实时音视频

高级功能

产品文档

腾讯云

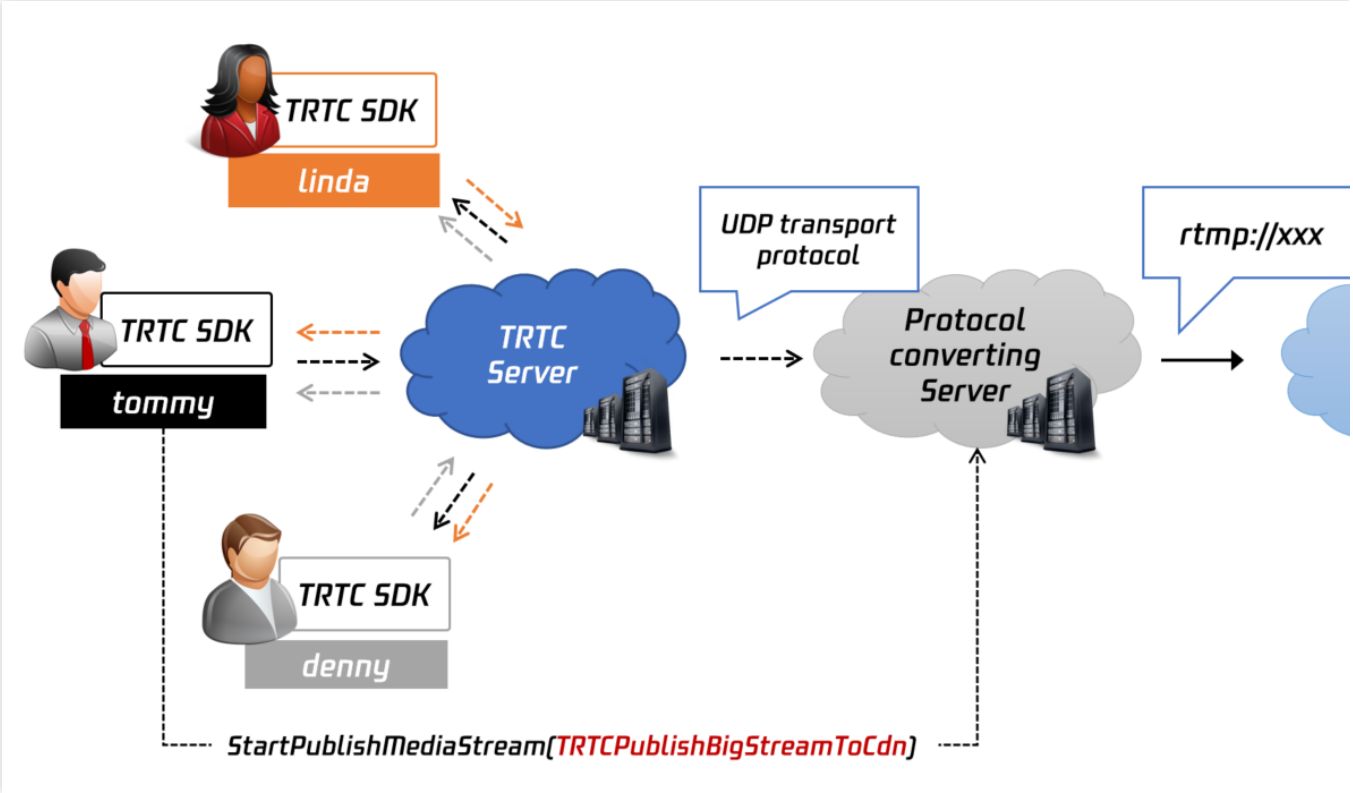
文档目录

高级功能
发布音视频流到直播 CDN

最近更新时间：2024-05-30 14:38:27

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。

发布当前用户的音视频流到直播CDN



前提条件

1.
前往
开通腾讯 云直播 服务（请注意确保账号已绑卡且未欠费，否则无法开通云直播服务）。云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。

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，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。

功能介绍

您可以使用 TRTCCloud 提供的接口 **startPublishMediaStream** 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，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。

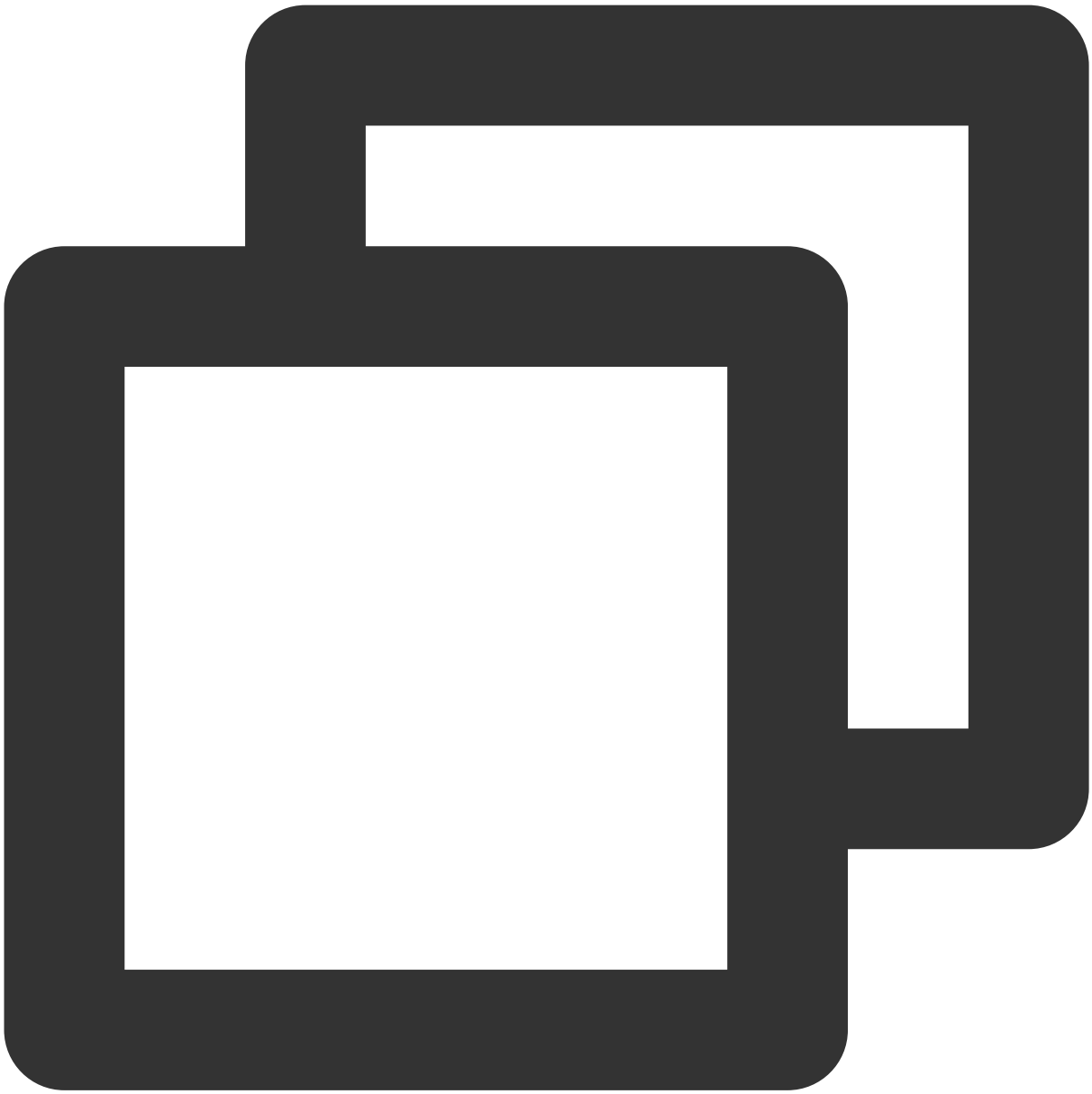
参考代码

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
ObjC

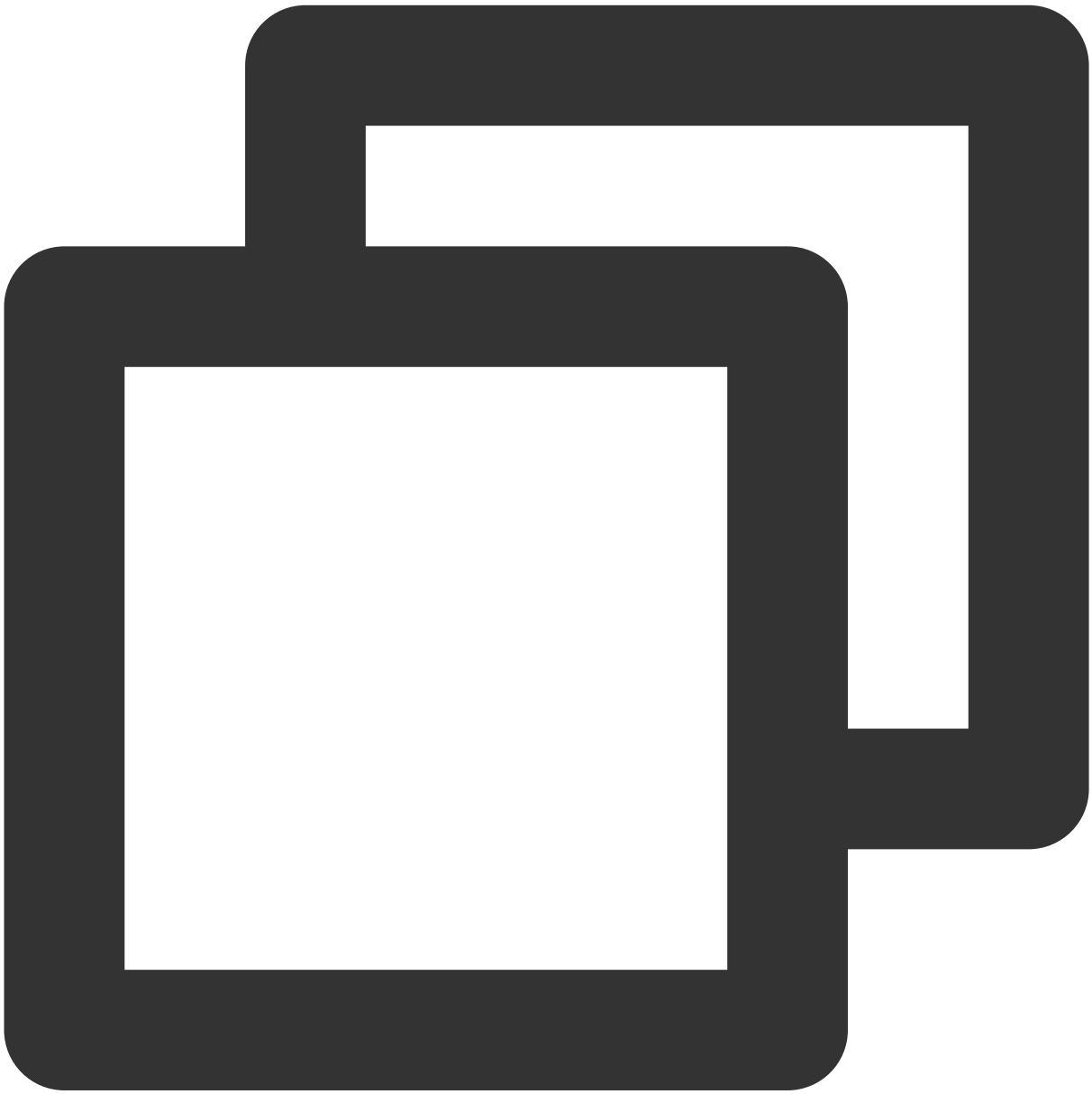
C++

Dart



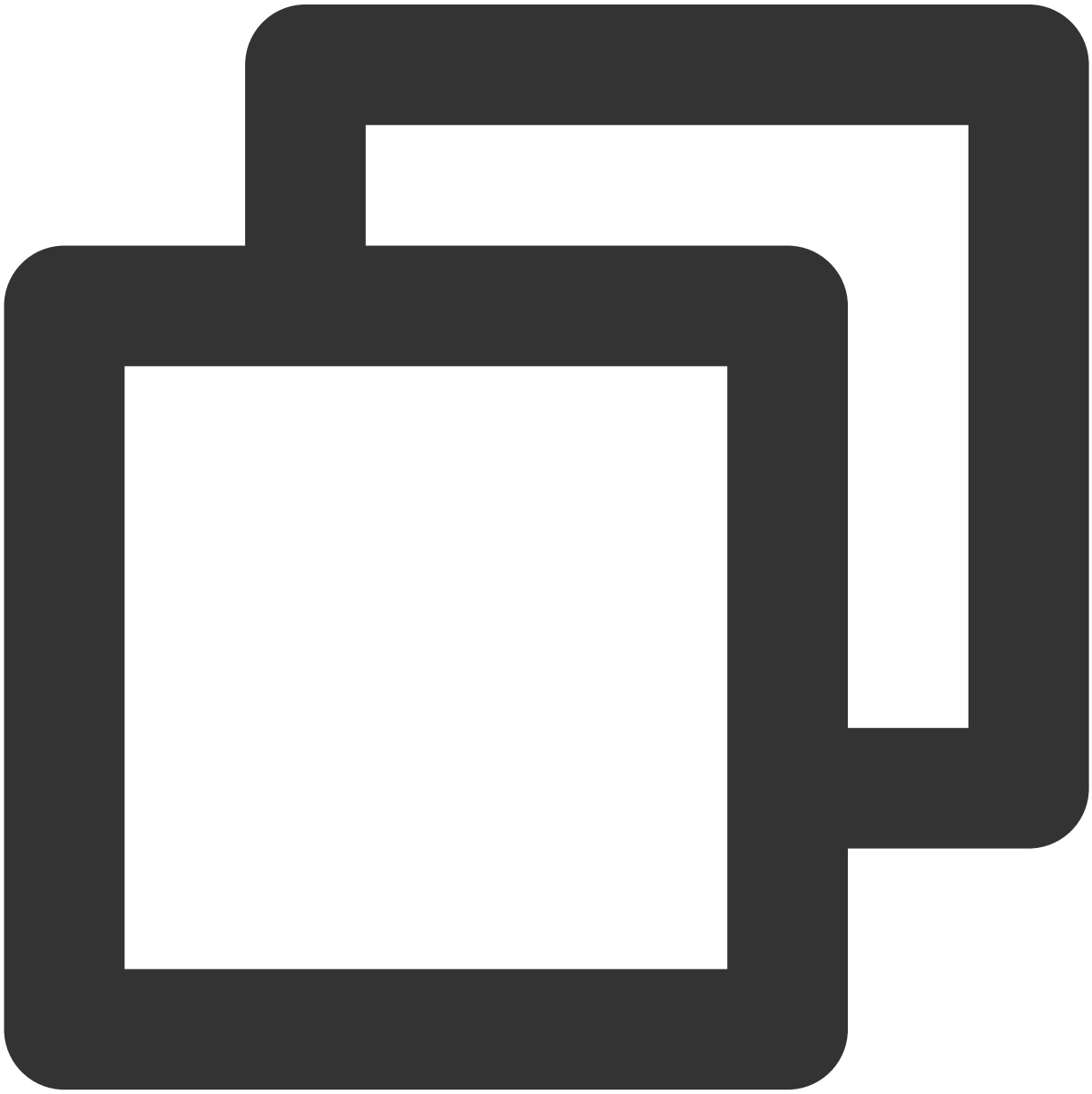
```
// 发布当前用户的音视频流到直播CDN
TRTCCloudDef.TRTCPublishTarget target = new TRTCCloudDef.TRTCPublishTarget();
target.mode = TRTC_PublishBigStream_ToCdn;

TRTCCloudDef.TRTCPublishCdnUrl cdnUrl= new TRTCCloudDef.TRTCPublishCdnUrl();
cdnUrl.rtmpUrl = "rtmp://tencent/live/bestnews";
cdnUrl.isInternalLine = true;
```

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

mTRTCCloud.startPublishMediaStream(target, null, null);
```

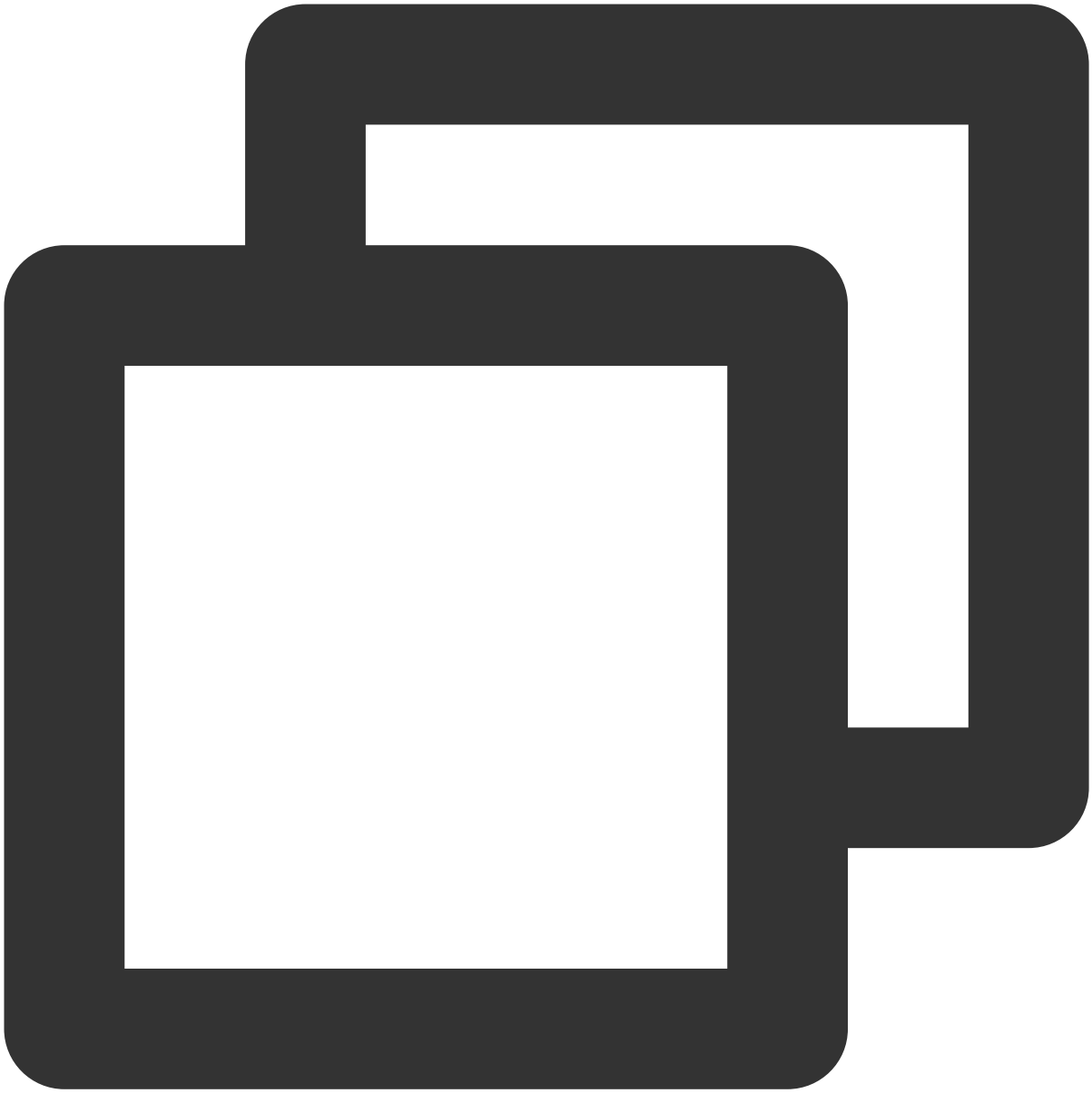


```
// 发布当前用户的音视频流到直播CDN
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishBigStreamToCdn;
```

