实时音视频

高级功能

产品文档

腾讯云

文档目录

高级功能
云端混流转码

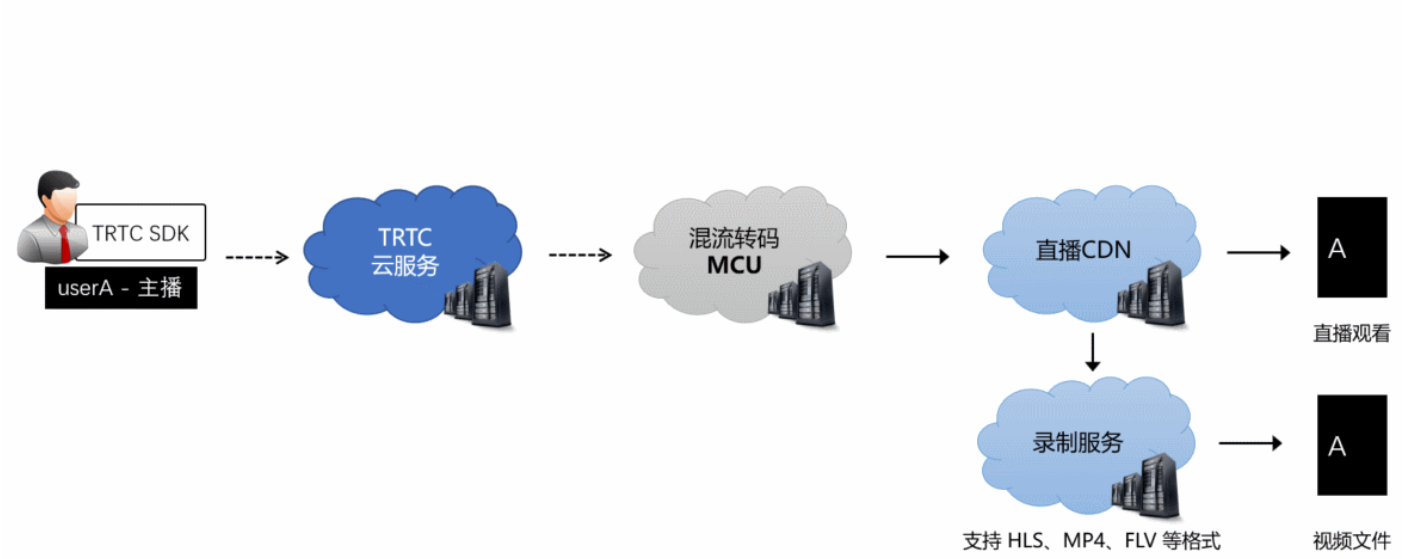
最近更新时间：2022-03-15 10:48:45

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：

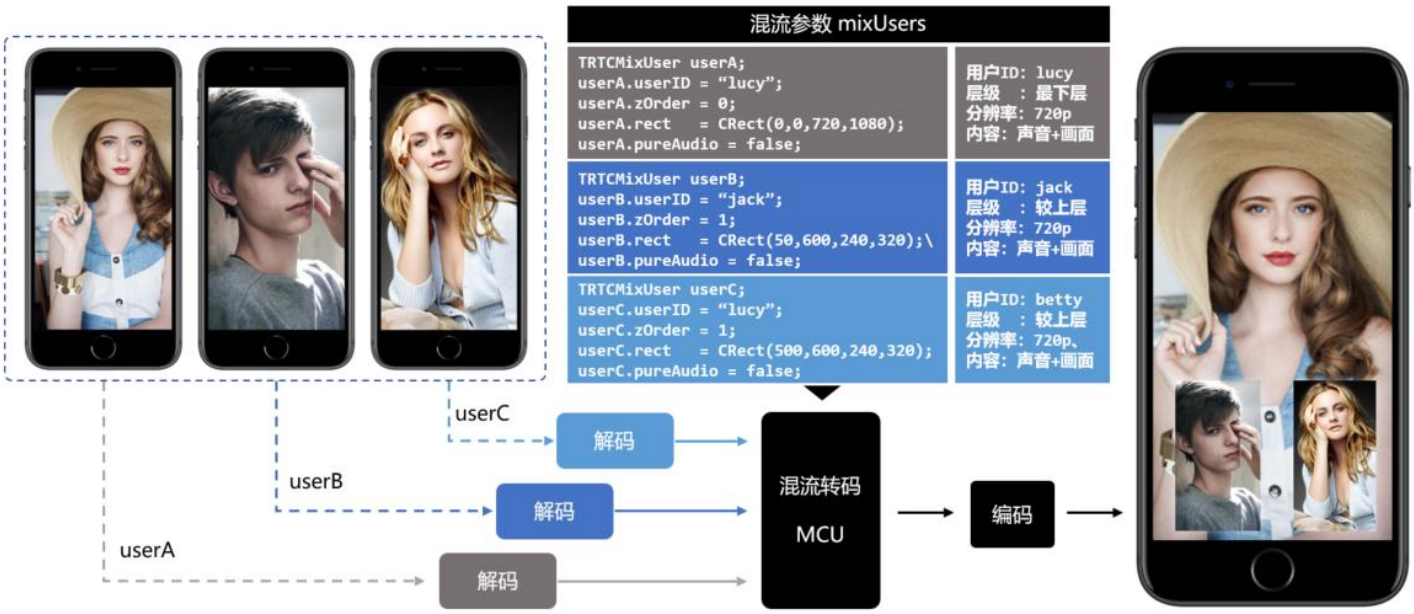


> **注意：**
> 方案二支持 iOS、Android、Windows、Mac、Electron、Flutter 和 Web 平台的 SDK，如果您希望在微信小程序上也实现混流功能，请使用方案一。

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）。



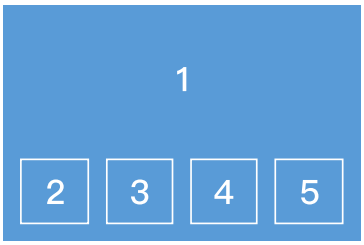
方案一：服务端 REST API 混流方案

启动混流

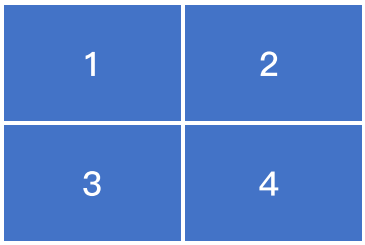
由您的服务器调用 REST API StartMCUMixTranscode 可以启动云端混流，对于此 API 您需关注如下细节：

步骤1：设置画面排版模式（必需）

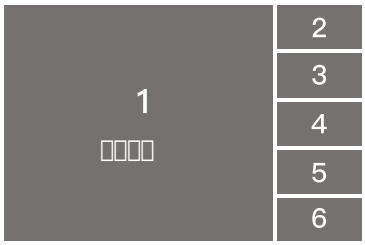
通过 StartMCUMixTranscode 中的 LayoutParams 参数，可以设置如下几种排版模式：



悬浮模板
LayoutParams.Template=0

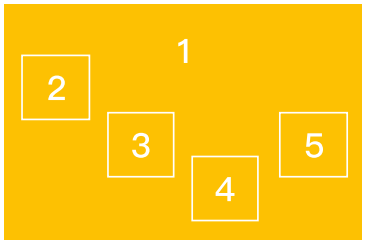


九宫格模板
LayoutParams.Template=1



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

名称	描述	推荐值
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 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

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）
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

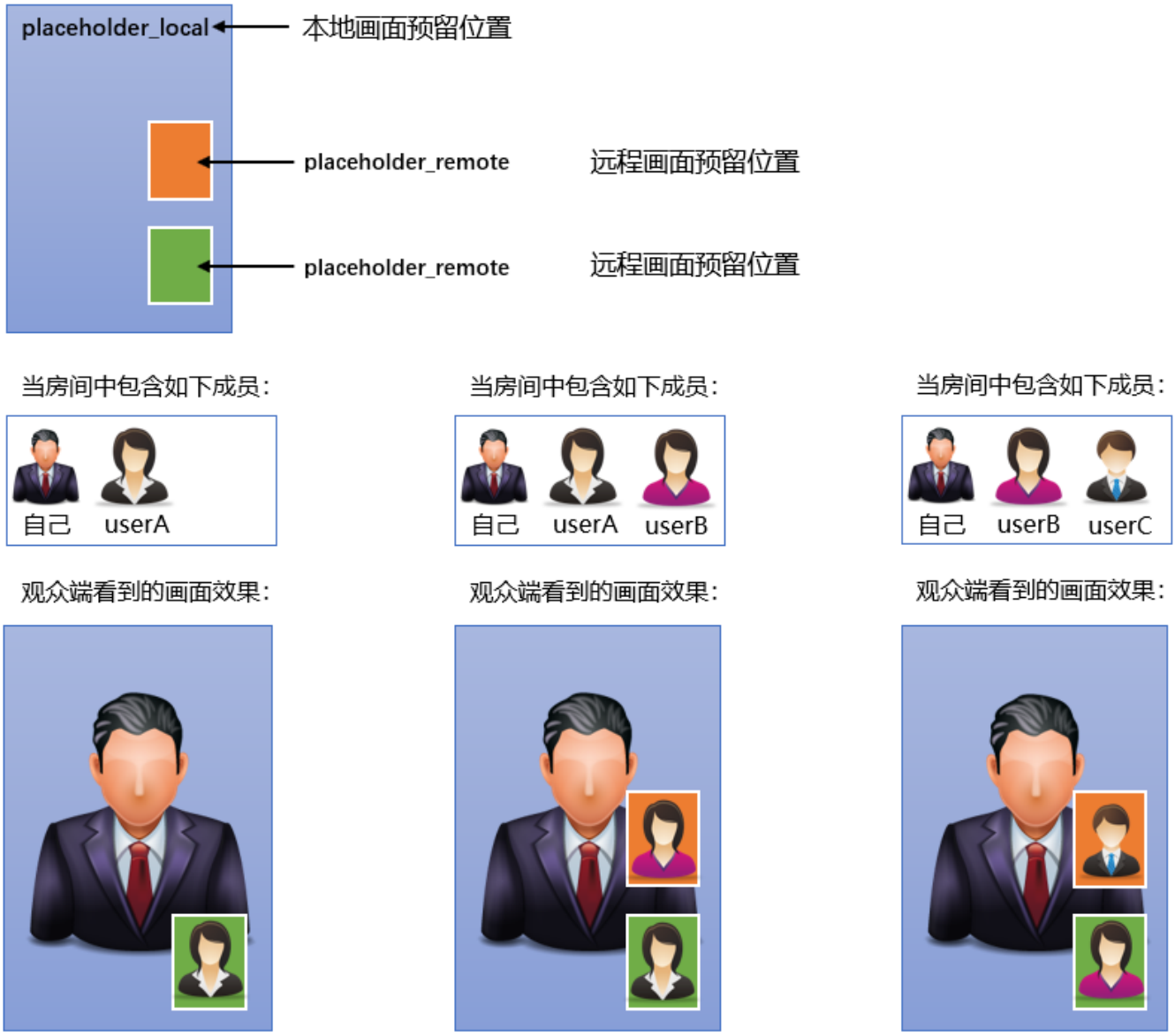
1. 在调用 enterRoom() 函数进入房间时，根据您的业务需要，设定 AppScene 参数为 TRTCAppSceneVideoCall 或 TRTCAppSceneLIVE。
2. 开启 旁路直播，并设定 TRTCParams 中的 streamId 参数，指定 MCU 输出的混合音频流的去处。
3. 调用 startLocalPreview() 和 startLocalAudio() 开启本地的音视频上行。

> 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。
   >?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 config 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。

实现云端录制与回放

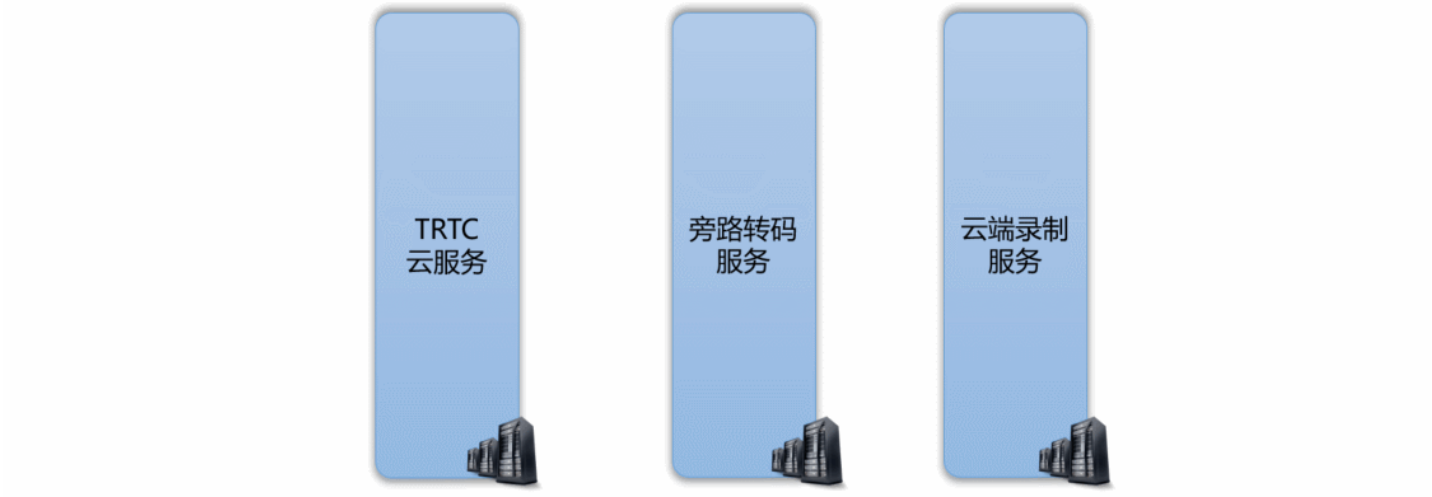
最近更新时间：2022-07-01 15:43:51

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。

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

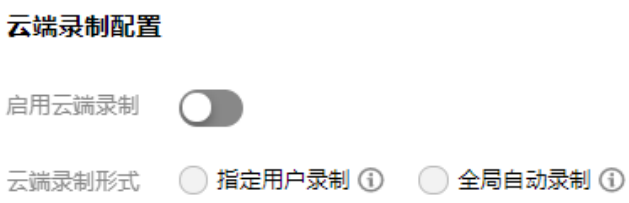
开通录制服务

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

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

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?

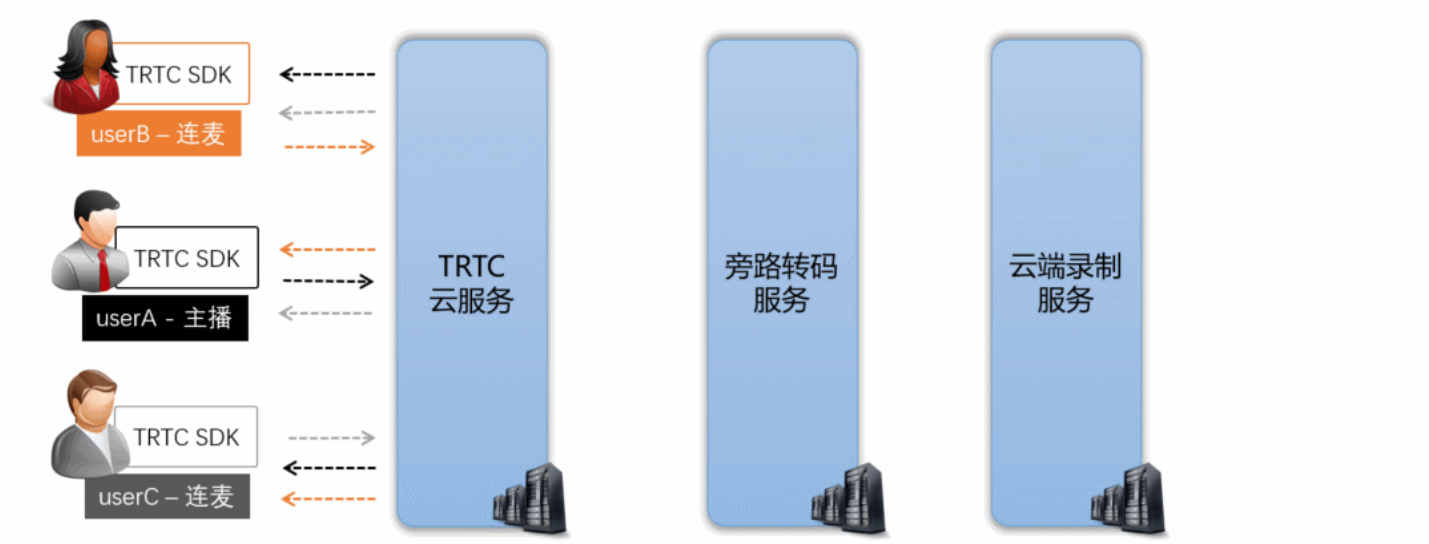
方案一：全局自动录制

- **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 、Web 等平台，暂不支持微信小程序。
- **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 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、Web 等终端发起录制控制，暂不支持微信小程序端发起控制。

方案三：指定用户录制（REST API）

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=ecd19683f6cfla7615f13670e0834de1&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。

> 说明：
> 更多回调字段说明，请参见 云直播-录制事件通知。

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

云端录制与回放功能使用到的功能包括：云服务资源（包括云端录制服务、云点播的回放文件存储与处理、云点播的播放服务等），和终端 SDK 播放点播视频的能力。可能会根据实际需求产生以下费用。

云服务消耗

云服务消耗包括了云端录制产生的费用和回放文件产生的消耗费用。您可根据实际需求进行使用。

> 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。

- **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，具体费用计算请参见 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 。

- **回放文件存储费用：云端视频存储服务产生的存储费用**
  录制出的视频文件存放于云点播服务，会使用云点播的云端存储服务。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 。

- **观看回放费用：云点播视频进行分发播放产生的播放费用**
  如果您录制的视频文件要被用于回看播放，会使用云点播的 CDN 播放功能。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。

SDK 播放授权

音视频通话（TRTC）全功能版本 SDK 提供了功能全面性能强大的视频播放能力，可轻松配合云点播实现视频播放功能。移动端 SDK 在10.1及以上的版本可通过获取指定 License 以解锁视频播放能力。

> ⚠ 注意：
> TRTC 的播放能力无需 License 授权。

您可直接 购买视频播放 License，或通过 购买的云点播流量包 免费获赠视频播放 License 或 短视频 License，两种 License 均可用于解锁 SDK 的视频播放功能。并且点播资源包可以抵扣云点播的播放产生的日结流量，详细说明请参见 云点播预付费资源包。
License 计费说明参见 腾讯云视立方 License，License 购买完成后可参考 License 操作指引 进行新增和续期等操作。

相关问题

实时音视频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。

实现 CDN 直播观看

最近更新时间：2022-05-30 19:53:04

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。

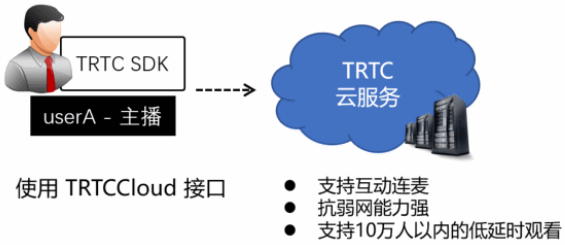
给 TRTC 对接 CDN 观看，一般被用于解决如下两类问题：

- **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

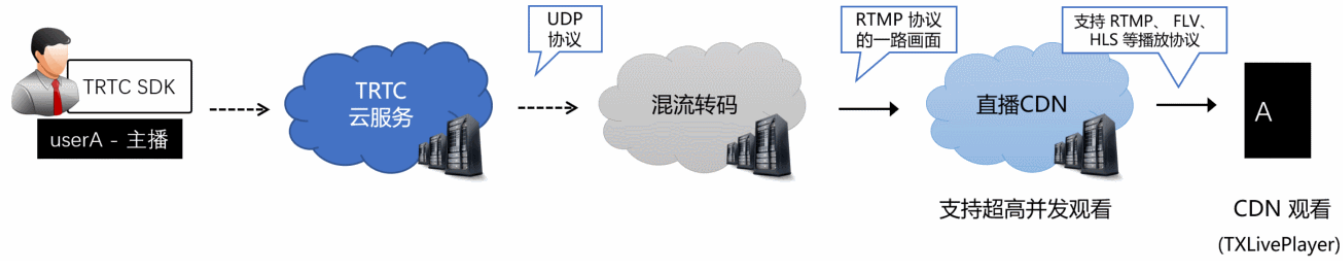
腾讯云会使用一批旁路转码集群，将 TRTC 中的音视频数据旁路到直播 CDN 系统中，该集群负责将 TRTC 所使用的 UDP 协议转换为标准的直播 RTMP 协议。

单路画面的旁路直播
当 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
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：



说明：

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。

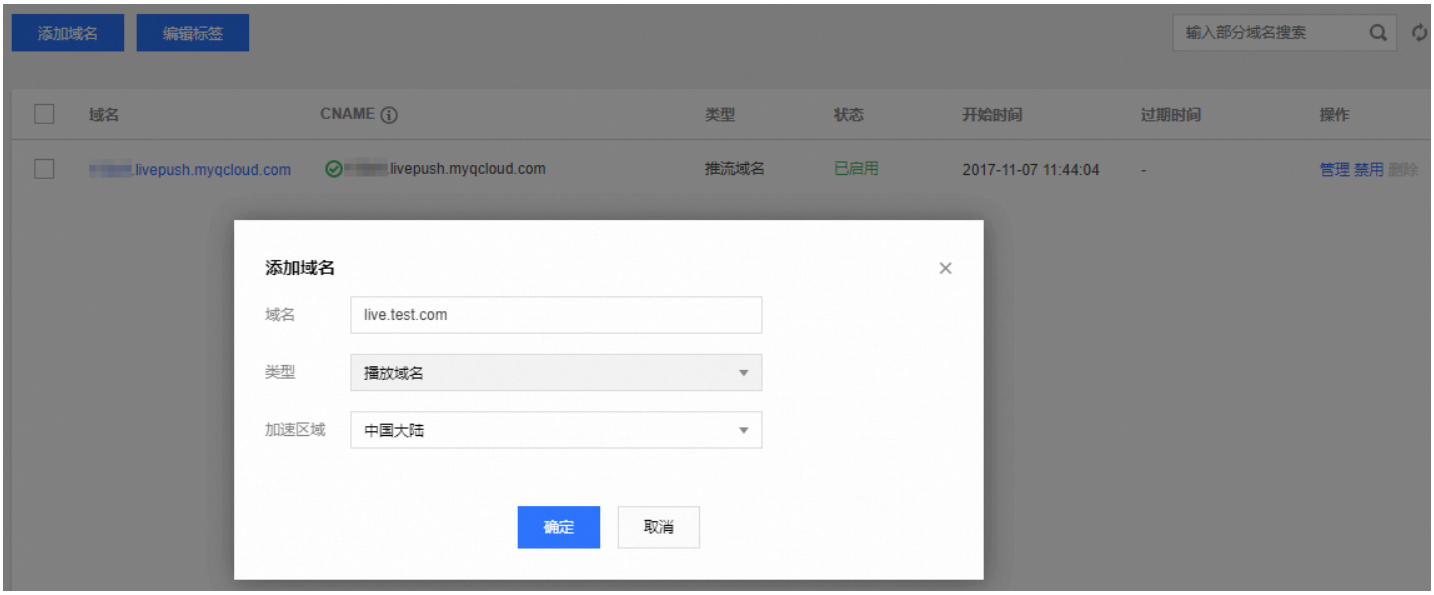
使用步骤

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 如何计算及使用 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：给 SDK 配置 License 授权

1. 获取 License 授权：

- 若您已获得相关 License 授权，需在 云直播控制台 获取 License URL 和 License Key。

- 若您暂未获得 License 授权，需先参考 新增与续期 License 进行申请。

2. 在您的 App 调用 SDK 相关功能之前（建议在 Application / - [AppDelegate application:didFinishLaunchingWithOptions:] 中）进行如下设置：**Android**

```
public class MApplication extends Application {

@Override
public void onCreate() {
super.onCreate();
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
@Override
public void onLicenceLoaded(int result, String reason) {
Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);
}
});
}
```

iOS

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOption
s {
NSString * const licenceURL = @"<获取到的licenseUrl>";
NSString * const licenceKey = @"<获取到的key>";

// V2TXLivePremier 位于 "V2TXLivePremier.h" 头文件中
[V2TXLivePremier setLicence:licenceURL key:licenceKey];
[V2TXLivePremier setObserver:self];
NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);
return YES;
}
```

```
#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

> ⚠ 注意：
> **License 中配置的 packageName/BundleId 必须和应用本身一致，否则会播放失败。**

步骤6：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	V2TXLivePlayer(iOS)	推荐 FLV
Android App	接入指引	V2TXLivePlayer(Android)	推荐 FLV
Web 浏览器	接入指引	-	• 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7：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 V2TXLivePlayer 播放器，可以达到下表中的延时标准：

旁路流类型	V2TXLivePlayer 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）

旁路流类型	V2TXLivePlayer 的播放模式	平均延时	实测效果
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 V2TXLivePlayer 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 V2TXLivePlayer**
  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。V2TXLivePlayer 有一个自研的播放引擎，具备延时调控的能力。
- **设置 V2TXLivePlayer 的播放模式为极速模式**
  可以通过设置 V2TXLivePlayer 的参数来实现极速模式，以 iOS 为例：

```
//自动模式
[_txLivePlayer setCacheParams:1 maxTime:5];
//极速模式
[_txLivePlayer setCacheParams:1 maxTime:1];
//流畅模式
[_txLivePlayer setCacheParams:5 maxTime:5];

//设置完成之后再启动播放
```

相关费用

使用 CDN 直播观看需要云直播服务资源和终端 SDK 直播播放能力的配合，可能会产生以下费用。

云服务消耗

CDN 直播观看需要使用**云直播**的资源进行直播分发。云直播的费用主要包括**基础服务费用和增值服务费用**：基础服务主要是直播推流/播放产生的消耗；增值服务是直播过程中，使用增值服务产生的消耗。

> ⚠ 注意：
> 本文中的价格为示例，仅供参考。最终价格与计费策略请以 云直播 的计费说明为准。

- **基础服务费用：**
  将 TRTC 的内容旁路到并云直播 CDN 观看时，**云直播**会收取观众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。
- **增值服务费用：**
  如果您使用了云直播的转码、录制、云导播等功能，会产生对应额外的增值服务费用。增值服务可按需使用进行付费。

SDK 播放授权

音视频通话（TRTC）SDK 提供了功能全面性能强大的直播播放能力，可轻松配合云直播实现 CDN 直播观看功能。移动端 SDK 在10.1及以上的版本可通过获取指定 License 以解锁直播播放能力。

> ⚠ 注意：
> TRTC 的播放能力无需 License 授权。

您可直接 购买视频播放 License，或通过 购买的云点播流量包 免费获赠视频播放 License 或 短视频 License，两种 License 均可用于解锁 SDK 的视频播放功能。并且点播资源包可以抵扣云点播的播放产生的日结流量，详细说明请参见 云点播预付费资源包。
License 计费说明参见 腾讯云视立方License，License 购买完成后可参考License操作指引进行新增和续期等操作。

