();
remote1.userId = "$PLACE_HOLDER_REMOTE$";
remote1.zOrder = 1;
remote1.roomId = “97392”; // 本地用户不用填写 roomID，远程需要
RECT rtRemote1 = new RECT() { //仅供参考
left = 420,
top = 81,
right = 240 + left,
bottom = 240 + top
};
remote1.rect = rtRemote1;
mixUsersArray[1] = remote1;

// 连麦者的画面位置
TRTCMixUser remote2 = new TRTCMixUser();
remote2.userId = "$PLACE_HOLDER_REMOTE$";
remote2.zOrder = 1;
remote2.roomId = “97393”; // 本地用户不用填写 roomID，远程需要
RECT rtRemote2 = new RECT() { //仅供参考
left = 660,
top = 400,
right = 240 + left,
bottom = 240 + top
};
rtRemote2.rect = rtRemote2;
mixUsersArray[2] = remote2;

// 发起云端混流
config.mixUsersArray = mixUsersArray;
trtc.setMixTranscodingConfig(config);
```

Flutter

```
TRTCCloud trtcCloud = await TRTCCloud.sharedInstance();
trtcCloud.setMixTranscodingConfig(TRTCTranscodingConfig(
appId: 1252463788, //仅供参考
bizId: 3891, //仅供参考
// 设置分辨率为720 × 1280, 码率为1500kbps，帧率为20FPS
videoWidth: 720,
videoHeight: 1280,
videoBitrate: 1500,
videoFramerate: 20,
videoGOP: 2,
audioSampleRate: 48000,
audioBitrate: 64,
audioChannels: 2,

// 采用预排版模式
mode: TRTCCloudDef.TRTC_TranscodingConfigMode_Template_PresetLayout,

mixUsers: [
// 主播摄像头的画面位置
TRTCMixUser(
userId: "$PLACE_HOLDER_LOCAL_MAIN$",
roomId: null, // 本地用户不用填写 roomID，远程需要
zOrder: 0, // zOrder 为0代表主播画面位于最底层
x: 0, //仅供参考
y: 0,
streamType: 0,
width: 300,
height: 400),
TRTCMixUser(
userId: '$PLACE_HOLDER_REMOTE$',
roomId: '256', // 本地用户不用填写 roomID，远程需要
zOrder: 1,
x: 100, //仅供参考
y: 100,
streamType: 0,
width: 160,
height: 200)
],
));
```

Web

```
try {
// 预排版模式
const config = {
mode: 'preset-layout',
videoWidth: 720,
videoHeight: 1280,
videoBitrate: 1500,
videoFramerate: 20,
videoGOP: 2,
audioSampleRate: 48000,
audioBitrate: 64,
audioChannels: 2,
// 预设一路本地摄像头、两路远端流的排版位置
mixUsers: [
{
width: 720,
height: 1280,
```

```
locationX: 0,
locationY: 0,
pureAudio: false,
userId: 'jack', // 本地摄像头占位，传入推摄像头的 client userId
zOrder: 1
},
{
width: 180,
height: 240,
locationX: 400,
locationY: 800,
pureAudio: false,
userId: '$PLACE_HOLDER_REMOTE$', // 远端流占位
zOrder: 2
},
{
width: 180,
height: 240,
locationX: 400,
locationY: 500,
pureAudio: false,
userId: '$PLACE_HOLDER_REMOTE$', // 远端流占位
zOrder: 2
}
];
}
await client.startMixTranscode(config);
} catch (error) {
console.error('startMixTranscode failed ', error);
}
```

⚠ 注意：

- 预排版模式下 setMixTranscodingConfig() 接口无需多次调用，在进房成功并开启本地音频上行后调用一次即可。
- Web 端接口命名与其他端稍有差异，详情请参见 Client.startMixTranscode()。

屏幕分享模式（ScreenSharing）

适用场景

屏幕分享模式适用于在线教育和互动课堂等场景，该类场景下您可以在调用 SDK 的 enterRoom 接口时将 AppScene 参数设定为 TRTCAppSceneLIVE 。
屏幕分享模式下，SDK 会先根据您所选定的目标分辨率构建一张画布。当老师未开启屏幕分享时，SDK 会将摄像头画面等比例拉伸绘制到该画布上；当老师开启屏幕分享后，SDK 会将屏幕分享画面绘制到同样的画布上。通过构建画布可以确保混流模块的输出分辨率一致，防止录制和网页观看的视频兼容性问题（普通播放器不支持分辨率会变化的视频）。

使用步骤

1. 在调用 enterRoom() 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 TRTCAppSceneLIVE。
2. 开启 旁路直播，并设定 TRTCParams 中的 streamId 参数，指定 MCU 输出的混合音视频流的去处。
3. 调用 startLocalPreview() 和 startLocalAudio() 开启本地的音视频上行。

> 说明：
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音视频上行才能构成混流的前提条件。

4. 调用 setMixTranscodingConfig() 接口启动云端混流，需要您在调用时将 TRTCTranscodingConfig 中的 mode 参数设定为 **TRTCTranscodingConfigMode_Template_ScreenSharing**，并指定 audioSampleRate、audioBitrate 和 audioChannels 等关乎音频输出质量的参数，以及 videoWidth、videoHeight、videoBitrate、videoFramerate 等关乎视频输出质量的参数。

> 说明：

> 若将 videoWidth 和 videoHeight 参数均指定为0，SDK 会自动根据用户当前屏幕的宽高比计算出一个合适的分辨率。

5. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。

> 注意：
>
> - 屏幕分享模式仅支持 Windows 和 Mac 平台。
> - 屏幕分享模式下 setMixTranscodingConfig() 接口无需多次调用，在进房成功并开启本地音频上行后调用一次即可。
> - 由于教学模式下的视频内容以屏幕分享为主，同时传输摄像头画面和屏幕分享画面非常浪费带宽。建议直接将摄像头画面和学生的画面通过 setLocalVideoRenderCallback() 和 setRemoteVideoRenderCallback() 接口自绘到当前屏幕上。
> - 通过将 TRTCTranscodingConfig 中的 videoWidth 和 videoHeight 参数均指定为 0，可以让 SDK 智能选择输出分辨率。如果老师当前屏幕宽度小于1920px，SDK 会使用老师当前屏幕的实际分辨率；如果老师当前屏幕宽度大于1920px，SDK 会根据当前屏幕宽高比，选择 1920 × 1080（16:9）、1920 × 1200（16:10）或1920 × 1440（4:3）。

全手动模式（Manual）

适用场景

全手动模式适合于上述自动模式均不适用的场景，全手动的灵活性最高，可以自由组合出各种混流方案，但易用性最差。
全手动模式下，您需要设置 TRTCTranscodingConfig 中的所有参数，并需要监听 TRTCCloudDelegate 中的 onUserVideoAvailable() 和 onUserAudioAvailable() 回调，以便根据当前房间中各个上麦用户的音视频状态不断地调整 mixUsers 参数，否则会导致混流失败。

使用步骤

1. 在调用 enterRoom() 函数进入房间时，根据您的业务需要，设定 AppScene 参数。
2. 开启 旁路直播，并设定 TRTCParams 中的 streamId 参数，指定 MCU 输出的混合音视频流的去处。
3. 根据您的业务需要，调用 startLocalAudio() 开启本地的音频上行（或同时调用 startLocalPreview() 开启视频上行）。

> 说明：
> 由于云端混流的本质是将多路流混合到当前（即发起混流指令的）用户所对应的音视频流上，因此当前用户本身必须有音视频上行才能构成混流的前提条件。

4. 调用 setMixTranscodingConfig() 接口启动云端混流，需要您在调用时将 TRTCTranscodingConfig 中的 mode 参数设定为 **TRTCTranscodingConfigMode_Manual**，并指定 audioSampleRate、audioBitrate 和 audioChannels 等关乎音频输出质量的参数。如果您的业务场景中也包含视频，需同时设置 videoWidth、videoHeight、videoBitrate、videoFramerate 等关乎视频输出质量的参数。
5. 监听 TRTCCloudDelegate 中的 onUserVideoAvailable() 和 onUserAudioAvailable() 回调，并根据需要指定 **mixUsers** 参数。

> 说明：
> 与预排版（PresetLayout）模式不同，Manual 需要您指定每一个 mixUser 中的 userId 参数为真实的连麦者 ID，并且也要根据该连麦者是否开启了视频，如实设定 mixUser 中的 pureAudio 参数。

6. 经过上述步骤，当前用户的旁路音频流中就会自动混合房间中其他用户的声音，之后您可以参考文档 CDN 直播观看 配置播放域名进行直播观看，也可以参考文档 云端录制 录制混合后的音频流。

> ⚠ 注意：
> 全手动模式下，您需要实时监听房间中连麦者的上麦下麦动作，并根据连麦者的人数和音视频状态，多次调用 setMixTranscodingConfig() 接口。

相关费用

费用的计算

云端混流转码需要对输入 MCU 集群的音视频流进行解码后重新编码输出，将产生额外的服务成本，因此 TRTC 将向使用 MCU 集群进行云端混流转码的用户收取额外的增值费用。云端混流转码费用根据**转码输出的分辨率大小**和**转码时长**进行计费，转码输出的分辨率越高、转码输出的时间越长，费用越高。详情请参见 云端混流转码计费说明。

费用的节约

- 在**基于服务端 REST API 混流方案下，要停止混流**，需要满足如下条件之一：
  - 房间里的所有用户（包括主播和观众）都退出了房间。
  - 调用 REST API StopMCUMixTranscode 主动停止混流。
- 在**基于客户端 SDK API 混流方案下，要停止混流**，需要满足如下条件之一：
  - 发起混流（即调用了客户端 API setMixTranscodingConfig）的主播退出了房间。
  - 调用 setMixTranscodingConfig 并将参数设置为 nil/null 主动停止混流。

在其他情况下，TRTC 云端都将会尽力持续保持混流状态。因此，为避免产生预期之外的混流费用，请在您不需要混流的时候尽早通过上述方法结束云端混流。

云端录制与回放

最近更新时间：2021-12-23 17:06:52

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

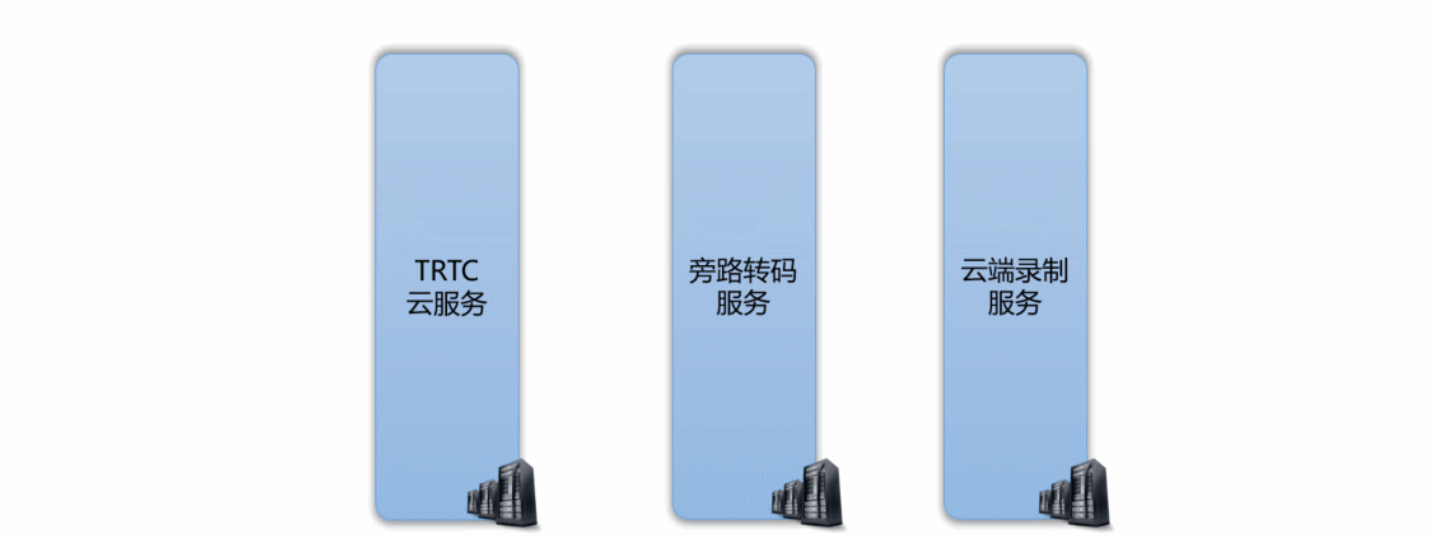
版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

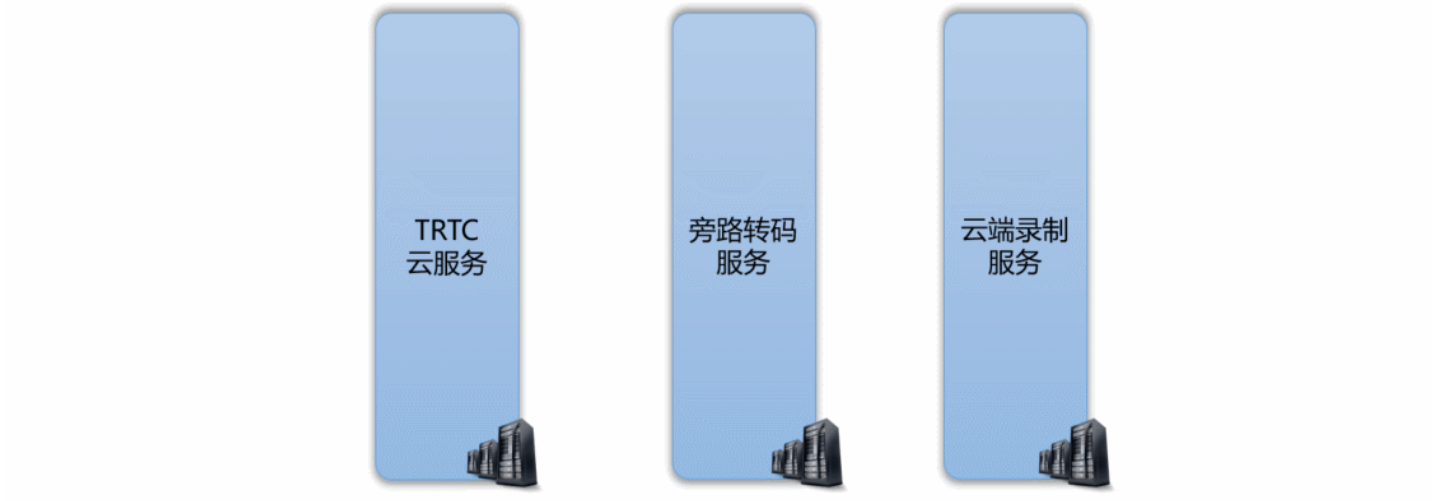
适用场景

在远程教育、秀场直播、视频会议、远程定损、金融双录、在线医疗等应用场景中，考虑取证、质检、审核、存档和回放等需求，常需要将整个视频通话或互动直播过程录制和存储下来。

音视频通话TRTC 的云端录制，可以将房间中的每一个用户的音视频流都录制成一个独立的文件：



也可以将房间中的多路音视频先进行 云端混流，再将混合后的音视频流录制成一个文件：



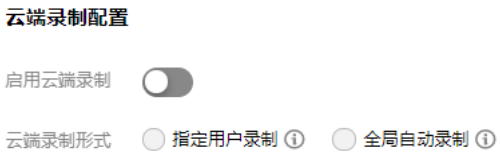
控制台指引

开通录制服务

1. 登录实时音视频控制台，在左侧导航栏选择 **应用管理**。
2. 单击目标应用所在行的**功能配置**，进入功能配置页卡。如果您还没有创建过应用，可以单击**创建应用**，填写应用名称，单击**确定**创建一个新的应用。
3. 单击**启用云端录制**右侧的  ，会弹出云端录制的设置页面。

选择录制形式

音视频通话TRTC 的云端录制服务提供了两种不同的录制形式：“全局自动录制”和“指定用户录制”：



- **全局自动录制**
  每一个 TRTC 房间中的每个用户的音视频上行流都会被自动录制下来，录制任务的启动和停止都是自动的，不需要您额外操心，比较简单和易用。具体的使用方法请阅读 方案一：全局自动录制。
- **指定用户录制**
  您可以指定只录制一部分用户的音视频流，这需要您通过客户端的 SDK API 或者服务端的 REST API 进行控制，需要额外的开发工作量。具体的使用方法请阅读 方案二：指定用户录制（SDK API）和 方案三：指定用户录制（REST API）。

选择文件格式

云端录制支持 HLS、MP4、FLV 和 AAC 四种不同的文件格式，我们以表格的形式列出四种不同格式的差异和适用场景，您可以结合自身业务的需要进行选择：

参数	参数说明
文件类型	支持以下文件类型： • HLS：该文件类型支持绝大多数浏览器在线播放，适合视频回放场景。选择该文件类型时，支持断点续录且不限制单个文件最大时长。 • FLV：该文件类型不支持在浏览器在线播放，但该格式简单容错性好。如果无需将录制文件存储在云点播平台，可以选择该文件类型，录制完成后立刻下载录制文件并删除源文件。 • MP4：该文件类型支持在 Web 浏览器在线播放，但此格式容错率差，视频通话过程中的任何丢包都会影响最终文件的播放质量。 • AAC：如果只需录制音频，可以选择该文件类型。
单个文件的最大时长（分钟）	• 根据实际业务需求设置单个视频文件的最大时长限制，超过长度限制后系统将会自动拆分视频文件。单位为分钟，取值范围1 – 120。 • 当**文件类型**设置为**HLS**时，不限制单个文件的最大时长，即该参数无效。
文件保存时长（天）	根据实际业务需求设置视频文件存储在云点播平台的天数。单位为天，取值范围0 – 1500，到期后文件将被点播平台自动删除且无法找回， 0表示永久存储。
续录超时时长（秒）	• 默认情况下，若通话（或直播）过程因网络波动或其他原因被打断，录制文件会被切断成多个文件。 • 如果需要实现“一次通话（或直播）只产生一个回放链接”，可以根据实际情况设置续录超时时长，当打断间隔不超过设定的续录超时时长时，一次通话（或直播）只会生成一个文件，但需要等待续录时间超时后才能收到录制文件。 • 单位为秒，取值范围1 – 1800，0表示断点后不续录。

> **说明：**
> HLS 支持最长三十分钟的续录，可以做到“一堂课只产生一个回放链接”，且支持绝大多数浏览器的在线观看，非常适合在线教育场景中的视频回放场景。

选择存储位置

TRTC 云端录制文件会默认存储于腾讯云点播服务上，如果您的项目中多个业务公用一个腾讯云点播账号，可能会有录制文件隔离的需求。您可以通过腾讯云点播的“子应用”能力，将 TRTC 的录制文件与其他业务区分开。

- **什么是点播主应用和子应用?**
  主应用和子应用是云点播上的一种资源划分的方式。主应用相当于云点播的主账号，子应用可以创建多个，每一个子应用相当于主账号下面的一个子账号，拥有独立的资源管理，存储区域可以跟其他的子应用相互隔离。

- **如何开启点播服务的子应用?**
  您可以根据文档 “如何开启点播子应用” 添加新的子应用，这一步需要您跳转到点播 控制台 中进行操作。

设置录制回调

- **录制回调地址：**
  如果您需要实时接收到新文件的 落地通知，可在此处填写您的服务器上用于接收录制文件的回调地址，该地址需符合 HTTP（或 HTTPS）协议。当新的录制文件生成后，腾讯云会通过该地址向您的服务器发送通知。

- **录制回调密钥：**
  回调密钥用于在接收回调事件时生成签名鉴权，该密钥需由大小写字母及数字组成，且不得超过32个字符。相关使用请参见 录制事件参数说明。

> 说明：
> 详细的录制回调接收和解读方案请参考文档后半部分的：接收录制文件。

录制控制方案

音视频通话 TRTC 提供了三种云端录制的控制方案，分别是 全局自动录制、指定用户录制（由 SDK API 控制）和 指定用户录制（由 REST API 控制）。对于其中的每一种方案，我们都会详细介绍：

- 如何在控制台中设定使用该方案?
- 如何开始录制任务?
- 如何结束录制任务?
- 如何将房间中的多路画面混合成一路?
- 录制下来的文件会以什么格式命名?
- 支持该种方案的平台有哪些?

方案一：全局自动录制

- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“全局自动录制”。
- **录制任务的开始**
  TRTC 房间中的每一个用户的音视频流都会被自动录制成文件，无需您的额外操作。
- **录制任务的结束**
  自动停止。即每个主播在停止音视频上行后，该主播的云端录制即会自行停止。如果您在 选择文件格式 时设置了“续录时间”，则需要等待续录时间超时后才能收到录制文件。
- **多路画面的混合**
  全局自动录制模式下的云端混流有两种方案，即“服务端 REST API 方案” 和 “客户端 SDK API 方案”，两套方案请勿混合使用：
  - 服务端 REST API 混流方案：需要由您的服务器发起 API 调用，不受客户端平台版本的限制。
  - 客户端 SDK API 混流方案：可以直接在客户端发起混流，目前支持 iOS、Android、Windows、Mac 和 Electron 等平台，暂不支持微信小程序和 Web 浏览器。
- **录制文件的命名**
  - 如果主播在进房时指定了 userDefineRecordId 参数，则录制文件会以 userDefineRecordId_streamType_开始时间_结束时间 来命名（streamType 有 main 和 aux 两个取值，main 代表主路，aux 代表辅路，辅路通常被用作屏幕分享）；
  - 如果主播在进房时没有指定 userDefineRecordId 参数，但指定了 streamId 参数，则录制文件会以 streamId_开始时间_结束时间 来命名；

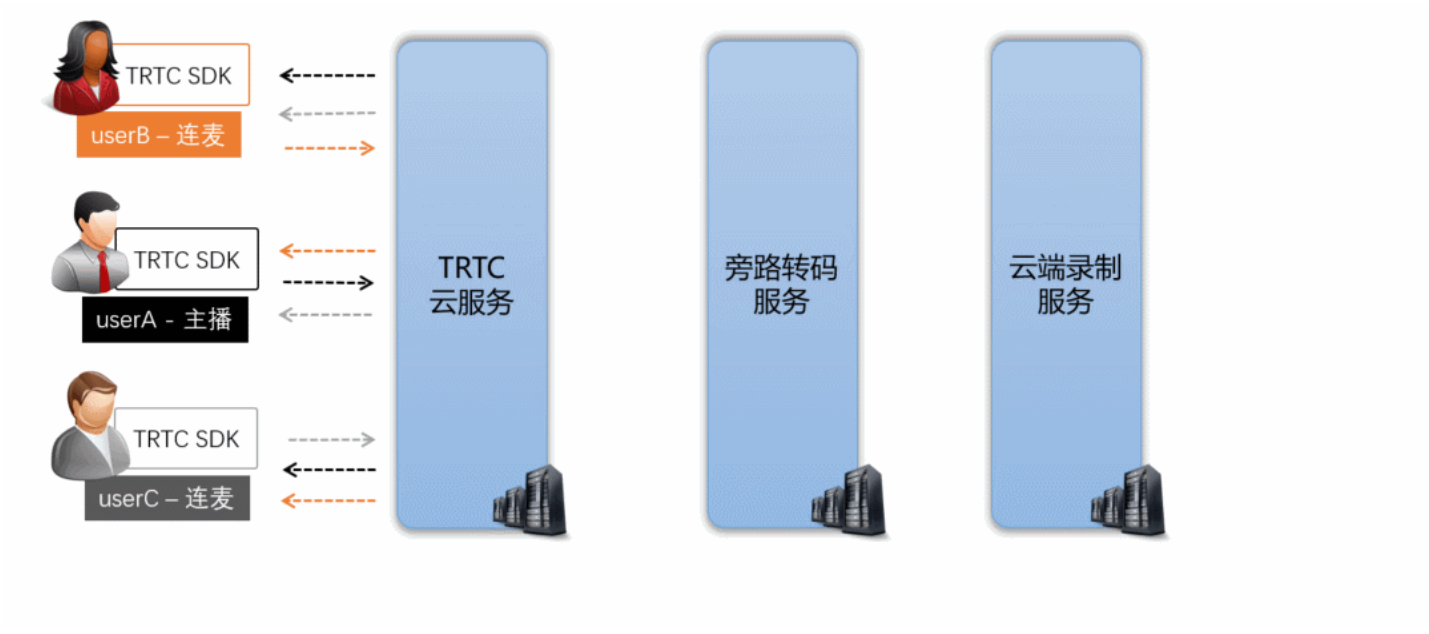
- 如果主播在进房时既没有指定 userDefineRecordId 参数，也没有指定 streamId 参数，则录制文件会以 sdkappid_roomid_userid_streamType_开始时间_结束时间 来命名（streamType 有 main 和 aux 两个取值，main 代表主路，aux 代表辅路，辅路通常被用作屏幕分享）。

- **已经支持的平台**
  由您的服务端控制，不受客户端平台限制。

方案二：指定用户录制（SDK API）

通过调用音视频通话 TRTC SDK 提供的一些 API 接口和参数，即可实现云端混流、云端录制和旁路直播三个功能：

云端能力	如何开始?	如何停止?
云端录制	进房时指定参数 TRTCParams 中的 userDefineRecordId 字段	主播退房时自动停止
云端混流	调用 SDK API setMixTranscodingConfig() 启动云端混流	发起混流的主播退房后，混流会自动停止，或中途调用 setMixTranscodingConfig() 并将参数设置为 null/nil 手动停止
旁路直播	进房时指定参数 TRTCParams 中的 streamId 字段	主播退房时自动停止



- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“指定用户录制”。

- **录制任务的开始**
  主播在进房时指定进房参数 TRTCParams 中的 userDefineRecordId 字段，之后该主播的上行音视频数据即会被云端录制下来，不指定该参数的主播不会触发录制任务。

