实时音视频

高级功能

腾讯云

文档目录

- 视频截图上传
- 高级权限控制
- 测试硬件设备
  - Android&iOS&Windows&Mac
  - Web
- 测试网络质量
  - Android&iOS&Windows&Mac&Flutter
  - Web

高级功能
服务端
发布音视频流到直播 CDN

最近更新时间：2025-07-03 15:26:12

本文将介绍如何将 TRTC 房间中的音视频流发布（也称作“转推”）到直播 CDN 上，用于兼容常规的直播播放器进行观看。

统一术语

为了便于后续理解，先统一术语：
旁路转推：单路用户流转推到 CDN，不需要转码。
混流转推：多路用户流混合转码成一路流（或者单路用户流转音视频格式），然后转推到 CDN，需要转码。

适用场景

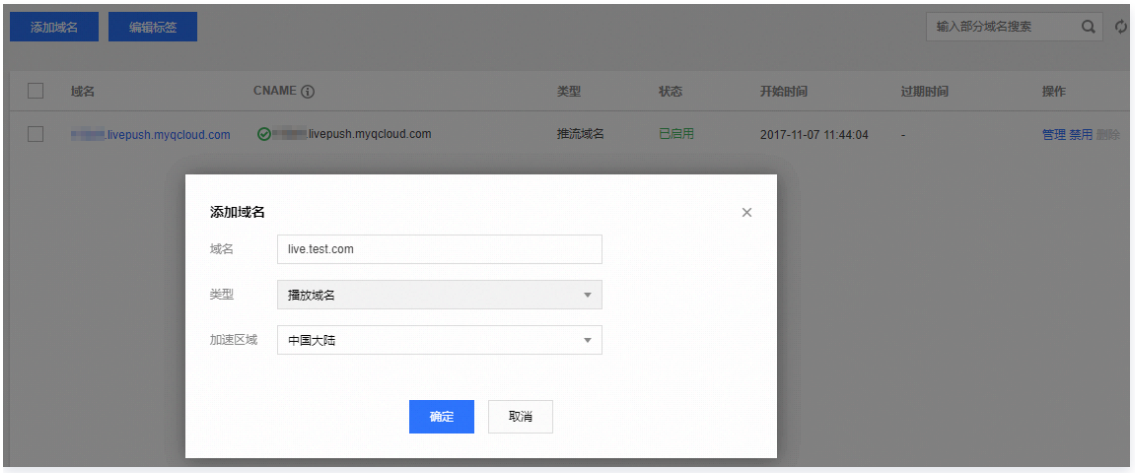
由于 TRTC 采用 UDP 协议进行传输音视频数据，而标准直播 CDN 则采用的 RTMP\HLS\FLV 等协议进行数据传输，所以需要将 TRTC 中的音视频数据**旁路**到直播 CDN 中，才能让观众通过直播 CDN 进行观看。
给 TRTC 对接 CDN 观看，一般被用于解决如下两类问题：

- **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。

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，具体操作请参见 添加自有域名 。

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。
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：**

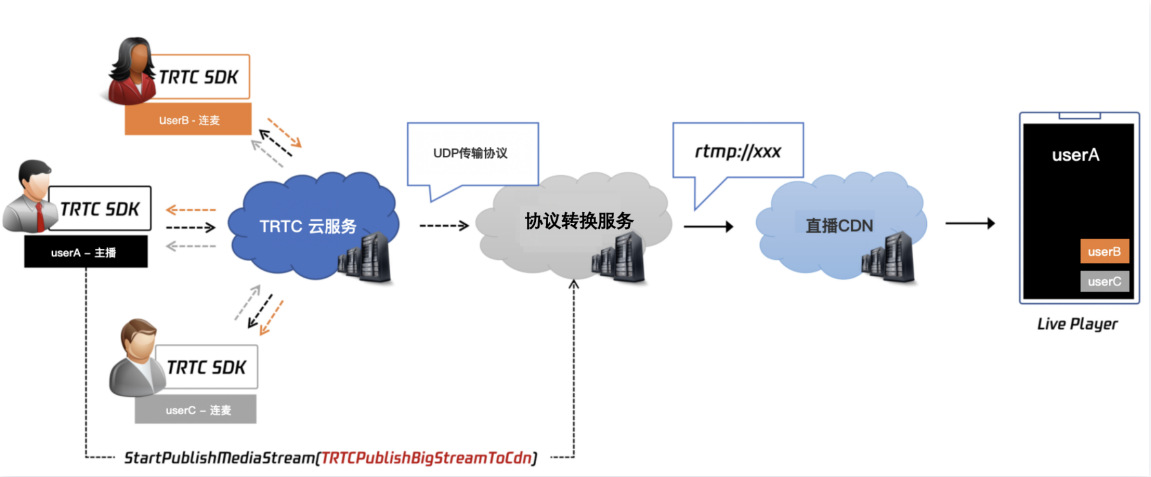
> **不需要添加推流域名**，在 步骤1 中开启旁路直播功能后，腾讯云会默认在您的云直播控制台中增加一个格式为 `xxxxx.livepush.myqcloud.com` 的推流域名，该域名为腾讯云直播服务和 TRTC 服务之间约定的一个默认推流域名。

转推控制方案

TRTC 提供几种发布音视频流到直播 CDN（即“转推”）的控制方案，分别是通过 终端 SDK API发起 、通过 服务端 Restful API 发起 和通过 自动旁路发起 ，几种方案具体说明如下：

方案一：终端 SDK 发起转推

步骤1：发布当前用户的音视频流到直播 CDN



功能介绍

您可以使用 TRTCCloud 提供的接口 startPublishMediaStream（以IOS为例） 将房间中当前用户的音视频流发布到直播 CDN 上（“发布到 CDN” 也被常称为“转推到CDN”）。
在这个过程中，TRTC 的云端服务器不会对音视频数据进行二次的转码加工，而是直接将音视频数据直接导入到直播 CDN 服务器上，因此整个过程所产生的费用是最低的。
但房间中的每一个音视频用户都需要有一条 CDN 上的直播流与之对应，因此如果房间中有多个用户的音视频流，就需要观众端使用多个直播播放器进行观看，且多条 CDN 直播流之间的播放进度可能相差很大。

操作指引

您可以按照如下指引将房间中当前用户的音视频流发布到直播 CDN 上。

1. 创建 TRTCPublishTarget 对象，并指定 TRTCPublishTarget 对象中的 mode 参数为 `TRTCPublishBigStreamToCdn` 或者 `TRTCPublishSubStreamToCdn`，其中前者是指发布当前用户的主路画面（一般是摄像头），后者是指发布当前用户的辅路画面（一般是屏幕分享）。
2. 指定 TRTCPublishTarget 对象中的 cdnUrlList 参数为一个或者多个 CDN 推流地址（标准的 CDN 推流地址一般以 `rtmp://` 作为 URL 的前缀）。如果您指定的 URL是腾讯云的直播 CDN 推流地址（可前往云直播控制台-地址生成器 生成），需要将 isInternalLine 设置为 `true`，否则请将其设置为 `false`。
3. 因为该模式下不涉及转码服务，所以请在调用接口时保持 `TRTCStreamEncoderParam` 和 `TRTCStreamMixingConfig` 两个参数为空。
4. 调用 startPublishMediaStream 接口，并通过 onStartPublishMediaStream 监听本地 API 调用是否成功，如果成功 onStartPublishMediaStream 中的 taskId 会返回一个不为空字符串。
5. 如果您需要停止发布动作，只需要调用 stopPublishMediaStream 并传入之前通过 onStartPublishMediaStream 获得 taskId 即可。

参考代码

如下这段代码的功能是发布当前用户的音视频流到直播CDN：

```java
// 发布当前用户的音视频流到直播CDN
```

```
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishBigStream_ToCdn;
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);
mTRTCCloud.startPublishMediaStream(target, null, null);
```

Objective-C

```
// 发布当前用户的音视频流到直播CDN
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;
[_trtcCloud startPublishMediaStream:target encoderParam:nil mixingConfig:nil];
```

C++

```
// 发布当前用户的音视频流到直播CDN
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl* cdn_url_list = new TRTCPublishCdnUrl[1];
cdn_url_list[0].rtmpUrl = "rtmp://tencent/live/bestnews";
cdn_url_list[0].isInternalLine = true;
target.cdnUrlList = cdn_url_list;
target.cdnUrlListSize = 1;
trtc->startPublishMediaStream(&target, nullptr, nullptr);
delete[] cdn_url_list;
```

Web

```
const options = {
  target: {
    publishMode: PublishMode.PublishMainStreamToCDN
  }
}
try {
  await trtc.startPlugin('CDNStreaming', options);
} catch (error) {
  console.error('CDNStreaming start failed', error);
}
```

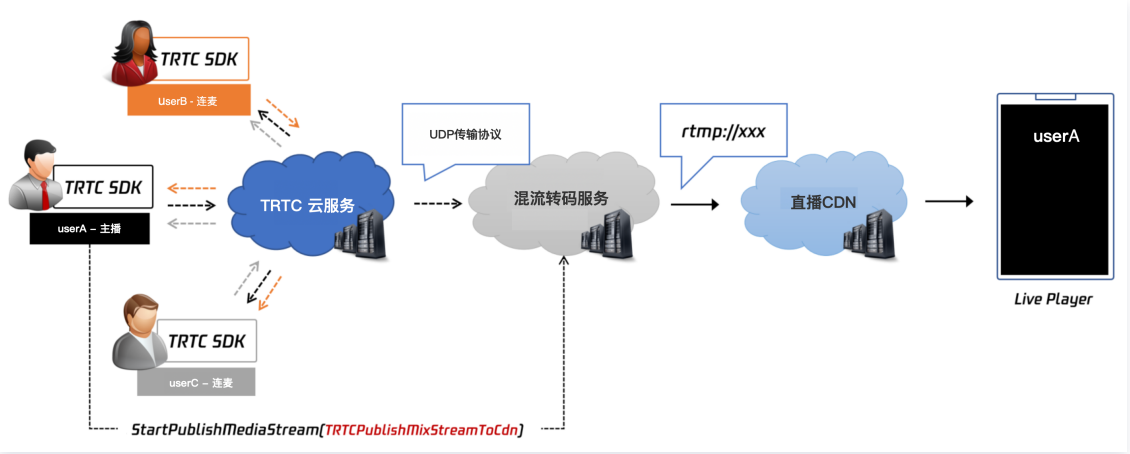
Dart

```
TRTCPublishTarget target = TRTCPublishTarget();
target.mode = TRTCPublishMode.TRTCPublishBigStreamToCdn;
TRTCPublishCdnUrl cdnUrlEntity = new TRTCPublishCdnUrl();
cdnUrlEntity.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrlEntity.isInternalLine = true;
```

```
target.cdnUrlList.add(cdnUrlEntity);

trtcCloud.startPublishMediaStream(target: target);
```

> ⚠ **注意：**
> - Web 端类名稍有差异，其他使用方式一致，详细信息请参见 CDNStreaming Plugin 。
> - Web 4.x 版本转推请参见： Client.startMixTranscode() 。

步骤2：发布混合后的音视频流到直播 CDN



功能介绍

如果您希望将 TRTC 房间中多个用户的音视频流混合成一路，并将混合后的音视频流发布到直播 CDN 上，可以调用 **startPublishMediaStream** 接口并同时指定 `TRTCStreamEncoderParam` 和 `TRTCStreamMixingConfig` 两个参数对混流和转码的细节进行控制。
由于 TRTC 中的多路音视频流需要先在云端进行解码，然后按照您指定的混流参数（ `TRTCStreamMixingConfig` ）进行混合，最终再通过您指定的参数（ `TRTCStreamEncoderParam` ）进行二次编码，最终才能合成一路音视频流并发布到直播 CDN 上，因此该模式下的转推服务需要额外的转码费用 。

操作指引

您可以按照如下指引将房间中多个用户的音视频流混合并发布到直播 CDN 上。

1. 创建 TRTCPublishTarget 对象，并指定 TRTCPublishTarget 中的 mode 参数为 `TRTCPublishMixStreamToCdn` 。
2. 指定 TRTCPublishTarget 对象中的 cdnUrlList 参数为一个或者多个 CDN 推流地址（标准的 CDN 推流地址一般以 `rtmp://` 作为 URL 的前缀）。如果您指定的 url 是腾讯云的直播 CDN 推流地址，需要将 isInternalLine 设置为 true，否则请将其设置为 false。
3. 通过参数 `TRTCStreamEncoderParam` 设置转码后的音视频流的编码参数：
   - **视频编码参数：**需要您指定混合后画面的分辨率、帧率、码率和编码的 GOP 大小，其中 GOP 推荐设置为 3s 即可，FPS 推荐设置为 15，码率和分辨率有一定的映射关系，如下表格列出了几种常用的分辨率以及其对应的推荐码率。

视频宽度	视频高度	视频帧率	视频GOP	视频码率
640	360	15	3	800kbps
960	540	15	3	1200kbps
1280	720	15	3	1500kbps
1920	1080	15	3	2500kbps

   - **音频编码参数：**需要您指定混合后音频的编码格式、编码码率、采样率和声道数。这一步首先需要您先确认一下您在调用 startLocalAudio 时第二个参数 AudioQuality 所指定的音质类型，然后根据您指定的音质类型填写此处的参数。

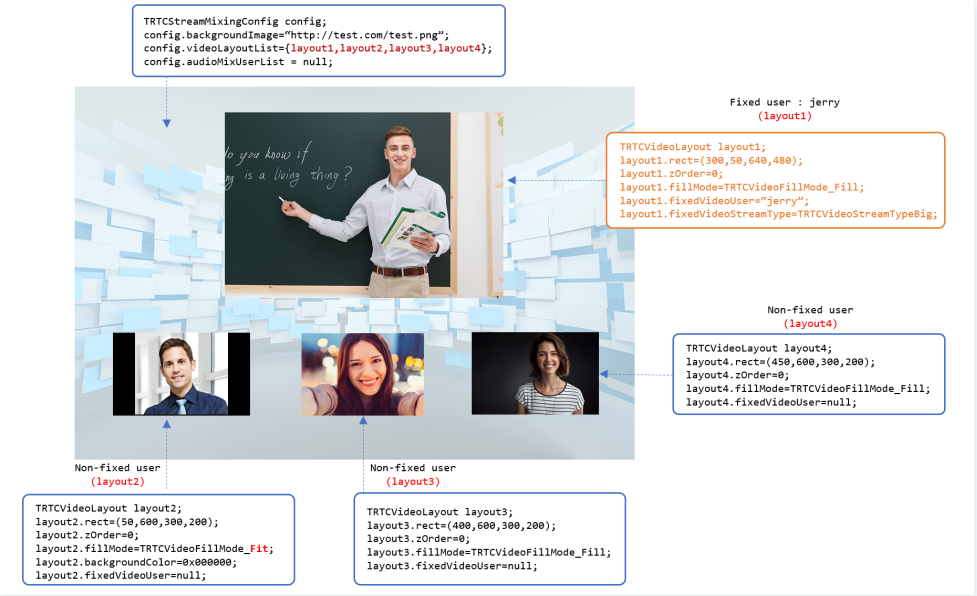
TRTC音频质量类型	音频采样率	音频声道数	音频码率

TRTCAudioQualitySpeech	48000	1	50kbps
TRTCAudioQualityDefault	48000	1	50kbps
TRTCAudioQualityMusic	48000	2	60kbps

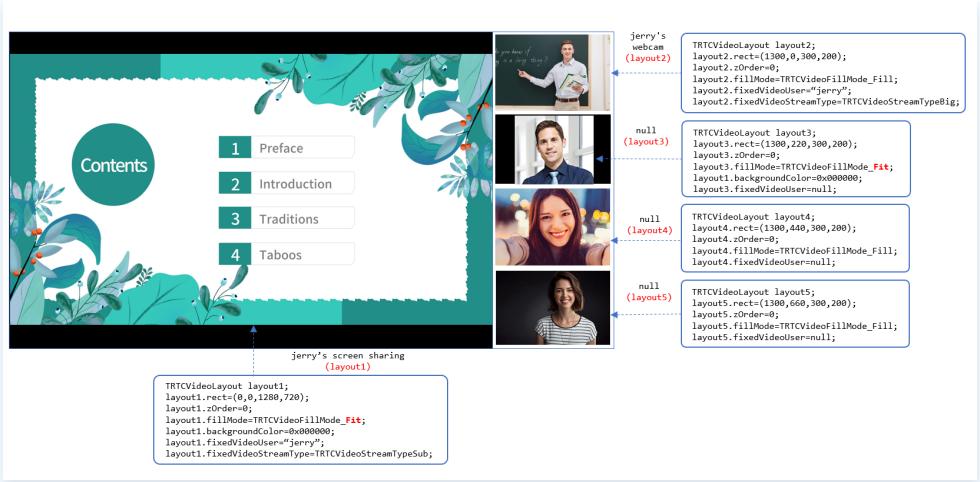
4. 通过参数 `TRTCStreamMixingConfig` 设置音频混流参数和画面排版模式：
   - **音频混流参数（audioMixUserList）**：默认情况下填空值即可，代表会混合房间中的所有音频，如果您只希望混合画面中某几个用户的声音，才需要指定该参数。
   - **画面布局参数（videoLayoutList）**：画面布局是由一个数组所定义的，数组中的每一个 TRTCVideoLayout 对象都代表了一块区域的位置、大小、背景颜色等等。如果您指定了 TRTCVideoLayout 中的 **fixedVideoUser** 字段，意味着这个 layout 对象所定义的区域被固定用来显示某个用户的画面。您也可以将 fixedVideoUser 设置为 null，这意味着您只是指定了该区域会有视频画面，但画面中的用户具体是谁是不确定的，交由 TRTC 混流服务器根据一定的规则来决定。

案例：

- **案例一：四个用户的画面被混合在了同一个画面中，我们还使用了一张图片作为背景画布**
  - layout1：定义了用户 jerry 的摄像头画面的大小和位置，画面大小为 640x480，位置在画面的中上部。
  - layout2、layout3、layout4：均没有指定具体的用户 Id，因此 TRTC 会根据一定的规则选择房间中 3 路用户的画面安置在这三个位置。



- **案例二：四位用户的摄像头画面和一路屏幕分享的画面被混合在了同一个画面中**
  - layout1：定义了用户 jerry 的屏幕分享画面的大小和位置，画面大小为 1280x720，填充模式为可能有黑边的 Fit 模式，背景填充色为黑色，位置在画面的左侧。
  - layout2：定义了用户 jerry 的摄像头画面的大小和位置，画面大小为 300x200，填充模式为 Fill 模式，位置在画面的右上角。
  - layout3、layout4、layout5：均没有指定具体的用户 Id，因此 TRTC 会根据一定的规则选择房间中 3 路用户的画面安置在这三个位置。



5. 调用 startPublishMediaStream 接口，并通过 onStartPublishMediaStream 监听本地 API 调用是否成功，如果成功 onStartPublishMediaStream 中的 taskId 会返回一个不为空字符串。
6. 如果您需要变更混流参数，例如想要调整多个画面的排版模式，您只需要通过第六步中的 taskId 调用 updatePublishMediaStream API 并传递新的 TRTCStreamMixingConfig 参数即可。TRTCStreamEncoderParam 不建议您在转推过程中进行变更，这会影响到 CDN 播放器的稳定性。
7. 如果您需要停止发布动作，只需要调用 stopPublishMediaStream 并传入之前通过 onStartPublishMediaStream 获得 taskId 即可。

参考代码

如下这段代码的功能是将房间中多个用户的音视频流混合并发布到直播 CDN 上：

java

```
// 指定发布模式为 TRTC_PublishMixedStream_ToCdn
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishMixedStream_ToCdn;
// 指定发布的 CDN 推流地址
TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
target.cdnUrlList.add(cdnUrl);
```

Objective-C

```
// 指定发布模式为 TRTCPublishMixStreamToCdn
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishMixStreamToCdn;
// 指定发布的 CDN 推流地址
TRTCPublishCdnUrl* cdnUrl = [[TRTCPublishCdnUrl alloc] init];
cdnUrl.rtmpUrl = @"rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = YES;
NSMutableArray* cdnUrlList = [NSMutableArray new];
[cdnUrlList addObject:cdnUrl];
target.cdnUrlList = cdnUrlList;
// 设置混合后的音视频流的二次编码参数
TRTCStreamEncoderParam* encoderParam = [[TRTCStreamEncoderParam alloc] init];
encoderParam.videoEncodedWidth = 1280;
encoderParam.videoEncodedHeight = 720;
encoderParam.videoEncodedFPS = 15;
encoderParam.videoEncodedGOP = 3;
```

```
encoderParam.videoEncodedKbps = 1000;
encoderParam.audioEncodedSampleRate = 48000;
encoderParam.audioEncodedChannelNum = 1;
encoderParam.audioEncodedKbps = 50;
encoderParam.audioEncodedCodecType = 0;
// 设置画面的布局参数
TRTCStreamMixingConfig* config = [[TRTCStreamMixingConfig alloc] init];
NSMutableArray* videoLayoutList = [NSMutableArray new];
TRTCVideoLayout* layout1 = [[TRTCVideoLayout alloc] init];
layout1.zOrder = 0;
layout1.rect = CGRectMake(0, 0, 720, 1280);
layout1.fixedVideoStreamType = TRTCVideoStreamTypeSub;
layout1.fixedVideoUser.intRoomId = 1234;
layout1.fixedVideoUser.userId = @"mike";
TRTCVideoLayout* layout2 = [[TRTCVideoLayout alloc] init];
layout2.zOrder = 0;
layout2.rect = CGRectMake(1300, 0, 300, 200);
layout2.fixedVideoStreamType = TRTCVideoStreamTypeBig;
layout2.fixedVideoUser.intRoomId = 1234;
layout2.fixedVideoUser.userId = @"mike";
TRTCVideoLayout* layout3 = [[TRTCVideoLayout alloc] init];
layout3.zOrder = 0;
layout3.rect = CGRectMake(1300, 220, 300, 200);
layout3.fixedVideoStreamType = TRTCVideoStreamTypeSub;
layout3.fixedVideoUser = nil;
[videoLayoutList addObject:layout1];
[videoLayoutList addObject:layout2];
[videoLayoutList addObject:layout3];
config.videoLayoutList = videoLayoutList;
config.audioMixUserList = nil;
// 发起混流
[_trtcCloud startPublishMediaStream:target encoderParam:encoderParam mixingConfig:config];
```

C++

```
// 指定发布模式为 TRTCPublishMixStreamToCdn
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishMixStreamToCdn;
// 指定发布的 CDN 推流地址
TRTCPublishCdnUrl* cdn_url = new TRTCPublishCdnUrl[1];
cdn_url[0].rtmpUrl = "rtmp://tencent/live/bestnews";
cdn_url[0].isInternalLine = true;
target.cdnUrlList = cdn_url;
target.cdnUrlListSize = 1;
// 设置混合后的音视频流的二次编码参数
TRTCStreamEncoderParam encoder_param;
encoder_param.videoEncodedWidth = 1280;
encoder_param.videoEncodedHeight = 720;
encoder_param.videoEncodedFPS = 15;
encoder_param.videoEncodedGOP = 3;
encoder_param.videoEncodedKbps = 1000;
encoder_param.audioEncodedSampleRate = 48000;
encoder_param.audioEncodedChannelNum = 1;
encoder_param.audioEncodedKbps = 50;
encoder_param.audioEncodedCodecType = 0;
// 设置画面的布局参数
TRTCStreamMixingConfig config;
```

```
TRTCVideoLayout* video_layout_list = new TRTCVideoLayout[3];
TRTCUser* fixedVideoUser0 = new TRTCUser();
fixedVideoUser0->intRoomId = 1234;
fixedVideoUser0->userId = "mike";
video_layout_list[0].zOrder = 0;
video_layout_list[0].rect.left = 0;
video_layout_list[0].rect.top = 0;
video_layout_list[0].rect.right = 720;
video_layout_list[0].rect.bottom = 1280;
video_layout_list[0].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeSub;
video_layout_list[0].fixedVideoUser = fixedVideoUser0;
TRTCUser* fixedVideoUser1 = new TRTCUser();
fixedVideoUser1->intRoomId = 1234;
fixedVideoUser1->userId = "mike";
video_layout_list[1].zOrder = 0;
video_layout_list[1].rect.left = 1300;
video_layout_list[1].rect.top = 0;
video_layout_list[1].rect.right = 300;
video_layout_list[1].rect.bottom = 200;
video_layout_list[1].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeBig;
video_layout_list[1].fixedVideoUser = fixedVideoUser1;
video_layout_list[2].zOrder = 0;
video_layout_list[2].rect.left = 1300;
video_layout_list[2].rect.top = 220;
video_layout_list[2].rect.right = 300;
video_layout_list[2].rect.bottom = 200;
video_layout_list[2].fixedVideoStreamType =
     TRTCVideoStreamType::TRTCVideoStreamTypeSub;
video_layout_list[2].fixedVideoUser = nullptr;
config.videoLayoutList = video_layout_list;
config.videoLayoutListSize = 3;
config.audioMixUserList = nullptr;
// 发起混流
trtc->startPublishMediaStream(&target, &encoder_param, &config);
delete fixedVideoUser0;
delete fixedVideoUser1;
delete[] video_layout_list;
```

Dart

```
TRTCPublishTarget target = TRTCPublishTarget();
target.mode = TRTCPublishMode.TRTCPublishMixStreamToCdn;
TRTCPublishCdnUrl cdnUrlEntity = new TRTCPublishCdnUrl();
cdnUrlEntity.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrlEntity.isInternalLine = true;
target.cdnUrlList.add(cdnUrlEntity);

TRTCStreamMixingConfig config = TRTCStreamMixingConfig();
TRTCUser selfUser = TRTCUser();
selfUser.userId = localUserId;
selfUser.intRoomId = localRoomId;

TRTCVideoLayout selfVideoLayout = TRTCVideoLayout();
selfVideoLayout.fixedVideoStreamType = TRTCVideoStreamType.TRTCVideoStreamTypeBig;
selfVideoLayout.rect = Rect(originX: 0, originY: 0, sizeWidth: 1080, sizeHeight: 1920);
```

```
selfVideoLayout.zOrder = 0;
selfVideoLayout.fixedVideoUser = selfUser;
selfVideoLayout.fillMode = TRTCVideoFillMode.TRTCVideoFillMode_Fit;
config.videoLayoutList.add(selfVideoLayout);TRTCUser remoteUser = TRTCUser();

remoteUser.userId = remoteUserId;
remoteUser.intRoomId = remoteRoomId;
TRTCVideoLayout remoteVideoLayout = TRTCVideoLayout();
remoteVideoLayout.fixedVideoStreamType = TRTCVideoStreamType.TRTCVideoStreamTypeBig;
remoteVideoLayout.rect = Rect(originX: 100, originY: 50, sizeWidth: 216, sizeHeight: 384);
remoteVideoLayout.zOrder = 1;
remoteVideoLayout.fixedVideoUser = remoteUser;
remoteVideoLayout.fillMode = TRTCVideoFillMode.TRTCVideoFillMode_Fit;
config.videoLayoutList.add(remoteVideoLayout);

TRTCStreamEncoderParam param = TRTCStreamEncoderParam();
param.videoEncodedWidth = 1080;
param.videoEncodedHeight = 1920;
param.videoEncodedKbps = 5000;
param.videoEncodedFPS = 30;
param.videoEncodedGOP = 3;
param.audioEncodedSampleRate = 48000;
param.audioEncodedChannelNum = 2;
param.audioEncodedKbps = 128;
param.audioEncodedCodecType = 2;

trtcCloud.startPublishMediaStream(target: target, config: config, params: param);
```

方案二：Restful API 发起转推

下文将介绍如何使用 REST API 将 TRTC 房间中的音视频流发布（也称作“转推”）到直播 CDN 上或者回推 TRTC 房间，用于兼容常规的直播播放器进行观看。

功能支持

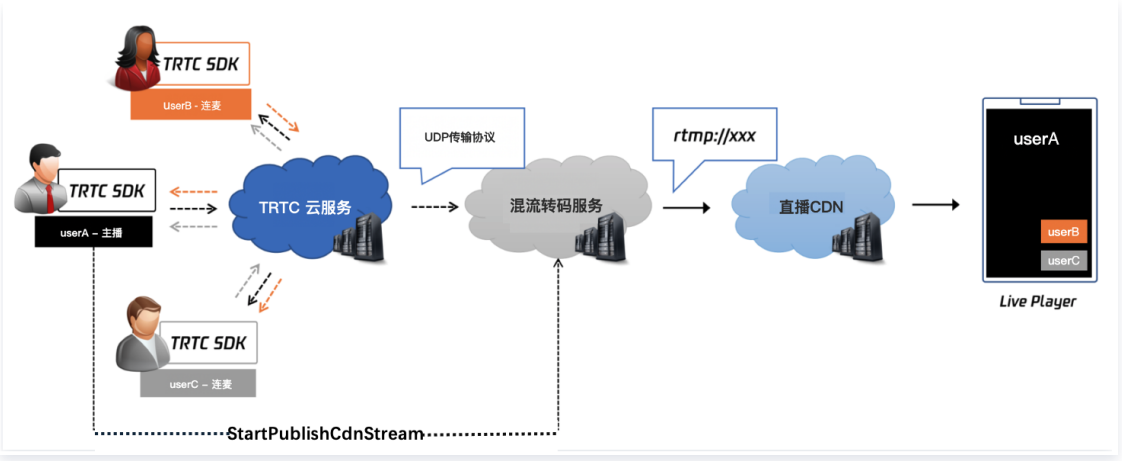
使用 Restful API 发起转推任务时，可以实现下面功能：

- 将单路音视频流转推到直播 CDN 和 TRTC 房间。
- 将多路音视频流混成一路新的流转推到直播 CDN 和 TRTC 房间。
- 支持输出纯音频和音视频。
- 支持自定义布局和动态模板。
- 支持设置背景图、占位图和水印图片。
- 支持对输入视频和图片进行裁剪和缩放。
- 支持在混合的音视频流中增加 SEI 信息。

原理解析

云端混流包含进房、拉流、解码、混合、编码、转推六个过程：

- **进房：**通过您指定的机器人信息，MCU 会创建伴生机器人实例进房。
- **拉流：**根据您指定的混流布局参数，MCU 机器人会拉取相关用户的音视频流。
- **解码：**MCU 会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 会根据您指定的混流布局参数，将多路画面混合在一起。同时，MCU 也会将解码后的多路音频信号进行混音处理。
- **编码：**MCU 会将混合后的画面和声音，根据您配置的输出编码参数进行二次编码，并封装成一路音视频流。
- **转推：**MCU 会将编码封装的音视频数据，分发给您配置的直播 CDN。



启动转推任务

由您的服务器调用 REST API StartPublishCdnStream 可以启动云端转推任务。发起方法如下：

1. **设置基本参数（必需）**

   您需要指定发起转推任务的基本信息，例如您的应用 ID（sdkappid）、主房间信息（RoomId）、主房间类型（RoomIdType）、是否转码（WithTranscoding）。您可以通过设置 WithTranscoding，决定是否转码，若 WithTranscoding 设置为 `true`，则为混流转推，若设置为 `false`，则为旁路转推。

字段名称	描述	必选
SdkAppId	TRTC 的 SdkAppId	是
RoomId	主房间信息 RoomId	是
RoomIdType	主房间信息 RoomType	是
WithTranscoding	是否转码	是

2. **设置机器人参数（必需）**

   您需要指定进房机器人的 AgentParams 参数，MCU 会根据您指定的参数创建实例进房。

字段名称	描述	必选
AgentParams.UserId	转推服务在 TRTC 房间使用的 UserId，请勿与房间内正常用户使用的 UserId 一致	是
AgentParams.UserSig	转推服务加入TRTC房间的用户签名	是
AgentParams.MaxIdleTime	空闲等待时间	否

3. **设置音频参数（必需）**

   如果您需要转推输出音频流，您需要指定 AudioParams 参数。MCU 会根据您配置的 AudioEncode，MCU 会输出对应格式的音频流。针对混流转推您可以配置 SubscribeAudioList 指定混流哪些用户的音频。

字段名称	描述	必选
AudioParams.AudioEncode	旁路/混流–音频输出编码参数	否
AudioParams.SubscribeAudioList	混流–音频用户白名单	否

   具体含义可以参见 McuAudioParams 中参数介绍。

4. **设置视频参数**

   如果您需要转推输出视频流，您需要指定 VideoParams.VideoEncode 参数。针对混流转推，MCU 会根据您配置的 VideoEncode，输出对应格式的视频流。针对旁路转推，当单流用户未上行视频时，MCU会根据您配置的 VideoEncode 补黑帧视频流。

针对混流转推，您还可以配置以下参数：

- 您可以配置 LayoutParams，指定您需要的画面布局。
- 您可以配置 BackGroundColor，指定您需要的画布背景色。
- 您可以配置 BackgroundImageUrl，指定您需要的画布背景图。
- 您可以配置 WaterMarkList，指定您需要的水印布局。

字段名称	描述	必选
VideoParams.VideoEncode	旁路/混流-视频输出编码参数	否
VideoParams.LayoutParams	混流-布局参数	否
VideoParams.BackGroundColor	混流-画布背景颜色	否
VideoParams.BackgroundImageUrl	混流-画布背景图 URL	否
VideoParams.WaterMarkList	混流-水印参数	否

具体含义可以参见 McuVideoParams 中参数介绍。

混流布局类型介绍：

VideoParams.LayoutParams.MixLayoutMode 存在四种布局模式，您可以根据您的需求选择一种布局。

动态布局（1：悬浮布局，2：屏幕分享布局，3：九宫格布局），静态布局（4：自定义布局（默认））。

- **悬浮布局**

  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。

- **屏幕分享布局**

  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 VideoParams.LayoutParams.MaxVideoUser 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。

- **九宫格布局**

  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。

- **自定义布局（默认）**

  适用于需要自定义排布各路画面位置的场景，您可以通过 VideoParams.LayoutParams 中的 MixLayoutList 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 MixLayoutList 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 MixLayoutList 数组中指定的各个位置上。最多支持混入16个输入流，如果子画面只设置占位图，也被算作一路。

如果 MixLayoutList 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 MixLayoutList 数组中预设的位置被用完后，排版引擎将不再混合其他用户的画面和声音。

具体含义可以参见 McuAudioParams 中参数介绍。

注意：

最多支持混入16路音视频流，如果用户只上行音频，也会被算作一路；自定义布局中，如果子画面只设置占位图，也被算作一路。

5. **设置单流旁路用户信息（单流旁路转推，必需）**

如果您需要单流旁路转推，此时 WithTranscoding 您应该设置成 false，同时您需要指定 SingleSubscribeParams 参数 MCU 会将您指定用户的音视频流分发到您指定的直播 CDN。

字段名称	描述	必选
SingleSubscribeParams.UserMediaStream.UserInfo	TRTC 用户参数	否
SingleSubscribeParams.UserMediaStream.StreamType	主辅路流类型	否

具体含义可以参见 SingleSubscribeParams 中参数介绍。

6. **设置转推 CDN 参数（转推 CDN 时填写）**

您需要指定分发 CDN 的 PublishCdnParams 参数，MCU 会将编码后的音视频流转发到您设置的 CDN 地址。

字段名称	描述	必选
PublishCdnParams.N.PublishCdnUrl	CDN 转推 URL	是
PublishCdnParams.N.IsTencentCdn	是否是腾讯云 CDN，0为转推非腾讯云 CDN，1为转推腾讯 CDN，不携带该参数默认为1。 **注意：** **1：为避免误产生转推费用，该参数建议明确填写，转推非腾讯云 CDN 时会产生转推费用，详情参见接口文档说明** **2：国内站默认只支持转推腾讯云 CDN，如您有转推第三方 CDN 需求，请联系腾讯云技术支持**	否

注意：
最多支持设置10个推流 CDN 参数。

7. **设置回推 TRTC 房间参数（回推 TRTC 房间时填写）**

您需要指定回推 TRTC 房间 的 FeedBackRoomParams 参数， MCU 会将编码后的音视频流转发到您设置的 TRTC 房间。

字段名称	描述	必选
FeedBackRoomParams.N.RoomId	回推房间的RoomId	是
FeedBackRoomParams.N.RoomIdType	回推房间的类型，0为整形房间号，1为字符串房间号	否
FeedBackRoomParams.N.UserId	回推房间使用的UserId **注意：** **这个UserId不能与其他TRTC或者转推服务等已经使用的UserId重复，建议可以把房间ID作为UserId的标识的一部分。**	是
FeedBackRoomParams.N.UserSig	回推房间UserId对应的用户签名，具体计算方法请参考TRTC计算 UserSig 的方案。	是

注意：
最多支持设置10个回推房间参数。

8. **设置 SEI 参数（非必需）**

如果您想要通过 SEI 给 CDN 观众发送自定义信息，可以通过设置 SeiParams 参数实现。

字段名称	描述	必选
McuSeiParams.LayoutVolume	音量布局 SEI，内容是固定的 JSON 结构，具体见下面说明。	否
McuSeiParams.PassThrough	透传 SEI。	否

TRTC 支持自定义透传 SEI（McuPassThrough）和音量布局 SEi（LayoutVolume）两种，具体说明如下：

- 自定义透传 SEI 内容完全由您的业务指定，设置后 CDN 观众会定时或每个关键帧收到 SEI 信息，支持通过更新转推任务接口修改 SEI 内容。
- 音量布局 SEI 内容由您业务指定的自定义信息（由 McuLayoutVolume 中的 AppData 参数指定）和 TRTC 混流引擎自动识别并添加的画布信息以及参与混流用户的位置和音量信息组成，设置后 CDN 观众会定时或每个关键帧收到 SEI 信息，支持通过更新转推任务接口修改自定义信息。音量布局 SEI 的内容是固定的 JSON 结构（如下面的音量布局 SEI 内容示例），其中 app_data 字段为业务指定的自定义信息，canvas 字段为 TRTC 混流引擎自动识别的混流输出的画布信息，regions 为 TRTC 混流引擎自动识别的参与混流用户的位置及音量信息等，布局中的 volume 为混流用户的音量（范围为0 - 255），值越大表示音量越大。