常见问题

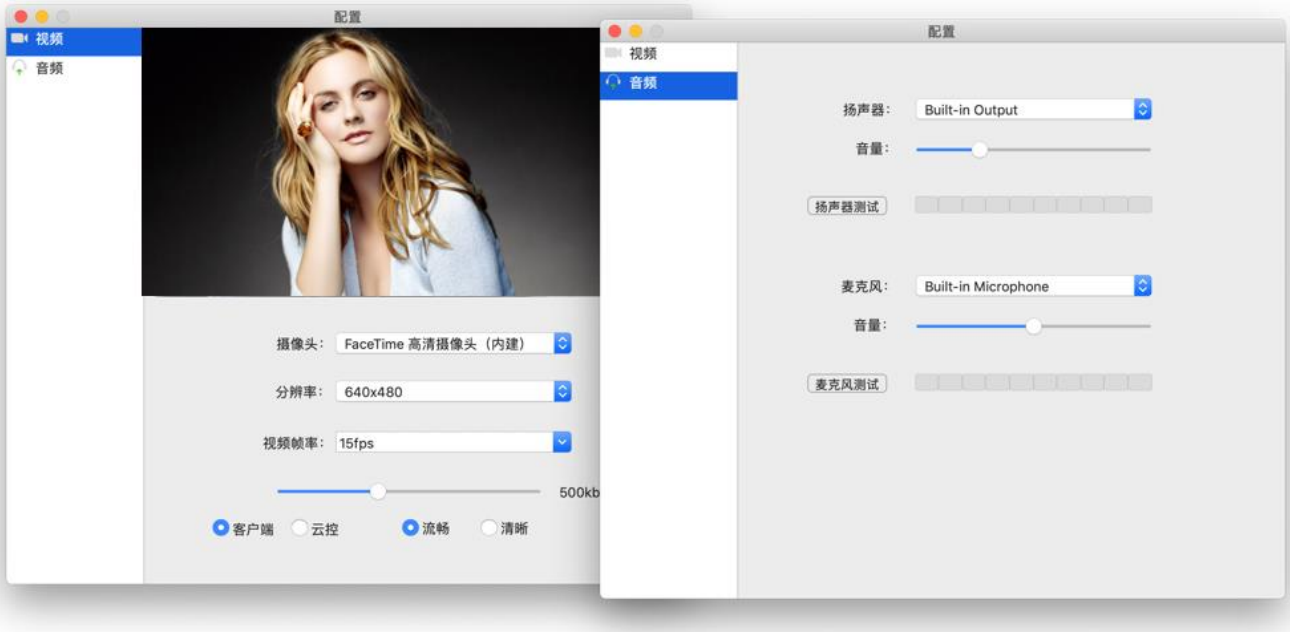
为什么房间里只有一个人时画面又卡又模糊？
请将 enterRoom 中 TRTCAppScene 参数指定为 **TRTCAppSceneLIVE**。
VideoCall 模式针对视频通话做了优化，所以在房间中只有一个用户时，画面会保持较低的码率和帧率以节省用户的网络流量，看起来会感觉又卡又模糊。

测试硬件设备
Android&iOS&Windows&Mac

最近更新时间：2021-08-13 15:04:01

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();
}
```

Web

最近更新时间：2022-06-24 16:44:52

在进行视频通话之前，建议先进行浏览器环境检测，以及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。

浏览器环境检测

在调用 SDK 的通信能力之前，建议您先使用 {@link TRTC.checkSystemRequirements checkSystemRequirements()} 接口检测 SDK 是否支持当前网页。如果 SDK 不支持当前浏览器，请根据用户设备类型建议用户使用 SDK 支持的浏览器。

```
TRTC.checkSystemRequirements().then(checkResult => {
if (checkResult.result) {
// 支持进房
if (checkResult.isH264DecodeSupported) {
// 支持拉流
}
if (checkResult.isH264EncodeSupported) {
// 支持推流
}
}
})
```

⚠ 注意：

当用户使用 SDK 支持的浏览器，且收到 TRTC.checkSystemRequirements 返回的检测结果为 false 时，可能是以下原因：

- **情况一：**请检查链接是否满足以下三种情况之一
  - localhost 域（ Firefox 浏览器支持 localhost 及本地 IP 访问 ）
  - 开启了 HTTPS 的域
  - 使用 file:/// 协议打开的本地文件
- **情况二：**Firefox 浏览器安装完成后需要动态加载 H264 编解码器，因此会出现短暂的检测结果为 false 的情况，请稍等再试或先使用其他推荐浏览器打开链接。

已知的浏览器使用限制说明

- **Firefox：**Firefox 只支持视频帧率为 30 fps, 如有帧率设置需求，请使用 SDK 支持的其他浏览器。
- **QQ 浏览器：**个别摄像头，麦克风正常的 Windows 设备在 localhost 环境下调用 localStream.initialize() 时抛出 NotFoundError 错误。

音视频设备测试

为保证用户在使用 TRTC-SDK 的过程中有更好的用户体验，我们建议您在用户加入 TRTC 房间之前，对用户设备及网络状况进行检测并给出建议和引导。
为方便您快速集成设备检测及网络检测功能，我们提供一下几种方式供您参考：

- rtc-detect 的 JS 库

- 设备检测的 React 组件
- TRTC 能力检测页面

rtc-detect 库

您可以使用 rtc-detect 用来检测当前环境对 TRTC SDK 的支持度，以及当前环境的详细信息。

安装

```
npm install rtc-detect
```

使用方法

```
import RTCDetect from 'rtc-detect';
// 初始化监测模块
const detect = new RTCDetect();
// 获得当前环境监测结果
const result = await detect.getReportAsync();
// result 包含了当前环境系统的信息，API 支持度，编解码支持度，设备相关的信息
console.log('result is: ' + result);
```

API

(async) isTRTCSupported()

判断当前环境是否支持 TRTC。

```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();

if (data.result) {
console.log('current browser supports TRTC.')
} else {
console.log(`current browser does not support TRTC, reason: ${data.reason}.`)
}
```

getSystem()

获取当前系统环境参数。

Item	Type	Description
UA	string	浏览器的 ua
OS	string	当前设备的系统型号
browser	object	当前浏览器信息{ name, version }

Item	Type	Description
displayResolution	object	当前分辨率 { width, height }
getHardwareConcurrency	number	当前设备 CPU 核心数

```
const detect = new RTCDetect();
const result = detect.getSystem();
```

getAPISupported()

获取当前环境 API 支持度。

Item	Type	Description
isUserMediaSupported	boolean	是否支持获取用户媒体数据流
isWebRTCSupported	boolean	是否支持 WebRTC
isWebSocketSupported	boolean	是否支持 WebSocket
isWebAudioSupported	boolean	是否支持 WebAudio
isScreenCaptureAPISupported	boolean	是否支持获取屏幕的流
isCanvasCapturingSupported	boolean	是否支持从 canvas 获取数据流
isVideoCapturingSupported	boolean	是否支持从 video 获取数据流
isRTPSenderReplaceTracksSupported	boolean	是否支持替换 track 时不和 peerConnection 重新协商
isApplyConstraintsSupported	boolean	是否支持变更摄像头的分辨率不通过重新调用 getUserMedia

```
const detect = new RTCDetect();
const result = detect.getAPISupported();
```

(async) getDevicesAsync()

获取当前环境可用的设备。

Item	Type	Description
hasWebCamPermissions	boolean	是否支持获取用户摄像头数据
hasMicrophonePermission	boolean	是否支持获取用户麦克风数据
cameras	array<CameraItem>	用户的摄像头设备列表，包含支持视频流的分辨率信息，最大宽高以及最大帧率（最大帧率有部分浏览器不支持）
microphones	array<DeviceItem>	用户的麦克风设备列表
speakers	array<DeviceItem>	用户的扬声器设备列表

CameraItem

Item	Type	Description
deviceId	string	设备 ID， 通常是唯一的，可以用于采集识别设备
groupId	string	组的标识符，如果两个设备属于同一个物理设备，他们就有相同的标识符
kind	string	摄像头设备类型：'videoinput'
label	string	描述该设备的标签
resolution	object	摄像头支持的最大分辨率的宽高和帧率 {maxWidth：1280，maxHeight：720，maxFrameRate：30}

DeviceItem

Item	Type	Description
deviceId	string	设备 ID， 通常是唯一的，可以用于采集识别设备
groupId	string	组的标识符，如果两个设备属于同一个物理设备，他们就有相同的标识符
kind	string	设备类型，例如：'audioinput'、'audiooutput'
label	string	描述该设备的标签

```
const detect = new RTCDetect();
const result = await detect.getDevicesAsync();
```

(async) getCodecAsync()

获取当前环境参数对编码的支持度。

Item	Type	Description
isH264EncodeSupported	boolean	是否支持 h264 编码
isH264DecodeSupported	boolean	是否支持 h264 解码
isVp8EncodeSupported	boolean	是否支持 vp8 编码
isVp8DecodeSupported	boolean	是否支持 vp8 解码

支持编码即支持发布音视频，支持解码即支持拉取音视频播放。

```
const detect = new RTCDetect();
const result = await detect.getCodecAsync();
```

(async) getReportAsync()

获取当前环境监测报告。

Item	Type	Description
system	object	和 getSystem() 的返回值一致
APISupported	object	和 getAPISupported() 的返回值一致
codecsSupported	object	和 getCodecAsync() 的返回值一致
devices	object	和 getDevicesAsync() 的返回值一致

```
const detect = new RTCDetect();
const result = await detect.getReportAsync();
```

(async) isHardWareAccelerationEnabled()

检测 Chrome 浏览器是否开启硬件加速。

> ⚠ 注意：
> 该接口的实现依赖于 WebRTC 原生接口，建议在 isTRTCSupported 检测支持后，再调用该接口进行检测。检测最长耗时 30s。经实测：
>
> 1. 开启硬件加速的情况下，该接口在 Windows 耗时 2s 左右， Mac 需耗时 10s 左右。
> 2. 关闭硬件加速的情况下，该接口在 Windows 和 Mac 耗时均为 30s。

```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();

if (data.result) {
const result = await detect.isHardWareAccelerationEnabled();
console.log(`is hardware acceleration enabled: ${result}`);
} else {
console.log(`current browser does not support TRTC, reason: ${data.reason}.`)
}
```

设备检测的 React 组件

设备检测 UI 组件特点

- 处理了设备连接及设备检测逻辑
- 处理了网络检测的逻辑
- 网络检测 Tab 页可选
- 支持中、英文两种语言

设备检测 UI 组件相关链接

- 组件 npm 包使用说明请参考：rtc-device-detector-react
- 组件源码调试请参考：github/rtc-device-detector

- 组件引用示例请参考：WebRTC API Example

设备检测 UI 组件界面

跳过检测

设备连接

设备检测前请确认设备连接了摄像头、麦克风、扬声器和网络

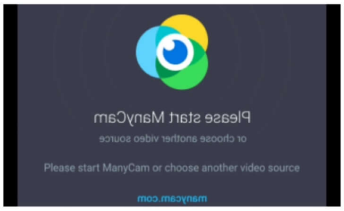
设备及网络连接成功，请开始设备检测

开始检测

跳过检测

CAMERA MICROPHONE SPEAKER NETWORK

摄像头选择 ManyCam Virtual Webcam (pl



是否可以清楚的看到自己?

看不到 看的到

跳过检测

CAMERA MICROPHONE SPEAKER NETWORK

麦克风选择 默认 - MacBook Pro麦克风 (B

对着麦克风说 哈喽 试试～

是否可以看到音量图标跳动?

看不到 看的到

跳过检测

CAMERA MICROPHONE SPEAKER NETWORK

扬声器选择 默认 - MacBook Pro扬声器 (B

请调高设备音量，点击播放下面的音频试试～

0:00 / 3:49

是否可以听到声音?

听不到 听的到

跳过检测

CAMERA MICROPHONE SPEAKER NETWORK

操作系统	MacOS
浏览器	Chrome 91.0.4472.114
是否支持TRTC	支持
是否支持屏幕分享	支持
网络延时	39ms
上行网络质量	极佳
下行网络质量	极佳

查看检测报告

跳过检测

检测报告

ManyCam Virtual Webcam (pleV:ideo)	正常
默认 - MacBook Pro麦克风 (Built-in)	正常
默认 - MacBook Pro扬声器 (Built-in)	正常
网络延时	39ms
上行网络质量	极佳
下行网络质量	极佳

重新检测 完成检测

设备及网络检测逻辑

1. 设备连接

设备连接的目的是检测用户使用的机器是否有摄像头，麦克风，扬声器设备，是否在联网状态。如果有摄像头，麦克风设备，尝试获取音视频流并引导用户授予摄像头，麦克风的访问权限。

- 判断设备是否有摄像头，麦克风，扬声器设备

```
import TRTC from 'trtc-js-sdk';

const cameraList = await TRTC.getCameras();
const micList = await TRTC.getMicrophones();
const speakerList = await TRTC.getSpeakers();
const hasCameraDevice = cameraList.length > 0;
const hasMicrophoneDevice = micList.length > 0;
const hasSpeakerDevice = speakerList.length > 0;
```

- 获取摄像头，麦克风的访问权限

```
navigator.mediaDevices
.getUserMedia({ video: hasCameraDevice, audio: hasMicrophoneDevice })
.then((stream) => {
// 获取音视频流成功
// ...
// 释放摄像头，麦克风设备
stream.getTracks().forEach(track => track.stop());
})
.catch((error) => {
// 获取音视频流失败
});
```

- 判断设备是否联网

```
export function isOnline() {
const url = 'https://web.sdk.qcloud.com/trtc/webrtc/assets/trtc-logo.png';
return new Promise((resolve) => {
try {
const xhr = new XMLHttpRequest();
xhr.onload = function () {
resolve(true);
};
xhr.onerror = function () {
resolve(false);
};
xhr.open('GET', url, true);
xhr.send();
} catch (err) {
```

```
// console.log(err);
}
});
}
const isOnline = await isOnline();
```

2. **摄像头检测**

摄像头检测为用户渲染了选中的摄像头采集的视频流，帮助用户确认摄像头是否可以正常使用。

- **获取摄像头列表，默认使用摄像头列表中的第一个设备**

```
import TRTC from 'trtc-js-sdk';
let cameraList = await TRTC.getCameras();
let cameraId = cameraList[0].deviceId;
```

- **初始化视频流并在 id 为 camera-video 的 dom 元素中播放流**

```
const localStream = TRTC.createStream({
video: true,
audio: false,
cameraId,
});
await localStream.initialize();
localStream.play('camera-video');
```

- **用户切换摄像头设备后更新流**

```
localStream.switchDevice('video', cameraId);
```

- **监听设备插拔**

```
navigator.mediaDevices.addEventListener('devicechange', async () => {
cameraList = await TRTC.getCameras();
cameraId = cameraList[0].deviceId;
localStream.switchDevice('video', cameraId);
})
```

- **检测完成后释放摄像头占用**

```
localStream.close();
```

3. **麦克风检测**

麦克风检测为用户渲染了选中的麦克风采集的音频流的音量，帮助用户确认麦克风是否可以正常使用。

- **获取麦克风列表，默认使用麦克风列表中的第一个设备**

```
import TRTC from 'trtc-js-sdk';

let microphoneList = await TRTC.getMicrophones();
let microphoneId = microphoneList[0].deviceId;
```

- 初始化音频流并在 id 为 audio-container 的 dom 元素中播放流

```
const localStream = TRTC.createStream({
video: false,
audio: true,
microphoneId,
});
await localStream.initialize();
localStream.play('audio-container');
timer = setInterval(() => {
const volume = localStream.getAudioLevel();
}, 100);
```

  - 用户切换麦克风设备后更新流

```
// 获取用户新选中的 microphoneId
localStream.switchDevice('audio', microphoneId);
```

- 监听设备插拔

```
navigator.mediaDevices.addEventListener('devicechange', async () => {
microphoneList = await TRTC.getMicrophones();
microphoneId = microphoneList[0].deviceId;
localStream.switchDevice('audio', microphoneId);
})
```

- 检测完成后释放麦克风占用，停止音量监听

```
localStream.close();
clearInterval(timer);
```

4. **扬声器检测**

   扬声器检测提供了音频播放器，用户可以通过播放音频确认选中的扬声器是否可以正常使用。

- 提供 mp3 播放器，提醒用户跳高设备播放音量，播放 mp3 确认扬声器设备是否正常

```
<audio id="audio-player" src="xxxxx" controls></audio>
```

- 检测结束后停止播放

```
const audioPlayer = document.getElementById('audio-player');
if (!audioPlayer.paused) {
audioPlayer.pause();
}
audioPlayer.currentTime = 0;
```

5. 网络检测

- 调用 TRTC.createClient 创建两个 Client，分别称为 uplinkClient 和 downlinkClient，这两个 Client 都进入同一个房间。
- 使用 uplinkClient 进行推流，监听 NETWORK_QUALITY 事件来检测上行网络质量。
- 使用 downlinkClient 进行拉流，监听 NETWORK_QUALITY 事件来检测下行网络质量。

整个过程可持续 15s 左右，最后取平均网络质量，从而大致判断出上下行网络情况。

> ⚠ 注意：
> 检测过程将产生少量的基础服务费用。如果未指定推流分辨率，则默认以 640*480 的分辨率推流。

```
let uplinkClient = null; // 用于检测上行网络质量
let downlinkClient = null; // 用于检测下行网络质量
let localStream = null; // 用于测试的流
let testResult = {
// 记录上行网络质量数据
uplinkNetworkQualities: [],
// 记录下行网络质量数据
downlinkNetworkQualities: [],
average: {
uplinkNetworkQuality: 0,
downlinkNetworkQuality: 0
}
}

// 1. 检测上行网络质量
async function testUplinkNetworkQuality() {
uplinkClient = TRTC.createClient({
sdkAppId: 0, // 填写 sdkAppId
userId: 'user_uplink_test',
userSig: '', // uplink_test 的 userSig
mode: 'rtc'
});

localStream = TRTC.createStream({ audio: true, video: true });
// 根据实际业务场景设置 video profile
localStream.setVideoProfile('480p');
await localStream.initialize();
```

```
uplinkClient.on('network-quality', event => {
const { uplinkNetworkQuality } = event;
testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
});

// 加入用于测试的房间，房间号需要随机，避免冲突
await uplinkClient.join({ roomId: 8080 });
await uplinkClient.publish(localStream);
}

// 2. 检测下行网络质量
async function testDownlinkNetworkQuality() {
downlinkClient = TRTC.createClient({
sdkAppId: 0, // 填写 sdkAppId
userId: 'user_downlink_test',
userSig: '', // userSig
mode: 'rtc'
});

downlinkClient.on('stream-added', async event => {
await downlinkClient.subscribe(event.stream, { audio: true, video: true });
// 订阅成功后开始监听网络质量事件
downlinkClient.on('network-quality', event => {
const { downlinkNetworkQuality } = event;
testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
});
})
// 加入用于测试的房间，房间号需要随机，避免冲突
await downlinkClient.join({ roomId: 8080 });
}

// 3. 开始检测
testUplinkNetworkQuality();
testDownlinkNetworkQuality();

// 4. 15s 后停止检测，计算平均网络质量
setTimeout(() => {
// 计算上行平均网络质量
if (testResult.uplinkNetworkQualities.length > 0) {
testResult.average.uplinkNetworkQuality = Math.ceil(
testResult.uplinkNetworkQualities.reduce((value, current) => value + current, 0) / testResult.uplinkNetworkQ
ualities.length
);
}
```

```
if (testResult.downlinkNetworkQualities.length > 0) {
// 计算下行平均网络质量
testResult.average.downlinkNetworkQuality = Math.ceil(
testResult.downlinkNetworkQualities.reduce((value, current) => value + current, 0) / testResult.downlinkNet
workQualities.length
);
}