```
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];
```

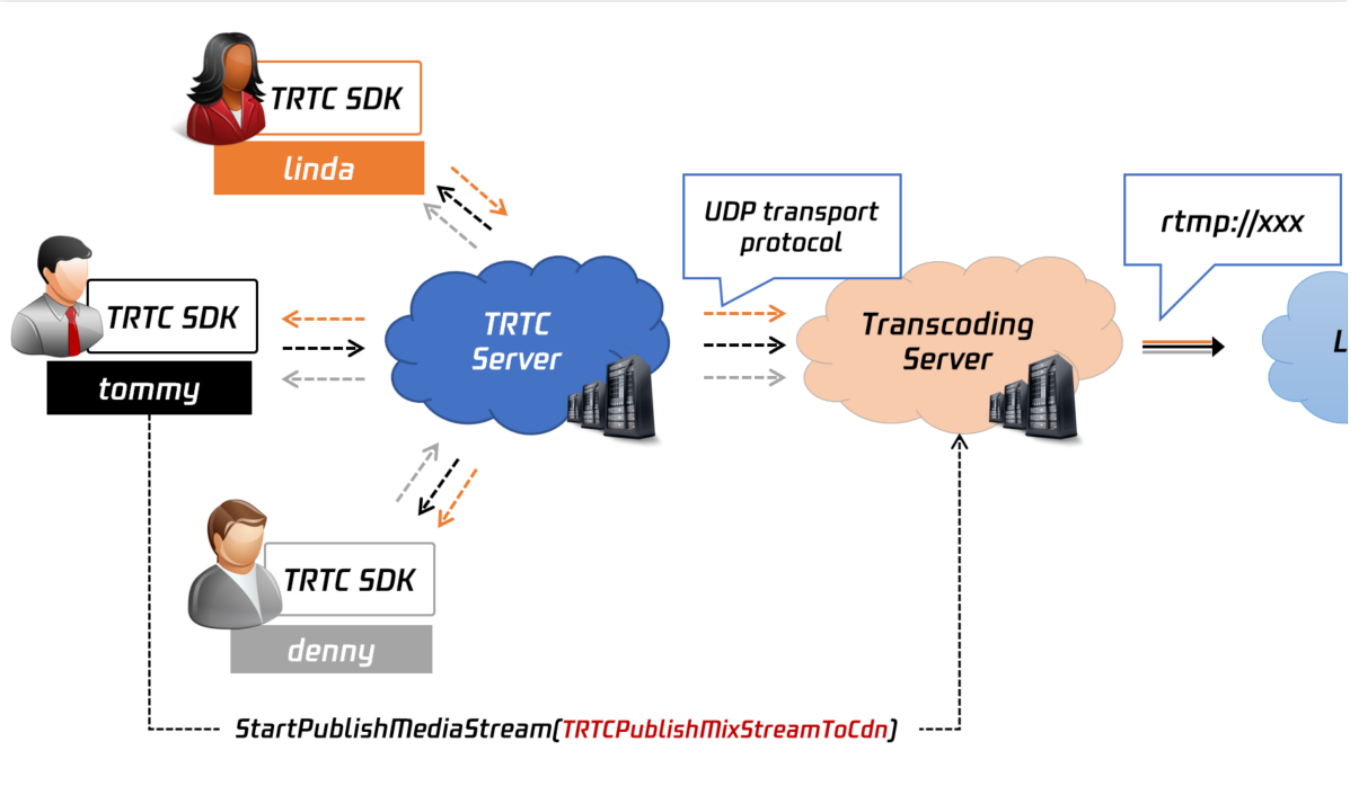


```
// 发布当前用户的音视频流到直播CDN
TRTCPublishTarget target;
target.mode = TRTCPublishMode::TRTCPublishBigStreamToCdn;

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;
```



```
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);
```

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。

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。

videoEncodedWidth	videoEncodedHeight	videoEncodedFPS	videoEncodedGOP	videoEncodedKbp
640	360	15	3	800kbps
960	540	15	3	1200kbps
1280	720	15	3	1500kbps
1920	1080	15	3	2500kbps

TRTCAudioQuality	audioEncodedSampleRate	audioEncodedChannelNum	audioEncodedKbps
TRTCAudioQualitySpeech	48000	1	50
TRTCAudioQualityDefault	48000	1	50
TRTCAudioQualityMusic	48000	2	60

4. 通过参数 `TRTCStreamMixingConfig` 设置音频混流参数和画面排版模式：

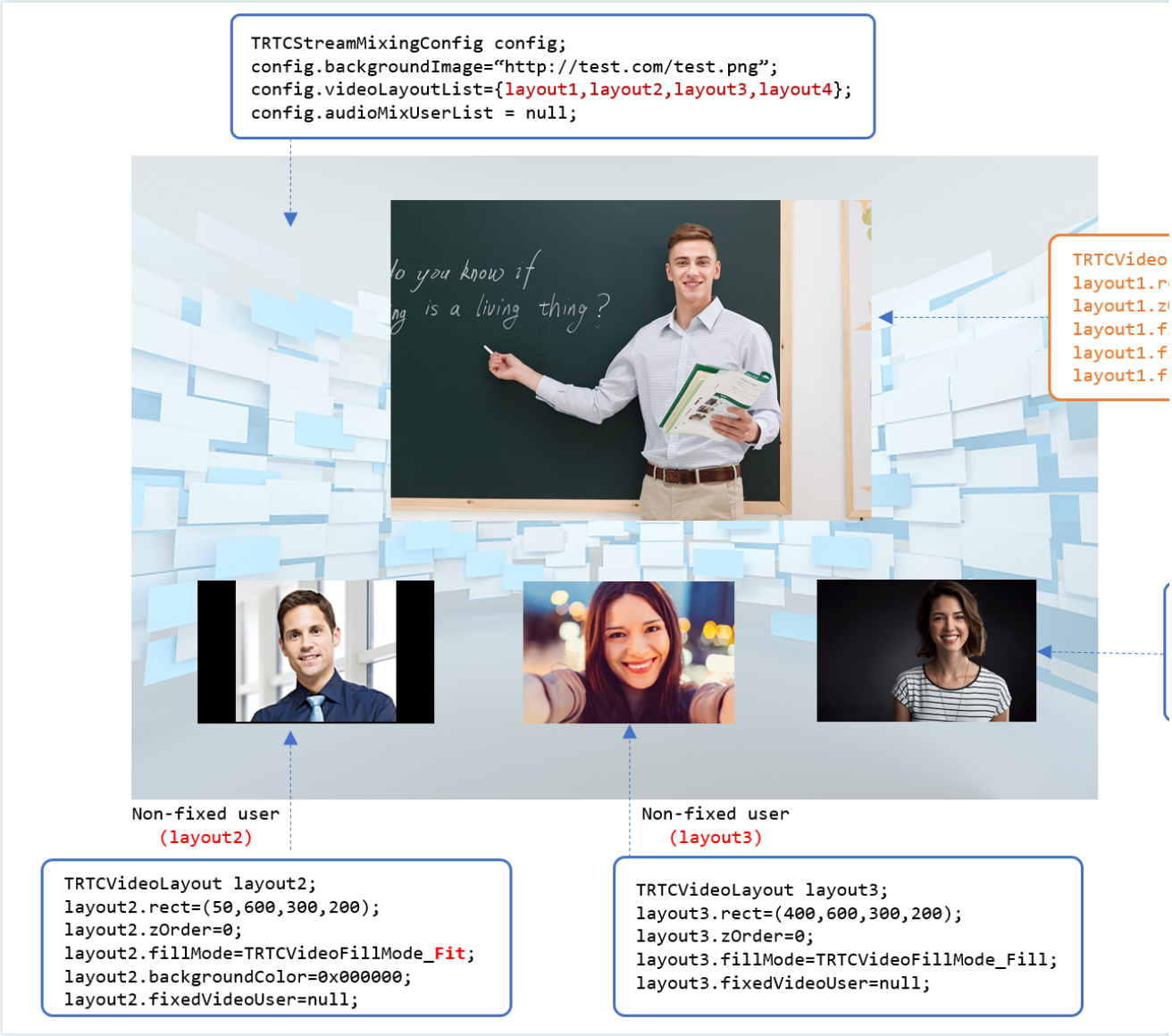
音频混流参数（audioMixUserList）：默认情况下填空值即可，代表会混合房间中的所有音频，如果您只希望混合方面中某几个用户的声音，才需要指定该参数。

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。

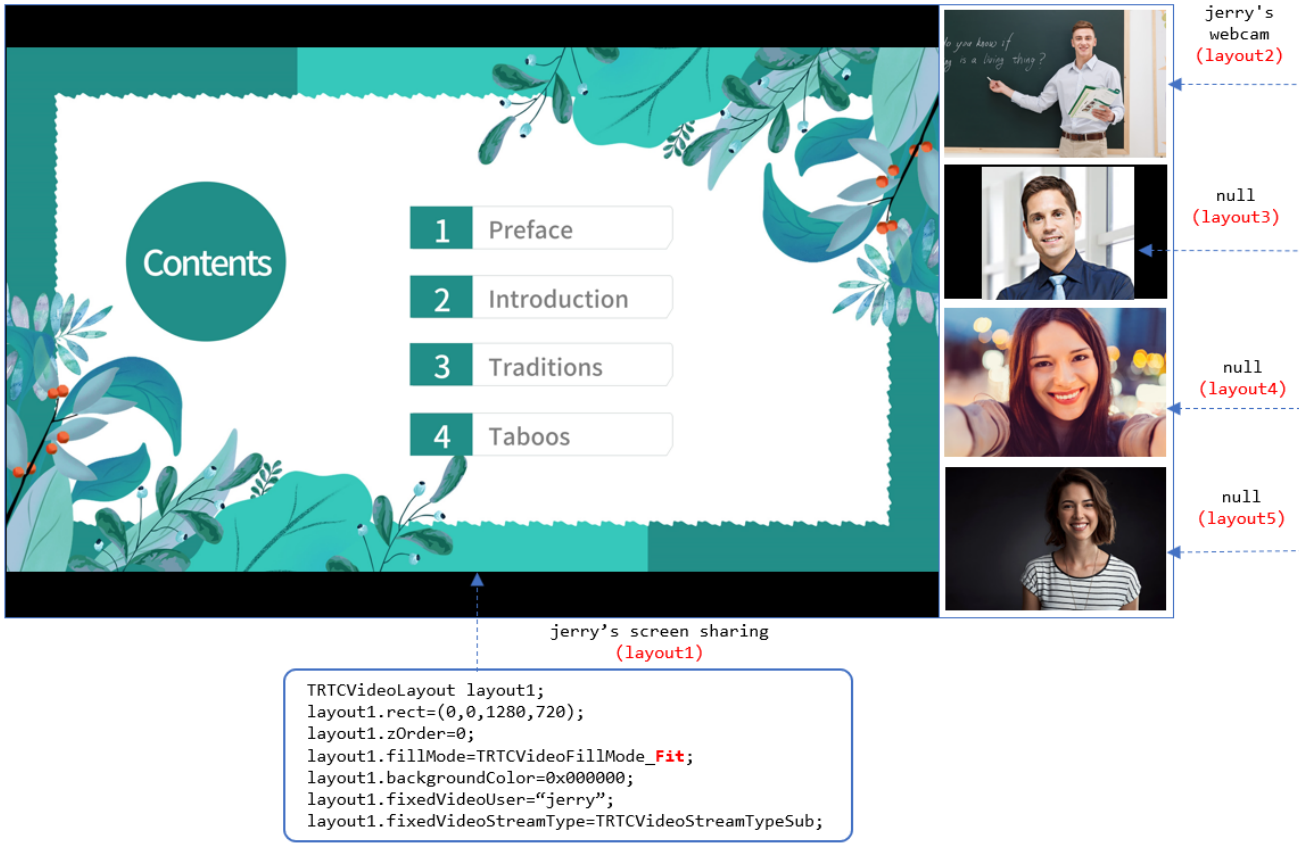


案例二：四位用户的摄像头画面和一路屏幕分享的画面被混合在了同一个画面中

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，比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

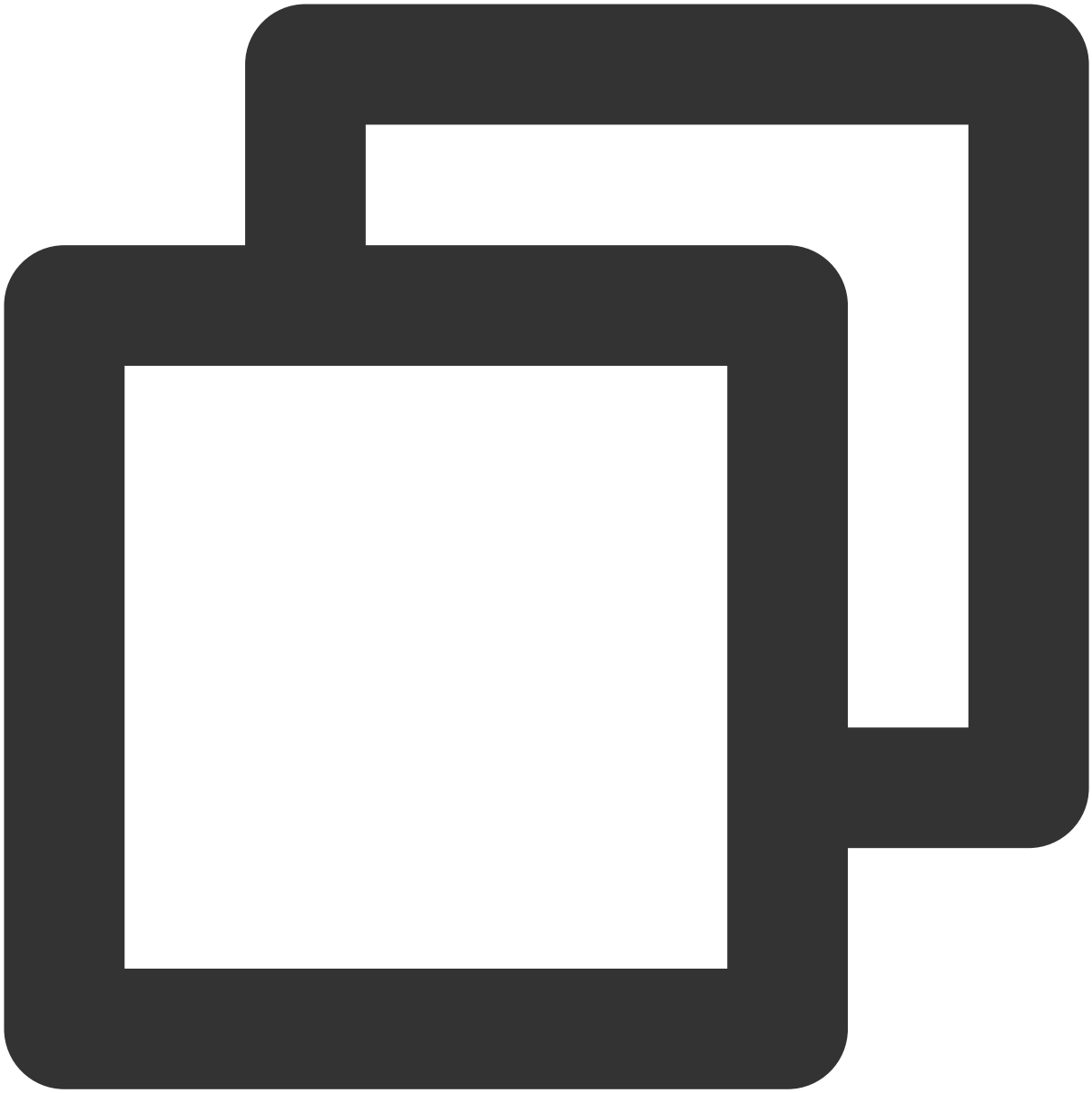
如下这段代码的功能是将房间中多个用户的音视频流混合并发布到直播 CDN 上：

Java

ObjC

C++

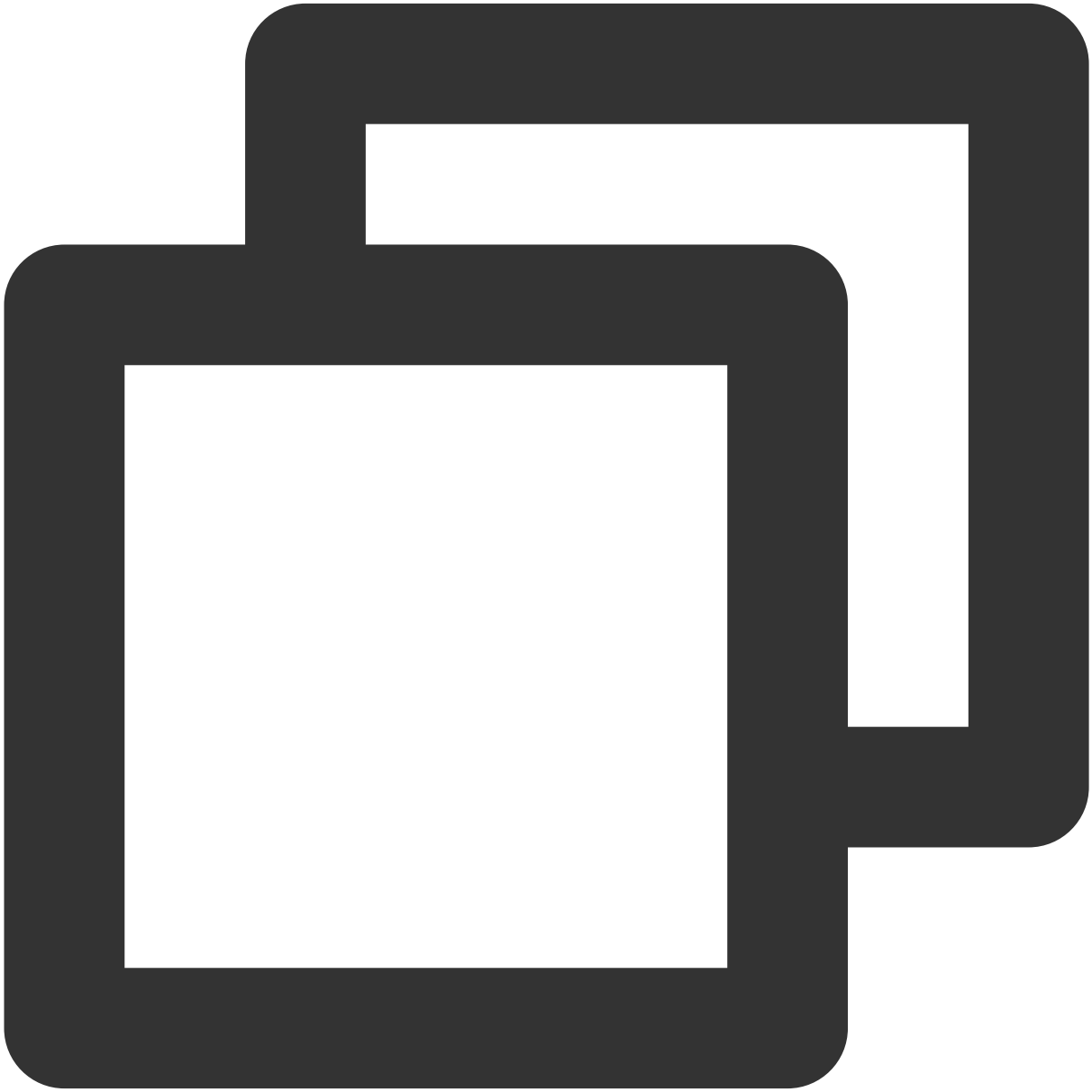
Dart



```
// 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);
```

```
// 设置混合后的音视频流的二次编码参数
TRTCCloudDef.TRTCStreamEncoderParam encoderParam
    = new TRTCCloudDef.TRTCStreamEncoderParam();
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
TRTCCloudDef.TRTCStreamMixingConfig mixingConfig =
      new TRTCCloudDef.TRTCStreamMixingConfig();
TRTCCloudDef.TRTCVideoLayout layout1 = new TRTCCloudDef.TRTCVideoLayout();
layout1.zOrder = 0;
layout1.x = 0;
layout1.y = 0;
layout1.width = 720;
layout1.height = 1280;
layout1.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB;
layout1.fixedVideoUser.intRoomId = 1234;
layout1.fixedVideoUser.userId = "mike";

TRTCCloudDef.TRTCVideoLayout layout2 = new TRTCCloudDef.TRTCVideoLayout();
layout2.zOrder = 0;
layout2.x = 1300;
layout2.y = 0;
layout2.width = 300;
layout2.height = 200;
layout2.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG;
layout2.fixedVideoUser.intRoomId = 1234;
layout2.fixedVideoUser.userId = "mike";