```
// 示例代码：指定录制用户 rexchang 的音视频流，文件 id 为 1001_rexchang
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
TRTCParams *param = [[TRTCParams alloc] init];
param.sdkAppId = 1400000123; // TRTC 的 SDKAppID，创建应用后可获得
param.roomId = 1001; // 房间号
param.userId = @"rexchang"; // 用户名
param.userSig = @"xxxxxxxx"; // 登录签名
param.role = TRTCRoleAnchor; // 角色：主播
param.userDefineRecordId = @"1001_rexchang"; // 录制 ID，即指定开启该用户的录制。
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE]; // 请使用 LIVE 模式
```

- **录制任务的结束**
  自动停止，当进房时指定 userDefineRecordId 参数的主播在停止音视频上行后，云端录制会自行停止。如果您在选择文件格式时设置了“续录时间”，则需要等待续

录时间超时后才能收到录制文件。

- **多路画面的混合**
  您可以通过调用 SDK API setMixTranscodingConfig() 将房间中其它用户的画面和声音混合到当前用户的这一路音视频流上。关于这一部分详细介绍，可以阅读文档：云端混流转码。

  > ⚠ 注意：
  > 在一个 TRTC 房间中，只由一个主播（推荐是开播的主播）来调用 setMixTranscodingConfig 即可，多个主播调用可能会出现状态混乱的错误。

- **录制文件的命名**
  录制文件会以 userDefineRecordId_开始时间_结束时间 的格式来命名。

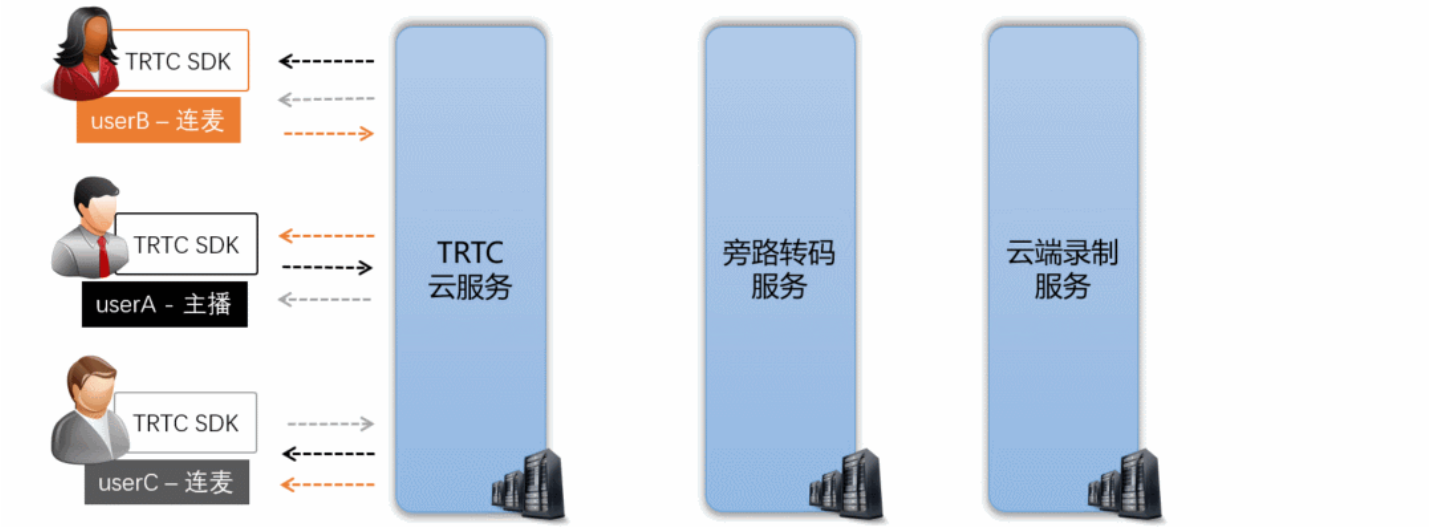
- **已经支持的平台**
  支持 iOS、Android、Windows、Mac、Electron 等终端发起录制控制，暂不支持由 Web 浏览器和微信小程序端发起控制。

方案三：指定用户录制（REST API）

音视频通话TRTC 的服务端提供了一对 REST API（ StartMCUMixTranscode 和 StopMCUMixTranscode ）用于实现云端混流、云端录制和旁路直播三个功能：

云端能力	如何开始?	如何停止?
云端录制	调用 StartMCUMixTranscode 时指定 OutputParams.RecordId 参数即可开始录制	自动停止，或中途调用 StopMCUMixTranscode 停止
云端混流	调用 StartMCUMixTranscode 时指定 LayoutParams 参数可设置布局模板和布局参数	所有用户退房后自动停止，或中途调用 StopMCUMixTranscode 手动停止
旁路直播	调用 StartMCUMixTranscode 时指定 OutputParams.StreamId 参数可启动到 CDN 的旁路直播	自动停止，或中途调用 StopMCUMixTranscode 停止

> ⓘ 说明：
> 由于这对 REST API 控制的是 TRTC 云服务中的核心混流模块 MCU，并将 MCU 混流后的结果输送给录制系统和直播 CDN，因此 API 的名字被称为 Start/StopMCUMixTranscode 。因此，从功能角度上来说， Start/StopMCUMixTranscode 不仅仅可以实现混流的功能，也可以实现云端录制和旁路直播 CDN 的功能。



- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“指定用户录制”。

- **录制任务的开始**
  由您的服务器调用 StartMCUMixTranscode ，并指定 OutputParams.RecordId 参数即可启动混流和录制。

```
// 代码示例：通过 REST API 启动云端混流和云端录制任务
https://trtc.tencentcloudapi.com/?Action=StartMCUMixTranscode
&SdkAppId=1400000123
&RoomId=1001
&OutputParams.RecordId=1400000123_room1001
&OutputParams.RecordAudioOnly=0
&EncodeParams.VideoWidth=1280
&EncodeParams.VideoHeight=720
&EncodeParams.VideoBitrate=1560
&EncodeParams.VideoFramerate=15
&EncodeParams.VideoGop=3
&EncodeParams.BackgroundColor=0
&EncodeParams.AudioSampleRate=48000
&EncodeParams.AudioBitrate=64
&EncodeParams.AudioChannels=2
&LayoutParams.Template=1
&<公共请求参数>
```

> ⚠ 注意：
> - 该 REST API 需要在房间中至少有一个用户进房（enterRoom）成功后才有效。
> - 使用 REST API 不支持单流录制，如果您需要单流录制，请选择 方案一 或者 方案二。

- **录制任务的结束**
  自动停止，您也可以中途调用 StopMCUMixTranscode 停止混流和录制任务。

- **多路画面的混合**
  在调用 StartMCUMixTranscode 时同时指定 LayoutParams 参数即可实现云端混流。该 API 支持在整个直播期间多次调用，即您可以根据需要修改 LayoutParams 参数并再次调用该 API 来调整混合画面的布局。但需要注意的是，您需要保持参数 OutputParams.RecordId 和 OutputParams.StreamId 在多次调用中的一致性，否则会导致断流并产生多个录制文件。

> ⓘ 说明：
> 关于云端混流的详细介绍，具体请参见 云端混流转码。

- **录制文件的命名**
  录制文件会以调用 StartMCUMixTranscode 时指定的 OutputParams.RecordId 参数来命名，命名格式为 OutputParams.RecordId_开始时间_结束时间 。

- **已经支持的平台**
  由您的服务端控制，不受客户端平台的限制。

查找录制文件

在开启录制功能以后，TRTC 系统中录制下来的文件就能在腾讯云点播服务中找到。您可以直接在云点播控制台手动查找，也可以由您的后台服务器使用 REST API 进行定时筛选：

方式一：在点播控制台手动查找

1. 登录 云点播控制台，在左侧导航栏选择**媒资管理**。
2. 单击列表上方的**前缀搜索**，选择**前缀搜索**，在搜索框输入关键词，例如1400000123_1001_rexchang_main，单击🔍，将展示视频名称前缀相匹配的视频文件。
3. 您可以根据创建时间筛选所需的目标文件。

方式二：通过点播 REST API 查找

腾讯云点播系统提供了一系列 REST API 来管理其上的音视频文件，您可以通过 搜索媒体信息 这个 REST API 来查询您在点播系统上的文件。您可以通过请求参数表中的 Text 参数进行模糊匹配，也可以根据 StreamId 参数进行精准查找。
REST 请求示例：

```
https://vod.tencentcloudapi.com/?Action=SearchMedia
&StreamId=stream1001
&Sort.Field=CreateTime
&Sort.Order=Desc
&<公共请求参数>
```

接收录制文件

除了 查找录制文件，您还可以通过配置回调地址，让腾讯云主动把新录制文件的消息推送给您的服务器。
房间里的最后一路音视频流退出后，腾讯云会结束录制并将文件转存到云点播平台，该过程大约默认需要30秒至2分钟（若您设置了续录时间为300秒，则等待时间将在默认基础上叠加300秒）。转存完成后，腾讯云会通过您在 设置录制回调 中设置的回调地址（HTTP/HTTPS）向您的服务器发送通知。

腾讯云会将录制和录制相关的事件都通过您设置的回调地址推送给您的服务器，回调消息示例如下图所示：