McuLayoutVolume 中的 AppData 参数设置为"test" 时，音量布局 SEI 内容示例：

```
{
    "app_data": "test",
    "canvas": {
        "w": 1280,
        "h": 720
    },
    "regions": [
        {
            "uid": "test1",
            "zorder": 1,
            "volume": 60,
            "x": 0,
            "y": 0,
            "w": 640,
            "h": 360
        },
        {
            "uid": "test2",
            "zorder": 1,
            "volume": 80,
            "x": 640,
            "y": 0,
            "w": 640,
            "h": 360
        }
    ],
    "ver": "1.0",
    "ts": 1648544726
}
```

更新转推任务

由您的服务器调用 REST API UpdatePublishCdnStream 可以更新云端转推任务。此时，您需要使用 StartPublishCdnStream 返回的 TaskId，发起更新转推。对于此 API 使用方法，您可以参见 启动转推任务 。

停止转推任务

由您的服务器调用 REST API StopPublishCdnStream 可以更新云端转推任务。此时，您需要使用StartPublishCdnStream返回的TaskId，发起停止转推。

字段名称	描述
SdkAppId	TRTC 的 SdkAppId
TaskId	转推任务唯一的 String Id

方案三：自动旁路发起转推

除了上面通过接口触发旁路的方式，TRTC 还提供了无需手动发起的自动旁路方案。使用自动旁路方案时，TRTC 房内的主播上行音视频后，会自动发起旁路任务，将主播的单路流推到 CDN ，主播退房后旁路任务自动结束。有两种方式可以触发自动旁路，分别为指定流自动旁路和全局自动旁路。

指定流自动旁路

旁路转推配置开启指定流旁路后，进房参数带了自定义流 ID 的主播上行音视频后会触发自动旁路，没有带的不会发起。
步骤1：在实时音视频控制台 > 应用管理 > 功能配置 > 旁路转推配置 配置指定流旁路。

步骤2：调用 `TRTCCloud` 的 `enterRoom` 函数时，通过其参数 `TRTCParams` 中的 `streamId` 参数指定自定义流 ID。

```
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
```

全局自动旁路

旁路转推配置开启全局自动旁路后，TRTC 房间内的主播，只要上行音视频就会触发自动旁路。

开启全局自动旁路后，也可以通过前面所述的进房参数指定推到直播的自定义流 ID ，如果没有指定，系统会生成一个缺省的流 ID，生成规则如下：

- **拼装流 ID 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）。
- **拼装流 ID 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

实现拉流播放与优化

给 SDK 配置 License 授权

实时音视频（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。若您使用移动端（iOS&Android）10.1及其之后版本的实时音视频（TRTC）SDK 实现 CDN 直播观看，则须配置 License 授权，否则可跳过本步骤。

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
*)launchOptions {
    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
```

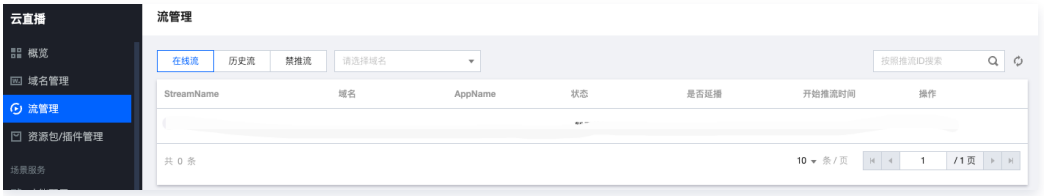
> ⚠ **注意：**
> License 中配置的 packageName/BundleId 必须和应用本身一致，否则会播放失败。

获取播放地址并对接播放

当您完成的推流操作后，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/AppName(默认live)/StreamName(流ID).flv
```

您的播放域名、AppName和StreamName可前往云直播控制台 中流管理进行查看：



您可以得到三路播放地址：

```
rtmp 协议的播放地址：rtmp://example.myhost.com/AppName_example/StreamName_example
flv 协议的播放地址：http://example.myhost.com/AppName_example/StreamName_example.flv
hls 协议的播放地址：http://example.myhost.com/AppName_example/StreamName_example.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

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）
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。

实时音视频费用

- **混流转码费用：**如果使用 发布混合后的音视频流到直播 CDN 的方式，将会产生混流费用，费用说明请见 云端混流转码计费说明 。如使用发布当前用户的音视频流（单流）到直播 CDN 将不会产生此部分费用。
- **旁路转推费用：**费用详情请见 转推计费说明 。
- **音视频时长费用：**音视频时长费用将根据房间内用户订阅实际产生的音视频费用正常收取，详情请见 音视频时长计费说明 。

其他云服务费用

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

实时音视频（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。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 房间内观众订阅流播放无需 License 授权。
> - 如果业务有点播播放或转推直播流播放使用到 SDK 中播放器能力则需要购买相应 License 授权。

您可直接 购买播放器 License 解锁 SDK 移动端或 Web 端视频播放功能，或通过 购买的云点播流量包 免费获赠播放器 License 或 短视频 License 解锁 SDK 移动端基础版的视频播放功能。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 音视频终端 SDK（腾讯云视立方）授权费用，License 购买完成后可参考 License 操作指引 进行新增和续期等操作。

费用的节约

基于客户端 SDK API 混流方案下，要停止后端混流任务，需要满足如下条件之一：
- 发起混流任务（即调用 `startPublishMediaStream` ）的主播退出了房间
- 调用 `stopPublishMediaStream` 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。

常见问题

1、能否监听 CDN 流的当前状态，状态异常时该如何处理？

可通过监听 `onCdnStreamStateChanged` 回调获取最新的后台任务状态更新回调。更多细节可参考 API 文档。

2、如何从单路转推切换到混流转推，是否需要手动停止再重新创建转推任务？

可以从单路推流任务直接切换到混流任务，只需对单路推流任务 `taskid` 发起 `updatePublishMediaStream` 推流任务变更即可。但是为了确保推流链接稳定，从单路推流切换到混流推流无法切换到纯音频或者纯视频模式。单路推流默认是音视频模式，切换后的混流也需要是音视频模式。

3、如何实现纯视频混流？

配置混流设置时， `TRTCStreamEncodeParam` 里音频相关参数不要设置且 `TRTCStreamMixingConfig` 里 `audioMixUserList` 设置为空。

4、可以给混流画面添加水印吗？

可以，可通过 `TRTCStreamMixingConfig` 的 `watermarkList` 进行设置。更多细节可参考 API 文档。

5、可以混流屏幕分享内容么，我们是做教培的，需要混流转推老师的屏幕分享画面？

可以的。建议您使用辅路推送屏幕分享画面，然后发起将老师的摄像头画面与屏幕分享画面配置到同一个混流任务内。设置混流辅路时只需配置 `TRTCVideoLayout` 的 `fixedVideoStreamType` 为 `TRTCVideoStreamTypeSub` 即可。

6、预设排版模式下，音频流的混合时怎么确定的？

使用预设排版模式的时候，混流里的音频将会从当前房间内所有上行音频里选取最多 16 路音频进行混合。

实现云端录制与回放

最近更新时间：2025-07-02 20:13:14

场景说明

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的情况。

> **说明：**
> 下文将针对实时音视频最新推出的云端录制能力进行使用说明。自**2022年08月01日**起新创建的应用（SdkAppId）录制功能类型为新版云端录制。若您当前的 TRTC 应用（sdkappid）使用的是旧版云端录制，详情请参见 旧版云端录制 。判断当前应用的云端录制的类型和旧版云端录制能力切换为新版的方式，详情请参见 控制台 > 云端录制说明 。

功能说明

通过 TRTC 的云端录制功能，您可以将房间中的每一个用户的音视频流都录制成独立的文件（单流录制），或者把同一个房间的音视频媒体流合流录制成一个文件（合流录制）。

- **订阅流：** 我们支持通过制定订阅用户的黑白名单的方式来指定您需要订阅的用户媒体流（**仅支持 API 录制**）。
- **转码参数：**合流的场景下，我们支持通过设置编解码的参数来指定录制的视频文件的质量。
- **合流参数：**合流的场景下，我们支持多种灵活可变的自动多画面布局模板和自定义布局模板。
- **文件存储：**支持存储到云点播 VOD 或对象存储 COS。
- **回调通知：**我们支持回调通知的能力，通过配置您的回调域名，云端录制的事件状态会通知到您的回调服务器。

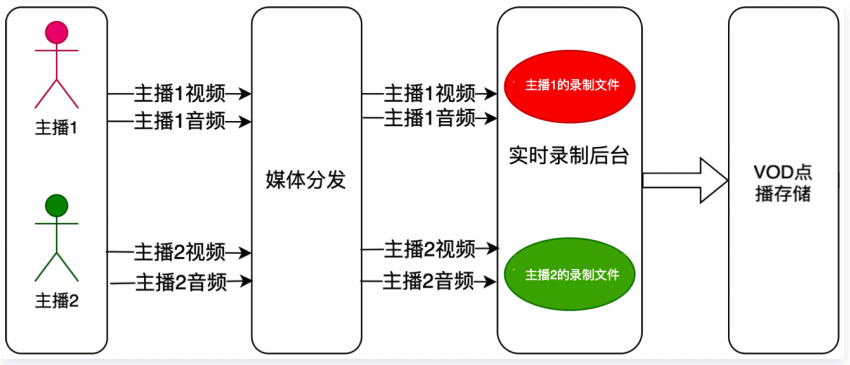
单流录制和合流录制说明

单流录制

如下图所示为单流录制的场景，房间1234里面主播1和主播2都上行了音视频流，假设您订阅了主播1和主播2的音视频流，并设置录制模式为单流录制，录制后台会分别拉取主播1和主播2的音视频流，并把他们录制成独立的媒体文件包含：

- 主播1的一个音视频录制文件。
- 主播2的一个音视频录制文件。

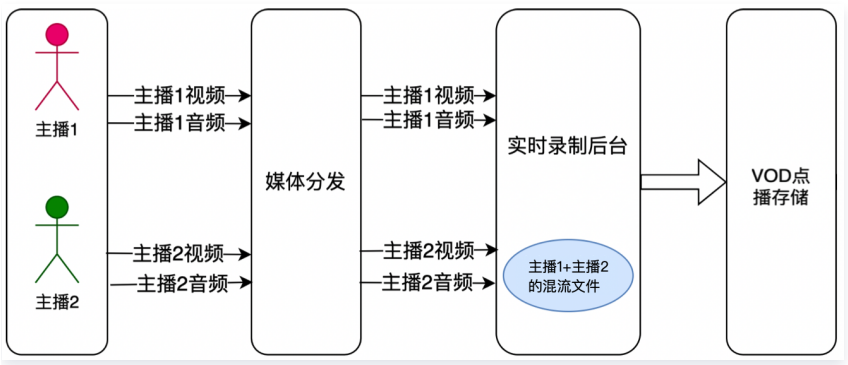
录制后台会把这些文件上传到您指定的云存储平台（云点播 VOD 或对象存储 COS）。具体录制流程如下：



合流录制

如下图所示为合流录制的场景，房间1234里面有主播1和主播2都上行了音视频流，假设您订阅了主播1和主播2的音视频流，设置录制模式为合流录制，录制后台会分别拉取主播1和主播2的音视频流，并把他们的视频流按照您配置多画面模板进行合流，音频流进行混音，最后把媒体流混合成一路媒体文件。包含：合流后的一个音视频录制文件，具体发起方式请见 API手动录制 。

录制后台会把这些文件上传到您指定的云存储平台。具体录制流程如下：



录制文件命名和文件切分说明

录制 MP4/AAC 文件名命名规则

- **单流录制 MP4/AAC 文件名规则：** `<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Index>.mp4/aac`
- **合流录制 MP4/AAC 文件名规则：** `<SdkAppId>_<RoomId>_<Index>.mp4/aac`

录制 HLS 文件名命名规则

- **单流录制 HLS 文件名规则：** `<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Type>.m3u8`
- **合流录制 HLS 文件名规则：** `<SdkAppId>_<RoomId>.m3u8`
- **字段含义说明：**

字段	含义
<SdkAppId>	录制任务的 SdkAppId
<RoomId>	录制的房间号，如果这里 RoomId 如果是字符串房间号，我们会对房间号先做 base64 操作，再把 base64 后的字符串中符号'/'替换成 '-'（中划线），符号'='替换成 '.'
<UserId>	录制的用户 ID，UserId 会先做 base64 操作，再把base64后的字符串中符号'/'替换成 '-'（中划线），符号'='替换成 '.'
<MediaId>	主辅流标识，main 代表主流（摄像头），aux 代表辅流（屏幕分享）
<Index>	如果没有触发切片逻辑（大小超过2GB或超过设置的切片时长）则无该字段，否则为切片的索引号，从1开始递增
<Type>	录制文件流类型，audio/video/audiovideo

> **说明：**
> 自定义设置文件名称前缀：使用 API 录制 存储至云点播 VOD 时，可通过 TencentVod 中的 UserDefineRecordId 参数自定义文件名称前缀，前缀与默认录制文件名之间用__UserDefine_u_分隔。

录制文件切分说明

- **录制 MP4/ACC 文件切分的条件：**
  - 录制切分时长可设置范围1-1440分钟，默认1440分钟。
  - 单个 MP4/AAC 文件大小达到 2GB。
- **录制 HLS 文件切分的条件：**
  - 录制任务持续时间超过14天时，HLS 文件将会被切分。

录制上传云存储说明

录制后台会在录制结束后将录制的文件通过您指定的方式上传到云存储平台（云点播 VOD 或对象存储 COS），并通过回调的形式把播放地址发送给您。如果录制模式为单流录制模式，每一个订阅的主播都会有一个对应的播放地址；如果录制模式为合流录制模式，只有一个合流后媒体的播放地址。

1. 通过 API 发起录制：在存储参数 StorageParams 中必须指定 CloudVod（存储至云点播 VOD）或 CloudStorage（存储至对象存储 COS 或第三方云存储）的参数，请确保已经开通对应的云存储服务且未欠费。
2. 上传任务的过程中使用 DescribeCloudRecording 只能查询到录制任务进行状态，不会携带录制文件的信息。

> ⚠ **注意：**
> 文件录制后会上传至您指定云存储平台（云点播 VOD 或对象存储 COS），为确保录制文件成功，请确保您指定的云点播 VOD 或对象存储 COS 服务可用。

API 接口和录制并发限制

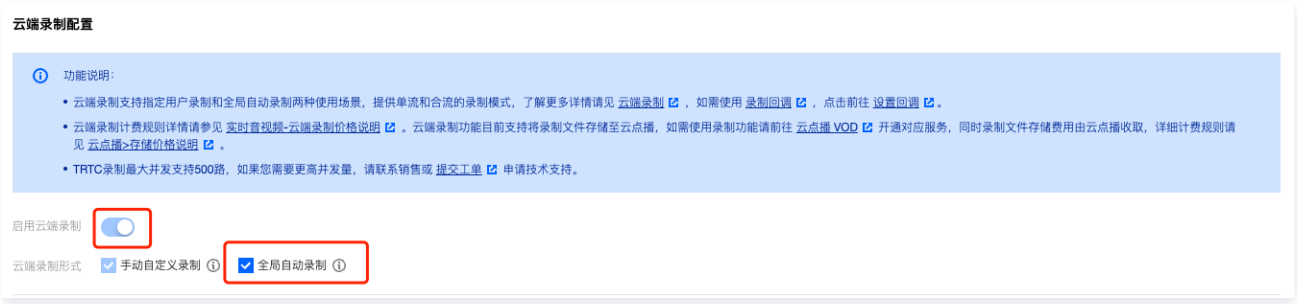
- 录制接口的调用频率限制为20qps（如需提高 QPS 请 提交工单）。
- 单个接口超时时间为6秒。
- 单个应用下**默认并发录制支持500路**（全局自动录制和 API 录制任务的总和），超过并发限制的任务会失败，如需更多并发路数，请 提交工单 联系我们。
- 单次录制任务最大支持同时订阅的房间内主播数为25个，主播只上行音频也会单独占据一路。

录制控制方案

TRTC 提供了两种云端录制方案，分别是 全局自动录制 和 API 手动录制，这两种方案并不冲突，可以同时使用两种录制方案，但会产生两份录制文件和费用。API 录制相比全局自动录制的优点是录制灵活、功能完备，客户可以指定录制订阅房间内的主播，自定义合流布局，录制中途更新布局和订阅等。全局自动录制的优点是录制不需客户启动和停止，由 TRTC 后台管理录制任务。

方案一：全局自动录制

TRTC 提供了一种无需手动发起并管理录制任务的自动录制方式，要使用该种录制方案，请前往**实时音视频控制台** > **应用管理** > **录制管理**中开启云端录制功能，完成全局自动录制模板配置并开启全局自动录制。
生效后（生效等待5-10分钟）TRTC 房间中的主播上行音视频后将触发启动录制任务，房间内主播都退房且超过设置的等待续录时间后将触发停止录制任务。



开启全局自动录制功能前请配置全局自动录制模板，全局自动录制支持 单流录制（即每个主播单个录制一个文件），开启后只对新创建的房间有效，对开启自动录制功能之前已经创建的房间不生效。

全局单流录制

全局单流录制录制格式支持音视频录制、纯音频录制和纯视频录制，录制文件支持 MP4、HLS 和 AAC（纯音频录制格式下），录制文件切片策略请参见 录制文件切分说明。

配置项	说明
录制模式	• 单流录制：房间中的每个主播的视频画面都会单独保存成一份文件 如需录制多个主播混合后的画面，请使用API手动合流录制
录制格式	• 音视频格式：录制房间内的音频和视频流，适用于视频通话、互动直播场景 • 纯音频格式：只录制房间内的音频流
文件格式	支持 MP4 、HLS 和 AAC（纯音频格式下）
单个录制文件时长	可用于指定录制文件切片时长，设置范围1-1440分钟，默认1440分钟

续录等待时长	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：秒，**取值范围1 – 86400（默认30s）**。 **注意：在续录等待期内，单流录制会按照音频时长收取录制费用，请合理设置。**
录制文件存储	支持存储至 云点播 VOD 或 对象存储 COS。 云点播：需支持指定云点播应用、录制文件在云点播的存储时间以及绑定点播任务流。 对象存储：存储至 COS 的服务由云点播联合提供，如需存储至 VOD，您需授权给 VOD 相关 COS 桶的读写权限，并需要完成您的存储桶 bucket 和云点播应用的绑定，绑定后云点播会为您创建一个应用，通过选择这个绑定的应用，可完成指定的存储桶设置。
回调地址与回调密钥	新版云端录制提供了详尽的录制事件功能，您可以配置可用的服务端url用于接收录制回调事件，同时支持配置回调密钥用于校验回调事件的安全性，更多请见。

说明：

- 单流录制模式下房间内的音视频流将按照推流参数进行每一路单独录制，无需设置转码。
- 续录等待时长未到期内录制机器人会在房间内继续等待主播上行进而完成录制，并不会因为主播退房后就立即结束，请合理设置。
- 单流录制最多录制一个房间内的25个主播，如果超过25个主播将会按照进房时间由先到后排序，录制前25位主播（如需单流录制超过25位主播，请参见 API 录制）。

方案二：API 手动录制

启动录制

通过您的后台服务调用 REST API（CreateCloudRecording）来启动云端的录制，需要重点关注参数— **任务 ID（TaskId）**；这个参数是本次录制任务的唯一标识，您需要保存下这个任务 ID 作为后续针对这个录制任务接口操作的输入参数。

> **说明：**
> 1. 发起云端录制任务的接口 CreateCloudRecording 中需要您指定分配录制机器人的进房参数 UserId 和 UserSig（如何获取 UserSig），请不要与您房间内的正常主播或观众使用的 UserId 重复且不可与正在录制中的房间内指定的录制机器人 UserId 一致，否则会导致录制任务失败。
> 2. **手动录制下，您可以前往控制台配置回调地址，以接受录制回调事件**，请见 录制回调说明 。

录制的模式（RecordMode）

- **单流录制**：实时录制房间内每个主播的音频视频单录制为一个音视频文件上传到云存储平台（云点播 VOD 或对象存储 COS）。
- **合流录制**：将房间内您所订阅所有主播的音视频流混录成一个音视频文件上传到云存储平台（云点播 VOD 或对象存储 COS）。

指定录制用户（SubscribeStreamUserIds）

默认情况下，云端录制会录制房间内所有的媒体流（最多25路），超过25个用户，默认录制最先进房的25位主播。您也可以通过参数 SubscribeStreamUserIds 指定想要录制或者不想录制的主播用户的黑白名单信息，当然我们也支持在录制的过程中进行更新操作。单流录制场景，如果房间内主播超过25人，可以通过设置订阅名单发起多次录制任务实现。

指定存储位置和录制格式（StorageParams）

存储位置：支持存储至云点播 VOD 或对象存储 COS，请通过在 StorageParams 中 CloudVod 参数进行指定您的指定存储参数。
录制格式：默认录制格式是 MP4, 如果需要录制成 HLS 格式, 可通过 CloudVod 下 TencentVod 内的 **MediaType** 设置为1来指定格式为 HLS；如果需要录制AAC 格式文件， 可通过 CloudVod 下 TencentVod 内的 **MediaType** 设置为2来指定格式为 AAC（仅在 StreamType =1纯音频录制时有效）

录制开始的时间的获取

通过订阅回调，监听录制回调事件。在事件类型311中的 StartTimeStamp 字段您可以获取到录制文件对应的录制起始时间戳，EndTimeStamp 字段可以获取到对应的录制结束时间戳。

```
{
    "EventGroupId": 3,
    "EventType": 311,
    "CallbackTs": 1622186289148,
    "EventInfo": {
        "RoomId": "xx",
        "EventTs": "1622186289",
        "UserId": "xx",
        "TaskId": "xx",
        "Payload": {
            "Status": 0,
            "TencentVod": {
              "UserId": "anchor1",
              "TrackType": "video",
              "MediaId": "main",
              "FileId": "xxxxx",
              "VideoUrl": "http://xxxxx",
              "CacheFile": "xxxxxx",
              "StartTimeStamp": 1622186279,
              "EndTimeStamp": 1622186811
        }
    }
}
```

```
}
```

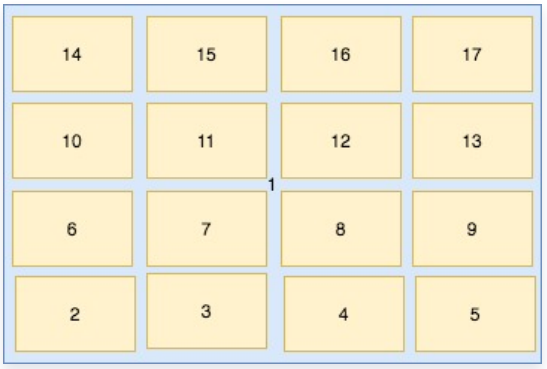
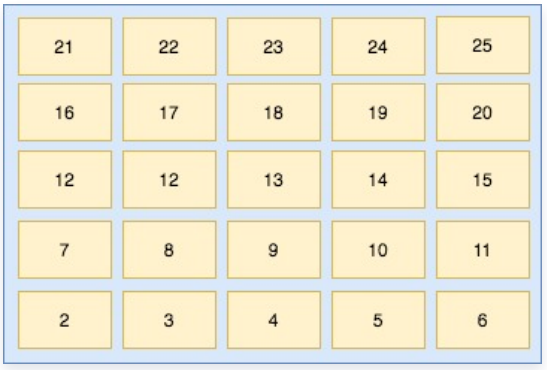
合流录制的布局模式参数（MixLayoutMode）

支持 悬浮布局 、屏幕分享布局 、九宫格布局（默认） 和 自定义布局 四种布局：

- **悬浮布局**

  默认第一个进入房间的主播（也可以指定一个主播）的视频画面会铺满整个屏幕。其他主播的视频画面从左下角开始依次按照进房顺序水平排列，显示为小画面，小画面悬浮于大画面之上。当画面数量小于等于17个时，每行4个（4 × 4排列）。当画面数量大于17个时，重新布局小画面为每行5个（5 × 5）排列。最多支持25个画面，如果用户只发送音频，仍然会占用画面位置。

  悬浮布局随着订阅的子画面增加按照下图进行变化：

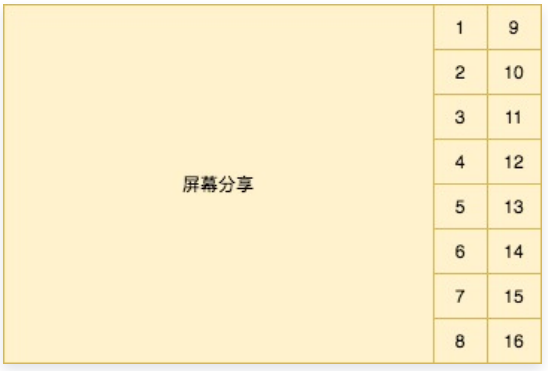
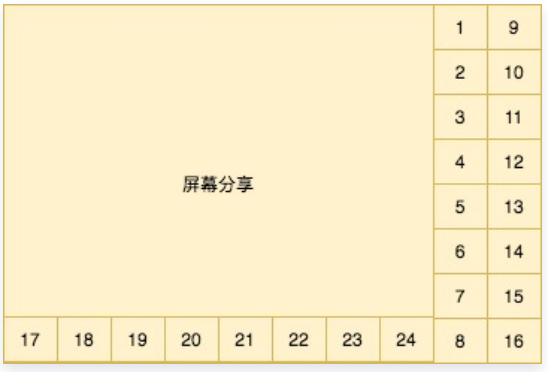
子画面小于等于17个时	• 每个小画面的宽和高分别为整个画布宽和高的0.235 • 相邻小画面的左右和上下间距分别为整个画布宽和高的 0.012 • 小画面距离画布的水平和垂直边距也分别为整个画布宽和高的 0.012	
子画面大于17个时	• 每个小画面的宽和高分别为整个画布宽和高的0.188 • 相邻小画面的左右和上下间距分别为整个画布宽和高的 0.01 • 小画面距离画布的水平和垂直边距也分别为整个画布宽和高的 0.01	

- **屏幕分享布局：**

  指定一个主播在屏幕左侧的大画面位置（如果不指定，那么大画面位置为背景色），其他主播自上而下依次垂直排列于右侧。当画面数量少于17个的时候，右侧每列最多8人，最多占据两列。当画面数量多于17个的时候，超过17个画面的主播从左下角开始依次水平排列。最多支持24个画面，如果主播只发送音频，仍然会占用画面位置。

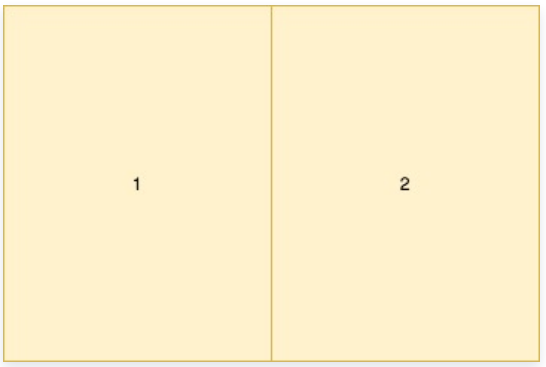
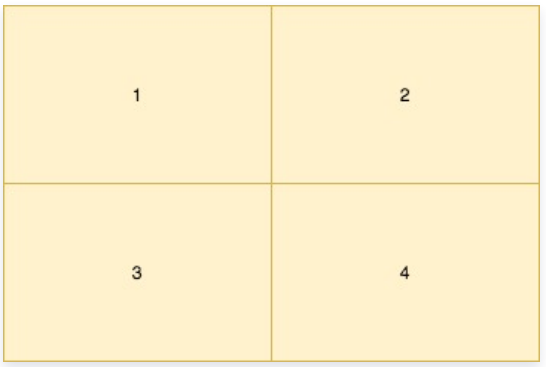
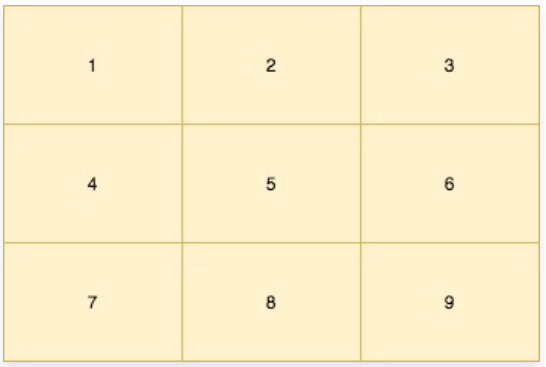
  屏幕分享布局随着订阅的子画面增加按照下图进行变化：

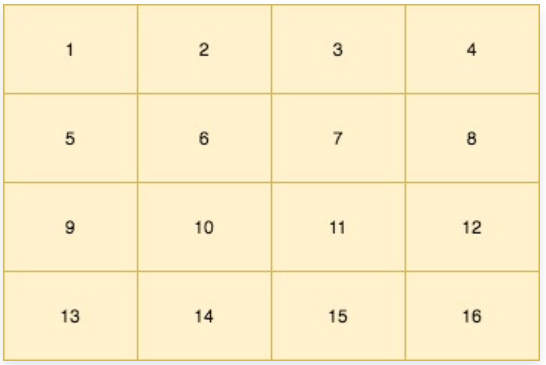
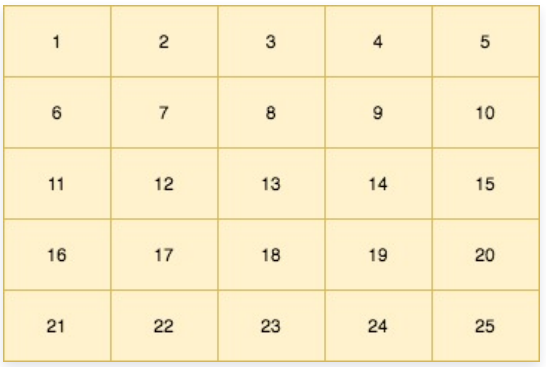
子画面小于等于5个时	• 右侧小画面的宽为整个画布宽的1/5，右侧小画面的高为整个画布高的1/4 • 左侧大画面的宽为整个画布宽的4/5，左侧大画面的高为整个画布高	

子画面大于5且小于等于7个时	• 右侧小画面的宽为整个画布宽的1/7，右侧小画面的高为整个画布高的1/6 • 左侧大画面的宽为整个画布宽的6/7，左侧大画面的高为整个画布高	
子画面大于7且小于等于9个时	• 右侧小画面的宽为整个画布宽的1/9，右侧小画面的高为整个画布高的1/8 • 左侧大画面的宽为整个画布宽的8/9，左侧大画面的高为整个画布高	
子画面大于9小于等于17个时	• 右侧小画面的宽为整个画布宽的1/10，右侧小画面的高为整个画布高的1/8 • 左侧大画面的宽为整个画布宽的4/5，左侧大画面的高为整个画布高	
子画面大于17个时	• 右（下）侧小画面的宽为整个画布宽的1/10，右（下）侧小画面的高为整个画布高的1/8 • 左侧大画面的宽为整个画布宽的4/5，左侧大画面的高为整个画布高的7/8	

- **九宫格布局**

  根据主播的数量自动调整每个画面的大小，每个主播的画面大小一致，最多支持25个画面。
  九宫格布局随着订阅的子画面增加按照下图进行变化：

子画面为1个时	每个小画面的宽和高分别为整个画布宽和高	
子画面为2个时	• 每个小画面的宽为整个画布宽的1/2 • 每个小画面的高为整个画布高	
子画面小于等于4个时	每个小画面的宽和高分别为整个画布宽和高的 1/2	
子画面小于等于9个时	每个小画面的宽和高分别为整个画布宽和高的 1/3	

子画面小于等于16个时	每个小画面的宽和高分别为整个画布宽和高的 1/4	
子画面大于16个时	每个小画面的宽和高分别为整个画布宽和高的1/5	

- **自定义布局**

  根据您的业务需要在 MixLayoutList 内自己定制每个主播画面的布局信息。

合流录制的水印参数 (MixWatermark)

我们支持在合流录制中添加图片水印，最大支持个数为25个，可以在画布任意位置添加水印。

字段名	解释
Top	水印相对左上角的垂直位移
Left	水印相对左上角的水平位移
Width	水印显示的宽度
Height	水印显示的高度
url	水印文件的存储 URL

查询录制（DescribeCloudRecording）

如果需要，您可以调用该接口查询录制服务的状态。

> ⚠ **注意：**
> - 只有录制任务存在的时候才能查询到信息，如果录制任务已经结束会返回错误。
> - 如果是上传云点播 VOD 任务，该接口返回的 StorageFile 为空。

更新录制（ModifyCloudRecording）

如果需要，您可以调用该接口修改录制服务的参数，如订阅黑白名单 SubscribeStreamUserIds（单流和合流录制有效），录制的模板参数 MixLayoutParams（合流录制有效）。

> ⚠ **注意：**

更新操作是全量覆盖的操作，并不是增量更新的操作，您每次更新都需要携带全量的信息，包括模板参数 MixLayoutParams 和黑白名单 SubscribeStreamUserIds，因此您需要保存之前的启动录制的参数或者重新计算完整的录制相关参数。

停止录制（DeleteCloudRecording）

在录制结束之后需要调用停止录制（DeleteCloudRecording）的接口来结束录制任务，否则录制任务会等待到达预设的超时时间 MaxIdleTime 后自动结束。

注意：
MaxIdleTime 的定义是房间内持续没有主播的状态超过 MaxIdleTime 的时长，这里如果房间是存在有主播，但是主播没有上行数据是不会进入超时的计时状态的，此时后台录制会持续工作。建议业务在录制结束的时候调用此接口结束录制任务。

录制回调事件

我们针对云端录制功能提供了多种的回调事件，帮助您及时了解录制任务的处理和完成情况，录制回调地址配置和事件说明请见 云端录制回调 。

录制文件管理

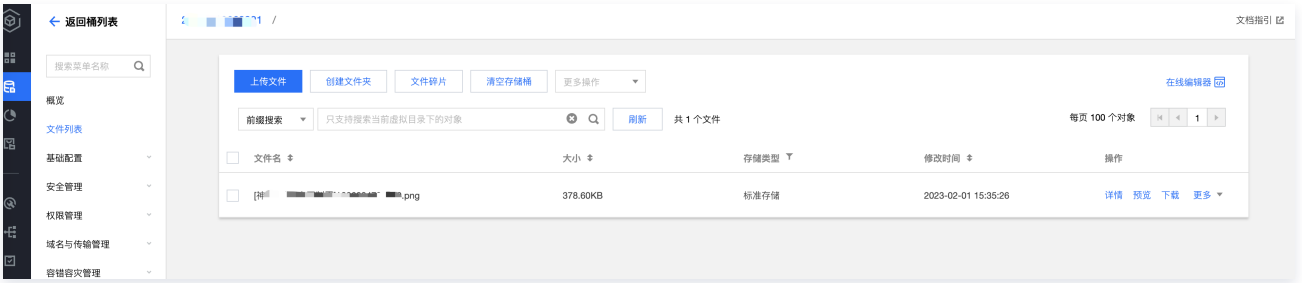
查找录制文件

结束房间完成录制任务后，TRTC 录制系统中录制下来的文件上传至您指定的云存储平台（云点播 VOD 或对象存储 COS）。您可以直接前往 云点播控制台 或 对象存储 COS 控制台 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，按照 录制文件命名规则 填入，例如合流录制下填入：`1400000123_1001`，单击 ，将展示视频名称前缀相匹配的视频文件。

3. 登录 对象存储 COS 控制台，选择您指定的存储桶 Bucket 进行查找：



另外您也可以前往 实时音视频控制台 > **录制管理** > **录制文件管理**中筛选对应点播应用查看对应的录制文件。

存储至腾讯云 COS 的目标录制文件，回调事件内本身不提供播放链接，但可以通过固定规则拼接。拼接规则如下：

`https://<bucket>.cos.<region>.myqcloud.com/<FileNamePrefix>/<Taskid>/<录制文件名>`

举个例子，客户发起录制对应的存储是：`trtc-test-125**39`，存储桶地域 `ap-shanghai`，录制文件前缀设置为 `prefix1/prefix2`，录制文件唯一任务Taskid 为 `m9-bVV*****zQtdRzgE.`，录制文件名 `140**311_68888.m3u8`，那么拼接出来的播放链接为：

```
https://trtc-test-125**39.cos.ap-shanghai.myqcloud.com/prefix1/prefix2/m9-bVV*****zQtdRzgE./140**311_68888.m3u8
```

> **注意：**
> 如果 Taskid 如果有'`+`'字符，需要转义成'`%2B`'。COS 存储桶需要设置公有访问，才可以正常播放录制文件。

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 `TrtcSdkAppIds`（对应发起录制的应用 `SdkAppid`）或 `TrtcRoomIds`（对应发起录制的房间号 `Roomid`）参数进行匹配查找。

REST API请求示例：

```
https://vod.tencentcloudapi.com/?Action=SearchMedia
&TrtcSdkAppIds=1400xxxx123
&TrtcRoomIds=1234
&Sort.Field=CreateTime
&Sort.Order=Desc
&<公共请求参数>
```

> **说明：**
> 如需下载录制文件，请前往 点播控制台-媒资管理 下，找到对应的录制文件，在“操作”中进行下载。

接收录制文件

除了 查找录制文件，您还可以通过在控制台 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，该过程大约默认需要30秒至数分钟（具体时间根据您所录制的文件大小而定，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，您可以通过接收**事件类型为311**的上传成功回调来获取录制文件的播放地址 VideoUrl，具体回调信息见下方：

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
```

```
    "EventTs": 1622191965,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "TencentVod": {
        "UserId": "xx",
        "TrackType": "audio_video",
        "MediaId": "main",
        "FileId": "xxxx",
        "VideoUrl": "http://xxxx",
        "CacheFile": "xxxx.mp4",
        "StartTimeStamp": xxxx,
        "EndTimeStamp": xxxx
      }
    }
  }
}
```

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。

获取点播地址（VideoUrl）

在 接收录制文件 时，可以获取回调消息中 VideoUrl 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 播放器（Player+）、直播 SDK 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：云端录制服务、云点播 VOD 或对象存储 COS 的回放文件存储与处理、云点播 VOD 或对象存储 COS 的播放服务，以及终端 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。