TRTCCloudDef.TRTCVideoLayout layout3 = new TRTCCloudDef.TRTCVideoLayout();
layout3.zOrder = 0;
layout3.x = 1300;
layout3.y = 220;
layout3.width = 300;
layout3.height = 200;
layout3.fixedVideoStreamType = TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_SUB;
layout3.fixedVideoUser = null;

mixingConfig.videoLayoutList.add(layout1);
```

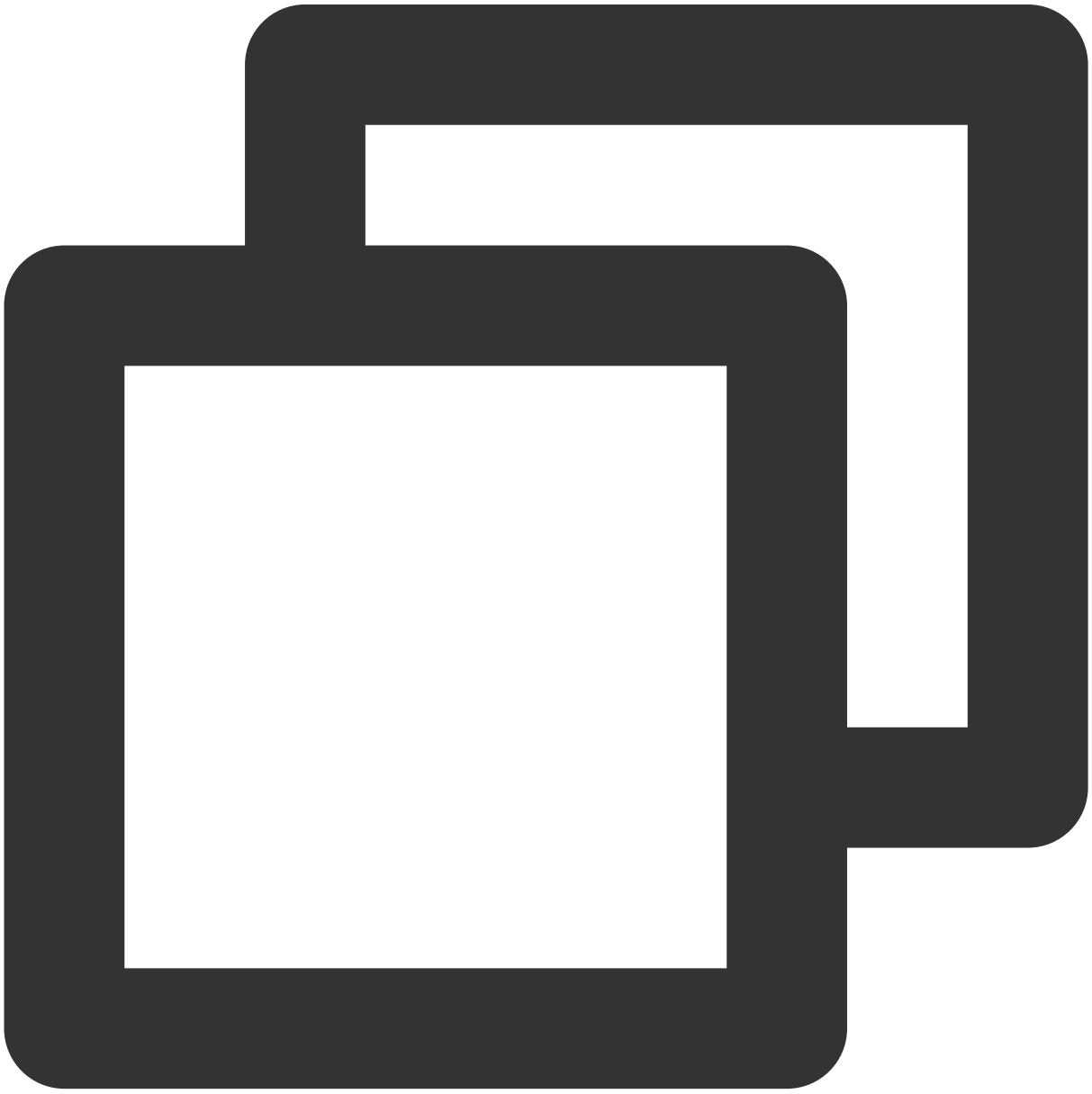
```
mixingConfig.videoLayoutList.add(layout2);
mixingConfig.videoLayoutList.add(layout3);
mixingConfig.audioMixUserList = null;

// 发起混流
mTRTCCloud.startPublishMediaStream(target, encoderParam, mixingConfig);
```



```
// 指定发布模式为 TRTCPublishMixStreamToCdn
```

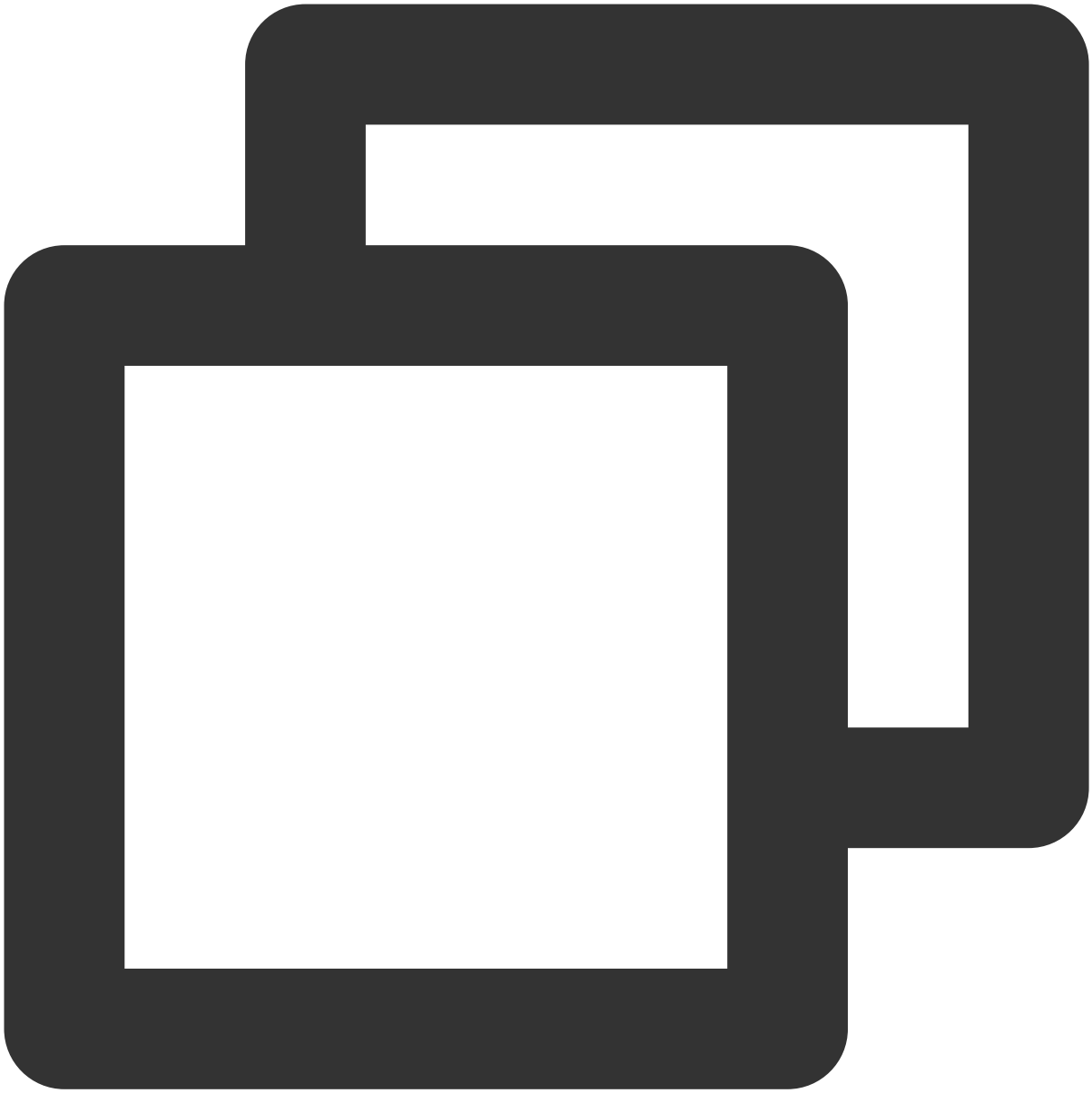
```
TRTCPublishTarget* target = [[TRTCPublishTarget alloc] init];
target.mode = TRTCPublishMixStreamToCdn;

// 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: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;
```



```
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: 19
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
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);
```

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？

答：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？

答：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？

答：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？

答：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？

答：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？

答：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。

相关费用

费用的计算

云端混流转码需要对输入 MCU 集群的音视频流进行解码后重新编码输出，将产生额外的服务成本，因此 TRTC 将向使用 MCU 集群进行云端混流转码的用户收取额外的增值费用。云端混流转码费用根据转码输入的分辨率大小和转码时长进行计费，转推至 CDN 则根据月峰值带宽进行计费。详情请参见 混流转推计费说明。

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。

高级权限控制

最近更新时间：2024-08-09 22:25:01

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	Web 端	Flutter
✓	✓	✓	✓	✓	✓	✓

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：

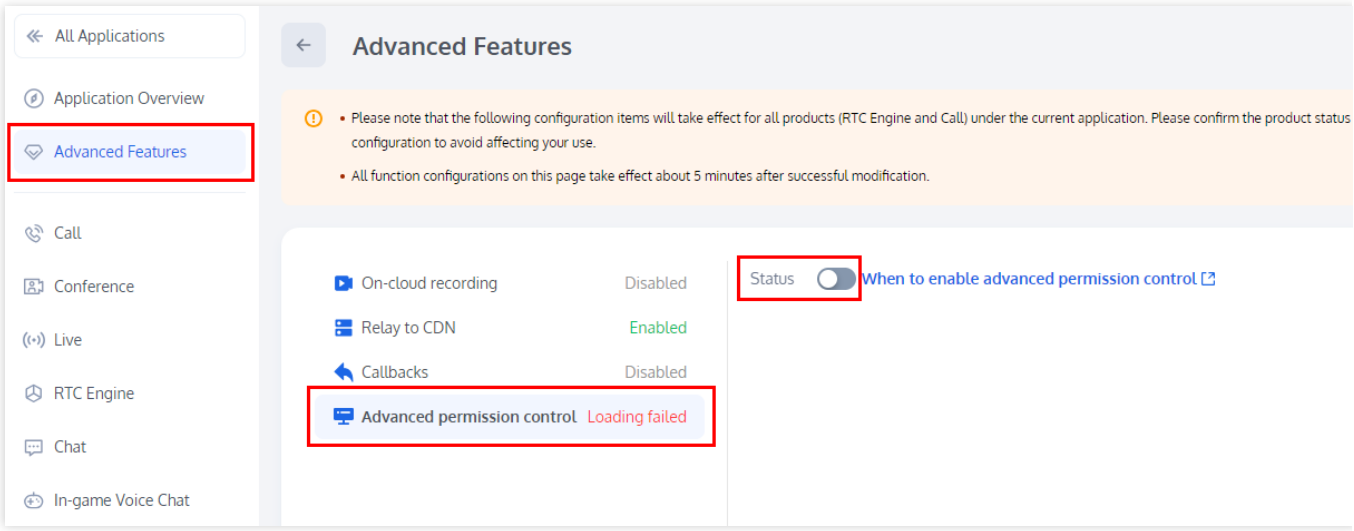
位数	二进制表示	十进制数字	权限含义
第 1 位	0000 0001	1	创建房间的权限
第 2 位	0000 0010	2	进入房间的权限
第 3 位	0000 0100	4	发送语音的权限

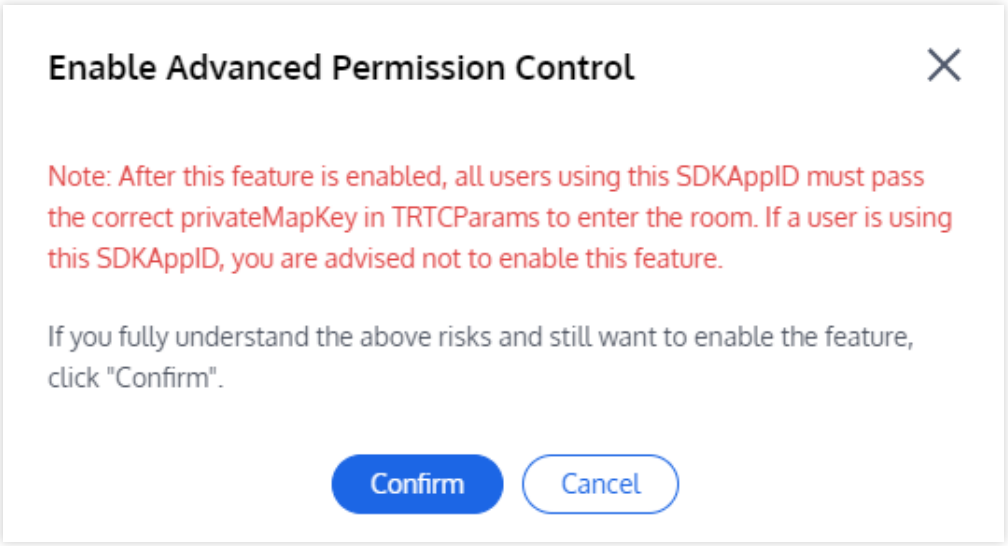
第 4 位	0000 1000	8	接收语音的权限
第 5 位	0001 0000	16	发送视频的权限
第 6 位	0010 0000	32	接收视频的权限
第 7 位	0100 0000	64	发送辅路（也就是屏幕分享）视频的权限
第 8 位	1000 0000	128	接收辅路（也就是屏幕分享）视频的权限

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. 前往 Tencent RTC 控制台 > 应用管理，单击需要修改功能配置的目标应用所在行的**管理**，选择左侧项目栏的**高级功能**。
2. 在**高级权限控制**中，单击右侧的**启用高级权限控制**右侧按钮，弹窗中单击 **Confirm** 完成启用。





注意：

当某一个 SDKAppid 开启高级权限控制后，使用该 SDKAppid 的所有用户都需要在 TRTCParams 中传入 `privateMapKey` 参数才能成功进房（如 步骤 2 所述），如果您线上有使用此 SDKAppid 的用户，请不要轻易开启此功能。

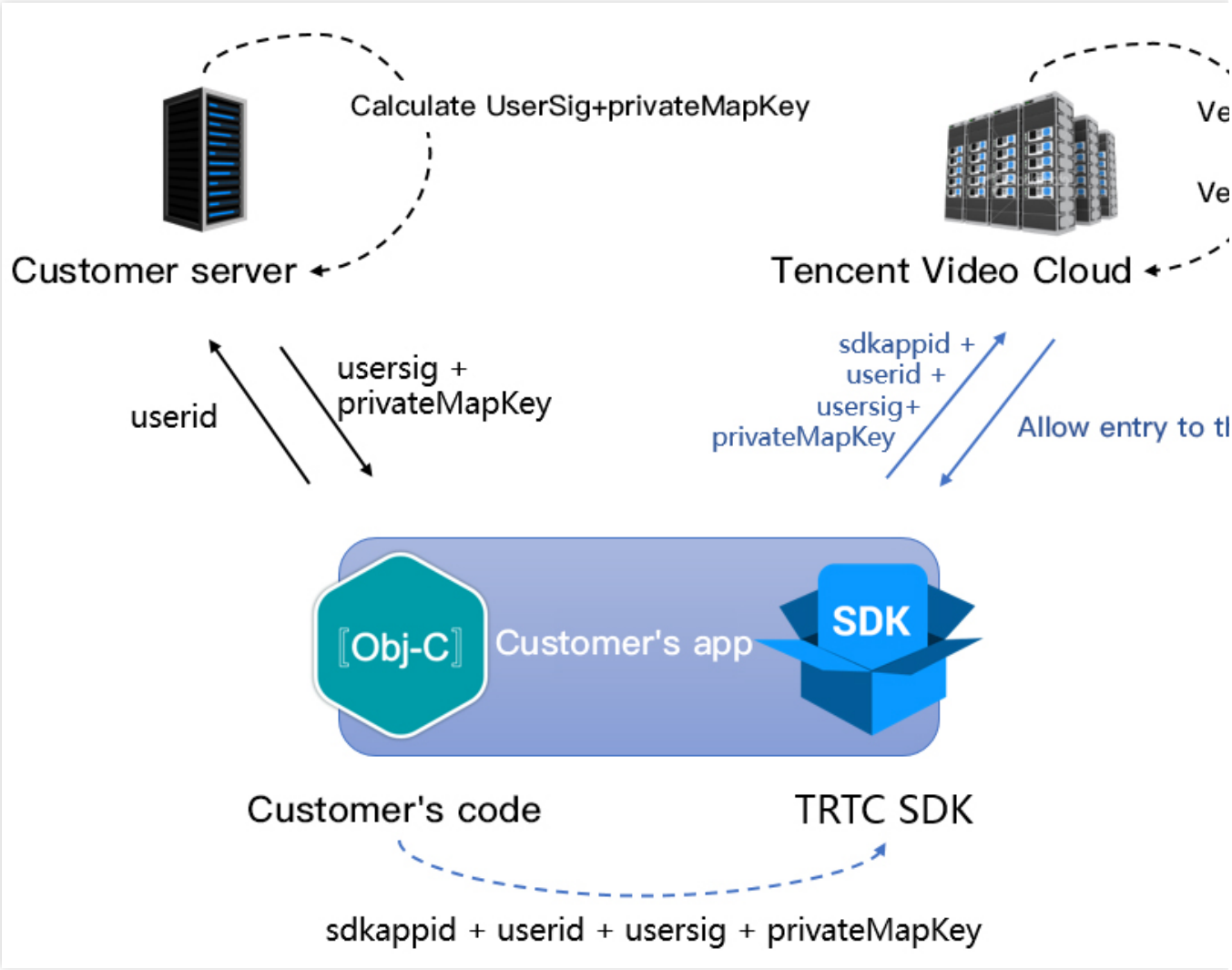
步骤2：在您的服务端计算 PrivateMapKey

由于 PrivateMapKey 的价值就是为了防止客户端被逆向破解，从而出现“非会员也能进高等级房间”的破解版本，所以它只适合在您的服务器计算再返回给您的 App，绝不能在您的 App 端直接计算。

我们提供了 Java、GO、PHP、Node.js、Python、C# 和 C++ 版本的 PrivateMapKey 计算代码，您可以直接下载并集成到您的服务端。

语言版本	关键函数	下载链接
Java	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
GO	`GenPrivateMapKey` 和 `GenPrivateMapKeyWithStringRoomID`	Github
PHP	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
Node.js	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
Python	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
C#	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	Github
C++	`genPrivateMapKey` 和 `genPrivateMapKeyWithStringRoomID`	GitHub

步骤3：由您的服务端将 PrivateMapKey 下发给您的 App



如上图所示，当您的服务器计算好 PrivateMapKey 之后，就可以下发给您的 App，您的 App 可以通过两种方案将 PrivateMapKey 传递给 SDK：

方案一：在 enterRoom 时传递给 SDK

如果想要控制用户进入房间的权限，您可以在调用 `TRTCCloud` 的 `enterRoom` 接口时，通过设置 TRTCParams 中的 **privateMapKey** 参数即可实现。
这种进房时校验 PrivateMapKey 的方案比较简单，非常适合于在用户进入房间前就能将用户权限确认清楚的场景。

方案二：通过实验性接口更新给 SDK

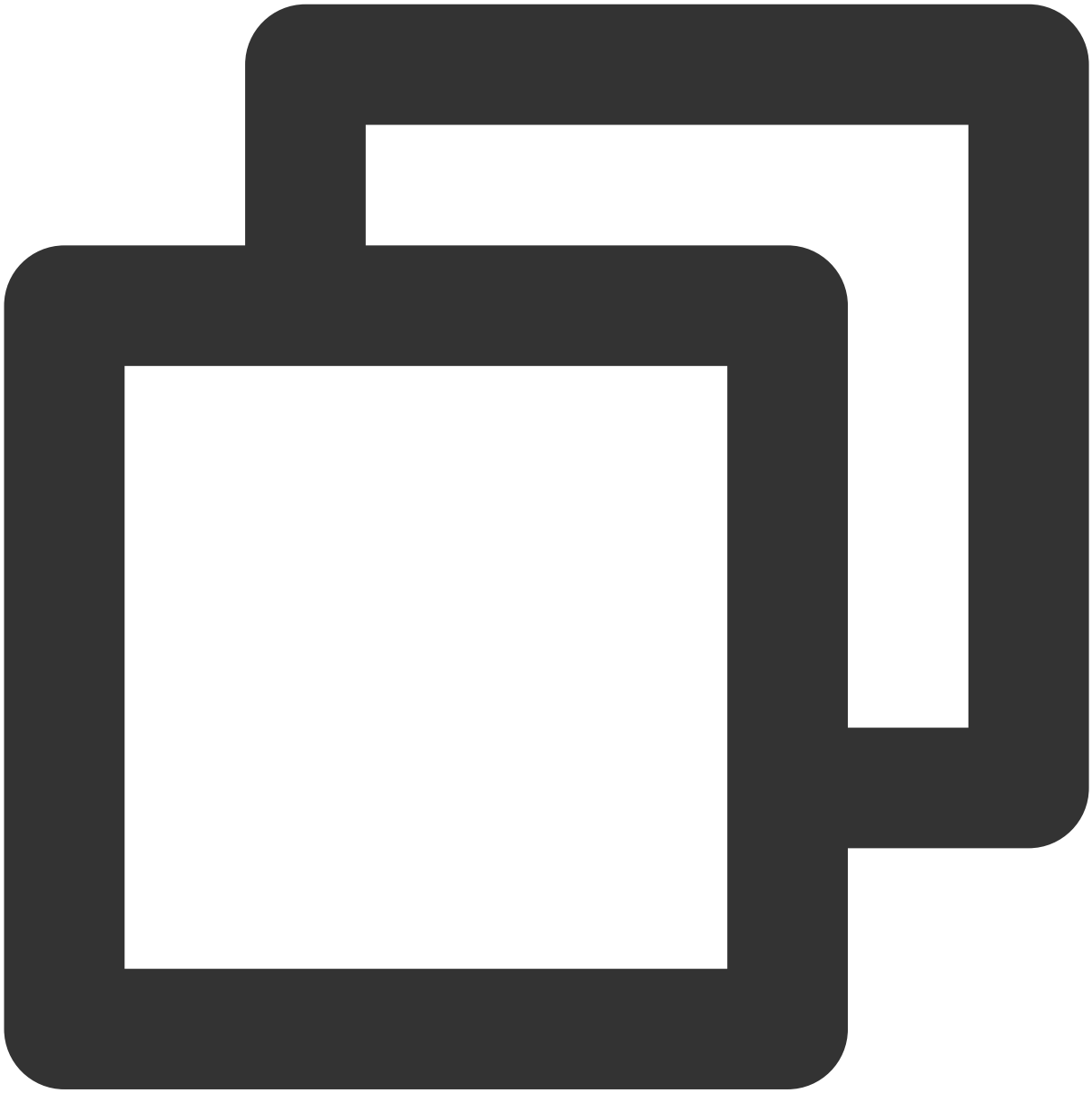
在直播场景中，往往都会有观众上麦变成主播的连麦场景。当观众变成主播时，TRTC 会再校验一次进房时在进房参数 `TRTCParams` 中携带的 PrivateMapKey，如果您将 PrivateMapKey 的有效期设置得比较短，例如“5分钟”，就会很容易触发校验失败进而导致用户被踢出房间。
要解决这个问题，除了可以延长有效期（例如将“5分钟”改成“6小时”），还可以在观众通过 `switchRole` 将自己的身份切换成主播之前，重新向您的服务器申请一个 privateMapKey，并调用 SDK 的实验性接口 `updatePrivateMapKey` 将其更新到 SDK 中，示例代码如下：

Android

iOS

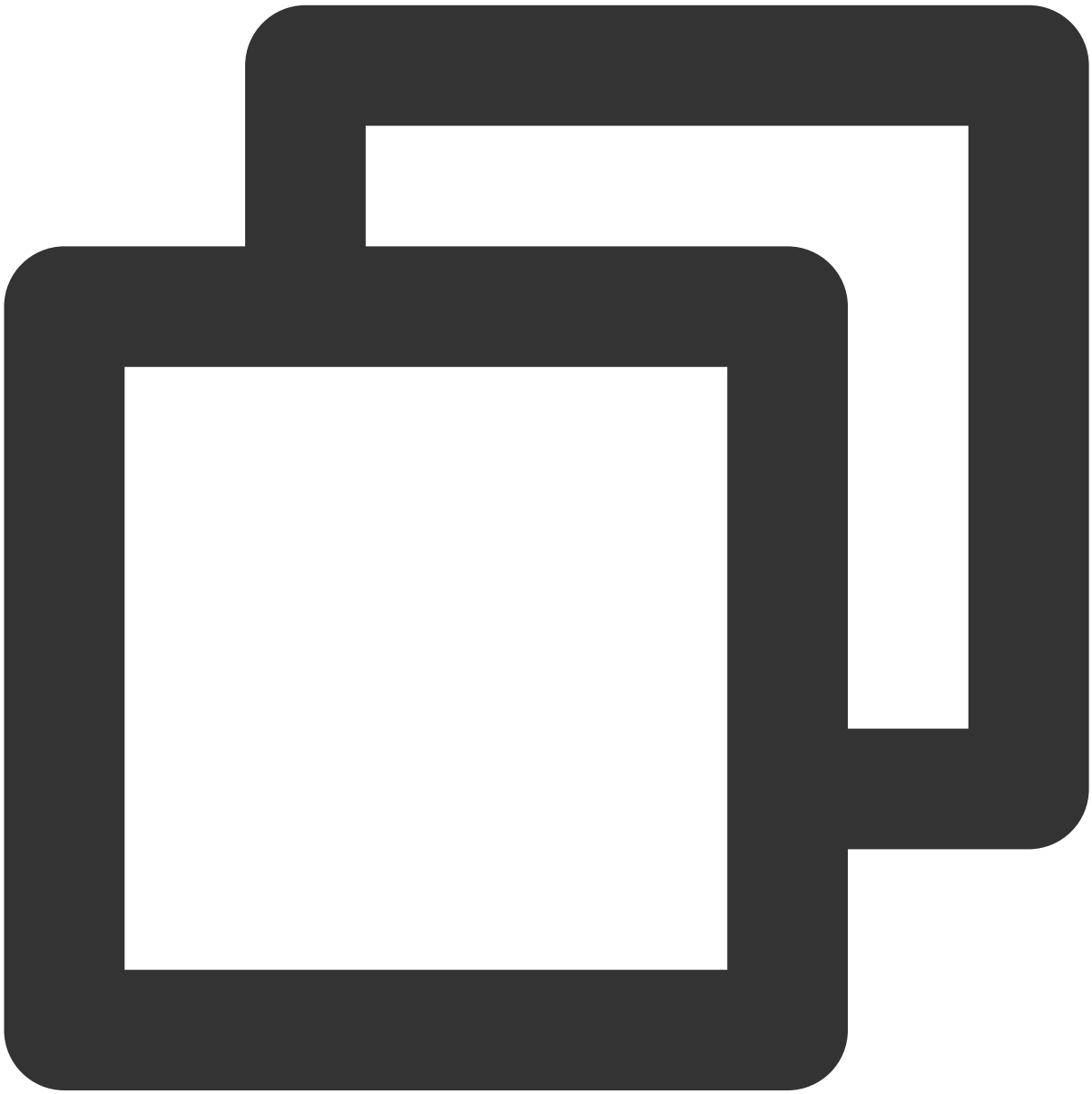
C++

C#



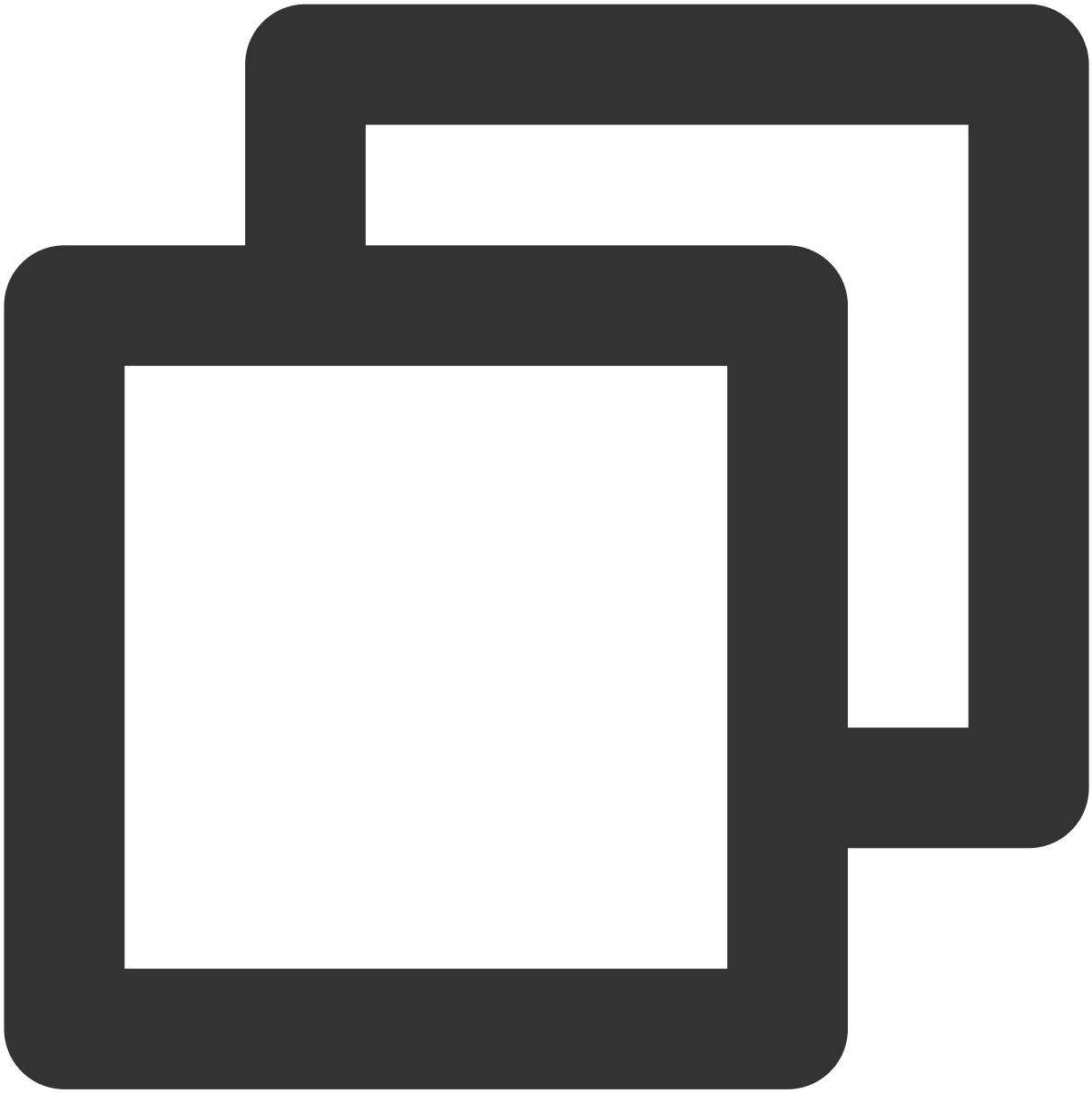
```
JSONObject jsonObject = new JSONObject();
try {
    jsonObject.put("api", "updatePrivateMapKey");
    JSONObject params = new JSONObject();
    params.put("privateMapKey", "xxxxx"); // 填写新的 privateMapKey
    jsonObject.put("params", params);
```