```
{
    "app": "8888.livepush.myqcloud.com",
    "appid": 1234567890,
    "appname": "trtc_1400000123",
    "channel_id": "1400000123_1001_rexchang_main",
    "duration": 39,
    "end_time": 1581926501,
    "end_time_usec": 817545,
    "event_type": 100,
    "file_format": "mp4",
    "file_id": "5285890798833426017",
    "file_size": 3337482,
    "media_start_time": 0,
    "record_bps": 0,
    "record_file_id": "5285890798833426017",
    "start_time": 1581926464,
    "start_time_usec": 588890,
    "stream_id": "1400000123_1001_rexchang_main",
    "stream_param": "txSecret=ecd19683f6cf1a7615f13670e0834de1&txTime=70d4653d&from=interactive&client_business_type=0&
                     sdkappid=1400000123&sdkapptype=1&groupid=1001&userid=cmV4Y2hhbmc=&ts=5e4a483c&userdefinerecordid=my
                     testfile&tinyid=1441152145405491 89&roomid=2294546&cliRecoId=0&trtcclientip=222.248.237.246&useMixPl
                     ayer=1",
    "task_id": "1802569503626226707",
    "video_id": "1234567890_46ac1271218249118752d57423120300",
    "video_url": "http://1234567890.vod2.myqcloud.com/5022b150vodcq1234567890/39e578f1228079589883342601 7/f0.mp4"
}
```


您可以通过下表中的字段来确定当前回调是对应的哪一次通话（或直播）：

序号	字段名	说明
1	event_type	消息类型，当 event_type 为100时，表示该回调消息为录制文件生成的消息。
2	stream_id	即直播 CDN 的 streamId，您可以在进房时通过设置 TRTCParams 中的 streamId 字段指定（推荐），也可以在调用 TRTCCloud 的 startPublishing 接口时通过参数 streamId 来指定。
3	stream_param.userid	用户名的 Base64 编码。
4	stream_param.userdefinerecordid	自定义字段，您可以通过设置 TRTCParams 中的 userDefineRecordId 字段指定。
5	video_url	录制文件的观看地址，可以用于 点播回放。

删除录制文件

腾讯云点播系统提供了一系列 REST API 来管理其上的音视频文件，您可以通过 删除媒体 API 删除某个指定的文件。
REST 请求示例：

```
https://vod.tencentcloudapi.com/?Action=DeleteMedia
&FileId=5285890798866415058 7
```

```
&<公共请求参数>
```

回放录制文件

在线教育等场景中，通常需要在直播结束后多次回放录制文件，以便充分利用教学资源。

选择文件格式（HLS）

在 设置录制格式 中选择文件格式为 HLS。
HLS 支持最长三十分钟的断点续录，可以做到“一场直播（或一堂课）只产生一个回放链接”，且 HLS 文件支持绝大多数浏览器在线播放，非常适合视频回放场景。

获取点播地址（video_url）

在 接收录制文件 时，可以获取回调消息中 video_url 字段，该字段为当前录制文件在腾讯云的点播地址。

对接点播播放器

根据使用平台对接点播播放器，具体操作参考如下：

- iOS 平台
- Android 平台
- Web 浏览器

> 注意：
> 建议使用 专业版 TRTC SDK，专业版集合了 超级播放器（Player+）、移动直播（MLVB）等功能，由于底层模块的高度复用，集成专业版的体积增量要小于同时集成两个独立的 SDK，并且可以避免符号冲突（symbol duplicate）的困扰。

相关费用

云端录制与回放的相关费用包含以下几项，其中录制费用是基础费用，其他费用则会根据您的使用情况而按需收取。

> 说明：
> 本文中的价格为示例，仅供参考。若价格与实际不符，请以 云端录制计费说明、云直播 和 云点播 的定价为准。

录制费用：转码或转封装产生的计算费用

由于录制需要进行音视频流的转码或转封装，会产生服务器计算资源的消耗，因此需要按照录制业务对计算资源的占用成本进行收费。

> 注意：
> - 自2020年7月1日起首次在 TRTC 控制台创建应用的腾讯云账号，使用云端录制功能后产生的录制费用以 云端录制计费说明 为准。
> - 在2020年7月1日之前已经在 TRTC 控制台创建过应用的腾讯云账号，无论是在2020年7月1日之前还是之后创建的应用，使用云端录制功能产生的录制费用均默认继续延用**直播录制**的计费规则。

直播录制计费的计算方法是按照并发录制的路数进行收费，并发数越高录制费用越高，具体计费说明请参见 云直播 > 直播录制。

例如，您目前有1000个主播，如果在晚高峰时，最多同时有500路主播的音视频流需要录制。假设录制单价为30元/路/月，那么总录制费用为 500路 × 30元/路/月 = 15000元/月 。
如果您在 设置录制格式 时同时选择了两种录制文件，录制费用和存储费用都会 × 2，同理，选择三种文件时录制费用和存储费用会 × 3。如非必要，建议只选择需要的一种文件格式，可以大幅节约成本。

存储费用：如将文件存储于腾讯云则会产生该费用

如果录制出的文件要存放于腾讯云，由于存储本身会产生磁盘资源的消耗，因此需要按照存储的资源占用进行收费。存储的时间越久费用也就越高，因此如无特殊需要，您可以将文件的存储时间设置的短一些来节省费用，或者将文件存放在自己的服务器上。存储费用可以选择 视频存储（日结）价格 进行日结计算，也可以购买 存储资源包。

例如，您通过 setVideoEncoderParam() 设置主播的码率（videoBitrate）为1000kbps，录制该主播的直播视频（选择一种文件格式），录制一小时大约会产生一个 (1000 / 8)KBps × 3600秒 = 450000KB = 0.45GB 大小的视频文件，该文件每天产生的存储费用约为 0.45GB × 0.0048 元/GB/日 = 0.00216元 。

观看费用：如将文件用于点播观看则会产生该费用

如果录制出的文件要被用于点播观看，由于观看本身会产生 CDN 流量消耗，因此需要按照点播的价格进行计费，默认按流量收费。观看的人数越多费用越高，观看费用可以选择 视频加速（日结）价格 进行日结计算，也可以购买 流量套餐包。

例如，您通过云端录制产生了一个1GB大小的文件，且有1000位观众从头到尾完整地观看了视频，大约会产生1TB的点播观看流量，那么按照阶梯价格表，1000位观众就会产生 1000 × 1GB × 0.23元/GB = 230元 的费用，按照流量套餐包则是175元。
如果您选择从腾讯云下载文件到您的服务器上，也会产生一次很小的点播流量消耗，并且会在您的月度账单中有所体现。

转码费用：如开启混流录制则会产生该费用

如果您启用了混流录制，由于混流本身需要进行解码和编码，所以还会产生额外的混流转码费用。 混流转码根据分辨率大小和转码时长进行计费，主播用的分辨率越高，连麦时间（通常在连麦场景才需要混流转码）越长，费用越高，具体费用计算可以参考 直播转码。

例如，您通过 setVideoEncoderParam() 设置主播的码率（videoBitrate）为1500kbps，分辨率为720P。如果有一位主播跟观众连麦了一个小时，连麦期间开启了 云端混流，那么产生的转码费用为 0.0325元/分钟 × 60分钟 = 1.95元 。

相关问题

TRTC 如何实现服务端录制?

服务端录制需要使用 Linux SDK。Linux SDK 暂未完全开放，若您需咨询或使用相关服务，请填写 Linux SDK 问卷。我们会在2个-3个工作日内完成评估并反馈结果。

变声和混响
iOS

最近更新时间：2021-08-12 20:00:27

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	✓	✓	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

录制变声混响

```
//获取 recorder 对象
recorder = [TXUGCRecord shareInstance];

// 设置混响
// TXRecordCommon.VIDOE_REVERB_TYPE_0 关闭混响
// TXRecordCommon.VIDOE_REVERB_TYPE_1 KTV
// TXRecordCommon.VIDOE_REVERB_TYPE_2 小房间
// TXRecordCommon.VIDOE_REVERB_TYPE_3 大会堂
// TXRecordCommon.VIDOE_REVERB_TYPE_4 低沉
// TXRecordCommon.VIDOE_REVERB_TYPE_5 洪亮
// TXRecordCommon.VIDOE_REVERB_TYPE_6 金属声
// TXRecordCommon.VIDOE_REVERB_TYPE_7 磁性
[recorder setReverbType:VIDOE_REVERB_TYPE_1];

// 设置变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0 关闭变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1 熊孩子
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2 萝莉
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3 大叔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4 重金属
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6 外国人
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7 困兽
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8 死肥仔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9 强电流
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10 重机械
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11 空灵
[record setVoiceChangerType:VIDOE_VOICECHANGER_TYPE_1];
```

说明：
变声混响只针对录制人声有效，针对 BGM 无效。

Android

最近更新时间：2021-08-12 20:00:37

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	✓	✓	✓	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

录制变声混响

```
// 设置混响
// TXRecordCommon.VIDOE_REVERB_TYPE_0 关闭混响
// TXRecordCommon.VIDOE_REVERB_TYPE_1 KTV
// TXRecordCommon.VIDOE_REVERB_TYPE_2 小房间
// TXRecordCommon.VIDOE_REVERB_TYPE_3 大会堂
// TXRecordCommon.VIDOE_REVERB_TYPE_4 低沉
// TXRecordCommon.VIDOE_REVERB_TYPE_5 洪亮
// TXRecordCommon.VIDOE_REVERB_TYPE_6 金属声
// TXRecordCommon.VIDOE_REVERB_TYPE_7 磁性
mTXCameraRecord.setReverb(TXRecordCommon.VIDOE_REVERB_TYPE_1);

// 设置变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0 关闭变声
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1 熊孩子
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2 萝莉
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3 大叔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4 重金属
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6 外国人
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7 困兽
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8 死肥仔
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9 强电流
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10 重机械
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11 空灵
mTXCameraRecord.setVoiceChangerType(TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1);
```

> 说明：
> 变声混响只针对录制人声有效，针对 BGM 无效。

设定画面质量

最近更新时间：2022-03-04 11:00:41

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

内容介绍

在 TRTCCloud 中，您可以通过以下方式调整画质：

- TRTCCloud.enterRoom 中的 TRTCAppScene 参数：用于选择您的应用场景。
- TRTCCloud.setVideoEncoderParam：用于设置编码参数。
- TRTCCloud.setNetworkQosParam：用于设置网络调控策略。

本文主要介绍如何配置上述参数，使音视频通话 TRTC SDK 的画质效果符合您的项目需要。
您也可以参考以下 Demo：

- iOS：SetVideoQualityViewController.m
- Android：SetVideoQualityActivity.java
- Windows：TRTCMainViewController.cpp

支持的平台

iOS	Android	Mac OS	Windows	桌面浏览器	Electron	微信小程序	Flutter
✓	✓	✓	✓	✓	✓	✓	✓

桌面浏览器端设定画面质量的详细操作，请参见 设定指引。

TRTCAppScene

- **VideoCall**
  对应视频通话场景，即绝大多数时间都是两人或两人以上视频通话的场景，内部编码器和网络协议优化侧重流畅性，降低通话延迟和卡顿率。
- **LIVE**
  对应直播场景，即绝大多数时间都是一人直播，偶尔有多人视频互动的场景，内部编码器和网络协议优化侧重性能和兼容性，性能和清晰度表现更佳。

TRTCVideoEncParam

推荐的配置

应用场景	videoResolution	videoFps	videoBitrate
视频通话（手机）	640x360	15	550kbps
视频会议（主画面 @ Mac Win）	1280x720	15	1200kbps
视频会议（主画面 @ 手机）	640x360	15	900kbps
视频会议（小画面）	320x180	15	250kbps
在线教育（老师 @ Mac Win）	960x540	15	850kbps

应用场景	videoResolution	videoFps	videoBitrate
在线教育（老师 @ iPad）	640x360	15	550kbps
在线教育（学生）	320x180	15	250kbps

各字段详解

- **(TRTCVideoResolution) videoResolution**
  编码分辨率，例如 640 x 360 是指编码出的画面的宽（像素） x 高（像素），我们在 TRTCVideoResolution 枚举定义里只定义了宽 >= 高的横屏（Landscape）分辨率，如果想要使用竖屏分辨率，需要将 resMode 设置为 Portrait。

> ⚠ 注意
> 由于很多硬件编解码器只支持能被 16 整除的像素宽度，所以 SDK 实际编码出的分辨率并不一定完全按照参数自定，而是会自动进行 16 整除修正。例如 640 x 360 的分辨率，在 SDK 内部有可能会适配为 640 × 368。

- **(TRTCVideoResolutionMode) resMode**
  指横屏或竖屏分辨率，由于 TRTCVideoResolution 中只定义了横屏分辨率，如果您希望使用 360 x 640 这样的竖屏分辨率，就需要指定 resMode 为 TRTCVideoResolutionModePortrait。一般 PC 和 Mac 采用横屏（Landscape）分辨率，手机采用竖屏（Portrait）分辨率。

- **(int) videoFps**
  帧率（FPS），也就是每秒钟要编码多少帧画面。推荐设置为 15 FPS，这样既能保证画面足够流畅，又不会因为每秒帧数太多而拉低单幅画面的清晰度。

  如果您对流畅度要求比较高，可以设置为 20 FPS 或 25 FPS。但请不要设置 25 FPS 以上的数值，因为电影的常规帧率也只有 24 FPS。

- **(int) videoBitrate**
  视频码率，即每秒钟编码器输出多少 Kbit 的编码后的二进制数据。如果您将 videoBitrate 设置为 800kbps，那么每秒钟编码器会产生 800kbit 的视频数据，这些数据如果存储成一个文件，那么文件大小就是 800kbit，也就是100KB，也就是 0.1M。

分辨率码率参照表

视频码率并不是越高越好，它跟分辨率之间要有比较恰当的映射关系，如下表所示。

分辨率定义	宽高比	建议码率	高端配置
TRTCVideoResolution_120_120	1:1	80kbps	120kbps
TRTCVideoResolution_160_160	1:1	100kbps	150kbps
TRTCVideoResolution_270_270	1:1	200kbps	300kbps
TRTCVideoResolution_480_480	1:1	350kbps	525kbps
TRTCVideoResolution_160_120	4:3	100kbps	150kbps
TRTCVideoResolution_240_180	4:3	150kbps	225kbps
TRTCVideoResolution_280_210	4:3	200kbps	300kbps
TRTCVideoResolution_320_240	4:3	250kbps	375kbps
TRTCVideoResolution_400_300	4:3	300kbps	450kbps
TRTCVideoResolution_480_360	4:3	400kbps	600kbps
TRTCVideoResolution_640_480	4:3	600kbps	900kbps
TRTCVideoResolution_960_720	4:3	1000kbps	1500kbps
TRTCVideoResolution_160_90	16:9	150kbps	250kbps
TRTCVideoResolution_256_144	16:9	200kbps	300kbps
TRTCVideoResolution_320_180	16:9	250kbps	400kbps
TRTCVideoResolution_480_270	16:9	350kbps	550kbps
TRTCVideoResolution_640_360	16:9	550kbps	900kbps

分辨率定义	宽高比	建议码率	高端配置
TRTCVideoResolution_960_540	16:9	850kbps	1300kbps
TRTCVideoResolution_1280_720	16:9	1200kbps	1800kbps

TRTCNetworkQosParam

QosPreference

在网络带宽比较充裕的情况下，清晰和流畅是可以兼顾的，但当用户的网络并不理想时，究竟是优先保证清晰还是优先保证流畅？您可以通过指定 TRTCNetworkQosParam 中的 preference 参数来做出选择。

- **流畅优先（TRTCVideoQosPreferenceSmooth）**
  在用户遭遇弱网环境时，画面会变得模糊，且会有较多马赛克，但可以保持流畅或轻微卡顿。
- **清晰优先（TRTCVideoQosPreferenceClear）**
  在用户遭遇弱网环境时，画面会尽可能保持清晰，但可能会更容易出现卡顿。

ControlMode

controlMode 参数选择 **TRTCQosControlModeServer** 即可，TRTCQosControlModeClient 是腾讯云研发团队做内部调试用的，请勿关注。

常见的误区

1. 分辨率越高越好?
较高的分辨率也需要较高的码率来支撑，如果分辨率选择 1280 x 720，但码率却指定为 200kbps，画面就会有大量的马赛克。推荐参考 分辨率码率参照表 进行设置。

2. 帧率越高越好?
由于摄像头采集的画面是曝光阶段中所有现实物体的完整映射，所以并不是帧率越高，感官就越流畅，这一点跟游戏里的FPS是不一样的。恰恰相反，帧率过高，会拉低每帧画面的画质，也会减少摄像机的曝光时间，效果可能会更差。

3. 码率越高越好?
较高的码率也需要较高的分辨率来匹配，对于 320 x 240 这样分辨率，1000kbps 的码率就很浪费了，推荐参考 分辨率码率参照表 进行设置。

4. 用 Wi-Fi 的时候就可以设置很高的分辨率和码率
并不是说 Wi-Fi 的网速是恒定不变的，如果离无线路由器较远， 或者路由器信道被占用，可能网速还不如 4G。
针对这种情况， TRTC SDK 提供了测速功能，可以在视频通话前先进行测速，根据打分值来确定网络好坏。

视频画面旋转和缩放

最近更新时间：2022-01-27 14:49:18

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

内容介绍

跟手机直播千篇一律的竖屏体验不同，音视频通话 TRTC 需要兼顾横屏和竖屏两种场景，因此就会有很多横竖屏的处理逻辑需要去应对，本文主要介绍：

- 如何实现竖屏模式，例如：微信的视频通话就是一个典型的竖屏体验模式。
- 如何实现横屏模式，例如：多人音视频房间 App（类似小鱼易连）往往都是采用横屏模式。
- 如何自定义控制本地画面和远程画面的旋转方向和填充模式。

TRTCVideoEncParam.videoResolution = 1280x720
TRTCVideoEncParam.resMode = Landscape
录制出的视频分辨率：1280 x 720
CDN直播观看分辨率：1280 x 720

TRTCVideoEncParam.videoResolution = 1280x720
TRTCVideoEncParam.resMode = Portrait
录制出的视频分辨率：720 x 1280
CDN直播观看分辨率：720 x 1280

平台支持

iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
✓	✓	✓	✓	✓	×	×

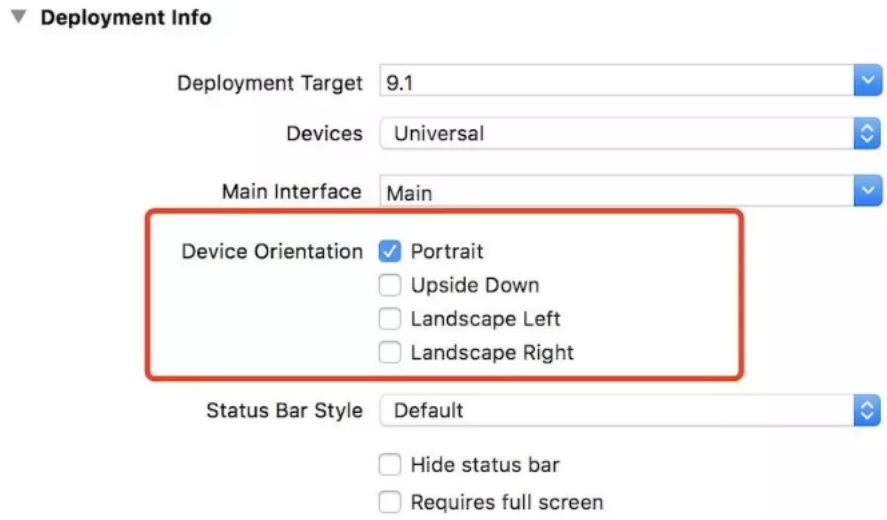
竖屏模式

如果要实现类似微信视频通话的体验模式，需要做两项工作：

1. 配置 App 的 UI 界面为竖屏

iOS 平台

可以直接在 XCode 的 **General** > **Deployment Info** > **Device Orientation**中进行设置：



您也可以通过实现 Appdelegate 中的 supportedInterfaceOrientationsForWindow 方法来达到相同目标：