// 检测结束，清理相关状态。
uplinkClient.leave();
downlinkClient.leave();
localStream.close();
}, 15 * 1000);
```

TRTC 能力检测页面

您可以在当前使用 TRTC SDK 的地方，使用 TRTC 检测页面 ，可用于探测当前环境，还可以点击生成报告按钮，得到当前环境的报告，用于环境检测，或者问题排查。

测试网络质量
Android&iOS&Windows&Mac

最近更新时间：2021-12-30 10:26:56

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。

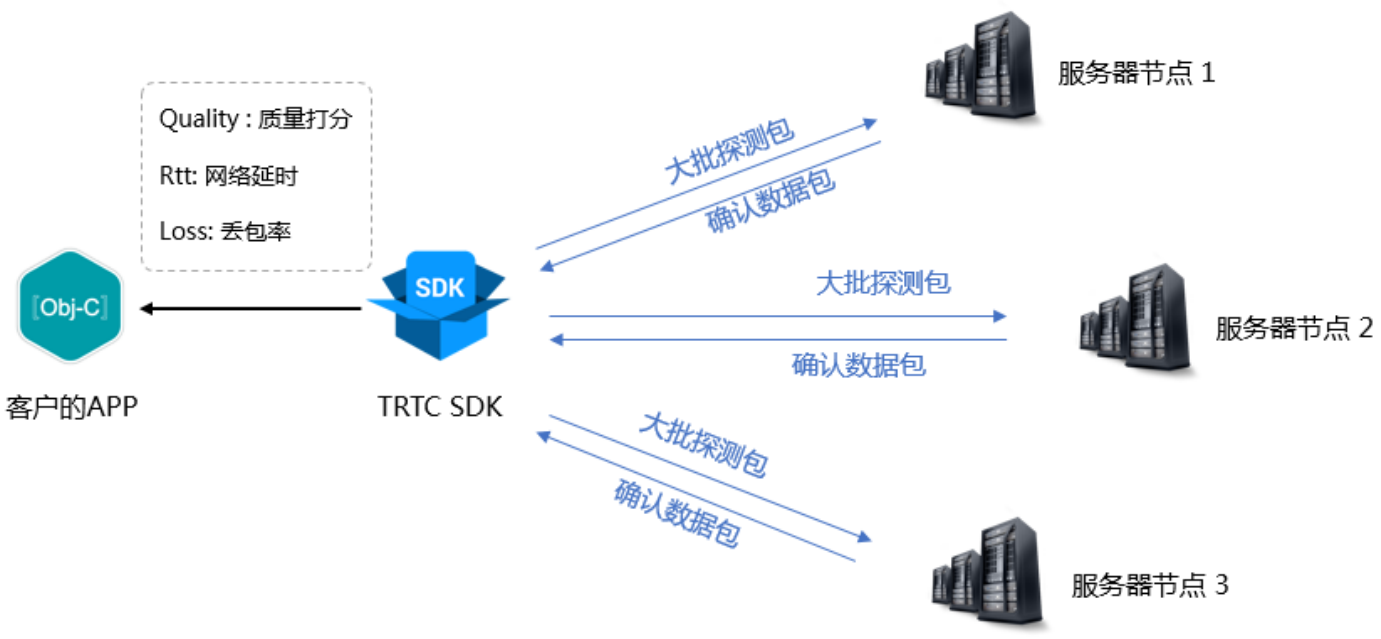
注意事项

- 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

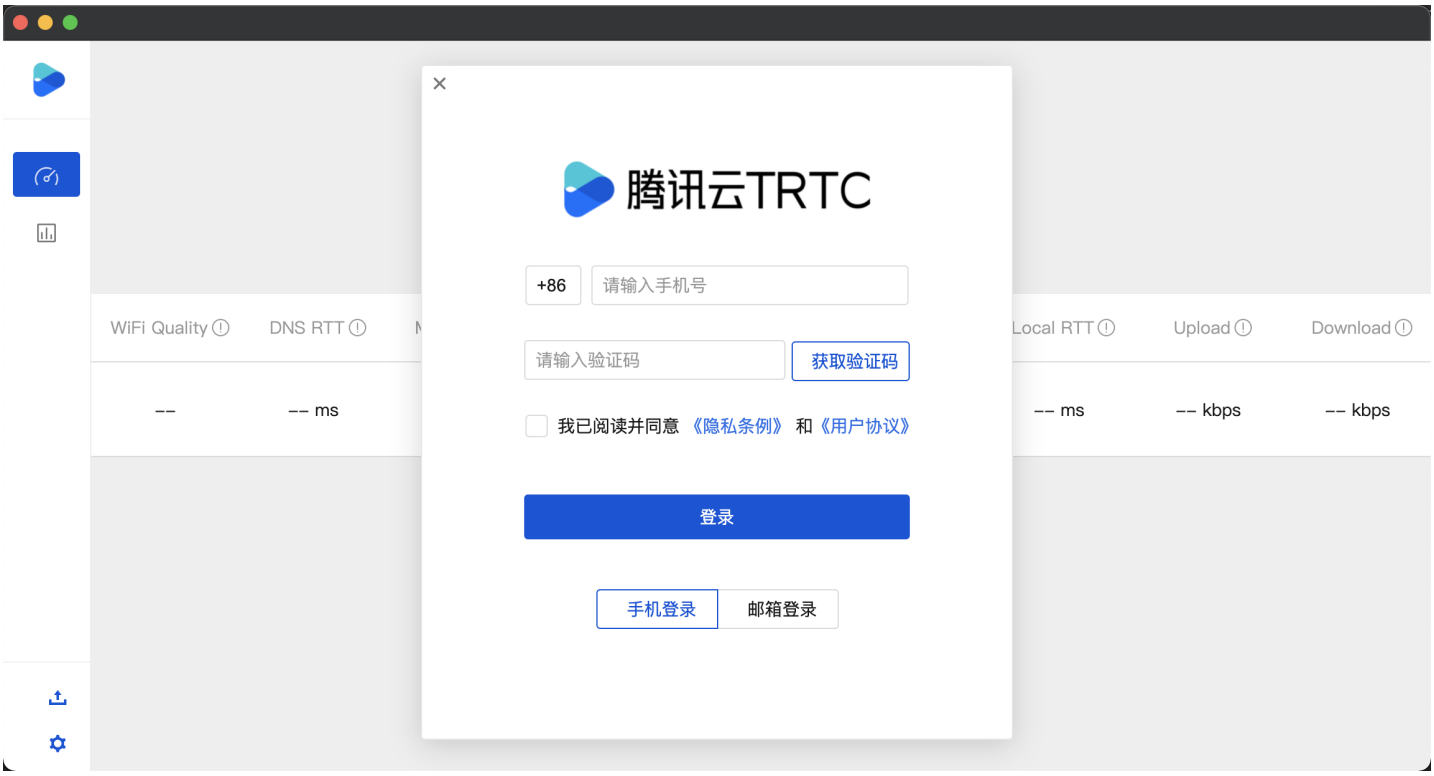
Mac | Windows

测试指标

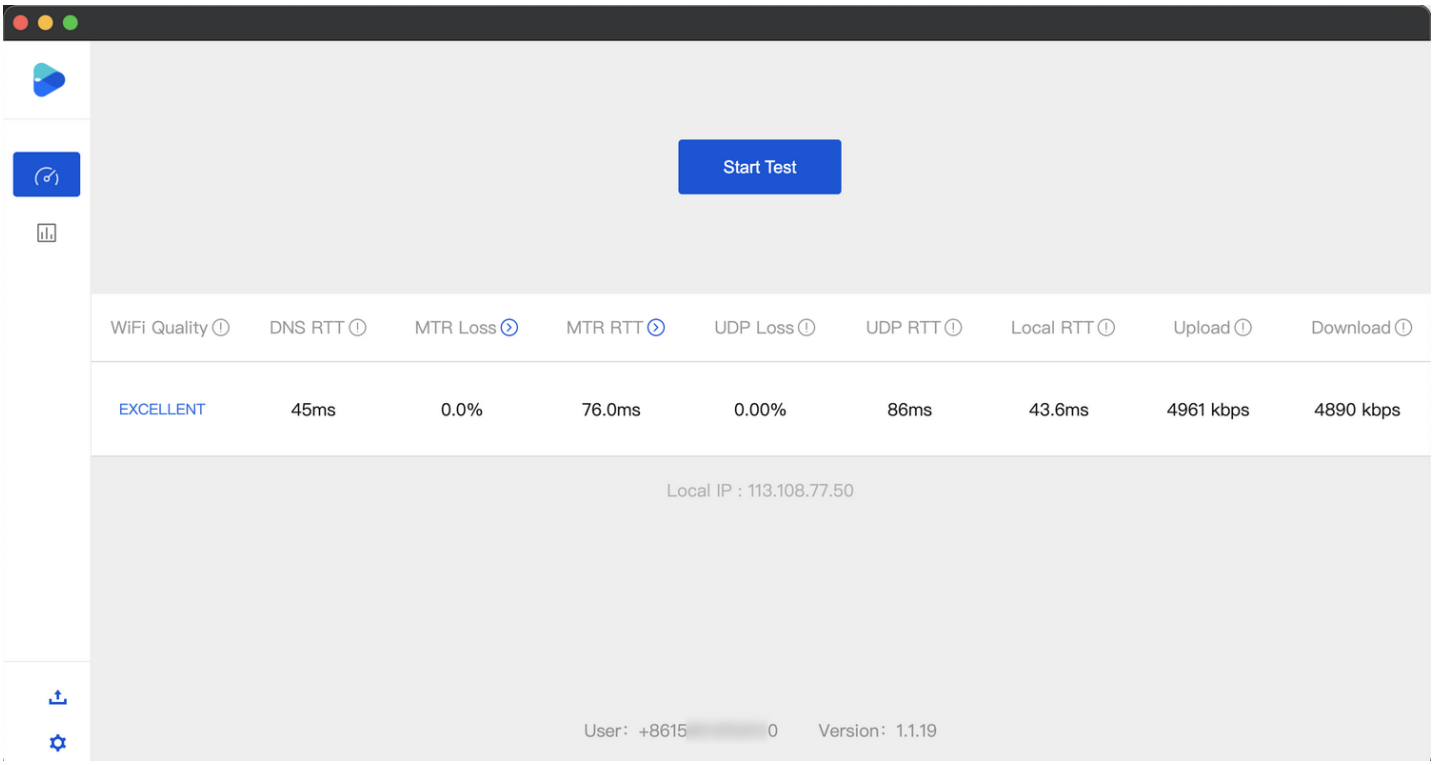
指标	含义
WiFi Quality	Wi-Fi 信号质量
DNS RTT	腾讯云的测速域名解析耗时
MTR	MTR 是一款网络测试工具，能探测客户端到 TRTC 节点的丢包率与延时，还可以查看路由中每一跳的具体信息
UDP Loss	客户端到 TRTC 节点的 UDP 丢包率
UDP RTT	客户端到 TRTC 节点的 UDP 延时
Local RTT	客户端到本地网关的延时
Upload	上行预估带宽
Download	下行预估带宽

工具截图

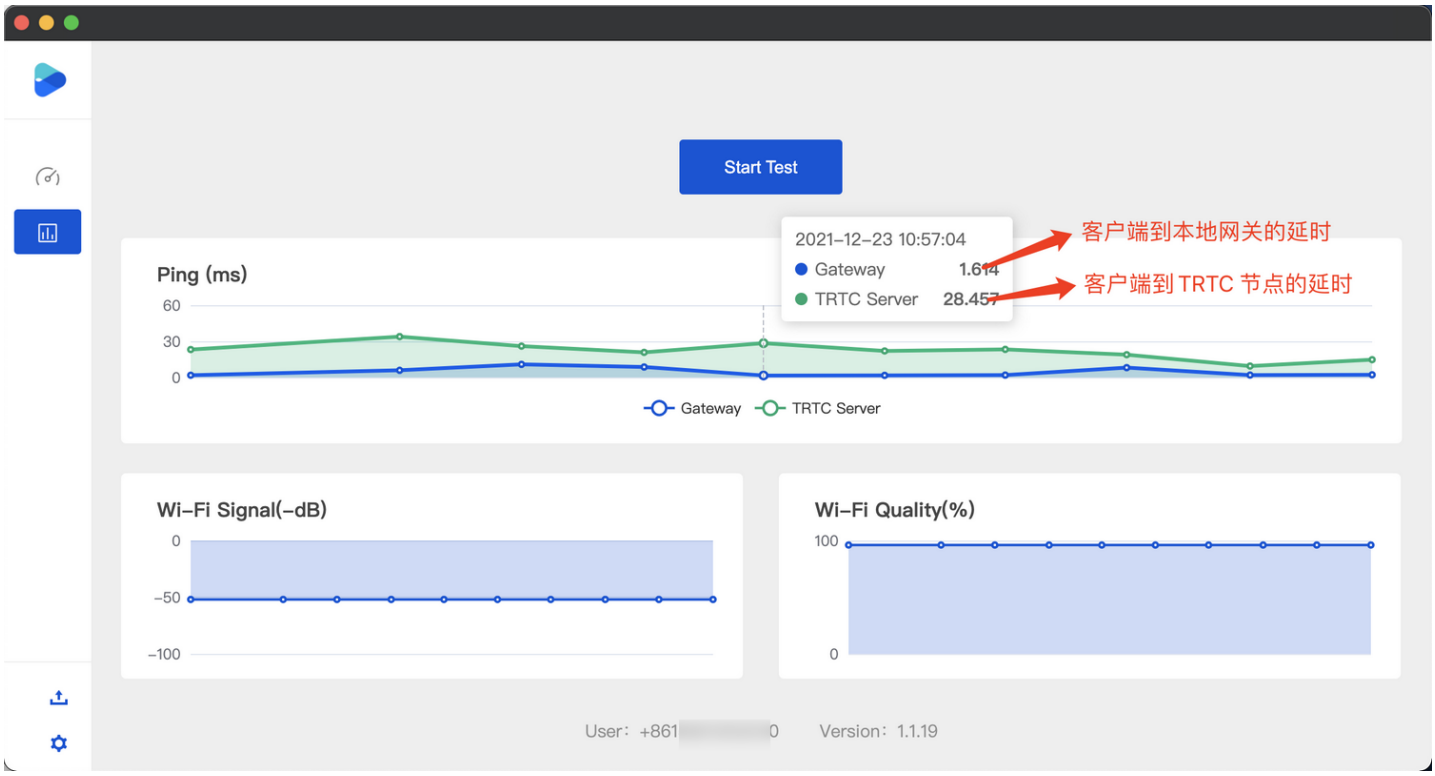
登录：


腾讯云TRTC
+86
请输入手机号
请输入验证码
获取验证码
我已阅读并同意 《隐私条例》 和 《用户协议》
登录
手机登录
邮箱登录
WiFi Quality
DNS RTT
Local RTT
Upload
Download
-- ms
-- kbps

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
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

Web

最近更新时间：2022-06-02 10:48:16

在进房之前，或者通话过程中，可以检测用户的网络质量，可以提前判断用户当下的网络质量情况。若用户网络质量太差，应建议用户更换网络环境，以保证正常通话质量。

本文主要介绍如何基于 NETWORK_QUALITY 事件实现通话前网络质量检测。通话过程中感知网络质量，只需监听 NETWORK_QUALITY 事件即可。

实现流程

1. 调用 TRTC.createClient 创建两个 Client，分别称为 uplinkClient 和 downlinkClient。
2. 这两个 Client 都进入同一个房间。
3. 使用 uplinkClient 进行推流，监听 NETWORK_QUALITY 事件来检测上行网络质量。
4. 使用 downlinkClient 进行拉流，监听 NETWORK_QUALITY 事件来检测下行网络质量。
5. 整个过程可持续 15s 左右，最后取平均网络质量，从而大致判断出上下行网络情况。

> ⚠ 注意：
>
> - 检测过程将产生少量的基础服务费用。如果未指定推流分辨率，则默认以 640*480 的分辨率推流。

代码示例

```
let uplinkClient = null; // 用于检测上行网络质量
let downlinkClient = null; // 用于检测下行网络质量
let localStream = null; // 用于测试的流
let testResult = {
// 记录上行网络质量数据
uplinkNetworkQualities: [],
// 记录下行网络质量数据
downlinkNetworkQualities: [],
average: {
uplinkNetworkQuality: 0,
downlinkNetworkQuality: 0
}
}

// 1. 检测上行网络质量
async function testUplinkNetworkQuality() {
uplinkClient = TRTC.createClient({
sdkAppId: 0, // 填写 sdkAppId
userId: 'user_uplink_test',
userSig: '', // uplink_test 的 userSig
mode: 'rtc'
```

```
});

localStream = TRTC.createStream({ audio: true, video: true });
// 根据实际业务场景设置 video profile
localStream.setVideoProfile('480p');
await localStream.initialize();

uplinkClient.on('network-quality', event => {
const { uplinkNetworkQuality } = event;
testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
});

// 加入用于测试的房间，房间号需要随机，避免冲突
await uplinkClient.join({ roomId: 8080 });
await uplinkClient.publish(localStream);
}

// 2. 检测下行网络质量
async function testDownlinkNetworkQuality() {
downlinkClient = TRTC.createClient({
sdkAppId: 0, // 填写 sdkAppId
userId: 'user_downlink_test',
userSig: '', // userSig
mode: 'rtc'
});

downlinkClient.on('stream-added', async event => {
await downlinkClient.subscribe(event.stream, { audio: true, video: true });
// 订阅成功后开始监听网络质量事件
downlinkClient.on('network-quality', event => {
const { downlinkNetworkQuality } = event;
testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
});
})
// 加入用于测试的房间，房间号需要随机，避免冲突
await downlinkClient.join({ roomId: 8080 });
}

// 3. 开始检测
testUplinkNetworkQuality();
testDownlinkNetworkQuality();

// 4. 15s 后停止检测，计算平均网络质量
setTimeout(() => {
// 计算上行平均网络质量
```

```
if (testResult.uplinkNetworkQualities.length > 0) {
testResult.average.uplinkNetworkQuality = Math.ceil(
testResult.uplinkNetworkQualities.reduce((value, current) => value + current, 0) / testResult.uplinkNetworkQualities.length
);
}

if (testResult.downlinkNetworkQualities.length > 0) {
// 计算下行平均网络质量
testResult.average.downlinkNetworkQuality = Math.ceil(
testResult.downlinkNetworkQualities.reduce((value, current) => value + current, 0) / testResult.downlinkNetworkQualities.length
);
}

// 检测结束，清理相关状态。
uplinkClient.leave();
downlinkClient.leave();
localStream.close();
}, 15 * 1000);
```

结果分析

经过上述步骤，可以拿到上行平均网络质量、下行平均网络质量。网络质量的枚举值如下所示：

数值	含义
0	网络状况未知，表示当前 client 实例还没有建立上行/下行连接
1	网络状况极佳
2	网络状况较好
3	网络状况一般
4	网络状况差
5	网络状况极差
6	网络连接已断开 注意：若下行网络质量为此值，则表示所有下行连接都断开了

建议：当网络质量大于3时，应引导用户检查网络并尝试更换网络环境，否则难以保证正常的音视频通话。
也可通过下述策略来降低带宽消耗：

- 若上行网络质量大于3，则可通过 LocalStream.setVideoProfile() 接口降低码率 或 LocalStream.muteVideo() 方式关闭视频，以降低上行带宽消耗。
- 若下行网络质量大于3，则可通过订阅小流（参考：开启大小流传输）或者只订阅音频的方式，以降低下行带宽消耗。

使用第三方美颜
腾讯特效
腾讯特效引擎

最近更新时间：2022-06-02 09:31:28

腾讯云视立方 · 腾讯特效引擎（Tencent Effect）SDK 是音视频终端 SDK（腾讯云视立方）的重要组成部分，提供美颜特效功能，基于优图精准的 AI 能力和天天 P 图丰富的实时特效处理，为各类视频处理场景提供丰富的产品能力。腾讯特效 SDK 支持与腾讯云视立方 · 直播 SDK 、短视频 SDK、音视频通话 SDK 等音视频终端产品集成，高效便捷，优势尤为明显。

- 腾讯特效 SDK 功能说明
- 腾讯特效 SDK 集成直播指引

SDK 功能说明

最近更新时间：2022-06-02 09:30:59

腾讯特效 SDK 共有 11 个套餐，11 个套餐分为 2 个系列：A 系列基础套餐 和 S 系列高级套餐。不同系列的不同套餐对应不同功能，各套餐支持的功能详情如下表。更多下载说明请参见 SDK 下载。

A 系列基础套餐功能

A 系列基础套餐提供通用美型功能，适用于对脸部美颜调整要求较低的客户。

套餐功能		套餐编号					
		A1 - 01	A1 - 02	A1 - 03	A1 - 04	A1 - 05	A1 - 06
基础功能	**基础美颜** 美白、磨皮、红润	✓	✓	✓	✓	✓	✓
	画面调整 对比度、饱和度、清晰度	✓	✓	✓	✓	✓	✓
	基础美型 大眼、瘦脸（自然、女神、英俊）	✓	✓	✓	✓	✓	✓
	滤镜 （默认 10 款通用滤镜）	✓	✓	✓	✓	✓	✓
可拓展功能	**贴纸** （赠送 10 款可选 2D 通用贴纸）	-	✓	✓	✓	✓	✓
	通用美型 SDK （窄脸/下巴/发际线/瘦鼻）	-	-	✓	-	-	-
	手势识别 （赠送 1 款指定手势贴纸）	-	-	-	✓	-	-
	人像分割 / 虚拟背景 （赠送 3 款指定分割贴纸）	-	-	-	-	✓	-
	美妆 （赠送 3 款指定整妆）	-	-	-	-	-	✓
SDK 下载	iOS & Android	下载					

S 系列高级套餐功能

S 系列高级套餐提供高级美型功能（包括特效贴纸和美妆），适用于对脸部美颜调整需求较高的客户。

套餐功能		套餐编号				
		S1 - 00	S1 - 01	S1 - 02	S1 - 03	S1 - 04
基础功能	**基础美颜** 美白、磨皮、红润	✓	✓	✓	✓	✓

	套餐功能	套餐编号				
		S1 – 00	S1 – 01	S1 – 02	S1 – 03	S1 – 04
	画面调整 对比度、饱和度、清晰度	✓	✓	✓	✓	✓
	高级美型 大眼、窄脸、瘦脸（自然、女神、英俊）、V脸、下巴、短脸、脸型、发际线、亮眼、眼距、眼角、瘦鼻、鼻翼、瘦颧骨、鼻子位置、白牙、去皱、去法令纹、去眼袋、嘴型、嘴唇厚度、口红、腮红、立体	✓	✓	✓	✓	✓
	滤镜 （默认通用滤镜）	✓	✓	✓	✓	✓
	贴纸 （默认通用 2D 贴纸）	–	✓	✓	✓	✓
	高级贴纸 （默认通用 3D 贴纸）	–	✓	✓	✓	✓
	美妆 整妆	–	✓	✓	✓	✓
可拓展功能	**手势识别** （赠送 1 款指定手势贴纸）	–	–	✓	–	✓
	人像分割 / 虚拟背景 （赠送 3 款指定分割贴纸）	–	–	–	✓	✓
SDK 下载	iOS & Android	下载				

SDK 集成指引（iOS）

最近更新时间：2022-07-01 15:00:36

集成准备

1. 下载并解压 Demo 包，将 Demo 工程中的 xmagic 模块（bundle，XmagicIconRes，Xmagic 文件夹）导入到实际项目工程中。
2. 导入 SDK 目录中的 libpag.framework，Masonry.framework，XMagic.framework，YTCommonXMagic.framework。
3. framework 签名 **General-->Masonry.framework** 和 **libpag.framework** 选 **Embed & Sign**。
4. 将 Bundle ID 修改成与申请的测试授权一致。

开发者环境要求

- 开发工具 XCode 11 及以上：App Store 或单击 下载地址。
- 建议运行环境：
  - 设备要求：iPhone 5 及以上；iPhone 6 及以下前置摄像头最多支持到 720p，不支持 1080p。
  - 系统要求：iOS 10.0 及以上。

C/C++层开发环境

XCode 默认 C++ 环境。

类型	依赖库
系统依赖库	• Accelerate • AssetsLibrary • AVFoundation • CoreMedia • CoreFoundation • CoreML • Foundation • JavaScriptCore • libc++.tbd • libz.b • libresolv.tbd • libsqlite3.0.tbd • MetalPerformanceShaders • MetalKit • MobileCoreServices • OpneAL • OpneGLES • Security • ReplayKit • SystemConfiguration • UIKit
自带的库	• YTCommon（鉴权静态库） • XMagic（美颜静态库） • libpag（视频解码动态库） • Masonry（控件布局库） • TXLiteAVSDK_Professional

	• TXFFmpeg • TXSoundTouch

SDK 接口集成

- 步骤一 和 步骤二 可参考 Demo 工程中，ThirdBeautyViewController 类 viewDidLoad，buildBeautySDK 方法；AppDelegate类的application方法进行了Xmagic鉴权。
- 步骤四 至 步骤七 可参考 Demo 工程的 ThirdBeautyViewController，BeautyView 类相关实例代码。

步骤一：初始化授权

1. 首先在工程 AppDelegate 的 didFinishLaunchingWithOptions 中添加如下鉴权代码，其中 LicenseURL 和 LicenseKey 为腾讯云官网申请到授权信息，请参见 License 指引：

```
[TXLiveBase setLicenceURL:LicenseURL key:LicenseKey];
```

2. Xmagic 鉴权：在相关业务模块的初始化代码中设置 URL 和 KEY，触发 License 下载，避免在使用前才临时去下载。也可以在 AppDelegate 的 didFinishLaunchingWithOptions 方法里触发下载。其中，LicenseURL 和 LicenseKey 是控制台绑定 License 时生成的授权信息。

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authresult, NSString * _Non
null errorMsg) {
if (authresult == TELicenseCheckOk) {
NSLog(@"鉴权成功");
} else {
NSLog(@"鉴权失败");
}
}];
```

鉴权 errorCode 说明：

错误码	说明
0	成功。Success
-1	输入参数无效，例如 URL 或 KEY 为空
-3	下载环节失败，请检查网络设置
-4	从本地读取的 TE 授权信息为空，可能是 IO 失败引起
-5	读取 VCUBE TEMP License文件内容为空，可能是 IO 失败引起
-6	v_cube.license 文件 JSON 字段不对。请联系腾讯云团队处理
-7	签名校验失败。请联系腾讯云团队处理
-8	解密失败。请联系腾讯云团队处理
-9	TELicense 字段里的 JSON 字段不对。请联系腾讯云团队处理

错误码	说明
-10	从网络解析的 TE 授权信息为空。请联系腾讯云团队处理
-11	把TE授权信息写到本地文件时失败，可能是 IO 失败引起
-12	下载失败，解析本地 asset 也失败
-13	鉴权失败
其他	请联系腾讯云团队处理

步骤二：设置 SDK 素材资源路径

```
CGSize previewSize = [self getPreviewSizeByResolution:self.currentPreviewResolution];
NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask
, YES) lastObject];
beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_config.json"];
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !isDir) {
NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfigPath encoding:NSUTF8StringEnc
oding error:nil];
NSError *jsonError;
NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8StringEncoding];
beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
options:NSJSONReadingMutableContainers
error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
@"root_path":[[NSBundle mainBundle] bundlePath],
@"tnn_"
@"beauty_config":beautyConfigJson
};
// Init beauty kit
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDict];
```

步骤三：添加日志和事件监听

```
// Register log
[self.beautyKit registerSDKEventListener:self];
[self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL];
```

步骤四：配置美颜各种效果

```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString *_Nonnull)propertyName with
Data:(NSString*_Nonnull)propertyValue withExtraInfo:(id _Nullable)extraInfo;
```

步骤五：进行渲染处理

在视频帧回调接口，构造 YTProcessInput 传入到 SDK 内做渲染处理，可参考 Demo 中的 ThirdBeautyViewController。

```
[self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft withOrientation:YtLightCameraRotation0]
```

步骤六：暂停/恢复 SDK

```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

步骤七：布局中添加 SDK 美颜面板

```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
if (@available(iOS 11.0, *)) {
gSafeInset = [UIApplication sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
{
gSafeInset = UIEdgeInsetsZero;
}

dispatch_async(dispatch_get_main_queue(), ^{
//美颜选项界面
_vBeauty = [[BeautyView alloc] init];
[self.view addSubview:_vBeauty];
[_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
make.width.mas_equalTo(self.view);
make.centerX.mas_equalTo(self.view);
make.height.mas_equalTo(254);
if(gSafeInset.bottom > 0.0){ // 适配全面屏
make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
} else {
make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
}
}];
```

```
_vBeauty.hidden = YES;
});
```

SDK 集成指引（Android）

最近更新时间：2022-07-01 15:00:46

步骤一：解压 Demo 工程

1. 下载集成了腾讯特效 TE 的 TRTC Demo 工程。本 Demo 基于腾讯特效 SDK S1-04 套餐构建。
2. 替换资源。由于本 Demo 工程使用的 SDK 套餐未必与您实际的套餐一致，因此要将本 Demo 中的相关 SDK 文件替换为您实际使用的套餐的 SDK 文件。具体操作如下：
   - 删除 xmagic 模块中 libs 目录下的 .aar文件，将 SDK 中 libs 目录下的 .aar 文件拷贝进 xmagic 模块中 libs 目录下。
   - 删除 xmagic 模块中 assets 目录下的所有文件，将 SDK 中的 assets/ 目录下的全部资源拷贝到 xmagic 模块../src/main/assets 目录下，如果 SDK 包中的 MotionRes 文件夹内有资源，将此文件夹也拷贝到 ../src/main/assets 目录下。
   - 删除 xmagic 模块中 jniLibs 目录下的所有 .so 文件，在 SDK 包内的 jniLibs 中找到对应的 .so 文件（由于 SDK 中 jinLibs 文件夹下的 arm64-v8a 和 armeabi-v7a 的 .so 文件在压缩包中，所以需要先解压），拷贝到 xmagic 模块中的 ../src/main/jniLibs 目录下。
3. 将 Demo 工程中的 xmagic 模块引入到实际项目工程中。

步骤二：打开 app 模块的 build.gradle

1. 将 applicationId 修改成与申请的测试授权一致的包名。
2. 添加 gson 依赖设置。

```
configurations{
all*.exclude group:'com.google.code.gson'
}
```

步骤三：SDK 接口集成

可参考 Demo 工程的 ThirdBeautyActivity 类。

1. **授权：**

```
//鉴权注意事项及错误码详情，请参考 https://cloud.tencent.com/document/product/616/65891#.E6.AD.A5.E9.AA.A4.E4.B8.80.EF.BC.9A.E9.89.B4.E6.9D.83
XMagicImpl.checkAuth((errorCode, msg) -> {
if (errorCode == TELicenseCheck.ERROR_OK) {
showLoadResourceView();
} else {
TXCLog.e(TAG, "鉴权失败，请检查鉴权url和key" + errorCode + " " + msg);
}
});
```

2. **初始化素材：**

```
private void showLoadResourceView() {
if (XmagicLoadAssetsView.isCopyedRes) {
XmagicResParser.parseRes(getApplicationContext());
initXMagic();
} else {
loadAssetsView = new XmagicLoadAssetsView(this);
loadAssetsView.setOnAssetsLoadFinishListener(() -> {
XmagicResParser.parseRes(getApplicationContext());
initXMagic();
});
}
}
```

3. **开启推流设置：**

```
mTRTCCloud.setLocalVideoProcessListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D, TRTCCloudD
ef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCVideoFrameListener() {
@Override
public void onGLContextCreated() {
}
@Override
public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame, TRTCCloudDef.TRTCVideoFrame dstFr
ame) {
}
@Override
public void onGLContextDestory() {
}
});
```

4. **将 textureId 传入到 SDK 内做渲染处理：**
   在 TRTCVideoFrameListener 接口的 onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame, TRTCCloudDef.TRTCVideoFrame dstFrame) **方法内添加如下代码：**

```
dstFrame.texture.textureId = mXMagic.process(srcFrame.texture.textureId, srcFrame.width, srcFrame.height);
```

5. **暂停/关闭 SDK：**
   onPause() 用于暂停美颜效果，可以在 Activity/Fragment 生命周期方法中执行，onDestroy 方法需要在 GL 线程调用（可以在 onTextureDestroyed 方法中调用 XMagicImpl 对象的 onDestroy()**），更多使用请参考 Demo。**

```
mXMagic.onPause(); //暂停，与Activity的onPause方法绑定
mXMagic.onDestroy(); //销毁，需要在GL线程中调用
```

6. **布局中添加 SDK 美颜面板：**

```
<RelativeLayout
android:layout_above="@+id/ll_edit_info"
android:id="@+id/livepusher_bp_beauty_pannel"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
```

7. 初始化面板：