云端录制费用

云端录制费用取决于您所录制的时长和画面分辨率，同时根据录制模式的不同（单流或者合流）进行区分定价，录制费用计算公式如下：
云端录制费用 = 录制音频费用 + 录制视频费用 = 录制音频输入时长用量 × 单路或多路对应的音频单价 + 录制视频各分辨率档位输入时长用量 × 单路或多路对应的相应视频分辨率档位单价

> **说明：**
> 更多详细云端录制费用说明和计费示例，请参见 云端录制计费说明。

文件存储费用

录制出的视频文件存放于云点播 VOD 或对象存储 COS 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。云点播VOD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；对象存储COS存储费用说明请见 按量计费（后付费）。

> **注意：**
> 特别说明：若您选择存储至对象存储 COS 将会收取录制文件投递至 COS 的费用，详见 投递费用说明 ，存储至 VOD 将不收取此项费用。

文件播放费用

如果您录制的视频文件要被用于回看播放，会使用云点播或对象存储的 CDN 播放功能。由于观看本身会产生 CDN 流量消耗，因此需要按照云点播或对象存储的价格进行计费，默认按流量收费。观看的人数越多费用越高，云点播播放费用可以选择 按量计费 进行日结或月结计算，也可以购买 流量套餐包 。对象存储 COS 播放费用请见 流量费用说明 。

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 VOD 或对象存储 COS 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 的音视频通话、直播的播放能力无需 License 授权。

您可直接 购买播放器 License，或通过 购买的云点播流量包 免费获赠播放器 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。

最佳实践

为了保障录制的高可用，建议客户在集成 REST ful API 的同时注意以下几点。

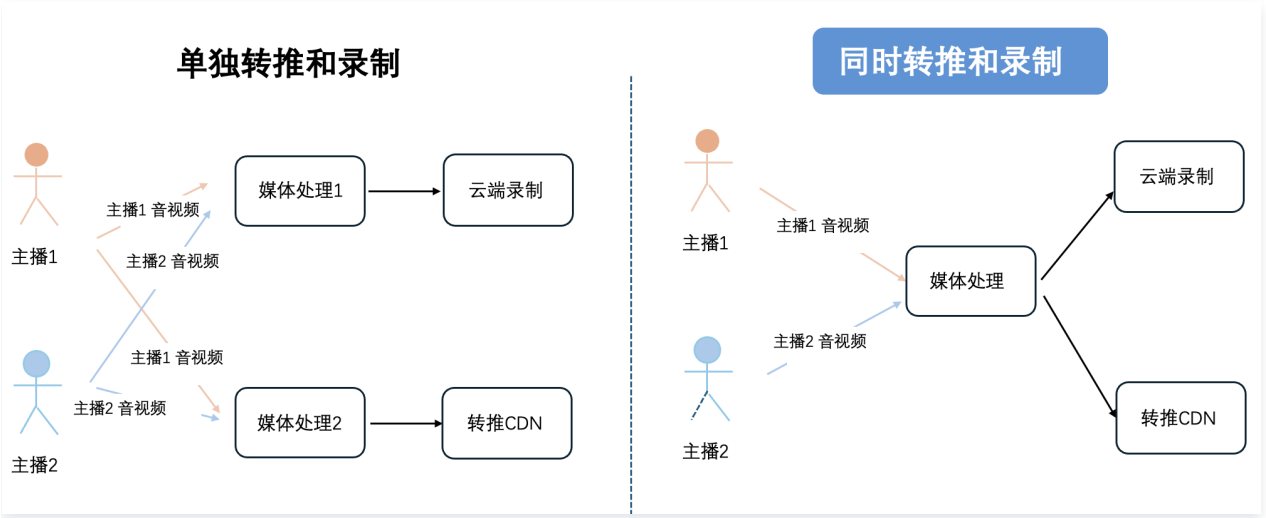
- 调用 CreateCloudRecording 请求后，请关注 HTTP response，如果请求失败，那么需要根据具体的状态码采取相应的重试策略。错误码是由“一级错误码”和“二级错误码”组合而成，例如：`InvalidParameter.SdkAppId` 。
  - 如果返回的 Code 是 `InValidParameter.xxxxx` ，说明输入的参数有误，请根据提示检查参数。
  - 如果返回的 Code 是 `InternalError.xxxxx` ，说明遇到了服务端错误，可以使用相同的参数重试多次，直到返回正常，拿到 taskid 为止。建议使用退避重试策略，如第一次3s重试，第二次6s重试，第三次12s重试，以此类推。
  - 如果返回的 Code 是 `FailedOperation.RestrictedConcurrency` ，说明客户的并发录制任务数，超过了后台预留的资源（默认是500路），请联系腾讯云技术支持来调整最高并发路数限制。
- 如果您有订阅录制回调，当收到 EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP 回调事件，LeaveCode 为500时，说明录制与主播数据长时间断开连接，请再次发起录制任务保证录制的可用性。
- 调用 CreateCloudRecording 接口时，指定的 UserId/UserSig 是录制作为单独的机器人用户加入房间的 ID，请不要和 TRTC 房间内的其他用户重复。同时，TRTC 客户端加入的房间类型必须和录制接口指定的房间类型保持一致，比如 SDK 创建房间用的是字符串房间号，那么云端录制的房间类型也需要相应设置成字符串房间号。
- 录制状态查询，客户可以通过以下几种方式来得到录制相应的文件信息：
  - 成功发起 CreateCloudRecording 任务后15s左右，调用 DescribeCloudRecording 接口查询录制文件对应的信息，如果查询到状态为 idle 说明录制没有拉到上行的音视频流，请检查房间内是否有主播上行。
  - 成功发起 CreateCloudRecording 后，在确保房间有上行音视频的情况下，可以按照录制文件名的生成规则来拼接录制文件名称。具体文件名规则请参见 录制文件名命名规则。
  - 录制文件的状态会通过回调发送到客户的服务器，如果订阅了相关回调，将会收到录制文件的状态信息。具体回调信息请参见 回调接口 。
- 录制用户（userid）的 UserSig 过期时间应该设置成比录制任务生命周期更长的时间，防止录制任务机器断网，在内部高可用生效的时候，恢复录制因为 UserSig 过期而失败。

同时发起云端录制与转推

最近更新时间：2024-11-12 11:58:52

功能说明

针对在同一房间内同时需要录制和转推主播的音视频流的情况，TRTC 推出了一项全新解决方案。该方案支持通过一次接口调用，将房间内主播的音视频流同时进行录制存储和转推 CDN。另外对于需要混流录制和混流转推 CDN 的场景，用户只需完成一次混流任务，与分别发起混流录制和混流转推功能相比，只需一次接口调用并可节省一次混流的成本。



录制文件命名和文件切分说明

录制 MP4/AAC 文件名命名规则

- **单流录制 MP4/AAC 文件名规则：** `<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Index>.mp4/aac`
- **混流录制 MP4/AAC 文件名规则：** `<SdkAppId>_<RoomId>_<Index>.mp4/aac`

录制 HLS 文件名命名规则

- **单流录制 HLS 文件名规则：** `<SdkAppId>_<RoomId>_UserId_s_<UserId>_UserId_e_<MediaId>_<Type>.m3u8`
- **混流录制 HLS 文件名规则：** `<SdkAppId>_<RoomId>.m3u8`
- **字段含义说明：**

字段	含义
<SdkAppId>	录制任务的 SdkAppId
<RoomId>	录制的房间号，如果这里 RoomId 如果是字符串房间号，我们会对房间号先做 base64 操作，再把 base64 后的字符串中符号'/'替换成 '-'（中划线），符号'='替换成 '.'
<UserId>	录制的用户 ID，UserId 会先做 base64 操作，再把base64后的字符串中符号'/'替换成 '-'（中划线），符号'='替换成 '.'
<MediaId>	主辅流标识，main 代表主流（摄像头），aux 代表辅流（屏幕分享）
<Index>	如果没有触发切片逻辑（大小超过2GB或超过设置的切片时长）则无该字段，否则为切片的索引号，从1开始递增
<Type>	录制文件流类型，audio/video/audiovideo

说明：
自定义设置文件名称前缀：使用 API 录制 存储至云点播 VOD 时，可通过 TencentVod 中的 UserDefineRecordId 参数自定义文件名称前缀，前缀与默认录制文件名之间用 "__UserDefine_u_" 分隔。

录制文件切分说明

- **录制 MP4/ACC 文件切分的条件：**
  - 录制切分时长可设置范围1 – 1440分钟，默认1440分钟。
  - 单个 MP4/AAC 文件大小达到2GB。
- **录制 HLS 文件切分的条件：**
  - 录制任务持续时间超过14天时，HLS 文件将会被切分。

录制上传云存储说明

录制后台会在录制结束后将录制的文件通过您指定的方式上传到云存储平台（云点播 VOD 或对象存储 COS），并通过回调的形式把播放地址发送给您。
如果录制模式为单流录制模式，每一个订阅的主播都会有一个对应的播放地址；如果发起混流录制，只有一个混合后媒体的播放地址。
通过 API 发起录制：在 McuRecordParams 中必须指定 McuStorageParams 的参数（云点播 VOD 或对象存储 COS），请确保已经开通对应的云存储服务且未欠费。

> ⚠ **注意：**
> 文件录制后会上传至您指定云存储平台（云点播 VOD 或对象存储 COS），为确保录制文件成功，请确保您指定的云点播 VOD 或对象存储 COS 服务可用。

API 接口和录制并发限制

- 接口的调用频率限制为20qps（如需提高 QPS 请 提交工单 ）。
- 单个接口超时时间为6秒。
- 单个应用下**默认并发录制支持500路（与 云端录制 共用）**，超过并发限制的任务会失败，如需更多并发路数，请 提交工单 联系我们。
- 单次录制任务最大支持同时订阅的房间内主播数为25个，主播只上行音频也会单独占据一路。

使用说明

目前支持通过 Restful API 接口：**启动转推任务** StartPublishCdnStream 同时设置录制相关参数（RecordParams）和转推相关参数（PublishCdnParams.N）即可同时发起录制与转推功能。此接口也可单独发起录制或转推功能。
针对录制参数的设置，提供以下两种设置方式：

录制参数设置方式	说明
Restful API 参数指定	如需灵活调整录制参数，可通过 RecordParams 来设置录制参数，此时 **RecordParams -> UniRecord 需设置为3。**
读取控制台录制模板	如录制参数相对固定，可通过在 控制台 > 应用管理 > 录制管理 下的配置录制模板，根据模板中的录制参数来发起录制，此时 **RecordParams -> UniRecord 需设置为2。** ⚠ **注意：** 通过控制台获取的录制参数时，仅使用您配置的单流录制参数，控制台录制模板中的混流参数不生效，如果发起混流，请通过 StartPublishCdnStream ->AudioParams 和 VideoParams 进行指定。

说明：

- 发起录制任务接口中需要您指定分配录制机器人的进房参数 UserId 和 UserSig（如何获取 UserSig），请不要与您房间内的正常主播或观众使用的 UserId 重复且不可与正在录制中的房间内指定的录制机器人 UserId 一致，否则会导致录制任务失败。
- **手动录制下，您可以前往控制台配置回调地址，以接受录制回调事件**，请见 录制回调说明 。

事件回调

我们针对云端录制和转推功能功能提供了多种的回调事件，帮助您及时了解任务的处理和完成情况，录制回调地址配置和事件说明请见 云端录制回调 和 旁路转推回调 。

录制文件管理

结束录制任务后，TRTC 录制系统中录制下来的文件上传至您指定的云存储平台（云点播 VOD 或对象存储 COS），详细说明请见 录制文件管理 。

相关费用

同时发起转推和录制功能，会产生 TRTC 云端录制费用 、TRTC 转推费用 、根据您指定存储云存储不同（云点播 VOD 或对象存储 COS）会由存储方收取回放文件存储与处理、播放费用。

实践教程

录制场景

根据录制需求场景的不同，我们提供以下实践说明：
场景一：在转推、媒体处理（混流、转码，加水印等）同时将对应结果同时进行录制下来。
场景二：续录多条流，即将不同时刻的多个主播画面录制到同一个文件中，实现续录。

场景一：录制 + 转推

当您需要对主播流进行转推，或者需要对上行流进行处理，同时有录制的需求时，可以使用转推录制接口，利用转推接口的流处理能力，对流进行灵活处理后进行录制。这里以云 API 为例介绍的转推录制使用方式（后续会支持终端 SDK 接口发起方式）。

1. “录制&转推&混流&水印”任务同时发起。

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: StartPublishCdnStream
<公共请求参数>
```

```
{
    "SdkAppId": ********,
    "RoomId": "350",
    "RoomIdType": 0,
    "AgentParams": {
        "UserId": "trtc_partner_test_1",
        "UserSig": "********",
        "MaxIdleTime": 10
    },
    "WithTranscoding": 1,
    "AudioParams": {
        "AudioEncode": {
            "Codec": 0,
            "SampleRate": 48000,
            "Channel": 2,
            "BitRate": 64
        },
        "SubscribeAudioList": [
            {
                "UserInfo": {
                    "RoomId": "350",
                    "RoomIdType": 0,
                    "UserId": "Trtc_User_1"
                }
            },
            {
                "UserInfo": {
                    "RoomId": "350",
                    "RoomIdType": 0,
                    "UserId": "Trtc_User_0"
                }
            }
        ]
    },
    "VideoParams": {
        "VideoEncode": {
            "Width": 1280,
            "Height": 720,
            "Fps": 15,
            "BitRate": 512,
            "Gop": 2
        },
        "LayoutParams": {
            "PureAudioHoldPlaceMode": 0,
            "MixLayoutMode": 1
        },
        "BackGroundColor": "0xFF0000",
        "WaterMarkList": [
            {
                "WaterMarkType": 0,
                "WaterMarkImage": {
                    "LocationX": 64,
                    "LocationY": 64,
                    "WaterMarkHeight": 64,
                    "WaterMarkWidth": 128,
                    "WaterMarkUrl": "https://xxxxxxxx.png",
                    "ZOrder": 3
                }
```

```
            }
        ]
    },
    "RecordParams": {
        "UniRecord": 3,
        "RecordFormat": ["mp4"],
        "StreamType": 0,
        "McuStorageParams": {
            "McuCloudVod": {
                "McuTencentVod": {
                    "ExpireTime": 86400
                }
            }
        }
    },
    "PublishCdnParams": [
        {
            "IsTencentCdn": 1,
            "PublishCdnUrl": "rtmp://xxxxxx"
        }
    ]
}
```

2. 使用 UpdatePublishCdnStream 将其切换为单流录制 + 转推任务。

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: UpdatePublishCdnStream
<公共请求参数>

{
    "SdkAppId": ********,
    "TaskId": "D0xMnLdRsmIO5+E09Y5wpUh+Q556USkfaMtHzbFuHc19muIw84b4EN1Bc0stJtznupVpRKeyjc-
0nDjD8V+HfU8A",
    "SequenceNumber": 5,
    "WithTranscoding": 0,
    "SingleSubscribeParams": {
        "UserMediaStream": {
            "UserInfo": {
                "RoomId": "350",
                "RoomIdType": 0,
                "UserId": "Trtc_User_0"
            },
            "StreamType": 0
        }
    }
}
```

3. 结束转推录制任务。

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: StopPublishCdnStream
```

```
<公共请求参数>

{
    "SdkAppId": ********,
    "TaskId": "D0xMnLdRsmIO5+E09Y5wpUh+Q556USkfaMtHzbFuHc19muIw84b4EN1Bc0stJtznupVpRKeyjc-
0nDjD8V+HfU8A"
}
```

场景二：续录多条流

为了实现多个任务的续录，这些任务发起时需要填写相同的 recordkey，并且前一个任务结束后，后一个任务要在指定的续录时间（通过控制台录制模板和Restful API 参数 RecordWaitTime 指定）内发起。

> **注意：**
> 若不需实现续录多条流，请勿将在发起任务时填写相同的 recordkey。

1. 首先发起录制任务 Task1。

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: StartPublishCdnStream
<公共请求参数>

{
    "SdkAppId": ********,
    "RoomId": "350",
    "RoomIdType": 0,
    "AgentParams": {
        "UserId": "trtc_partner_test_1",
        "UserSig": "*********",
        "MaxIdleTime": 10
    },
    "WithTranscoding": 1,
    "AudioParams": {
        "AudioEncode": {
            "Codec": 0,
            "SampleRate": 48000,
            "Channel": 2,
            "BitRate": 64
        },
        "SubscribeAudioList": [
            {
                "UserInfo": {
                    "RoomId": "350",
                    "RoomIdType": 0,
                    "UserId": "Trtc_User_1"
                }
            },
            {
                "UserInfo": {
                    "RoomId": "350",
                    "RoomIdType": 0,
                    "UserId": "Trtc_User_0"
                }
            }
```

```
        ]
    },
    "VideoParams": {
        "VideoEncode": {
            "Width": 1280,
            "Height": 720,
            "Fps": 15,
            "BitRate": 512,
            "Gop": 2
        },
        "LayoutParams": {
            "PureAudioHoldPlaceMode": 0,
            "MixLayoutMode": 1
        }
    },
    "RecordParams": {
        "UniRecord": 3,
        "RecordFormat": ["mp4"],
        "StreamType": 0,
        "RecordKey": "recordkey2403122301",
        "McuStorageParams": {
            "McuCloudVod": {
                "McuTencentVod": {
                    "ExpireTime": 86400
                }
            }
        }
    },
    "PublishCdnParams": [
        {
            "IsTencentCdn": 1,
            "PublishCdnUrl": "rtmp:xxxxxxx"
        }
    ]
}
```

2. 结束录制任务 Task1，结束时不指定 recordkey。

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: StopPublishCdnStream
<公共请求参数>

{
    "SdkAppId": ********,
    "TaskId": "D0xMgKlRssqyrNUNqIPkpuwPuC1GLuIfSZN4zbFuHc19muIw84b4EN1Bc0stJtznufPhuidu6JHPSpuljN"
}
```

3. 使用同样的 recordkey 发起续录任务 Task2。

> **说明：**
> RecordParams 中的参数以第一次任务为准，后续续录任务指定的 RecordParams 参数不生效，所以录制文件的前缀还是"prefix_test"，而不是“prefix_changed”

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: StartPublishCdnStream
<公共请求参数>

{
    "SdkAppId": ********,
    "RoomId": "350",
    "RoomIdType": 0,
    "AgentParams": {
        "UserId": "trtc_partner_test_1",
        "UserSig": "*********",
        "MaxIdleTime": 10
    },
    "WithTranscoding": 0,
    "AudioParams": {
        "AudioEncode": {
            "Codec": 0,
            "SampleRate": 48000,
            "Channel": 2,
            "BitRate": 64
        }
    },
    "VideoParams": {
        "VideoEncode": {
            "Width": 1280,
            "Height": 720,
            "Fps": 15,
            "BitRate": 512,
            "Gop": 2
        }
    },
    "SingleSubscribeParams": {
        "UserMediaStream": {
            "UserInfo": {
                "RoomId": "350",
                "RoomIdType": 0,
                "UserId": "Trtc_User_0"
            },
            "StreamType": 0
        }
    },
    "RecordParams": {
        "RecordKey": "recordkey2403122301",
        "UniRecord": 3,
        "RecordFormat": ["mp4"],
        "StreamType": 0,
        "McuStorageParams": {
            "McuCloudVod": {
                "McuTencentVod": {
                    "ExpireTime": 86400
                }
            }
        }
    },
    "PublishCdnParams": [
```

```
        {
            "IsTencentCdn": 1,
            "PublishCdnUrl": "rtmp://xxxxxxx"
        }
    ]
}
```

4. 指定 recordkey，停止录制任务 Task2。

```
POST / HTTP/1.1
Host: trtc.tencentcloudapi.com
Content-Type: application/json
X-TC-Action: StopPublishCdnStream
<公共请求参数>

{
    "SdkAppId": ********,
    "RecordKey": "recordkey2403122301",
    "TaskId": "D0zNfGlRsohMM0sTZo5hlGZD4gkBQp4fCZh4zbFuHc19muIw84b4EN1Bc0stJtznuZDTgfDCsYnCozJw"
}
```

说明：

1. 带 recordkey A，发起任务 A；带 recordkey A，发起任务 B，录制会从录任务 A 切换到录任务 B，虽然任务 A 没退出，但不会再录制。
2. 转推录制的参数以第一次任务发起为准，后续续录任务指定的录制参数（RecordParams 中的参数）不生效。
3. 如果转推录制发起时指定了 recordkey，结束转推录制分几种情况：
   3.1 结束转推录制指定同样的 recordkey：
   - 如果此时 recordkey 对应的录制任务正在录制当前流，则转推和录制都立即结束，续录时间不生效。
   - 如果此时 recordkey 对应的录制任务没有录制当前流，则转推立即结束，录制任务不受影响继续进行。如果想要结束录制，需要通过结束录制当前正在录制的转推任务实现。

   补充说明：例如使用 recordkey A 先发起转推任务 1，然后用同样的 recordkey A 发起了转推任务 2，此时 recordkey A 对应的录制任务会切换到转推任务 2进行录制；当停止转推任务 1 时 即使填写了 recordkey A，但是因为当前 recordkey A 已经不再录制 转推任务 1，因此无法将录制任务 A 结束；只有在结束 转推任务 2 时才能调用结束任务接口来结束录制任务 A。
   3.2 结束转推录制不指定 recordkey：则当前转推结束，录制任务等待续录时间后结束；续录时间内使用同样的 recordkey 发起转推录制任务，则两次转推续录到一个文件中。

监听服务端事件回调
房间与媒体回调

最近更新时间：2025-01-06 17:22:42

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），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}。

注意：

- 您的 HTTP 应答包长（header+body）需要控制在2000个字节以内。
- POST 请求的 JSON 包体不会删减已有字段，但会根据业务需求添加新字段。在集成回调时，您需要适应这些新增字段的情况。

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

> **说明：**
> 录制事件组相关说明请参见 实现云端录制与回放 。

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
EVENT_TYPE_STOP_ASSIT	206	停止推送辅路数据

注意：
退出房间只会回调104事件，不会回调202跟204事件。104事件相当于包含了202和204事件。手动关闭视频/音频，才会回调202/204事件。

事件回调示例

101

```
{    "EventGroupId": 1,
     "EventType":    101,
     "CallbackTs":   1687770730166,
     "EventInfo":    {
         "RoomId":   12345,
         "EventTs":  1687770730,
         "EventMsTs":    1687770730160,
         "UserId":   "test"
         }
}
```

102

```
{
     "EventGroupId": 1,
     "EventType":    102,
     "CallbackTs":   1687771618531,
     "EventInfo":    {
         "RoomId":   "12345",
         "EventTs":  1687771618,
         "EventMsTs":    1687771618457
      }
}
```

103

```
{
     "EventGroupId": 1,
     "EventType":    103,
     "CallbackTs":   1687770731932,
     "EventInfo":    {
         "RoomId":   12345,
         "EventTs":  1687770731,
         "EventMsTs":    1687770731831,
         "UserId":   "test",
         "Role": 21,
         "TerminalType": 2,
         "UserType": 3,
         "Reason":   1
     }
}
```

104

```
{
    "EventGroupId": 1,
    "EventType":    104,
    "CallbackTs":   1687770731922,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687770731,
        "EventMsTs":    1687770731898,
        "UserId":   "test",
        "Role": 20,
        "Reason":   1
    }
}
```

105

```
{
    "EventGroupId": 1,
    "EventType":    105,
    "CallbackTs":   1687772245596,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687772245,
        "EventMsTs":    1687772245537,
        "UserId":   "test",
        "Role": 21
    }
}
```

201

```
{
    "EventGroupId": 2,
    "EventType":    201,
    "CallbackTs":   1687771803198,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687771803,
        "EventMsTs":    1687771803192,
        "UserId":   "test"
    }
}
```

202

```
{
    "EventGroupId": 2,
    "EventType":    202,
    "CallbackTs":   1687771919458,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687771919,
```

```
        "EventMsTs":    1687771919447,
        "UserId":   "test",
        "Reason":   0
    }
}
```

203

```
{
    "EventGroupId": 2,
    "EventType":    203,
    "CallbackTs":   1687771869377,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687771869,
        "EventMsTs":    1687771869365,
        "UserId":   "test"
    }
}
```

204

```
{
    "EventGroupId": 2,
    "EventType":    204,
    "CallbackTs":   1687770732498,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687770732,
        "EventMsTs":    1687770732383,
        "UserId":   "test",
        "Reason":   0
    }
}
```

205

```
{
    "EventGroupId": 2,
    "EventType":    205,
    "CallbackTs":   1687772013823,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687772013,
        "EventMsTs":    1687772013753,
        "UserId":   "test"
    }
}
```

206

```
{
    "EventGroupId": 2,
    "EventType":    206,
```

```
    "CallbackTs":   1687772015054,
    "EventInfo":    {
        "RoomId":   12345,
        "EventTs":  1687772015,
        "EventMsTs":    1687772015032,
        "UserId":   "test",
        "Reason":   0
    }
}
```

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、停止媒体流时携带）
ClientIpv4	String	客户端 Ipv4 地址（option：使用Ipv4进房时，103事件携带）
ClientIpv6	String	客户端 Ipv6 地址（option：使用Ipv6进房时，103事件携带）

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制终止，只能等待后台超时离开，此时回调 reason 为2**
停止媒体流	0：正常停止 1：超时停止 **注意：后台连续30秒(默认)没有收到媒体流，将回调停止媒体流事件，并携带 Reason 为1，表示超时停止。**

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\"Even
tInfo\":\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}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
```

```java
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
//# 返回值：
//#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#   Info：成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```python
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
# 参数：
#   key：控制台配置的密钥key
#   body：腾讯云回调返回的body体
#   sign：腾讯云回调返回的签名值sign
# 返回值：
#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'),
digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
```

```
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}"
    var key = "789"

    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

旁路转推回调

最近更新时间：2023-07-31 17:32:42

服务端转推回调支持将您使用旁路转推REST API 产生转推 CDN 的事件，以 HTTP/HTTPS 请求的形式通知到您的服务器。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“转推时间组-CDN 推流正在进行”事件的包体示例。

```
{
    "EventGroupId": 4,
    "EventType": 401,
    "CallbackTs": 1622186275913,
    "EventInfo": {
        "RoomId": "xx",
        "RoomType": 1,
        "EventTsMs": 1622186275913,
        "UserId": "xx",
        "TaskId": "xx",
        "Payload": {
            "Url": "rtmp://tencent-url/xxxx"
            "Status": 2                          /表示该转推任务正在向腾讯云CDN推流/
        }
    }
}
```

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，混流转推事件固定为4
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_CLOUD_PUBLISH	4	转推事件组

事件类型

字段名	值	含义
EVENT_TYPE_CLOUD_PUBLISH_CDN_STATUS	401	云端转推 CDN 状态回调

事件信息

字段名	类型	含义
RoomId	String	房间名（类型与客户端房间号类型一致）
RoomType	Number	0表示数字房间号，1表示字符串房间号
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
UserId	String	发起任务时指定的伴生机器人的用户 ID（AgentParams.UserId）
TaskId	String	任务 ID
Payload	JSON Object	事件的详细信息

Payload（详细信息）

字段名	值	含义
Url	String	推流的目的 URL 地址
Status	Number	转推状态
ErrorCode	Number	错误码
ErrorMsg	String	错误信息

转推状态

字段名	值	含义	回调频率
PUBLISH_CDN_STREAM_STATE_IDLE	0	推流未开始或已结束	仅回调1次

PUBLISH_CDN_STREAM_STATE_CONNECTING	1	正在连接 TRTC 服务器和 CDN 服务器	每5秒回调1次，60秒超时后不再回调
PUBLISH_CDN_STREAM_STATE_RUNNING	2	CDN 推流正在进行	仅回调1次
PUBLISH_CDN_STREAM_STATE_RECOVERING	3	TRTC 服务器和 CDN 服务器推流中断，正在恢复	每5秒回调1次，60秒超时后不再回调
PUBLISH_CDN_STREAM_STATE_FAILURE	4	TRTC 服务器和 CDN 服务器推流中断，且恢复或连接超时	仅回调1次
PUBLISH_CDN_STREAM_STATE_DISCONNECTING	5	正在断开 TRTC 服务器和 CDN 服务器	仅回调1次

转推状态推荐处理

状态	处理方法
PUBLISH_CDN_STREAM_STATE_IDLE	表示 URL 移除成功，无需处理。
PUBLISH_CDN_STREAM_STATE_CONNECTING	• 表示 URL 正在连接中，每隔5s回调一次，直到连接成功回调 `PUBLISH_CDN_STREAM_STATE_RUNNING`，或者60s后回调 `PUBLISH_CDN_STREAM_STATE_FAILURE` 。您可以在收到 `PUBLISH_CDN_STREAM_STATE_FAILURE` 的时候替换有问题的 URL，调用 `UpdatePublishCdnStream` 更新 Publish 参数。 • 如果您的业务对时间比较敏感，可以在收到2个或以上的 `PUBLISH_CDN_STREAM_STATE_CONNECTING` 回调后，替换有问题的 URL，调用 `UpdatePublishCdnStream` 更新 Publish 参数。
PUBLISH_CDN_STREAM_STATE_RUNNING	表示 URL 推流成功，无需处理。
PUBLISH_CDN_STREAM_STATE_RECOVERING	• 表示推流过程发生了中断，正在重连中，每隔5s回调一次，直到重连成功回调 `PUBLISH_CDN_STREAM_STATE_RUNNING`，或者60s后回调 `PUBLISH_CDN_STREAM_STATE_FAILURE` 。通常为网络抖动，无需处理。 • 如果 `PUBLISH_CDN_STREAM_STATE_RECOVERING` 和 `PUBLISH_CDN_STREAM_STATE_RUNNING` 短时间内交替出现，您需要检查下是否存在多任务使用相同的推流 URL 。
PUBLISH_CDN_STREAM_STATE_FAILURE	表示推流 URL ，在60s内建连失败或者恢复推流失败，此时您可以替换有问题的 URL ，调用 `UpdatePublishCdnStream` 更新 Publish 参数。
PUBLISH_CDN_STREAM_STATE_DISCONNECTING	表示，正在移除推流 URL ，移除成功后，会回调PUBLISH_CDN_STREAM_STATE_IDLE，无需处理。

基本回调转移示例

- **发起转推/新增转推地址到转推成功的事件转移**
  `PUBLISH_CDN_STREAM_STATE_CONNECTING` -> `PUBLISH_CDN_STREAM_STATE_RUNNING`
- **停止转推/删除转推地址到停止转推成功的事件转移**
  `PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_DISCONNECTING` -> `PUBLISH_CDN_STREAM_STATE_IDLE`
- **转推过程中，链接失败到重试链接成功的事件转移**
  `PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_RECOVERING` -> `PUBLISH_CDN_STREAM_STATE_RUNNING`
- **转推过程中，链接失败到重试链接超时失败的事件转移**
  `PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_RECOVERING` -> `PUBLISH_CDN_STREAM_STATE_FAILURE` -> `PUBLISH_CDN_STREAM_STATE_IDLE`

⚠ **注意：**

推流回调有可能会乱序到达您的回调服务器，此时您需要根据 EventInfo 中的 EventMsTs 做事件排序，如果您只关心 URL 最新状态，可以忽略后续到达的过期事件。

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\"Even
tInfo\":\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}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
//# 返回值：
//#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#   Info：成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
```

```
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
# 参数：
#   key：控制台配置的密钥key
#   body：腾讯云回调返回的body体
#   sign：腾讯云回调返回的签名值sign
# 返回值：
#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'),
digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
```

```
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

云端录制和页面录制回调

最近更新时间：2024-10-12 16:42:51

本文针对 新版云端录制功能 和 页面录制功能 的回调事件进行具体说明，旧版云端录制的SdkAppId配置回调录制请见 旧版录制回调 。

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并在控制台完成 回调配置 。

- **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。

回调接口

您可以提供一个接收回调的 HTTP/HTTPS 服务网关来订阅回调消息。当相关事件发生时，云录制系统会回调事件通知到您的消息接收服务器。

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}。

云端录制参数说明

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，云端录制固定为3，页面录制固定为8
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

事件类型说明：

字段名	类型	含义
EVENT_TYPE_CLOUD_RECORDING_RECORDER_START	301	云端录制模块启动
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	云端录制模块退出
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	云端录制文件上传任务启动，仅在选择对象存储时回调
EVENT_TYPE_CLOUD_RECORDING_FILE_INFO	304	云端录制 生成 m3u8 索引文件，第一次生成并且上传成功后回调，仅在通过 API 录制 选择对象存储时回调
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	云端录制文件上传结束，仅在通过 API 录制 选择对象存储时回调
EVENT_TYPE_CLOUD_RECORDING_FAILOVER	306	云端录制发生迁移，原有的录制任务被迁移到新负载上时触发
EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE	307	云端录制 生成 m3u8 索引文件（切出第一个 ts 切片）生成后回调，仅在通过 API 录制 选择对象存储时回调
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	云端录制下载解码图片文件发生错误
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	云端录制 MP4 录制任务结束，仅在通过 API录制 选择对象存储时回调(控制台开启自动录制，选择授权给点播的 cos 作为存储时，请关注311事件)
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	云端录制 VOD 录制任务上传媒体资源完成，在选择**云点播**时和通过控制台**自动录制存储至 cos** 时回调（录制文件结束后携带点播索引信息，请订阅此类型回调事件）
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	云端录制 VOD 录制任务结束，仅在选择云点播时回调

> ⚠ **注意：**
> 301 – 309区间的回调状态为实时录制的中间状态，可以更加清晰的知晓录制任务的进行过程并记录状态，实际录制文件上传到点播成功会回调311事件，整体任务结束回调312事件。

事件信息说明：

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 (不建议使用该字段，建议使用EventMsTs)
EventMsTs	Number	事件发生的 Unix 时间戳，单位为毫秒
UserId	String	录制机器人的用户 ID

TaskId	String	录制 ID，一次云端录制任务唯一的 ID
Payload	JsonObject	根据不同事件类型定义不同

- **事件类型为301**（EVENT_TYPE_CLOUD_RECORDING_RECORDER_START）时 Payload 的定义：

字段名	类型	含义
Status	Number	0：代表录制模块启动成功 1：代表录制模块启动失败

```
{
    "EventGroupId": 3,
    "EventType": 301,
    "CallbackTs": 1622186275913,
    "EventInfo": {
        "RoomId": "xx",
        "EventTs": "1622186275",
        "EventMsTs": 1622186275757,
        "UserId": "xx",
        "TaskId": "xx",
        "Payload": {
            "Status": 0
        }
    }
}
```

- **事件类型为302**（EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP）时 Payload 的定义：

字段名	类型	含义
LeaveCode	Number	0：代表录制模块正常调用停止录制退出 1：录制机器人被客户踢出房间 2：客户解散房间 3：服务器将录制机器人踢出 4：服务器解散房间 99：代表房间内除了录制机器人没有其他用户流，超过指定时间退出 100：房间超时退出 101：同一用户重复进入相同房间导致机器人退出

```
{
  "EventGroupId": 3,
  "EventType": 302,
  "CallbackTs": 1622186354806,
  "EventInfo": {
    "RoomId": "xx",
    "EventTs": "1622186354",
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "LeaveCode": 0
    }
  }
}
```

- **事件类型为303**（EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START）时 Payload 的定义：

字段名	类型	含义
Status	Number	0：代表上传模块正常启动 1：代表上传模块初始化失败。

- **事件类型为304**（EVENT_TYPE_CLOUD_RECORDING_FILE_INFO）时 Payload 的定义：

字段名	类型	含义
FileList	String	生成的 M3U8 文件名

- **事件类型为305**（EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP）时 Payload 的定义：

字段名	类型	含义
Status	Number	0：代表此次录制上传任务已经完成，所有的文件均已上传到指定的第三方云存储 1：代表此次录制上传任务已经完成，但至少有一片文件滞留在服务器或者备份存储上 2：代表滞留在服务器或者备份存储上的文件已经恢复上传到指定的第三方云存储 注意：305代表 hls 文件上传结束事件

- **事件类型为306**（EVENT_TYPE_CLOUD_RECORDING_FAILOVER）时 Payload 的定义：

字段名	类型	含义
Status	Number	0：代表此次迁移已经完成

```
{
  "EventGroupId": 3,
  "EventType": 306,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

- **事件类型为307**（EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE）时 Payload 的定义：

字段名	类型	含义
FileName	String	m3u8 文件名
UserId	String	录制文件对应的用户 ID
TrackType	String	音视频类型 audio/video/audio_video
BeginTimeStamp	String	录制开始时，服务器Unix时间戳(毫秒)

- **事件类型为309**（EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR）时 Payload 的定义：

字段名	类型	含义
Url	String	下载失败的 URL

```
{
  "EventGroupId": 3,
  "EventType": 309,
  "CallbackTs": 1622191989674,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Url": "http://xx"
    }
  }
}
```

- **事件类型为310**（EVENT_TYPE_CLOUD_RECORDING_MP4_STOP）时 Payload 的定义：

> **说明：**
> 310 是上传mp4文件到客户指定第三方云存储COS完成后的回调事件，一个任务对应一个310事件（一个事件中对应本次任务中所有的录制文件信息）

段名	类型	含义
Status	Number	0：代表此次录制 mp4 任务已经正常退出，所有的文件均已上传到指定的第三方云存储 1：代表此次录制 mp4 任务已经正常退出，但至少有一片文件滞留在服务 器或者备份存储上 2：代表此次录制 mp4 任务异常退出(可能原因是拉取 cos 的 hls 文件失败)
FileList	Array	所有生成的 mp4 文件名
FileMessage	Array	所有生成的 mp4 文件信息
FileName	String	mp4 文件名
UserId	String	mp4 文件对应的用户 ID（当录制模式为混流模式时，此字段为空）
TrackType	String	audio 音频 / video 纯视频 / audio_video 音视频
MediaId	String	主辅流标识，main 代表主流（摄像头），aux 代表辅流（屏幕分享），mix 代表混流录制
StartTimeStamp	Number	mp4 文件开始的 Unix 时间戳(毫秒)
EndTimeStamp	Number	mp4 文件结束的 Unix 时间戳(毫秒)