```
- (UIInterfaceOrientationMask)application:(UIApplication *)application
supportedInterfaceOrientationsForWindow:(UIWindow *)window
{

return UIInterfaceOrientationMaskPortrait ;

}
```

> 说明：
> CSDN 上有一篇文章 iOS横竖屏旋转及其基本适配方法，详细介绍了 iOS 平台中关于屏幕方向的一些开发经验。

Android 平台

通过指定 activity 的 screenOrientation 属性为 portrait，即可指定该界面为竖屏模式：

```
<activity android:name=".trtc.TRTCMainActivity"
android:launchMode="singleTask" android:windowSoftInputMode="adjustPan"
android:screenOrientation="portrait" />
```

2. 配置 SDK 使用竖屏分辨率

在使用 TRTCCloud 的 setVideoEncoderParam 接口设置视频编码参数时，将 resMode 指定为 TRTCVideoResolutionModePortrait 即可。
示例代码如下：

iOS 平台

```
TRTCVideoEncParam* encParam = [TRTCVideoEncParam new];
encParam.videoResolution = TRTCVideoResolution_640_360;
encParam.videoBitrate = 600;
encParam.videoFps = 15;
encParam.resMode = TRTCVideoResolutionModePortrait; //设置分辨率模式为竖屏模式

[trtc setVideoEncoderParam: encParam];
```

Android 平台

```
TRTCCloudDef.TRTCVideoEncParam encParam = new TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_640_360;
encParam.videoBitrate = 600;
encParam.videoFps = 15;
encParam.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT; //设置分辨率模式为竖屏模式
trtc.setVideoEncoderParam(encParams);
```

横屏模式

如果希望 App 是横屏体验，那么您需要做的工作跟竖屏模式类似，只是将第一步和第二步中的参数都进行相应的调整即可。
尤其是 第二步 中，TRTCVideoEncParam 中的 resMode 值：

- 在 iOS 平台中应该指定为 TRTCVideoResolutionModeLandscape。
- 在 Android 平台中应该指定为 TRTC_VIDEO_RESOLUTION_MODE_LANDSCAPE。

自定义控制

TRTC SDK 本身提供了大量的接口函数可以操控本地和远程画面的旋转方向和填充模式：

接口函数	功能作用	备注说明
setLocalViewRotation	本地预览画面的顺时针旋转角度	支持顺时针旋转90度、180度和270度三个方向
setLocalViewFillMode	本地预览画面的填充模式	是裁剪还是留黑边
setRemoteViewRotation	远端视频画面的顺时针旋转角度	支持顺时针旋转90度、180度和270度三个方向
setRemoteViewFillMode	远端视频画面的填充模式	是裁剪还是留黑边
setVideoEncoderRotation	设置编码器输出的画面顺时针旋转角度	支持顺时针旋转90度、180度和270度三个方向

GSensorMode

考虑到画面旋转牵扯到录制和 CDN 旁路直播的各种适配问题，音视频通话 TRTC SDK 仅提供了一种简单的重力感应自适应功能，您可以通过 TRTCCloud 的 setGSensorMode 接口来开启。

该功能目前仅支持180度上下旋转的自适应，也就是当用户自己的手机上下颠倒180度时，对方看到的画面朝向还是会保持不变（旋转90度或者270度的适应尚不支持）。而且这种自适应是基于对编码器的方向调整而实现的，因此录制出的视频，以及小程序和 H5 端看到的视频画面也能做到保持原方向不变。

> **注意：**
> 重力感应自适应的另一种实现方案是在每一帧视频信息里都带上当前视频的重力朝向，然后在远程用户那里自适应的调整渲染方向，但这种方案需要引入额外的转码资源才能解决录制出的视频朝向跟期望的视频朝向保持一致的问题，因此并不推荐。

音视频设备测试

最近更新时间：2021-08-13 15:05:25

版本支持

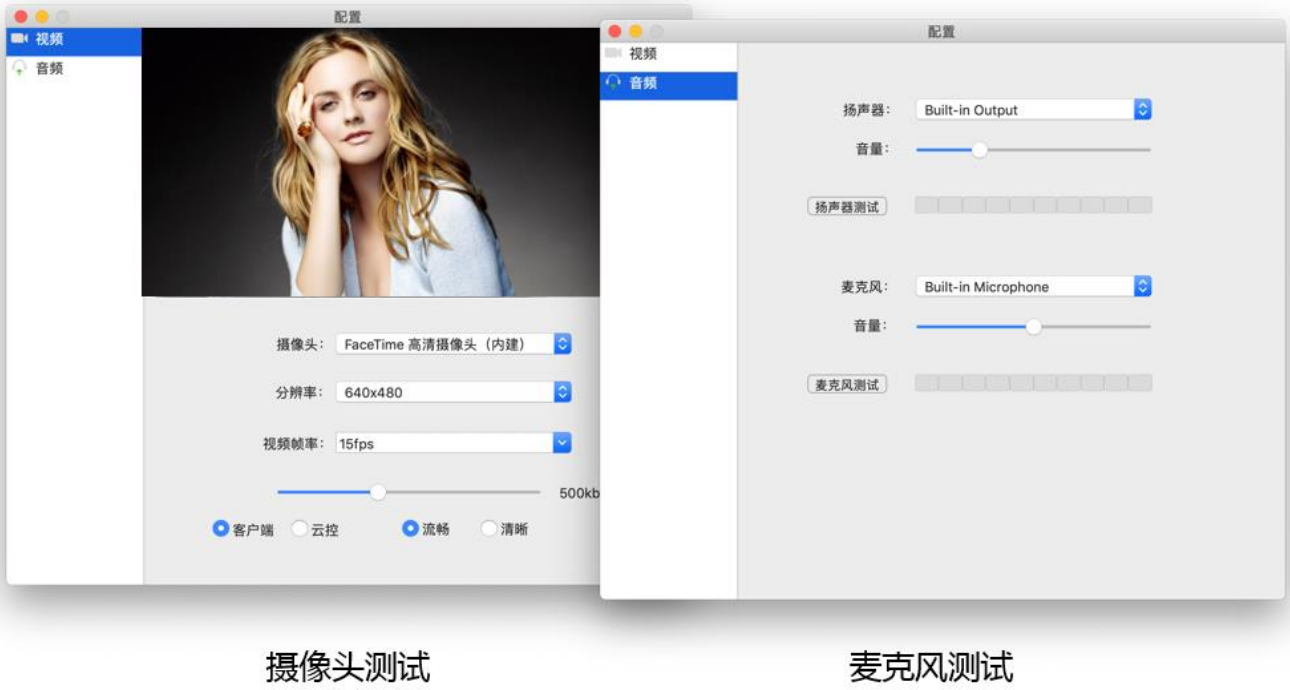
本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

内容介绍

在进行视频通话之前，建议先进行摄像头和麦克风等设备的测试，否则等用户真正进行通话时很难发现设备问题。



摄像头测试　　麦克风测试

支持此功能的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
×	×	✓	✓	✓	×	✓（Web 端）

测试摄像头

使用 TRTCCloud 的 startCameraDeviceTestInView 接口可以进行摄像头测试，在测试过程中可以通过调用 setCurrentCameraDevice 函数切换摄像头。

Mac平台

```
// 显示摄像头测试界面（支持预览摄像头，切换摄像头）
- (IBAction)startCameraTest:(id)sender {
// 开始摄像头测试, cameraPreview为macOS下的NSView或者iOS平台的UIView
[self.trtcCloud startCameraDeviceTestInView:self.cameraPreview];
}
```

```
//关闭摄像头测试界面
- (void)windowWillClose:(NSNotification *)notification{
// 结束摄像头测试
[self.trtcCloud stopCameraDeviceTest];
}
```

Windows平台（C++）

```
// 启动摄像头测试。传入需要渲染视频的控件句柄。
void TRTCMainViewController::startTestCameraDevice(HWND hwnd)
{
trtcCloud->startCameraDeviceTest(hwnd);
}
// 关闭摄像头测试
void TRTCMainViewController::stopTestCameraDevice()
{
trtcCloud->stopCameraDeviceTest();
}
```

Windows平台（C#）

```
// 启动摄像头测试。传入需要渲染视频的控件句柄。
private void startTestCameraDevice(Intptr hwnd)
{
mTRTCCloud.startCameraDeviceTest(hwnd);
}

// 关闭摄像头测试
private void stopTestCameraDevice()
{
mTRTCCloud.stopCameraDeviceTest();
}
```

麦克风测试

使用 TRTCCloud 的 startMicDeviceTest 函数可以测试麦克风音量，回调函数会返回实时的麦克风音量值。

Mac平台

```
// 麦克风测试示例代码
-(IBAction)micTest:(id)sender {
NSButton *btn = (NSButton *)sender;
if (btn.state == 1) {
//开始麦克风测试
__weak __typeof(self) wself = self;
[self.trtcCloud startMicDeviceTest:500 testEcho:^(NSInteger volume) {
dispatch_async(dispatch_get_main_queue(), ^{
// 刷新麦克风音量的进度条
[wself _updateInputVolume:volume];
});
}];
btn.title = @"停止测试";
}
else{
//结束麦克风测试
[self.trtcCloud stopMicDeviceTest];
[self _updateInputVolume:0];
btn.title = @"开始测试";
```

```
}
}
```

Windows平台（C++）

```
// 麦克风测试示例代码
void TRTCMainViewController::startTestMicDevice()
{
// 设置音量回调频率，此处500ms回调一次，在 onTestMicVolume 回调接口监听。
uint32_t interval = 500;
// 开始麦克风测试
trtcCloud->startMicDeviceTest(interval);
}

// 结束麦克风测试
void TRTCMainViewController::stopTestMicDevice()
{
trtcCloud->stopMicDeviceTest();
}
```

Windows平台（C#）

```
// 麦克风测试示例代码
private void startTestMicDevice()
{
// 设置音量回调频率，此处500ms回调一次，在 onTestMicVolume 回调接口监听。
uint interval = 500;
// 开始麦克风测试
mTRTCCloud.startMicDeviceTest(interval);
}

// 结束麦克风测试
private void stopTestMicDevice()
{
mTRTCCloud.stopMicDeviceTest();
}
```

扬声器测试

使用 TRTCCloud 的 startSpeakerDeviceTest 函数会通过播放一段默认的 mp3 音频测试扬声器是否在正常工作。

Mac平台

```
// 扬声器测试示例代码
// 以作为 NSButton 的点击事件为例，在 xib 中设置 Button 在 On 和 Off 下的标题分别为"结束测试"和"开始测试"
- (IBAction)speakerTest:(NSButton *)btn {
NSString *path = [[NSBundle mainBundle] pathForResource:@"test-32000-mono" ofType:@"mp3"];
if (btn.state == NSControlStateValueOn) {
// 单击"开始测试"
__weak __typeof(self) wself = self;
[self.trtcEngine startSpeakerDeviceTest:path onVolumeChanged:^(NSInteger volume, BOOL playFinished) {
// 以下涉及 UI 操作，需要切换到 main queue 中执行
dispatch_async(dispatch_get_main_queue(), ^{
// 这里 _updateOutputVolume 为更新界面中的扬声器音量指示器
[wself _updateOutputVolume:volume];
if (playFinished) {
// 播放完成时将按钮状态置为"开始测试"
sender.state = NSControlStateValueOff;
}
```

```
});
}];
} else {
// 单击"结束测试"
[self.trtcEngine stopSpeakerDeviceTest];
[self _updateOutputVolume:0];
}
}

// 更新扬声器音量指示器
- (void)_updateOutputVolume:(NSInteger)volume {
// speakerVolumeMeter 为 NSLevelIndicator
self.speakerVolumeMeter.doubleValue = volume / 255.0 * 10;
}
```

Windows平台（C++）

```
// 扬声器测试示例代码
void TRTCMainViewController::startTestSpeakerDevice(std::string testAudioFilePath)
{
// testAudioFilePath 音频文件的绝对路径，路径字符串使用 UTF-8 编码格式，支持文件格式: wav、mp3。
// 从 onTestSpeakerVolume 回调接口监听扬声器测试音量值。
trtcCloud->startSpeakerDeviceTest(testAudioFilePath.c_str());
}
// 结束扬声器测试
void TRTCMainViewController::stopTestSpeakerDevice() {
trtcCloud->stopSpeakerDeviceTest();
}
```

Windows平台（C#）

```
// 扬声器测试示例代码
private void startTestSpeakerDevice(string testAudioFilePath)
{
// testAudioFilePath 音频文件的绝对路径，路径字符串使用 UTF-8 编码格式，支持文件格式: wav、mp3。
// 从 onTestSpeakerVolume 回调接口监听扬声器测试音量值。
mTRTCCloud.startSpeakerDeviceTest(testAudioFilePath);
}

// 结束扬声器测试
private void stopTestSpeakerDevice() {
mTRTCCloud.stopSpeakerDeviceTest();
}
```

通话前网络测试

最近更新时间：2021-12-30 10:29:21

普通用户很难评估网络质量，建议您在进行视频通话之前先进行网络测试，通过测速可以更直观地评估网络质量。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	-	-	✓	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

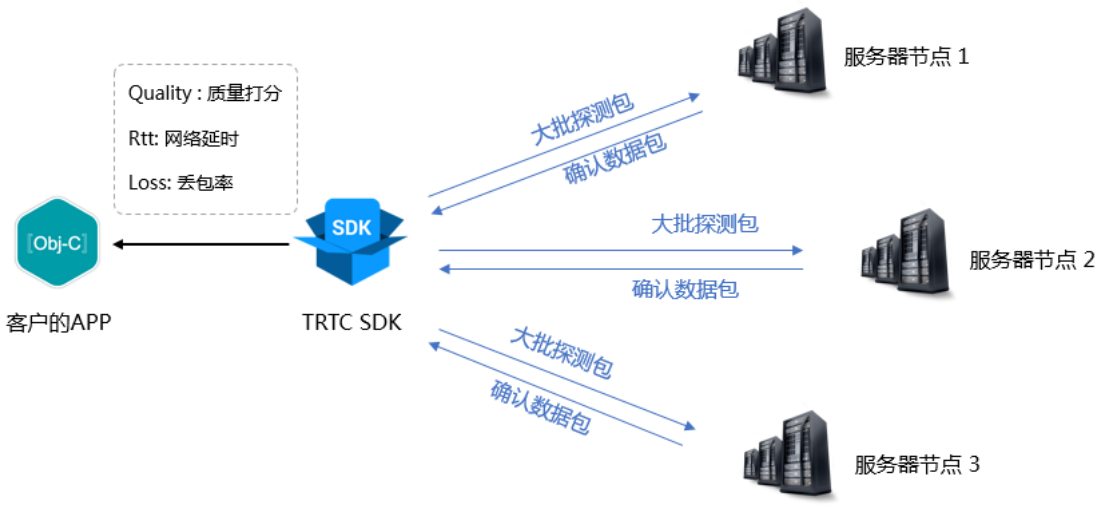
注意事项

- 视频通话期间请勿测试，以免影响通话质量。
- 测速本身会消耗一定的流量，从而产生极少量额外的流量费用（基本可以忽略）。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
✓	✓	✓	✓	✓	×	✓（参考：Web 端教程）

测速的原理



- 测速的原理是 SDK 向服务器节点发送一批探测包，然后统计回包的质量，并将测速的结果通过回调接口通知出来。
- 测速的结果将会用于优化 SDK 接下来的服务器选择策略，因此推荐您在用户首次通话前先进行一次测速，这将有助于我们选择最佳的服务器。同时，如果测试结果非常不理想，您可以通过醒目的 UI 提示用户选择更好的网络。
- 测速的结果（TRTCSpeedTestResult）包含如下几个字段：

字段	含义	含义说明
success	是否成功	本次测试是否成功
errMsg	错误信息	带宽测试的详细错误信息
ip	服务器 IP	测速服务器的 IP

quality	网络质量评分	通过评估算法测算出的网络质量，loss 越低，rtt 越小，得分也就越高
upLostRate	上行丢包率	范围是[0 – 1.0]，例如0.3代表每向服务器发送10个数据包，可能有3个会在中途丢失
downLostRate	下行丢包率	范围是[0 – 1.0]，例如0.2代表从服务器每收取10个数据包，可能有2个会在中途丢失
rtt	网络延时	代表 SDK 跟服务器一来一回之间所消耗的时间，这个值越小越好，正常数值在 10ms – 100ms 之间
availableUpBandwidth	上行带宽	预测的上行带宽，单位为kbps， -1表示无效值
availableDownBandwidth	下行带宽	预测的下行带宽，单位为kbps， -1表示无效值

如何测速

通过 TRTCCloud 的 startSpeedTest 功能可以启动测速功能，测速的结果会通过回调函数返回。

Objective-C