```
    mTRTCCloud.callExperimentalAPI(jsonObject.toString());
} catch (JSONException e) {
    e.printStackTrace();
}
```

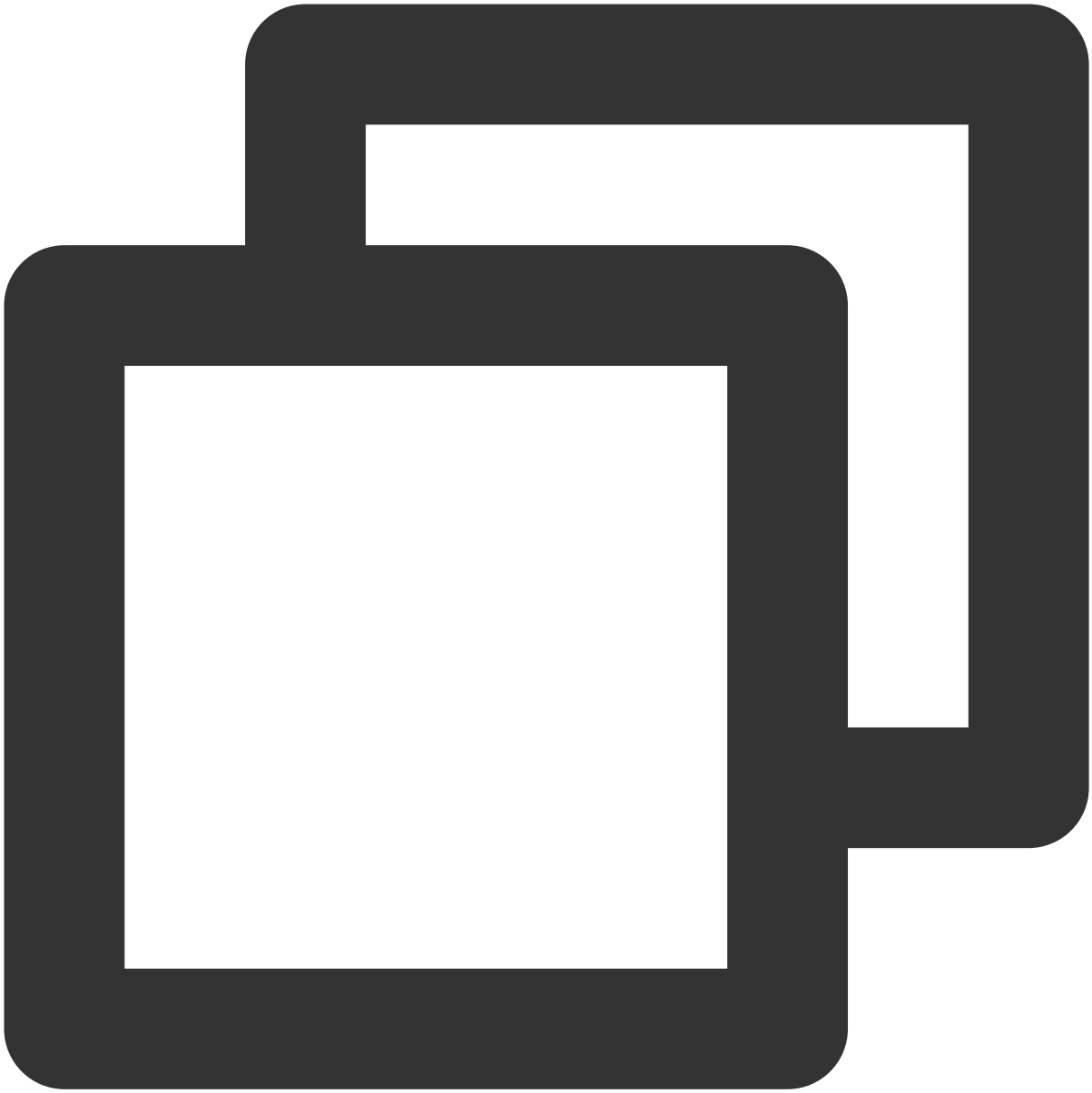


```
NSMutableDictionary *params = [[NSMutableDictionary alloc] init];
[params setObject:@"xxxxx" forKey:@"privateMapKey"]; // 填写新的 privateMapKey
NSDictionary *dic = @{@"api": @"updatePrivateMapKey", @"params": params};
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:dic options:0 error:NULL
NSString *jsonStr = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEn
```

```
[WXTRTCCloud sharedInstance] callExperimentalAPI:jsonStr];
```



```
std::string api = "{\\"api\\":\\"updatePrivateMapKey\\",\\"params\\":{\\"privateMap
TRTCCloudCore::GetInstance()->getTRTCCloud()->callExperimentalAPI(api.c_str());
```