```
private void initXMagic() {
if (mXMagic == null) {
mXMagic = new XMagicImpl(this, mBeautyPanelView);
}else {
mXMagic.onResume();
}
}
```

具体操作请参见 Demo 程的 ThirdBeautyActivity.initXMagic(); **法。**

相芯美颜 SDK 接入

最近更新时间：2022-05-27 11:26:49

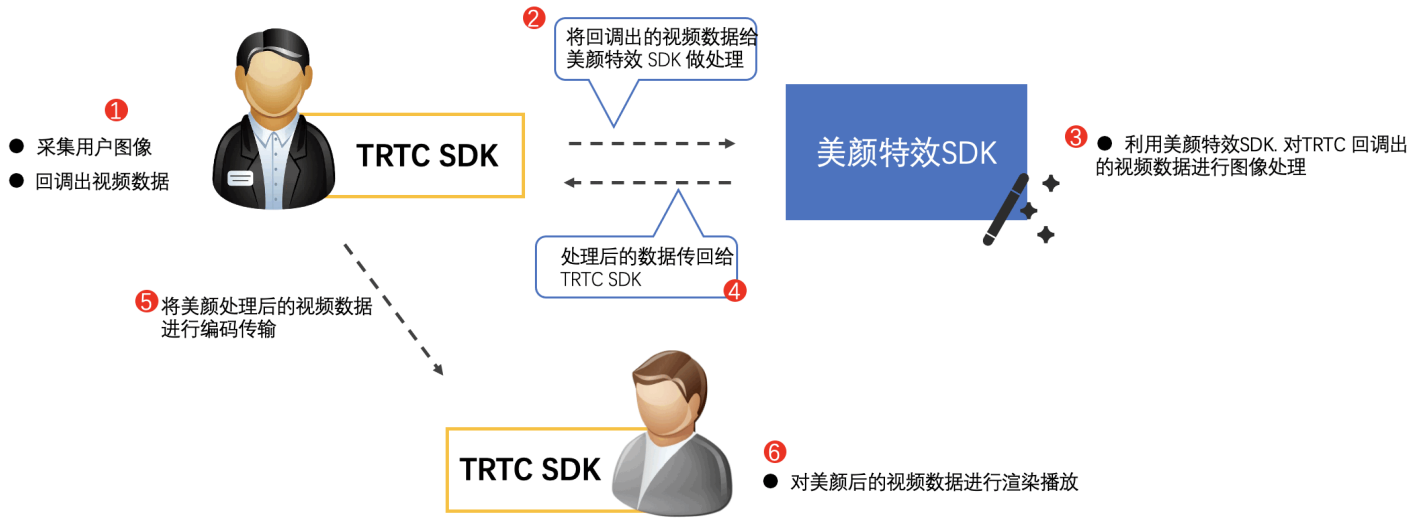
适用场景

实时音视频 TRTC 支持接入第三方美颜特效 SDK，目前 TRTC 已与多家美颜特效 SDK 供应商紧密合作，为您在多个场景提供接入便捷、性能卓越的“TRTC+美颜特效”解决方案。

结合 TRTC 低延时、高质量、高稳定的视频通信能力和美颜、美妆、美体、AR特效、虚拟背景等美颜特效能力可以触达秀场直播、视频相亲、互动课堂、视频会议、互动游戏等业务场景，快速构建具备美颜特效能力的实时音视频应用。

原理解析

从 TRTC 8.1 版本开始，TRTC SDK 提供了新的接口 setLocalVideoProcessListener，TRTC SDK 会在编码和渲染之前，将 TRTC SDK 采集到的图像通过该接口回调出来。您可以使用该接口，对回调出来的图像进行二次处理（例如：使用第三方美颜 SDK 进行美颜处理），并将处理后的图像通过参数传递给 TRTC SDK，TRTC SDK 后续渲染和编码都将使用二次处理后的图像。



操作步骤

步骤1：集成 TRTC

您可采用**自动加载 aar** 和**手动下载 aar** 的方式集成 TRTC。具体集成操作请参见 快速集成TRTC。

步骤2：集成美颜特效 SDK

参考您所对接的第三方特效产品集成文档，集成美颜特效 SDK。

以相芯美颜特效 SDK 为例，具体集成方法请参见 快速集成 NAMA SDK。

步骤3：初始化美颜特效 SDK

由于 SDK 和大部分特效产品内部都使用 OpenGL 来处理图像，因此使用 TEXTURE_2D 作为两个 SDK 对接格式，性能会最好。以相芯为例：可以在初始化 FURenderer 时指定输入格式：

iOS

```
// 初始化并对 SDK 进行授权
[[FURenderer shareRenderer] setupWithData:bundleData
dataSize:bundleDataSize
ardata:NULL
authPackage:secretKey
authSize:secretKeySize];
```

Android

```
FURenderer mFURenderer = new FURenderer.Builder(this)
.setInputTextureType(FURenderer.INPUT_TEXTURE_2D) // 指定输入格式为 TEXTURE_2D
.setCameraFacing(cameraFacing)
.setInputImageOrientation(CameraUtils.getCameraOrientation(cameraFacing))
.build();
```

步骤4：设置自定义预处理回调

通过调用 TRTCCloud 中的 setLocalVideoProcessListener 来设置自定义美颜的回调，并指定回调的格式为 TEXTURE_2D：

iOS

```
[[TRTCCloud sharedInstance] setLocalVideoProcessDelegete:self
pixelFormat:TRTCVideoPixelFormat_NV12
bufferType:TRTCVideoBufferType_PixelBuffer];
```

Android

```
mTRTCCloud.setLocalVideoProcessListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D,
TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, mVideoFrameListener);
```

步骤5：调用美颜特效 SDK 接口

自定义预处理回调接口的定义如下：

iOS

```
@protocol TRTCVideoFrameDelegate <NSObject>
@optional

- (uint32_t)onProcessVideoFrame:(TRTCVideoFrame * _Nonnull)srcFrame
dstFrame:(TRTCVideoFrame * _Nonnull)dstFrame;
- (void)onGLContextDestory;
@end
```

Android

```
public interface TRTCVideoFrameListener {
void onGLContextCreated();
int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame,
TRTCCloudDef.TRTCVideoFrame dstFrame);
void onGLContextDestory();
}
```

接口	说明
onGLContextCreated	在 TRTC SDK 创建 OpenGL 环境时回调，此时可以通知第三方美颜 SDK，以便它可以做一些自己的 OpenGL 资源初始化工作。
onProcessVideoFrame	是每帧都会进行回调，其中 srcFrame 是 SDK 采集到的图像数据，dstFrame 是用来存放二次处理后的图像。回调出来时，dstFrame 已经分配好用来存储图像的对象（dstFrame.texture、dstFrame.data、dstFrame.buffer），您可以直接使用。您也可以自己分配内存，并将 dstFrame 中对应字段修改为您自己分配的对象。
onGLContextDestory	在 TRTC SDK 销毁 OpenGL 环境时回调，此时可以通知第三方美颜 SDK，以便它可以做一些自己的 OpenGL 资源清理工作。

相芯接入示例

为了帮助您更好地理解和运用自定义预处理来对接第三方美颜特效 SDK，我们以相芯为例向您展示接入示例代码：

iOS

```
- (uint32_t)onProcessVideoFrame:(TRTCVideoFrame * _Nonnull)srcFrame
dstFrame:(TRTCVideoFrame * _Nonnull)dstFrame {
self.frameID += 1;
dstFrame.pixelBuffer = [[FURenderer shareRenderer] renderPixelBuffer:srcFrame.pixelBuffer
withFrameId:self.frameID
items:self.renderItems
itemCount:self.renderItems.count];
return 0;
}
```

Android

```
private final TRTCVideoFrameListener mVideoFrameListener = new TRTCVideoFrameListener() {
@Override
public void onGLContextCreated() {
// OpenGL环境创建时，需要调用相芯的onSurfaceCreated，以便其初始化OpenGL资源
mFURenderer.onSurfaceCreated();
mFURenderer.setUseTexAsync(true);
}
@Override
public int onProcessVideoFrame(TRTCVideoFrame srcFrame, TRTCVideoFrame dstFrame) {
// 相芯的处理接口会生成自己的纹理对象，因此通过修改 dstFrame 对应字段来将处理好后的纹理传给 TRTC SDK
```

```
dstFrame.texture.textureId = mFURenderer.onDrawFrameSingleInput(srcFrame.texture.textureId, srcFrame.widt
h, srcFrame.height);
return 0;
}
@Override
public void onGLContextDestory() {
// OpenGL环境销毁时，需要调用相芯的onSurfaceDestroyed，用于销毁对应的OpenGL资源
mFURenderer.onSurfaceDestroyed();
}
};
```

注意事项

- 相芯 SDK 7.2.0 版本存在兼容问题，需要更新到相芯 SDK 7.3.0 版本使用。
- 之前使用 setLocalVideoRenderListener 来实现自定义预处理的客户，可以更换为新接口来实现同样的功能，新接口新增了 OpenGL 环境生命周期的回调。

高级权限控制

最近更新时间：2021-11-09 10:22:36

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：

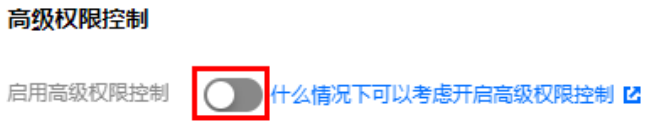
位数	二进制表示	十进制数字	权限含义
第 1 位	0000 0001	1	创建房间的权限
第 2 位	0000 0010	2	进入房间的权限
第 3 位	0000 0100	4	发送语音的权限
第 4 位	0000 1000	8	接收语音的权限
第 5 位	0001 0000	16	发送视频的权限
第 6 位	0010 0000	32	接收视频的权限
第 7 位	0100 0000	64	发送辅路（也就是屏幕分享）视频的权限

位数	二进制表示	十进制数字	权限含义
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。

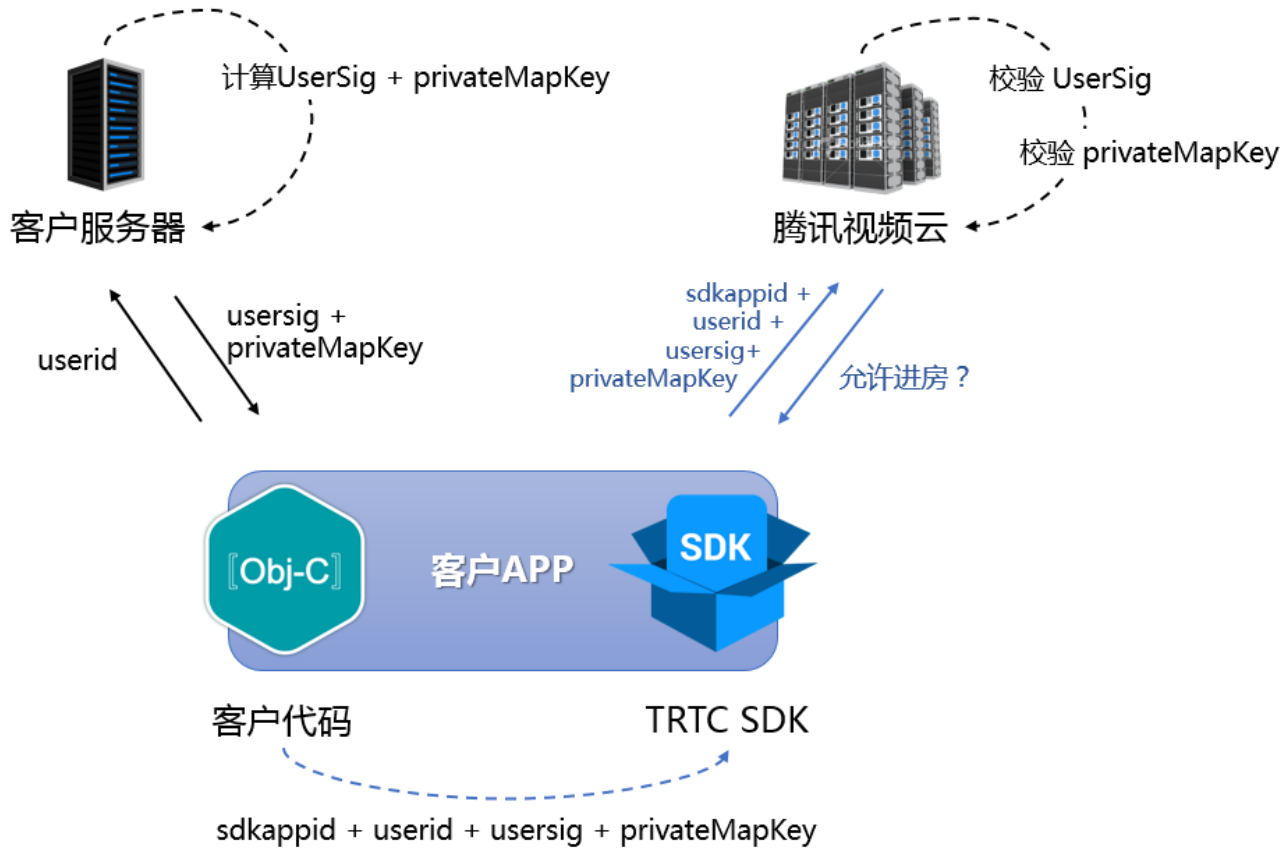
步骤2：在您的服务端计算 PrivateMapKey

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。

自定义视频采集和渲染 Android&iOS&Windows&Mac

最近更新时间：2022-06-02 10:48:39

本文档主要介绍如何使用 TRTC SDK 实现自定义视频采集和渲染，分为：视频采集、视频渲染两个部分。

自定义视频采集

TRTC SDK 的自定义视频采集功能的开启分为两步，即：开启功能、发送视频帧给 SDK，具体 API 使用步骤见下文，同时我们也提供有对应平台的的API-Example：

- Android
- iOS
- Windows

开启自定义视频采集功能

首先，您需要调用 TRTCCloud 的 enableCustomVideoCapture 接口开启 TRTC SDK 自定义视频采集的功能，开启后会跳过 TRTC SDK 自己的摄像头采集和图像处理逻辑，仅保留编码和传输能力，示例代码如下：

Android

```
TRTCCloud mTRTCCloud = TRTCCloud.shareInstance();
mTRTCCloud.enableCustomVideoCapture(TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, true);
```

iOS&Mac

```
self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud enableCustomVideoCapture:TRTCVideoStreamTypeBig enable:YES];
```

Windows

```
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->enableCustomVideoCapture(TRTCVideoStreamType::TRTCVideoStreamTypeBig, true);
```

发送自定义视频帧

然后您就可以使用 TRTCCloud 的 sendCustomVideoData 接口向 TRTC SDK 发送您自己的视频数据，示例代码如下：

> **说明：**
> 为了避免不必要的性能损失，对于输入 TRTC SDK 的视频数据，在不同平台上有不同的格式要求，更多信息，详见我们的 API 文档：简体中文 、English；

Android

```
// Android 平台有 Buffer 和 Texture 两种方案，此处以 Texture 方案为例，推荐！
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(TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, videoFrame);
```

iOS&Mac

```
// 在 iOS/Mac 平台上，摄像头原生采集的视频格式即是 NV12，原生支持且性能最佳的视频帧格式是CVPixelBufferRef，同时支持I420、OpenGL 2D纹理格式。此处以CVPixelBufferRef为例，推荐！
TRTCVideoFrame *videoFrame = [[TRTCVideoFrame alloc] init];
videoFrame.pixelFormat = TRTCVideoPixelFormat_NV12;
videoFrame.bufferType = TRTCVideoBufferType_PixelBuffer;
videoFrame.pixelBuffer = imageBuffer;
videoFrame.timestamp = timeStamp;

[[TRTCCloud sharedInstance] sendCustomVideoData:TRTCVideoStreamTypeBig frame:videoFrame];
```

Windows

```
// Windows 平台目前只支持 Buffer 的方案，推荐以此方式实现功能。
liteav::TRTCVideoFrame frame;
frame.timestamp = getTRTCShareInstance()->generateCustomPTS();
frame.videoFormat = liteav::TRTCVideoPixelFormat_I420;
frame.bufferType = liteav::TRTCVideoBufferType_Buffer;
frame.length = buffer_size;
frame.data = array.data();
frame.width = YUV_WIDTH;
frame.height = YUV_HEIGHT;
getTRTCShareInstance()->sendCustomVideoData(&frame);
```

自定义视频渲染

自定义渲染主要分为：本地预览画面的渲染、和远端用户画面的渲染，基本原理：设置本地/远端的自定义渲染回调，然后 TRTC SDK 会通过回调函数 onRenderVideoFrame 中传递出来对应的视频帧（即TRTCVideoFrame），然后就开发者可以根据收到的视频帧进行自定义渲染了，这个流程需要具备一定的OpenGL 基础，我们也提供有对应平台的的API-Example：

- Android：
- iOS

- Windows

设置本地预览画面的渲染回调

Android

```
mTRTCCloud.setLocalVideoRenderListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D, TRTCCloudD
ef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCVideoRenderListener() {
@Override
public void onRenderVideoFrame(String userId int streamType, TRTCCloudDef.TRTCVideoFrame frame) {
// 详见TRTC-API-Example 中自定义渲染的工具类：com.tencent.trtc.mediashare.helper.CustomFrameRender
}
});
```

iOS&Mac

```
self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud setLocalVideoRenderDelegate:self pixelFormat:TRTCVideoPixelFormat_NV12 bufferType:TRTCVide
oBufferType_PixelBuffer];
```

Windows

```
// 具体实现请参考 TRTC-API-Example-Qt 中 test_custom_render.cpp 的实现。
void TestCustomRender::onRenderVideoFrame(
const char* userId,
liteav::TRTCVideoStreamType streamType,
liteav::TRTCVideoFrame* frame) {
if (gl_yuv_widget_ == nullptr) {
return;
}

if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
// 调整渲染窗口
emit renderViewSize(frame->width, frame->height);
// 绘制视频帧
gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
frame->width, frame->height);
}
}
```

设置远端用户画面的渲染回调

Android

```
mTRTCCloud.setRemoteVideoRenderListener(userId, TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_I420, TRTCClou
dDef.TRTC_VIDEO_BUFFER_TYPE_BYTE_ARRAY, new TRTCCloudListener.TRTCVideoRenderListener() {
@Override
public void onRenderVideoFrame(String userId, int streamType, TRTCCloudDef.TRTCVideoFrame frame) {
```

```
// 详见TRTC-API-Example 中自定义渲染的工具类：com.tencent.trtc.mediashare.helper.CustomFrameRender
}
});
```

iOS&Mac

```
- (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);
});
}
```

Windows

```
// 具体实现请参考 TRTC-API-Example-Qt 中 test_custom_render.cpp 的实现。
void TestCustomRender::onRenderVideoFrame(
const char* userId,
liteav::TRTCVideoStreamType streamType,
liteav::TRTCVideoFrame* frame) {
if (gl_yuv_widget_ == nullptr) {
return;
}

if (streamType == liteav::TRTCVideoStreamType::TRTCVideoStreamTypeBig) {
// 调整渲染窗口
emit renderViewSize(frame->width, frame->height);
// 绘制视频帧
gl_yuv_widget_->slotShowYuv(reinterpret_cast<uchar*>(frame->data),
frame->width, frame->height);
```

```
}
}
```

Web

最近更新时间：2022-06-02 10:48:46

本文主要介绍本地流的自定义采集和音视频流的自定义播放渲染等高阶用法。

自定义采集

本地流在通过 {@link TRTC.createStream createStream()} 创建时，可以指定使用 SDK 的默认采集方式，

如下，从摄像头和麦克风采集音视频数据：

```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
// local stream initialized success
});
```

或者，采集屏幕分享流：

```
const localStream = TRTC.createStream({ userId, audio: false, screen: true });
localStream.initialize().then(() => {
// local stream initialized success
});
```

上述两种本地流的创建方式都是使用 SDK 的默认采集方式。为了便于开发者对音视频流进行预处理，createStream 支持从外部音视频源创建本地流，通过这种方式创建本地流，开发者可以实现自定义采集，比如说：

- 可以通过使用 getUserMedia 采集摄像头和麦克风音视频流。
- 通过 getDisplayMedia 采集屏幕分享流。
- 通过 captureStream 采集页面中正在播放的音视频。
- 通过 captureStream 采集 canvas 画布中的动画。

采集页面中正在播放的视频源

```
// 检测您当前的浏览器是否支持从 video 元素采集 stream
const isVideoCapturingSupported = () => {
['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) => {
if (item in document.createElement('video')) {
return true;
}
});
return false;
};

// 检测您当前的浏览器是否支持从 video 元素采集 stream
```

```
if (!isVideoCapturingSupported()) {
console.log('your browser does not support capturing stream from video element');
return
}
// 获取您页面在播放视频的 video 标签
const video = document.getElementById('your-video-element-ID');
// 从播放的视频采集视频流
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];

const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSource: videoTrack });

// 请确保视频属性跟外部传进来的视频源一致，否则会影响视频通话体验
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
// local stream initialized success
});
```

采集 canvas 中的动画

```
// 检测您当前的浏览器是否支持从 canvas 元素采集 stream
const isCanvasCapturingSupported = () => {
['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) => {
if (item in document.createElement('canvas')) {
return true;
}
});
return false;
};

// 检测您当前的浏览器是否支持从 canvas 元素采集 stream
if (!isCanvasCapturingSupported()) {
console.log('your browser does not support capturing stream from canvas element');
return
}
// 获取您的 canvas 标签
const canvas = document.getElementById('your-canvas-element-ID');

// 从 canvas 采集 15 fps 的视频流
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];
```

```
const localStream = TRTC.createStream({ userId, videoSource: videoTrack });

// 请确保视频属性跟外部传进来的视频源一致，否则会影响视频通话体验
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
// local stream initialized success
});
```

自定义播放渲染

对于通过 TRTC.createStream() 创建并初始化好的本地流或者通过 Client.on('stream-added') 接收到的远端流，可以通过音视频流对象的 {@link Stream#play Stream.play()} 方法进行音频和视频的播放渲染，Stream.play() 内部会自动创建音频播放器和视频播放器并将相应地

如果 App 想用自己的播放器，可以绕过 Stream.play()/stop() 方法调用，通过 {@link Stream#getAudioTrack Stream.getAudioTrack()}/{@link Stream#getVideoTrack Stream.getVideoTrack()} 方法获取相应的音视频轨道，然后利用自己的播放器进行音视频的播放渲染。使用这种自定义播放渲染方式后，Stream.on('player-state-changed') 事件将不会被触发，App 需要自行监听音视频轨道 MediaStreamTrack 的 mute/unmute/ended 等事件来判断当前音视频数据流的状态。

同时，App 层需要监听 Client.on('stream-added') 、Client.on('stream-updated') 和 Client.on('stream-removed') 等事件来处理音视频流的生命周期。

⚠ 注意：

- 在 'stream-added' 与 'stream-updated' 两个事件的处理回调中，都必须检查是否有音频或视频 track，在 ’ stream-updated ‘ 事件处理中，如果有音频或视频 track，那么请务必更新播放器并使用最新的音视频 track 进行播放。
- 线上示例

自定义音频采集和播放 Android&iOS&Windows&Mac

最近更新时间：2022-06-20 17:05:11

本文档主要介绍如何使用 TRTC SDK 实现自定义音频采集和获取，分为：音频采集、音频获取两个部分。

自定义音频采集

TRTC SDK 的自定义音频采集功能的开启分为两步，即：开启功能、发送音频帧给 SDK，具体 API 使用步骤见下文，同时我们也提供有对应平台的的 API-Example：

- Android
- iOS
- Windows

开启自定义音频采集功能

首先，您需要调用 TRTCCloud 的 enableCustomAudioCapture 接口开启 TRTC SDK 自定义音频采集的功能，示例代码如下：

Android

```
TRTCCloud mTRTCCloud = TRTCCloud.shareInstance();
mTRTCCloud.enableCustomAudioCapture(true);
```

iOS&Mac

```
self.trtcCloud = [TRTCCloud sharedInstance];
[self.trtcCloud enableCustomAudioCapture:YES];
```

Windows

```
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->enableCustomAudioCapture(true);
```

发送自定义音频帧

然后您就可以使用 TRTCCloud 的 sendCustomAudioData 接口向 TRTC SDK 填充您自己的声音数据，示例代码如下：

Android

```
TRTCCloudDef.TRTCAudioFrame trtcAudioFrame = new TRTCCloudDef.TRTCAudioFrame();
trtcAudioFrame.data = data;
trtcAudioFrame.sampleRate = sampleRate;
trtcAudioFrame.channel = channel;
trtcAudioFrame.timestamp = timestamp;
mTRTCCloud.sendCustomAudioData(trtcAudioFrame);
```

iOS&Mac

```
TRTCAudioFrame *audioFrame = [[TRTCAudioFrame alloc] init];
audioFrame.channels = audioChannels;
audioFrame.sampleRate = audioSampleRate;
audioFrame.data = pcmData;

[self.trtcCloud sendCustomAudioData:audioFrame];
```

Windows

```
liteav::TRTCAudioFrame frame;
frame.audioFormat = liteav::TRTCAudioFrameFormatPCM;
frame.length = buffer_size;
frame.data = array.data();
frame.sampleRate = 48000;
frame.channel = 1;
getTRTCShareInstance()->sendCustomAudioData(&frame);
```

> ⚠ **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，我们也提供有对应平台的的 API-Example：

- Android：
- iOS
- Windows

设置音频回调函数

Android

```
mTRTCCloud.setAudioFrameListener(new TRTCCloudListener.TRTCAudioFrameListener() {
@Override
public void onCapturedRawAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {

}

@Override
public void onLocalProcessedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {

}
```

```
@Override
public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame, String s) {

}

@Override
public void onMixedPlayAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {

}

@Override
public void onMixedAllAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
// 详见TRTC-API-Example 中自定义渲染的工具类：com.tencent.trtc.mediashare.helper.CustomFrameRender
}
});
```

iOS&Mac

```
[self.trtcCloud setAudioFrameDelegate:self];
// MARK: - TRTCAudioFrameDelegate
- (void)onCapturedRawAudioFrame:(TRTCAudioFrame *)frame {
NSLog(@"onCapturedRawAudioFrame");
}

- (void)onLocalProcessedAudioFrame:(TRTCAudioFrame *)frame {
NSLog(@"onLocalProcessedAudioFrame");
}
- (void)onRemoteUserAudioFrame:(TRTCAudioFrame *)frame userId:(NSString *)userId {
NSLog(@"onRemoteUserAudioFrame");
}

- (void)onMixedPlayAudioFrame:(TRTCAudioFrame *)frame {
NSLog(@"onMixedPlayAudioFrame");
}
- (void)onMixedAllAudioFrame:(TRTCAudioFrame *)frame {
NSLog(@"onMixedAllAudioFrame");
}
```

Windows

```
// 设置音频数据自定义回调
liteav::ITRTCCloud* trtc_cloud = liteav::ITRTCCloud::getTRTCShareInstance();
trtc_cloud->setAudioFrameCallback(callback)

// 音频数据自定义回调
```

```
virtual void onCapturedRawAudioFrame(TRTCAudioFrame* frame) {
}

virtual void onLocalProcessedAudioFrame(TRTCAudioFrame* frame) {
}

virtual void onPlayAudioFrame(TRTCAudioFrame* frame, const char* userId) {
}

virtual void onMixedPlayAudioFrame(TRTCAudioFrame* frame) {
}
```

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。

Web

最近更新时间：2022-06-02 10:49:03

本文主要介绍本地流的自定义采集和音视频流的自定义播放渲染等高阶用法。

自定义采集

本地流在通过 {@link TRTC.createStream createStream()} 创建时，可以指定使用 SDK 的默认采集方式，

如下，从摄像头和麦克风采集音视频数据：

```
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
// local stream initialized success
});
```

或者，采集屏幕分享流：

```
const localStream = TRTC.createStream({ userId, audio: false, screen: true });
localStream.initialize().then(() => {
// local stream initialized success
});
```

上述两种本地流的创建方式都是使用 SDK 的默认采集方式。为了便于开发者对音视频流进行预处理，createStream 支持从外部音视频源创建本地流，通过这种方式创建本地流，开发者可以实现自定义采集，比如说：

- 可以通过使用 getUserMedia 采集摄像头和麦克风音视频流。
- 通过 getDisplayMedia 采集屏幕分享流。
- 通过 captureStream 采集页面中正在播放的音视频。
- 通过 captureStream 采集 canvas 画布中的动画。

采集页面中正在播放的视频源

```
// 检测您当前的浏览器是否支持从 video 元素采集 stream
const isVideoCapturingSupported = () => {
['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) => {
if (item in document.createElement('video')) {
return true;
}
});
return false;
};

// 检测您当前的浏览器是否支持从 video 元素采集 stream
```

```
if (!isVideoCapturingSupported()) {
console.log('your browser does not support capturing stream from video element');
return
}
// 获取您页面在播放视频的 video 标签
const video = document.getElementById('your-video-element-ID');
// 从播放的视频采集视频流
const stream = video.captureStream();
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];

const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSource: videoTrack });

// 请确保视频属性跟外部传进来的视频源一致，否则会影响视频通话体验
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
// local stream initialized success
});
```

采集 canvas 中的动画

```
// 检测您当前的浏览器是否支持从 canvas 元素采集 stream
const isCanvasCapturingSupported = () => {
['captureStream', 'mozCaptureStream', 'webkitCaptureStream'].forEach((item) => {
if (item in document.createElement('canvas')) {
return true;
}
});
return false;
};