```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
- (void)onLogin:(NSString *)userId userSig:(NSString *)userSid
{
TRTCSpeedTestParams *params;
// sdkAppID 为控制台中获取的实际应用的 AppID
params.sdkAppID = sdkAppId;
params.userID = userId;
params.userSig = userSig;
// 预期的上行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
params.expectedUpBandwidth = 5000;
// 预期的下行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
params.expectedDownBandwidth = 5000;
[trtcCloud startSpeedTest:params];
}
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
// 测速完成后，会回调出测速结果
}
```

Java

```
//启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
public void onLogin(String userId, String userSig)
{
TRTCCloudDef.TRTCSpeedTestParams params = new TRTCCloudDef.TRTCSpeedTestParams();
params.sdkAppId = GenerateTestUserSig.SDKAPPID;
params.userId = mEtUserId.getText().toString();
params.userSig = GenerateTestUserSig.genTestUserSig(params.userId);
params.expectedUpBandwidth = Integer.parseInt(expectUpBandwidthStr);
params.expectedDownBandwidth = Integer.parseInt(expectDownBandwidthStr);
// sdkAppID 为控制台中获取的实际应用的 AppID
trtcCloud.startSpeedTest(params);
}

// 监听测速结果，继承 TRTCCloudListener 并实现如下方法
void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result)
{
// 测速完成后，会回调出测速结果
}
```

C++

```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
void onLogin(const char* userId, const char* userSig)
{
TRTCSpeedTestParams params;
// sdkAppID 为控制台中获取的实际应用的 AppID
params.sdkAppID = sdkAppId;
params.userId = userid;
param.userSig = userSig;
// 预期的上行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
param.expectedUpBandwidth = 5000;
// 预期的下行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
param.expectedDownBandwidth = 5000;
trtcCloud->startSpeedTest(params);
}

// 监听测速结果
void TRTCCloudCallbackImpl::onSpeedTestResult(
const TRTCSpeedTestResult& result)
{
// 测速完成后，会回调出测速结果
}
```

C#

```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
private void onLogin(string userId, string userSig)
{
TRTCSpeedTestParams params;
// sdkAppID 为控制台中获取的实际应用的 AppID
params.sdkAppID = sdkAppId;
params.userId = userid;
param.userSig = userSig;
// 预期的上行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
param.expectedUpBandwidth = 5000;
// 预期的下行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
param.expectedDownBandwidth = 5000;
mTRTCCloud.startSpeedTest(params);
}

// 监听测速结果
public void onSpeedTestResult(TRTCSpeedTestResult result)
{
// 测速完成后，会回调出测速结果
}
```

测速工具

如果您不想通过调用接口的方式来进行网络测速，TRTC 还提供了桌面端的网络测速工具程序，帮助您快速获取详细的网络质量信息。

下载链接

Mac | Windows

测试指标

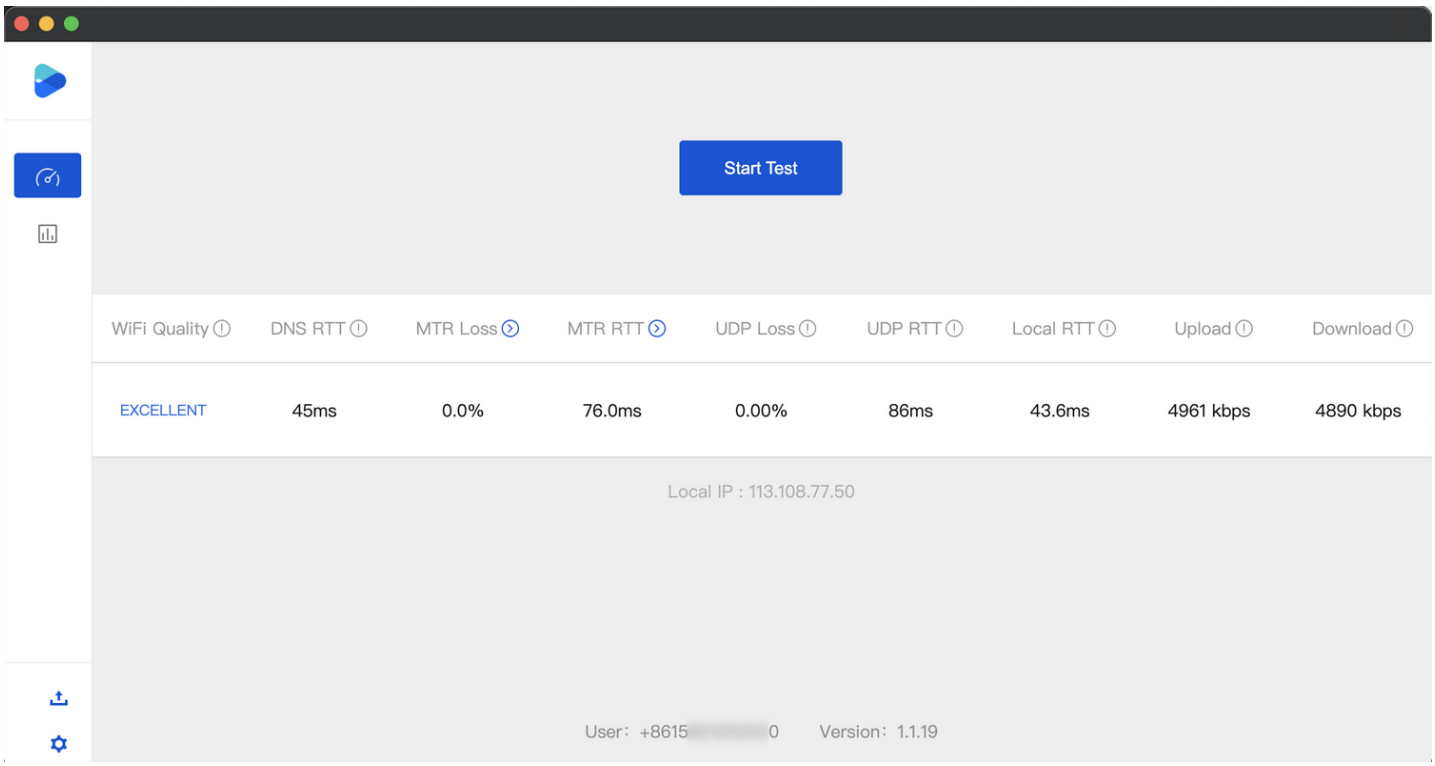
指标	含义
WiFi Quality	Wi-Fi 信号质量

指标	含义
DNS RTT	腾讯云的测速域名解析耗时
MTR	MTR 是一款网络测试工具，能探测客户端到 TRTC 节点的丢包率与延时，还可以查看路由中每一跳的具体信息
UDP Loss	客户端到 TRTC 节点的 UDP 丢包率
UDP RTT	客户端到 TRTC 节点的 UDP 延时
Local RTT	客户端到本地网关的延时
Upload	上行预估带宽
Download	下行预估带宽

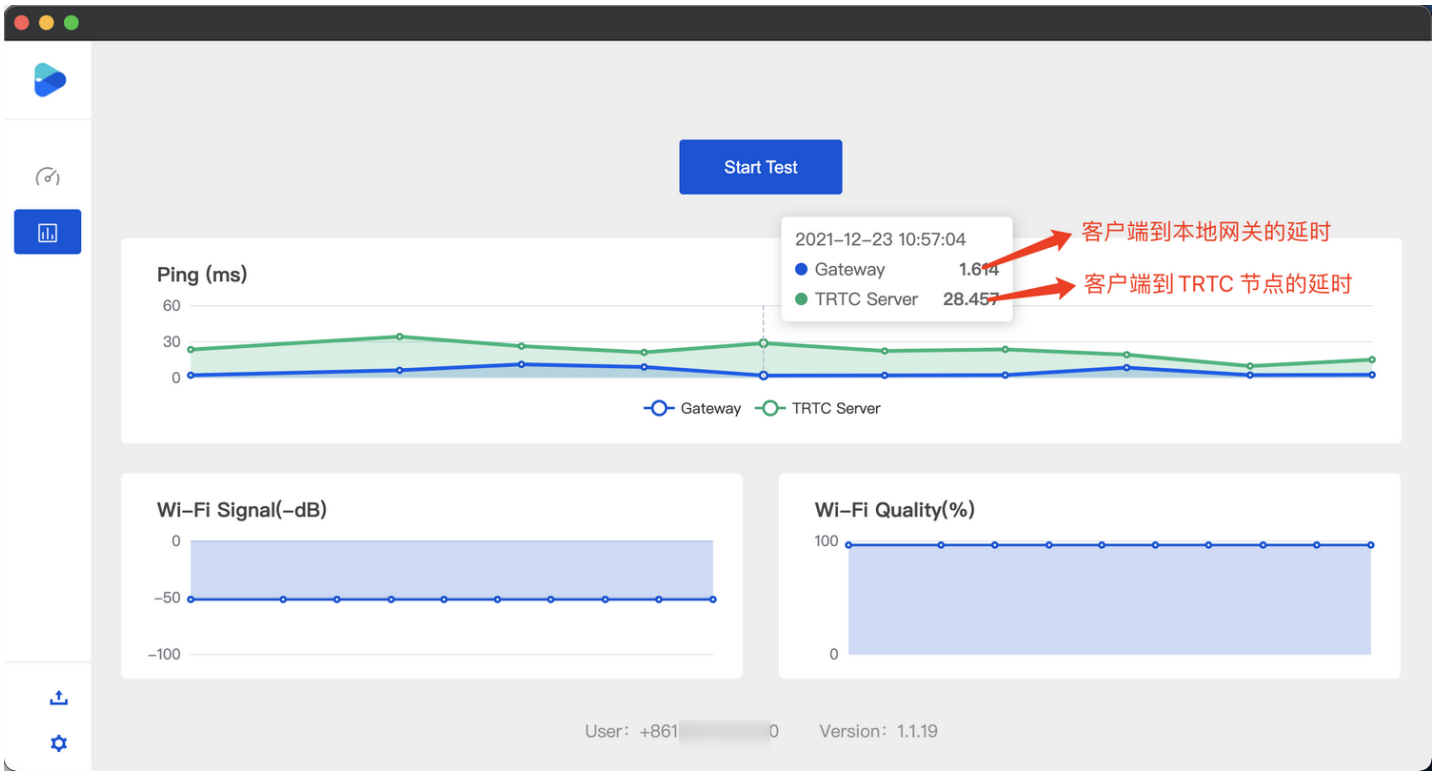
工具截图

登录：

快速测试：


Start Test
WiFi Quality
DNS RTT
MTR Loss
MTR RTT
UDP Loss
UDP RTT
Local RTT
Upload
Download
EXCELLENT
45ms
0.0%
76.0ms
0.00%
86ms
43.6ms
4961 kbps
4890 kbps
Local IP : 113.108.77.50
User: +8615 0
Version: 1.1.19

持续测试：


Start Test
2021-12-23 10:57:04
Gateway 1.614
TRTC Server 28.457
客户端到本地网关的延时
客户端到 TRTC 节点的延时
Ping (ms)
60
30
0
Gateway
TRTC Server
Wi-Fi Signal(-dB)
0
-50
-100
Wi-Fi Quality(%)
100
0
User: +861 0
Version: 1.1.19

开启高级权限控制

最近更新时间：2021-12-23 17:07:39

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

内容介绍

如果您希望给某些房间中加入进房限制或者上麦限制，也就是仅允许指定的用户去进房或者上麦，而您又担心在客户端判断权限很容易遭遇破解攻击，那么可以考虑**开启高级权限控制**。

在如下场景下，您并不需要开启高级权限控制的：

- 情况1：本身希望越多的人观看越好，对进入房间的权限控制无要求。
- 情况2：对攻击者破解客户端的防范需求不迫切。

在如下场景下，建议您开启高级权限控制以获得更佳的安全性：

- 情况1：对安全性要求较高的视频通话或者语音通话场景。
- 情况2：对不同房间设置不同进入权限的场景。
- 情况3：对观众上麦有权限控制的场景。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
✓	✓	✓	✓	✓	✓	✓

高级权限控制的原理

开启高级权限控制后，TRTC 的后台服务系统就不会仅校验 UserSig 这一个“进房票据”，还会校验一个叫做 **PrivateMapKey** 的“权限票据”，权限票据中包含了一个加密后的 roomid 和一个加密后的“权限位列表”。

由于 PrivateMapKey 中包含 roomid，所以当用户只提供了 UserSig 没有提供 PrivateMapKey 时，并不能进入指定的房间。

PrivateMapKey 中的“权限位列表”使用了一个 byte 中的 8 个比特位，分别代表了持有该票据的用户，在该票据指定的房间中所拥有的八种具体的功能权限：

位数	二进制表示	十进制数字	权限含义
第 1 位	0000 0001	1	创建房间的权限
第 2 位	0000 0010	2	进入房间的权限
第 3 位	0000 0100	4	发送语音的权限
第 4 位	0000 1000	8	接收语音的权限
第 5 位	0001 0000	16	发送视频的权限
第 6 位	0010 0000	32	接收视频的权限
第 7 位	0100 0000	64	发送辅路（也就是屏幕分享）视频的权限
第 8 位	1000 0000	128	接收辅路（也就是屏幕分享）视频的权限

开启高级权限控制

步骤1：在 TRTC 控制台中开启高级权限控制

1. 在腾讯云实时音视频控制台中单击左侧的 **应用管理**。
2. 在右侧的应用列表中选择想要开启高级权限控制的一款应用，并单击 **功能配置**。
3. 在“功能配置”页卡中打开 **启用高级权限控制** 按钮，单击 **确定**，即可开启高级权限控制。

> ⚠ 注意：
> 当某一个 SDKAppid 开启高级权限控制后，使用该 SDKAppid 的所有用户都需要在 TRTCParams 中传入 privateMapKey 参数才能成功进房（如 步骤 2 所述），如果您线上有使用此 SDKAppid 的用户，请不要轻易开启此功能。

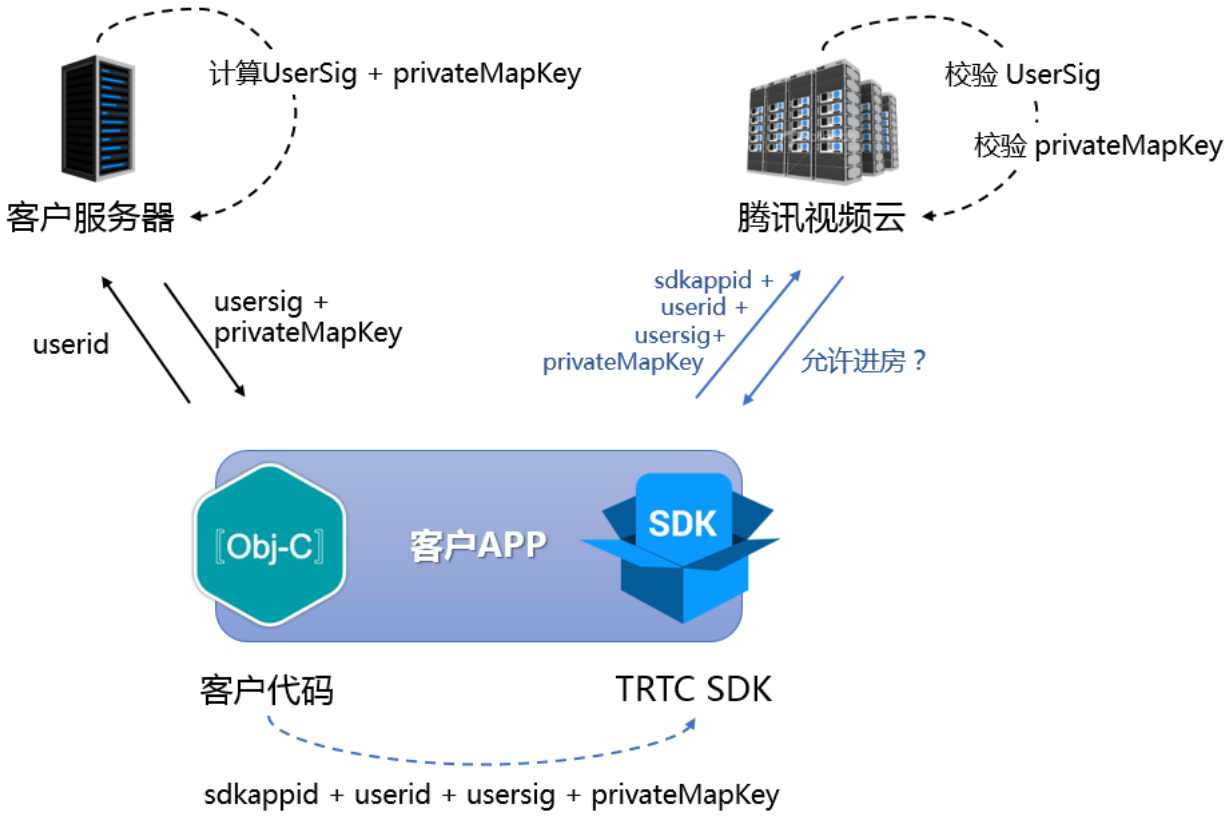
步骤2：在您的服务端计算 PrivateMapKey

由于 PrivateMapKey 的价值就是为了防止客户端被逆向破解，从而出现“非会员也能进高等级房间”的破解版本，所以它只适合在您的服务器计算再返回给您的 App，绝不能在您的 App 端直接计算。

我们提供了 Java、GO、PHP、Node.js、Python、C# 和 C++ 版本的 PrivateMapKey 计算代码，您可以直接下载并集成到您的服务端。

语言版本	关键函数	下载链接
Java	genPrivateMapKey 和 genPrivateMapKeyWithStringRoomID	Github
GO	GenPrivateMapKey 和 GenPrivateMapKeyWithStringRoomID	Github
PHP	genPrivateMapKey 和 genPrivateMapKeyWithStringRoomID	Github
Node.js	genPrivateMapKey 和 genPrivateMapKeyWithStringRoomID	Github
Python	genPrivateMapKey和 genPrivateMapKeyWithStringRoomID	Github
C#	genPrivateMapKey和 genPrivateMapKeyWithStringRoomID	Github
C++	genPrivateMapKey和 genPrivateMapKeyWithStringRoomID	GitHub

步骤3：由您的服务端将 PrivateMapKey 下发给您的 App



如上图所示，当您的服务器计算好 PrivateMapKey 之后，就可以下发给您的 App，您的 App 可以通过两种方案将 PrivateMapKey 传递给 SDK：

方案一：在 enterRoom 时传递给 SDK

如果想要控制用户进入房间的权限，您可以在调用 TRTCCloud 的 enterRoom 接口时，通过设置 TRTCParams 中的 **privateMapKey** 参数即可实现。

这种进房时校验 PrivateMapKey 的方案比较简单，非常适合于在用户进入房间前就能将用户权限确认清楚的场景。

方案二：通过实验性接口更新给 SDK

在直播场景中，往往都会有观众上麦变成主播的连麦场景。当观众变成主播时，TRTC 会再校验一次进房时在进房参数 TRTCParams 中携带的 PrivateMapKey，如果您将 PrivateMapKey 的有效期设置得比较短，例如“5分钟”，就会很容易触发校验失败进而导致用户被踢出房间。

要解决这个问题，除了可以延长有效期（例如将“5分钟”改成“6小时”），还可以在观众通过 switchRole 将自己的身份切换成主播之前，重新向您的服务器申请一个 privateMapKey，并调用 SDK 的实验性接口 updatePrivateMapKey 将其更新到 SDK 中，示例代码如下：

Android