```
std::string api = "{\\"api\\":\\"updatePrivateMapKey\\",\\"params\\":{\\"privateMap
mTRTCCloud.callExperimentalAPI(api);
```

常见问题

1. 线上的房间为什么都进不去了？

房间权限控制一旦开启后，当前 SDKAppid 下的房间就需要在 `TRTCParams` 中设置 privateMapKey 才能进入，所以如果您线上业务正在运营中，并且线上版本并没有加入 privateMapKey 的相关逻辑，请不要开启此开关。

2. PrivateMapKey 和 UserSig 有什么区别？

UserSig 是 TRTCParams 的必选项，作用是检查当前用户是否有权使用 TRTC 云服务，用于防止攻击者盗用您的 SDKAppid 账号内的流量。
PrivateMapKey 是 TRTCParams 的非必选项，作用是检查当前用户是否有权进入指定 roomid 的房间，以及该用户在该房间所能具备的权限，当您的业务需要对用户进行身份区分的时候才有必要开启。

输入媒体流进房

最近更新时间：2024-08-12 17:57:06

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 RTC-Engine 包月套餐 **基础版**或**专业版**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 混流转码与旁路转推计费说明。

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。

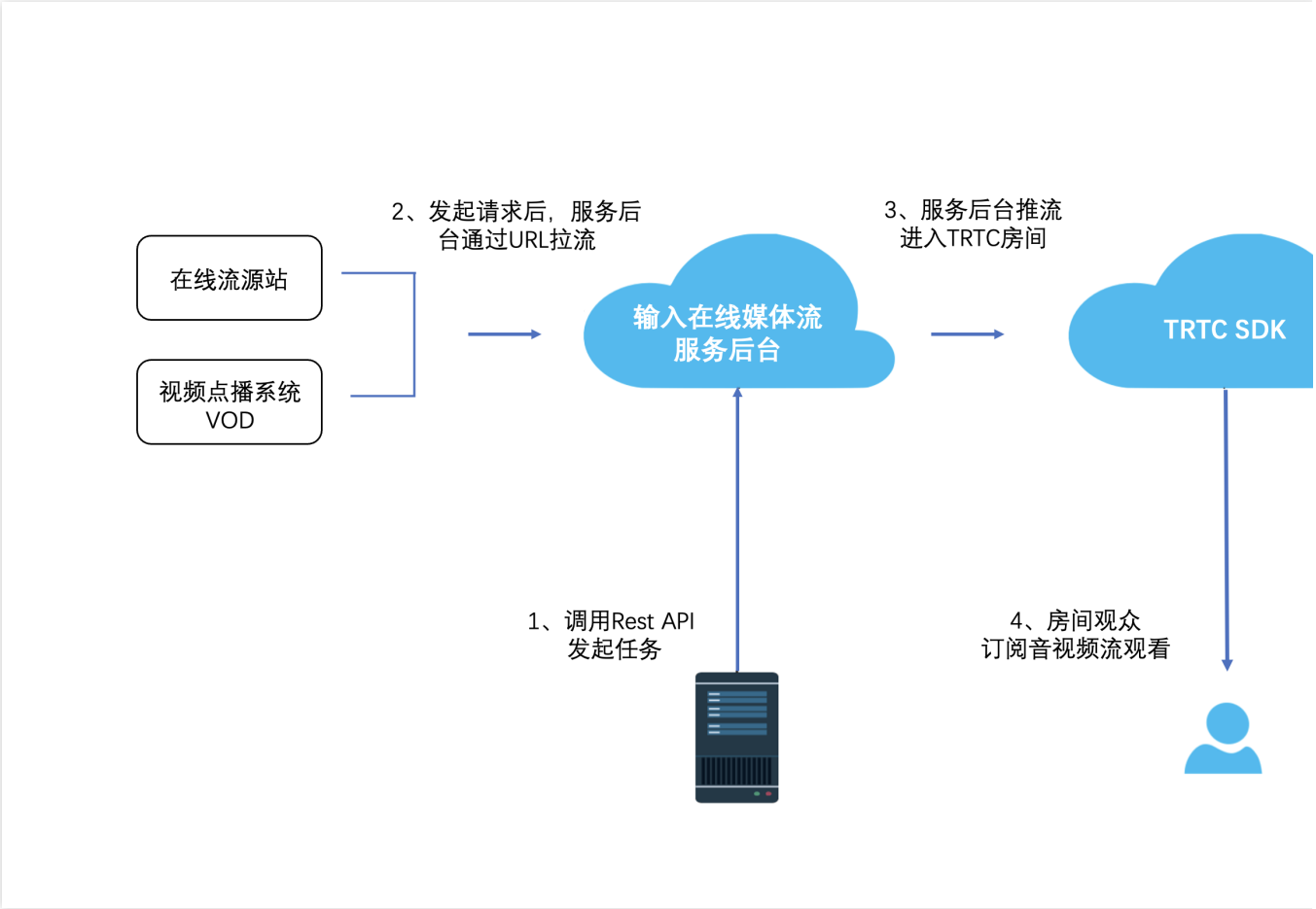
输入在线媒体流

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：详细输入在线媒体流事件，前往查看。

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

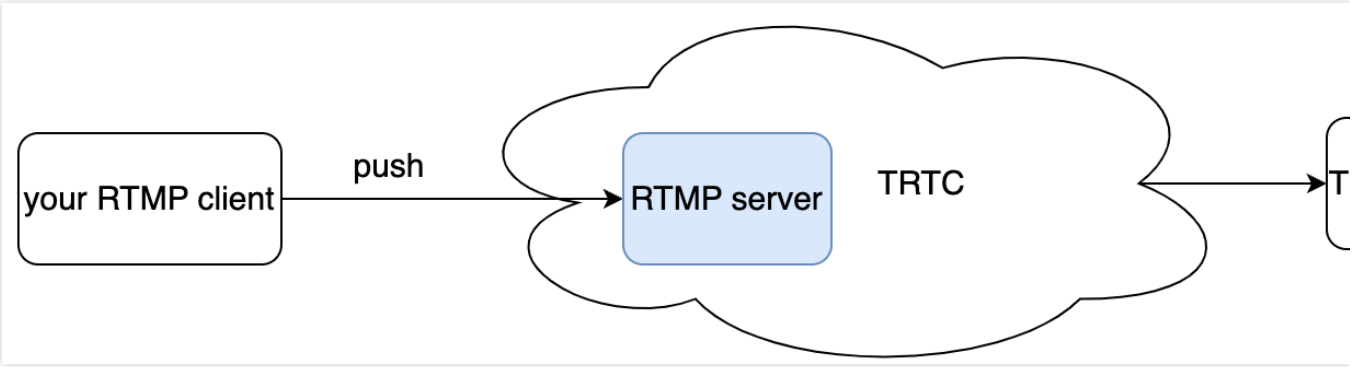
TRTC 支持将**本地媒体文件、音视频设备采集音视频**通过 RTMP 标准协议推流到 TRTC 房间内。为降低客户接入门槛，您可根据实际情况选择安装 OBS 、FFmpeg 或其他 RTMP 库进行推流。OBS 是一款好用的第三方开源程序直播流媒体内容制作软件，为用户提供免费使用，它可支持 OS X、Windows、Linux 操作系统，适用多种直播场景，满足大部分直播行为的操作需求，您可以到 OBS 官网 下载最新版本软件，使用 OBS 推流时无需安装插件。

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。

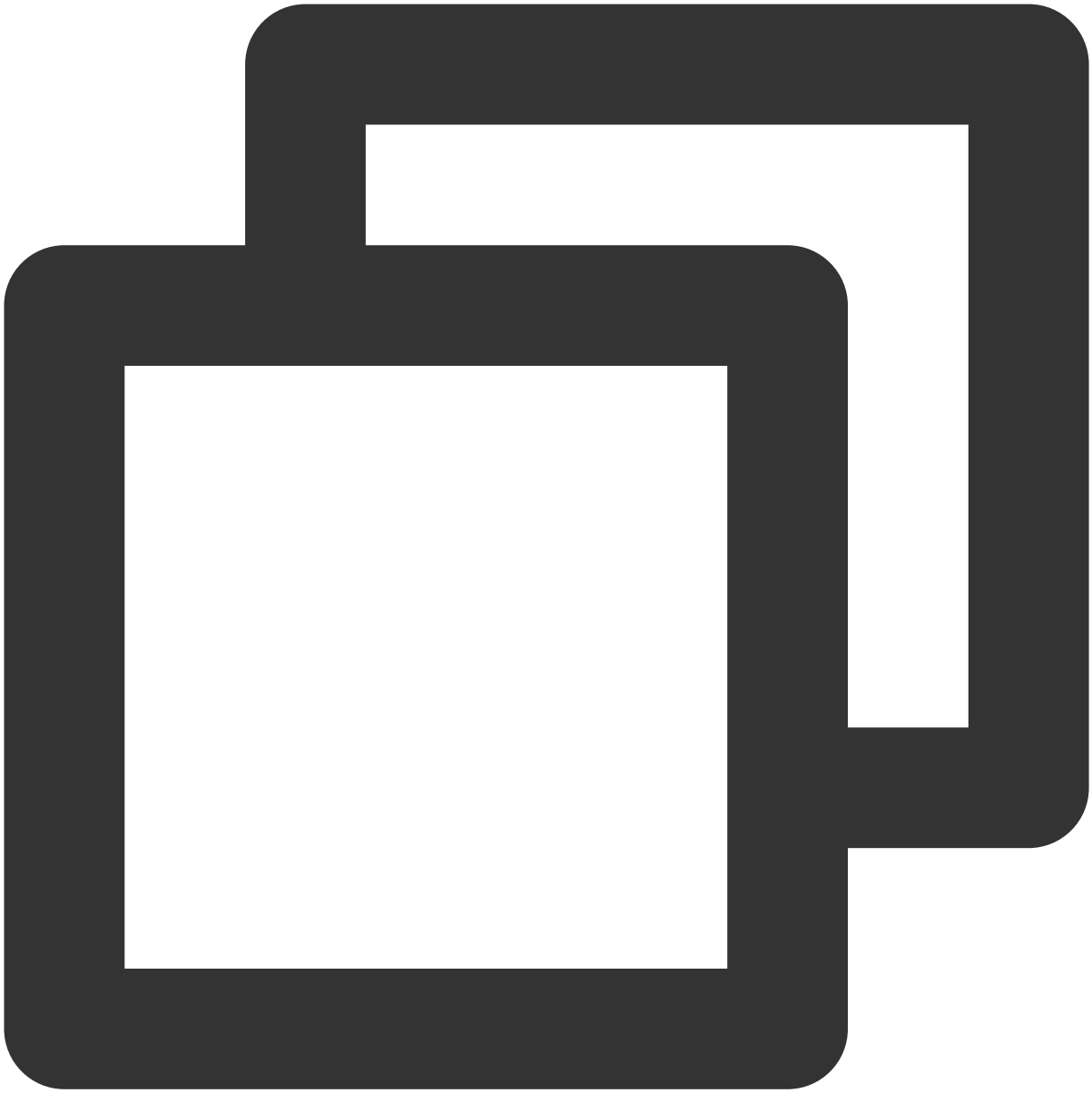
网络架构

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。



流地址生成

推流地址



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

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。

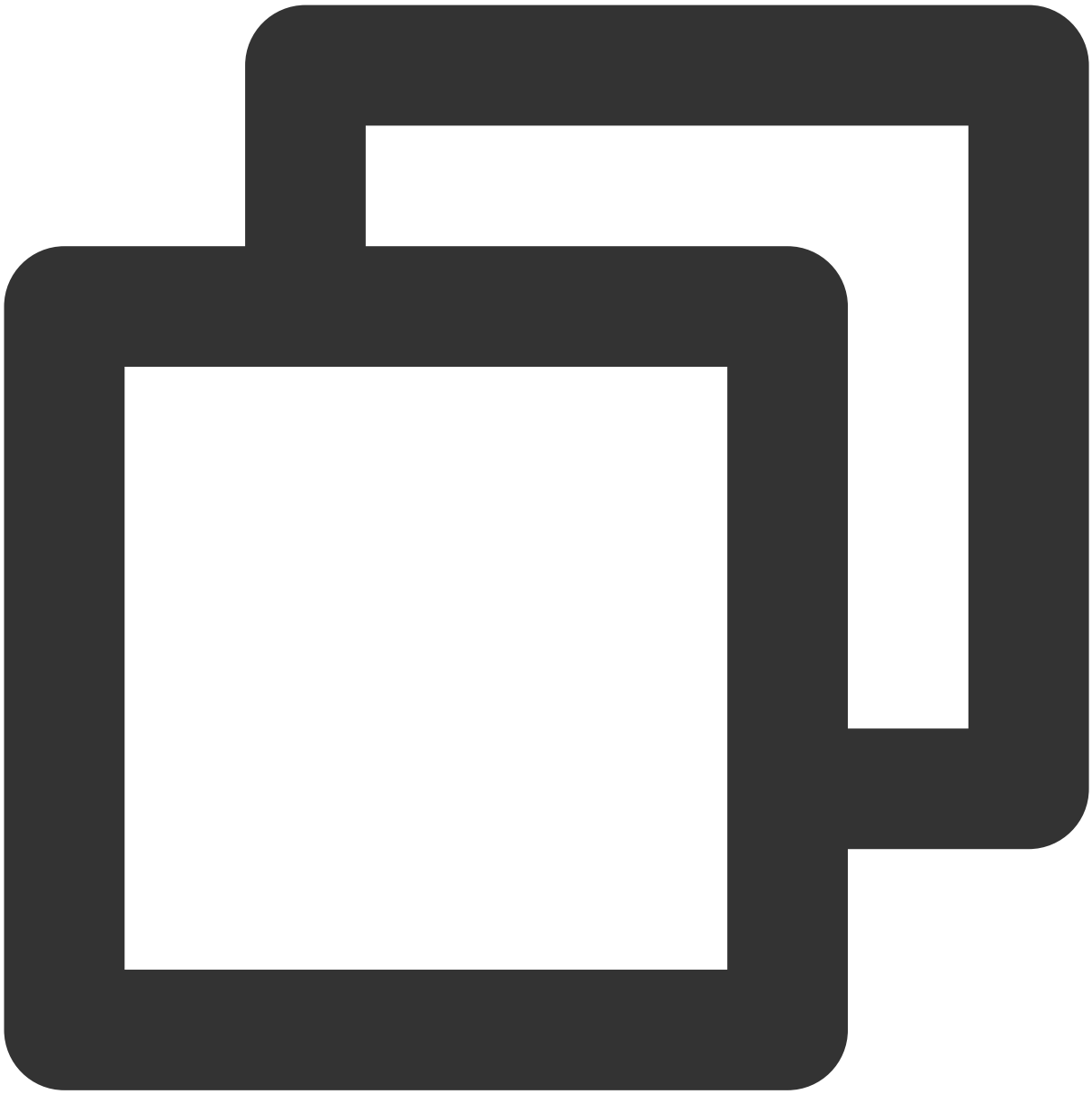
警告：

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://intl-rtmp.rtc.qq.com/push/hello-string-room?sdkappid=140*****66&userid=*****
```

使用示例

您可以使用支持 RTMP 协议的软件或者代码库推拉流，下面列举几种。

OBS 推流

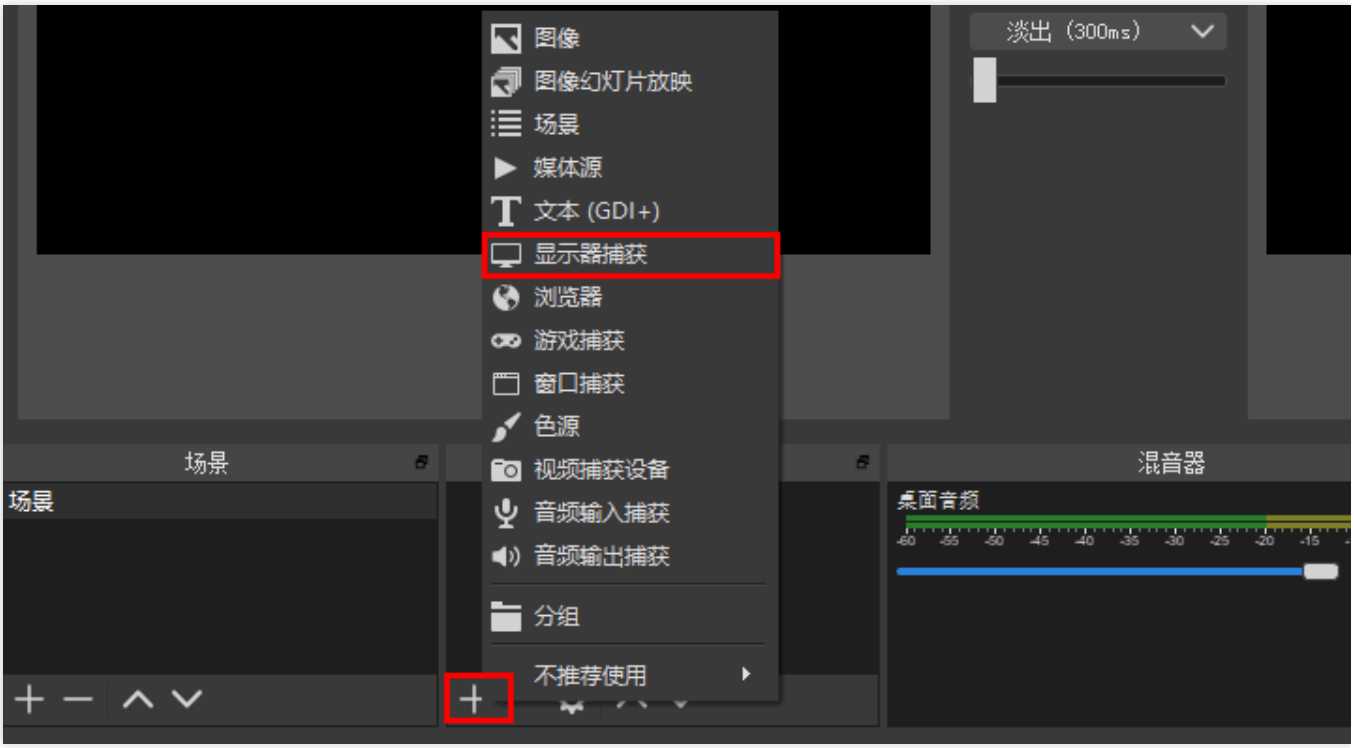
准备工作

安装并打开 OBS 工具进行下述操作。

步骤1：选择输入源

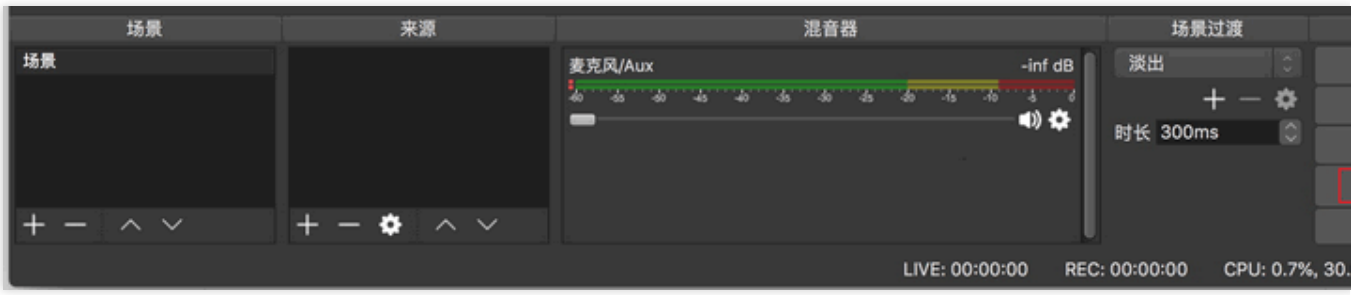
查看底部工具栏的**来源**标签，单击**+**，根据您的业务需要选择输入源。常用来源输入有：

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）

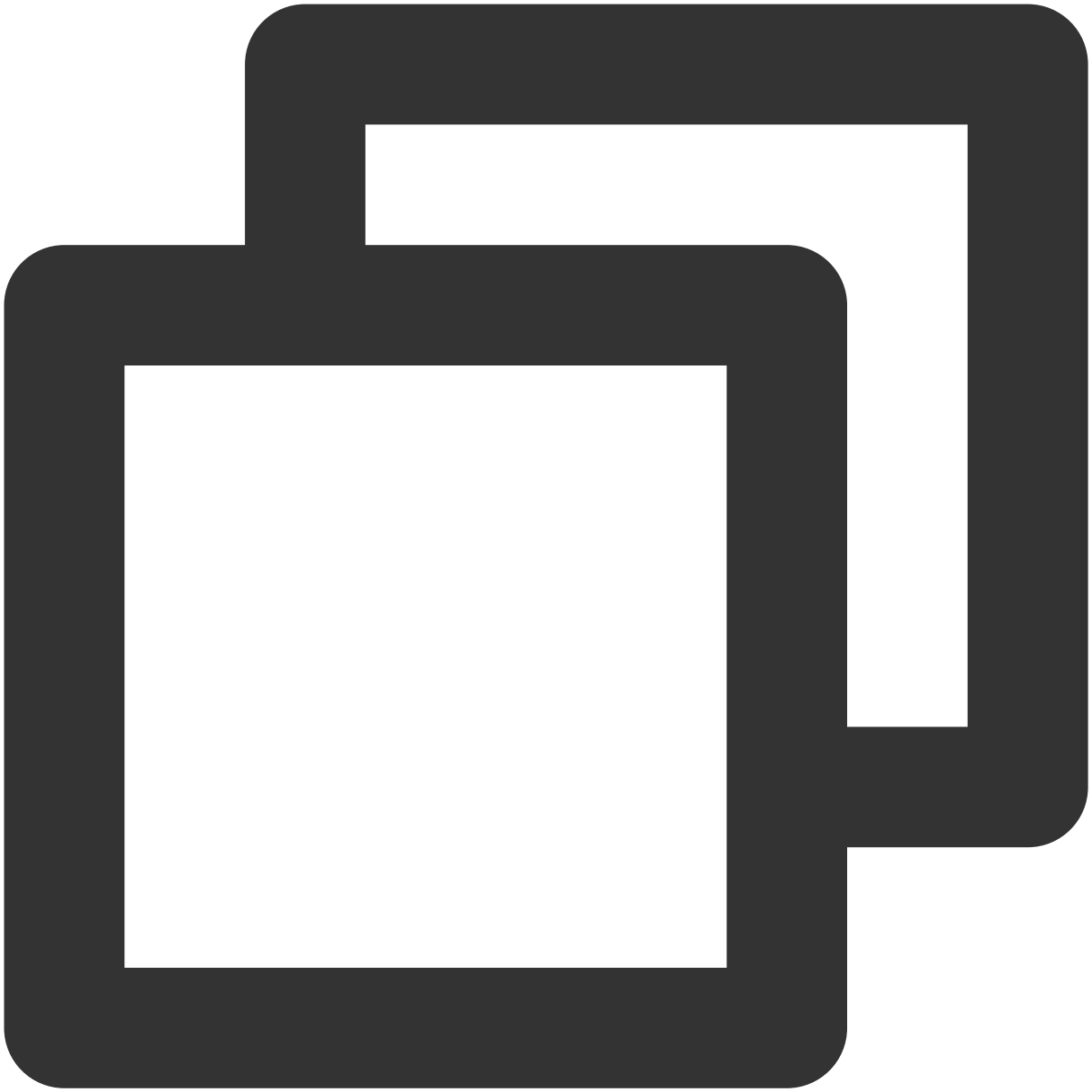


步骤2：设置推流参数

1. 通过底部工具栏的**控件 > 设置**按钮进入设置界面。

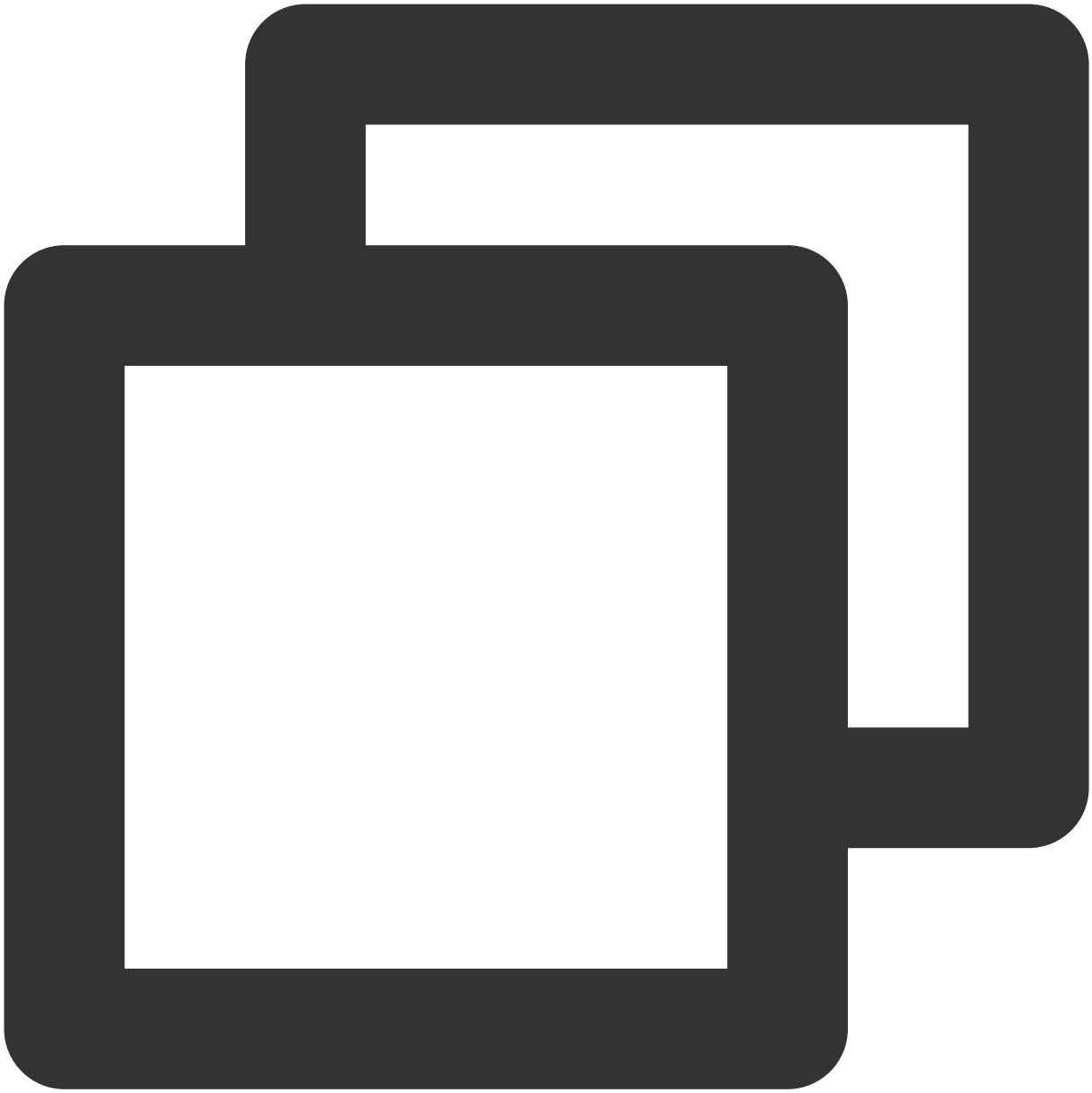


2. 单击**推流**进入推流设置页签，选择服务类型为**自定义**。
3. 服务器填写：`rtmp://intl-rtmp.rtc.qq.com/push/` 。
4. 填写串流密钥格式如下：