// 检测您当前的浏览器是否支持从 canvas 元素采集 stream
if (!isCanvasCapturingSupported()) {
console.log('your browser does not support capturing stream from canvas element');
return
}
// 获取您的 canvas 标签
const canvas = document.getElementById('your-canvas-element-ID');

// 从 canvas 采集 15 fps 的视频流
const fps = 15;
const stream = canvas.captureStream(fps);
const videoTrack = stream.getVideoTracks()[0];
```

```
const localStream = TRTC.createStream({ userId, videoSource: videoTrack });

// 请确保视频属性跟外部传进来的视频源一致，否则会影响视频通话体验
localStream.setVideoProfile('480p');

localStream.initialize().then(() => {
// local stream initialized success
});
```

自定义播放渲染

对于通过 TRTC.createStream() 创建并初始化好的本地流或者通过 Client.on('stream-added') 接收到的远端流，可以通过音视频流对象的 {@link Stream#play Stream.play()} 方法进行音频和视频的播放渲染，Stream.play() 内部会自动创建音频播放器和视频播放器并将相应地

如果 App 想用自己的播放器，可以绕过 Stream.play()/stop() 方法调用，通过 {@link Stream#getAudioTrack Stream.getAudioTrack()}/{@link Stream#getVideoTrack Stream.getVideoTrack()} 方法获取相应的音视频轨道，然后利用自己的播放器进行音视频的播放渲染。使用这种自定义播放渲染方式后，Stream.on('player-state-changed') 事件将不会被触发，App 需要自行监听音视频轨道 MediaStreamTrack 的 mute/unmute/ended 等事件来判断当前音视频数据流的状态。

同时，App 层需要监听 Client.on('stream-added') 、Client.on('stream-updated') 和 Client.on('stream-removed') 等事件来处理音视频流的生命周期。

> ⚠ 注意：
>
> - 在 'stream-added' 与 'stream-updated' 两个事件的处理回调中，都必须检查是否有音频或视频 track，在 ’ stream-updated ‘ 事件处理中，如果有音频或视频 track，那么请务必更新播放器并使用最新的音视频 track 进行播放。
> - 线上示例

发送和接收消息

最近更新时间：2022-06-08 17:59:45

内容介绍

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。

支持的平台

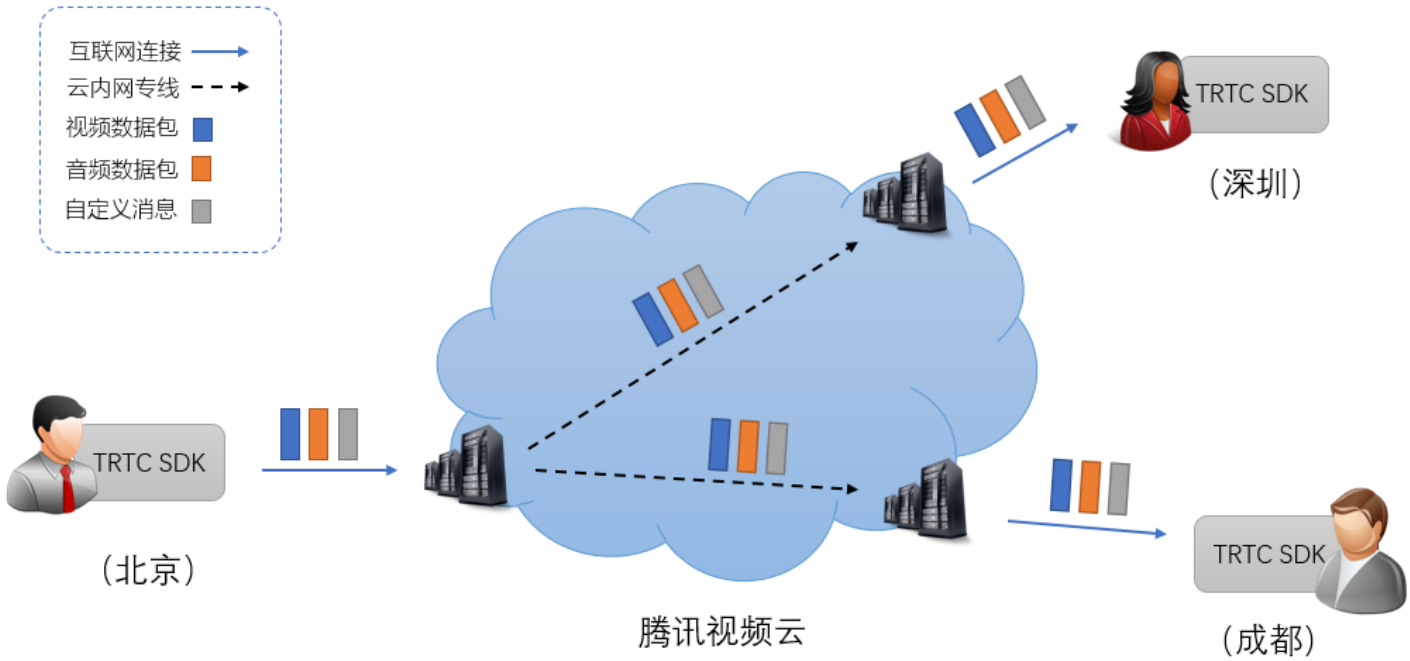
iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
✓	✓	✓	✓	✓	×	×

说明：

Web 端和微信小程序端客户，可通过 IM 提供的时通讯来实现。登录IM后发送消息来进行业务逻辑处理，详情请参见 Web&小程序&uni-app SDK API。

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。
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:(NS
Data *)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。

监听服务端事件回调

最近更新时间：2022-06-02 10:49:17

事件回调服务支持将实时音视频业务下的事件，以 HTTP/HTTPS 请求的形式通知到您的服务器。事件回调服务已集成房间事件组（Room Event）和媒体事件组（Media Event）下的一些事件，您可以向腾讯云提供相关的配置信息来开通该服务。

配置信息

实时音视频 TRTC 控制台支持自助配置回调信息，配置完成后即可接收事件回调通知。详细操作指引请参见 回调配置。

> 注意：
> 您需要提前准备以下信息：
> - **必要项**：接收回调通知的 HTTP/HTTPS 服务器地址。
> - **可选项**：计算签名的密钥 key，由您自定义一个最大32个字符的 key，以大小写字母及数字组成。

超时重试

事件回调服务器在发送消息通知后，5秒内没有收到您的服务器的响应，即认为通知失败。首次通知失败后会立即重试，后续失败会以10秒的间隔继续重试，直到消息存续时间超过1分钟，不再重试。

事件回调消息格式

事件回调消息以 HTTP/HTTPS POST 请求发送给您的服务器，其中：

- **字符编码格式**：UTF-8。
- **请求**：body 格式为 JSON。
- **应答**：HTTP STATUS CODE = 200，服务端忽略应答包具体内容，为了协议友好，建议客户应答内容携带 JSON：{"code":0}。
- **包体示例**：下述为“房间事件组-进入房间”事件的包体示例。

```
{
"EventGroupId": 1, #房间事件组
"EventType": 103, #进入房间事件
"CallbackTs": 1615554923704, #回调时间，单位毫秒
"EventInfo": {
"RoomId": 12345, #数字房间号
"EventTs": 1615554922, #事件发生时间，单位秒
"UserId": "test", #用户ID
"UniqueId": 1615554922656, #唯一标识符
"Role": 20, #用户角色，主播
"TerminalType": 3, #终端类型，IOS端
"UserType": 3, #用户类型，Native SDK
"Reason": 1 #进房原因，正常进房
}
}
```

参数说明

回调消息参数

- 事件回调消息的 header 中包含以下字段：

字段名	值
Content-Type	application/json
Sign	签名值
SdkAppId	sdk application id

- 事件回调消息的 body 中包含以下字段：

字段名	类型	含义
EventGroupId	Number	事件组 ID
EventType	Number	回调通知的事件类型
CallbackTs	Number	事件回调服务器向您的服务器发出回调请求的 Unix 时间戳，单位为毫秒
EventInfo	JSON Object	事件信息

事件组 ID

字段名	值	含义
EVENT_GROUP_ROOM	1	房间事件组
EVENT_GROUP_MEDIA	2	媒体事件组

事件类型

字段名	值	含义
EVENT_TYPE_CREATE_ROOM	101	创建房间
EVENT_TYPE_DISMISS_ROOM	102	解散房间
EVENT_TYPE_ENTER_ROOM	103	进入房间
EVENT_TYPE_EXIT_ROOM	104	退出房间
EVENT_TYPE_CHANGE_ROLE	105	切换角色
EVENT_TYPE_START_VIDEO	201	开始推送视频数据
EVENT_TYPE_STOP_VIDEO	202	停止推送视频数据
EVENT_TYPE_START_AUDIO	203	开始推送音频数据
EVENT_TYPE_STOP_AUDIO	204	停止推送音频数据
EVENT_TYPE_START_ASSIT	205	开始推送辅路数据

字段名	值	含义
EVENT_TYPE_STOP_ASSIT	206	停止推送辅路数据

事件信息

字段名	类型	含义
RoomId	String/Number	房间名（类型与客户端房间号类型一致）
EventTs	Number	事件发生的 Unix 时间戳，单位为秒（兼容保留）
EventMsTs	Number	事件发生的 Unix 时间戳，单位为毫秒
UserId	String	用户 ID
UniqueId	Number	唯一标识符（option：房间事件组携带） 当客户端发生了一些特殊行为，例如切换网络、进程异常退出及重进等，此时您的回调服务器可能会收到同一个用户多次进房和退房回调，UniqueId 可用于标识用户的同一次进退房
Role	Number	角色类型（option：进退房时携带）
TerminalType	Number	终端类型（option：进房时携带）
UserType	Number	用户类型（option：进房时携带）
Reason	Number	具体原因（option：进退房时携带）

> **注意：**
> 我们已发布“过滤客户端特殊行为导致的重复回调”策略。如果您是2021年07月30日之后接入回调服务，默认走新策略，房间事件组不再携带 UniqueId（唯一标识符）。

角色类型

字段名	值	含义
MEMBER_TRTC_ANCHOR	20	主播
MEMBER_TRTC_VIEWER	21	观众

终端类型

字段名	值	含义
TERMINAL_TYPE_WINDOWS	1	Windows 端
TERMINAL_TYPE_ANDROID	2	Android 端
TERMINAL_TYPE_IOS	3	iOS 端
TERMINAL_TYPE_LINUX	4	Linux 端
TERMINAL_TYPE_OTHER	100	其他

用户类型

字段名	值	含义
USER_TYPE_WEBRTC	1	webrtc
USER_TYPE_APPLET	2	小程序
USER_TYPE_NATIVE_SDK	3	Native SDK

具体原因

字段名	含义
进房	• 1：正常进房 • 2：切换网络 • 3：超时重试 • 4：跨房连麦进房
退房	• 1：正常退房 • 2：超时离开 • 3：房间用户被移出 • 4：取消连麦退房 • 5：强杀 **注意：Android 系统无法捕捉进程被强杀，只能等待后台超时离开，此时回调 reason 为2**

计算签名

签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：

```
//签名 Sign 计算公式中 key 为计算签名 Sign 用的加密密钥。
Sign = base64 ( hmacsha256(key, body) )
```

注意：

body 为您收到回调请求的原始包体，不要做任何转化，示例如下：

```
body="{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t103,\n\t\"CallbackTs\":\t1615554923704,\n\t\"EventInfo\":\t{\n\t\t\"RoomId\":\t12345,\n\t\t\"EventTs\":\t1608441737,\n\t\t\"UserId\":\t\"test\",\n\t\t\"UniqueId\":\t1615554922656,\n\t\t\"Role\":\t20,\n\t\t\"Reason\":\t1\n\t}\n}"
```

打通 RTMP 推流协议

最近更新时间：2022-06-02 10:49:25

打通 RTMP 推流协议

方案背景

为降低客户接入门槛，TRTC 支持 RTMP 标准协议推流，您可根据实际情况选择安装 OBS 或 FFmpeg 进行推流。OBS 是一款好用的第三方开源程序直播流媒体内容制作软件，为用户提供免费使用，它可支持 OS X、Windows、Linux 操作系统，适用多种直播场景，满足大部分直播行为的操作需求，您可以到 OBS 官网 下载最新版本软件，使用 OBS 推流时无需安装插件。

本功能目前免费开放内测中（后续若收费会提前通知），但接入的 RTMP 流会作为房间中的虚拟用户产生正常的通话费用，详情参见 计费概述。如果您需要申请使用此功能，请单击 立即申请。

> ⚠ 注意：
> 不支持 RTMP 从 TRTC 拉流，如果需要旁路 CDN 直播观看，请参见 实现 CDN 直播观看。

应用场景

场景类型	说明
在线教育场景	老师展示视频课件教学视频时，可以通过 PC 端 OBS 或者 FFmpeg 把绝大多数媒体格式以 RTMP 推流至 TRTC 房间，房间内的学生通过 TRTC SDK 拉流，可以保证观看到相同进度的教学视频，课件播放跳转进度、调整速度、切换下一章等全部可由老师控制，各学生端观看对齐课堂秩序好，教学质量更稳定。
一起看球赛场景	比赛流媒体是赛事供应方固定以 RTMP 格式流的方式提供赛事画面，通过 RTMP 协议推流至 TRTC 房间，实现 TRTC 房间内同步观看超低延时的比赛直播，配合 TRTC 的实时互动能力，与好友语音/视频讨论，一起喝彩加油，不会错过每一个精彩瞬间的共享体验。
更多场景	任何基于媒体流的实时互动体验玩法，均可通过 RTMP 协议推流帮您实现，等多玩法等待您的探索。

OBS 推流设置

准备工作

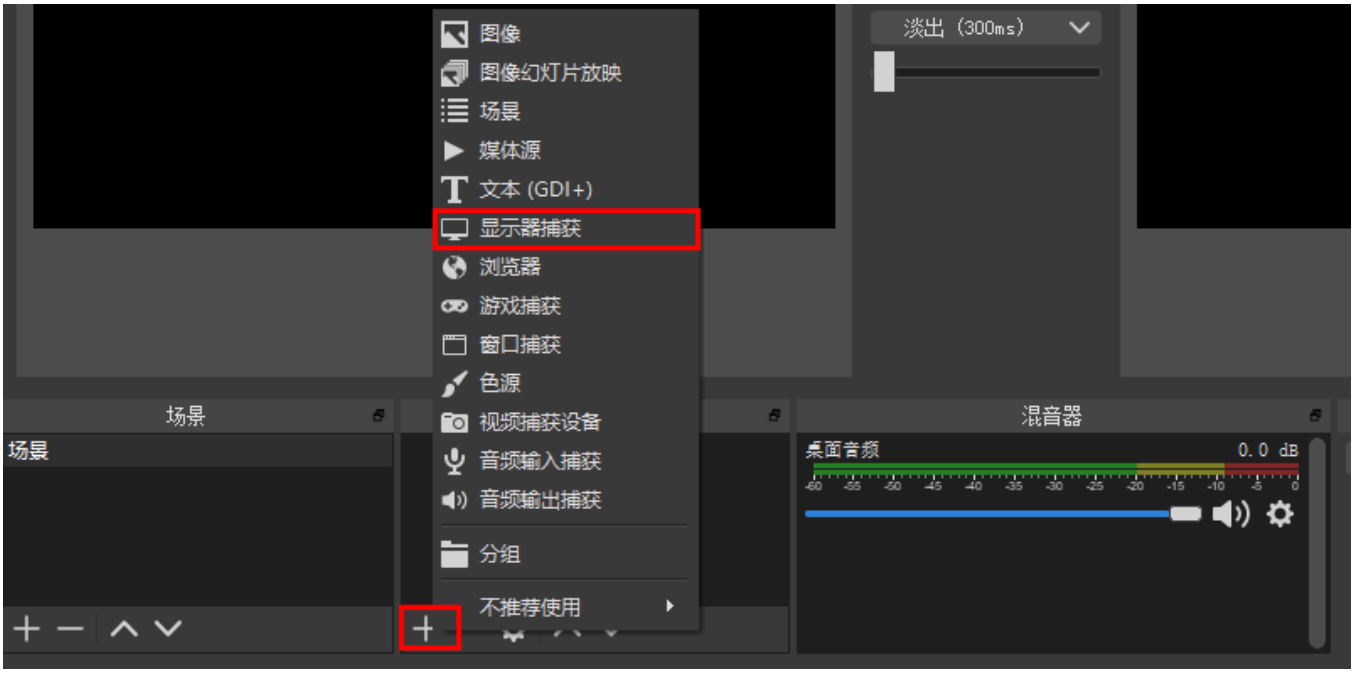
安装并打开 OBS 工具进行下述操作。

步骤1：选择输入源

查看底部工具栏的**来源**标签，单击 + 按钮，根据您的业务需要选择输入源。常用来源输入有：

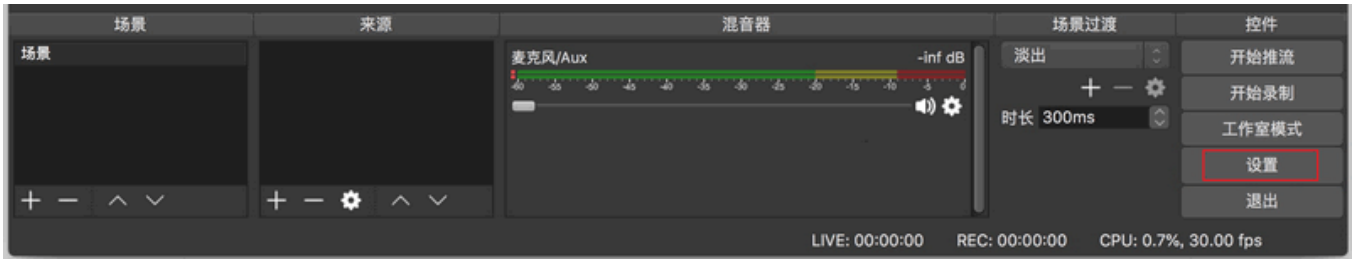
输入源	说明
图像	适用于单张图像直播
图像幻灯片放映	可循环或者顺序多张播放图片
场景	实现各种强大的直播效果。此时，另一个场景是作为来源被添加进当前场景的，可以实现整个场景的插入

输入源	说明
媒体源	可上传本地视频，并本地点播视频文件进行直播化处理
文本	实时添加文字在直播窗口中
窗口捕获	可根据您选择的窗口进行实时捕获，直播仅显示您当前窗口内容，其他窗口不会进行直播捕获
视频捕获设备	实时动态捕捉摄像设备，可将摄像后的画面进行直播
音频输入捕获	用于音频直播活动（音频输入设备）
音频输出捕获	用于音频直播活动（音频输出设备）



步骤2：设置推流参数

1. 通过底部工具栏的 **控件>设置** 按钮进入设置界面。



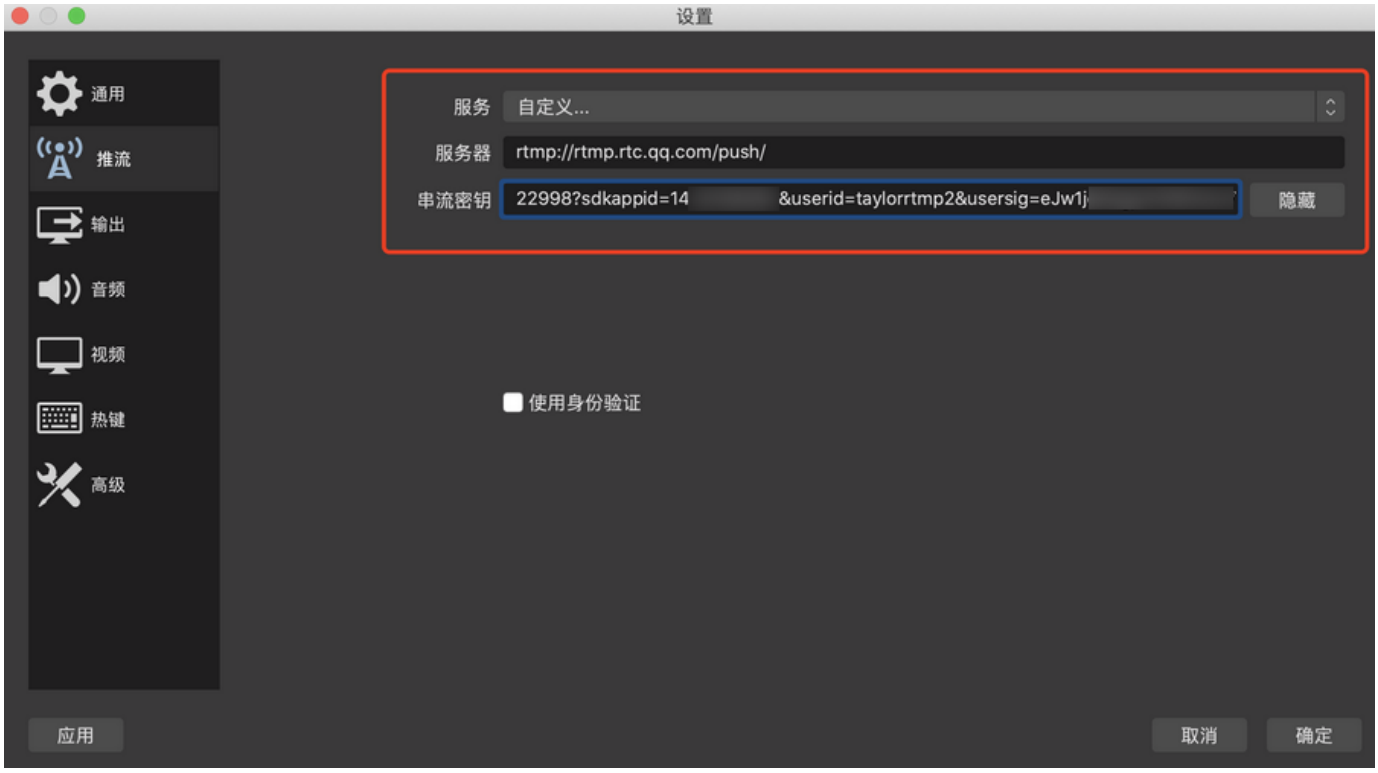
2. 单击 **推流** 进入推流设置页签，选择服务类型为**自定义**。
3. 服务器填写：rtmp://rtmp.rtc.qq.com/push/。
4. 填写串流密钥格式如下：