```
JSONObject jsonObject = new JSONObject();
try {
jsonObject.put("api", "updatePrivateMapKey");
JSONObject params = new JSONObject();
params.put("privateMapKey", "xxxxx"); // 填写新的 privateMapKey
jsonObject.put("params", params);
mTRTCCloud.callExperimentalAPI(jsonObject.toString());
} catch (JSONException e) {
e.printStackTrace();
}
```

iOS

```
NSMutableDictionary *params = [[NSMutableDictionary alloc] init];
[params setObject:@"xxxxx" forKey:@"privateMapKey"]; // 填写新的 privateMapKey
NSDictionary *dic = @{@"api": @"updatePrivateMapKey", @"params": params};
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:dic options:0 error:NULL];
```

```
NSString *jsonStr = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
[WXTRTCCloud sharedInstance] callExperimentalAPI:jsonStr];
```

C++

```
std::string api = "{\"api\":\"updatePrivateMapKey\",\"params\":{\"privateMapKey\":"xxxxx"}}";
TRTCCloudCore::GetInstance()->getTRTCCloud()->callExperimentalAPI(api.c_str());
```

C#

```
std::string api = "{\"api\":\"updatePrivateMapKey\",\"params\":{\"privateMapKey\":"xxxxx"}}";
mTRTCCloud.callExperimentalAPI(api);
```

常见问题

1. 线上的房间为什么都进不去了?

房间权限控制一旦开启后，当前 SDKAppid 下的房间就需要在 TRTCParams 中设置 privateMapKey 才能进入，所以如果您线上业务正在运营中，并且线上版本并没有加入 privateMapKey 的相关逻辑，请不要开启此开关。

2. PrivateMapKey 和 UserSig 有什么区别?

- UserSig 是 TRTCParams 的必选项，作用是检查当前用户是否有权使用 TRTC 云服务，用于防止攻击者盗用您的 SDKAppid 账号内的流量。
- PrivateMapKey 是 TRTCParams 的非必选项，作用是检查当前用户是否有权进入指定 roomid 的房间，以及该用户在该房间所能具备的权限，当您的业务需要对用户进行身份区分的时候才有必要开启。

发送自定义消息

最近更新时间：2021-08-12 20:00:56

音视频通话 TRTC SDK 提供了发送自定义消息的功能，通过该功能，角色为主播的用户都可以向同一个视频房间里的其他用户广播自己的定制消息。

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

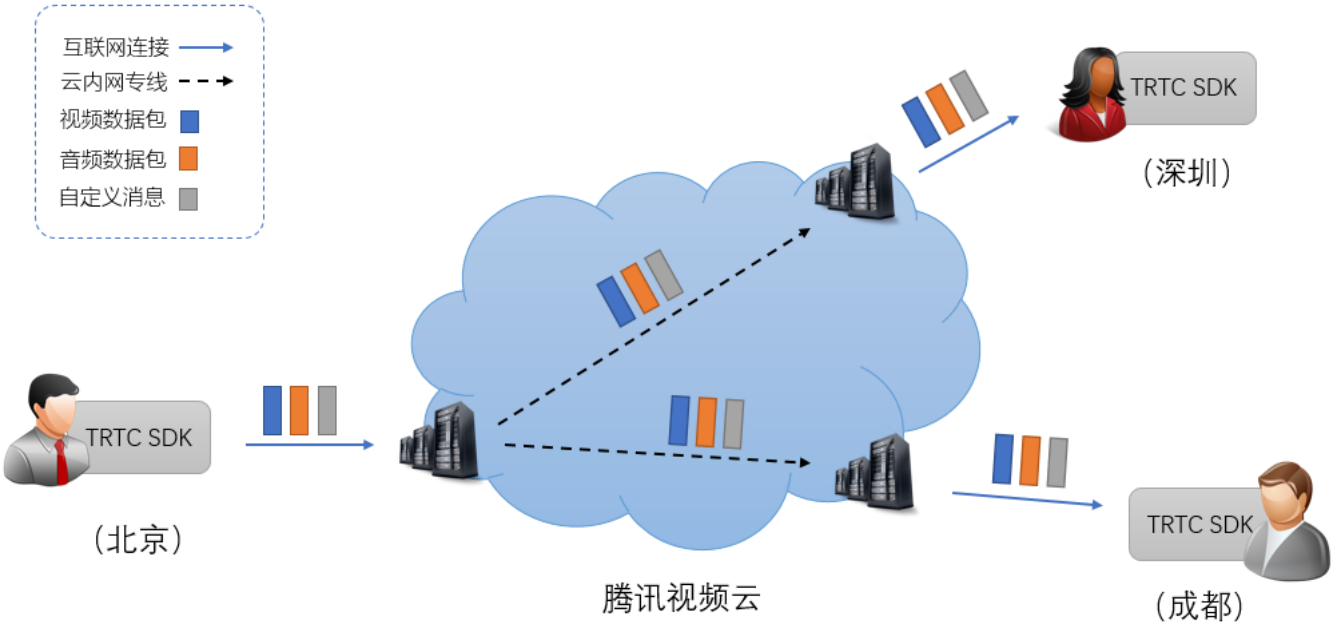
不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
✓	✓	✓	✓	✓	×	×

发送接收原理

某一个用户的自定义消息会被夹在音视频数据流中，随着音视频数据一起传输给房间里的其他用户。由于音视频线路本身并不是100%可靠的，为了提高可靠性，TRTC SDK 内部本身实现了一些可靠性保护机制。



消息发送

通过调用 TRTCCloud 的 sendCustomCmdMsg 接口发送的，发送时需要指定四个参数：

参数名	参数说明
cmdID	消息ID，取值范围为 1 ~ 10，不同业务类型的消息应当使用不同的 cmdID。
data	待发送的消息，最大支持 1KB（1000字节）的数据大小。
reliable	是否可靠发送，可靠发送的代价是会引入一定的延时，因为接收端要暂存一段时间的数据来等待重传。

参数名	参数说明
ordered	是否要求有序，即是否要求接收端接收的数据顺序和发送端发送的顺序一致，这会带来一定的接收延时，因为在接收端需要暂存并排序这些消息。

> ⚠ 注意：
> 请将 reliable 和 ordered 同时设置为 YES 或 NO，暂不支持交叉设置。

Objective-C

```
//发送自定义消息的示例代码
- (void)sendHello {
// 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例
NSInteger cmdID = 0x1;
NSData *data = [@"Hello" dataUsingEncoding:NSUTF8StringEncoding];
// reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
[trtcCloud sendCustomCmdMsg:cmdID data:data reliable:YES ordered:YES];
}
```

Java

```
//发送自定义消息的示例代码
public void sendHello() {
try {
// 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例
int cmdID = 0x1;
String hello = "Hello";
byte[] data = hello.getBytes("UTF-8");
// reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
trtcCloud.sendCustomCmdMsg(cmdID, data, true, true);

} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}
```

C++

```
// 发送自定义消息的示例代码
void sendHello()
{
// 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例
uint32_t cmdID = 0x1;
uint8_t* data = { '1', '2', '3' };
uint32_t dataSize = 3; // data的长度
// reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
trtcCloud->sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

C#

```
// 发送自定义消息的示例代码
private void sendHello()
{
// 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例
uint cmdID = 0x1;
byte[] data = { '1', '2', '3' };
uint dataSize = 3; // data的长度
```

```
// reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
mTRTCCloud.sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

消息接收

当房间中的一个用户通过 sendCustomCmdMsg 发出自定义消息后，房间中其他的用户可以通过 SDK 回调中的 onRecvCustomCmdMsg 接口来接收这些消息。

Objective-C

```
//接收和处理房间内其他人发送的消息
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UInt32)seq message:(NSData *)message
{
// 接收到 userId 发送的消息
switch (cmdId) // 发送方和接收方协商好的cmdId
{
case 0:
// 处理cmdId = 0消息
break;
case 1:
// 处理cmdId = 1消息
break;
case 2:
// 处理cmdId = 2消息
break;
default:
break;
}
}
```

Java

```
//继承 TRTCCloudListener，实现 onRecvCustomCmdMsg 方法接收和处理房间内其他人发送的消息
public void onRecvCustomCmdMsg(String userId, int cmdId, int seq, byte[] message) {
// 接收到 userId 发送的消息
switch (cmdId) // 发送方和接收方协商好的cmdId
{
case 0:
// 处理cmdId = 0消息
break;
case 1:
// 处理cmdId = 1消息
break;
case 2:
// 处理cmdId = 2消息
break;
default:
break;
}
```

C++

```
// 接收和处理房间内其他人发送的消息
void TRTCCloudCallbackImpl::onRecvCustomCmdMsg(
const char* userId, int32_t cmdId, uint32_t seq, const uint8_t* msg, uint32_t msgSize)
{
// 接收到 userId 发送的消息
switch (cmdId) // 发送方和接收方协商好的cmdId
{
```

```
case 0:
// 处理cmdId = 0消息
break;
case 1:
// 处理cmdId = 1消息
break;
case 2:
// 处理cmdId = 2消息
break;
default:
break;
}
}
```

C#

```
// 接收和处理房间内其他人发送的消息
public void onRecvCustomCmdMsg(string userId, int cmdId, uint seq, byte[] msg, uint msgSize)
{
// 接收到 userId 发送的消息
switch (cmdId) // 发送方和接收方协商好的cmdId
{
case 0:
// 处理cmdId = 0消息
break;
case 1:
// 处理cmdId = 1消息
break;
case 2:
// 处理cmdId = 2消息
break;
default:
break;
}
}
```

使用限制

由于自定义消息享受比音视频数据更高的传输优先级，如果自定义数据发送过多，音视频数据可能会被干扰到，从而导致画面卡顿或者模糊。所以，我们针对自定义消息的发送进行了如下的频率限制：

- 自定义消息会被云广播给房间内所有用户，所以每秒最多能发送 30 条消息。
- 每个消息包（即 data 的大小）最大为 1 KB，超过则很有可能会被中间路由器或者服务器丢弃。
- 每个客户端每秒最多能发送总计 8 KB 数据，也就是如果每个数据包都是 1KB，那么每秒钟您最多只能发送 8 个数据包。

自定义采集和渲染

最近更新时间：2022-03-04 11:00:25

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	–	–	–	✓	–	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

内容介绍

- **自定义视频采集**
  如果您自研（或者购买第三方）美颜和特效处理模块，则需要自己采集和处理摄像头拍摄画面，您可以通过 TRTCCloud 的 enableCustomVideoCapture 接口关闭音视频通话 TRTC SDK 自己的摄像头采集和图像处理逻辑。然后您可以使用 sendCustomVideoData 接口向音视频通话 TRTC SDK 填充您自己的视频数据。
- **自定义视频渲染**
  音视频通话 TRTC SDK 使用 OpenGL 进行视频画面的渲染，如果您是用在游戏开发中，或者需要在自己的界面引擎中嵌入音视频通话 TRTC SDK，那么就要自己渲染视频画面。
- **自定义音频采集**
  如果您是在特殊硬件设备上使用音视频通话 TRTC SDK，当需要外接声音采集设备并自己采集声音数据时，您可以通过 TRTCCloud 的 enableCustomAudioCapture 接口关闭音视频通话 TRTC SDK 默认的声音采集流程。然后您可以使用 sendCustomAudioData 接口向 TRTC SDK 填充您自己的声音数据。

  > ⚠ 注意：
  > 开启自定义音频采集后，有可能会导致回声抵消（AEC）的功能失效。

- **获取音频原数据**
  声音模块是一个高复杂度的模块，SDK 需要严格控制声音设备的采集和播放逻辑。在某些场景下，当您需要获取远程用户的音频数据或者需要获取本地麦克风采集到的音频数据时，可以通过音视频通话 TRTC SDK 提供的相应的回调接口来实现。

支持的平台

iOS	Android	Mac OS	Windows	微信小程序	Web 端	Flutter 端
✓	✓	✓	✓	×	✓（Web 端）	×

自定义视频采集

您可以通过 TRTCCloud 的 enableCustomVideoCapture 接口关闭 TRTC SDK 自己的摄像头采集和图像处理逻辑。然后您可以使用 sendCustomVideoData 接口向 TRTC SDK 填充您自己的视频数据。

在 sendCustomVideoData 接口中有一个叫 TRTCVideoFrame 的参数，它表示一帧视频画面。为了避免不必要的性能损失，对于输入音视频通话 TRTC SDK 的视频数据，在不同平台上有不同的格式要求，具体要求如下：

iOS 平台

TRTC SDK 支持 NV12 和 i420 两种 iOS 版本的 YUV 数据格式。在 iOS 平台上，比较高性能的图像传递方式是 CVPixelBufferRef，因此我们建议参数格式如下：

参数名称	参数类型	推荐取值	备注说明
pixelFormat	TRTCVideoPixelFormat	TRTCVideoPixelFormat_NV12	iOS 平台上摄像头原生采集出的视频格式即是 NV12。
bufferType	TRTCVideoBufferType	PixelBuffer	iOS 中原生支持的视频帧格式，性能最佳。
pixelBuffer	CVPixelBufferRef	如果 TRTCVideoBufferType 是 PixelBuffer，则需填写。	iPhone 摄像头采集的数据是 NV12 格式的 PixelBuffer。

参数名称	参数类型	推荐取值	备注说明
data	NSData*	如果 TRTCVideoBufferType 是 NSData，则需填写。	性能不如 PixelBuffer。
timestamp	uint64_t	0	可以填0，这样 SDK 会自定填充 timestamp 字段，但请**均匀**地控制 send
width	uint64_t	视频画面的宽度	请严格填写传入画面的像素宽度。
height	uint32_t	视频画面的高度	请严格填写传入画面的像素高度。
rotation	TRTCVideoRotation	不填写	• 默认不填写。 • 如果需要对画面进行旋转，可以填写TRTCVideoRotation_0、TRTCVideo TRTCVideoRotation_180、TRTCVideoRotation_270。SDK 会根据这 个竖屏画面，传入TRTCVideoRotation_90后，SDK 会旋转成横屏显示。

示例代码

在 Demo 文件夹中有一个叫做 LocalVideoShareViewController.m 的文件，它展示了如何从一个本地视频文件中读取出 NV12 格式的 PixelBuffer，并通过 SDK 进行后续处理。

```
//组装一个 TRTCVideoFrame 并将其送给 trtcCloud 对象
TRTCVideoFrame* videoFrame = [TRTCVideoFrame new];
videoFrame.bufferType = TRTCVideoBufferType_PixelBuffer;
videoFrame.pixelFormat = TRTCVideoPixelFormat_NV12;
videoFrame.pixelBuffer = imageBuffer;
videoFrame.rotation = rotation;
videoFrame.timestamp = timeStamp;