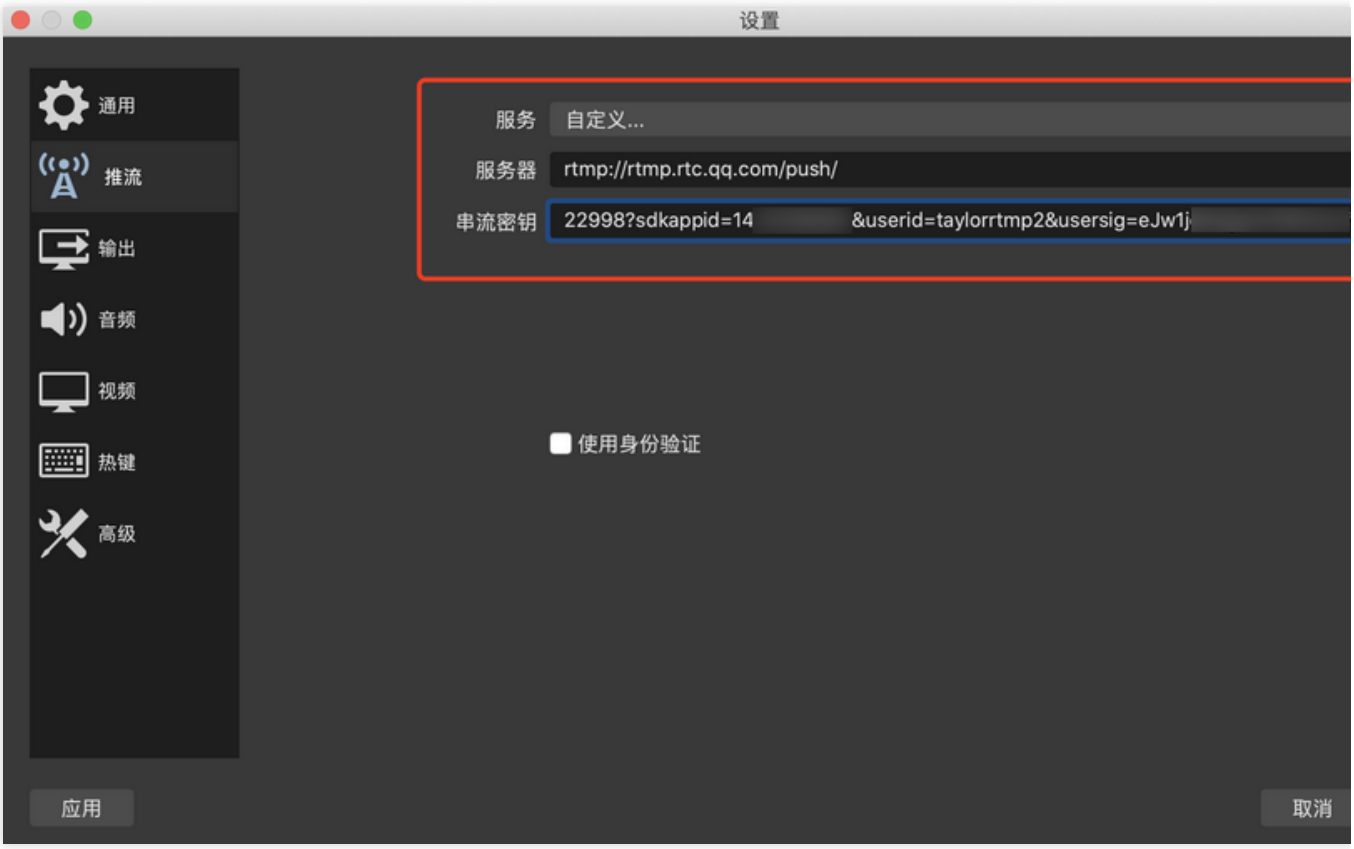


```
房间号?sdkappid=应用&userid=用户名&usersig=签名
```

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**********
```



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。

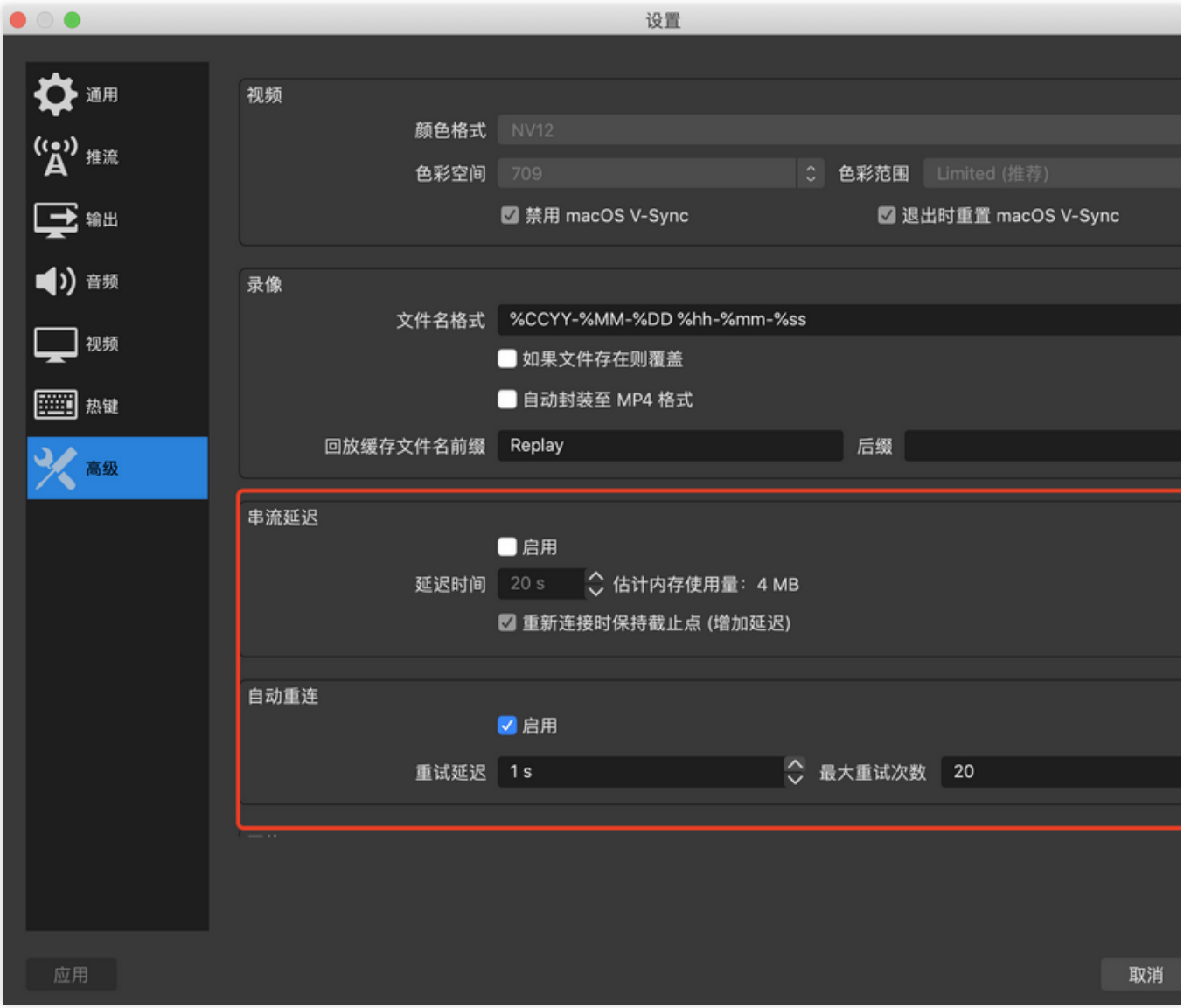
步骤4：设置视频选项

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

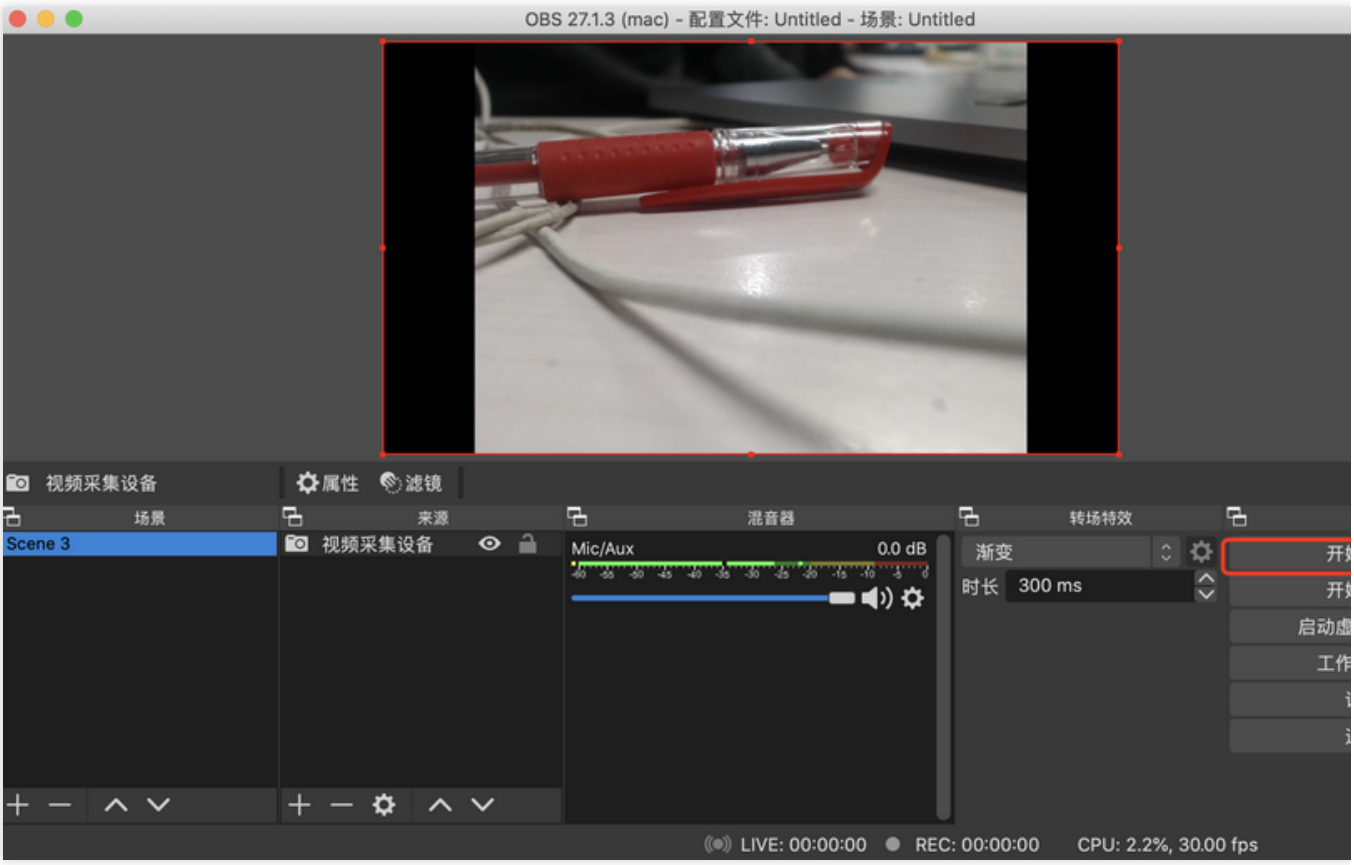
建议不启用**串流延迟**以减少端到端延迟。

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。

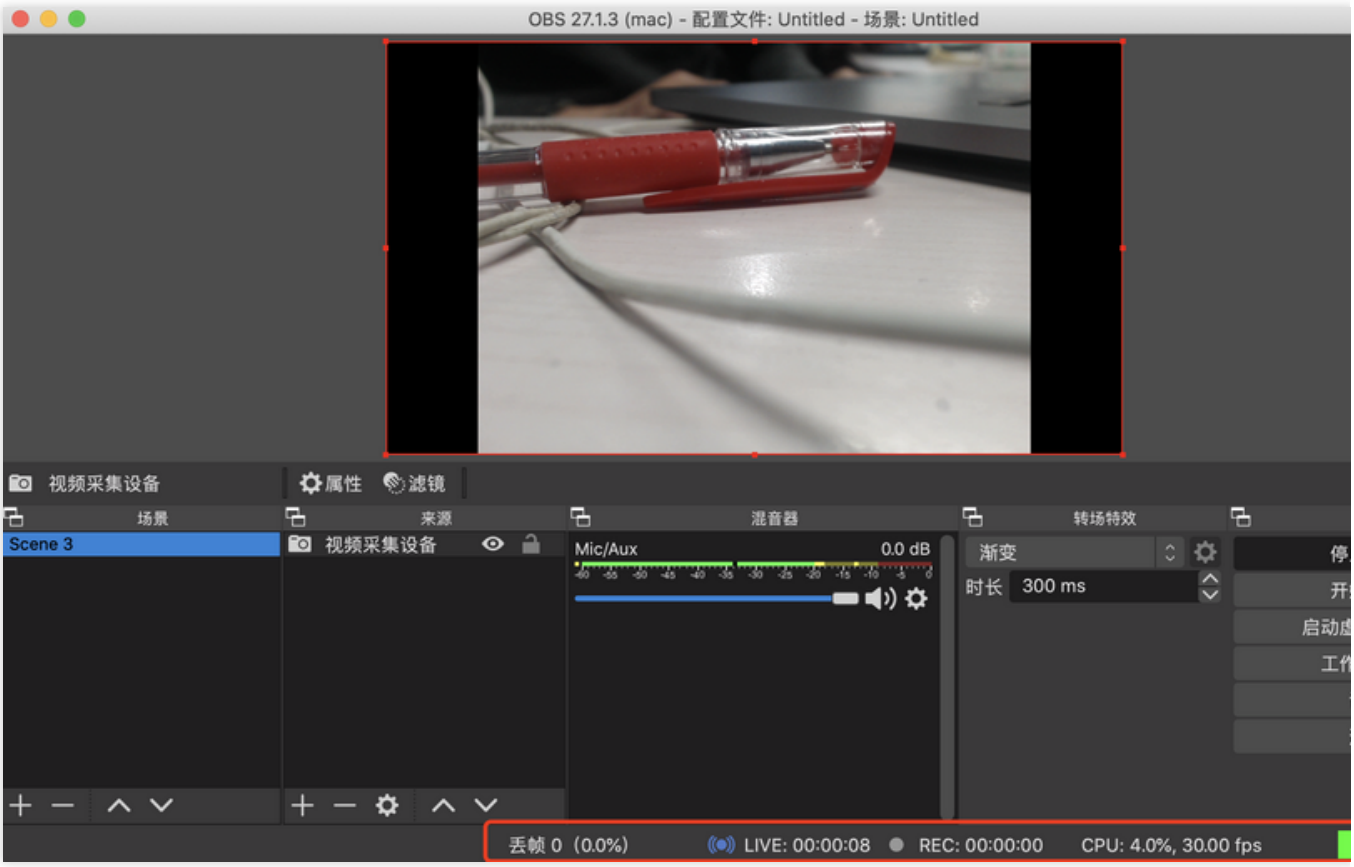


步骤6：单击推流

1. 查看 OBS 底部工具栏的**控件**，单击**开始推流**。

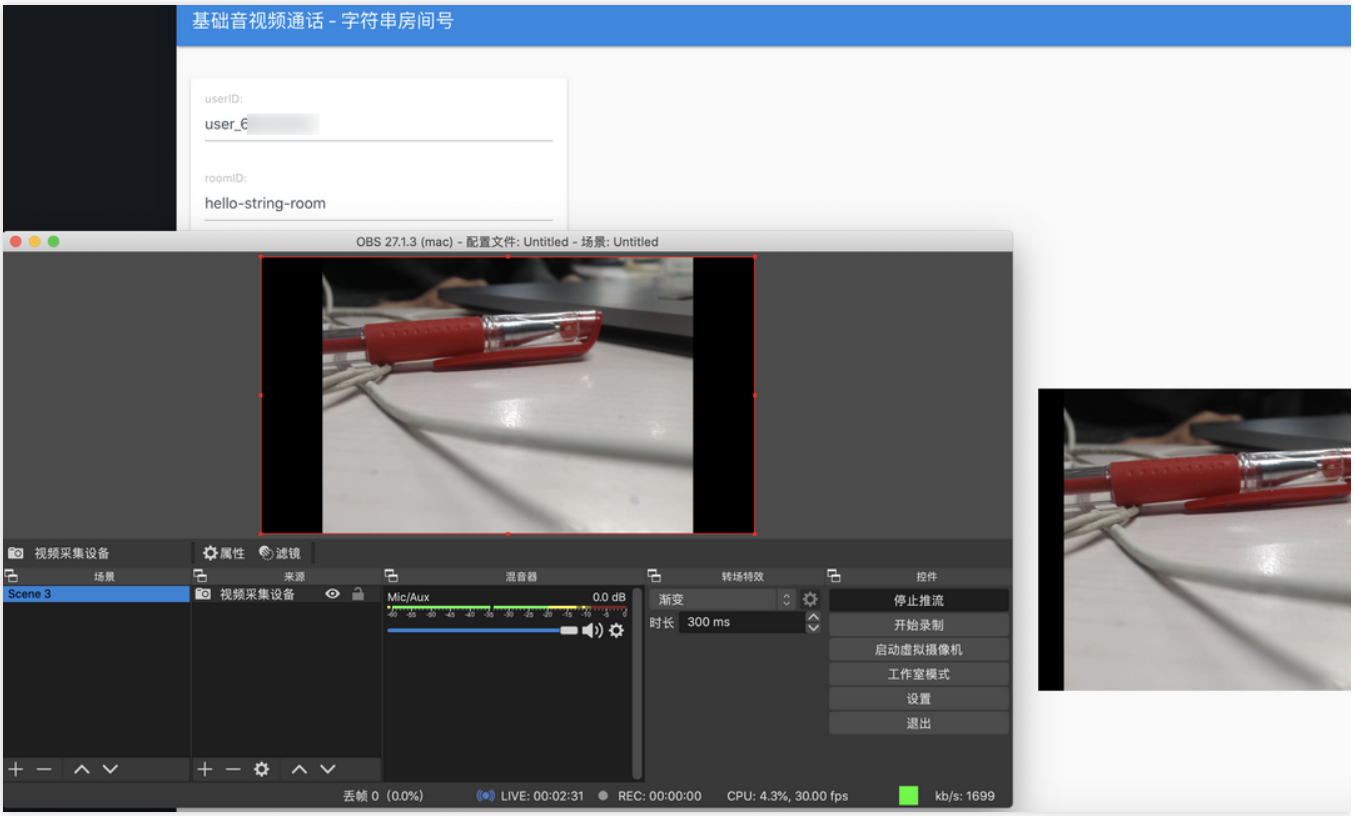


2. 推流成功后，正常情况在界面底部会展示推流状态，TRTC 控制台仪表盘 上有该用户进房记录。



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。

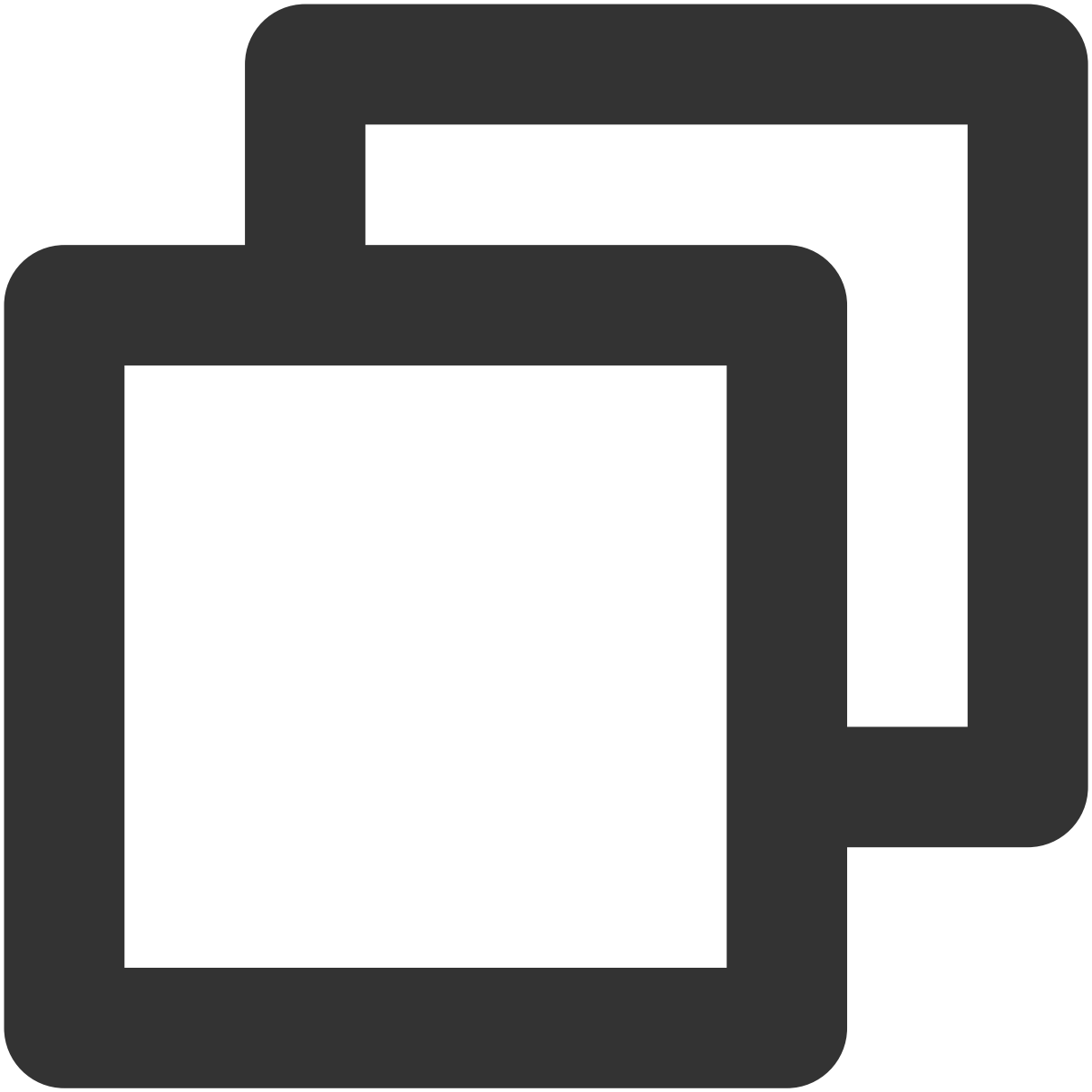


FFmpeg 推流

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_optio
```