```
{
  "EventGroupId": 3,
  "EventType": 310,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191989,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
```

```
    "Payload": {
      "Status": 0,
      "FileList": ["xxxx1.mp4", "xxxx2.mp4"],
      "FileMessage": [
        {
            "FileName": "xxxx1.mp4",
            "UserId": "xxxx",
            "TrackType": "audio_video",
            "MediaId": "main",
            "StartTimeStamp": 1622186279145,
            "EndTimeStamp": 1622186282145
        },
        {
            "FileName": "xxxx2.mp4",
            "UserId": "xxxx",
            "TrackType": "audio_video",
            "MediaId": "main",
            "StartTimeStamp": 1622186279153,
            "EndTimeStamp": 1622186282153
        }
     ]

    }
  }
}
```

- **事件类型为311**（EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT）时 Payload 的定义：

字段名	类型	含义
Status	Number	0：代表本录制文件正常上传至点播平台 1：代表本录制文件滞留在服务器或者备份存储上 2：代表本录制文件上传点播任务异常
UserId	String	本录制文件对应的用户 ID（当录制模式为合流模式时，此字段为空）
TrackType	String	audio 音频 / video 纯视频 / audio_video 音视频
MediaId	String	主辅流标识，main 代表主流（摄像头），aux 代表辅流（屏幕分享），mix 代表混流录制
FileId	String	本录制文件在点播平台的唯一 ID
VideoUrl	String	本录制文件在点播平台的播放地址
CacheFile	String	本录制文件对应的 MP4/HLS 文件名
StartTimeStamp	Number	本录制文件开始的 UNIX 时间戳（毫秒）
EndTimeStamp	Number	本录制文件结束的 UNIX 时间戳（毫秒）
Errmsg	String	status 不为0时，对应的错误信息

上传成功的回调：

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
  "EventInfo": {
```

```
    "RoomId": "20015",
    "EventTs": 1622191965,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "TencentVod": {
        "UserId": "xx",
        "TrackType": "audio_video",
        "MediaId": "main",
        "FileId": "xxxx",
        "VideoUrl": "http://xxxx",
        "CacheFile": "xxxx.mp4",
        "StartTimeStamp": 1622186279153,
        "EndTimeStamp": 1622186282153
      }
    }
  }
}
```

上传失败的回调：

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 1,
      "Errmsg": "xxx",
      "TencentVod": {
        "UserId": "123",
        "TrackType": "audio_video",
        "CacheFile": "xxx.mp4"
      }
    }
  }
}
```

> **说明：**
> 录制完成收到311回调后到**文件上传完成**，根据您本次录制文件的大小不同可能需等待30s – 3min。

- **事件类型为312**（EVENT_TYPE_CLOUD_RECORDING_VOD_STOP）时 Payload 的定义：

字段名	类型	含义
Status	Number	0：代表本次上传 VOD 任务已经正常退出 1：代表本次上传 VOD 任务异常退出

```
{
```

```
  "EventGroupId": 3,
  "EventType": 312,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0
    }
  }
}
```

页面录制参数说明

页面录制事件类型：

字段名	类型	含义
EVENT_TYPE_WEB_RECORDER_START	801	页面录制模块启动
EVENT_TYPE_WEB_RECORDER_STOP	802	页面录制模块退出
EVENT_TYPE_WEB_RECORDER_STATUS_UPDATE	803	页面录制模块录制状态更新
EVENT_TYPE_WEB_RECORDER_RESOURCE_LIMIT	804	页面录制任务资源受限，结束录制任务，近在录制时长超限、录制分辨率超限时回调

页面录制事件信息说明：

字段名	类型	含义
EventMsTs	Number	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	录制 ID，一次页面录制任务唯一的 ID
Payload	JsonObject	根据不同事件类型定义不同
EventMessage	String	对应Status的事件信息描述

- **事件类型为801（EVENT_TYPE_WEB_RECORDER_START）时 Payload 的定义：**

字段名	类型	含义
Status	Number	1：代表页面录制模块启动成功 2：代表页面录制模块启动失败 3：代表录制过程中异常退出 4：代表录制任务已迁移 5：代表录制任务异常，停止录制
EventMessage	String	Success：页面录制模块启动成功，开始录制 StartRecording error：页面录制模块启动失败 Goto url timeout：录制页面访问超时 Exception error, exit task：录制异常结束 Chrome exception, exit task：Chrome异常退出，录制结束 Recording tasks have been migrated：录制任务已迁移 Page load timeout：页面加载超时

```
{
```

```
  "EventGroupId": 8,
  "EventType": 801,
  "CallbackTs": 1622186275913,
  "EventInfo": {
    "EventMsTs": 1622186275757,
    "TaskId": "xx",
    "Payload": {
      "Status": 1,
      "EventMessage": "Success"
    }
  }
}
```

说明：

- 收到状态码为2的801回调时，请检查 RecordUrl 是否可以正常访问。
- 801回调的状态码3和状态码4，为页面录制的中间状态，便于录制任务出现异常时的排查；页面录制会自动对上述两种状态进行处理，客户无需进行处理。

注意：

收到状态码为5的801回调时，录制任务将被终止，并不再重试，请您及时联系业务人员进行排查。

- **事件类型为802**（EVENT_TYPE_WEB_RECORDER_STOP）时 Payload 的定义：

字段名	类型	含义
Status	Number	1：代表本次页面录制任务正常结束
EventMessage	String	Success：页面录制模块正常调用停止录制退出

```
{
  "EventGroupId": 8,
  "EventType": 802,
  "CallbackTs": 1622186275913,
  "EventInfo": {
    "EventMsTs": 1622186275757,
    "TaskId": "xx",
    "Payload": {
      "Status": 1,
      "EventMessage": "Success"
    }
  }
}
```

说明：

收到802回调说明页面录制任务已完成，实际录制文件上传完成会回调311事件，整体完成回调312事件。

- **事件类型为803**（EVENT_TYPE_WEB_RECORDER_STATUS_UPDATE）时 Payload 的定义：

字段名	类型	含义
Status	Number	1：代表页面录制任务页面刷新 2：代表页面录制任务暂停录制 3：代表页面录制任务恢复录制

EventMessage	String	PageRefresh：刷新录制页面 RecordPaused：录制任务暂停 RecordResume：录制任务恢复 Page load timeout：页面加载超时

```
{
  "EventGroupId": 8,
  "EventType": 803,
  "CallbackTs": 1622186275913,
  "EventInfo": {
    "EventMsTs": 1622186275757,
    "TaskId": "xx",
    "Payload": {
      "Status": 1,
      "EventMessage": "PageRefresh"
    }
  }
}
```

- **事件类型为804**（EVENT_TYPE_WEB_RECORDER_RESOURCE_LIMIT）时 Status 的定义：

字段名	类型	含义
Status	Number	1：代表本次页面录制任务达到设定的最大录制时间，结束录制 2：代表本次页面录制任务的录制页面中出现了超过设定的最大录制分辨率的视频流，结束录制
EventMessage	String	Over time limit：录制时长达到设定的最大录制时长，自动结束录制 Over resolution limit：录制任务中出现超过最大分辨率限制的视频源，自动结束录制

```
{
  "EventGroupId": 8,
  "EventType": 804,
  "CallbackTs": 1622186275913,
  "EventInfo": {
    "EventMsTs": 1622186275757,
    "TaskId": "xx",
    "Payload": {
      "Status": 1,
      "EventMessage": "Over time limit"
    }
  }
}
```

说明：
收到804回调状态码为2时，请检查录制过程中是否出现超出分辨率限制（1920*1080）的视频流。

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\"Even
tInfo\":\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}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
//# 返回值：
//#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#   Info：成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
```

```
# 参数：
#   key：控制台配置的密钥key
#   body：腾讯云回调返回的body体
#   sign：腾讯云回调返回的签名值sign
# 返回值：
#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'),
digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
```

```
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

输入在线媒体流回调

最近更新时间：2024-07-17 17:33:31

服务端输入在线媒体流回调支持将您使用 输入在线媒体流 REST API 产生输入在线媒体流的事件，以 HTTP/HTTPS 请求的形式通知到您的服务器。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“输入在线媒体流开始成功”事件的包体示例。

```
{
        "EventGroupId": 7,
        "EventType":    701,
        "CallbackMsTs":   1701937900012,
        "EventInfo":    {
                "EventMsTs": 1701937900013,
                "TaskId":"xx",
                "Status":0
        }
}
```

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，混流转推事件固定为4
EventType	Number	回调通知的事件类型
CallbackMsTs	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_STREAM_INGEST	7	输入在线媒体流事件组

事件类型

字段名	值	含义
EVENT_TYPE_STREAM_INGEST_START	701	输入在线媒体流开始
EVENT_TYPE_STREAM_INGEST_STOP	702	输入在线媒体流停止

事件类型为（EVENT_TYPE_STREAM_INGEST_START 701）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	输入在线媒体流任务 ID
Status	Number	输入在线媒体流开始状态

输入在线媒体流开始状态

字段名	值	含义	回调频率
STATUS_START_SUCCESS	0	输入在线媒体流开始成功	成功时回调1次
STATUS_START_FAILURE	1	输入在线媒体流开始失败	失败时回调一次
STATUS_START_AGAIN	2	输入在线媒体流再次开始	在0，1，3秒时重试，重试时回调

输入在线媒体流状态推荐处理

状态	处理方法
STATUS_START_SUCCESS	表示成功，无需处理。
STATUS_START_FAILURE	当您收到三个输入在线媒体流失败的状态，请检查源流URL，重新发起输入在线媒体流
STATUS_START_AGAIN	• 开始输入在线媒体流1分钟内收到：表示URL建连失败，或RTMP推流失败，系统自动触发重试，若最终失败请检查URL是否正常建立连接 • 开始输入在线媒体流1分钟后收到：可能是源流或后台网络抖动引起触发重新拉起，无需处理。

基本回调转移示例

- **输入在线媒体流失败/输入在线媒体流再次开始/输入在线媒体流开始成功的事件转移**
  `STATUS_START_FAILURE` -> `STATUS_START_AGAIN` -> `STATUS_START_SUCCESS`

> ⚠ **注意：**
> 输入在线媒体流回调有可能会乱序到达您的回调服务器，此时您需要根据 EventInfo 中的 EventMsTs 做事件排序，如果您只关心 URL 最新状态，可以忽略后续到达的过期事件。

事件类型为（EVENT_TYPE_STREAM_INGEST_STOP 702）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	输入在线媒体流任务 ID
Status	Number	输入在线媒体流停止状态

输入在线媒体流停止状态

字段名	值	含义	回调频率
STATUS_STOP_SUCCESS	0	输入在线媒体流停止成功	成功时回调1次

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\"Ebody="
{\"EventGroupId\":7,\"EventType\":701,\"CallbackMsTs\":1701937900012,\"EventInfo\":
{\"EventMsTs\":1701937900012,\"TaskId\":\"WMdqEeEgj2ksqnyUsuXC+qLkVypGmwjrgh1JC6ZefVP+rvsidDnZs
Aw8uWgX0XRGvdSVfAMunise2kcZaefdgHvx3-
M2v6fmTjRNgg..\",\"Status\":0}}"ventGroupId\":\t1,\n\t\"EventType\":\t103,\n\t\"CallbackTs\":\t
1615554923704,\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}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
//# 返回值：
//#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
```

```
//#   Info: 成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
# 参数：
#   key：控制台配置的密钥key
#   body：腾讯云回调返回的body体
#   sign：腾讯云回调返回的签名值sign
# 返回值：
#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'),
digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
```

```
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
```

```
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

AI 实时对话与语音转文字回调

最近更新时间：2024-11-04 19:44:42

本文用于介绍 AI 服务（ AI 实时对话 和 语音转文字 功能）相关的云 API 接口产生的事件，以 HTTP/HTTPS 请求的形式通知到您的服务器。您可以向腾讯云提供相关的配置信息来开通该服务。您也可以结合TRTC的 房间与媒体回调 使用，实现更多自定义逻辑。

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“启动AI对话任务成功”事件的包体示例。

```
{
    "EventGroupId":  9,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "hKPD2Q7kBVzu-6ezFiqmcEBJQCykqbZrS9OOTE46uYlb4NvQDIaEXlpOlLXFtGBiado5oP0zfLDZs",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
    }
}
```

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，混流转推事件固定为4
EventType	Number	回调通知的事件类型
CallbackMsTs	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_AI_SERVICE	9	AI 服务事件组

事件类型

字段名	值	含义
EVENT_TYPE_AI_SERVICE_START	901	AI 任务开始状态回调
EVENT_TYPE_AI_SERVICE_STOP	902	AI 任务结束状态回调
EVENT_TYPE_AI_SERVICE_MSG	903	回调完整的一句话。 • AI 实时对话：识别出完整的一句话后回调 • 语音转文字：回调转录的完整句子

事件类型为（EVENT_TYPE_AI_SERVICE_START 901）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号
Payload.Status	Number	• 0：启动 AI 任务成功 • 1：启动 AI 任务失败

```
{
    "EventGroupId": 9,
    "EventType": 901,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
    }
```

```
}
```

事件类型为（EVENT_TYPE_AI_SERVICE_STOP 902）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号
Payload.LeaveCode	Integer	• 0：正常调用停止接口后任务退出 • 1：业务自己踢掉转录机器人后任务退出 • 2：业务自己解散房间后任务退出 • 3：TRTC 服务端踢掉机器人 • 4：TRTC 服务端解散房间 • 98：内部异常错误，建议业务进行重试 • 99：代表房间内除了转录机器人没有其他用户流，超过指定时间退出

```
{
    "EventGroupId": 9,
    "EventType": 902,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "LeaveCode": 0
        }
    }
}
```

事件类型为（EVENT_TYPE_AI_SERVICE_MSG 903）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号
Payload	JSON Object	为JSON对象，与客户端自定义消息回调格式一致 { "UserId":"", "Text":"", "StartTimeMs":1234, "EndTimeMs":1269, "RoundId":"xxxxxx" // uuid

		}

```
{
    "EventGroupId":  9,
    "EventType":     903,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
       "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "UserId":"",
            "Text":"",
            "StartTimeMs":1234,
            "EndTimeMs":1269,
            "RoundId":"xxxxxx"
        }
    }
}
```

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\"Ebody="
{\"EventGroupId\":7,\"EventType\":701,\"CallbackMsTs\":1701937900012,\"EventInfo\":
{\"EventMsTs\":1701937900012,\"TaskId\":\"WMdqEeEgj2ksqnyUsuXC+qLkVypGmwjrgh1JC6ZefVP+rvsidDnZs
Aw8uWgX0XRGvdSVfAMunise2kcZaefdgHvx3-
M2v6fmTjRNgg..\",\"Status\":0}}"ventGroupId\":\t1,\n\t\"EventType\":\t103,\n\t\"CallbackTs\":\t
1615554923704,\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}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
```

```
//# 返回值:
//#   Status: OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#   Info: 成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能: 第三方回调sign校验
# 参数:
#   key: 控制台配置的密钥key
#   body: 腾讯云回调返回的body体
#   sign: 腾讯云回调返回的签名值sign
# 返回值:
#   Status: OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info: 成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'),
digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict
```

```
if __name__ == '__main__':
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
```

```
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

签名校验示例

最近更新时间：2024-08-09 19:11:21

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 。在配置回调信息前，您需提前准备计算签名的 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。
本文档将帮助您在计算签名后，如何校验签名进行示例。

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，需要完整保留\n\t转义字符**，示例如下：

```
body="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t103,\n\t\"CallbackTs\":\t1615554923704,\n\t\"Even
tInfo\":\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}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
//# 返回值：
//#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#   Info：成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        hmacSha256.init(secret_key);
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        // 请确保 body 为您收到回调请求的原始包体，不要做任何转化，需要完整保留\n\t转移字符，示例如下：
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}";
```

```
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
# 参数：
#   key：控制台配置的密钥key
#   body：腾讯云回调返回的body体
#   sign：腾讯云回调返回的签名值sign
# 返回值：
#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'), body.encode('utf-8'),
digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
     # 请确保 body 为您收到回调请求的原始包体，不要做任何转化，需要完整保留\n\t转移字符，示例如下：
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" + "\t\"EventType\":\t204,\n" +
"\t\"CallbackTs\":\t1664209748188,\n" + "\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php
```

```php
class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
// 请确保 body 为您收到回调请求的原始包体，不要做任何转化，需要完整保留\n\t转移字符，示例如下：
$body="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}";

$api = new  TlsEventSig($key, $body);
echo $api->genEventSig();
```

Golang

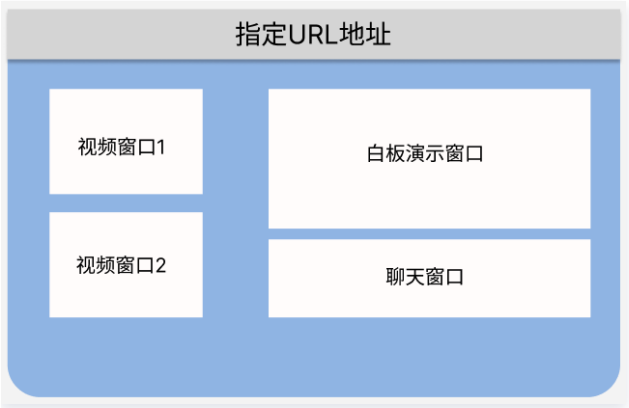
```golang
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
     // 请确保 body 为您收到回调请求的原始包体，不要做任何转化，需要完整保留\n\t转移字符，示例如下：
    var body= "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1608086882372,\n\t\"EventInfo\
":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"EventTs\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}
\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(body,key))
}

func hmacsha256(body string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(body))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

实现页面录制

最近更新时间：2025-06-05 18:23:52

应用场景

在 Web 端的远程教育、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的情况。页面录制提供在云端录制任何一个浏览器页面并存储的能力，实现随时随地回看。
以在线教育，视频会议场景为例，通过页面录制可以全场景录制页面内的全部元素，包括音视频画面，白板演示以及聊天窗口等各类内容，并保证通话内容与白板时间同步，可以完整的录制这个“课堂”或“会议”的所有实时信息，达到所见即所得的目的。



费用说明：
- 能力位解锁：使用页面录制功能需为您发起调用的应用（SDKAppId）订阅 TRTC 包月套餐 部分版本才可使用，前往购买 。
- 用量费用：使用页面录制功能会产生用量费用，详细请见 页面录制费用 说明。

功能说明

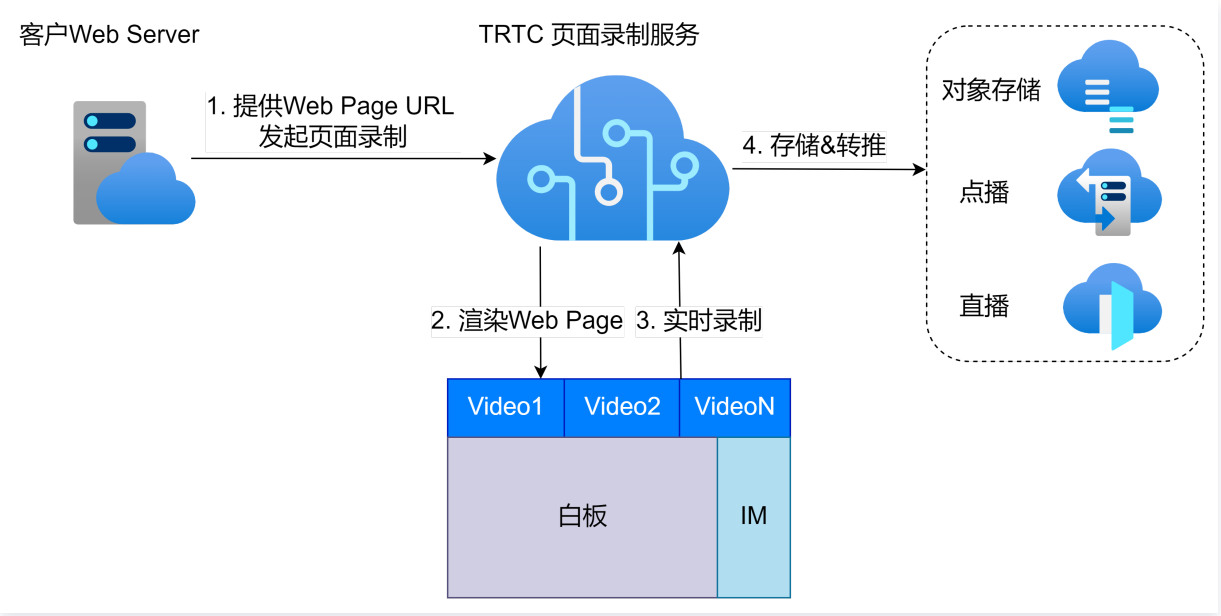
通过 TRTC 的页面录制功能，您可以获取整个浏览器页面的所有原始内容，并按照需要上传到指定的对象存储平台或点播平台。录制结果文件支持 MP4 和 HLS 格式。
- 录制浏览器页面： 我们通过指定浏览器页面，可以实现录制实时浏览器页面的能力。
- 录制模式：支持 页面录制 和 页面录制与转推 两种模式。
- 录制文件格式：支持 MP4 格式和 HLS 格式。
- 输出分辨率：在不超过1920*1080的限制下，我们支持设置不同的输出分辨率。
- 文件存储：支持存储到对象存储COS 或存储到云点播VOD
- 回调通知：我们支持回调通知的能力，通过配置您的回调域名，页面录制的事件状态会通知到您的回调服务器。录制回调地址配置和事件说明请见 云端录制和页面录制回调 。

注意：
待录制页面需要 URL 加载完即播放，且不需要任何形式的交互操作，其他更多注意事项请参见 接入注意事项 。

录制模式说明

页面录制

如下图所示为使用页面录制的经典场景：
1. 客户可以通过云 API 发起页面录制请求，在该请求中指定待录制的 Web Page 的 URL，以及录制存储参数。
2. TRTC 页面录制服务会在云端访问指定的 URL 并实时渲染，保留 Web Page 所呈现的所有原始内容。
3. TRTC 页面录制服务对渲染出的 Web Page 进行实时录制，使录制结果能够还原 Web Page 的全景真实效果。
4. 在录制任务结束后，会根据录制参数生成 HLS 或者 MP4 文件并上传到指定云存储平台（目前支持腾讯云COS和腾讯云VOD）。



页面录制并转推

TRTC 对于具有同时使用页面录制并转推场景的客户，提供了可一次调用发起可同时实现转推并录制的使用方式（在页面录制请求中设置转推参数），在进行录制的同时，将音视频流实时转推到 CDN 平台，实现转推观看和录制内容完全一致。

> ⚠ **注意：**
> 使用页面录制并转推模式时，转推相关操作请关注 旁路转推回调，若同时发起转推功能会产生 旁路转推费用 。

文件切分说明

录制 MP4 文件切分的条件：

- 录制切分时长可设置范围1-1440分钟，默认120分钟。
- 单个 MP4 文件大小达到 2GB。

录制上传云存储说明

录制后台会在录制结束后将录制的文件通过您指定的方式上传到云存储平台（云点播 VOD 或对象存储 COS），并通过回调的形式把录制结果（播放地址或录制文件名）发送给您。

- 上传对象存储COS时，为确保您能够获取到媒体文件存储地址，请您关注310回调。310回调把录制任务 ID(TaskId) 以及录制文件名(FileList)发送给您。您需要根据页面录制请求中的第三方存储桶信息（StorageParams.CloudStorage.Bucket）、第三方云存储的地域信息(StorageParams.CloudStorage.Region)、文件位置信息(StorageParams.CloudStorage.FileNamePrefix)以及录制任务ID(TaskId)和录制文件名(FileList)自行拼接出媒体文件播放地址。
- 上传云点播VOD时，为确保您能够获取到媒体文件播放地址，请您关注311回调。311回调会把录制文件在点播平台的播放地址发送给您。

> ⚠ **注意：**
> - 文件录制后会上传至您指定云存储平台（云点播 VOD 或对象存储 COS），为确保录制文件成功，请确保您指定的云点播 VOD 或对象存储 COS 服务可用。
> - 页面录制仅支持上传单一云存储平台（云点播 VOD 或对象存储 COS），不支持同时设置对应存储参数。
> - 上传对象存储 COS 时，310回调返回的录制任务 ID(TaskId)会额外携带类似“_StartTimeMs_1717156238963”这样的后缀，后缀中的时间戳长度固定为13位，您可以按实际业务需要对返回的录制任务 ID 进行截取。

API 使用接口说明

API 接口和录制并发限制

- 录制接口的调用频率限制为20qps。

- 单个接口超时时间为5秒。
- 单个应用下默认并发录制支持200路，超过并发限制的任务会失败。

发起录制

通过调用 API（ StartWebRecord ）来启动页面录制，需要重点关注响应结果中的参数——任务 ID（ TaskId ）；这个参数是本次录制任务的唯一标识，您需要保存下这个任务 ID 作为后续针对这个录制任务接口操作的输入参数。

> **说明：**
> 您可以前往控制台配置回调地址，以接收录制回调事件，请见 云端录制和页面录制回调说明 。

查询录制状态

通过调用 API（ DescribeWebRecord ）来查询录制任务状态，输入参数为录制时响应结果中的任务 ID（ TaskId，这个参数是本次录制任务的唯一标识），或在输入参数中使用发起页面录制时输入的SdkAppId和RecordId，通过上述参数可以查询到对应录制任务的录制状态。

- 录制任务进行中：调用 API 返回的响应中的 Status 为1时，代表录制任务正在进行中。
- 录制任务已结束：调用 API 返回的响应中的 Message 为“**task not exist**”时，代表录制任务已结束或尚未启动。

停止录制

通过调用 API（ StopWebRecord ）来停止录制任务，需要使用发起录制时响应结果中的参数——任务 ID（ TaskId ）；这个参数是本次录制任务的唯一标识，通过这个参数就可以停止对应录制任务。

> **说明：**
> 您可以通过启动录制中的 MaxDurationLimit 参数来指定录制任务的持续时间，录制任务持续时长达到指定的MaxDurationLimit值时，会自动停止录制，从而省去调用停止录制 API 的操作。默认录制任务最大录制时长为10小时。

录制回调事件

页面录制功能提供了多种的回调事件，帮助您及时了解录制任务的处理进度和完成情况，录制回调地址配置和事件说明请见 云端录制和页面录制回调 。

录制文件管理

查找录制文件

录制任务结束后，TRTC 页面录制中录制下来的文件上传至您指定的云存储平台（云点播VOD或对象存储 COS）。您可以直接前往 云点播控制台 或 对象存储COS控制台 查找。

> **说明：**
> 对于对象存储 COS，如果您在启动录制参数中设置了 FileNamePrefix 参数，录制文件将保存在您指定的存储桶 `Bucket/${FileNamePrefix}/${TaskId}` 下；否则，录制文件将直接保存在存储桶 `Bucket/${TaskId}` 下。

接收录制文件

录制文件上传云点播 VOD 时，除了手动查找录制文件，您还可以通过在控制台 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，该过程大约默认需要30秒至数分钟（具体时间根据您所录制的文件大小而定）。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，您可以通过接收事件类型为311的上传成功回调来获取录制文件的播放地址 VideoUrl，具体回调信息见下方：

```
{
  "EventGroupId": 3,
  "EventType": 311,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
```

```
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
      "Status": 0,
      "TencentVod": {
        "UserId": "xx",
        "TrackType": "audio_video",
        "MediaId": "main",
        "FileId": "xxxx",
        "VideoUrl": "http://xxxx",
        "CacheFile": "xxxx.mp4",
        "StartTimeStamp": "xxxx",
        "EndTimeStamp": "xxxx"
      }
    }
  }
}
```

接入注意事项

针对 Web 应用的限制

- 由页面录制生成的视频分辨率上限为 1920 × 1080。
- 待录制的网页中任何视频源的分辨率不应超过 1920 × 1080。
- 待录制页面的下行带宽不得超过 5 Mbps，上行带宽不得超过 5 Mbps。
- 待录制页面不应使用 WebGL 功能。
- 请确保您的Web应用不会过度占用CPU、内存和带宽，并且该Web应用的使用应符合法律法规。
- 页面录制支持 在无用户交互的情况下自动播放已启用autoplay属性的video元素。然而，如果待录制的网页中的video元素未启用autoplay属性，其内容将不会自动播放，这可能导致页面录制无法录制该网页。
- 待录制页面不应跳转至不同域名的 URL，并尽可能避免其他形式的跳转。如果待录制的页面需要登录操作，请先处理登录操作，然后进行录制。否则，录制结果可能会一直是待登录界面。

云 API 请求

- 从请求发起到开始页面录制，可能会有约5秒的延迟。建议提前发起录制请求，以确保录制内容的完整性。
- 页面录制不支持更改布局。
- 如果您在StartWebRecord方法中填入的RecordUrl无法正常打开，录制服务将在StartWebRecord成功后自动退出。您可以参考云端录制集成最佳实践，使用退避策略多次调用DescribeWebRecord，以确认录制服务已正常启动。

检测页面加载超时

页面录制场景下，网络异常等偶然因素可能会造成以下问题：

- 无法正常访问待录制页面，如页面加载失败或时间过长，无法获知真正开始有效录制的时间点，可能会丢录重要内容。
- 可以正常访问待录制页面，但未能正确加载页面中的 HTML 元素。
- 录制过程中未能正常加载页面中发生变化的 HTML 元素，从而导致录制内容与预期不一致。
- 未能正常播放待录制页面中的音视频。

为了确保页面录制的内容与预期一致，建议您按照以下方案来提高页面录制的可靠性。

1. **设置页面加载超时时间**

   调用 StartWebRecord 方法时通过 `ReadyTimeout` 字段设置页面加载超时的时间限制。

   `ReadyTimeout`：Number 类型，单位为秒，取值范围 [0,60]：

   - `0` 或不设置，表示不检测页面加载状态。
   - `≥ 1`，表示页面加载超时时间。
   - `< 0` 或非整数，表示设置错误，云API会返回错误信息。

> ⚠ **注意：**
> 当您设置了 `ReadyTimeout` 时，请务必确保待录制页面有判断页面是否加载完成，以避免因未检测到页面加载就绪从而导致录制任务启动失败。

2. **判断加载是否完成并通知浏览器**

> ⚠ **注意：**
> 您需要自行判断页面是否加载完成，然后实现后续逻辑。

页面加载完成

如果页面加载完成，则在设定的 `ReadyTimeout` 时间内调用 `notifyReady` 方法通知浏览器页面加载成功。
`notifyReady` 的调用示例如下：

```
// notifyReady调用示例
<script>
function notifyReady() {
    if (window.notifyReady) {
        window.notifyReady();
    }
}
</script>
```

页面加载超时

如果页面加载超时，即在设定的 `ReadyTimeout` 时间内未调用 `notifyReady` 方法通知浏览器，则浏览器自动重新加载页面。您会收到 803 事件回调（EVENT_TYPE_WEB_RECORDER_STATUS_UPDATE），其中 `Status` 字段为 `1`， `EventMessage` 字段为 `Page load timeout`。

- 如果重新加载成功，则参考页面加载完成的逻辑，调用 `notifyReady` 方法通知浏览器。
- 如果页面加载再次超时，则表示页面重新加载失败，录制服务停止。您会收到 801 事件回调（EVENT_TYPE_WEB_RECORDER_START）通知您录制任务启动失败，其中 `Status` 字段 为 `2`， `EventMessage` 字段为 `Page load timeout`。在收到回调通知后，您可根据实际业务需要，决定是否重新发起录制任务。

其他说明

- 在录制过程中，如果当前MP4文件的时长超过maxVideoDuration的值或大小超过2GB时，录制服务将创建一个新的MP4文件。
- 如果您在 start 方法中填入待录制页面的 URL 会自动触发 Web 客户端发布音视频流，录制服务也会成为一个发流端，因此，您的应用中可能会出现一个绿色背景色的用户画面。为规避该问题，您可以在待录制页面的 URL 中增加查询字段 is_recorder=1，例如："https://url?is_recorder=1"，并在该页面内添加以下逻辑：
  - 如果 is_recorder 为 1，则 Web 客户端不发布音视频流。
  - 如果 is_recorder 不为 1，则 Web 客户端发布音视频流。
- 进行页面录制时，录制服务相当于一个使用 Web 应用的客户端，因此，如果您的Web应用包含用户列表，建议您在用户列表中隐藏该用户。

拓展场景

TRTC 页面录制解决方案不仅可以录制 TRTC 的 RTC 会话，对于传入的任意可访问的页面，该方案均可以录制。因此对于开发者来说，借助于 TRTC 页面录制的能力，可以衍生出更多的创意玩法，例如：

1. 通过技术手段，将多人的本地页面及云端渲染页面的操作同步，通过页面录制，将多人协作的过程录制下来，作为后续的教程资料。
2. 多人可以使用页面录制方案录制一个视频源，并转推到直播 CDN，后续多人可以通过直播流一起观看。

后续 TRTC 页面录制方案，会在录制及转推基础能力上，结合AI探索更多的音视频处理增值服务，助力客户进一步降本增效，扩展业务边界。

语音转文字

最近更新时间：2025-06-05 18:23:52

应用场景

TRTC 支持**语音转文字功能**，将房间内指定用户或所有用户的音频流识别成对应的文字，实现实时字幕等效果。

前提条件

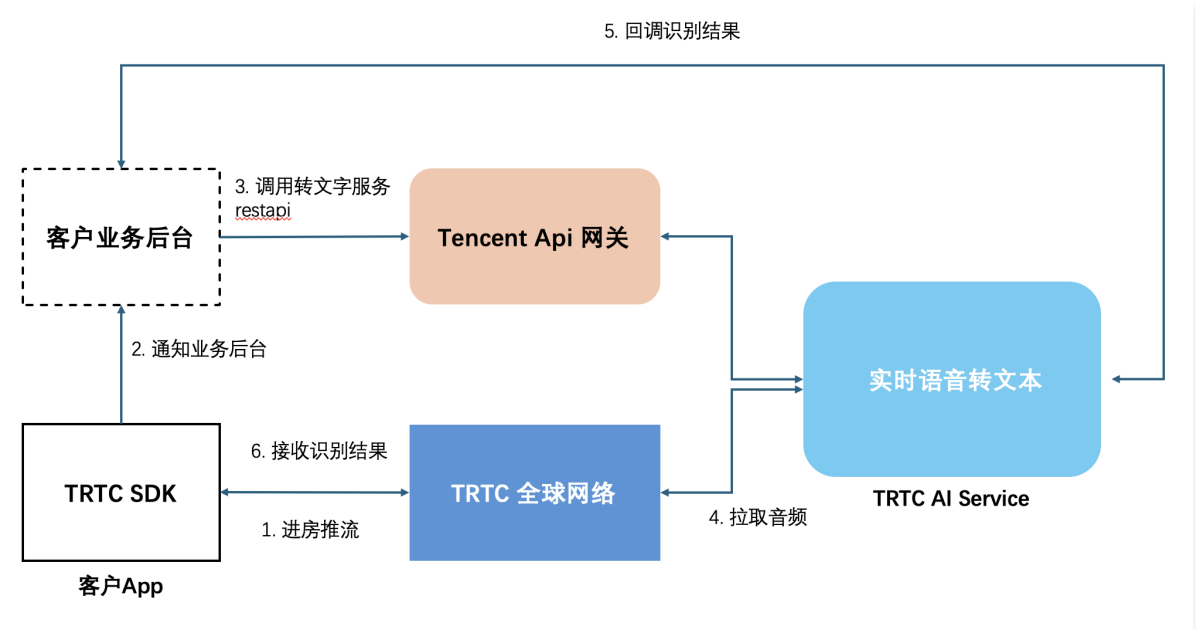
- 登录 TRTC 控制台，开通 TRTC 服务并 创建应用 。
- 前往 控制台 > 功能配置 > 增值功能 开启 AI 智能识别功能。

说明：

- 开启前需领取**包月套餐体验版**、购买 AI 智能识别套餐包 、或订阅 订阅 TRTC 包月套餐 部分版本才可使用，前往购买 。
- 语音转文字功能会根据调用量产生费用，详情请参见 费用详情 。

功能说明

任务发起后，TRTC AI Service 通过识别机器人进入 TRTC 房间拉流指定用户或所有用户的流，进行语音转文字识别，将识别结果实时回调给客户端和服务端。



接入说明

第一步：接收语音转文字识别结果

方式一：通过客户端 SDK 接收文字消息

通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实时的语音转文字的结果数据。
客户端回调消息格式如下，以 Web 端为例：

```
trtc.on(TRTC.EVENT.CUSTOM_MESSAGE, event => { // receive custom message
    // event.userId: 语音识别机器人的userId
    // event.cmdId: 消息Id, 转录和字幕固定为1
    // event.seq: 消息的序号
    // event.data: ArrayBuffer 类型， 转录或字幕的内容，见下方data字段说明
    const data = new TextDecoder().decode(event.data)
    // data 字段说明如下
    console.log(`received custom msg from ${event.userId}, message: ${ data }`)
})
```

data 字段说明

实时的字幕消息

字段名	类型	含义
type	Integer	10000: 实时字幕与完整的一句话下发消息类型
sender	String	说话人的 userid
receiver	Array	接受者 userid 列表，该消息实际是在房间内广播
payload.text	String	识别出的文本。Unicode 编码
payload.start_time	String	消息产生的时间 任务启动后的绝对时间
payload.end_time	String	消息结束的时间 任务启动后的绝对时间
payload.end	Boolean	如果为 true，代表这是一句完整的话

```
{
  "type": 10000,
  "sender": "user_a",
  "payload": {
     "text":"",
     "start_time":"00:00:02",
     "end_time":"00:00:05",
     "end": true
  }
}
```

说明：
回调示例说明：
转录：会将完整的一句话转录并推送
“今天天气怎么样？”
字幕：将句子分段推送，后一段会包含前一段，确保实时性。
“今天”
“今天天气”
“今天天气怎么样”

顺序说明：字幕消息 > 字幕消息 > .... > 字幕消息(end = true)