[trtcCloud sendCustomVideoData:videoFrame];
```

Android 平台

Android 平台有以下两种方案：

- **buffer 方案**：对接起来比较简单，但是性能较差，不适合分辨率较高的场景。
  buffer 方案要求直接向音视频通话 TRTC SDK 塞入 byte[] 格式的数组，支持 i420 和 NV21 两种 YUV 格式。

参数名称	参数类型	推荐取值	备注说明
pixelFormat	int	TRTC_VIDEO_PIXEL_FORMAT_I420 或 TRTC_VIDEO_PIXEL_FORMAT_NV21	i420 和 NV21 是两种不同的 YUV 数据格式。
bufferType	int	TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY	指明使用 data 的方式传递 YUV 数据。
texture	TRTCTexture	buffer 方案不填写	如果选择 i420 或者 NV21 两种格式，则此参数不填写。
data	byte[]	YUV 格式的数据 buffer	内部包装的是 Java 类型的内存，适合在 Java 层使用。
buffer	ByteBuffer	buffer 方案不填写	内部包装的是 C/C++ 类型的内存，适合在 JNI 层使用。
width	uint64_t	视频画面的宽度	请严格填写传入画面的像素宽度。
height	uint32_t	视频画面的高度	请严格填写传入画面的像素高度。
timestamp	long	视频帧的采集时间	可以填0，这样 SDK 会自定填充 timestamp 字段，但请**均匀**地控制 sendCustomVideoData 的调用间隔。
rotation	int	不填写	• 默认不填写。 • 如果需要对画面进行旋转，可以填写0、90、180、270。SDK 会根据这个值将视频顺时针旋转对应角度。例如一个竖屏画面，传入90后，SDK 会旋转成横屏显示。

- **texture 方案**：对接需要一定的 OpenGL 基础，但是性能较好，尤其是画面分辨率较高的时候。
  texture 需要向音视频通话 TRTC SDK 中传递 OpenGL 纹理，为了保证该方案能够正常运作，需要您提前设置好 OpenGL 环境，因此该方案的对接难度非常高。如果您没有学习过 OpenGL，建议直接使用我们提供的示例代码，它在 Demo 中的 mediashare 文件夹下，包含以下文件：

文件名	源码逻辑
LocalVideoShareActivity.java	演示如何通过 TRTCloud 的 sendCustomVideoData 函数向 SDK 投喂视频纹理以及 通过 TRTCloud 的 sendCustomAudioData 函数向 SDK 投喂音频数据。
Utils.java	封装了获取文件路径及媒体格式的函数。
helper	封装了 OpenGL 环境以及读取视频文件数据的函数，使其更好用一些。

> ⚠ 注意：
> 对于 640 x 360 以上的分辨率，我们建议使用 texture 方案，以避免过高的 CPU 使用率。

示例代码

LocalVideoShareActivity.java 中的代码原理比较复杂：

i. 代码首先初始化了 mVideoFrameReadListener 的成员变量，等待视频数据回调。
ii. 代码接下来使用了一个叫 MediaFileSyncReader 的模块，调用 start 方法从一个本地视频文件中读取出一帧帧的视频画面。
iii. 然后将视频数据通过 onFrameAvailable 回调，在这个回调函数里，我们将获得的 texture 通过 sendCustomVideoData 函数传递给 SDK：

```
public void onFrameAvailable(EGLContext eglContext, int textureId, int width, int height, long timestamp) {
TRTCCloudDef.TRTCVideoFrame videoFrame = new TRTCCloudDef.TRTCVideoFrame();
videoFrame.texture = new TRTCCloudDef.TRTCTexture();
videoFrame.texture.textureId = textureId;
videoFrame.texture.eglContext14 = eglContext;
videoFrame.width = width;
videoFrame.height = height;
videoFrame.timestamp = timestamp;
videoFrame.pixelFormat = TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D;
videoFrame.bufferType = TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE;

mTRTCCloud.sendCustomVideoData(videoFrame);
}
```

Windows 平台

在 Windows 平台上，仅支持 TRTCVideoPixelFormat_I420，我们建议参数格式如下：

参数名称	参数类型	推荐取值	备注说明
videoFormat	TRTCVideoPixelFormat	TRTCVideoPixelFormat_I420	仅支持TRTCVideoPixelFormat_I420。
bufferType	TRTCVideoBufferType	TRTCVideoBufferType_Buffer	仅支持 TRTCVideoBufferType_Buffer。
textureId	int	不需要填写	Windows平台不支持纹理
data	char *	视频帧 buffer	一帧I420数据。
timestamp	uint64_t	0	可以填0，这样 SDK 会自定填充 timestamp 字段，但请均匀地控制 sen
width	uint64_t	视频画面的宽度	请严格填写传入画面的像素宽度。
height	uint32_t	视频画面的高度	请严格填写传入画面的像素高度。
rotation	TRTCVideoRotation	画面旋转角度	• 默认不填写。 • 如果需要对画面进行旋转，可以填写TRTCVideoRotation_0、TRTCVide TRTCVideoRotation_180、TRTCVideoRotation_270。SDK 会根据这个 个竖屏画面，传入TRTCVideoRotation_90后，SDK 会旋转成横屏显示。

示例代码

在 Demo 文件中有一个叫做 sendCustomVideoFrame 的函数，它展示了如何从一个本地视频文件中读取出 I420 格式的 buffer，并通过 SDK 进行后续处理。

```
//组装一个 TRTCVideoFrame 并将其送给 trtcCloud 对象
TRTCVideoFrame frame;
frame.videoFormat = LiteAVVideoPixelFormat_I420;
frame.length = bufferSize;
frame.data = _video_buffer;
frame.width = _video_width;
frame.height = _video_height;
m_pCloud->sendCustomVideoData(TRTCVideoStreamTypeBig, & frame);
```

自定义视频渲染

音视频通话 TRTC SDK 使用 OpenGL 进行视频画面的渲染，如果您是用在游戏开发中，或者需要在自己的界面引擎中嵌入音视频通话 TRTC SDK，那么就要自己渲染视频画面。

iOS 平台（包括 Mac）

通过 TRTCCloud 的 setLocalVideoRenderDelegate 和 setRemoteVideoRenderDelegate 可以设置本地和远程画面的自定义渲染回调，相关参数如下：

参数名称	参数类型	推荐取值	备注说明
pixelFormat	TRTCVideoPixelFormat	TRTCVideoPixelFormat_NV12	-
bufferType	TRTCVideoBufferType	TRTCVideoBufferType_PixelBuffer	iOS 中原生支持的视频帧格式，性能最佳。

示例代码

如果 pixelFormat 选择了 TRTCVideoPixelFormat_NV12，bufferType 选择了 TRTCVideoBufferType_PixelBuffer，那么整个您可以很方便地将一帧 NV12 格式的 PixelBuffer 转成一副视频画面。在 Demo 文件夹中有一个叫做 TestRenderVideoFrame.m 的文件，其中就用如下的示例代码展示了如何使用该方法。

```
- (void)onRenderVideoFrame:(TRTCVideoFrame *)frame
userId:(NSString *)userId
streamType:(TRTCVideoStreamType)streamType
{
//userId是nil时为本地画面，否则为远端画面
CFRetain(frame.pixelBuffer);
__weak __typeof(self) weakSelf = self;
dispatch_async(dispatch_get_main_queue(), ^{
TestRenderVideoFrame* strongSelf = weakSelf;
UIImageView* videoView = nil;
if (userId) {
videoView = [strongSelf.userVideoViews objectForKey:userId];
}
else {
videoView = strongSelf.localVideoView;
}
videoView.image = [UIImage imageWithCIImage:[CIImage imageWithCVImageBuffer:frame.pixelBuffer]];
videoView.contentMode = UIViewContentModeScaleAspectFit;
CFRelease(frame.pixelBuffer);
});
}
```

Android 平台

通过 TRTCCloud 的 setLocalVideoRenderListener 和 setRemoteVideoRenderListener 可以设置本地和远程画面的自定义渲染回调，相关参数如下：

参数名称	参数类型	推荐取值	备注说明
pixelFormat	TRTCVideoPixelFormat	TRTC_VIDEO_PIXEL_FORMAT_NV21	自定义渲染暂时不支持 texture 方案。

参数名称	参数类型	推荐取值	备注说明
bufferType	TRTCVideoBufferType	TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY 或 TRTC_VIDEO_BUFFER_TYPE_BYTE_BUFFER	BYTE_BUFFER 适合在 jni 层使用， BYTE_ARRAY 则可用于 Java 层的直接操作。

示例代码

```
public void onRenderVideoFrame(String userId, int streamType, final TRTCCloudDef.TRTCVideoFrame frame) {
mEglCore.makeCurrent();
GLES20.glViewport(0, 0, mSurfaceSize.width, mSurfaceSize.height);
GLES20.glBindFramebuffer(GLES20.GL_FRAMEBUFFER, 0);
GLES20.glClearColor(0, 0, 0, 1.0f);
GLES20.glClear(GLES20.GL_DEPTH_BUFFER_BIT | GLES20.GL_COLOR_BUFFER_BIT);
mNormalFilter.onDraw(frame.texture.textureId, mGLCubeBuffer, mGLTextureBuffer);
mEglCore.swapBuffer();
}
```

Windows 平台

通过 TRTCCloud 的 setLocalVideoRenderCallback 和 setRemoteVideoRenderCallback 可以设置本地和远程画面的自定义渲染回调，相关参数如下：

参数名称	参数类型	推荐取值	备注说明
pixelFormat	TRTCVideoPixelFormat	TRTCVideoPixelFormat_BGRA32	支持回调 TRTCVideoPixelFormat_I420 或 TRTCVideoPixelFormat_BGRA32 像素格式的视频帧。
bufferType	TRTCVideoBufferType	TRTCVideoBufferType_Buffer	目前只支持 TRTCVideoBufferType_Buffer。
callback	ITRTCVideoRenderCallback*	ITRTCVideoRenderCallback*	自定义渲染回调。

示例代码

如果 pixelFormat 选择了 TRTCVideoPixelFormat_BGRA32，bufferType 选择了 TRTCVideoBufferType_Buffer，那么整个您可以很方便地将一帧 BGRA32 格式的 PixelBuffer 转成一幅视频画面。在 Demo 中有一个叫做 TXLiveAvVideoView.cpp 的文件，其中就用如下的示例代码展示了如何使用该方法。

```
void TXLiveAvVideoView::renderFitMode(HDC hDC, unsigned char * buffer, int width, int height, RECT& rcImage)
{
Point origin;
origin.X = m_rcItem.left, origin.Y = m_rcItem.top;
int viewWith = m_rcItem.right - m_rcItem.left;
int viewHeight = m_rcItem.bottom - m_rcItem.top;

bool bReDrawBg = false;

if (m_bmi.bmiHeader.biWidth != width || m_bmi.bmiHeader.biHeight != height)
{
memset(&m_bmi, 0, sizeof(m_bmi));
m_bmi.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
m_bmi.bmiHeader.biWidth = width;
m_bmi.bmiHeader.biHeight = height;
m_bmi.bmiHeader.biPlanes = 1;
m_bmi.bmiHeader.biBitCount = 32;
m_bmi.bmiHeader.biCompression = BI_RGB;
bReDrawBg = true;
}
//开始处理渲染
::SetStretchBltMode(hDC, COLORONCOLOR);
if (bReDrawBg)
::PatBlt(hDC, 0 + origin.X, 0 + origin.Y, viewWith, viewHeight, BLACKNESS);
::StretchDIBits(hDC, x + origin.X, y + origin.Y, width, height, 0, 0, width, height, m_argbRenderFrame.frameBuf, &m_bmi, DIB_RGB_COLORS, S
```

```
RCCOPY);
}
```

自定义音频采集

您可以通过 TRTCCloud 的 enableCustomAudioCapture 接口关闭 TRTC SDK 默认的声音采集流程。然后您可以使用 sendCustomAudioData 接口向音视频通话 TRTC SDK 填充您自己的声音数据。

在 sendCustomAudioData 接口中有一个叫 TRTCAudioFrame 的参数，它表示一帧 20ms 长度的音频数据：

- 通过 sendCustomAudioData 向 SDK 投送的数据必须是 PCM 格式的未经压缩的音频裸数据，不可以是 AAC 或者其他的压缩格式。
- sampleRate 和 channels 分辨代表声音采样率和声道数，请保持跟传入的 PCM 数据严格一致。
- 每帧音频数据的时长推荐是20ms，我们可以计算一下，如果 sampleRate 为48000，声道数是1（单声道），那么每次调用 sendCustomAudioData 时传入的 buffer 长度应该是：48000 * 0.02s * 1 * 16bit = 15360bit = 1920字节。
- timestamp 可以为0，当 timestamp 为0时，SDK 会自动填充音频时间戳。因此，为了确保音频时间戳的稳定，请**均匀**地调用 sendCustomAudioData，保持20ms一次的频率，否则会导致声音出现断断续续的效果。

> 注意：
> 使用 sendCustomAudioData 有可能会导致回声抵消（AEC）的功能失效。

获取音频原数据

声音模块是一个高复杂度的模块，SDK 需要严格控制声音设备的采集和播放逻辑。在某些场景下，当您需要获取远程用户的音频数据或者需要获取本地麦克风采集到的音频数据时，可以通过 TRTCCloud 对应的不同平台的接口，来给 SDK 集成三个回调函数。

> TRTCCloud 对应的不同平台的接口分别为：
> - **iOS**：setAudioFrameDelegate。
> - **Android**：setAudioFrameListener。
> - **Windows**：setAudioFrameCallback。

接口	说明
onCapturedAudioFrame	获取本地麦克风采集到的音频源数据。在非自定义采集模式下，SDK 会负责麦克风的声音采集工作，但您可能也需要拿到 SDK 采集到的声音源数据，通过此回调函数就可以获取。
onPlayAudioFrame	该函数会回调每一路远程用户的声音数据，这是混音前的数据。如果您要对某一路的语音进行语音识别，就可以用到这个回调。
onMixedPlayAudioFrame	各路音频数据混合后，在送到扬声器播放之前，会通过此函数回调出来。

> 注意：
> - 不要在上述回调函数中做任何耗时操作，建议直接拷贝，并通过另一线程进行处理，否则会导致声音断断续续或者回声抵消（AEC）失效的问题。
> - 上述回调函数中回调出来的数据都只允许读取和拷贝，不能修改，否则会导致各种不确定的后果。

Mac 端分享系统声音

最近更新时间：2021-12-23 17:07:59

版本支持

本页文档所描述功能，在腾讯云视立方中支持情况如下：

版本名称	基础直播 Smart	互动直播 Live	短视频 UGSV	音视频通话 TRTC	播放器 Player	全功能
支持情况	-	-	-	✓	-	✓
SDK 下载	下载	下载	下载	下载	下载	下载

不同版本 SDK 包含的更多能力，具体请参见 SDK 下载。

场景痛点及解决方案

在屏幕分享等应用场景中，常需要共享系统音频给对方，而 Mac 电脑默认声卡不支持采集系统音频，所以在 Mac 电脑上共享系统音频比较困难。基于此，音视频通话 TRTC 提供了在 Mac 端录制系统音频的功能来满足该场景需求，具体接入步骤见下文。

集成说明

步骤1：集成 TRTCPrivilegedTask 库

SDK 需要使用 TRTCPrivilegedTask 库来获取 root 权限，从而将虚拟声卡插件 TRTCAudioPlugin.driver 安装至系统目录 /Library/Audio/Plug-Ins/HAL 。

使用CocoaPods集成

1. 打开您当前项目根目录下的 Podfile 文件，添加下面的内容：

```
platform :osx, '10.10'