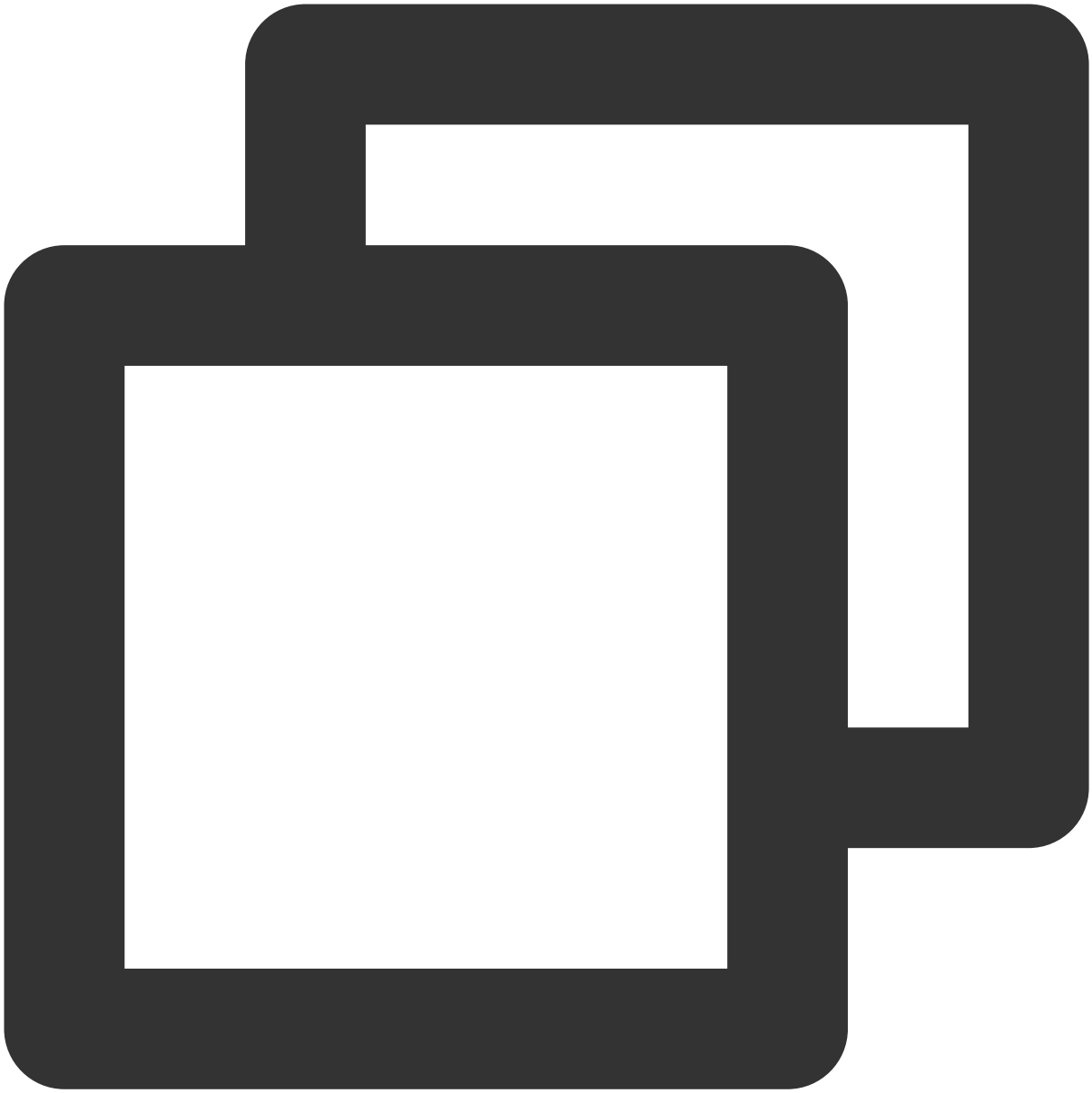
常见的 FFmpeg 选项

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

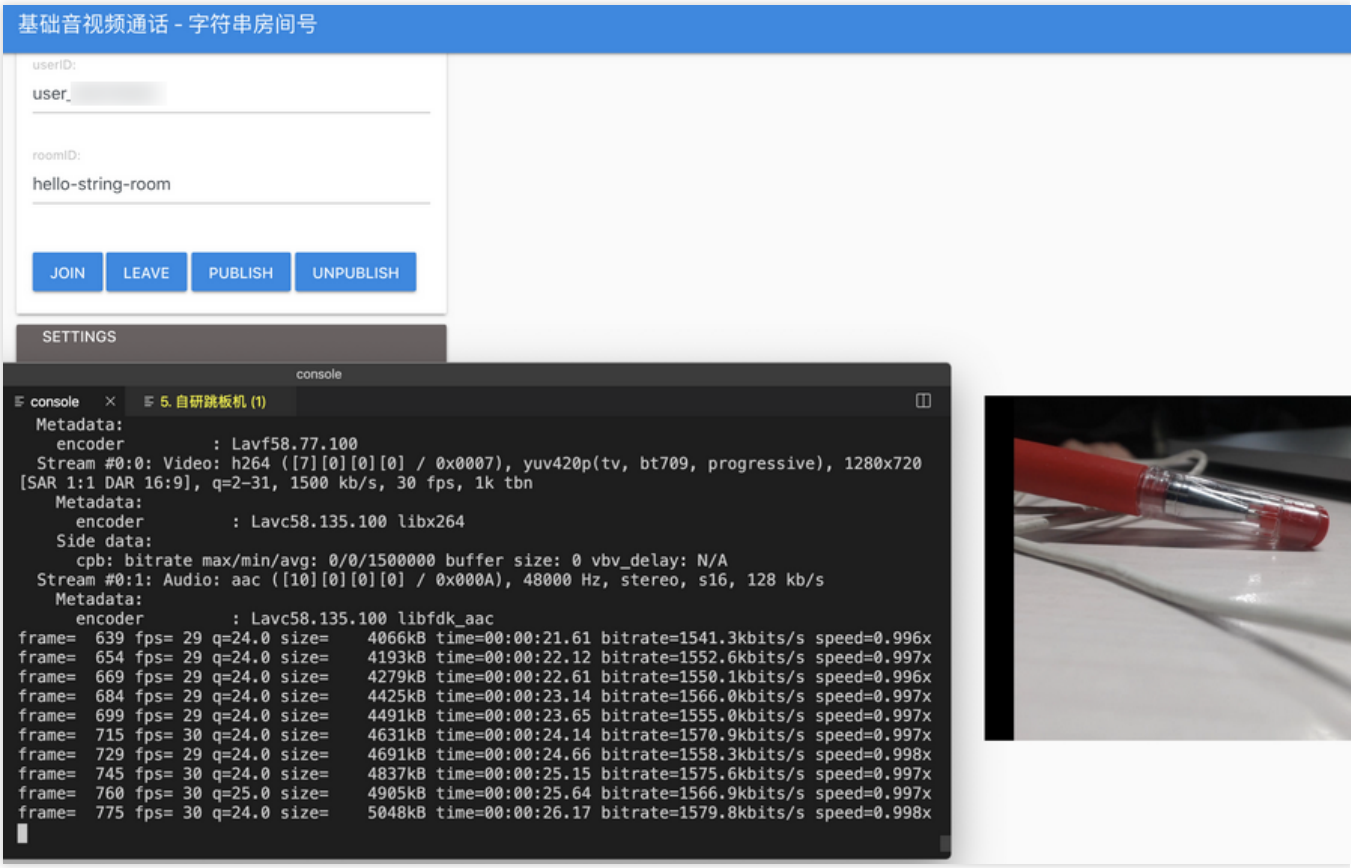
下面的例子是读取文件推到 TRTC，注意 URL 两边加引号。



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

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。


基础音视频通话 - 字符串房间号
userID:
user_
roomID:
hello-string-room
JOIN
LEAVE
PUBLISH
UNPUBLISH
SETTINGS
console
console
5. 自研跳板机 (1)
Metadata:
encoder : Lavf58.77.100
Stream #0:0: Video: h264 ([7][0][0][0] / 0x0007), yuv420p(tv, bt709, progressive), 1280x720 [SAR 1:1 DAR 16:9], q=2-31, 1500 kb/s, 30 fps, 1k tbn
Metadata:
encoder : Lavc58.135.100 libx264
Side data:
cpb: bitrate max/min/avg: 0/0/1500000 buffer size: 0 vbv_delay: N/A
Stream #0:1: Audio: aac ([10][0][0][0] / 0x000A), 48000 Hz, stereo, s16, 128 kb/s
Metadata:
encoder : Lavc58.135.100 libfdk_aac
frame= 639 fps= 29 q=24.0 size= 4066kB time=00:00:21.61 bitrate=1541.3kbits/s speed=0.996x
frame= 654 fps= 29 q=24.0 size= 4193kB time=00:00:22.12 bitrate=1552.6kbits/s speed=0.997x
frame= 669 fps= 29 q=24.0 size= 4279kB time=00:00:22.61 bitrate=1550.1kbits/s speed=0.996x
frame= 684 fps= 29 q=24.0 size= 4425kB time=00:00:23.14 bitrate=1566.0kbits/s speed=0.997x
frame= 699 fps= 29 q=24.0 size= 4491kB time=00:00:23.65 bitrate=1555.0kbits/s speed=0.997x
frame= 715 fps= 30 q=24.0 size= 4631kB time=00:00:24.14 bitrate=1570.9kbits/s speed=0.997x
frame= 729 fps= 29 q=24.0 size= 4691kB time=00:00:24.66 bitrate=1558.3kbits/s speed=0.998x
frame= 745 fps= 30 q=24.0 size= 4837kB time=00:00:25.15 bitrate=1575.6kbits/s speed=0.997x
frame= 760 fps= 30 q=25.0 size= 4905kB time=00:00:25.64 bitrate=1566.9kbits/s speed=0.997x
frame= 775 fps= 30 q=24.0 size= 5048kB time=00:00:26.17 bitrate=1579.8kbits/s speed=0.998x

使用美颜特效

Web

最近更新时间：2024-04-16 16:35:15

功能描述

本文将介绍如何在音视频互动过程中实现美颜特效的功能。

前提条件

需要使用美颜功能的 SDKAppID，请确保已开通 **RTC-Engine 专业版包月套餐**。包月套餐相关说明请参见文档 RTC-Engine 包月套餐计费说明。

TRTC Web SDK 版本 ≥ 5.5.0。

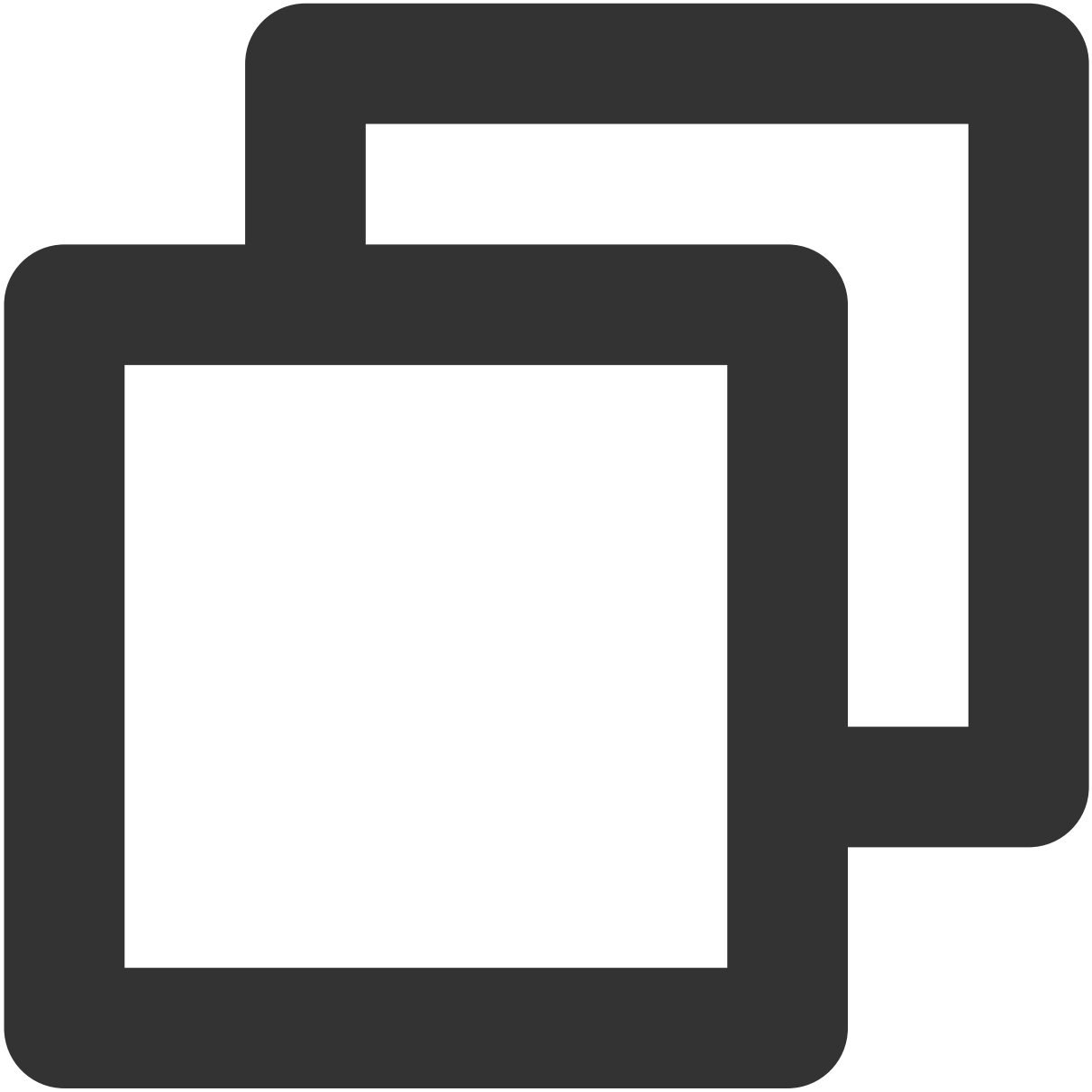
Web 平台各系统及配置要求如下表：

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	
	Android	Chrome浏览器 火狐浏览器	30	高端机（高通骁龙 8 Gen1 等）	建议使用 Chrome 或火狐浏览器等主流浏览器
			20	中端机（天玑 8000-MAX 等）	
			10	低端机（高通骁龙 660 等）	
	iOS	Chrome Safari Firefox	30	iPhone 13	要求 iOS 14.4 以上系统- 建议使用 Chrome 或 Safari 浏览器
			20	iPhone xr	