```
房间号?sdkappid=应用&userid=用户名&usersig=签名
```

其中房间号、应用、用户名、签名需要换成业务的，例如：

22998?sdkappid=140*****66&userid=******rtmp2&usersig=eJw1jdE**********ZLgi5UAgOzoMhrayt*cjbmiCJ699T09juc833IMT94Ld7I0iHZqVDzvVAqkZsG-IKlzLiXOnEhswHu1iUyTc9pv*****D8MQwoA496Ke6U1ip4EAH4UMc5H9pSmv6MeTBWLamhwFnWRBZ8qKGRj8Yp-wVbv*mGMVZqS7w-mMDQL

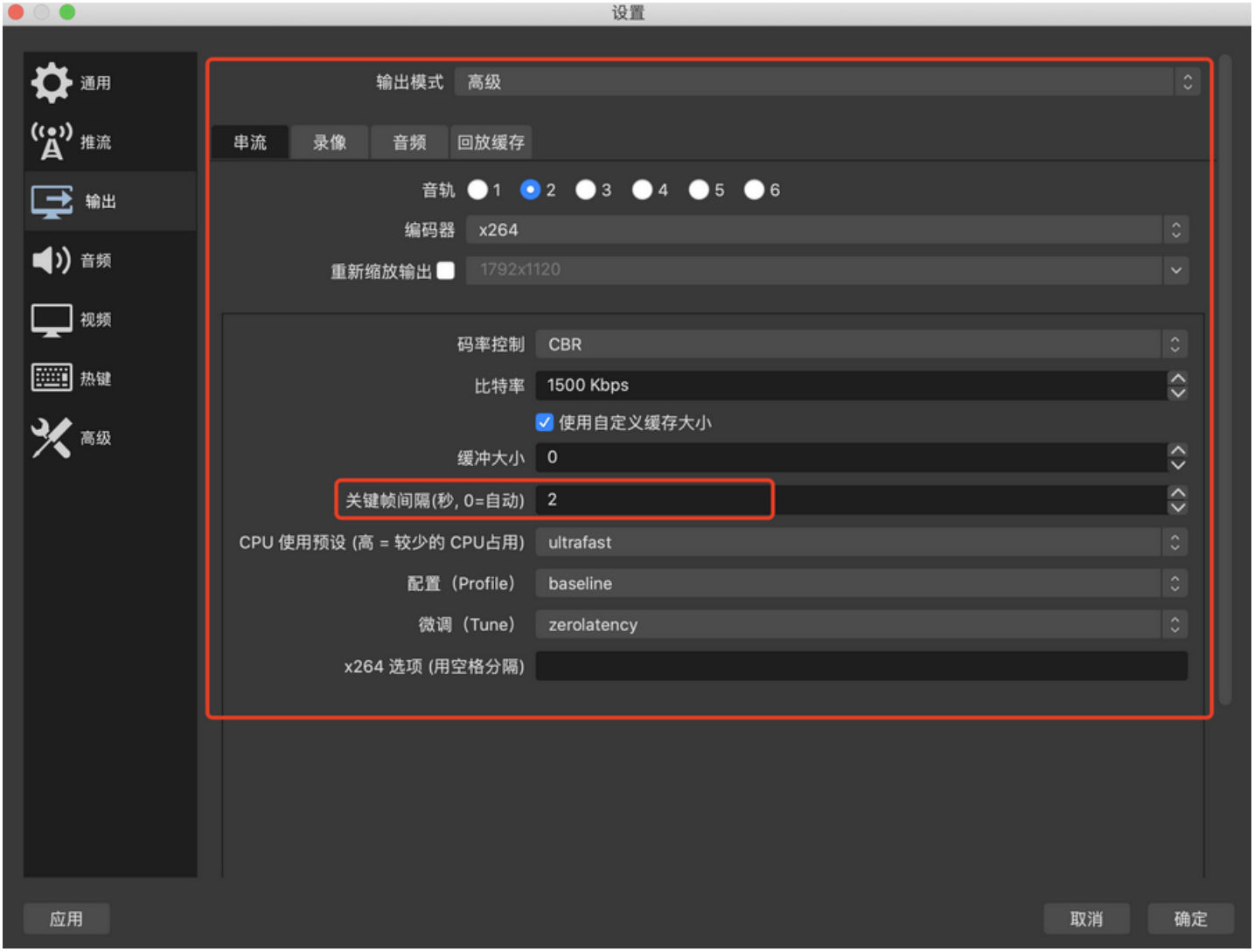
- 为简化参数，只支持字符串房间号，不超过64个字符，字符只能是数字、字母、下划线。TRTC 其他端如果要观看 RTMP 流，需要使用**字符串房间号进房**。
- usersig 的生成规则，请参见 UserSig 相关（**请注意签名要在有效期内**）。
- 以上服务器地址 + 串流密钥组成 RTMP 推流地址，也可以供 FFmpeg 或其他 RTMP 库推流。



步骤3：设置输出

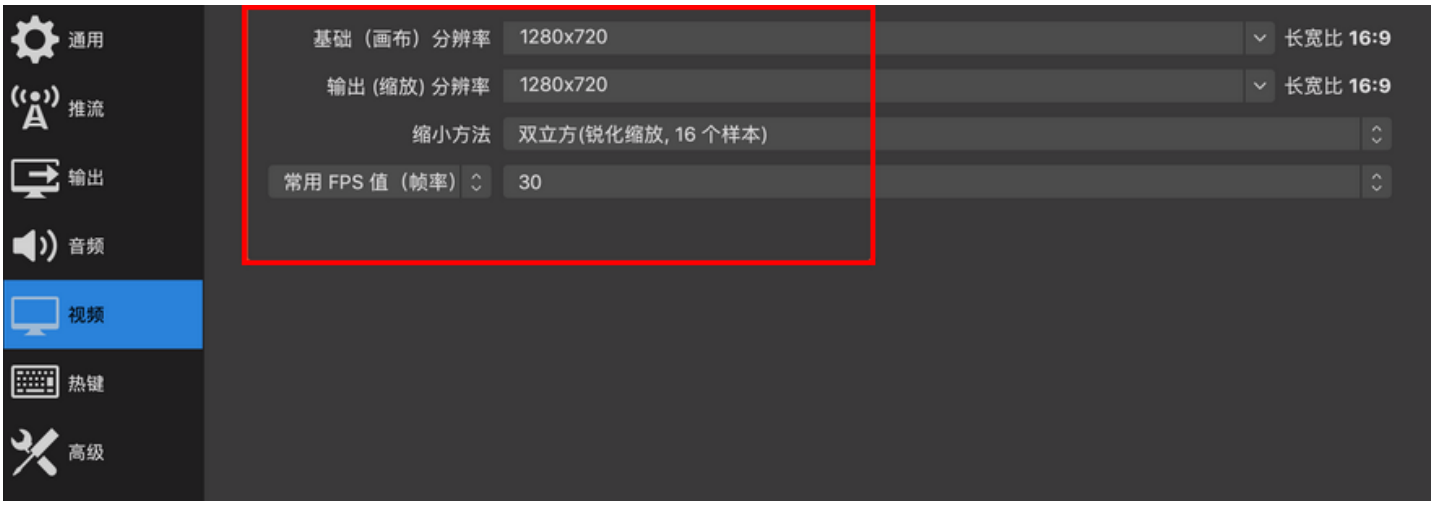
RTMP 后台不支持传输 B 帧，用户可以通过如下设置调整推流端软件的视频编码参数来去除 B 帧。

1. 在**设置**中单击**输出**页签进行配置。
2. 在**输出模式**中选择**高级**，**关键帧间隔建议填写1或2**，单击**确定**保存设置。



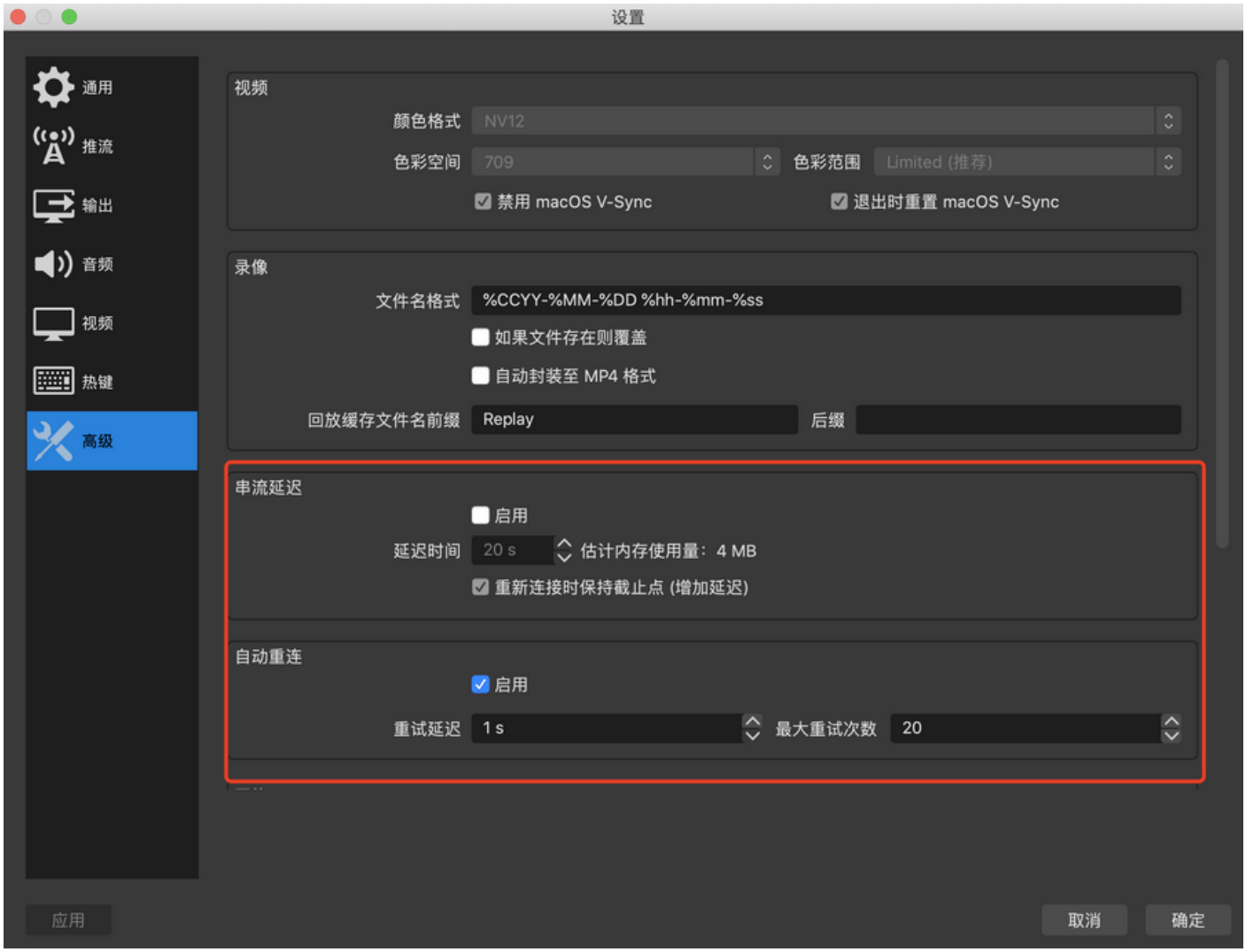
步骤4：设置视频选项

在**设置**中单击**视频**页签，设置分辨率和帧率。分辨率决定了观众看到的画面清晰程度，分辨率越高画面越清晰。FPS 是视频帧率，它控制观看视频的流畅，普通视频帧率有24帧 – 30帧，低于16帧画面看起来有卡顿感，而游戏对帧率要求比较高，一般小于30帧游戏会显得不连贯。



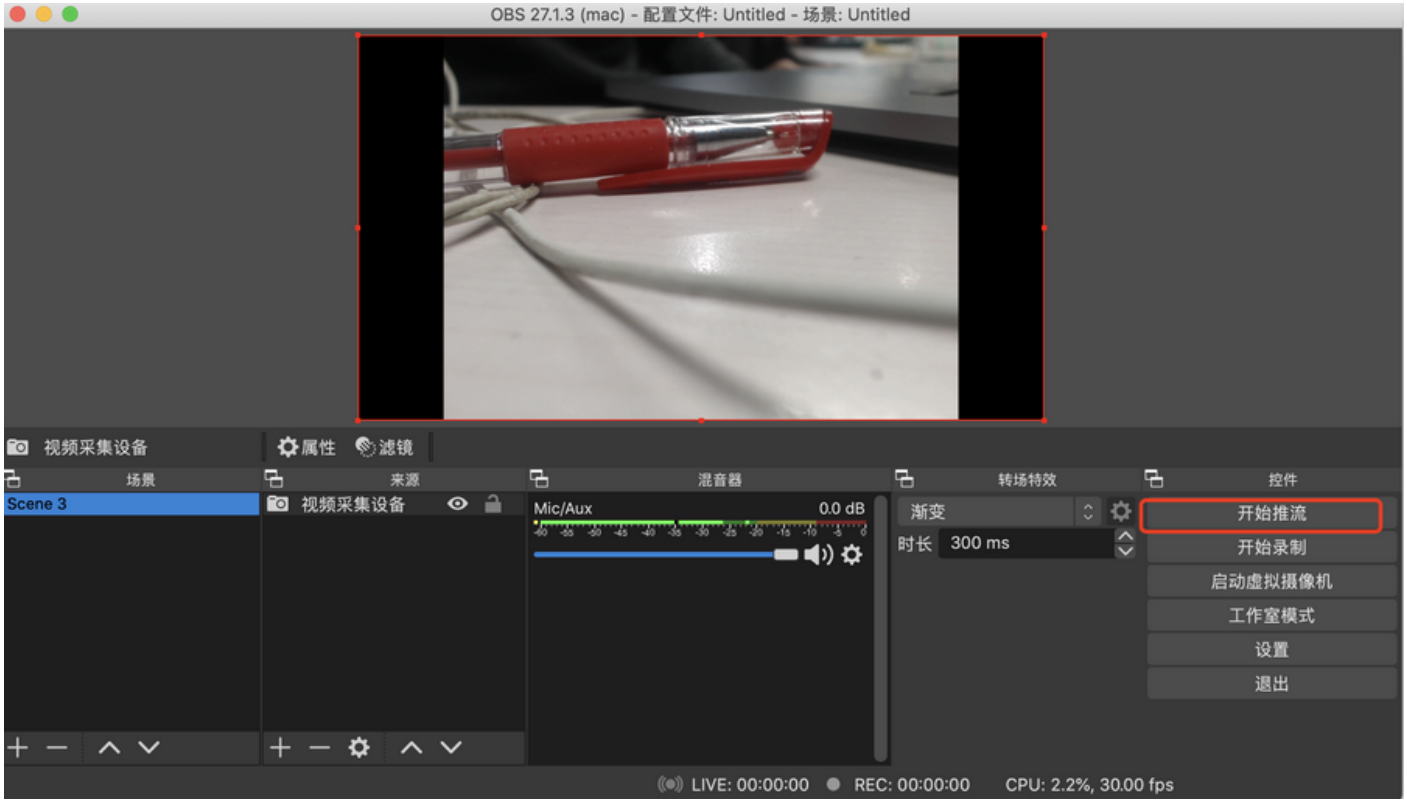
步骤5：设置高级选项

- 建议不启用**串流延迟**以减少端到端延迟。
- 启动**自动重连**，建议设置**重试延迟**时长尽量短，网络抖动时如果连接断开可尽快重连上。

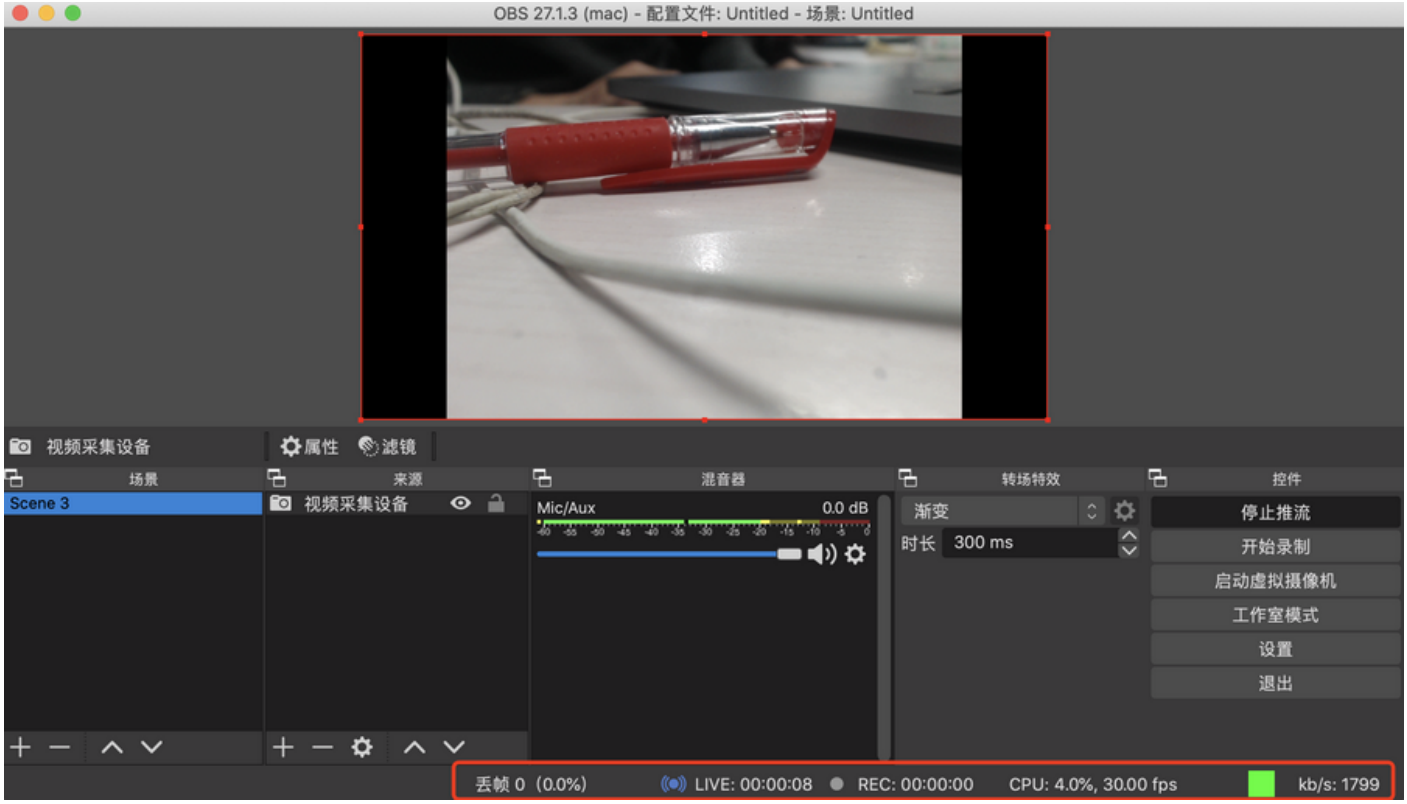


步骤6：单击推流

1. 查看 OBS 底部工具栏的 **控件**，单击 **开始推流**。

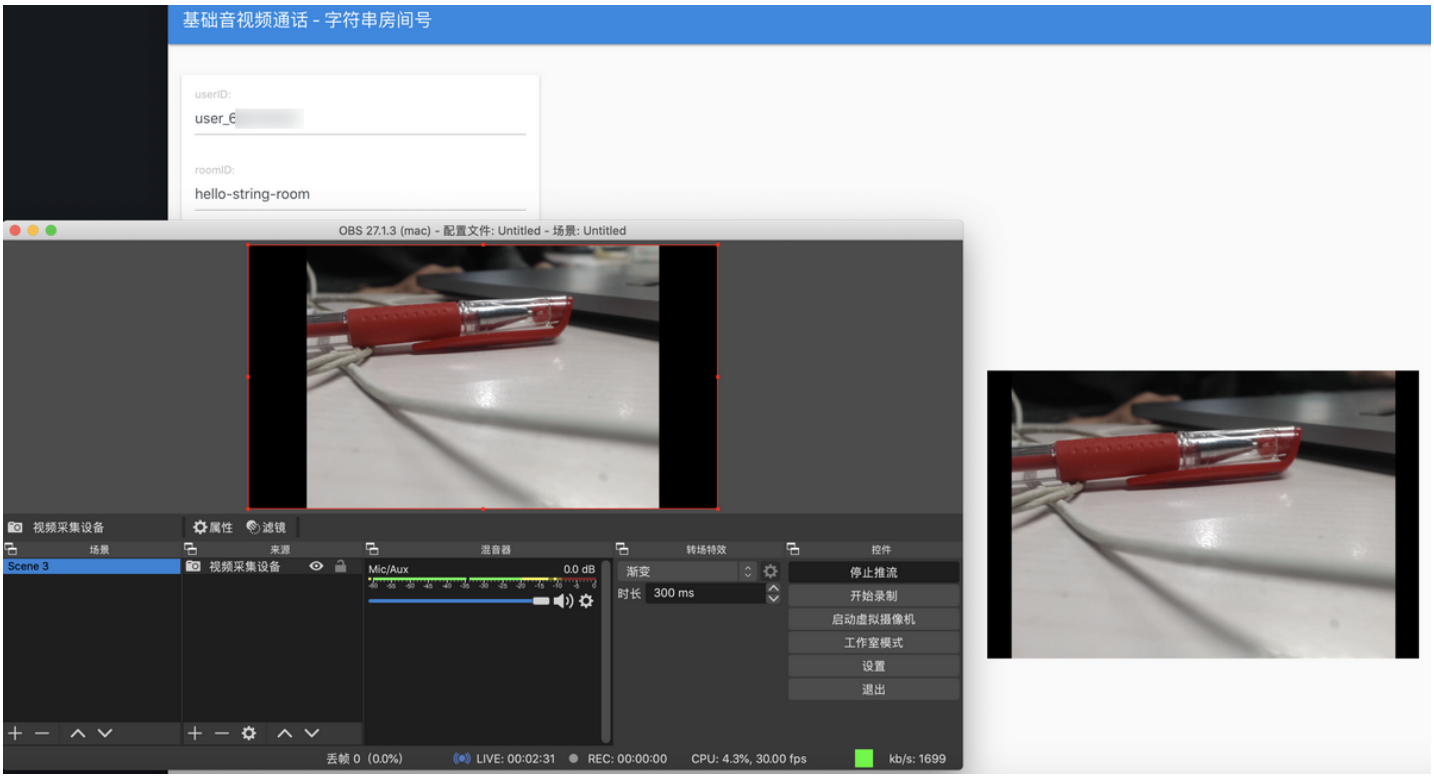


2. 推流成功后，正常情况在界面底部会展示推流状态，TRTC 控制台仪表盘上有该用户进房记录。



步骤7：其他端观看

如前面 设置推流参数 所说，TRTC 其他端进房需要使用字符串房间号，Web 端 观看 RTMP 流的效果如下所示：



FFmpeg 设置

如果需要用命令行或其他 RTMP 库推流，请参见 设置推流参数 中服务器地址 + 串流密钥组成标准 RTMP 推流地址，可供 FFmpeg 或其他 RTMP 库推流，视频编码使用 H.264，音频编码使用 AAC，容器格式使用 FLV，建议 GOP 设置为2s或1s。
FFmpeg 不同场景下指令配置参数不同，因此需要您具有一定的 FFmpeg 使用经验，以下列出 FFmpeg 常用命令行选项，更多 FFmpeg 选项请参见 FFmpeg 官网。

FFmpeg 命令行

```
ffmpeg [global_options] {[input_file_options] -i input_url} ... {[output_file_options] output_url}
```

常见的 FFmpeg 选项

选项	说明
-re	以 native 帧率读取输入，通常只用于读取本地文件

其中 output_file_options 可配置选项包括：

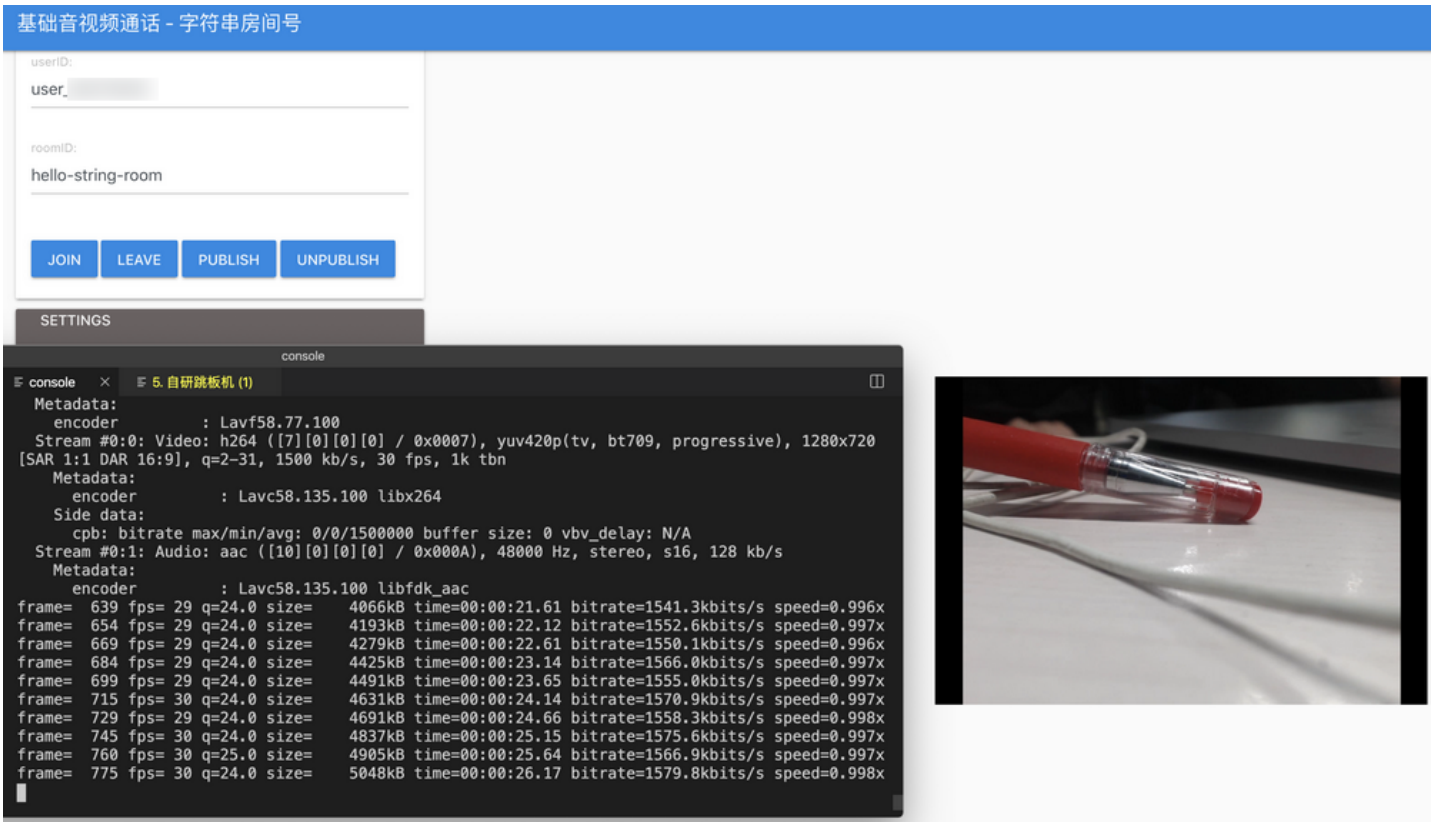
选项	说明
-c:v	视频编码，建议用 libx264
-b:v	视频码率，例如 1500k 表示 1500kbps
-r	视频帧率

选项	说明
-profile:v	视频 profile，指定 baseline 将不编码 B 帧，TRTC 后端不支持 B 帧
-g	GOP 帧数间隔
-c:a	音频编码，建议用 libfdk_aac
-ac	声道数，填2或1
-b:a	音频码率
-f	指定格式，固定填 flv，发送到 TRTC 使用 FLV 容器封装

下面的例子是读取文件推到 TRTC。

```
ffmpeg -loglevel info -re -i sample.flv -c:v libx264 -preset fast -profile:v baseline -g 30 -sc_threshold 0 -b:v 1500k -c:a libfdk_aac -ac 2 -b:a 128k -f flv 'rtmp://rtmp.rtc.qq.com/push/hello-string-room?userid=rtmpForFfmpeg&sdkappid=140xxxxxx&usersig=xxxxxxxxxx'
```

Web 端观看效果



音视频内容安全审核
接入指引

最近更新时间：2022-05-05 14:35:55

音视频内容安全审核由 T-Sec 天御 提供实时的音视频内容识别与审核服务，客户只需要调用天御音视频流接口即可实时检测音视频流中是否出现违规内容，音视频安全审核服务会通过回调把违规信息发送给客户指定的回调 URL。

实现原理

直播内容安全通过“哑终端”的形式进入指定的 TRTC 房间，作为“观众”拉取音视频流，并针对拉取到音视频流进行内容审核，然后通过回调把违规信息发送到用户指定的 HTTP/HTTPS 服务上。

开通服务

1. 进入 腾讯云 TRTC 控制台，开通 TRTC 服务。TRTC 会分配 sdk_app_id，在创建内容审核时需要用到。（已经成功接入 TRTC 可以忽略该步骤）。
2. 进入 内容安全控制台，选择要使用的产品。选择直播音频或者直播视频时：
   - 如果是实时语音聊天应用，则选择直播音频内容安全 [ams]。
   - 如果是实时视频通话或者单人/多人互动直播，则选择直播视频内容安全 [vm]。

3. 选择产品后会弹出服务开通界面，单击**开通**。

> ⚠ 注意：
> 由于音视频产品需要使用用户的 COS 桶进行存储审核的音频切片和图片帧，单击**开通**后，需要用户授权内容安全产品访问用户 COS 桶，单击**授权**后，会弹出授权成功界面。

4. 开通服务后，在服务管理可以看到授权的 COS 桶信息。也可以在 COS 的控制台界面中看到该 COS 桶，用户可以根据实际情况在 COS 控制台中设置授权的 COS 桶中音频切片/图片截帧信息的存储周期。

5. 开通完服务，用户可以在 腾讯云接口测试界面 发起审核，接入流程见 审核接入流程。

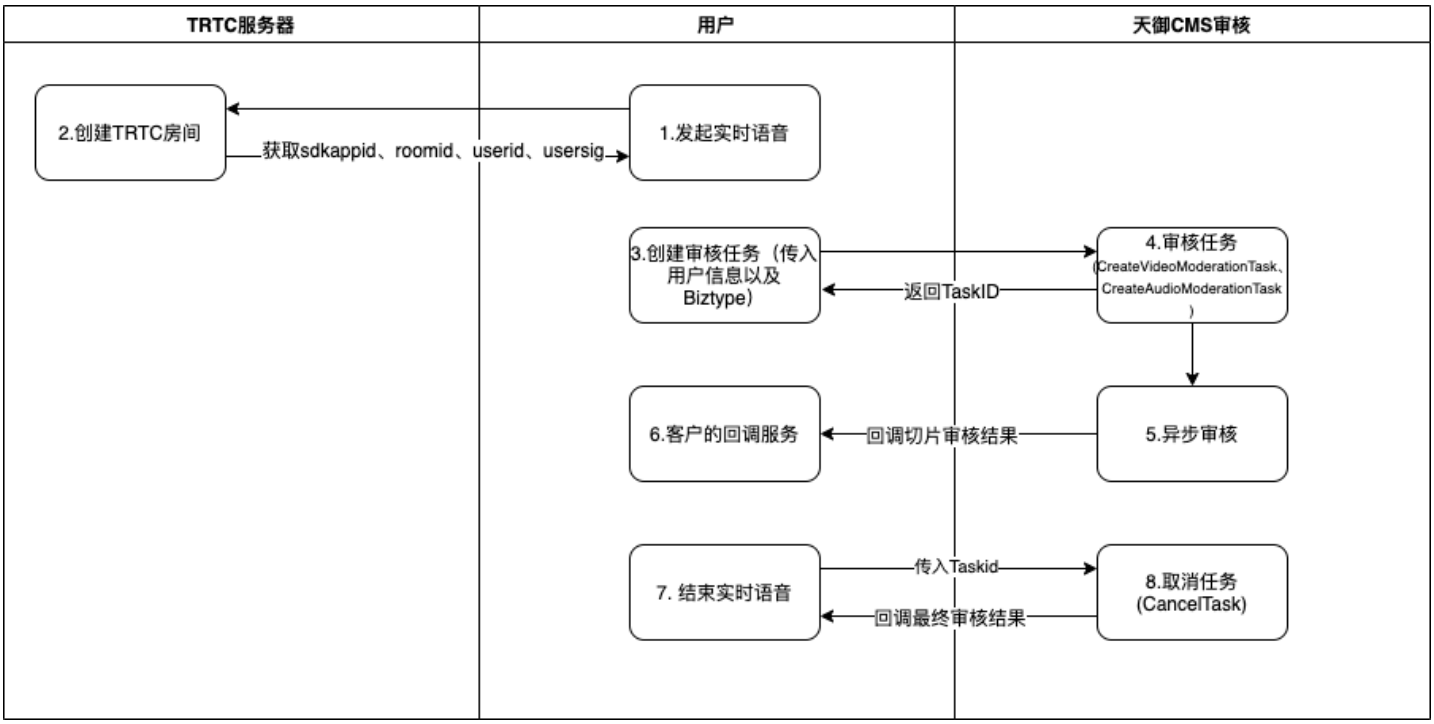
> **说明：**
> 天御音视频内容审核费用由 T-Sec 天御收取，详情请参见 视频内容安全计费方式 和 音频内容安全计费方式。

接入流程

最近更新时间：2022-05-09 10:26:59

审核接入流程

接入流程图



接入步骤

1. 用户通过 TRTC 提供的 SDK 创建房间发起实时音视频后，会获取到 sdk_app_id（TRTC 控制台创建应用时的应用 ID），room_id（创建的房间号），然后通过 TRTC SDK 生成一个该房间的内容审核用户（建议每个房间生成不同内容审核的 user_id），此时可以获取到 user_id 和 user_sig（内容安全将使用该用户作为观众进入该房间拉取直播流数据）。
2. 将上一步获取到的 sdk_app_id、room_id、user_id 和 user_sig 拼装成 TRTC 审核 URL，URL 格式如下：