target 'Your Target' do
pod 'TRTCPrivilegedTask', :podspec => 'https://pod-1252463788.cos.ap-guangzhou.myqcloud.com/liteavsdkspec/TRTCPrivilegedTask.podspec'
end
```

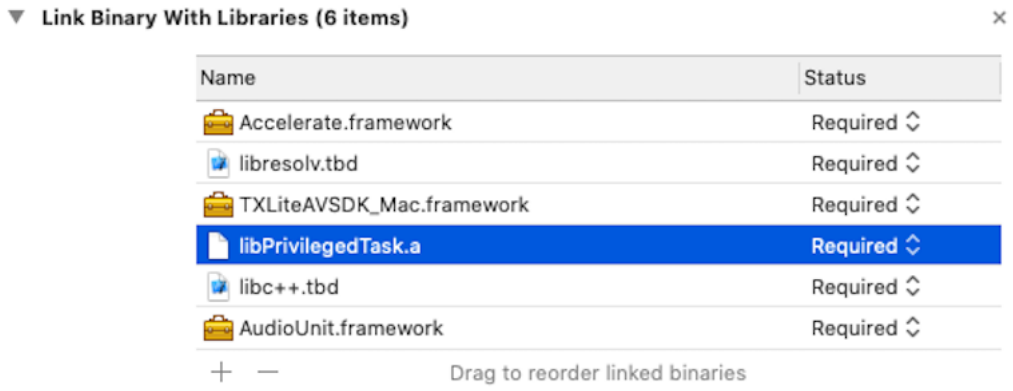
2. 执行 pod install 命令安装 TRTCPrivilegedTask 库。

> 说明：
>
> - 如果项目根目录下没有 Podfile 文件，请先执行 pod init 命令新建文件再添加以下内容。
> - CocoaPods 的安装方法，请参见 CocoaPods 官网安装说明。

手动集成

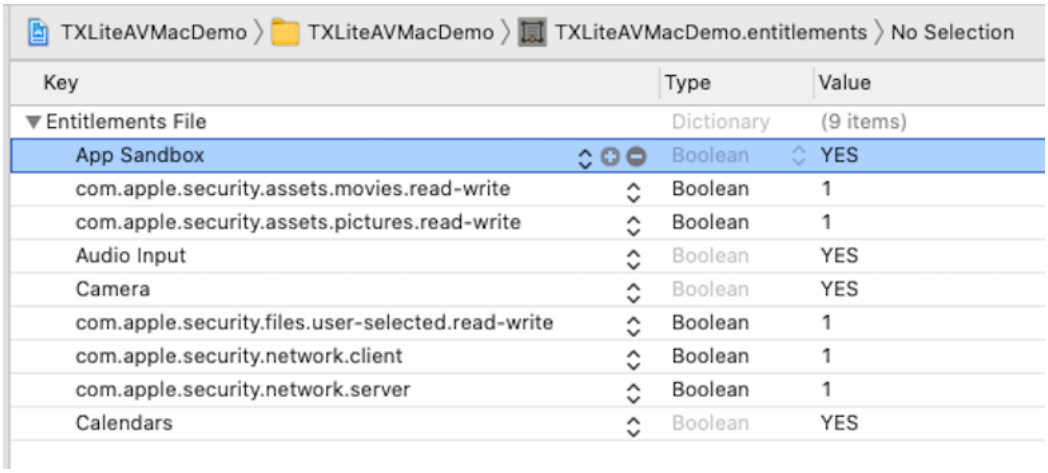
1. 下载 TRTCPrivilegedTask 库。
2. 打开您的 Xcode 工程项目，导入解压后的文件 libPrivilegedTask.a 到您的工程。

3. 选择要运行的 target，选中 Build Phases 项，展开Link Binary with Libraries 项，单击底下的+，添加依赖库 libPrivilegedTask.a。



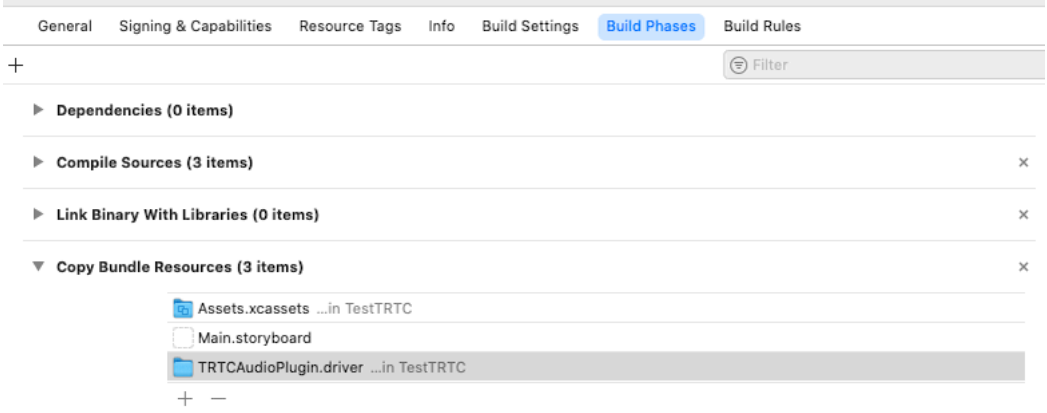
步骤2：取消 App Sandbox 功能

在 App 的 entitlements 描述文件中，删除 **App Sandbox** 条目。

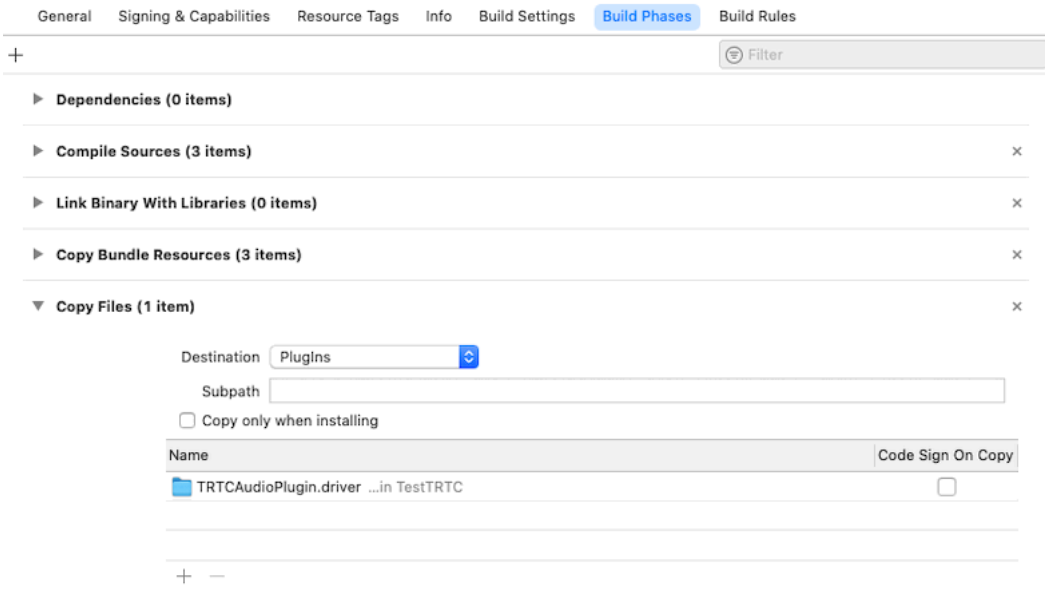


步骤3：虚拟声卡插件打包

在 集成 TRTCPrivilegedTask 库 和 取消 App Sandbox 功能 后，首次使用系统音频录制功能时，SDK 会从网络下载虚拟声卡插件并安装。如果想加速这个过程，可以将放置在 TXLiteAVSDK_TRTC_Mac.framework 的 PlugIns 目录下的虚拟声卡插件 TRTCAudioPlugin.driver 打包至 App Bundle 的 Resources 目录。如下图：



或者拷贝至 App Bundle 的 PlugIns 目录。如下图：



步骤4：开始系统声音采集

调用 startSystemAudioLoopback 接口开始系统声音采集，并将其混入上行音频流中，接口执行完成会通过 onSystemAudioLoopbackError 回调成功或失败的结果。

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud startLocalAudio];
[trtcCloud startSystemAudioLoopback];
```

⚠ 注意：

集成 TRTCPrivilegedTask 库和取消 App Sandbox 功能后，首次调用 startSystemAudioLoopback 会获取 root 权限。如下图：

在用户单击好后，开始自动安装虚拟声卡插件。

步骤5：停止系统声音采集

调用 stopSystemAudioLoopback 接口停止系统声音采集。

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud stopSystemAudioLoopback];
```

步骤6：设置系统声音采集音量

调用 setSystemAudioLoopbackVolume 接口设置系统声音的采集音量。

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud setSystemAudioLoopbackVolume:80];
```

集成小结

- 音视频通话TRTC 在 Mac 端是通过虚拟声卡插件 TRTCAudioPlugin.driver 来录制系统音频。这个虚拟声卡插件需要拷贝至系统目录 /Library/Audio/Plug-Ins/HAL 并重启音频服务后生效。可以通过 启动台 的 其他 文件夹中 音频 MIDI 设置 应用来检查虚拟声卡插件是否安装成功。在该应用的设备列表中，有名称为 “TRTC Audio Device” 的设备即表明 TRTC 的虚拟声卡插件安装成功。
- 上述步骤中的 集成 TRTCPrivilegedTask 库 和 取消 App Sandbox 功能 是为 TRTC SDK 自动安装虚拟声卡插件提供 root 权限。如不集成 TRTCPrivilegedTask 库并保留 App Sandbox 功能， SDK 不会自动安装自动安装虚拟声卡插件，但如果系统中已安装好虚拟声卡插件，录制系统音频的功能仍然可以正常使用。

> 说明：
> 除上述方案外，也可以手动安装虚拟声卡插件来集成该功能：
>
> i. 将放置在 TXLiteAVSDK_TRTC_Mac.framework 的 PlugIns 目录下的 TRTCAudioPlugin.driver 拷贝至系统目录 /Library/Audio/Plug-Ins/HAL。
> ii. 重启系统的音频服务。
>
> ```
> sudo cp -R TXLiteAVSDK_TRTC_Mac.framework/PlugIns/TRTCAudioPlugin.driver /Library/Audio/Plug-Ins/HAL
> sudo kill -9 `ps ax|grep 'coreaudio[a-z]' |awk '{print $1}'`
> ```

注意事项

- App Sandbox 功能取消后，App 内获取到的用户路径会发生变化。
  通过 NSSearchPathForDirectoriesInDomains 等系统方法获取到的 ~/Documents、 ~/Library 等目录会从沙盒目录切换成用户目录 /Users/用户名/Documents、 /Users/用户名/Library。
- 集成 TRTCPrivilegedTask 库，可能会使 App 无法上架到 Mac App Store。
  SDK 自动安装虚拟声卡插件时需要关闭 App Sandbox 功能，并获取 root 权限，App 可能会无法上架到 Mac App Store。详情请参见 App Store Review Guidelines 。
  如需上架 App Store 或者使用 Sandbox 功能，建议选择手动安装虚拟音频插件的方案。

新版连麦方案

最近更新时间：2022-03-17 16:10:50

连麦方案升级说明

直播连麦方案现已升级，腾讯云视立方为用户提供**基于实时音视频 TRTC 能力实现的新版连麦方案**。老用户仍可继续使用旧版连麦方案（RTMP_ACC），也可选择快速切换至新版连麦方案（RTC），接入指引请参见 RTC观众连麦。方案对比如下：

对比项	旧版连麦方案（RTMP_ACC）	新版连麦方案（RTC）
协议	RTMP 基于 TCP 协议	RTC 基于 UDP 协议（更适合流媒体传输）
QoS	弱网抗性能力弱	50%丢包率可正常视频观看，70%丢包率可正常语音连麦
支持区域	仅支持中国内地（大陆）地区	全球覆盖
使用产品	需开通移动直播、云直播服务	需开通移动直播、云直播、实时音视频服务
价格	0.016元/分钟	阶梯价格，详情请参见 RTC 连麦方案怎么计算费用

新版连麦方案特性

新版连麦方案（RTC）提供了更加简单灵活的全新接口：V2TXLivePusher （推流）、V2TXLivePlayer （拉流），同时在互动直播场景下的延时特性上有较大提升，主播连麦的延时 < 200ms，主播和观众的延时在 100ms – 1000ms。适用于更加灵活、更低延时、更多人数的直播互动场景。

在开播端，基于实时音视频 TRTC 来实现，通过 TRTC 协议完成主播推流。仅需1个TRTC应用 + 1个推流域名，即可把主播的直播流推流至云端。在观众端，默认情况下，使用 CDN 方式进行拉流观看， CDN 观看费用较低。如果主播端有 PK 需求，直接互相播放对方的流即可。RTC PK 需要另外开通服务，具体步骤请参见 配置连麦或 PK 能力 配置。

体验新版连麦方案

为了更好地服务用户，我们在 音视频终端 SDK 控制台 整理推出新版连麦管理功能，通过简单配置即可快速跑通 MLVB-API-Example Demo，体验新版连麦方案，全方位管理连麦应用，实时查看连麦应用相关用量统计，以及辅助用户快速生成可用于新版连麦方案的 TRTC 推拉流地址和 CDN 播放地址。

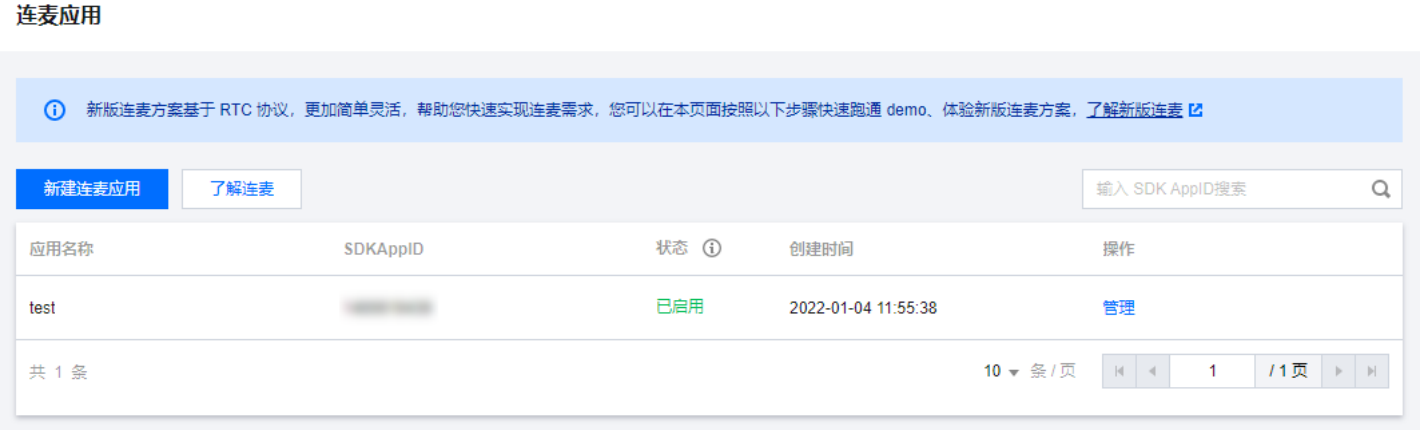
腾讯云视立方将新版连麦管理的多个功能集成至一个控制台，便于用户快捷使用，满足使用场景的实现。具体分为 快速上手、连麦应用、用量统计 和 地址生成器 四个功能页面。

快速上手

快速上手嵌合移动直播终端组件，通过在线下载 SDK 源码，简单配置 License、连麦应用以及域名后，即可在线快速跑通 Demo，体验观众连麦和主播 PK 两类典型互动直播场景。详细操作指引请参见 快速上手。

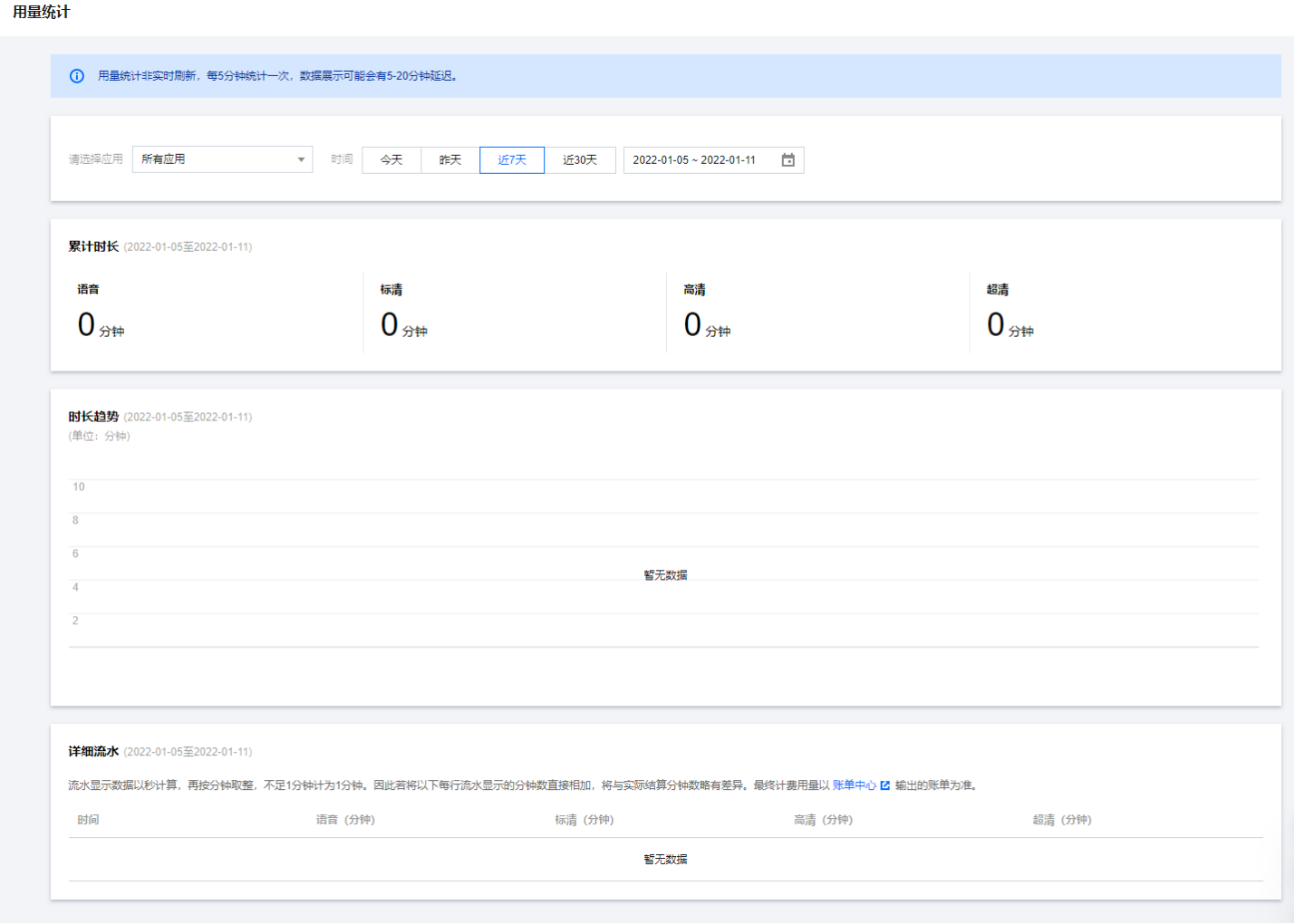
连麦应用

基于新版连麦方案，提供连麦应用管理功能，包含连麦场景所必须的所有功能，方便您在一个控制台即可完成操作，您可在此新建连麦应用并进行应用信息的查看和管理、查看用量统计概览、SDK 下载、快速设置 CDN 播放以及混流和录制相关功能。详细操作指引请参见 连麦应用。



用量统计

腾讯云视立方提供连麦用量统计查询功能，便于您在本页面查看连麦应用的相关用量及详细流水信息。详细操作指引请参见 用量统计。



地址生成器

为了便于您快速使用新版连麦方案，我们提供快速生成可用的 TRTC 推/拉流地址和 CDN 播放地址的工具。与云直播控制台的地址生成器不同，您可根据业务需求，在连麦管理的地址生成器中选择观众连麦 or 主播 PK 场景，填写配置信息后快速一键生成该场景的 TRTC 推拉流地址和各协议的 CDN 播放地址。同时提供配套的场景解析图供您了解连麦原理并按步骤完成 SDK 接入。详细操作指引请参见 地址生成器。

地址生成器

您可在本地址生成器中，快速获取用于观众连麦/主播 PK 连麦场景的 trtc 推/拉流地址和 CDN 播放地址。

观众连麦 | 主播PK

选择连麦应用 * test

主播 Stream Id * 请设置自定义流ID
仅支持英文字母、数字和符号

主播 User Id * 请设置自定义UserId
仅支持英文字母、数字和符号

连麦观众 Stream Id * 请设置自定义流ID
仅支持英文字母、数字和符号

连麦观众 User Id * 请设置自定义UserId
仅支持英文字母、数字和符号

选择域名 * 请选择

App Name live
默认为 live，支持自定义设置，可输入英文字母、数字和符号

有效时间 * 2022-01-12 09:20

生成地址 拼接地址说明