方式二：通过服务端回调接收

语音转文字服务同时提供了服务端事件回调，便于您的服务接收实时对话的消息，查看 详细回调事件 。

第二步：发起语音转文字任务

TRTC 提供以下云 API 用于发起和管理语音转文字任务，具体如下：

- 开始语音转文字任务：StartAITranscription
- 查询语音转文字任务：DescribeAITranscription
- 停止语音转文字任务：StopAITranscription

注意：
语音转文字功能单个 SDKAppId 任务并发数限制**100路**，如需提升提交工单处理。

音视频内容安全审核
自动审核接入

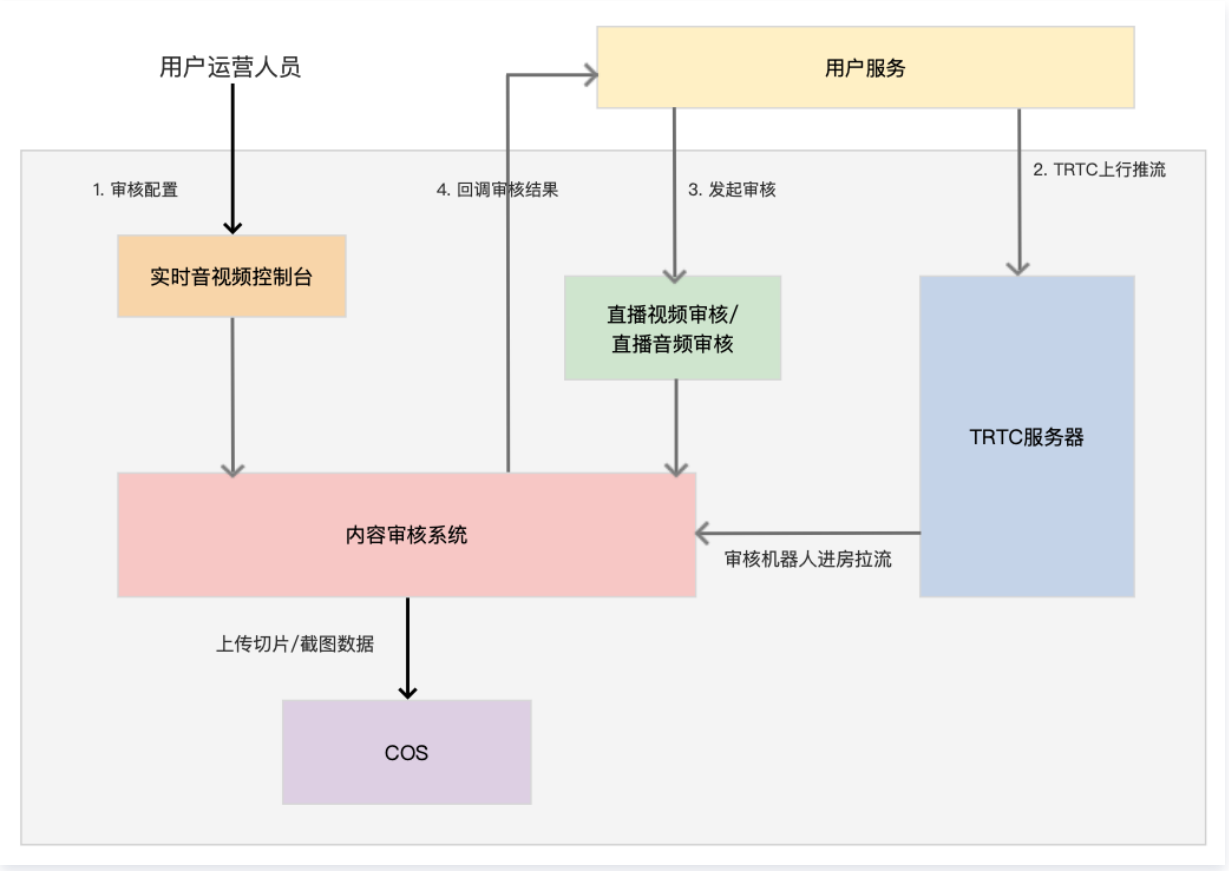
最近更新时间：2024-07-10 11:39:21

TRTC 联合 T-Sec 天御，提供了实时的音视频内容识别与告警服务，客户在使用实时音视频服务时，支持手动或全局自动发起策略进行音视频内容的识别和告警：

- **手动自定义审核**：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。
- **全局自动审核**：客户可指定审核策略和审核流类型，TRTC云端自动帮忙完成应用下所有房间内的音视频内容审核，并通过回调把违规信息发送给客户指定的回调 URL，无需手动发起审核。

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。



费用说明

音视频内容审核费用由 T-Sec 天御收取，新用户开启审核服务后，会赠送新手使用套餐包，消耗完后，默认按照后付费价格进行扣除，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 。

> **注意**
> **首次**开通账号可免费试用36000张图片与600分钟音频识别服务，可用于直播视频审核服务；同时可**免费试用10小时**识别服务，可用于直播音频审核服务，**试用套餐包有效期为1个月，两种审核服务的套餐包不通用**。套餐包用量查询和购买请前往 直播视频套餐包管理 和 直播音频套餐包管理 。
> 当房间内发起任务需要将房间内的音视频流输出至房间外时（包括录制、审核、转推等场景），系统会分配对应的机器人将作为一个虚拟观众加入房间（多个同类型的任务也会有多个机器人进房），订阅需要输出的音视频流，从而产生音视频时长费用。

使用流程

1. 进入 腾讯云 TRTC 控制台，完成应用创建，并进入应用管理，进入内容安全审核模块。

2. 未开通审核服务应用请先开通内容安全审核服务，并授权角色服务权限（腾讯云 COS 相关权限）用于存储相关审核结果。（每个应用需要单独开启审核服务）

3. 内容安全审核默认支持**手动自定义审核**，手动审核需要您进行相关开发集成，如需接入请参见 手动审核接入指引 。

如需使用**全局自动审核**，即希望当前应用下的所有房间内的音视频流都按照指定的审核策略进行自动审核，请单击右侧**编辑**，完成自动审核配置。

> **说明：**
> 全局自动审核我们为您提供了默认审核策略场景，即“音视频互动直播场景”，其中包含了适用于音视频通话的直播视频审核策略和适用于音视频互动直播的审核策略，您可以根据业务场景在此策略场景基础上进行调整，操作指引请参见 控制台指南-应用管理-内容安全 。

配置项	说明
审核模式	支持手动自定义审核（默认）和全局自动审核
审核类型	• **直播视频审核**：将对房间内的音视频流进行审核，适用于视频通话、互动直播应用 • **直播音频审核**：仅对房间内的音频流进行审核，适用于语音聊天应用
审核流类型	• **单流**：将房间内的每一路音视频流进行单独审核 • **混流**：将房间内的所有音视频混合成一路进行审核（混流审核无法获取具体违规的角色信息，适用于审核房间维度下是否违规的场景）
审核策略场景	默认提供音视频场景下的审核策略：**音视频互动直播场景**，您可在此审核策略上进行修改调整审核策略
回调地址	审核过程中接受审核信息回调的地址，回调 URL 协议头：`http`、`https` 等，仅可包含以下字符：a-z、A-Z、0-9、-、_、?、%、=、#、.、/和+
回调密钥	回调密钥由大小写字母及数字组成，不超过128个字符，用于保证回调数据的安全性，具体签名验证示例见 回调签名验证

注意：
手动审核默认开启，全局自动审核需手动开启，开启生效时间为10分钟，生效后对于开启后新创建的房间内的音视频流进行自动审核，当您开启了全局自动审核开关后，请勿在使用相同的应用(SDKAppId)发起手动审核，将会产生冲突导致审核失败。

4. **完成配置等待10分钟生效后，新创建房间中的音视频流将会自动按照您设置的审核策略进行审核（手动审核将由您控制审核发起）。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。**

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 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。
视频审核任务回调字段说明请见 查看视频任务详情 中的输出参数，音频审核任务回调字段说明请见 查看音频任务详情 中的输出参数，TRTC 回调的片段会存储在 COS上名称为：`tianyu-content-moderation-{账号appid}` 的存储桶中，文件名为：

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

说明：

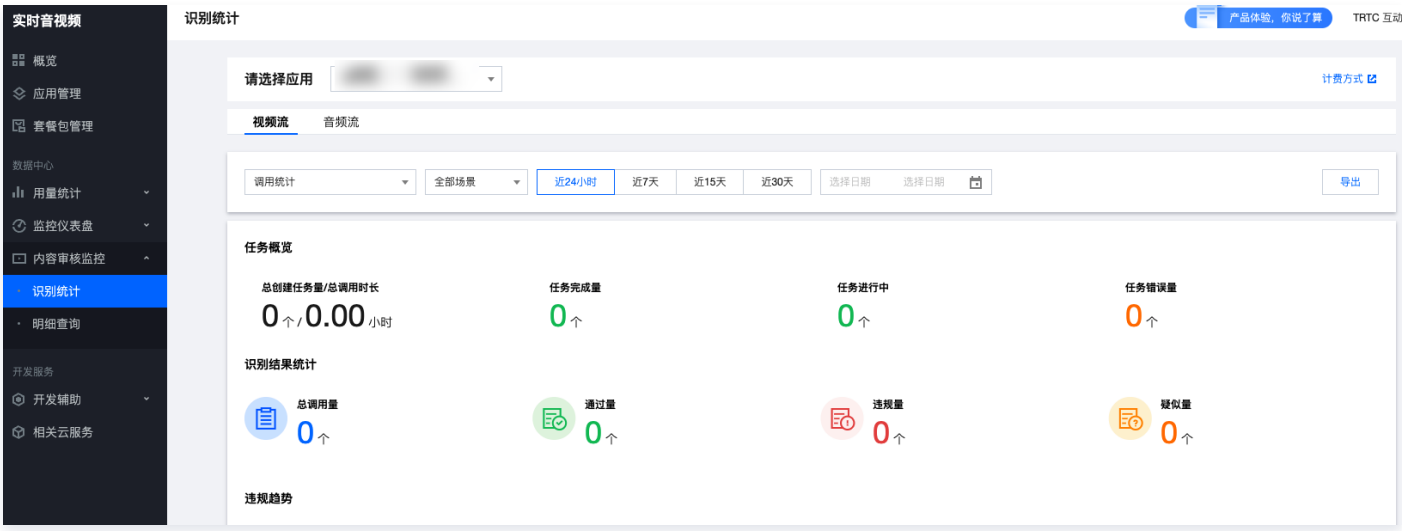
- 回调内容字段说明请参见 查看任务详情 中的输出参数。
- 上述信息存储在审核回调的 `ImageSegments` / `AudioSegments` 字段中，您可以用来确认审核的 `sdk_app_id` 、 `room_id` 、和 `user_id` （混流审核下的 `user_id` 统一为 mixer，混流审核不支持区分具体的用户 `user_id` ）。

  图片审核回调示例：

{"TaskId":"w-video-ZSS_LFTB3lSvDHmP", "DataId":"1111", "BizType":"zpj_test", "Name":"", "Status":"RUNNING", "Type":"VIDEO", "Suggestion":"Block", "Labels":[], "InputInfo":null, "MediaInfo":null, "AudioText":"", "ImageSegments":[], "AudioSegments":[{"Result":{"HitFlag":1, "Url":"https://cos.ap-guangzhou.myqcloud.com/tianyu-cms-ap-guangzhou-1257157379/segment-/trtc/1400188999/screenshot_6541925_abc123_1696919763.jpg", "Suggestion":"Block".....省略后续内容}

  根据上面回调信息，sdkappid：1400188999，roomid：6541925，主播 userid：abc123，审核时间戳：1696919763。

5. 如需查看审核明细和审核结果统计数据，请前往**实时音视频控制台指南 > 用量统计 > 内容审核监控** 进行查看。



手动审核接入

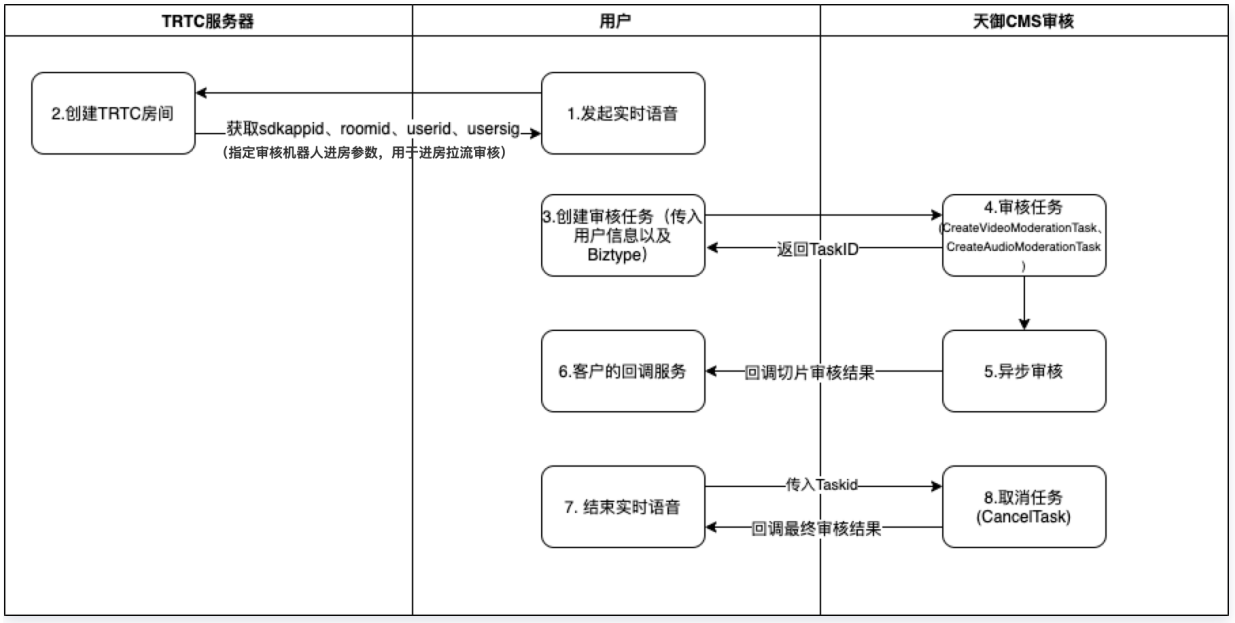
最近更新时间：2025-07-02 20:13:14

以下文档用于介绍自定义手动审核的集成方式，需要您来控制审核任务的发起与结束。如果您希望使用的是全局自动审核功能，只需在控制台完成配置即可，具体请参见 全局自动审核 。

> **注意：**
> 本文介绍如何针对某个应用发起手动内容审核任务，手动审核不可与全局自动审核同时发起，请确认当前应用在控制台的全局自动审核开关已关闭，关闭方式请见 审核配置说明 。

审核接入流程

接入流程图



接入步骤

1. **确认审核机器人进房参数：**
   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），指定进房审核用户user_id（作为审核机器人进房拉流，**不能与当前房间内正常用户的user_id相同**），根据 user_id 获取参数 user_sig（获取方式请参见 user_sig 相关 ）。
2. **拼接 TRTC 审核 URL：**
   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。
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。 注：room_id长度需小于16字符。
user_id	是	用户 ID，手动指定一个内容审核用户user_id（该审核用户仅作为审核机器人进房拉流，不能与当前房间内正常用户的userid相同）。
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。（混流审核下违规回调无法获取具体的违规用户，适用于房间维度的合规审核场景，单流审核可通过回调确定具体的违规 user_id。）
room_id_type	否	房间类型，默认可不传，可传入值为：string/number。
sub_stream	否	标识是否审核辅流（屏幕分享画面），bool 类型，true 为审核辅流，false 为不审核辅流，默认 false。

3. **调用审核任务创建接口：**

根据您的审核需求，如果需要对房间内的音、视频流进行审核，通过以下 创建视频审核任务 接口，发起该房间的音、视频内容安全审核。

参数名称	必选	类型	描述
Action	是	String	公共参数，本接口取值：CreateVideoModerationTask。
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。
Type	是	String	该参数用于传入审核任务的任务类型，取值：VIDEO（点播视频），LIVE_VIDEO（直播视频），TRTC 审核场景请传入 **LIVE_VIDEO（直播视频）**。
Tasks.N	是	Array of TaskInput	该字段表示输入的音频审核任务信息（即上述拼接的 TRTC 审核任务 URL），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3-32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。**
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 回调签名验证示例 。
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 回调内容示例 。 备注：音频默认截取时长为15秒，视频截帧默认为5秒截取一张图片；若用户自行配置截取间隔，则按照用户配置返回相应片段。
Priority	否	Integer	可选参数，该参数用于传入审核任务的优先级。当您有多个视频审核任务排队时，可以根据这个参数控制排队优先级，用于处理插队等逻辑；该参数默认值为0。

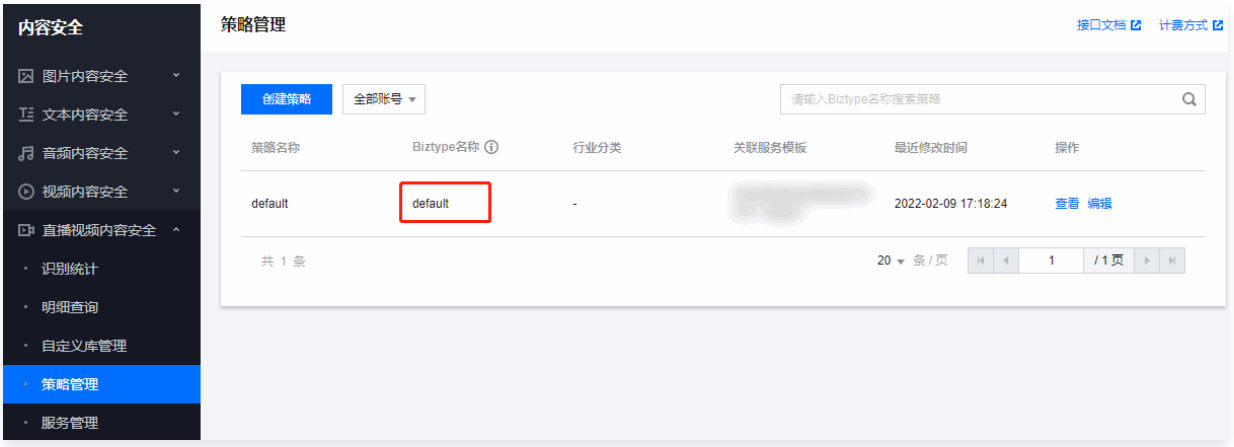
若仅需对房间内的音频流进行审核，可通过 创建音频审核任务 接口，发起该房间的音频内容安全审核。

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（即上述拼接的TRTC审核任务URL），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。
Type	是	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），TRTC 审核场景请传入 **LIVE_AUDIO（直播音频）**。
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。**
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 回调签名验证示例 。
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 回调示例 。

以上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
  根据您调用审核任务的类型进行传入，音视频审核任务接口（CreateVideoModerationTask）该字段传入LIVE_VIDEO，音频审核任务接口（CreateAudioModerationTask）该字段传入LIVE_AUDIO。
- 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 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。

   视频审核任务回调字段说明请见 查看视频任务详情 中的输出参数，音频审核任务回调字段说明请见 查看音频任务详情 中的输出参数，TRTC 回调的片段会存储在 cos 上名称为：tianyu-content-moderation-{账号appid}的存储桶中，文件名为：

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

   说明：
   上述信息存储在审核回调的 ImageSegments/AudioSegments 字段中，您可以用来确认审核的 sdk_app_id、room_id、和 user_id（混流审核下的 user_id 统一为 mixer，混流审核不支持区分具体的用户 user_id）。
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。

注意：
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，内容安全是以观众身份从 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，否则会导致审核失败。
- 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. 内容安全是以观众身份从 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** > UserSig 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。需要注意的是，混流审核下无法区分违规用户的 user_id，如需识别具体的违规用户，需要开启单流审核，以 user_id 区分用户。图片和音频在 COS 中的存储文件名如下：

- 图片格式：`trtc/{{sdk_app_id}}/screenshot_{{room_id}}_{{user_id}}_{{timestamp}}.jpg`
- 音频格式：`trtc/{{sdk_app_id}}/audio_{{room_id}}_{{user_id}}_{{timestamp}}.mp3`

> **注意：**
> user_id 为主播号。

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

- 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
- 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

输入媒体流进房

最近更新时间：2024-12-02 10:05:02

概述

一起看、一起听、一起玩、一起学……原来需要线下面对面才能实现的各种体验正被不断搬到线上。相隔千里还能和好友们一起看电影、一起听音乐，然后一起交流吐槽，这样神奇的实时互动体验正受到当下年轻人的喜爱，并成为如今音视频产品的重点玩法和主流方向。
TRTC 提供 输入在线媒体流 和 RTMP 推流进房 两种推流进房方案，有各自对应的适用场景，具体如下：

- 输入在线媒体流 用于**拉取云端在线媒体流（在线流或云端点播文件）**推流至 TRTC 房间内进行观看.
- RTMP 推流进房 **用于将本地媒体文件、音视频设备采集音视频**通过 RTMP 标准协议推流到 TRTC 房间内。

> **说明：**
> 相关费用如下：
> - 功能位解锁：**输入在线媒体流**和 **RTMP 推流进房**功能需订阅 TRTC包月套餐 **尊享版**或**旗舰版**解锁。
> - 用量费用：
> - 使用推流功能会进行转码操作，产生转码费用，详情参见 云端混流转码计费说明 。
> - 收取推流机器人在房产生的音频时长费用（注：输入在线媒体流功能产生的机器人在房费用将限免于2024年8月15日，从2024年8月16日起开始收取）。
> - 房间内观众订阅推流进房的音视频内容会正常产生音视频通话费用，详情参见 音视频时长计费说明 。

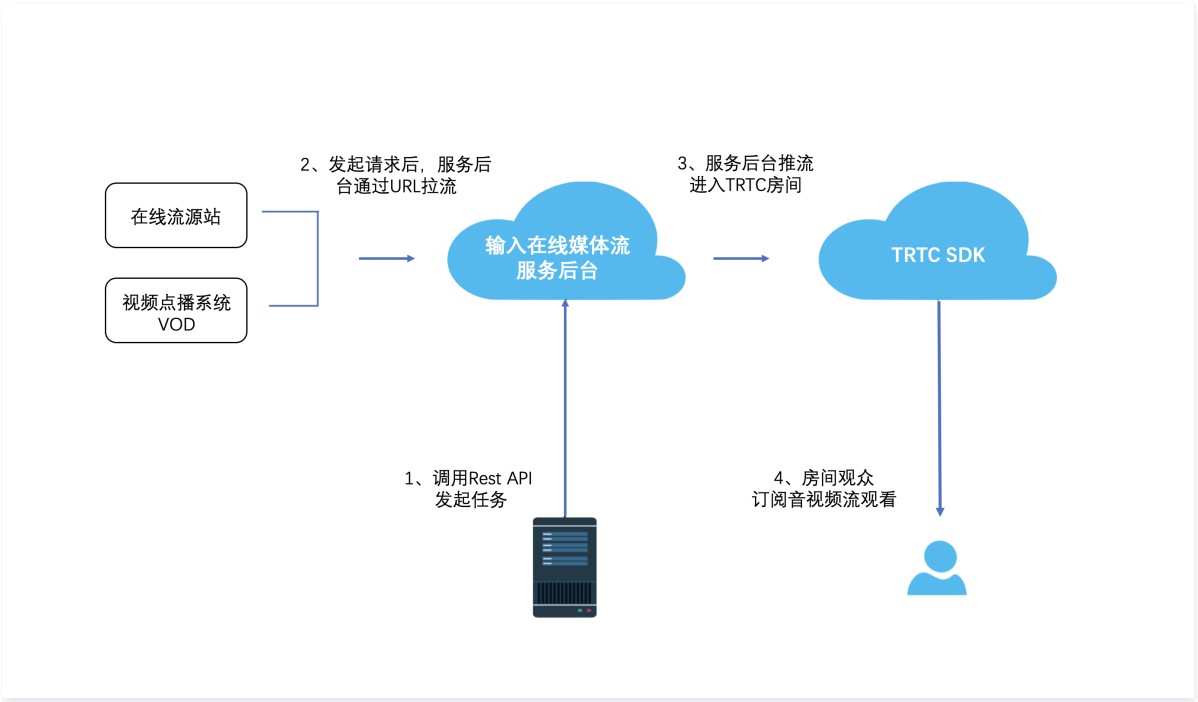
输入在线媒体流

应用场景

场景类型	说明
AI 互动课堂	依托 TRTC 输入在线媒体流能力，平台可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，在保证教学效果的同时大幅降低运营成本。上课前，平台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 TRTC 输入在线媒体流能力将对应视频推送到 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 TRTC 输入在线媒体流能力将直播流推送到 TRTC 房间，实现房间内超低延时同步观看，配合 TRTC 的实时互动能力，观众可实时交流，一起加油喝彩，沉浸式观赛。电影、音乐等点播类节目，同样可以通过该能力输入至 TRTC 房间，帮助用户实时共享，与好友边看边聊。

功能架构

1. 用户使用 REST API 创建输入在线媒体流任务，输入在线媒体流任务由中转服务（relay server）执行。
2. 中转服务拉取在线流或者点播文件。
3. 中转服务将拉取到的音视频推至 TRTC 房间，中转服务中会自动生成一个虚拟主播用户，该用户的用户名和要进入的房间号在创建任务时指定。
4. TRTC 其他端可观看这路流，也可以复用 TRTC 录制、转推等能力。



功能描述

输入在线媒体流功能说明如下：

类型	描述
发起任务方式	用户可以通过 REST API 发起输入在线媒体流任务，观众可观看这路流，支持录制、转推等功能。
多种源流协议和格式	协议：HTTP、HTTPS、RTMP、HLS 格式：FLV、MP3、MP4、MPEG-TS、MOV、MKV、M4A 视频编码：H.264、VP8 音频编码：AAC、OPUS
服务端回调	输入在线媒体流任务创建和结束时可回调给业务侧服务器，用于业务侧做逻辑，详细输入在线媒体流事件，前往查看。

相关 Rest API

- 开启输入在线媒体流：StartStreamIngest
- 停止输入在线媒体流：StopStreamIngest
- 查询输入在线媒体流：DescribeStreamIngest

RTMP 推流进房

TRTC 支持将**本地媒体文件、音视频设备采集音视频**通过 RTMP 标准协议推流到 TRTC 房间内。为降低客户接入门槛，您可根据实际情况选择安装 OBS 、FFmpeg 或其他 RTMP 库进行推流。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。

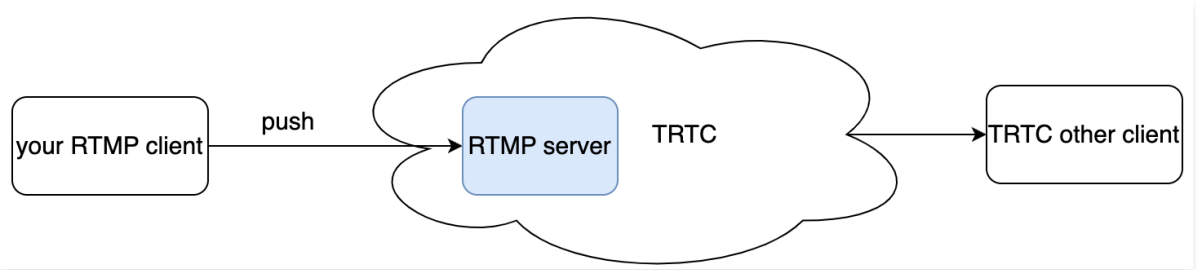
应用场景

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。

网络架构

RTMP 属于 TRTC 的一个子模块，能与 TRTC 其他端互通，互通延迟在正常情况下小于600ms，也可使用 TRTC 录制、转推等已有能力。网络架构如下图所示。**不支持使用 RTMP 从 TRTC 拉流，只支持 RTMP 推流。**



流地址生成

推流地址

```
rtmp://rtmp.rtc.qq.com/push/房间号?sdkappid=应用&userid=用户名&usersig=签名
```

- 主域名是 rtmp.rtc.qq.com，备域名 rtmp.cloud-rtc.com，如果主域名解析有问题，可使用备域名。
- RTMP appName 是 push。
- 地址中的房间号、应用、用户名、签名需要换成业务的。
- 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。

> **警告：**
> 1. TRTC 其他端如果要观看 RTMP 流，请使用**字符串房间号进房**。
> 2. 以 小程序端 为例填写 enterRoom 接口 strRoomID 字段，其他端参考相应的 API 文档。

- usersig 的生成规则，请参见 UserSig 相关 （**请注意签名要在有效期内**）。

示例：

```
rtmp://rtmp.rtc.qq.com/push/hello-string-room?
sdkappid=140*****66&userid=******rtmp2&usersig=eJw1jdE********RBZ8qKGRj8Yp-wVbv*mGMVZqS7w-mMDQL
```

使用示例

您可以使用支持 RTMP 协议的软件或者代码库推流，下面列举几种。

OBS 推流

准备工作

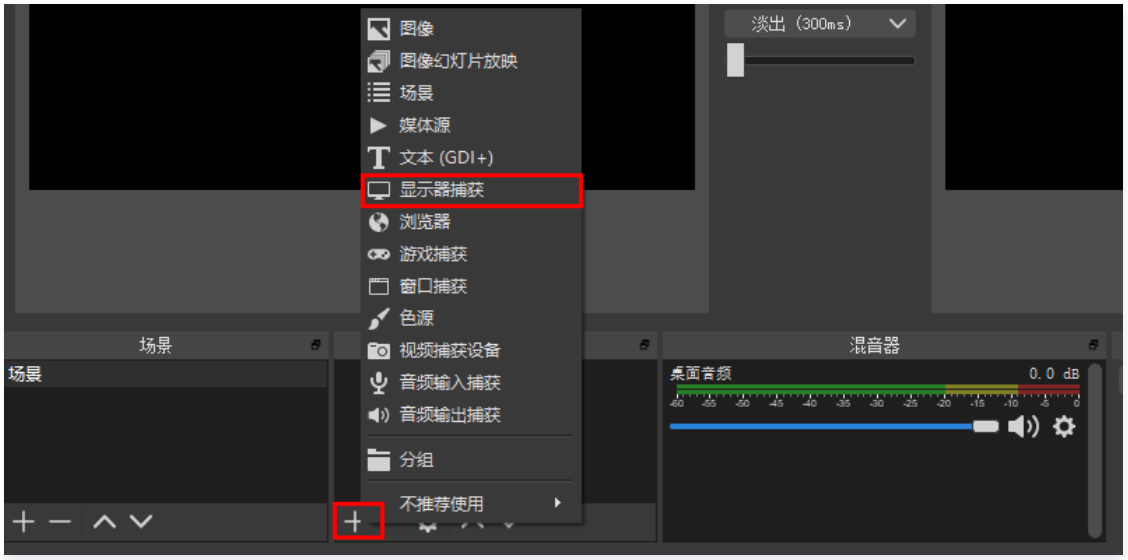
安装并打开 OBS 工具进行下述操作。

步骤1：选择输入源

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

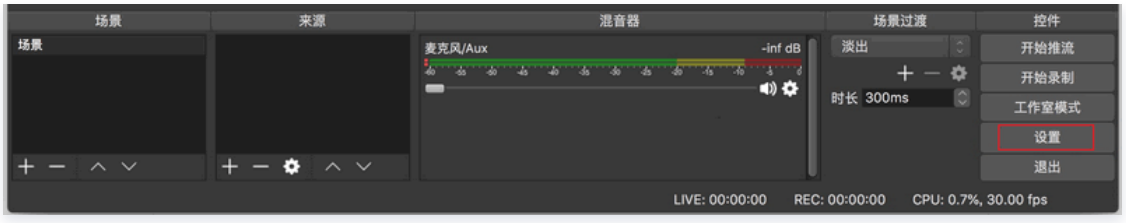
输入源	说明

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
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。



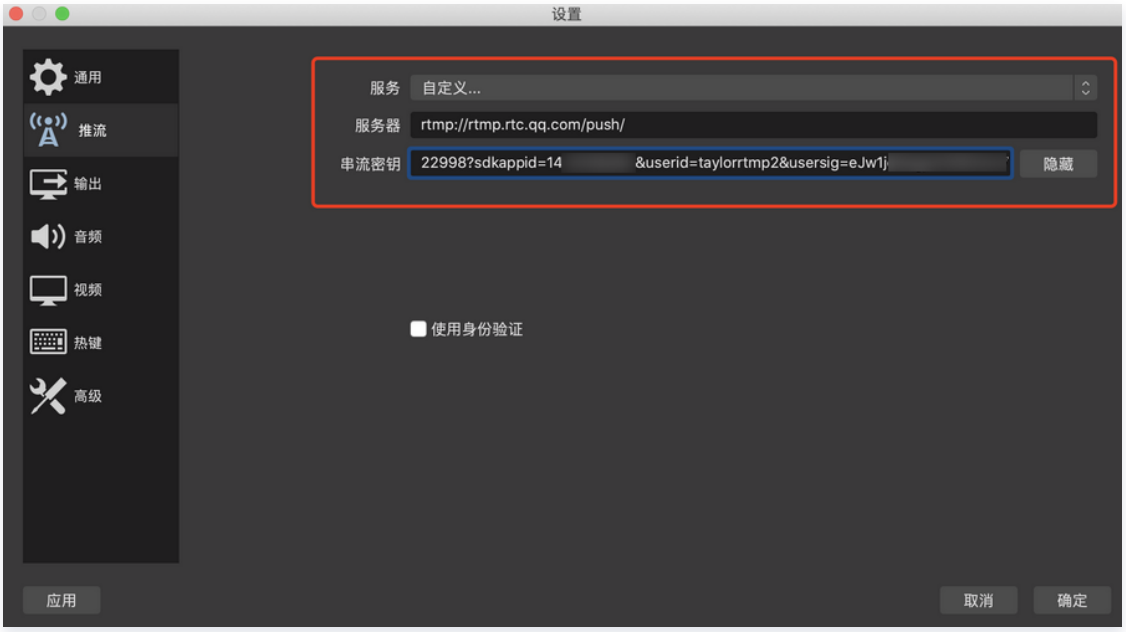
2. 单击**推流**进入推流设置页签，选择服务类型为**自定义**。
3. 服务器填写：`rtmp://rtmp.rtc.qq.com/push/`。
4. 填写串流密钥格式如下：

```
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，参考流地址生成章节。例如：

```
hello-string-room?
sdkappid=140*****66&userid=******rtmp2&usersig=eJw1jdE**************ZLgi5UAgOzoMhrayt*cjbmiCJ699T0
9juc833IMT94Ld7I0iHZqVDzvVAqkZsG-
```