```
trtc://trtc.tencentcloudapi.com/moderation?sdk_app_id=xxxx&room_id=xxxx&user_id=xxxx&user_sig=xxxx
```

> 注意：
>
> - URL 中的每个参数值在拼装前需要先 escape 一下，防止有特殊字符无法解析，特别是 user_sig。
> - 本文档涉及的 TRTC 相关参数：sdk_app_id、user_id、user_sig、room_id，分别对应您接入 TRTC SDK 中 SdkAppId、UserId、UserSig、RoomId。

参数名称	必选	描述
sdk_app_id	是	TRTC 控制台创建应用时的应用 ID。

参数名称	必选	描述
room_id	是	创建的房间号，TRTC 的 room_id 有数值和字符串两种形式，默认是数值，如果为字符串的 room_id，则需要将 room_id_type 设置为 string。
user_id	是	用户 ID，建议每个房间生成不同内容审核的 user_id。
user_sig	是	user_sig 是基于 sdk_app_id 和 user_id 计算出的安全签名。
mix	否	取值时 true 为混流审核，false 为单流审核。不填默认是混流审核。
room_id_type	否	房间类型，默认可不传，可传入值为：string/number

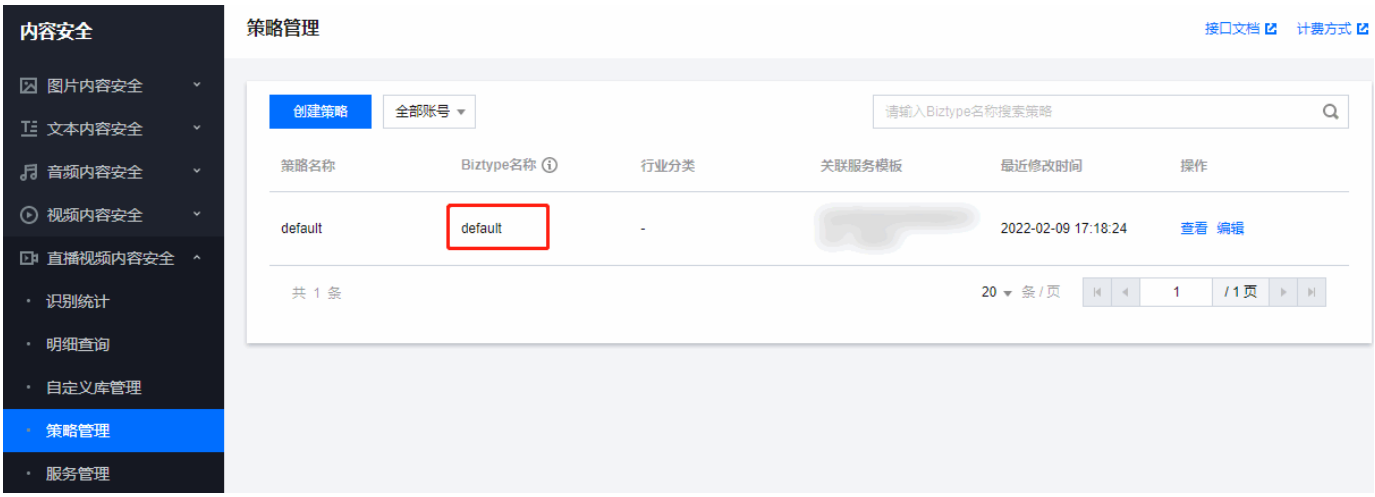
3. 通过 创建审核任务 接口，发起该房间的内容安全审核。

参数名称	必选	类型	描述
Action	是	String	公共参数，本接口取值：CreateAudioModerationTask。
Version	是	String	公共参数，本接口取值：2020-12-29。
Region	否	String	公共参数，本接口不需要传递此参数。
Tasks.N	是	Array of TaskInput	该字段表示输入的音频审核任务信息，具体输入内容请参见 TaskInput 数据结构的详细描述。 备注：最多同时可创建**10个任务**。
BizType	否	String	该字段表示策略的具体编号，用于接口调度，在内容安全控制台中可配置。若不传入 Biztype 参数（留空），则代表采用默认的识别策略；传入则会在审核时根据业务场景采取不同的审核策略。 备注：Biztype 仅为数字、字母与下划线的组合，长度为3个-32个字符；不同 Biztype 关联不同的业务场景与识别能力策略，调用前请确认正确的 Biztype。
Type	否	String	该字段表示输入的音频审核类型，取值为：**AUDIO**（点播音频）和 **LIVE_AUDIO**（直播音频），默认值为 AUDIO。
Seed	否	String	可选参数，该字段表示回调签名的 key 信息，用于保证数据的安全性。 签名方法为在返回的 HTTP 头部添加 X-Signature 的字段，值为： seed + body 的 SHA256 编码和 Hex 字符串，在收到回调数据后，可以根据返回的body，用 **sha256(seed + body)**，计算出 X-Signature 进行验证。 具体使用实例请参见 回调签名示例。
CallbackUrl	否	String	可选参数，该字段表示接受审核信息回调的地址，格式为 URL 链接默认格式。配置成功后，审核过程中产生的违规音频片段将通过此接口发送。回调返回内容格式请参见 回调签名示例。

该接口需注意以下几点：

- **BizType**
  用户的审核策略，可以通过控制台上的 策略管理 根据不同的需求创建不同的审核策略，在开通服务时，会自动创建一个 BizType

为“default”的审核策略，测试时，可使用该 BizType 传入。



- **Type**
  根据实际的应用类型进行传入，TRTC实时音视频根据应用类型调用不同的产品的接口。（例如：实时语音房场景调用 ams 产品的接口，Type 输入 LIVE_AUDIO；视频聊天调用 vm 产品的接口，Type 填写 LIVE_VIDEO）。
- **CallbackUrl**
  该地址用于用户接口直播过程中和直播结束的审核结果，可以在控制台配置界面中设置，也可以通过接口传入。回调格式同 查询任务详情接口 的输出参数。
- **Tasks.N.Input.Url**
  填入 第2步 拼装的 TRTC 审核 URL。输入示例：

```
p = {
    "BizType": "default", # 通过控制台策略管理创建
    "Type": "LIVE_VIDEO",   # LIVE_VIDEO、LIVE_AUDIO为直播音视频
    "CallbackUrl": "callback_server_address", # 回调地址
    "Tasks": [{
        "DataId": "test_0020211025", # 数据ID，由用户传入，可方便对文件进行标识和管理
        "Name": "test测试", # 任务名称
        "Input": {
            "Type": "URL",  # 视频资源类型，目前仅支持URL方式传入
            "Url": "video_url"   # 视频资源链接
        }
    }]
}
```

4. 创建审核任务成功，腾讯云内容安全审核服务会返回 TaskId、TaskId 作为内容审核针对该房间审核的唯一ID。

5. 在审核过程中，用户的回调服务器会持续接收到内容安全审核服务的审核结果，针对 TRTC 是在直播过程中，每个用户的音频切片和图片截帧的审核结果。用户可以根据该结果决定是否封禁直播间，或者是针对直播间发出警告。

6. 在直播结束后，用户需要调用结束审核接口（传入创建审核时的 TaskId），关闭直播间审核。

7. 内容安全审核服务在接收到结束审核请求后，停止拉取该直播间的数据流，并根据审核策略，推导出该直播间的最终审核结果，同时发送审核结束的回调信息给用户。TRTC 回调的片段会存储在 cos 上面，文件名为：

```
trtc/{{sdk_app_id}}/screenshot_{{room_id}}_{{user_id}}{{timestamp}}.jpg（图片格式）
trtc/{{sdk_app_id}}/audio{{room_id}}_{{user_id}}_{{timestamp}}.mp3（音频格式）
```

> ⚠ 注意：
>
> - **混流的 user_id 统一为 mixer。**

- 直播内容安全审核过程中，如果出现直播流中断，或者持续拉不到数据流，会进行拉流重试，在一段时间内（不同的错误码重试逻辑会有差异）拉取不到数据流，会认为该直播间已经关闭，此时会发送审核结束回调给用户。用户需要在接收到回调后判断该直播间是否为关闭状态，如果不为关闭状态，需要重新发起审核，以此保证直播间不会漏过）

常见问题

1. TRTC 进行内容安全审核传入的 user_id 用户是否可以看见?

不可见，内容安全是以观众审核从TRTC拉流，对于用户端是看不到该user_id的。

2. 内容安全拉取 TRTC 的数据流是否需要收费?

内容安全是以观众身份从 TRTC 拉取数据流进行审核的，TRTC 默认计费方式按订阅时长（拉流）计费，所以这里会产生费用，具体的收费请参见 音视频计费时长说明。

> 注意：
>
> - 创建审核任务的 user_id 不能同房间内的任何一个 user_id 相同。
> - TRTC room_id 的限制是 uint，房间号取值区间为1 – 4294967295，由开发者自行维护和分配。

3. 内容安全是以观众审核从 TRTC 拉流，那这个内容审核用户，是按照正常创建用户创建吗?

是的，我们是用这个用户进入房间拉数据的。他也是一个正常的用户。

4. 对 TRTC 的音频、视频做内容审核，分别需要多久才能返回审核结果?

片段会即时返回。音频是 0.2 的实时率，图片会在拿到数据之后 1s 内返回。

5. 什么是 user_sig?

具体请参见 user_sig 相关。

6. TRTC 怎么校验生成的 user_sig 是否正确? 进房报错 -3319、-3320 错误怎么排查?

可登录实时音视频控制台，选择 **开发辅助** > **user_sig 生成&校验** 校验 user_sig。

7. 多人语聊房中，多路流，怎么判断谁违规了吗?

目前只能通过我们返回的回调上的地址去区分，我们回调的片段会存储在 cos 上面，文件名是 trtc_[room_id]_[user_id]_timestamp 的格式。

8. 对于拼装的 TRTC 审核 URL 有什么需要注意的点?

由于 user_sig 中包含有特殊符号，拼装成 url 前要先进行 escape 才能放到 URL 中。

> 说明：
> 更多问题，请参见 TRTC 常见问题文档。

附录

- **音频审核服务**提供以下四个审核接口，通过单击链接查看详细接口文档：
  - 创建审核任务
  - 结束审核

- 查询审核任务列表
- 查询审核任务详情

- **视频审核服务**提供以下四个审核接口，通过单击链接查看详细接口文档：
  - 创建审核任务
  - 结束审核
  - 查询审核任务列表
  - 查询审核任务详情

导入在线媒体流

最近更新时间：2022-06-02 10:49:48

应用场景

AI 互动课堂

通过录播真人教学视频结合 AI 技术进行线上直播互动教学。

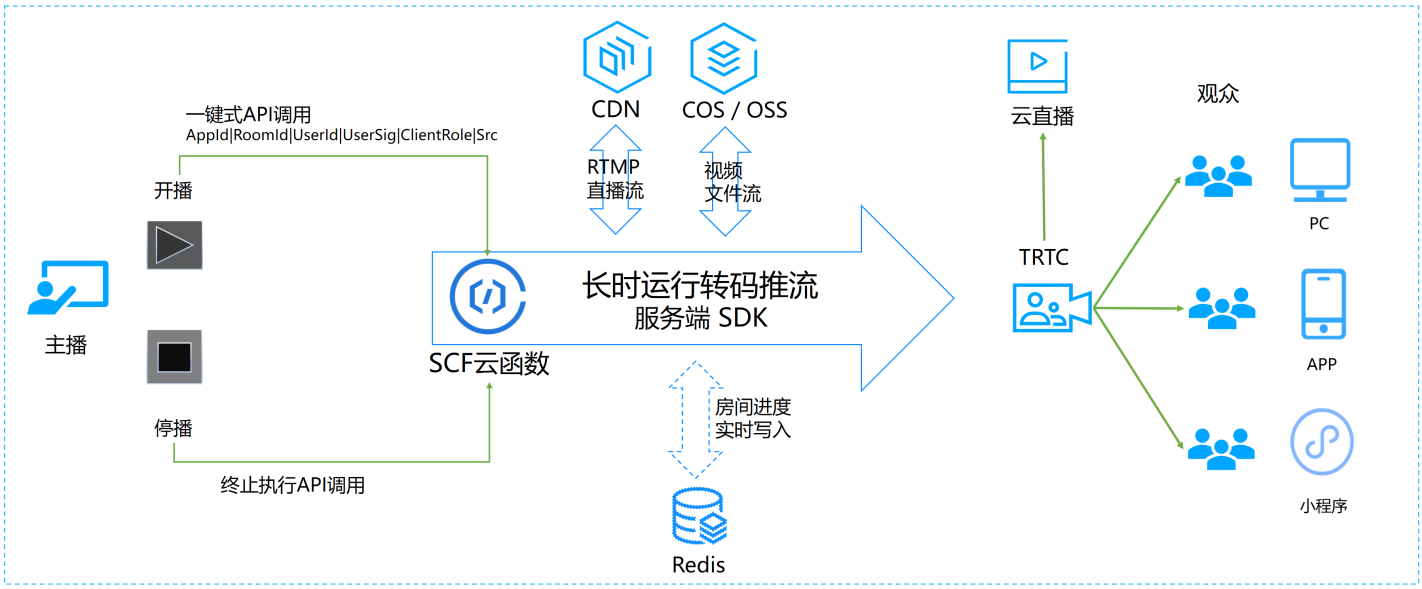
- 上课前，根据教师的课程设置，将知识点讲解、互动提问、问题反馈和解答等信息录制成视频片段，上传到视频库。
- 课堂中，通过云函数将已有的录播视频推送到 TRTC 房间进行直播。学生通过语音、触屏实现互动式学习。服务端通过 AI 技术，智能识别学生的实时语音和作答，并根据学生的表现，无缝切换教学片段，实时给予不同的反馈，从而提供个性化的教学体验。

“一起看” 房间服务

- 游戏直播、秀场、体育赛事等直播类内容，可以通过云函数将 RTMP 直播流推送到 TRTC 房间，实时交流，带动热点。
- 电影、音乐等点播类节目，可以通过云函数将媒体文件转换为在线媒体流输入至 TRTC 房间，增值服务，打造社区圈层。
- 云函数一键触发、免运维、弹性伸缩等特性可以快速支撑实时互动娱乐社交应用的构建。
- 云函数的可编程性，可以快速整合其他云服务及三方服务，扩展业务边界，高效创新玩法。

操作场景

本文为您介绍如何 使用 API 网关调用云函数，将已有的录播视频或者 RTMP 直播流推送到实时音视频 TRTC 房间进行直播。如您需开启推流直播的实时记录，可以选择使用 Redis，API 网关会将进度实时写入 Redis。工作流程如下图所示：



API 网关调用涉及的参数如下：

参数名称	类型	必选	描述
VideoSrc	String	是	被推流的视频源，例如 https://test-123456789.cos.ap-shanghai.myqcloud.com/video/1.mp4。
SdkAppId	Int	是	应用 ID，用于区分不同 TRTC 应用。

参数名称	类型	必选	描述
RoomId	Int	否	整型房间号 ID，用于在一个 TRTC 应用中唯一标识一个房间。
StrRoomId	String	否	字符串房间号 ID，RoomId 与 StrRoomId 必须配置一项，如果 RoomId 与 StrRoomId 同时配置，则使用 RoomId。
Mode	String	否	• vod：点播模式，即推流为某个录制好的文件，默认模式。 • live：直播模式，即推流为 rtmp 直播源。
UserId	String	是	推流用户 ID，用于在一个 TRTC 应用中唯一标识一个用户。
UserSig	String	是	推流用户签名，用于对一个用户进行登录鉴权认证。
Redis	Boolean	否	是否使用 Redis，默认为 false。
RedisHost	String	否	Redis 为 true 时，redis 的 host 地址。
RedisPort	Integer	否	Redis 为 true 时，redis 的访问端口号。
RedisPassword	String	否	Redis 为 true 时，redis 的访问密码。

说明：

- 如果 redis 值为 false，从 videoSrc 视频源拉流进行直播推流，直播流将从最新开始。
- 如果 redis 值为 true，对于同一个 videoSrc 视频源，API 网关将先在 redis 中查询是否有上一次直播流推流记录：
  - 若存在记录，则恢复上一次推流。
  - 若无记录，则重新开始推流。直播推流进度通过回调实时写入 redis。

操作步骤

创建云函数

1. 登录云函数控制台，选择左侧导航栏中的【函数服务】。

2. 在“函数服务”页面上方选择**北京**地域，并单击【新建】进入新建函数页面，根据页面相关信息提示进行配置。如下图所示：

- **创建方式**：选择【模板创建】。
- **模糊搜索**：输入“TRTC直播推流”，并进行搜索。
  单击模板中的【查看详情】，即可在弹出的“模板详情”窗口中查看相关信息，支持下载操作。

3. 单击【下一步】，根据页面相关信息提示进行配置。如下图所示：

- **函数名称**：默认填充。
- **异步执行**：勾选以开启。开启后，函数将以异步执行模式响应事件，事件调用无需阻塞等待处理结果，事件将在被调用后进入异步执行状态。
- **状态追踪**：勾选以开启。开启后，针对异步执行的事件，将开始记录响应事件的实时状态，并提供事件的统计、查询及终止服务，产生的事件状态数据将为您保留3天。
- **执行超时时间**：可根据需要自行修改。

4. 配置 API 网关触发器，默认新建 API 服务，不开启集成响应。您也可以选择自定义创建，自定义创建时确保集成响应关闭。如下图所示：

5. 单击【完成】即可完成函数创建和 API 网关触发器创建。
6. 如需使用 Redis 实时记录推流进度，由于 Redis 只能私有网络访问，因此必须将云函数的 VPC 配置在与 Redis 在同一个私有网络下。如下图所示：

创建 TRTC 应用

1. 登录实时音视频控制台，选择左侧导航栏中的【开发辅助】>【快速跑通 Demo】。

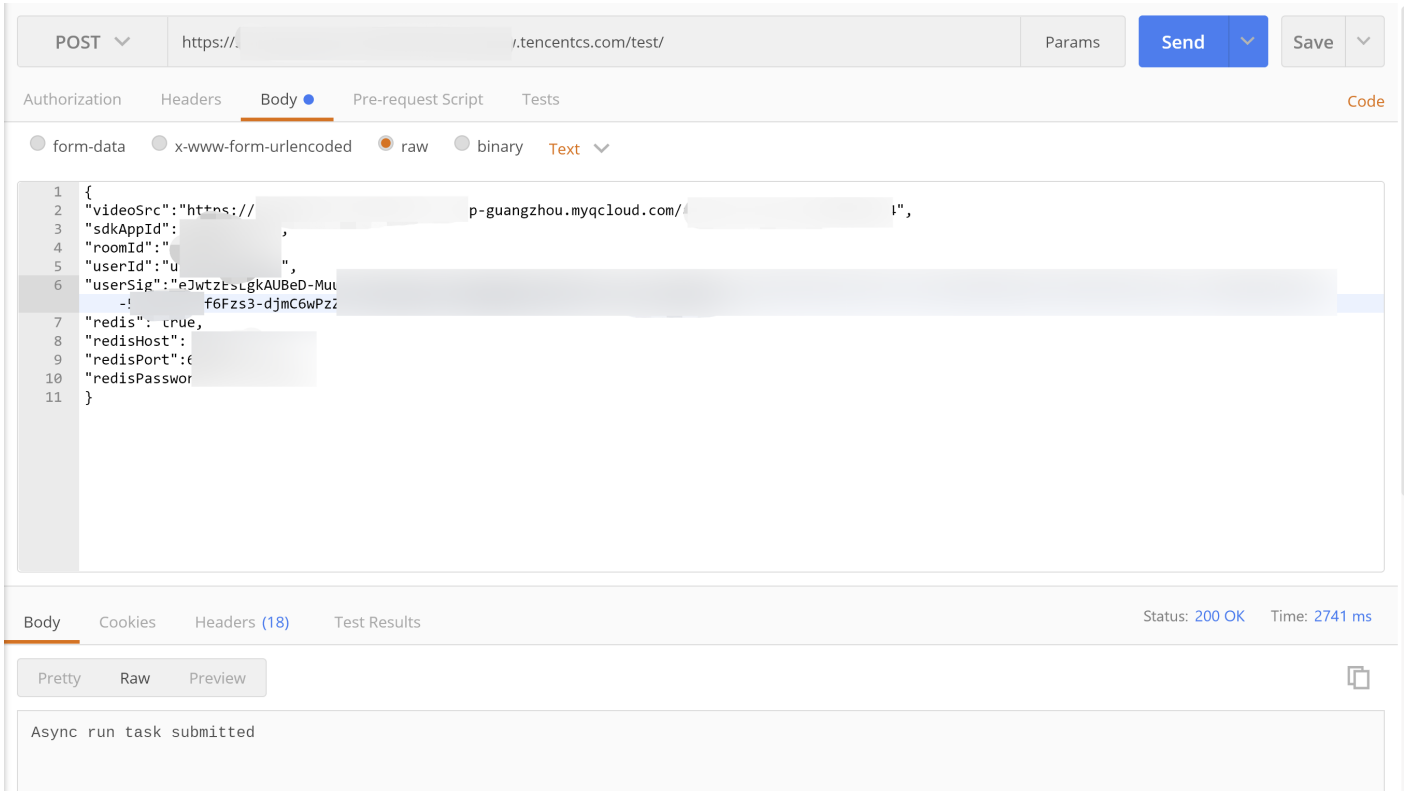
2. 填写 Demo 名称，单击【创建】完成应用创建。您可以根据自己的客户端选择模板试运行。

测试函数功能

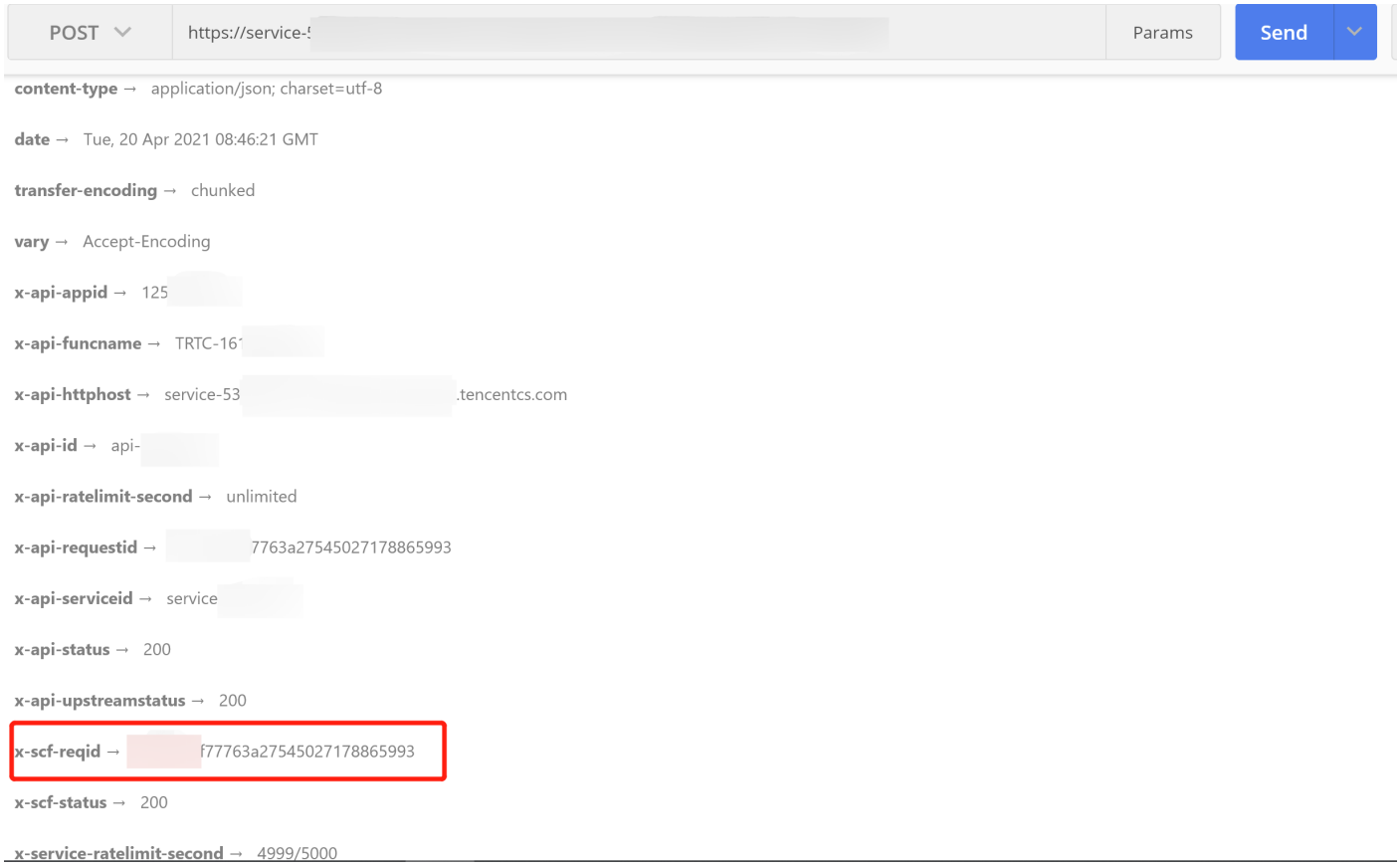
1. 使用 Postman 构造 HTTP 请求。示例如下：