```
IKlzLiXOnEhswHu1iUyTc9pv*****D8MQwoA496Ke6U1ip4EAH4UMc5H9pSmv6MeTBWLamhwFnWRBZ8qKGRj8Yp-
wVbv*mGMVZqS7w-mMDQL
```

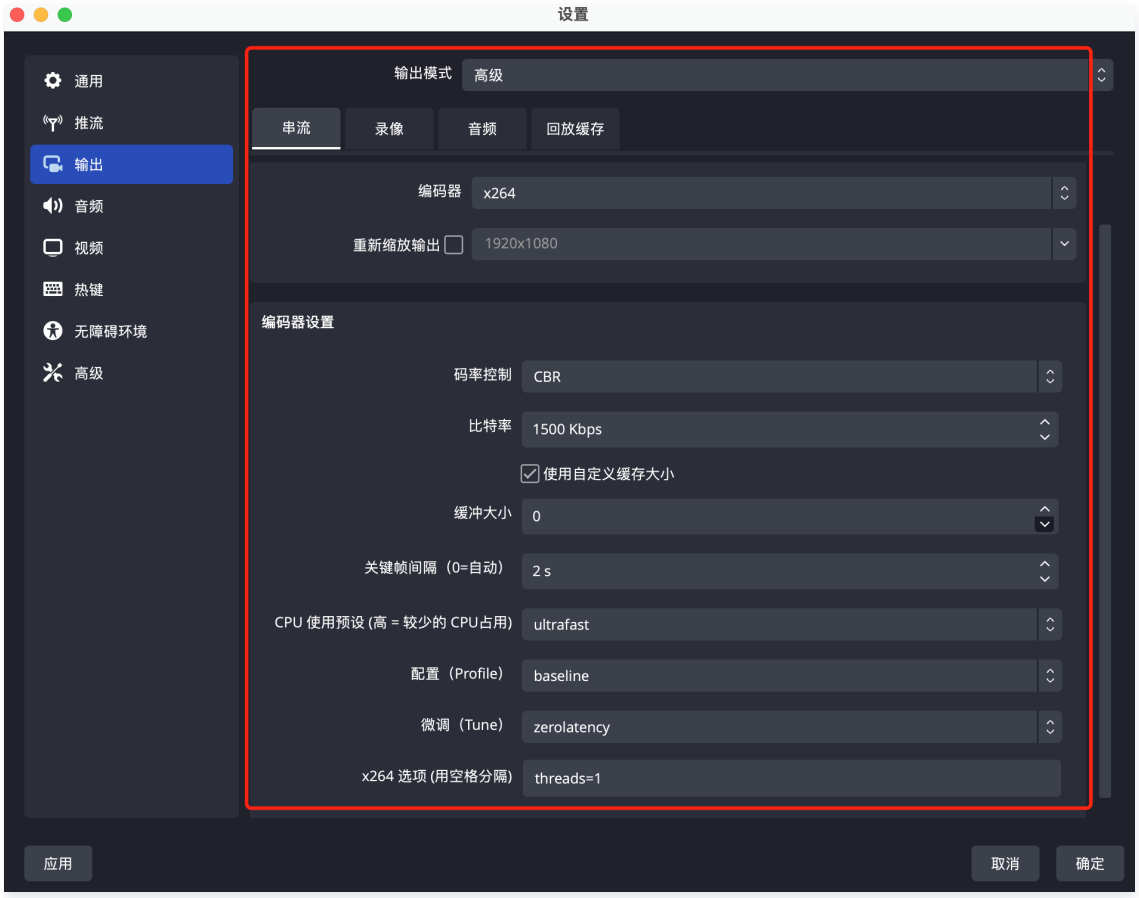


步骤3：设置输出

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。

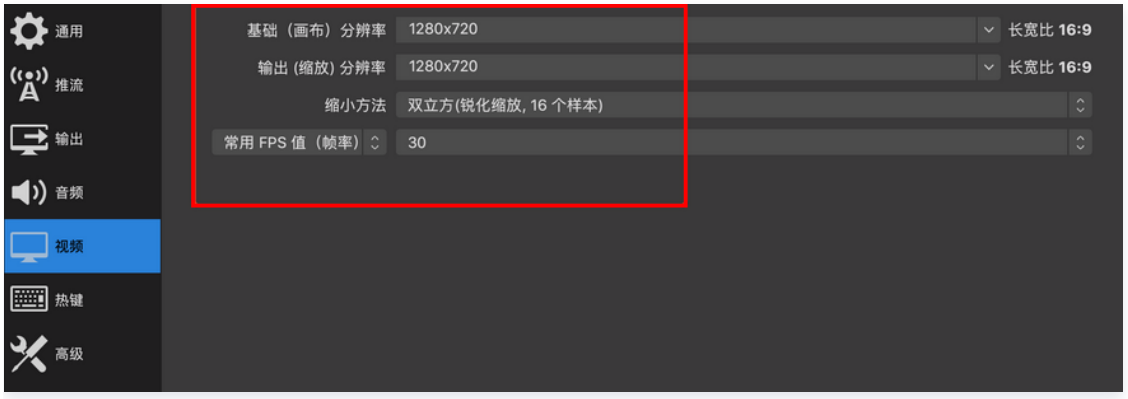
1. 在**设置**中单击**输出**页签进行配置。
2. 在**输出模式**中选择**高级**，**关键帧间隔建议填写1或2**，CPU 使用预设为 ultrafast，配置选择 baseline，微调选择 zerolatency，x264 选项填写 threads=1，单击**确定**保存设置。

> **警告：**
> 推流需要去除 B 帧，否则推流后连接会被断开，下面的配置选择 baseline 可去除 B 帧。



步骤4：设置视频选项

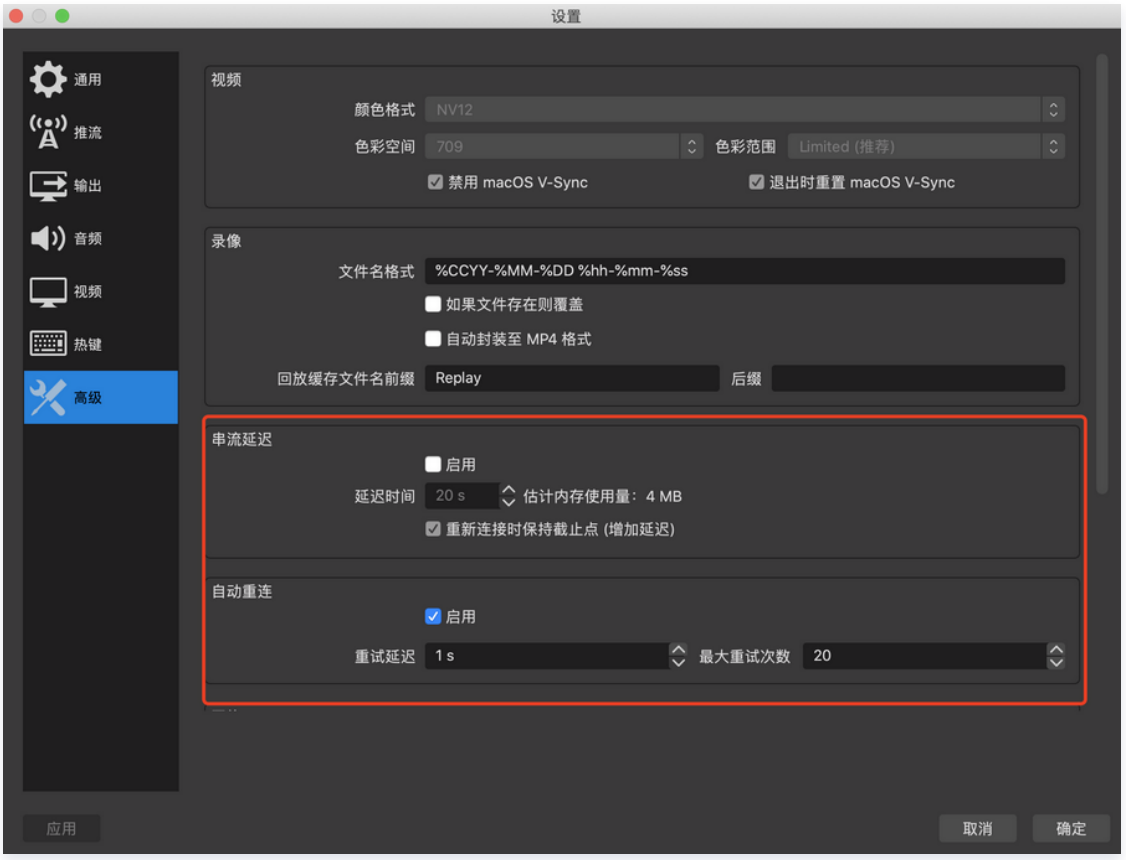
在**设置**中单击**视频**页签，设置分辨率和帧率。分辨率决定了观众看到的画面清晰程度，分辨率越高画面越清晰。FPS 是视频帧率，它控制观看视频的流畅，普通视频帧率有24帧 – 30帧，低于16帧画面看起来有卡顿感，而游戏对帧率要求比较高，一般小于30帧游戏会显得不连贯。



步骤5：设置高级选项

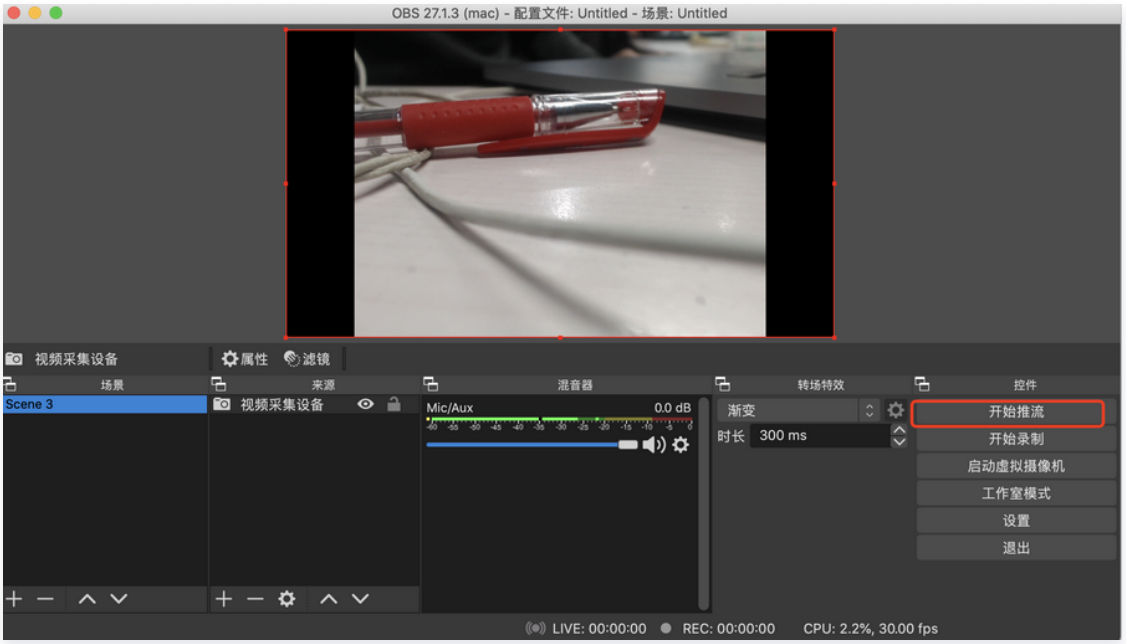
- 建议不启用**串流延迟**以减少端到端延迟。

- 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

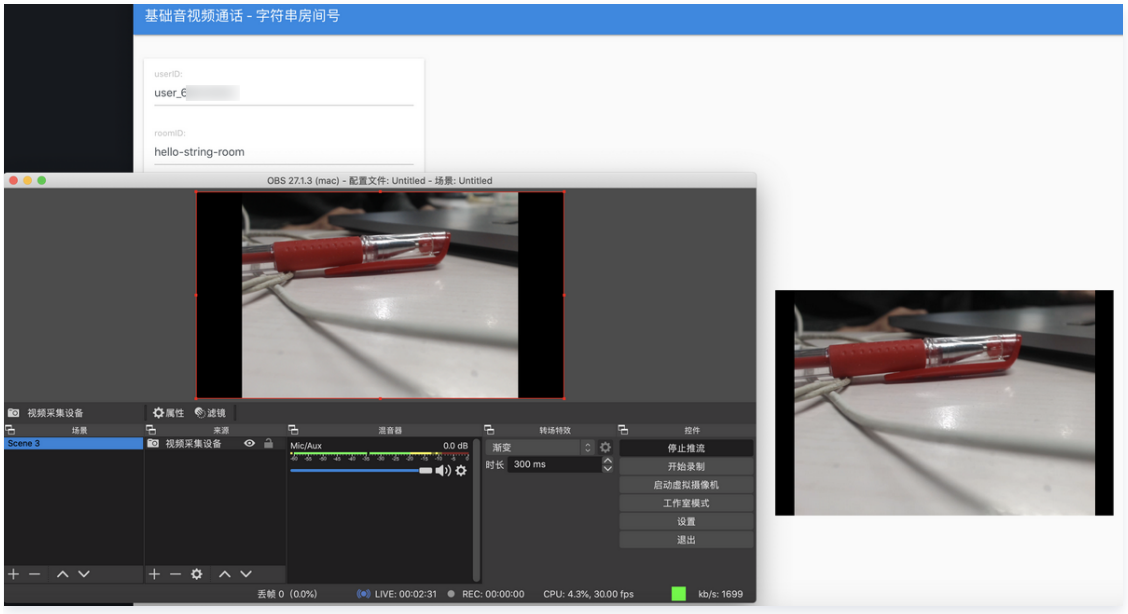
1. 查看 OBS 底部工具栏的**控件**，单击**开始推流**。



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，您也可以选择使用其他端观看。



FFmpeg 推流

如果需要用命令行或其他 RTMP 库推流，使用完整的流地址供 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：

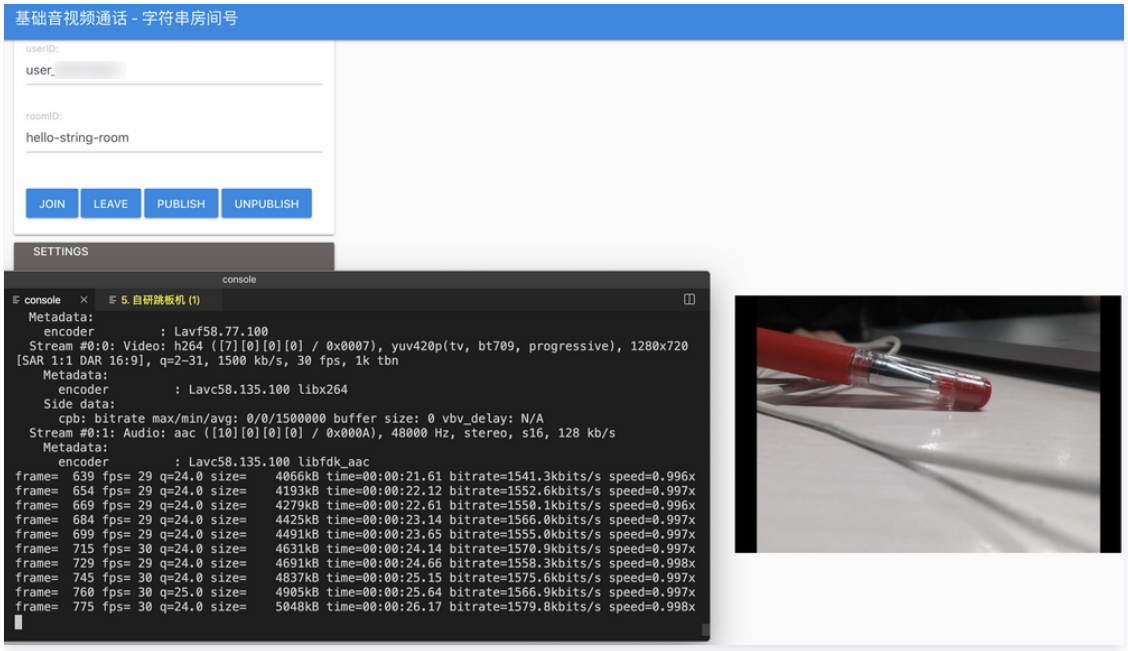
选项	说明
-c:v	视频编码，建议用 libx264
-b:v	视频码率，例如 1500k 表示 1500kbps
-r	视频帧率
-profile:v	视频 profile，指定 baseline 将不编码 B 帧，TRTC 后端不支持 B 帧
-g	GOP 帧数间隔
-c:a	音频编码，建议用 libfdk_aac
-ac	声道数，填2或1
-b:a	音频码率
-f	指定格式，固定填 `flv`，发送到 TRTC 使用 FLV 容器封装

下面的例子是读取文件推到 TRTC，注意 URL 两边加引号。

```
ffmpeg -loglevel debug -re -i sample.flv -c:v libx264 -preset ultrafast -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'
```

其他端观看

下面是使用 Web 端 观看的效果，您也可以选择使用其他端观看。



FAQ

推流失败

常见原因

- 没买套餐包或过期。
- 签名错误或过期。
- 推了B帧（仪表盘上的现象是“推流一秒就结束”），可设置 baseline 编码。

其他原因

- 如果是嵌入式硬件设备推流，可能将 URL 截断。
- 推了 H.265，改为 H.264。
- 端上 set chunk 太大，改为 1360。

卡顿、花屏

观察腾讯云实时音视频 控制台仪表盘，查看推流帧率是否稳定，如果稳定则大概率是播放端的问题，建议排查播放端；**如果帧率不稳**可排查以下几点：

- 检查推流客户端本地 CPU、内存是否高负载。如果使用的是 OBS 推流，观察软件底部状态栏，有丢帧、网络、CPU、帧率等信息。
- 检查本地**网络带宽**是否足够。ping 推流的域名观察 RTT；使用 网络诊断工具 检测推流域名，查看带宽，最好能达到 10M。
- 推流端可尝试降低码率、帧率减少客户端压力，参考正文中 OBS 的设置，720p 建议码率 1500 Kbps。

延迟大

- 拉流端如果使用主播角色，延迟通常低于观众角色，如果不是主播角色可尝试对比一下观察是否有改善。
- 推流端本地编码和网络影响较大。可尝试不同平台测试，如果使用的是 OBS，可尝试 Windows 系统推流；ping 推流域名观察 RTT。

其他端看不到推的流

推流端使用的是字符串房间号，拉流端使用了数字房间号，修改拉流端，改为字符串房间号进房。

频繁断开重推

- 用户名重名，两处互踢引起，请确保单个 sdkappid 下的用户名 userid 全局唯一。
- 推了 B 帧，可设置 baseline 编码。

服务端回调

RTMP 推流用户也是 TRTC 房间中的一个用户，和其他端用户没有本质区别，参见 TRTC 事件回调 。

使用业务侧域名

设置业务侧域名 CNAME 到官方域名，后续也建议这样使用。

资源访问管理
访问管理综述

最近更新时间：2024-12-04 16:01:22

> **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：2022-10-10 11:06:29

> **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
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	获取应用列表

	• 监控仪表盘 • 开发辅助 > UserSig 生成&校验 • 应用管理	
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
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
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：2025-06-19 15:15:22

> **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账号

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：
   3.1 搜索并勾选预设策略 `QcloudTRTCFullAccess` 。
   3.2 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账号

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。

解除子账号的 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：2025-07-02 20:13:14

> **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级别。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**即可完成自定义策略的创建。

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

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账号的方法同 将 TRTC 全读写访问权限授予已存在的子账号 。

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

客户端
实现 AI 降噪

最近更新时间：2025-06-05 18:23:52

AI 降噪技术，源于腾讯天籁实验室的 AI 算法，能够智能地检测和去除传播信号中混杂的噪声干扰。这一技术显著提高了语音质量，增强了声音的清晰度，并改善了用户的听感体验。它使用户在办公室、网吧、商场、户外等多种环境中都能享受到清晰、稳定的声音体验。

在线体验

您也可以进入我们的 实时音视频体验馆，在线体验 AI 降噪能力带来的优秀声音效果。

前提条件

- 登录 TRTC 控制台，开通 TRTC 服务并 创建应用 。
- 前往 TRTC 购买页为指定应用开通包月套餐才可使用，详情请参见 包月套餐计费说明，并前往 控制台 > 功能配置 > 增值功能 开启 AI 降噪功能。

功能说明

Android

您只需要调用 startLocalAudio 时选择 TRTCAudioQualitySpeech 参数， 即可在应用中享受高质量的降噪效果。

```
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );
```

iOS

您只需要调用 startLocalAudio 时选择 TRTCAudioQualitySpeech 参数，即可在应用中享受高质量的降噪效果。

```
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];
```

Mac

您只需要调用 startLocalAudio 时选择 TRTCAudioQualitySpeech 参数，即可在应用中享受高质量的降噪效果。

```
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];
```

Windows

您只需要调用 startLocalAudio 时选择 TRTCAudioQualitySpeech 参数，即可在应用中享受高质量的降噪效果。

```
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->startLocalAudio(TRTCAudioQualitySpeech);
```

Web

部署降噪需要的资源

动态加载文件依赖：AI 降噪插件依赖一些文件。为保证浏览器可以正常加载和运行这些文件，您需要完成以下步骤：

- 将 node_modules/trtc-sdk-v5/plugins/ai-denoiser 目录下的denoiser-wasm.js文件发布至 CDN 或者静态资源服务器中，并且处于同一个公共路径下。如果您需要使用降噪功能，需要传入上述公共路径的 URL，插件会动态加载依赖文件，详细说明 。
- 开启降噪

```
await trtc.startLocalAudio();
await trtc.startPlugin('AIDenoiser', {
  assetsPath: 'XXXXX/assets/', // 例：denoiser-wasm.js 文件存放在 assets 目录下
  sdkAppId: 123456,
  userId: 'user_123',
  userSig: 'XXXXXXXX'
});
```

- 关闭降噪

```
await trtc.stopPlugin('AIDenoiser');
```

使用虚拟背景
iOS&Android

最近更新时间：2025-06-17 11:51:22

功能描述

虚拟背景技术是一种利用先进的图像处理算法，将视频或照片中的人物与背景分离，并能够对背景进行虚化或替换的技术。TRTC 支持 iOS 和 Android 端使用虚拟背景能力**限时内测中**，该能力在视频会议、直播、摄影后期处理等领域有着广泛的应用。
两大核心功能分别是**背景虚化**和**背景替换**。
背景虚化：可以实现让背景变得模糊，让观看者的注意力更集中在视频中的人物上。
背景替换：可以将人物完全从原始背景中移除，并放到一个新的背景前面。

开通条件

- 需要使用虚拟背景功能的 SDKAppID，请确保已开通 **TRTC 旗舰版包月套餐**。包月套餐相关说明请参见文档 包月套餐计费说明 。
- TRTC SDK **Professional** 版本 ≥ **11.8**。

效果示例

原始图片	背景虚化	背景图片
		

开通指引

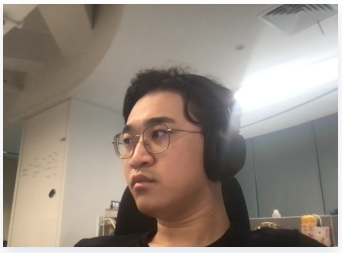
该能力**仅在购买 TRTC 旗舰版套餐后提供**，如您已经完成购买，请 联系我们 协助开通。

Web

最近更新时间：2024-07-09 14:42:21

功能描述

本文将介绍如何在通话过程中实现虚拟背景的功能。功能展示如下：

原始摄像头	背景虚化	背景图片
		

前提条件

- 需要使用虚拟背景功能的 SDKAppID，请确保已开通 **TRTC 旗舰版包月套餐**。包月套餐相关说明请参见文档 包月套餐计费说明 。
- TRTC Web SDK 版本 ⩾ 5.2.0。
- Web 平台各系统及配置要求如下表：

平台	操作系统	浏览器版本	fps	推荐配置	备注
Web	Windows	• Chrome 90+ • Firefox 90+ • Edge 97+	30	• 内存：16GB • CPU：i5-10500 • GPU：独显 2GB	建议使用最新版 Chrome 浏览器（**开启浏览器硬件加速**）
			15	• 内存：8GB • CPU：i3-8300 • GPU：intel 核显 1GB	
	Mac	• Chrome 98+ • Firefox 96+ • Safari 14+	30	• 2019年 MacBook • 内存：16GB(2667MHz) • CPU：i7(6核 2.60GHz) • GPU：AMD Radeon 5300M	

实现流程

1. 引入并注册插件

```
import { VirtualBackground } from 'trtc-sdk-v5/plugins/video-effect/virtual-background';

let trtc = TRTC.create({ plugins: [VirtualBackground] });
```

2. 开启本地摄像头

```
await trtc.startLocalVideo();
```

3. 开启虚拟背景插件

```
await trtc.startPlugin('VirtualBackground', {
  sdkAppId: 123123,
  userId: 'userID_123',
  userSig: 'your_userSig'
});
```

4. 按需更新参数

```
// 改为图片背景
await trtc.updatePlugin('VirtualBackground', {
  type: 'image',
  src: 'https://picsum.photos/seed/picsum/200/300'
});
```

5. 关闭虚拟背景

```
await trtc.stopPlugin('VirtualBackground');
```

API 说明

trtc.startPlugin('VirtualBackground', options)

用于开启虚拟背景。

options

Name	Type	Attributes	Description
sdkAppId	number	必填	当前应用 ID
userId	string	必填	当前用户 ID
userSig	string	必填	用户 ID 对应的 UserSig
type	string	选填	• `image` 图片背景 • `blur` 虚化背景（默认）
src	string	type 为 `image` 时必填	图片地址，如 `https://picsum.photos/seed/picsum/200/300`
onError	(event) => {}	选填	运行过程中发生错误的回调 • event.extraCode=10000003 渲染耗时长 • event.extraCode=10000006 浏览器特性支持不足，可能会出现卡顿情况 推荐处理方法可参考 常见问题

Example:

```
await trtc.startPlugin('VirtualBackground', {
  sdkAppId: 123123,
  userId: 'userID_123',
  userSig: 'your_userSig',
  type: 'image',
  src: 'https://picsum.photos/seed/picsum/200/300'
});
```

trtc.updatePlugin('VirtualBackground', options)

可修改虚拟背景参数。

options

Name	Type	Attributes	Description
type	string	选填	• `image` 图片背景 • `blur` 虚化背景（默认）
src	string	type 为 `image` 时必填	图片地址，如 `https://picsum.photos/seed/picsum/200/300`

Example:

```
await trtc.updatePlugin('VirtualBackground', {
  type: 'blur'
});
```

trtc.stopPlugin('VirtualBackground')

关闭虚拟背景。

常见问题

1. 在 Chrome 中运行 Demo 发现画面颠倒且卡顿?

本插件使用 GPU 进行加速，您需要在浏览器设置中找到使用硬件加速模式并启用。可以将 `chrome://settings/system` 复制到浏览器地址栏，并且打开硬件加速模式。

2. 当设备性能不足造成延迟高，提示渲染耗时长?

可通过监听事件，降低视频分辨率或者帧率。

```
function onError(event) {
  const { extraCode } = event;
  if (extraCode === 10000003 || extraCode === 10000006) {
    // 降低分辨率帧率
    await trtc.updateLocalVideo({
      option: {
        profile: '480p_2'
      },
    });
    // await trtc.stopPlugin('VirtualBackground'); // 或者关闭插件
  }
}

await trtc.startPlugin('VirtualBackground', {
  ...// 其他参数
  onError,
});
```

使用美颜特效
腾讯特效引擎

最近更新时间：2022-10-10 11:06:26

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：2025-07-02 20:13:14

腾讯特效 SDK（移动端/PC端）能力以套餐和原子能力的形式提供。分为 4 个系列：A 系列基础套餐、S 系列高级套餐、V 系列虚拟人套餐 和 X 系列原子能力。不同系列的不同套餐对应不同功能；原子能力 提供单算法能力，集成更加灵活，业务拓展性高。
套餐与原子能力支持的功能详情如下表，更多下载说明请参见 SDK 下载。

> **说明：**
> Web 美颜特效提供的能力以套餐的形式，具体套餐内容可参见 价格说明；Web端 SDK 提供 NPM 包和 JS 文件两种接入方式供客户选择，更多详情参考 Web 美颜 SDK 接入。

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 20 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		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	S1 – 07
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	✓	✓	✓	✓	✓	✓
	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20款通用滤镜）	✓	✓	✓	✓	✓	✓
	贴纸 （赠送 10 款指定 2D 通用贴纸）	–	✓	✓	✓	✓	✓
	高级贴纸 （赠送 3 款指定 3D 通用贴纸）	–	✓	✓	✓	✓	✓
	美妆 （赠送 3 款指定整妆）	–	✓	✓	✓	✓	✓
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）	–	–	–	✓	✓	✓
	美形美体 一键瘦身、长腿、瘦腿、瘦肩、小头	–	–	–	–	–	✓
SDK 下载		下载					

V 系列虚拟人套餐

V 系列虚拟人套餐提供虚拟形象 Animoji、形象制作（捏脸）与驱动等功能，适用于虚拟社交、虚拟直播等场景。

套餐功能		套餐编号		
		V1 – 00	V1 – 01	V1 – 02
基础功能	**虚拟形象** 自研3D渲染轻量引擎	✓	✓	✓
	捏脸 DIY 支持眼、鼻、嘴、脸型、头发等50+细节维度	✓	–	✓
	面部点位识别与驱动 人脸256关键点识别跟踪、52种面部表情绑定和驱动	–	✓	✓
SDK 下载	iOS & Android	下载		

X 系列原子能力

X 系列提供单独的算法能力，集成更加灵活，业务拓展性更高，适用于对算法能力有需求的客户。

功能	能力编号	
	X1 – 01	X1 – 02
功能名称	人像分割	人脸点位
功能详解	在直播、会议等场景实现虚拟背景，实时精准分割，支持自定义背景	人脸检测（识别人脸出框、多人脸、面部遮挡），256个面部关键点位识别与输出
SDK 下载	下载	

SDK 集成指引（iOS）

最近更新时间：2025-06-26 16:19:12

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. **如果使用的 XMagic SDK 版本在2.5.0之前，**导入 SDK 目录中的 `libpag.framework`、`Masonry.framework`、`XMagic.framework`、`YTCommonXMagic.framework`。**如果使用的 XMagic SDK 版本在2.5.1及以后，**导入 SDK 目录中的 `libpag.framework`、`Masonry.framework`、`XMagic.framework`、`YTCommonXMagic.framework`、`Audio2Exp.framework`、`TEFFmpeg.framework`。
3. framework 签名 **General-->Masonry.framework** 和 **libpag.framework** 选 **Embed & Sign**。**YTCommonXMagic.framework** 在版本2.5.1之前选 **Do Not Embed**，在版本2.5.1及以后选 **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 • Audio2Exp（xmagic sdk version在2.5.1及以后的版本才有） • TEFFmpeg（xmagic sdk version在2.5.1及以后的版本才有）

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` 类相关示例代码。

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。SDK版本在2.5.1以前，`TELicenseCheck.h` 在 `XMagic.framework` 里面；SDK版本在2.5.1及以后，`TELicenseCheck.h` 在 `YTCommonXMagic.framework` 里面。

```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authresult, NSString
* _Nonnull 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
-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:NSUTF8StringEncoding 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 withData:(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：2023-03-09 16:02:17

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 程中的 xmagic 模块引 到实际项 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 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 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,
TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCVideoFrameListener() {
 @Override
 public void onGLContextCreated() {
 }
 @Override
 public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame, TRTCCloudDef.TRTCVideoFrame
dstFrame) {
 }
 @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.h
eight);
```

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。**

SDK 集成指引（Web）

最近更新时间：2024-12-20 17:09:44

> **说明：**
> 本教程基于 5.x TRTC Web SDK 实现，若您使用 4.x 版本 SDK，可参见 接入指南 。

准备工作

- 请阅读 Web 美颜特效 SDK 接入指南，熟悉 SDK 基本用法。
- 请阅读 TRTC 快速集成(Web)，了解 TRTC Web SDK 基本用法，并完成基础设置。
- 请阅读 TRTC 快速跑通 Web Demo，并先尝试在本地项目中运行 TRTC Web Demo。

开始使用

步骤1：Web 美颜特效 SDK 引入

由于 TRTC 的 Web Demo 项目已经做的比较完善，我们在此基础上进行少量改造即可。
在页面（PC Web 端）中引入 js 脚本：

```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resources/latest/webar-sdk.umd.js"></script>
```

> **注意：**
> 这里是示例项目，为了方便使用 script 标签方式引入，您也可以参见 接入指南 中的方法，用 npm 包的方式引入。

步骤2：理解 TRTC 流初始化逻辑

1. 在 TRTC 的 Demo 项目里，查看 TRTC 进房过程，进房后，TRTC 通过 startLocalVideo 和 startLocalAudio 方法对本地设备进行采集，创建流对象并发布到房前房间中：

```
const trtc = TRTC.create();
await trtc.enterRoom({ roomId: 8888, sdkAppId, userId, userSig });
// 采集默认麦克风并发布
await trtc.startLocalAudio();
// 采集cameraList的摄像头并发布
await trtc.startLocalVideo({
  view: document.getElementById("localVideo"), // 在 id 为 localVideo 的元素中预览视频
});
```

   以上为最基本的采集本地音视频流并发布到指定房间中的方式。
2. TRTC 提供了 updateLocalVideo 接口，用于更新视频流，我们可以使用自定义采集的方式，获取美颜处理过的流，传递给此接口使用。

```
// 获取自定义的stream
const stream = await ar.getOutput();

// 更新 trtc 的视频流
await trtc.updateLocalVideo({
  option: { videoTrack: mediaStream.getVideoTracks()[0] },
});
```

3. 为了获取经过美颜处理的自定义流，我们需要先 初始化 Web 美颜特效 SDK 。

步骤3：初始化 Web 美颜特效 SDK

示例代码如下：

```
const { ArSdk } = window.AR

/** ----- 鉴权配置 ----- */

/**
 * 腾讯云账号 APPID
 *
 * 进入[腾讯云账号中心](https://console.cloud.tencent.com/developer) 即可查看 APPID
 */
const APPID = ''; // 此处请填写您自己的参数

/**
 * Web LicenseKey
 *
 * 登录音视频终端 SDK 控制台的[Web License 管理](https://console.cloud.tencent.com/vcube/web)，创建项目即可获得 LicenseKey
 */
const LICENSE_KEY = ''; // 此处请填写您自己的参数

/**
 * 计算签名用的密钥 Token
 *
 * 注意：此处仅用于 DEMO 调试，正式环境中请将 Token 保管在服务端，签名方法迁移到服务端实现，通过接口提供，前端调用拉取签名，参考
 * [签名方法](https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90.8D.E6.96.B9.E6.B3.95)
 */
const token = ''; // 此处请填写您自己的参数

/** ---------------------- */

/**
 * 定义获取签名方法
 *
 * 注意：此处方案仅适用于 DEMO 调试，正式环境中签名方法推荐在服务端实现，通过接口提供，前端调用拉取签名
 * 如：
 * async function () {
 *  return fetch('http://xxx.com/get-ar-sign').then(res => res.json());
 * };
 */
const getSignature = function () {
    const timestamp = Math.round(new Date().getTime() / 1000);
    const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
    return { signature, timestamp };
};

// 获取 trtc 未处理过的的视频流
const arInputStream = new MediaStream([trtc.getVideoTrack()]);

// ar sdk 基础配置参数
const config = {
    input: arInputStream,
    auth: {
        licenseKey: LICENSE_KEY,
        appId: APPID,
        authFunc: getSignature
```

```
    },
    // 初始美颜效果（可选参数）
    beautify: {
        whiten: 0.1, // 美白 0-1
        dermabrasion: 0.5, // 磨皮 0-1
        lift: 0.3, // 瘦脸 0-1
        shave: 0, // 削脸 0-1
        eye: 0, // 大眼 0-1
        chin: 0, // 下巴 0-1
    }
}

// config 传入 ar sdk
const ar = new ArSdk(config);

// created回调里可以获取内置特效与滤镜列表进行界面展示
ar.on('created', () => {
    // 获取内置美妆、贴纸
    ar.getEffectList({
        Type: 'Preset'
    }).then((res) => {
        const list = res.map(item => ({
            name: item.Name,
            id: item.EffectId,
            cover: item.CoverUrl,
            url: item.Url,
            label: item.Label,
            type: item.PresetType,
        }));
        const makeupList = list.filter(item=>item.label.indexOf('美妆')>=0)
        const stickerList = list.filter(item=>item.label.indexOf('贴纸')>=0)

    }).catch((e) => {
        console.log(e);
    });
    // 获取内置滤镜
    ar.getCommonFilter().then((res) => {
        const filterList = res.map(item => ({
            name: item.Name,
            id: item.EffectId,
            cover: item.CoverUrl,
            url: item.Url,
            label: item.Label,
            type: item.PresetType,
        }));
    }).catch((e) => {
        console.log(e);
    });
});

ar.on('ready', (e) => {
    // 在 ready 回调里及该事件之后，可使用下述接口来设置相应的美颜特效
    // ar.setBeautify() 设置美颜
    // ar.setEffect() 设置美妆、贴纸
    // ar.setFilter() 设置滤镜

    // 更新 trtc 的视频流
    const mediaStream = await ar.getOutput();
```

```
    await trtc.updateLocalVideo({
      option: { videoTrack: mediaStream.getVideoTracks()[0] },
    });
});

ar.on('error', (e) => {
    console.log(e);
});
```

上述代码对 Web 美颜特效 SDK 进行了初始化配置，美颜 SDK 的输入为 TRTC 对象 getVideoTrack 接口获取的未处理过的视频流。

步骤4：更新 TRTC 的流

Web 美颜特效 SDK 初始化完成后就可以使用 `getOutput` 方法获取输出的流，再调用 TRTC 实例的 updateLocalVideo 接口，更新本地流并发布到房间中。

```
const mediaStream = await ar.getOutput();
// 更新 trtc 的视频流
await trtc.updateLocalVideo({
  option: { videoTrack: mediaStream.getVideoTracks()[0] },
});
```

步骤5：运行 Demo，体验效果

> ⚠ **注意：**
> 示例项目需您自行启动本机 Web 服务，并保证通过指定端口号可访问到 HTML 文件（示例代码位于 `TRTC_Web(5.x)` 文件夹，运行 `quick-demo-js/index.html` ）。

您在进入房间后，等待短暂的美颜初始化后，便可以查看到实际的美颜效果，成功后可以新开浏览器标签页进入刚才创建的房间模拟其他人加入房间的效果。

示例代码

您可以下载 示例代码 解压后查看，美颜相关的主要改动部分在 TRTC_Web(5.x) 文件夹， `quick-demo-js/index.html` 和 `quick-demo-js/js/index.js` 中。请提前申请好 TRTC 密钥 及 Web 美颜特效 License 相关信息。

SDK 集成指引（Flutter）

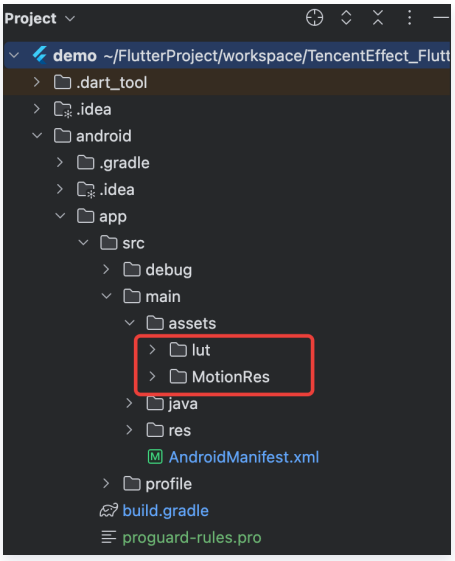
最近更新时间：2025-04-10 11:03:42

步骤一：集成腾讯特效资源

1. 下载 Demo 工程。
2. 迁移腾讯特效资源

Android

1. 在您工程下的 `android/app` 模块下找到 `src/main/assets` 文件夹，将 demo工程中 `demo/android/app/src/main/assets` 中的 lut 和 MotionRes 文件夹复制到您工程的 `android/app/src/main/assets` 中，如果您的工程没有 assets 文件夹可以手动创建一个。



2. 在您项目的 `android/app/build.gradle` 文件中，添加 Android 端美颜 SDK 的依赖。具体依赖根据您选择的套餐包而定，例如您选择的是S1-04套餐，则添加如下：

```
dependencies {
    implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

> **说明：**
> 各套餐对应的 maven 地址，请参见 文档，SDK 的最新版本号可以在 版本历史 中查看。

3. 如果您使用的 Android 版美颜 SDK 小于3.9版本，您需要在 app 模块下找到 AndroidManifest.xml 文件，在 application 表填内添加如下标签：

```
    <uses-native-library
            android:name="libOpenCL.so"
            android:required="false" />
        //true 表示libOpenCL是当前app必需的。如果没有此库，系统将不允许app安装
        //false 表示libOpenCL不是当前app必需的。无论有没有此库，都可以正常安装app。如果设备有此库，美颜特效SDK里的GAN类型特效能正常生效（例如童话脸、国漫脸）。如果设备没有此库，GAN类型不会生效，但也不影响SDK内其他功能的使用。
        //关于uses-native-library的说明，请参考Android 官网介绍：https://developer.android.com/guide/topics/manifest/uses-native-library-element
```

添加后如下图：

```
<application>
    <uses-native-library
        android:name="libOpenCL.so"
        android:required="false" />
</application>
```

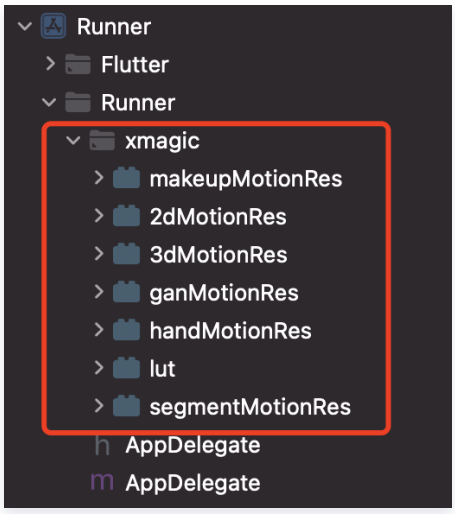
4. 混淆配置

在启用代码优化及混淆功能（minifyEnabled = true）构建 release 包时，编译工具可能移除未通过 Java/Kotlin 层显式调用的代码。若这些代码被 native 层动态调用，将触发 NoSuchMethodError 异常（如 no xxx method）。
建议在 proguard-rules.pro 通过 ProGuard 规则主动保留 Xmagic 模块的必要代码：

```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
-keep class androidx.exifinterface.** { *;}
-keep class com.tencent.effect.** { *;}
```

iOS

将 demo 工程中 ios/Runner 目录下的 xmagic 文件夹复制到您工程中 ios/Runner 目录下，添加后如下图：



注意：
上述从 demo 工程中复制的素材是测试素材，正式素材需要您在购买套餐之后联系腾讯特效美颜的 工作人员 进行获取并重新添加。

步骤二：集成 tencent_effect_flutter

您可以通过以下方式在您的 Flutter 工程中依赖 tencent_effect_flutter

1. 远程依赖：

在您的 pubspec.yaml 文件中添加如下引用：

```
  tencent_effect_flutter:
   git:
     url: https://github.com/Tencent-RTC/TencentEffect_Flutter
```

2. 本地依赖：

从 github 上下载最新版本的 tencent_effect_flutter ,而后在 pubspec.yaml 文件中添加如下引用：

```
tencent_effect_flutter:
    path: path to tencent_effect_flutter
```

步骤三：美颜与 TRTC 关联

Android

在应用的 application 类的 onCreate 方法（或 FlutterActivity 的 onCreate 方法）中添加如下代码：

```
TRTCPlugin.setBeautyProcesserFactory(new XmagicProcesserFactory());
```

iOS

在 ios/Runner 目录下的 AppDelegate 文件中的 didFinishLaunchingWithOptions 方法中添加如下代码：

Swift

```
let instance = XmagicProcesserFactory()
TencentRTCCloud.setBeautyProcesserFactory(factory: instance)
```

Object-C

```
XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
[TencentRTCCloud setBeautyProcesserFactoryWithFactory:instance];
```

步骤四：美颜资源初始化与授权

1. 资源初始化

v0.3.5.0及之后

```
TencentEffectApi.getApi()?.setResourcePath(resourceDir);

TencentEffectApi.getApi()?.initXmagic((result) {
    // TODO
});
```

v0.3.1.1版本及之前

```
TencentEffectApi.getApi()?.initXmagic(dir,(reslut) {
    //TODO
});
```

2. 美颜授权

```
TencentEffectApi.getApi()?.setLicense(licenseKey, licenseUrl, (errorCode, msg) {
    if (errorCode == 0) {
       // Success
    }
});
```

步骤五：开启/关闭美颜

完成上述操作后，您可以通过 TRTC 的隐藏接口来开启/关闭美颜：

```
_enableCustomBeautyByNative(bool open) {
  trtcCloud.callExperimentalAPI("{\"api\": \"enableVideoProcessByNative\", \"params\": {\"enable\":
$open}}");
}
```

> ⚠ **注意：**
> 在页面中开启美颜，关闭相机的时候需要先关闭美颜，开启和关闭是成对使用。

文档参考

以上您已经完成了 TRTC 与腾讯特效美颜的关联，您可以通过以下文档来进一步了解如何使用腾讯特效美颜：

- 腾讯特效美颜 API 文档
- 美颜参数说明

实现跨房连麦PK
Android&IOS

最近更新时间：2024-04-15 22:05:51

前言

默认情况下，只有同一房间内的用户才能进行音视频通话，不同房间的音视频流是互相独立的。但是，您可以通过调用 请求跨房通话（Android 为例）接口，将其他房间内某主播的音视频流发布到您所在的房间。同时，此接口也会将您的音视频流发布到目标主播的房间。
换句话说，您可以使用此接口让两个在不同房间的主播进行音视频流的分享，使每个房间的观众都能观看到这两位主播的音视频。这个功能可以用于实现主播间的 PK 功能。

根据以下文档，您可以实现在两个或更多房间内的主播进行跨房连麦 PK 的场景，并支持不同端的主播进行跨房连麦。

实现方式

在实现跨房连麦之前，需要确认以下实时音视频的一些基础信息：

- userId：用户的唯一标识 ID。
- roomId：数字类型房间 ID。
- strRoomId：字符串类型房间 ID。

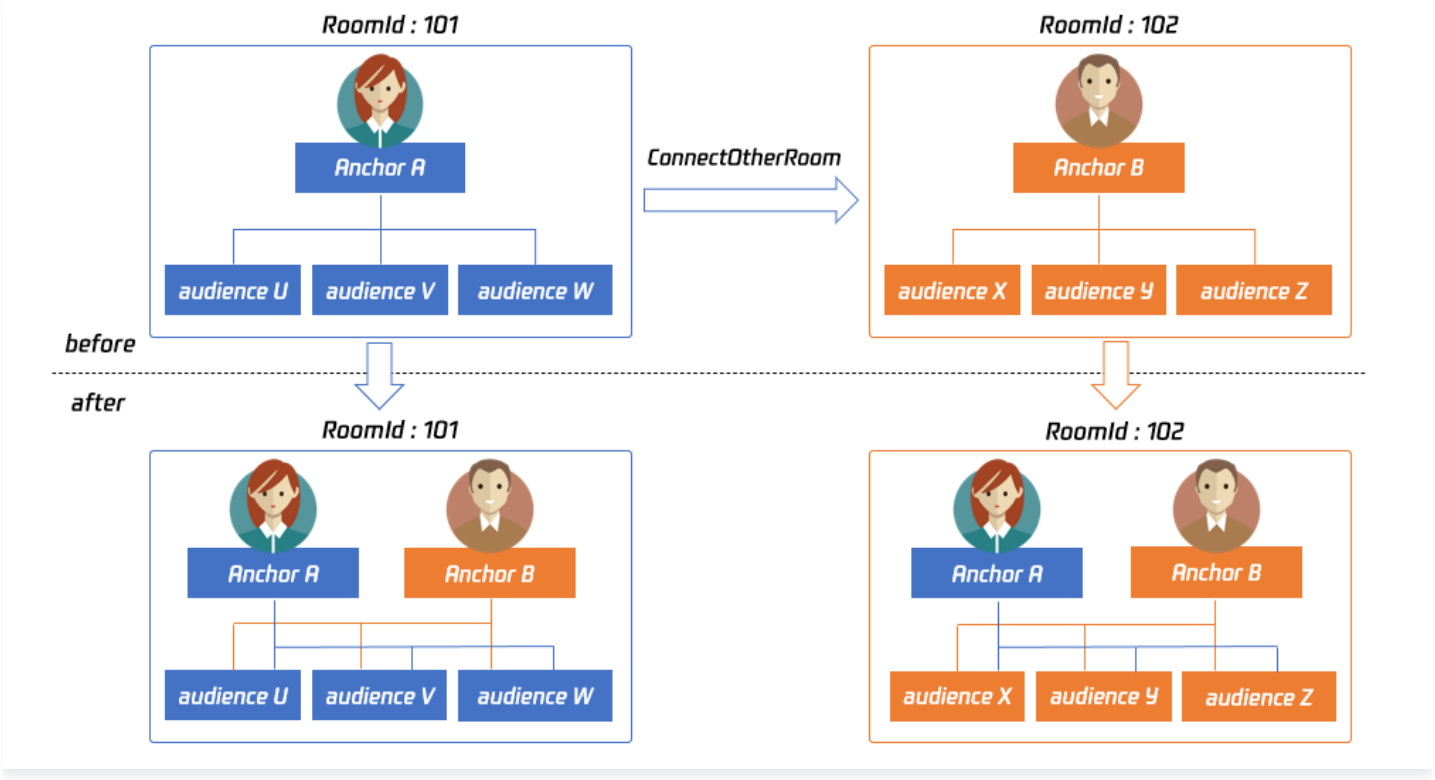
> **注意：**
> 混流双方的房间类型需要相同，不可以混用。

跨房连麦

当房间“101”中的主播 A 通过“connectOtherRoom()”跟房间“102”中的主播 B 建立跨房通话后：

- 房间“101”中的用户都会收到主播 B 的 `onRemoteUserEnterRoom(B)` 和 `onUserVideoAvailable(B,true)` 这两个事件回调，即房间“101”中的用户都可以订阅主播 B 的音视频。
- 房间“102”中的用户都会收到主播 A 的 `onRemoteUserEnterRoom(A)` 和 `onUserVideoAvailable(A,true)` 这两个事件回调，即房间“102”中的用户都可以订阅主播 A 的音视频。

> **说明：**
> 跨房通话的请求结果会通过 TRTCCloudListener 中的 onConnectOtherRoom 回调通知给您。



跨房通话的参数考虑到后续扩展字段的兼容性问题，暂时采用了 JSON 格式的参数：

情况一：数字房间号

如果房间“101”中的主播 A 要跟房间“102”中的主播 B 连麦，那么主播 A 调用该接口时需要传入： `{"roomId": 102, "userId": "userB"}`

示例代码如下：

```
JSONObject jsonObj = new JSONObject();
jsonObj.put("roomId", 102);
jsonObj.put("userId", "userB");
trtc.ConnectOtherRoom(jsonObj.toString());
```

情况二：字符串房间号

如果您使用的是字符串房间号，务必请将 json 中的 `roomId` **替换成** `"strRoomId": {"strRoomId": "102", "userId": "userB"}`

示例代码如下：

```
JSONObject jsonObj = new JSONObject();
jsonObj.put("strRoomId", "102");
jsonObj.put("userId", "userB");
trtc.ConnectOtherRoom(jsonObj.toString());
```

发布跨房音视频到 CDN

对于标准直播拉流（CDN 拉流）的场景来说，主播跨房间连麦之后需要将双方主播连麦的音视频混流后发布到 CDN。如果您有此类需求，参见 发布音视频流到直播 CDN 将多个房间内的主播画面混合后转推至直播 CDN，以实现更多场景的播放观看需求。

Web

最近更新时间：2024-08-29 15:42:32

功能描述

默认情况下，只有同一个房间中的用户之间可以进行音视频通话，不同的房间之间的音视频流是相互隔离的。跨房连麦功能可以让不同房间的用户进行音视频通话。

在直播场景中，通过该功能使得双方主播可以互相通话，双方的观众也可以观看对方主播，本文主要介绍如何使用 CrossRoom 插件实现跨房连麦需求。

使用步骤

> ⚠ **注意：**
> 确保您的 SDK 版本 ⩾ v5.8.0 。

开始跨房连麦

```
import { CrossRoom } from 'trtc-sdk-v5/plugins/cross-room'

const trtc = TRTC.create({ plugins: [CrossRoom] });

await trtc.enterRoom({ sdkAppId, userId: 'user1', userSig, roomId: 7777 });

// 将当前房间 7777 的 user1 和对端房间 8888 的 user2 打通。user2 的流会推到 7777 房间，user1 的流会推到 8888 房间。
await trtc.startPlugin('CrossRoom', {
  roomId: 8888,
  userId: 'user2'
});
```

更新跨房连麦对端主播的 mute 状态

当开启跨房连麦后，对端房间主播的流推到当前房间，本房间内的所有用户都将收到该主播发布的音视频流。

您可以通过调用该接口，限制跨房主播在本房间内的上行能力，禁止或允许跨房主播发布音频/主路视频/辅路视频，该行为会影响房间内的所有用户。

在禁用跨房主播某种上行能力后，本房间内所有用户将无法收到对应音视频流，且无法再订阅对应的音视频。

```
await trtc.updatePlugin('CrossRoom', {
  updateList: [{
    roomId: 8888,
    userId: 'user2'
    muteAudio: true,
    muteVideo: false,
    muteSubStream: false,
  }]
})
```

停止跨房连麦

用户级别的跨房连麦调用下列接口后，会将当前发起跨房连麦的所有跨房取消。

```
await trtc.stopPlugin('CrossRoom')
```

服务端

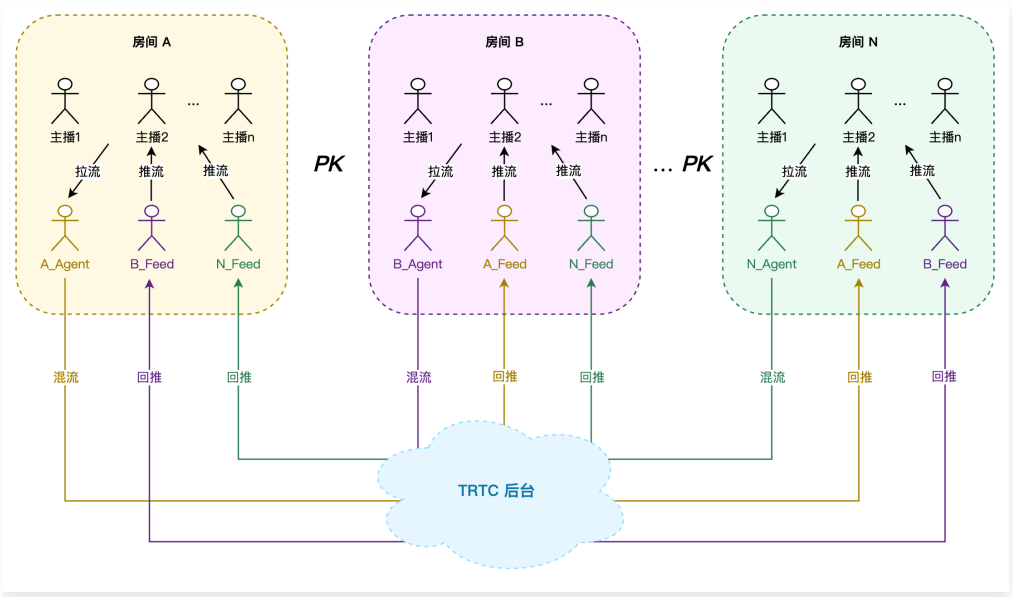
最近更新时间：2024-04-15 22:05:51

适用场景

直播间里，为了增进直播气氛、快速吸粉，主播可以邀请其他直播间的主播进行连麦互动或在线 PK。连麦直播间内的观众可以同时收听或观看多个主播互动音视频内容，能够增强互动直播的趣味性。

方案原理

服务端启动多个混流转推任务，每个转推任务都会拉起一个 Agent 机器人用户进入己方 TRTC 房间进行拉流，同时会拉起一个或多个 Feed 机器人用户将混合的音视频流回推到其他参与跨房 PK 连麦的 TRTC 房间。这样不同房间里的用户就可以通过订阅其他房间混流回推的音视频流，从而实现跨房 PK 连麦。



实现流程

实时互动跨房连麦

1. 步骤一：房间 A 主播向房间 B 主播和房间 N 主播发起跨房 PK 请求（业务信令）。
2. 步骤二：房间 B 主播和房间 N 主播同意跨房 PK 请求（业务信令）。
3. 步骤三：业务后台同时启动 N 个混流回推房间任务 StartPublishCdnStream。

- 任务一：A_Agent 机器人接收 A 房间主播媒体流，经 TRTC 后台混流后由 A_Feed 机器人回推到 B 房间和 N 房间。
- 任务二：B_Agent 机器人接收 B 房间主播媒体流，经 TRTC 后台混流后由 B_Feed 机器人回推到 A 房间和 N 房间。
- 任务 N：N_Agent 机器人接收 N 房间主播媒体流，经 TRTC 后台混流后由 N_Feed 机器人回推到 A 房间和 B 房间。

4. 步骤四：房间 A、房间 B、房间 N 的用户互相拉取房间中混流回推的音视频流，开始进行跨房 PK。
5. 步骤五：跨房 PK 结束，业务后台通过 TaskId 停止 N 个混流回推房间任务 StopPublishCdnStream 。

> **注意：**
> - 本方案最多支持 11 个房间同时进行跨房 PK 连麦，每个房间最多支持 16 个主播同时参与连麦。
> - 机器人 ID 不能与房间内的普通用户 ID 冲突，否则会导致转推任务由于机器人用户被踢出 TRTC 房间而异常结束。

旁路直播跨房连麦

1. 业务后台在每个旁路直播间启动一个旁路转推 CDN 任务 StartPublishCdnStream 。
2. 房间 A 主播向房间 B 主播和房间 N 主播发起跨房 PK 请求（业务信令）。
3. 房间 B 主播和房间 N 主播同意跨房 PK 请求（业务信令）。
4. 业务后台同时启动 N 个混流回推房间任务 StartPublishCdnStream 。
   - 任务一：A_Agent 机器人接收 A 房间主播媒体流，经 TRTC 后台混流后由 A_Feed 机器人回推到 B 房间和 N 房间。
   - 任务二：B_Agent 机器人接收 B 房间主播媒体流，经 TRTC 后台混流后由 B_Feed 机器人回推到 A 房间和 N 房间。
   - 任务 N：N_Agent 机器人接收 N 房间主播媒体流，经 TRTC 后台混流后由 N_Feed 机器人回推到 A 房间和 B 房间。
5. 房间 A、房间 B、房间 N 的用户互相拉取房间中混流回推的音视频流，开始进行跨房 PK。
6. 业务后台更新参与跨房 PK 的房间中原有的旁路转推 CDN 任务 UpdatePublishCdnStream，混合其他房间回推的音视频流。
7. 跨房 PK 结束，业务后台通过 TaskId 停止 N 个混流回推房间任务 StopPublishCdnStream 。
8. 业务后台更新参与跨房 PK 的房间中原有的旁路转推 CDN 任务 UpdatePublishCdnStream，剔除其他房间回推的音视频流。

> **注意：**
> - 根据转推目标的不同，旁路转推 CDN 对应参数 McuPublishCdnParam，回推 TRTC 房间对应参数 McuFeedBackRoomParams 。
> - 若直播间为单主播直播场景，则在启动转推任务时可选择单流旁路转推 SingleSubscribeParams，从而节省混流转码费用。

示例代码

下面以纯音频场景为例，展示跨房 PK 混流回推房间任务的参数体示例。

任务一

```
{
  "SdkAppId": 1400000000,
  "RoomId": "A",
  "RoomIdType": 1,
  "AgentParams": {
    "UserId": "A_Agent",
    "UserSig": "eJwtjMEKgkAUAP9lz2Hv6b40oU...",
    "MaxIdleTime": 50
  },
  "WithTranscoding": 1,
  "AudioParams": {
    "AudioEncode": {
      "Codec": 0,
      "SampleRate": 48000,
      "Channel": 2,
      "BitRate": 64
    }
```

```
  },
  "FeedBackRoomParams": [
    {
      "RoomId": "B",
      "RoomIdType": 1,
      "UserId": "A_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    },
    {
      "RoomId": "N",
      "RoomIdType": 1,
      "UserId": "A_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    }
  ]
}
```

任务二

```
{
  "SdkAppId": 1400000000,
  "RoomId": "B",
  "RoomIdType": 1,
  "AgentParams": {
    "UserId": "B_Agent",
    "UserSig": "eJwtjMEKgkAUAP9lz2Hv6b40oU...",
    "MaxIdleTime": 50
  },
  "WithTranscoding": 1,
  "AudioParams": {
    "AudioEncode": {
      "Codec": 0,
      "SampleRate": 48000,
      "Channel": 2,
      "BitRate": 64
    }
  },
  "FeedBackRoomParams": [
    {
      "RoomId": "A",
      "RoomIdType": 1,
      "UserId": "B_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    },
    {
      "RoomId": "N",
      "RoomIdType": 1,
      "UserId": "B_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    }
  ]
}
```

任务 N

```
{
```

```
  "SdkAppId": 1400000000,
  "RoomId": "N",
  "RoomIdType": 1,
  "AgentParams": {
    "UserId": "N_Agent",
    "UserSig": "eJwtjMEKgkAUAP9lz2Hv6b40oU...",
    "MaxIdleTime": 50
  },
  "WithTranscoding": 1,
  "AudioParams": {
    "AudioEncode": {
      "Codec": 0,
      "SampleRate": 48000,
      "Channel": 2,
      "BitRate": 64
    }
  },
  "FeedBackRoomParams": [
    {
      "RoomId": "A",
      "RoomIdType": 1,
      "UserId": "N_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    },
    {
      "RoomId": "B",
      "RoomIdType": 1,
      "UserId": "N_Feed",
      "UserSig": "eJwtzEELgkAUBOD-sldD3745..."
    }
  ]
}
```

注意：
纯音频场景下，TRTC 后台会默认混合房间内所有主播音频流，亦可通过音频参数 McuAudioParams 指定音频混流黑白名单。

发送和接收消息

最近更新时间：2024-09-02 14:48:42

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

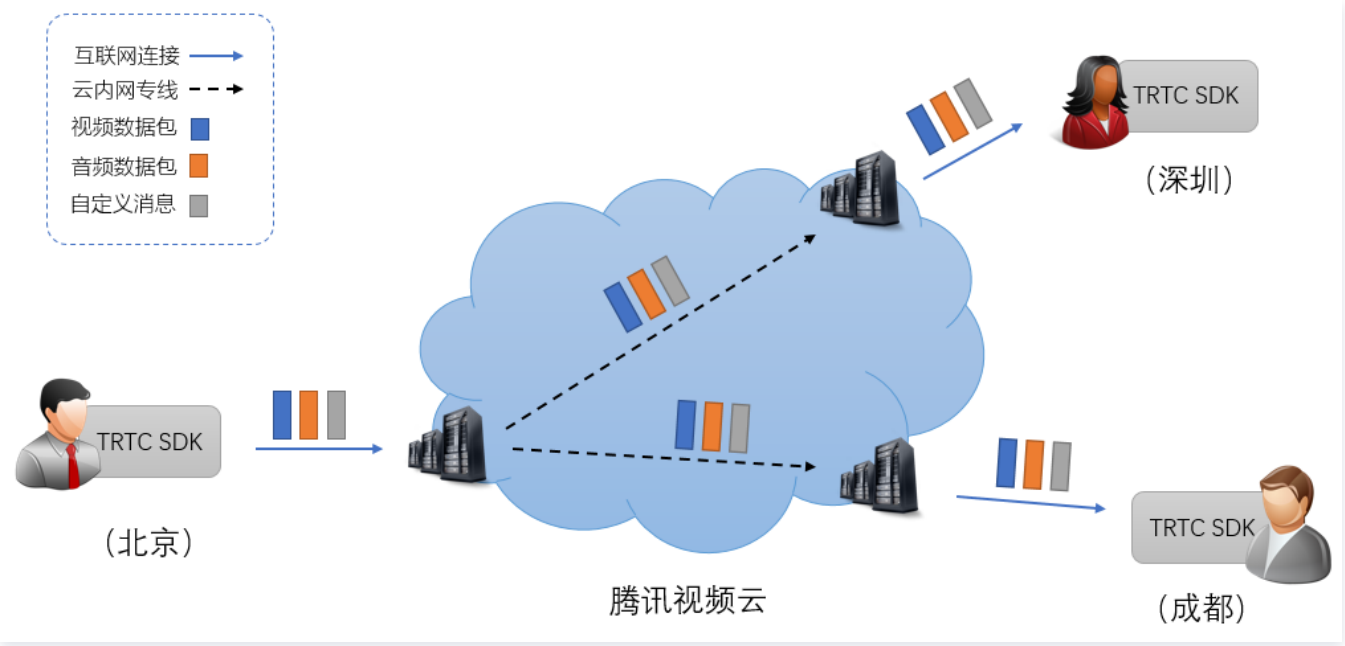
iOS	Android	Mac OS	Windows	Electron	Flutter	微信小程序	Web 端
✓	✓	✓	✓	✓	✓	×	✓

说明：

- Web 端参考接口文档：trtc.sendCustomMessage 。
- 微信小程序端客户，可通过 IM 提供的即时通信来实现。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：**
> 1、请将 reliable 和 ordered 同时设置为 YES 或 NO，暂不支持交叉设置。
> 2、仅**主播身份**可以发送自定义消息。

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);
}
```

Web

```
// send custom message
trtc.sendCustomMessage({
  cmdId: 1,
  data: new TextEncoder().encode('hello').buffer
});
```

Dart

```
//发送自定义消息的示例代码
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";
    // 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, hello, true, true);
  } catch (e) {
    print(e);
  }
}
```

消息接收

当房间中的一个用户通过 `sendCustomCmdMsg` 发出自定义消息后，房间中其他的用户可以通过 SDK 回调中的 `onRecvCustomCmdMsg` 接口来接收这些消息。

Objective-C

```
//接收和处理房间内其他人发送的消息
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UInt32)seq message:
(NSData *)message
{
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId = 1消息
```

```
        break;
    case 2:
        // 处理cmdId = 2消息
        break;
    default:
        break;
    }
}
```

Java

```
//继承 TRTCCloudListener，实现 onRecvCustomCmdMsg 方法接收和处理房间内其他人发送的消息
public void onRecvCustomCmdMsg(String userId, int cmdId, int seq, byte[] message) {
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId = 1消息
        break;
    case 2:
        // 处理cmdId = 2消息
        break;
    default:
        break;

}
```

C++

```
// 接收和处理房间内其他人发送的消息
void TRTCCloudCallbackImpl::onRecvCustomCmdMsg(
                            const char* userId, int32_t cmdId, uint32_t seq, const uint8_t* msg,
uint32_t msgSize)
{
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId = 1消息
        break;
    case 2:
        // 处理cmdId = 2消息
        break;
    default:
        break;
    }
}
```

C#

```
// 接收和处理房间内其他人发送的消息
public void onRecvCustomCmdMsg(string userId, int cmdId, uint seq, byte[] msg, uint msgSize)
{
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId = 1消息
        break;
    case 2:
        // 处理cmdId = 2消息
        break;
    default:
        break;
    }
}
```

Web

```
// receive custom message
trtc.on(TRTC.EVENT.CUSTOM_MESSAGE, event => {
   // event.userId: 远端发消息的 userId
   // event.cmdId: 您自定义的消息 Id
   // event.seq: 消息的序号
   // event.data: 消息内容，ArrayBuffer 类型
   console.log(`received custom msg from ${event.userId}, message: ${new
TextDecoder().decode(event.data)}`)
})
```

Dart

```
// 注册trtc回调
trtcCloud.registerListener(_onRtcListener);

// 实现 onRecvCustomCmdMsg 方法接收和处理房间内其他人发送的消息
_onRtcListener(type, param) async {
  if (type == TRTCCloudListener.onRecvCustomCmdMsg) {
    // 接收到 userId 发送的消息
    String userId = param['userId'];
    // 发送方和接收方协商好的cmdId
    switch (param['cmdID']) {
      case 0:
      // 处理 cmdID = 0 消息
        break;
      case 1:
      // 处理 cmdID = 1 消息
        break;
      case 2:
      // 处理 cmdID = 2 消息
```

```
          break;
        default:
          break;
      }
    }
  }
```

使用限制

由于自定义消息享受比音视频数据更高的传输优先级，如果自定义数据发送过多，音视频数据可能会被干扰到，从而导致画面卡顿或者模糊。所以，我们针对自定义消息的发送进行了如下的频率限制：

- 自定义消息会被云广播给房间内所有用户，所以每秒最多能发送 30 条消息。
- 每个消息包（即 data 的大小）最大为 1 KB，超过则很有可能会被中间路由器或者服务器丢弃。
- 每个客户端每秒最多能发送总计 8 KB 数据，也就是如果每个数据包都是 1KB，那么每秒钟您最多只能发送 8 个数据包。

自定义视频采集和渲染 Android&iOS&Windows&Mac

最近更新时间：2025-01-03 18:03:02

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 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：sendCustomVideoData 。

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,
TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCVideoRenderListener() {
    @Override
```

```
    public void onRenderVideoFrame(String suserId 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:TRTCVideoBufferType_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,
TRTCCloudDef.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：2023-11-23 18:00:42

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。

> **说明：**
> 本教程基于 5.0 TRTC Web SDK 实现，若您使用 4.x.x 版本 SDK，可参考 此教程 。

自定义采集

默认情况下，`trtc.startLocalVideo()` ，`trtc.startLocalAudio()` 是开启摄像头和麦克风采集。
如果您需要自定义采集，可以通过 `trtc.startLocalVideo()` / `trtc.startLocalAudio()` 方法的 option.videoTrack/option.audioTrack 参数来指定。
获取 audioTrack，videoTrack 通常有以下几种方式：

- 通过 getUserMedia 采集摄像头和麦克风。
- 通过 getDisplayMedia 采集屏幕分享。
- 通过 videoElement.captureStream 采集 video 标签中正在播放的音视频。
- 通过 canvas.captureStream 采集 canvas 画布中的动画。

采集 video 标签中正在播放的视频

```
// 检测您当前的浏览器是否支持从 video 元素采集
if (!HTMLVideoElement.prototype.captureStream) {
  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];

trtc.startLocalVideo({ option:{ videoTrack } });
trtc.startLocalAudio({ option:{ audioTrack } });
```

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
if (!HTMLCanvasElement.prototype.captureStream) {
  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];

trtc.startLocalVideo({ option:{ videoTrack } });
```

自定义播放渲染

在正常情况下，在 startLocalVideo() startRemoteVideo() 时，传入 view 参数，SDK 会在指定的 element 标签下，创建 `video` 标签播放视频画面。
如果您需要自定义播放渲染，不需要 SDK 播放视频，可参考如下步骤实现：

- 在 startLocalVideo 或 startRemoteVideo 方法调用时不填 view 参数或 view 参数传入 null
- 通过 TRTC.getVideoTrack() 方法获取相应的 videoTrack
- 利用自己的播放器进行视频的播放渲染。
- 使用这种自定义播放渲染方式后，VIDEO_PLAY_STATE_CHANGED 事件将不会被触发，您需要自行监听视频轨道 MediaStreamTrack 的 `mute/unmute/ended` 等事件来判断当前视频数据流的状态。
- 对于远端视频，还需要监听 REMOTE_VIDEO_AVAILABLE，REMOTE_VIDEO_UNAVAILABLE 事件来处理远端视频的生命周期。

自定义渲染本地视频

```
await trtc.startLocalVideo();
const videoTrack = trtc.getVideoTrack();

// 使用自定义的播放器进行视频播放渲染
const videoElement = document.getElementById('video-element');
videoElement.srcObject = new MediaStream([videoTrack]);
videoElement.play();
```

自定义渲染远端视频

```
trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  // 只拉流，不播放
  await trtc.startRemoteVideo({ userId, streamType })
  const videoTrack = trtc.getVideoTrack({ userId, streamType });

  // 使用自定义的播放器进行视频播放渲染
  const videoElement = document.getElementById('remote-video-element');
  videoElement.srcObject = new MediaStream([videoTrack]);
  videoElement.play();
});
```

自定义音频采集和播放 Android&iOS&Windows&Mac

最近更新时间：2025-03-27 11:23:48

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 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 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：2023-11-23 18:00:42

本文主要介绍自定义采集音频流的高阶用法。暂不支持自定义音频播放。

> **说明：**
> 本教程基于 5.0 TRTC Web SDK 实现，若您使用 4.x.x 版本 SDK，可参考 此教程 。

自定义音频采集

默认情况下 `trtc.startLocalAudio()` 是麦克风采集。如果您需要自定义采集，可以通过 `trtc.startLocalAudio()` 方法的 option.audioTrack 参数来指定。
获取 audioTrack 通常有以下几种方式：

- 通过 getUserMedia 采集麦克风。
- 通过 audioElement.captureStream 采集 audio 标签中正在播放的音频。

采集 audio 标签中正在播放的视频

```
// 检测您当前的浏览器是否支持从 video 元素采集
if (!HTMLAudioElement.prototype.captureStream) {
  console.log('your browser does not support capturing stream from audio element');
  return
}
// 获取您页面在播放的 audio 标签
const audio = document.getElementByID('your-audio-element-ID');
// 从播放的音频采集音频流
const stream = audio.captureStream();
const audioTrack = stream.getAudioTracks()[0];

trtc.startLocalAudio({ option:{ audioTrack } });
```

Flutter

最近更新时间：2023-12-28 17:09:21

本文档主要介绍如何使用 TRTC Flutter SDK 实现自定义音频原数据获取。

获取音频原数据

Flutter TRTC SDK 提供两种音频原数据的获取方式：

- Native 接入。
- 直接使用 Flutter 的 Dart 接口。

由于将高频且庞大的音频原数据从 Native 传输到 Dart 层会耗费较多性能， 我们推荐使用 Native 接入，获取音频原数据。

1. Native 接入

具体接入过程和接入效果可使用 demo 体验。

1.1 在 Native 层监听音频原数据，获取音频原数据。

java

```
void enableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance(getApplicationContext()).setAudioFrameListener(new
AudioFrameListener());
    result.success("");
}

void disableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance(getApplicationContext()).setAudioFrameListener(null);
    result.success("");
}

class AudioFrameListener implements TRTCCloudListener.TRTCAudioFrameListener {
    @Override
    public void onCapturedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
        // TODO
    }

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

    @Override
    public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame, String s) {
        // TODO
    }

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

    @Override
    public void onMixedAllAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
        // TODO
    }

    @Override
```

```
    public void onVoiceEarMonitorAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame) {
        // TODO
    }
}
```

swift

```
let listener = AudioFrameProcessListener()
func enableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance().setAudioFrameDelegate(listener)
    result(nil)
}

func disableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance().setAudioFrameDelegate(nil)
    result(nil)
}

class AudioFrameProcessListener: NSObject, TRTCAudioFrameDelegate {
    func onCapturedAudioFrame(_ frame: TRTCAudioFrame) {
        //MARK: TODO
    }

    func onLocalProcessedAudioFrame(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }

    func onRemoteUserAudioFrame(_ frame: TRTCAudioFrame, userId: String) {
        // MARK: TODO
    }

    func onMixedAllAudioFrame(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }

    func onMixedPlay(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }

    func onVoiceEarMonitorAudioFrame(_ frame: TRTCAudioFrame) {
        // MARK: TODO
    }
}
```

1.2 使用 Method Channel 实现开始/停止获取音频原数据。
步骤一：在 Dart 层实现获取音频原数据的开始/停止接口。

```
final channel = MethodChannel('TRCT_FLUTTER_EXAMPLE');

void enableAudioFrame() async {
   await channel.invokeMethod('enableTRTCAudioFrameDelegate');
}

void disableAudioFrame() async {
   await channel.invokeMethod('disableTRTCAudioFrameDelegate');
}
```

步骤二：在 Native 层实现获取音频原数据的开始/停止接口。

java

```
public class MainActivity extends FlutterActivity {
    private static final String channelName = "TRCT_FLUTTER_EXAMPLE";

    private MethodChannel channel;

    @Override
    public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
        super.configureFlutterEngine(flutterEngine);

        channel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessenger(),
channelName);
        channel.setMethodCallHandler(((call, result) -> {
            switch (call.method) {
                case "enableTRTCAudioFrameDelegate":
                    enableTRTCAudioFrameDelegate();
                    break;
                case "disableTRTCAudioFrameDelegate":
                    disableTRTCAudioFrameDelegate();
                    break;
                default:
                    break;
            }
        }));
    }
}
```

swift

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
    var channel: FlutterMethodChannel?

    override func application(_ application: UIApplication,
                             didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        GeneratedPluginRegistrant.register(with: self)

        guard let controller = window?.rootViewController as? FlutterViewController else {
            fatalError("Invalid root view controller")
        }
        channel = FlutterMethodChannel(name: "TRCT_FLUTTER_EXAMPLE", binaryMessenger:
controller.binaryMessenger)
        channel?.setMethodCallHandler({ [weak self] call, result in
            guard let self = self else { return }
            switch (call.method) {
            case "enableTRTCAudioFrameDelegate":
                self.enableTRTCAudioFrameDelegate()
                break
            case "disableTRTCAudioFrameDelegate":
                self.disableTRTCAudioFrameDelegate()
                break
            default:
                break
```

```
            }
        })
        return super.application(application, didFinishLaunchingWithOptions: launchOptions)
    }
}
```

2. Flutter 层接口接入

目前 Flutter Dart 接口仅支持 onCapturedAudioFrame 接口的使用。具体使用方法如下：

```
TRTCCloud trtcCloud = (await TRTCCloud.sharedInstance())!;

// 开启音频原数据获取
final audioFrameListener = TRTCAudioFrameListener(
  onCapturedAudioFrame: (audioFrame) {
    // TODO
  }
);
trtcCloud.setAudioFrameListener(audioFrameListener);