```
{
"SdkAppId": 1400000000,
"StrRoomId": "98915abc",
"UserId": "user_55952144",
"Mode": "vod",
"UserSig": "eJwtzN0KgkAQBeB32dtCZqcd-6CbMMRYIrGiu6jcZAhL1NogevdMvxxxxxxxxxxxxxxxxxxAIdENM*c27uLV*552dj6iNRQCiVGgdNfjtVFecilApABSg9OTTmXXFtOiciBIBBWy7xxxxxxxxRHbbjo-kFrN0UvOfuRH1tMyiDeLGcPu7ocxxxxxxxxxx5uL7A0DEMb8_",
"Redis": false,
"VideoSrc": "https://test-123456789.cos.ap-shanghai.myqcloud.com/video/1.mp4"
}
```

如下图所示：

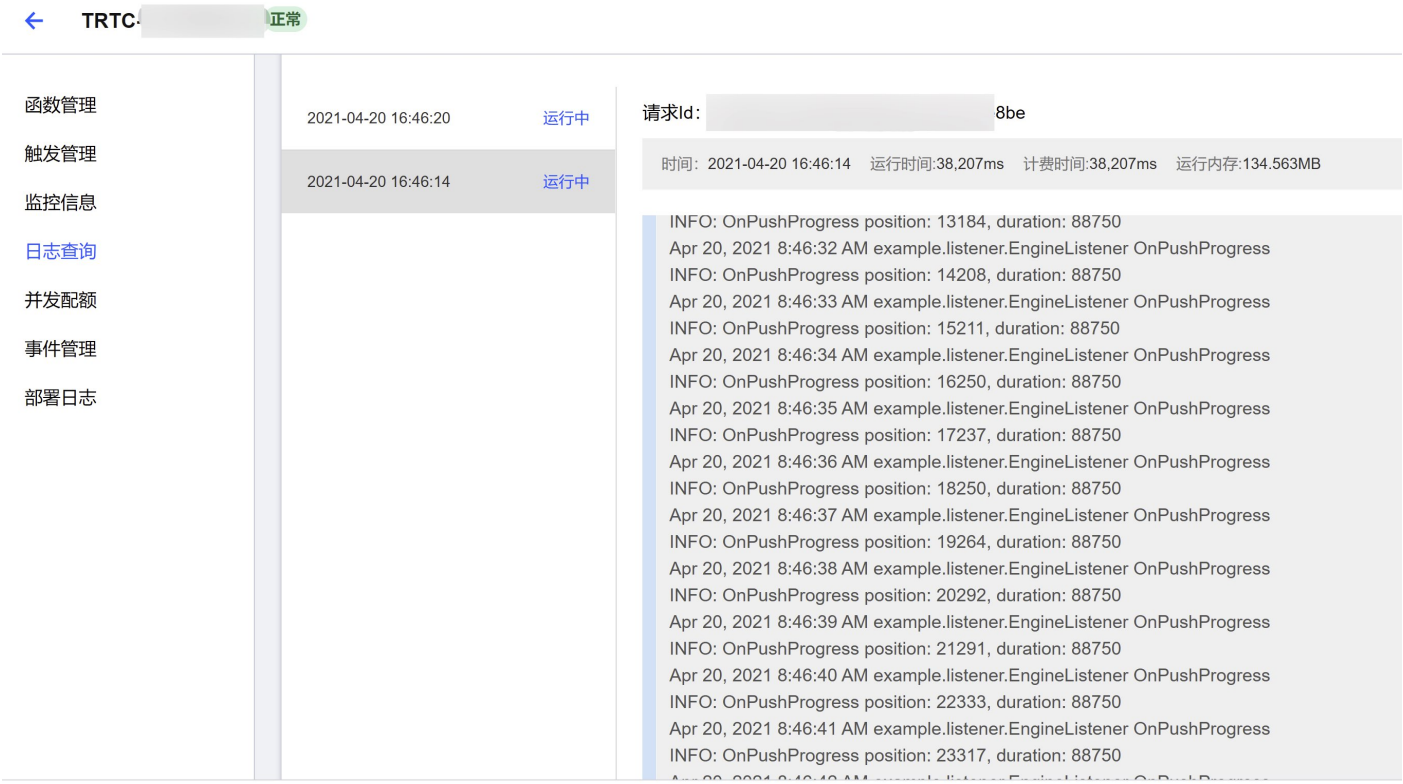


2. 请求发送后会收到异步函数响应 “Async run task submitted”，此次函数的 RequstId 会通过 HTTP 头部信息中的 x-scf-reqid 返回。如下图所示：

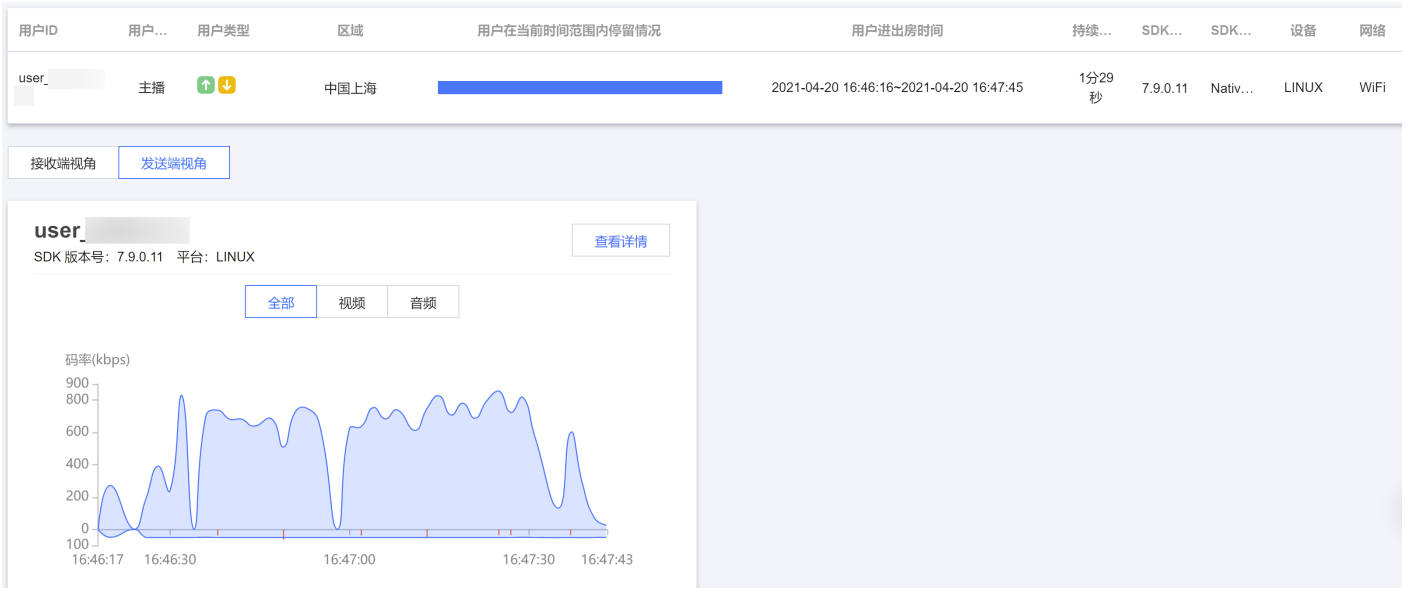


3. 在云函数控制台【函数服务】页面中，单击上述 创建云函数 步骤中创建的云函数名称，进入“函数详情”页面。

4. 在“函数详情”页面中选择【日志查询】页签，可以查看到打印出的推流日志信息。如下图所示：



5. 切换至 实时音视频控制台，在“监控仪表盘”页面单击房间 ID，查看推流监控详情信息。如下图所示：



6. 如需在推流过程中停止推流，可以调用 终止异步函数接口 InvokeRequestId 参数停止推流（**必须开启状态追踪**）。
其中 InvokeRequestId 可从上述步骤2的响应头部信息 x-scf-reqid 中获取。

资源访问管理
访问管理综述

最近更新时间：2021-03-30 16:23:20

> ⚠ 注意：
> 本文档主要介绍 **实时音视频 TRTC** 访问管理功能的相关内容，其他产品访问管理相关内容请参见 支持 CAM 的产品。

访问管理（Cloud Access Management，**CAM**）是腾讯云提供的一套 Web 服务，它主要用于帮助客户安全管理腾讯云账户下的资源的访问权限。通过 CAM，您可以创建、管理和销毁用户（组），并通过身份管理和策略管理控制哪些人可以使用哪些腾讯云资源。

实时音视频 TRTC 已接入 **CAM**，开发者可以根据自身需要为子账号分配合适的 TRTC 访问权限。

基础入门

在使用 TRTC 访问管理前，您需要对 CAM 和 TRTC 的基本概念有所了解，涉及的概念主要有：

- CAM 相关：用户、策略
- TRTC 相关：应用、SDKAppID

适用场景

腾讯云产品维度权限隔离

某企业内有多个部门在使用腾讯云，其中 A 部门只负责对接 TRTC。A 部门的人员需要有访问 TRTC 的权限，但不能有访问其他腾讯云产品的权限。该企业可以通过主账号为 A 部门创建一个子账号，只授予该子账号 TRTC 相关权限，然后将该子账号提供给 A 部门使用。

TRTC 应用维度权限隔离

某企业内有多个业务在使用 TRTC，相互之间需要进行隔离。隔离包括资源隔离和权限隔离两个方面，前者由TRTC 应用体系提供，后者则由 TRTC 访问管理来实现。该企业可以为每个业务创建一个子账号，授予相关的 TRTC 应用权限，使得每个业务只能访问和自己相关的应用。

TRTC 操作维度权限隔离

某企业的一个业务在使用 TRTC，该业务的产品运营人员需要访问 TRTC 控制台，获取用量统计信息，同时不允许其进行敏感操作（如修改旁路推流、云端录制配置等），以免误操作影响业务。这时可以先创建自定义策略，该策略拥有 TRTC 控制台登录、用量统计相关 API 的访问权限，然后创建一个子账号，与上述策略绑定，将该子账号提供给产品运营人员。

授权粒度

访问管理的核心功能可以表达为：**允许或禁止某账号对某些资源进行某些操作**。TRTC 访问管理支持 资源级授权，资源的粒度是 TRTC 应用，操作的粒度是 云 API，包括 服务端 API 以及访问 TRTC 控制台时可能会用到的 API。详细说明请参见 可授权的资源及操作。

能力限制

- TRTC 访问管理的资源粒度为 应用，不支持对更细粒度的资源（如应用信息、配置信息等）做授权。
- TRTC 访问管理不支持 项目，建议您通过 标签 来管理云服务资源。

可授权的资源及操作

最近更新时间：2022-03-16 17:00:06

> ⚠ 注意：
> 本文档主要介绍 **实时音视频 TRTC** 访问管理功能的相关内容，其他产品访问管理相关内容请参见 支持 CAM 的产品。

访问管理的核心功能可以表达为：**允许或禁止某账号对某些资源进行某些操作**。TRTC 访问管理支持 资源级授权，资源的粒度是 TRTC 应用，操作的粒度是 云 API，包括 服务端 API 以及访问 TRTC 控制台时可能会用到的 API。

如有 TRTC 访问管理需求，请登录腾讯云 主账号 使用 预设策略 或 自定义策略 完成具体授权操作。

可授权的资源类型

TRTC 访问管理可授权的资源类型为 应用。

支持资源级授权的 API

除了部分 不支持资源级授权的 API ，本小节列出的所有 API 操作均支持资源级授权。 授权策略语法 中对这些 API 操作的**资源语法描述**均相同，具体为：

- **授权所有应用访问权限：**qcs::trtc::uin/${uin}:sdkappid/*。
- **授权单个应用访问权限：**qcs::trtc::uin/${uin}:sdkappid/${SdkAppId}。

服务端 API 操作

接口名称	接口分类	功能描述
DismissRoom	房间管理	解散房间
RemoveUser	房间管理	移出用户
RemoveUserByStrRoomId	房间管理	移出用户（字符串房间号）
DismissRoomByStrRoomId	房间管理	解散房间（字符串房间号）
StartMCUMixTranscode	混流转码	启动云端混流
StopMCUMixTranscode	混流转码	结束云端混流转码
StartMCUMixTranscodeByStrRoomId	混流转码	启动云端混流（字符串房间号）
StopMCUMixTranscodeByStrRoomId	混流转码	结束云端混流（字符串房间号）
CreateTroubleInfo	通话质量监控	创建异常信息
DescribeAbnormalEvent	通话质量监控	查询异常体验事件
DescribeCallDetail	通话质量监控	查询用户列表与通话指标
DescribeHistoryScale	通话质量监控	查询历史房间和用户数

接口名称	接口分类	功能描述
DescribeRoomInformation	通话质量监控	查询房间列表
DescribeUserInformation	通话质量监控	查询历史用户列表

控制台 API 操作

接口名称	使用模块	功能描述
DescribeAppStatList	TRTC 控制台 • 概览 • 用量统计 • 监控仪表盘 • 开发辅助 > UserSig 生成&校验 • 应用管理	获取应用列表
DescribeSdkAppInfo	TRTC 控制台 应用管理 > 应用信息	获取应用信息
ModifyAppInfo	TRTC 控制台 应用管理 > 应用信息	编辑应用信息
ChangeSecretKeyFlag	TRTC 控制台 应用管理 > 应用信息	修改权限密钥状态
CreateWatermark	TRTC 控制台 应用管理 > 素材管理	上传图片
DeleteWatermark	TRTC 控制台 应用管理 > 素材管理	删除图片
ModifyWatermark	TRTC 控制台 应用管理 > 素材管理	编辑图片
DescribeWatermark	TRTC 控制台 应用管理 > 素材管理	查找图片
CreateSecret	TRTC 控制台 应用管理 > 快速上手	创建对称式加密密钥
ToggleSecretVersion	TRTC 控制台应用管理 > 快速上手	切换密钥版本（公私钥/对称式加密密钥）
DescribeSecret	TRTC 控制台 • 开发辅助 > 快速跑通 Demo • 开发辅助 > UserSig 生成&校验 • 应用管理 > 快速上手	获取对称式加密密钥
DescribeTrtcAppAndAccountInfo	TRTC 控制台 开发辅助 > UserSig 生成&校验	获取应用及账号信息来获取公私钥
CreateSecretUserSig	TRTC 控制台 开发辅助 > UserSig 生成&校验	使用对称式加密密钥生成 UserSig
DescribeSig	TRTC 控制台 • 开发辅助 > UserSig 生成&校验 • 应用管理 > 快速上手	获取使用旧版公私钥生成的UserSig
VerifySecretUserSig	TRTC 控制台 开发辅助 > UserSig 生成&校验	对称式加密密钥生成的 UserSig 校验
VerifySig	TRTC 控制台 开发辅助 > UserSig 生成&校验	公私钥生成的 UserSig 校验

接口名称	使用模块	功能描述
CreateSpearConf	TRTC 控制台 应用管理 > 画面设定	新增画面设定配置。此功能设置卡片仅对 iLiveSDK 1.9.6 及之前的版本可见，TRTC SDK 6.0及以后版本请参见 设定画面质量
DeleteSpearConf	TRTC 控制台 应用管理 > 画面设定	删除画面设定配置。此功能设置卡片仅对 iLiveSDK 1.9.6 及之前的版本可见，TRTC SDK 6.0及以后版本请参见 设定画面质量
ModifySpearConf	TRTC 控制台 应用管理 > 画面设定	修改画面设定配置。此功能设置卡片仅对 iLiveSDK 1.9.6 及之前的版本可见，TRTC SDK 6.0及以后版本请参见 设定画面质量
DescribeSpearConf	TRTC 控制台 应用管理 > 画面设定	获取画面设定配置。此功能设置卡片仅对 iLiveSDK 1.9.6 及之前的版本可见，TRTC SDK 6.0及以后版本请参见 设定画面质量
ToggleSpearScheme	TRTC 控制台 应用管理 > 画面设定	切换画面设定场景。此功能设置卡片仅对 iLiveSDK 1.9.6 及之前的版本可见，TRTC SDK 6.0及以后版本请参见 设定画面质量

不支持资源级授权的 API

由于特殊限制，下述 API 不支持资源级授权：

服务端 API 操作

接口名称	接口分类	功能描述	特殊限制说明
DescribeDetailEvent	通话质量监控	获取详细事件	输入参数无 SDKAppID ，无法进行资源级授权。
DescribeRecordStatistic	其他接口	查询云端录制计费时长	业务原因，暂不支持资源级授权
DescribeTrtcInteractiveTime	其他接口	查询音视频互动计费时长	业务原因，暂不支持资源级授权
DescribeTrtcMcuTranscodeTime	其他接口	查询旁路转码计费时长	业务原因，暂不支持资源级授权

控制台 API 操作

接口名称	使用模块	功能描述	特殊限制说明
DescribeTrtcStatistic	TRTC 控制台 • 概览 • 用量统计	获取计费时长用量统计数据	该接口包含返回全量 SDKAppID 的统计数据，限制非全量 SDKAppID 将返回错误。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
DescribeDurationPackages	TRTC 控制台 • 概览 • 套餐包管理	获取预付费套餐包列表	预付费套餐包为单个腾讯云账号下的所有 TRTC 应用共享，套餐包信息中无 SDKAppID 参数，无法进行资源级授权

接口名称	使用模块	功能描述	特殊限制说明
GetUserList	TRTC 控制台 监控仪表盘	获取用户列表	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
GetUserInfo	TRTC 控制台 监控仪表盘	获取用户信息	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
GetCommState	TRTC 控制台 监控仪表盘	获取通话状态	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
GetElasticSearchData	TRTC 控制台 监控仪表盘	查询 ES 数据	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
CreateTrtcApp	TRTC 控制台 • 开发辅助 > 快速跑通 Demo • 应用管理	创建 TRTC 应用	输入参数无 SDKAppID，无法进行资源级授权。SDKAppID 是 TRTC 应用的唯一标识，创建应用之后才有 SDKAppID 信息
HardDescribeMixConf	TRTC 控制台 应用管理 > 功能配置	查询自动旁路推流状态	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
ModifyMixConf	TRTC 控制台 应用管理 > 功能配置	开启/关闭自动旁路推流	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
RemindBalance	TRTC 控制台 套餐包管理	获取预付费套餐包余额告警信息	预付费套餐包为单个腾讯云账号下的所有 TRTC 应用共享，套餐包信息中无 SDKAppID 参数，无法进行资源级授权

注意：

针对不支持资源级授权的 API 操作，您仍然可以通过 自定义策略 向用户授予使用该操作的权限，但是策略语句的资源元素必须指定为 * 。

预设策略

最近更新时间：2021-07-26 11:20:15

> ⚠ 注意：
> 本文档主要介绍 **实时音视频 TRTC** 访问管理功能的相关内容，其他产品访问管理相关内容请参见 支持 CAM 的产品。

TRTC 访问管理实质上是将子账号与策略进行绑定，或者说将策略授予子账号。开发者可以在控制台上直接使用预设策略来实现一些简单的授权操作，复杂的授权操作请参见 自定义策略。

TRTC 目前提供了以下预设策略：

策略名称	策略描述
QcloudTRTCFullAccess	TRTC 全读写访问权限
QcloudTRTCReadonlyAccess	TRTC 只读访问权限

预设策略使用示例

新建拥有 TRTC 全读写访问权限的子帐号

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【 用户列表 】，单击【 新建用户 】。
2. 在“新建用户”页面选择【 自定义创建 】，进入“新建子用户”页面。

> ⓘ 说明：
> 请根据 CAM 自定义创建子用户 的操作指引完成“设置用户权限”之前的步骤。

3. 在“设置用户权限”页面：
   i. 搜索并勾选预设策略 QcloudTRTCFullAccess。
   ii. 单击【 下一步 】。
4. 在“审阅信息和权限”分栏下单击【 完成 】，完成子用户的创建，在成功页面下载并保管好该子用户的登录链接和安全凭证，其中包含的信息如下表：

信息	来源	作用	是否必须保存
登录链接	在页面中复制	方便登录控制台，省略填写主账号的步骤	否
用户名	安全凭证 CSV 文件	登录控制台时填写	是
密码	安全凭证 CSV 文件	登录控制台时填写	是
SecretId	安全凭证 CSV 文件	调用服务端 API 时使用，详见 访问密钥	是
SecretKey	安全凭证 CSV 文件	调用服务端 API 时使用，详见 访问密钥	是

5. 将上述登录链接和安全凭证提供给被授权方，后者即可使用该子用户对 TRTC 做所有操作，包括访问 TRTC 控制台、请求 TRTC 服务端 API 等。

将 TRTC 全读写访问权限授予已存在的子帐号

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【 用户列表 】，单击想要进行授权的子账号。
2. 单击“用户详情”页面权限栏的【 添加策略 】，如果子账号的权限非空，则单击【 关联策略 】。
3. 选择【 从策略列表中选取策略关联 】，搜索并勾选预设策略 QcloudTRTCFullAccess。后续按页面提示完成授权流程即可。

解除子帐号的 TRTC 全读写访问权限

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【 用户列表 】，单击想要解除授权的子账号。
2. 在“用户详情”页面权限栏找到预设策略 QcloudTRTCFullAccess，单击右侧的【 解除 】。按页面提示完成解除授权流程即可。

自定义策略

最近更新时间：2021-07-26 11:21:08

> **注意：**
> 本文档主要介绍 **实时音视频 TRTC** 访问管理功能的相关内容，其他产品访问管理相关内容请参见 支持 CAM 的产品。

在 TRTC 访问管理中使用 预设策略 来实现授权虽然方便，但权限控制粒度较粗，不能细化到 TRTC 应用 和 云 API 粒度。如果开发者要求精细的权限控制能力，则需要创建自定义策略。

自定义策略创建方法

自定义策略有多种创建方法，下方表格展示各种方法的对比，具体操作流程请参考下文。

创建入口	创建方法	效力（Effect）	资源（Resource）	操作（Action）	灵活性	难度
CAM 控制台	策略生成器	手动选择	语法描述	手动选择	中	中
CAM 控制台	策略语法	语法描述	语法描述	语法描述	高	高
CAM 服务端 API	CreatePolicy	语法描述	语法描述	语法描述	高	高

> **说明：**
> - TRTC **不支持**按产品功能或项目来创建自定义策略。
> - **手动选择**指用户在控制台所展示的候选项列表中选择对象。
> - **语法描述**指通过 授权策略语法 来描述对象。

授权策略语法

资源语法描述

如上文所述，TRTC 权限管理的资源粒度是应用。应用的策略语法描述方式遵循 CAM 资源描述方式。在下文的示例中，开发者的主账号 ID 是 12345678，开发者创建了三个应用：SDKAppID 分别是 1400000000，1400000001 和 1400000002。

- 实时音视频所有应用的策略语法描述

```
"resource": [
"qcs::trtc::uin/12345678:sdkappid/*"
]
```

- 单个应用的策略语法描述

```
"resource": [
"qcs::trtc::uin/12345678:sdkappid/1400000001"
]
```

- 多个应用的策略语法描述

```
"resource": [
"qcs::trtc::uin/12345678:sdkappid/1400000000",
"qcs::trtc::uin/12345678:sdkappid/1400000001"
]
```

操作语法描述

如上文所述，实时音视频权限管理的操作粒度是云 API，详情请参见 可授权的资源及操作。在下文的示例中，以 DescribeAppStatList（获取应用列表）、DescribeSdkAppInfo（获取应用信息）等云 API 为例。

- 实时音视频所有云 API 的策略语法描述

```
"action": [
"name/trtc:*"
]
```

- 单个云 API 操作的策略语法描述

```
"action": [
"name/trtc:DescribeAppStatList"
]
```

- 多个云 API 操作的策略语法描述

```
"action": [
"name/trtc:DescribeAppStatList",
"name/trtc:DescribeTrtcAppAndAccountInfo"
]
```

自定义策略使用示例

使用策略生成器

在下文示例中，我们将创建一个自定义策略。该策略允许对1400000001这个实时音视频应用进行任何操作，除了 RemoveUser 这个服务端 API。

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【策略】，单击【新建自定义策略】。
2. 选择【按策略生成器创建】，进入策略创建页面。
3. 选择服务和操作。
   - 【效果(Effect)】配置项选择【允许】。
   - 【服务(Service)】配置项选择【实时音视频】。
   - 【操作(Action)】配置项勾选所有项。
   - 【资源(Resource)】配置项按照 资源语法描述 说明填写 qcs::trtc::uin/12345678:sdkappid/1400000001。
   - 【条件(Condition)】配置项无需配置。
   - 单击【添加声明】，页面最下方会出现一条“允许对实时音视频应用1400000001进行任何操作”的声明。

4. 在同个页面中继续添加另一条声明。
   - 【效果(Effect)】配置项选择【拒绝】。
   - 【服务(Service)】配置项选择【实时音视频】。
   - 【操作(Action)】配置项勾选RemoveUser（可通过搜索功能快速查找）。
   - 【资源(Resource)】配置项按照 资源语法描述 说明填写 qcs::trtc::uin/12345678:sdkappid/1400000001。
   - 【条件(Condition)】配置项无需配置。
   - 单击【添加声明】，页面最下方会出现一条“拒绝对实时音视频应用1400000001进行 RemoveUser 操作”的声明。
5. 单击【下一步】，按需修改策略名称（也可以不修改）。
6. 单击【完成】完成自定义策略的创建。

后续将该策略授予其他子帐号的方法同 将 TRTC 全读写访问权限授予已存在的子帐号。

使用策略语法

在下文示例中，我们将创建一个自定义策略。该策略允许对1400000001和1400000002这两个实时音视频应用进行任何操作，但不允许对1400000001进行 RemoveUser 操作。

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【策略】，单击【新建自定义策略】。
2. 选择【按策略语法创建】，进入策略创建页面。
3. 在【选择模板类型】框下选择【空白模板】。

> 说明：
> 策略模板，指新策略是现有策略（预置策略或自定义策略）的一个拷贝，然后在此基础上做调整。在实际使用中，开发者可以根据情况选择合适的策略模板，降低编写策略内容的难度和工作量。

4. 单击【下一步】，按需修改策略名称（也可以不修改）。
5. 在【编辑策略内容】编辑框中填写策略内容。本示例的策略内容为：

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": [
"name/trtc:*"
],
"resource": [
"qcs::trtc::uin/12345678:sdkappid/1400000001",
"qcs::trtc::uin/12345678:sdkappid/1400000002"
]
},
{
"effect": "deny",
"action": [
"name/trtc:RemoveUser"
],
"resource": [
```

```
"qcs::trtc::uin/12345678:sdkappid/1400000001"
]
}
]
}
```

> **说明：**
> 策略内容需遵循 CAM 策略语法逻辑，其中资源和操作两个元素的语法请参见上文 资源语法描述 和 操作语法描述 所述。

6. 单击【创建策略】完成自定义策略的创建。

后续将该策略授予其他子帐号的方法同 将 TRTC 全读写访问权限授予已存在的子帐号。

使用 CAM 提供的服务端 API

对于大多数开发者来说，在控制台完成权限管理操作已经能满足业务需求。但如果需要将权限管理能力自动化和系统化，则可以基于服务端 API 来实现。
策略相关的服务端 API 属于 CAM，具体请参见 CAM 官网文档。此处仅列出几个主要接口：

- 创建策略
- 删除策略
- 绑定策略到用户
- 解除绑定到用户的策略