// 停止音频原数据获取
trtcCloud.setAudioFrameListener(null);
```

分层编码与兴趣区域编码

最近更新时间：2025-06-05 18:23:52

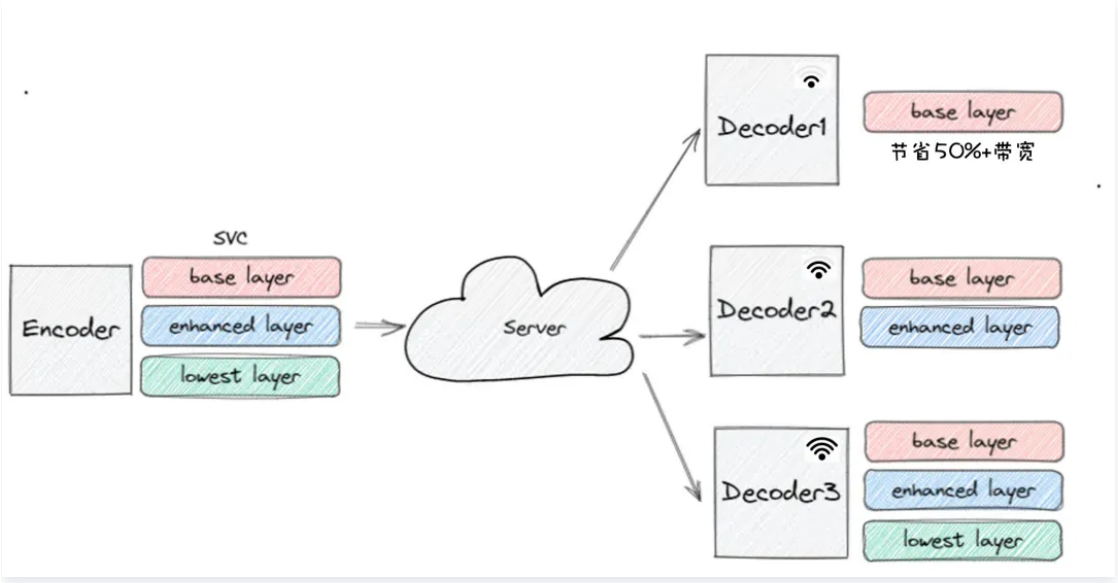
实时音视频（TRTC）技术致力于在各种环境下提供低延时和高质量的视频体验。通过与腾讯多媒体实验室的深入合作，TRTC 推出**可分层视频编码 SVC** 和**兴趣区域编码 ROI** 两项智能编码技术，这些技术旨在帮助用户在不同网络条件下实现更为流畅、高效和稳定的音视频通话体验。

前提条件

- 登录 TRTC 控制台，开通 TRTC 服务并 创建应用。
- 前往 TRTC 购买页，针对需要使用加密能力的 SDKAppid 购买 TRTC **旗舰版或旗舰版 Plus** 包月套餐，以解锁**可分层视频编码**或**兴趣区域编码 ROI** 能力位。包月套餐相关说明请参见文档 包月套餐计费说明。

可分层视频编码 SVC

视频作为二维信息相比于音频数据能传递的信息更多，理论上所需的带宽也更高。这不仅对编码器压缩性有极高的要求，同时因为信息量大，在弱网下丢包的几率也相对更高。因此可分层编码 SVC 功能，在高压缩率的同时最大化弱网抗性，保障弱网环境下依然可以有效地传输视频数据。



可分层编码 SVC 功能在解锁对应能力位后默认自动生效，无需 API 调用。由于此功能会通过 SDK 优化音视频的编码功能，可能会与您集成的第三方 SDK 冲突，订阅 TRTC **旗舰版或旗舰版 Plus** 包月套餐解锁功能后，此功能开关默认关闭，请前往 控制台 > 应用概览 > 增值功能，开启可分层编码 SVC 能力位。

兴趣区域编码 ROI

ROI 功能能够优化码率在重要信息上的分配效率，使人眼感兴趣的区域编码更为清晰。对于 ROI 区域，支持外部传入及内部检测两种模式—如果外部有美颜等前处理模块可使用外部传入模式，避免重复计算；对于大多数的场景，外部并不存在 ROI 检测结果，此时可开启耗时极低的内部检测算法，节省计算资源的同时也能提升编码画质。

通过 ROI 技术，低码率下能显著提升主观质量，高码率下则能够在保证画质相同的情况下节省20%左右的带宽。

实现方式

兴趣区域视频编码 ROI 功能需调用实验性接口（ callExperimentalAPI ，以 Android 为例）设置感兴趣区域，示例：

```
//设置 ROI 参数
{
    "api":"UpdateRoiConfig",
    "params":{
          "roiConfigArray":[
              {
                    "stream":1,
                    "x":0,
                    "y":0,
                    "width":100,
                    "height":100,
                    "level":2
               },
               {
                    "stream":1,
                    "x":20,
                    "y":200,
                    "width":100,
                    "height":100,
                    "level":2
               }
          ]
     }
}
```

字段名	说明
stream	流类型：0大流，1小流，2辅流。
x,y,w,h	roi 区域坐标点，以编码输出的分辨率为参考。
level	[0, 12] , 就是 roi 的强度， 值越大，roi 区域的效果更明显，但非 roi 区域可能会更模糊。

媒体流私有加密

最近更新时间：2025-06-05 18:23:52

功能介绍

在金融等对用户隐私数据要求严苛的行业场景中，为确保用户数据在网络传输中的安全性，以及保障用户信息与数据的绝对安全，常需额外采用媒体流加密手段。实时音视频 TRTC 在既有的默认加密算法基础上，更进一步提供了媒体流私有加密能力，从而为用户数据的安全性提供坚实屏障。

前提条件

- 登录 TRTC 控制台，开通 TRTC 服务并 创建应用 。
- 前往 TRTC 购买页，针对需要使用加密能力的 SDKAppid 购买 TRTC **旗舰版或旗舰版 Plus** 包月套餐，以解锁 **SDK 私有加密**能力位。包月套餐相关说明请参见文档 包月套餐计费说明 。
- 因合规管理要求，开后 SDK 私有加密能力需进行业务信息审核 提交申请，我们将在一个工作日内完成审核，审核通过后即可接入使用（申请开通的 SDKAppid 需订阅旗舰版套餐服务）。

注意事项

- 使用私有加密的 TRTC 音视频通话房间，不支持使用云端录制、旁路推流等媒体服务，不支持服务端本地录制服务。
- 当前仅限 iOS、Android、Windows、macOS 支持，其他平台暂不支持。

实现流程

使用私有加密方案

在加入房间前，调用 `enablePayloadPrivateEncryption` 方法启用私有加密，请参见以下步骤分别生成并设置密钥和盐。

> ⚠ **注意：**
> - 同一房间内所有用户必须使用相同的加密模式、密钥和盐。
> - 用户退出房间后，SDK 会自动关闭私有加密。如需重新开启私有加密，您需要在用户再次加入房间前调用该方法。

生成并设置密钥

1. 在您的服务端，参见以下命令通过 OpenSSL 随机生成 String 型的密钥。

```
// 随机生成一个 string 型、16 字节或 32 字节的密钥，并将该密钥传入 TRTCPayloadPrivateEncryp
openssl rand -hex 16
a2e898d07a304246044f899a16123263

openssl rand -hex 32
8301281ec074a4cb2bd31aa40ad795d15a190d56fb73408db91244c5a3f90a2d
```

> ⚠ **注意：**
> 生成的密匙长度在于您选择的加密算法，若您选择的加密算法为 TRTCEncryptionAlgorithmAes128Gcm ，需生成16字节的密匙，若您选择的加密算法为 TRTCEncryptionAlgorithmAes256Gcm ，需生成 32 字节的密匙。

2. 客户端从服务端获取 String 型密钥，并在调用 `enablePayloadPrivateEncryption` 时传入 SDK。

生成并设置盐

1. 在您的服务端，参见以下命令通过 OpenSSL 随机生成 Base64 编码、32 字节的盐。

```
// 随机生成一个 Base64 编码、32 字节的盐，并将该盐传入 TRTCPayloadPrivateEncryptionConfig
openssl rand -base64 32
3ZZ0nV/rDVUzTa6tXyz+F7rrUYIcxRqX5fiUto/FbZA=
```

2. 客户端从服务端获取 Base64 编码的盐。
3. 客户端将盐值从 Base64 编码解码为长度为 32 的 uint8_t 数组，然后在调用 `enablePayloadPrivateEncryption` 时传入SDK。

示例代码

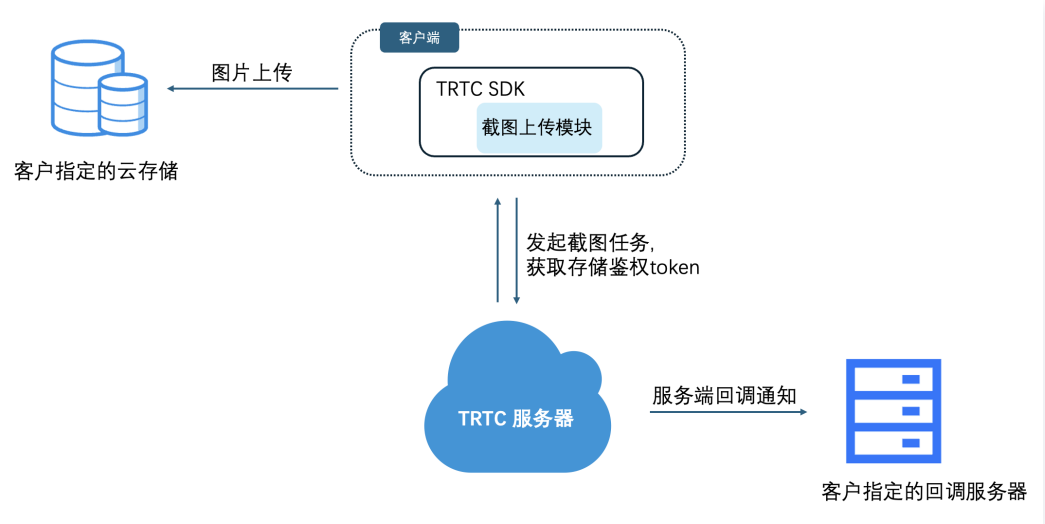
```
#include <boost/archive/iterators/binary_from_base64.hpp>
#include <boost/archive/iterators/transform_width.hpp>
#include <string>
#include <vector>
#include <algorithm>
#include <stdint.h>
#include "ITRTCCloud.h"

liteav::ITRTCCloud* trtcCloud;

// 获取在服务端生成的密钥和盐
bool getKeyAndSaltFromSever(std::string& secret, std::string& saltBase64);

// 声明一个工具函数，用于将 Base64 转换成 uint8_t
bool decodeBase64(const std::string& input, std::vector<uint8_t>& output)
{
    output.resize(32);
    typedef
boost::archive::iterators::transform_width<boost::archive::iterators::binary_from_base64<std::string:
:const_iterator>, 8, 6> Base64DecodeIterator;
    try {
        std::copy(Base64DecodeIterator(input.begin()), Base64DecodeIterator(input.end()),
output.begin());
    } catch (...) {
        return false;
    }
    return true;
}

int enablePayloadPrivateEncryption() {
    std::string key;
    std::string saltBase64;
    std::vector<uint8_t> salt;
    if(!getKeyAndSaltFromSever(key, saltBase64))
        return -1;
    if(trtcCloud && decodeBase64(saltBase64, salt)) {
        liteav::TRTCPayloadPrivateEncryptionConfig config;
        // 设置加密算法为 TRTCEncryptionAlgorithmAes128Gcm
        config.encryptionAlgorithm = TRTCEncryptionAlgorithm::TRTCEncryptionAlgorithmAes128Gcm;
        // 设置密钥
        config.encryptionKey = key.c_str();
        // 设置盐
        memcpy(config.encryptionSalt, salt.data(), sizeof(config.encryptionSalt));
        // 启用私有加密
        return trtcCloud->enablePayloadPrivateEncryption(true, config);
    }
    return -1;
}
```

视频截图上传

最近更新时间：2025-06-05 18:23:52

功能说明

实时音视频 TRTC 支持通过 SDK API 发起截图上传功能，用户可将截图用于第三方审核、封面图设置等场景，满足用户的使用需求。



前提条件

- 登录 TRTC 控制台，开通 TRTC 服务并 创建应用。
- 前往 控制台 > 功能配置 > 增值功能 开启视频截图上传功能，配置指定存储的对象存储 COS 桶。

说明：

- **视频截图上传**功能需通过购买包月套餐-**旗舰版、旗舰版 Plus** 或者领取包月套餐-**体验版**解锁。包月套餐相关说明请参见文档 包月套餐计费说明。
- **视频截图上传**功能会根据产生的截图用量产生费用，更多请见 费用详情。
- 使用**视频截图上传**功能，请提交工单联系我们获取最新版本 SDK（目前Android、iOS/Mac、Windows支持）.

功能说明

1. 参见 前提条件，开启功能开关并设置存储位置。
2. 通过 SDK 实验性接　callExperimentalAPI 来使　此功能，传参要求为 JSON 字符串，参数说明如下：

```
{
```

```
    "api": "enableAutoSnapshotAndUpload",
    "params" : {
    "enable": 1, //启动/停止自动截图，int，必填。字段取值：0 停止、1 启动，传值不能为空，支持传入 bool 值（true 和 false）。
    "intervalS": 1,// 截图间隔，int，可选。间隔单位为秒，缺省值 3 秒，最小间隔 1秒。字段传值不能为空，小于 1 则取最小值 1 秒。
    "streamType": 0, // 流类型，int，可选。字段取值：0 主路、2辅路，传值不能为空。缺省行为：启动截图时若不传入此字段则启动不生效，停止截图时若不传入此字段任务全部停止。
    "extraInfo": "customized messages" // 截图上传附加信息，string，可选。此信息会通过服务端回调通知给您的业务后台。
    }

}
```

注意：
- 截图上传任务在 enterRoom 成功后才会启动。
- 建议在 startLocalPreview 成功后调用此方法，避免截图上传任务失败。

接收服务端事件回调

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“转推时间组-CDN 推流正在进行”事件的包体示例。

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，截图事件（EVENT_GROUP_SCREEN_ SHOT）值为 6
EventType	Number	回调通知的事件类型，视频截图（EVENT_TYPE_VIDEO_SCREENSHOT）值为 601
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

事件信息说明：

字段名	类型	含义
eventId	String	当次回调的事件 ID
callbackData	String	截图上传附加信息，通过客户端的 extraInfo 上报
pictureURL	String	截图的 URL
code	Number	任务执　状态码，默认为 0 表示任务执　成功
msg	String	任务执　描述信息
roomID	String/Number	房间号
streamType	String	截图的流类型，主路（BigStream）或辅路（SubStream）
userID	String	截图　户名
timestamp	Number	截图 UTC 时间戳，精确到毫秒

回调请求示例：

```
{
   "EventGroupId": 6,
   "EventType": 601,
    "CallbackTs": 1698410059705,
   "EventInfo": {
       "eventID": "ap-guangzhou-1400000000-1698410059243691647-60022-jpg.jpg",
       "callbackData": "test",
       "pictureURL": "https://sotest-1200000000.cos.ap-
guangzhou.myqcloud.com/1400000000/ap-guangzhou-1400000000-1698410059243691647-
60022-jpg.jpg",
       "code": 0,
       "msg": "",
       "roomID": "464884",
       "streamType": "BigStream",
       "userID": "dd",
       "timestamp": 1698410059693
    }
}
```

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　式计算出签名，相同则说明是腾讯云的实时　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 用的加密秘钥。
```

```
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，需要完整保留\n\t转义字符**，示例如下：

```
body="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t103,\n\t\"CallbackTs\":\t1615554923704,\n\t\"Even
tInfo\":\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}"
```

签名校验示例（Java）

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//# key：控制台配置的密钥key
//# body：腾讯云回调返回的body体
//# sign：腾讯云回调返回的签名值sign
//# 返回值：
//# Status OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//# Info：成功/失败信息



public class checkSign {
    public static String getResultSign(String key, String body) throws

Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(),
"HmacSHA256");
        hmacSha256.init(secret_key);
        return
Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" +
"\t\"EventType\":\t204,\n" + "\t\"CallbackTs\":\t1664209748188,\n" +
"\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" + "\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" + "\t}\n" +
"}";

        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);


        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败''}");
        }
```

```
    }
}
```

说明：
更多签名示例说明可参见 签名校验示例 。

回调不通的常见原因

如果遇到回调不通的情况，建议先依照如下清单排查设置的回调服务是否存在问题。

回调不通的现象	可能存在的原因
回调 URL 访问超时	1. 法完成 DNS 解析，请确认该域名是否在公 效。（例如，回调 HOST 为 `http://notexist.com`，该域名不存在， 法完成 DNS 解析） 2. 法访问到回调 URL 中配置的 IP，请确认该 IP 是否公 可达。（例如，回调 HOST 为 `http://10.0.0.1`，该域名为内 IP， 法访问到该 IP） 3. 回调服务防 墙策略限制，请检查防 墙配置。（例如，App 回调服务器拒绝了所有到达 80 端 的请求）
回调服务拒绝访问	可以访问到 HOST，但链接建 失败，请确认 WebServer 已经正确启动。（例如：回调服务器的 WebServer 并未启动，或者端 配置错误。）
回调服务 HTTPS 证书配置错误	回调 式为 HTTPS（或 HTTPS 双向认证），能够访问到回调服务器，但判定 WebServer 配置的证书 法。请确认 HTTPS 证书配置正确。
回调服务 HTTPS 双向认证配置错误	回调 式为 HTTPS 双向认证，校验回调服务器的证书合法，但回调服务器校验实时 视频 TRTC 的证书失败。
回调服务 HTTP 返回码 200	回调请求成功，但应答报 中的 HTTP 返回码 200。
回调应答包体解析失败	回调请求包体 JSON 格式。

高级权限控制

最近更新时间：2022-10-10 11:06:27

内容介绍

如果您希望给某些房间中加入进房限制或者上麦限制，也就是仅允许指定的用户去进房或者上麦，而您又担心在客户端判断权限很容易遭遇破解攻击，那么可以考虑**开启高级权限控制**。

在如下场景下，您并不需要开启高级权限控制的：

- 情况1：本身希望越多的人观看越好，对进入房间的权限控制无要求。
- 情况2：对攻击者破解客户端的防范需求不迫切。

在如下场景下，建议您开启高级权限控制以获得更佳的安全性：

- 情况1：对安全性要求较高的视频通话或者语音通话场景。
- 情况2：对不同房间设置不同进入权限的场景。
- 情况3：对观众上麦有权限控制的场景。

支持的平台

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

高级权限控制的原理

开启高级权限控制后，TRTC 的后台服务系统就不会仅校验 UserSig 这一个“进房票据”，还会校验一个叫做 **PrivateMapKey** 的“权限票据”，权限票据中包含了一个加密后的 roomid 和一个加密后的“权限位列表”。

由于 PrivateMapKey 中包含 roomid，所以当用户只提供了 UserSig 没有提供 PrivateMapKey 时，并不能进入指定的房间。

PrivateMapKey 中的“权限位列表”使用了一个 byte 中的 8 个比特位，分别代表了持有该票据的用户，在该票据指定的房间中所拥有的八种具体的功能权限：

位数	二进制表示	十进制数字	权限含义
第 1 位	0000 0001	1	创建房间的权限
第 2 位	0000 0010	2	进入房间的权限
第 3 位	0000 0100	4	发送语音的权限
第 4 位	0000 1000	8	接收语音的权限
第 5 位	0001 0000	16	发送视频的权限
第 6 位	0010 0000	32	接收视频的权限
第 7 位	0100 0000	64	发送辅路（也就是屏幕分享）视频的权限
第 8 位	1000 0000	128	接收辅路（也就是屏幕分享）视频的权限

开启高级权限控制

步骤1：在 TRTC 控制台中开启高级权限控制

1. 在腾讯云实时音视频控制台中单击左侧的 **应用管理**。
2. 在右侧的应用列表中选择想要开启高级权限控制的一款应用，并单击 **功能配置**。

3. 在“功能配置”页卡中打开 **启用高级权限控制** 按钮，单击 **确定**，即可开启高级权限控制。

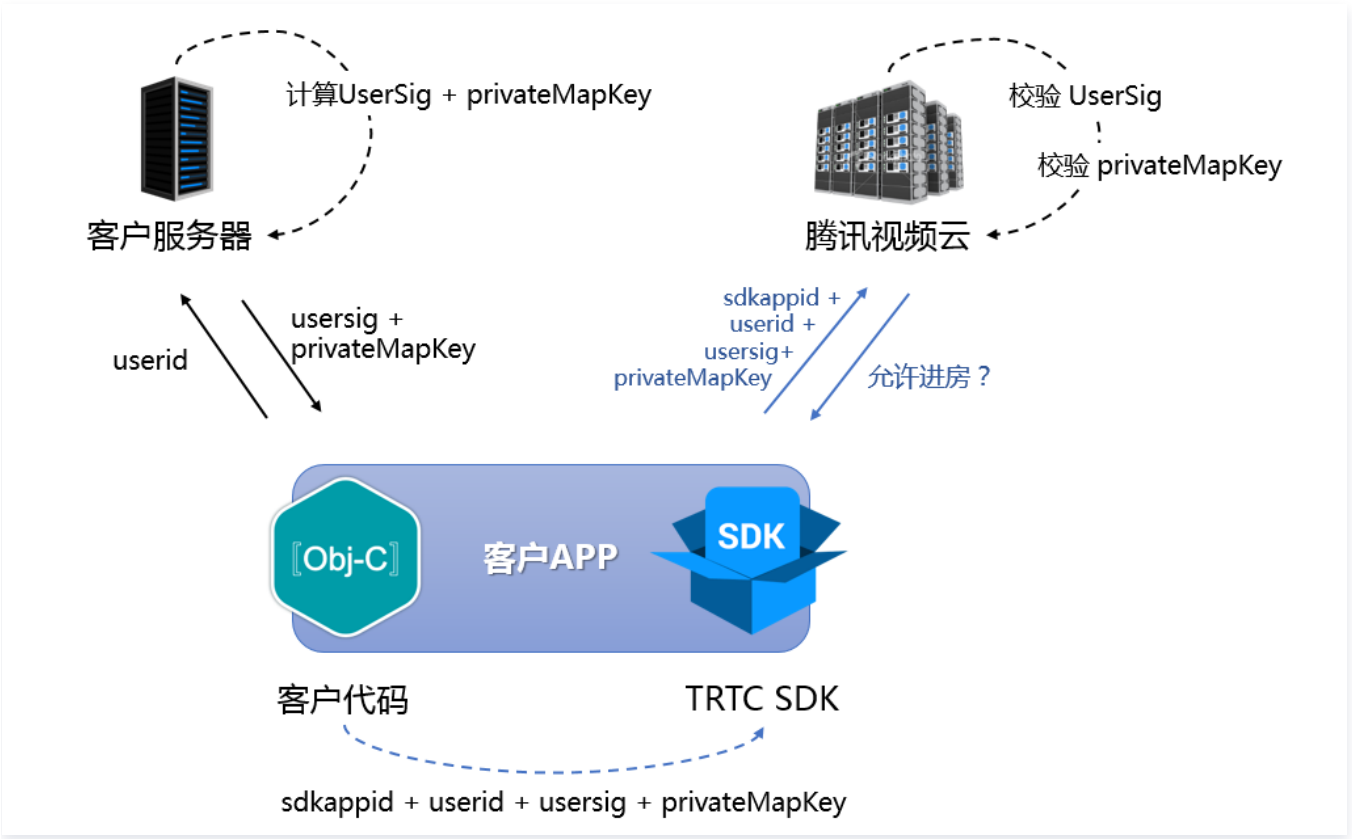
> **注意**
> 当某一个 SDKAppid 开启高级权限控制后，使用该 SDKAppid 的所有用户都需要在 TRTCParams 中传入 `privateMapKey` 参数才能成功进房（如 步骤 2 所述），如果您线上有使用此 SDKAppid 的用户，请不要轻易开启此功能。

步骤2：在您的服务端计算 PrivateMapKey

由于 PrivateMapKey 的价值就是为了防止客户端被逆向破解，从而出现“非会员也能进高等级房间”的破解版本，所以它只适合在您的服务器计算再返回给您的 App，绝不能在您的 App 端直接计算。
我们提供了 Java、GO、PHP、Node.js、Python、C# 和 C++ 版本的 PrivateMapKey 计算代码，您可以直接下载并集成到您的服务端。

语言版本	关键函数	下载链接
Java	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
GO	`GenPrivateMapKey` 和 `GenPrivateMapKeyWithStringRoomID`	Github
PHP	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
Node.js	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
Python	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
C#	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
C++	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	GitHub

步骤3：由您的服务端将 PrivateMapKey 下发给您的 App



如上图所示，当您的服务器计算好 PrivateMapKey 之后，就可以下发给您的 App，您的 App 可以通过两种方案将 PrivateMapKey 传递给 SDK：

方案一：在 enterRoom 时传递给 SDK

如果想要控制用户进入房间的权限，您可以在调用 `TRTCCloud` 的 `enterRoom` 接口时，通过设置 TRTCParams 中的 **privateMapKey** 参数即可实现。
这种进房时校验 PrivateMapKey 的方案比较简单，非常适合于在用户进入房间前就能将用户权限确认清楚的场景。

方案二：通过实验性接口更新给 SDK

在直播场景中，往往都会有观众上麦变成主播的连麦场景。当观众变成主播时，TRTC 会再校验一次进房时在进房参数 `TRTCParams` 中携带的 PrivateMapKey，如果您将 PrivateMapKey 的有效期设置得比较短，例如“5分钟”，就会很容易触发校验失败进而导致用户被踢出房间。

当 SDK **版本在10.2之前**的客户，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);
```

当 SDK 版本在10.2及之后，可以直接通过切换角色接口 `switchRole(TRTCRoleType role, const char* privateMapKey)` 设置 privateMapKey，请勿使用上述实验性接口的方式。

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。**

测试硬件设备
Android&iOS&Windows&Mac

最近更新时间：2024-07-17 09:53:21

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：2024-09-02 17:35:11

> 本教程基于 5.0 TRTC Web SDK 实现，若您使用 4.x.x 版本 SDK，可参考 此教程 。

浏览器环境检测

在开始音视频通话之前，建议您先使用 `TRTC.isSupported()` 接口检测 SDK 是否支持当前网页。如果 SDK 不支持当前浏览器，建议用户使用最新版的 Chrome 浏览器、Edge 浏览器、Safari 浏览器、Firefox 浏览器。

```
TRTC.isSupported().then(checkResult => {
  // 不支持使用 SDK，引导用户使用最新版的 Chrome 浏览器。
  if (!checkResult.result) {}
  // 不支持发布视频
  if (!checkResult.result.isH264EncodeSupported) {}
  // 不支持拉视频
  if (!checkResult.result.isH264DecodeSupported) {}
})
```

⚠ 当用户使用 SDK 支持的浏览器，且收到 `TRTC.isSupported` 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，请稍等再试或引导使用其他浏览器。

使用设备检测插件

检测用户的媒体设备（摄像头、麦克风、扬声器）以及网络，建议在用户进房之前使用，支持移动端适配。

> ⚠ **注意：**
>
> - TRTC Web SDK 版本 ⩾ 5.8.0。
> - 此设备检测插件的 z-index 设定为9999 ，开发者需确保自身层级 z-index 低于9999 。

实现流程

```
import { DeviceDetector } from 'trtc-sdk-v5/plugins/device-detector';
const trtc = TRTC.create({ plugins: [DeviceDetector] });

// 1. Test Media Device Only
const result = await trtc.startPlugin('DeviceDetector');

// 2. Test Media Device & Network Quality
const options = {
    networkDetect: { sdkAppId, userId, userSig }
}
const resultWithNetwork = await trtc.startPlugin('DeviceDetector', options);
```

API 说明

trtc.startPlugin('DeviceDetector', options)

第一个参数为 'DeviceDetector' 字符串，用于开启设备检测插件。

options.networkDetect(optional)

Name	Type	Attributes	Description
sdkAppId	`number`	必填	从控制台获得，您的 sdkAppId
userId	`string`	必填	上行用户 ID，自行决定，与 downlinkUserId 不一致，与用于生成 userSig 的 userId一致
userSig	`string`	必填	userSig 签名，由 sdkAppId，userId，sdkSecretKey 作为参数生成，生产环境由向服务器请求生成，参见 UserSig
downlinkUserId	`string`	选填	下行用户 ID, 自行决定，与 userId 不一致，与用于生成 userSig 的 userId 一致，填写 downlinkUserId 和 downlinkUserSig 将进行下行网络测试
downlinkUserSig	`string`	选填	userSig 签名，由 sdkAppId，downlinkUserId，sdkSecretKey 作为参数生成，生产环境由向服务器请求生成，参见 UserSig
roomId	`number`	选填	选填，默认为8080，数字类型的房间号，取值要求为 [1, 4294967294] 的整数

startPlugin 返回结果

> **说明：**
> 若用户在检测界面点击跳过检测，返回 `undefiend` 。

Name	Type	Description
camera	`object`	{ isSuccess: boolean, device: MediaDeviceInfo }
microphone	`object`	{ isSuccess: boolean, device: MediaDeviceInfo }
speaker	`object`	{ isSuccess: boolean, device: MediaDeviceInfo }
network	`object`	网络检测结果 (startPlugin传入networkDetect才检测返回) { isSuccess: boolean, result: { quality: number, rtt: number}} quality为网络质量，默认为上行网络质量，传入下行userId和userSig时为上下行网络质量中较差值 **网络质量** 0：网络状况未知，表示当前 TRTC 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

插件 Demo

插件接入方式和效果体验，可前往 插件 Demo 页面。

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&Flutter

最近更新时间：2024-08-15 16:19:31

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。

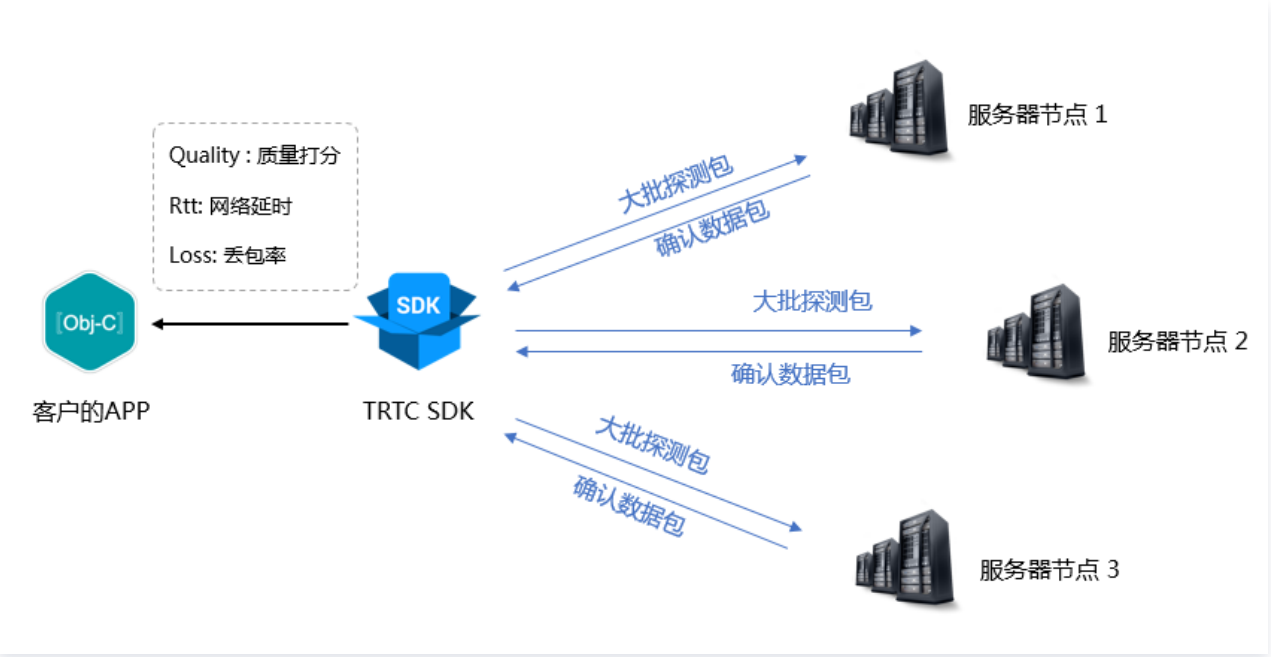
注意事项

- 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;
    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);
}

// 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;
    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;
    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

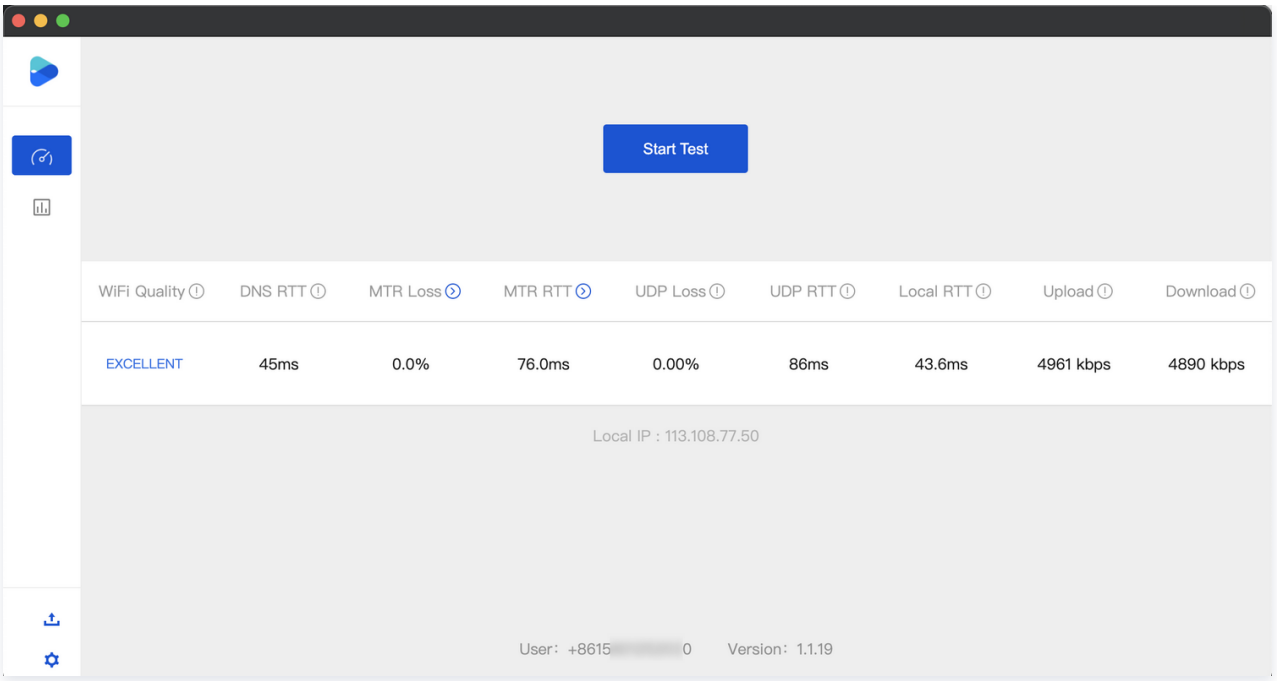
测试指标

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

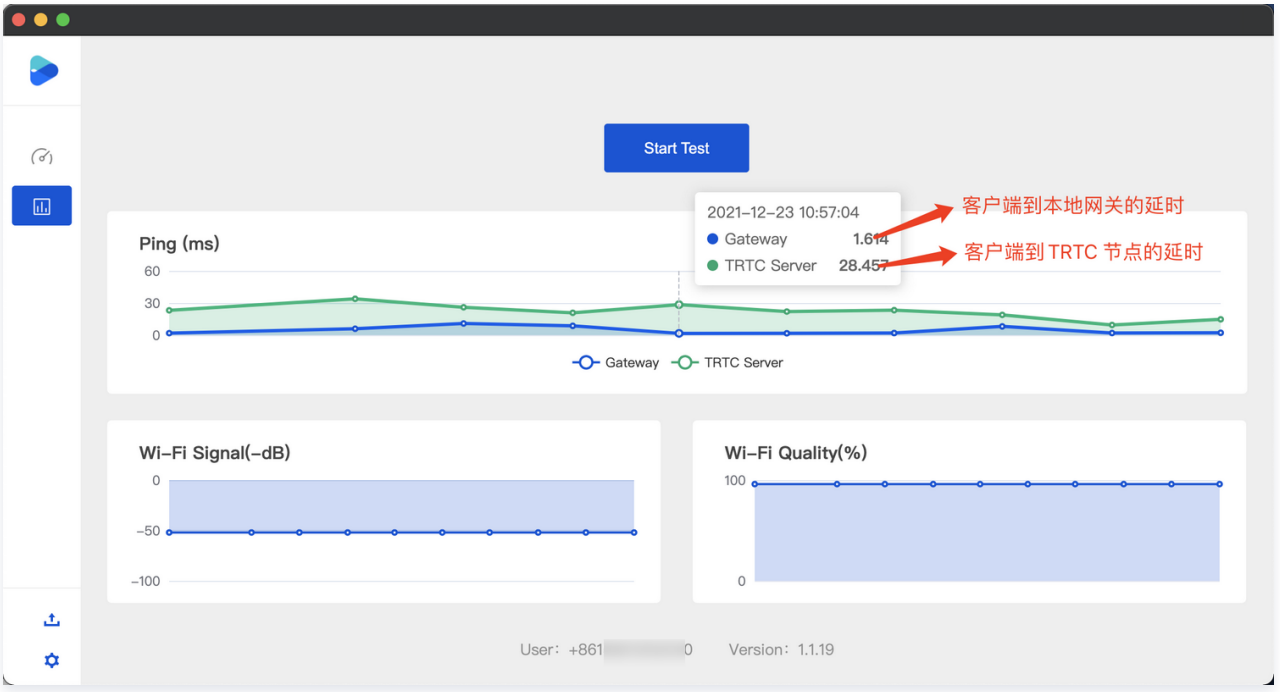
工具截图

- 登录：

- 快速测试：


Start Test
WiFi Quality
DNS RTT
MTR Loss
MTR RTT
UDP Loss
UDP RTT
Local RTT
Upload
Download
EXCELLENT
45ms
0.0%
76.0ms
0.00%
86ms
43.6ms
4961 kbps
4890 kbps
Local IP : 113.108.77.50
User: +8615 0
Version: 1.1.19

- **持续测试：**


Start Test
2021-12-23 10:57:04
Gateway 1.614
TRTC Server 28.457
客户端到本地网关的延时
客户端到 TRTC 节点的延时
Ping (ms)
60
30
0
Gateway
TRTC Server
Wi-Fi Signal(-dB)
0
-50
-100
Wi-Fi Quality(%)
100
0
User: +861 0
Version: 1.1.19

Web

最近更新时间：2024-12-13 21:04:53

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。**

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。