使用步骤

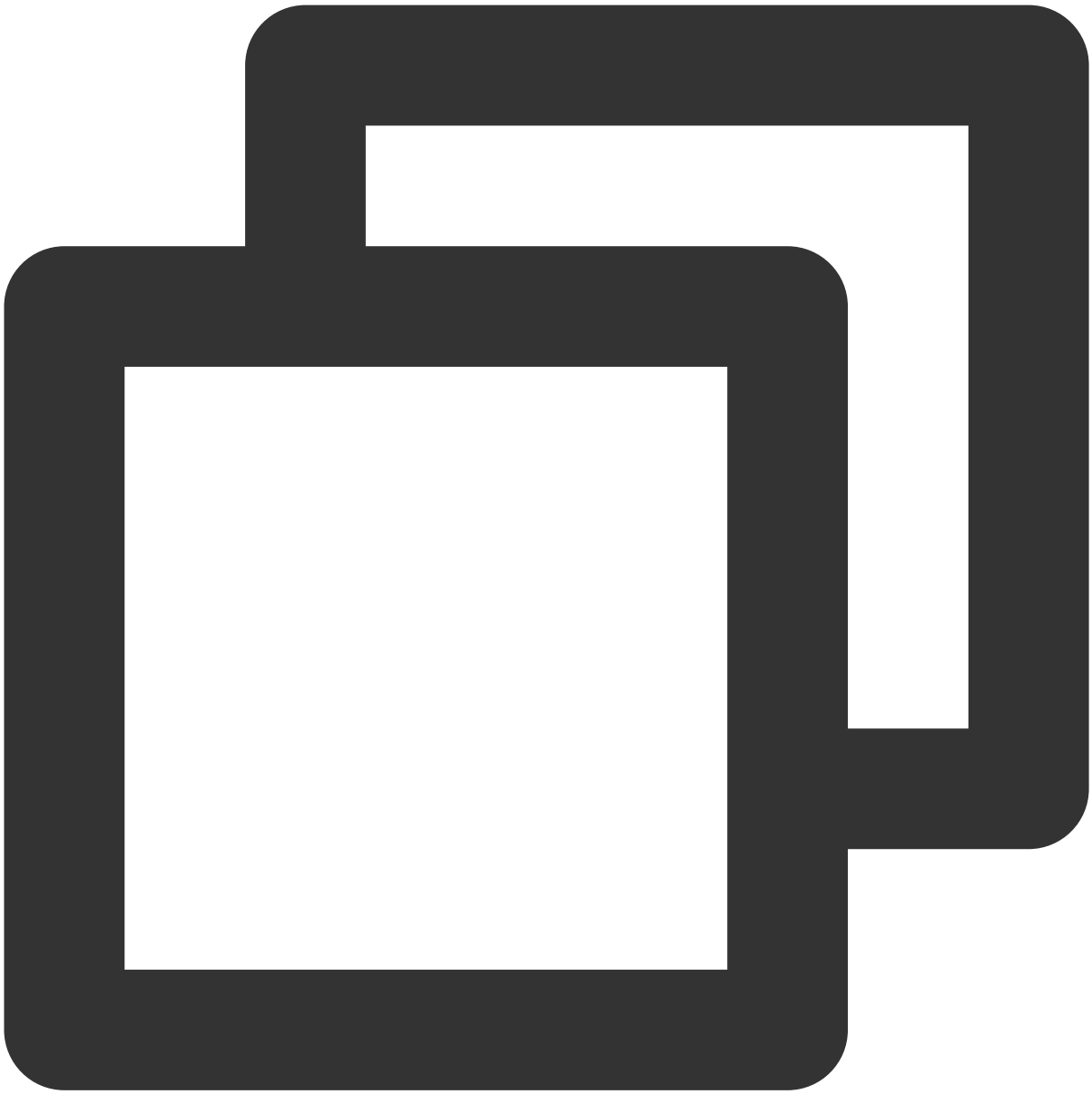
一、注册插件

在使用之前，首先需要注册插件。以下是注册插件的示例代码：



```
import TRTC from 'trtc-sdk-v5';
import { Beauty } from 'trtc-sdk-v5/plugins/video-effect/beauty';

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

二、开启本地摄像头



```
await trtc.startLocalVideo({
  view: 'local-video' // 填入自己业务需要的 div id
});
```

三、使用美颜和内置 2D 特效

美颜效果

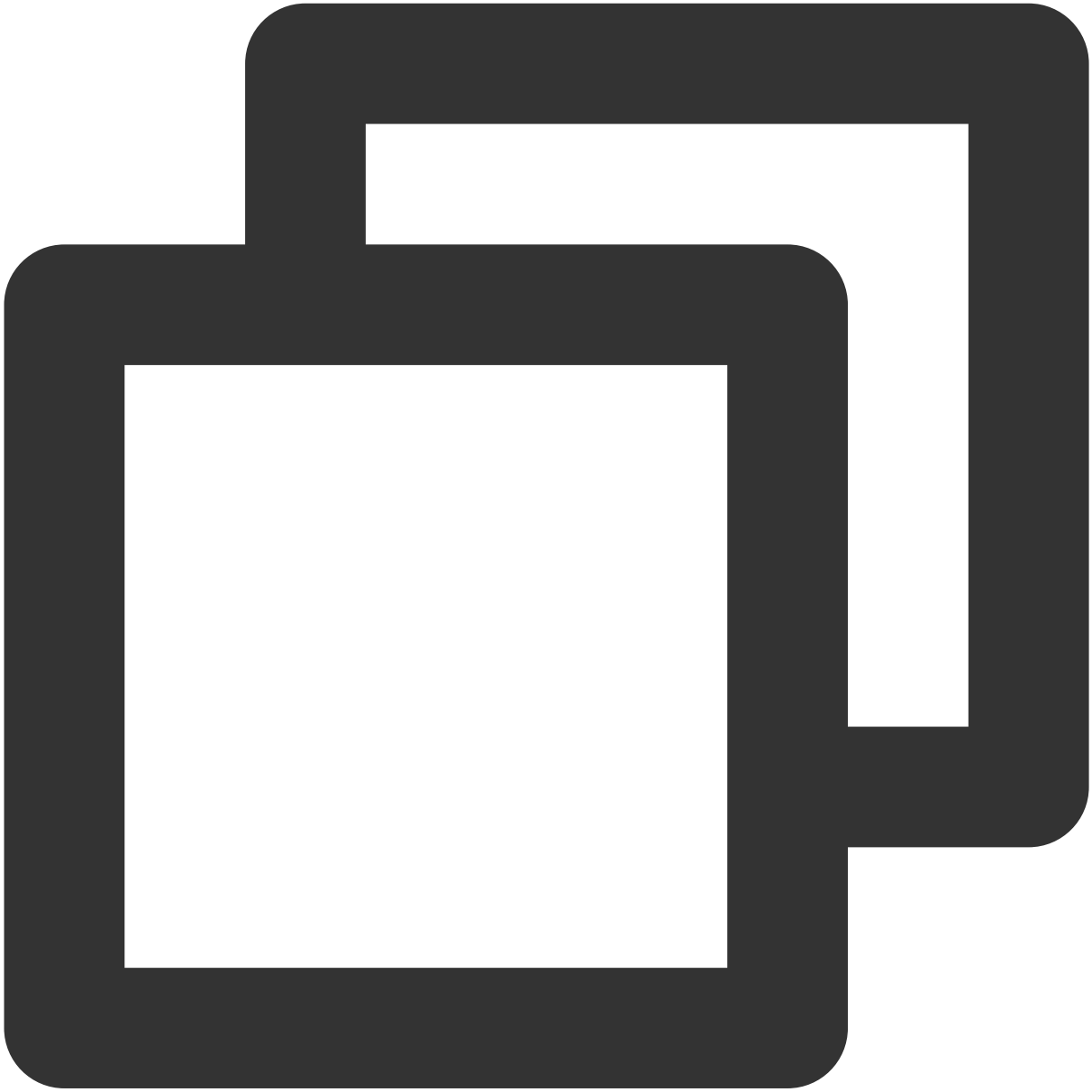
美颜功能的开启、更新和停止通过三个接口来控制：

`trtc.startPlugin('Beauty', options)` 开启美颜。

`trtc.updatePlugin('Beauty', options)` 更新美颜参数。

`trtc.stopPlugin('Beauty')` 停止美颜。

其中 `options` 参数支持 6 种美颜参数：



```
type BeautifyOptions = {
  whiten?: number; // 美白 0-1
  dermabrasion?: number; // 磨皮 0-1
  lift?: number; // 窄脸 0-1
  shave?: number; // 削脸 0-1
```

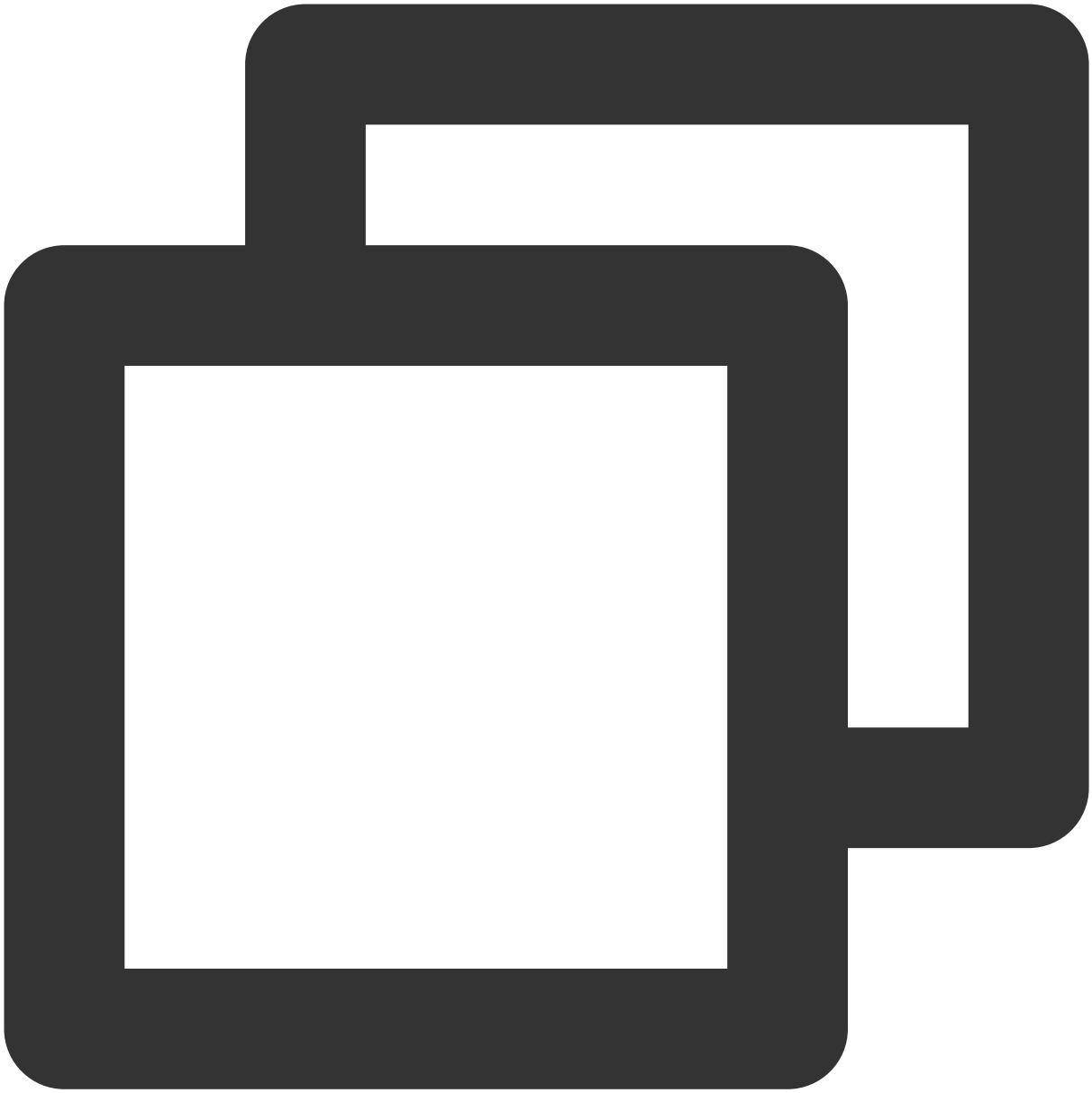
```
  eye?: number; // 大眼 0-1
  chin?: number; // 下巴 0-1
}
```

内置 2D 特效贴纸

本插件提供内置的 2D 特效贴纸，下面是列表：

特效 ID	名称
EDE5D18065451445	奶瓶面膜
7A62218064EF7959	毛毡发箍
B4D63180653948C3	刘海发带
D7C6418064B90F4C	可爱喵
1D6EB18069D76A62	波点女孩
CBE7618065D9D76F	爱心烟花
9B86618065596018	可爱猪猪
428C518065369ACF	双麻花
B30E218064F7B397	元气贴纸
AE3C81806521B49B	兔兔酱

可以在参数中传入 `EffectListOptions` 类型的数组，将 `EffectId` 填入 `id` 字段，`intensity` 用于控制透明度。传入的 `EffectId` 对应的贴纸将会生效。更多信息请参考示例代码。

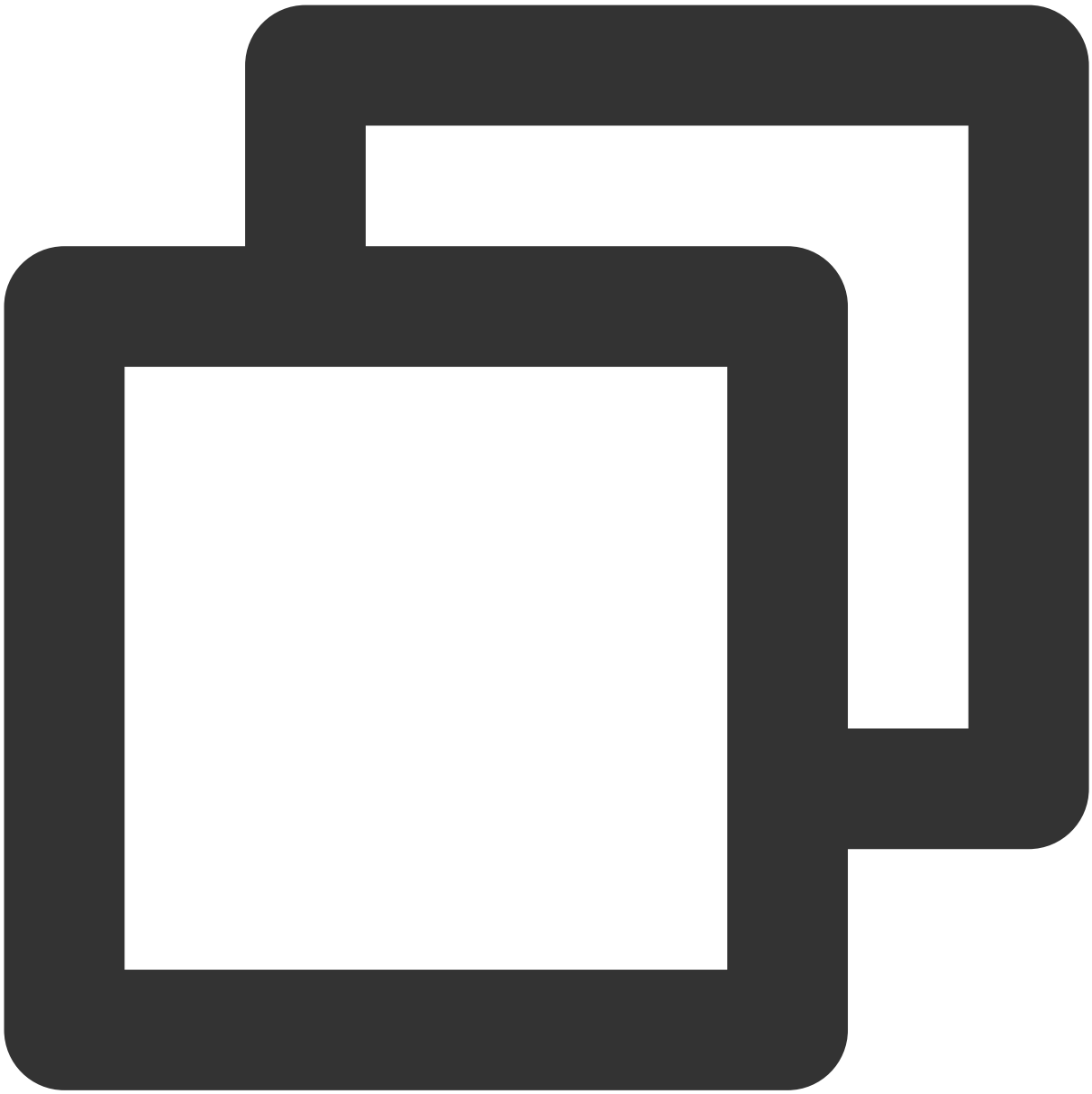


```
type EffectListOptions = {
  id: string;
  intensity?: number; // specify the strength of the effect
}
```

示例代码

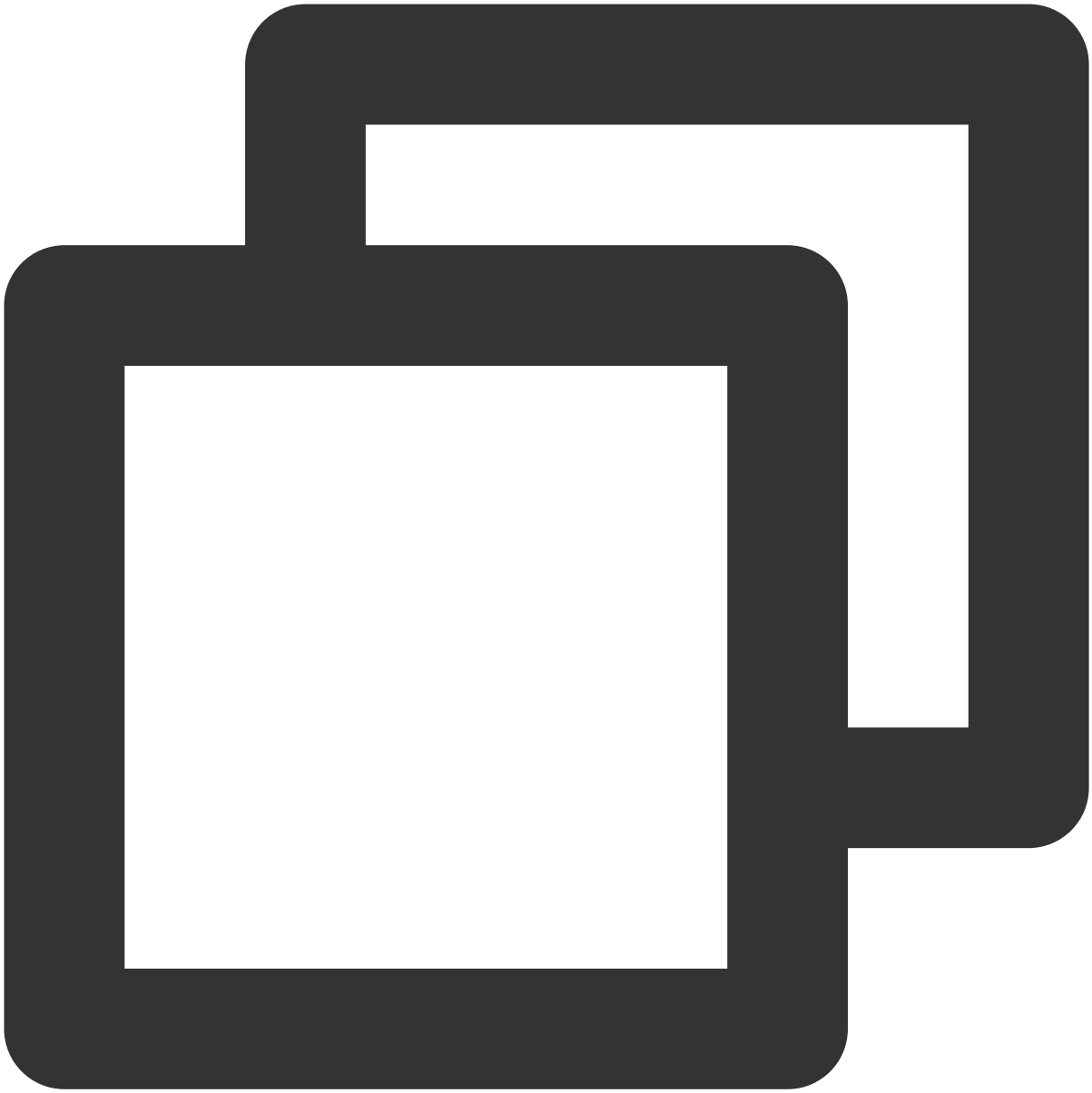
以下是代码示例：

开启插件时，`options` 还需要额外携带当前用户的登录信息（sdkAppId、userId、userSig）用作校验。



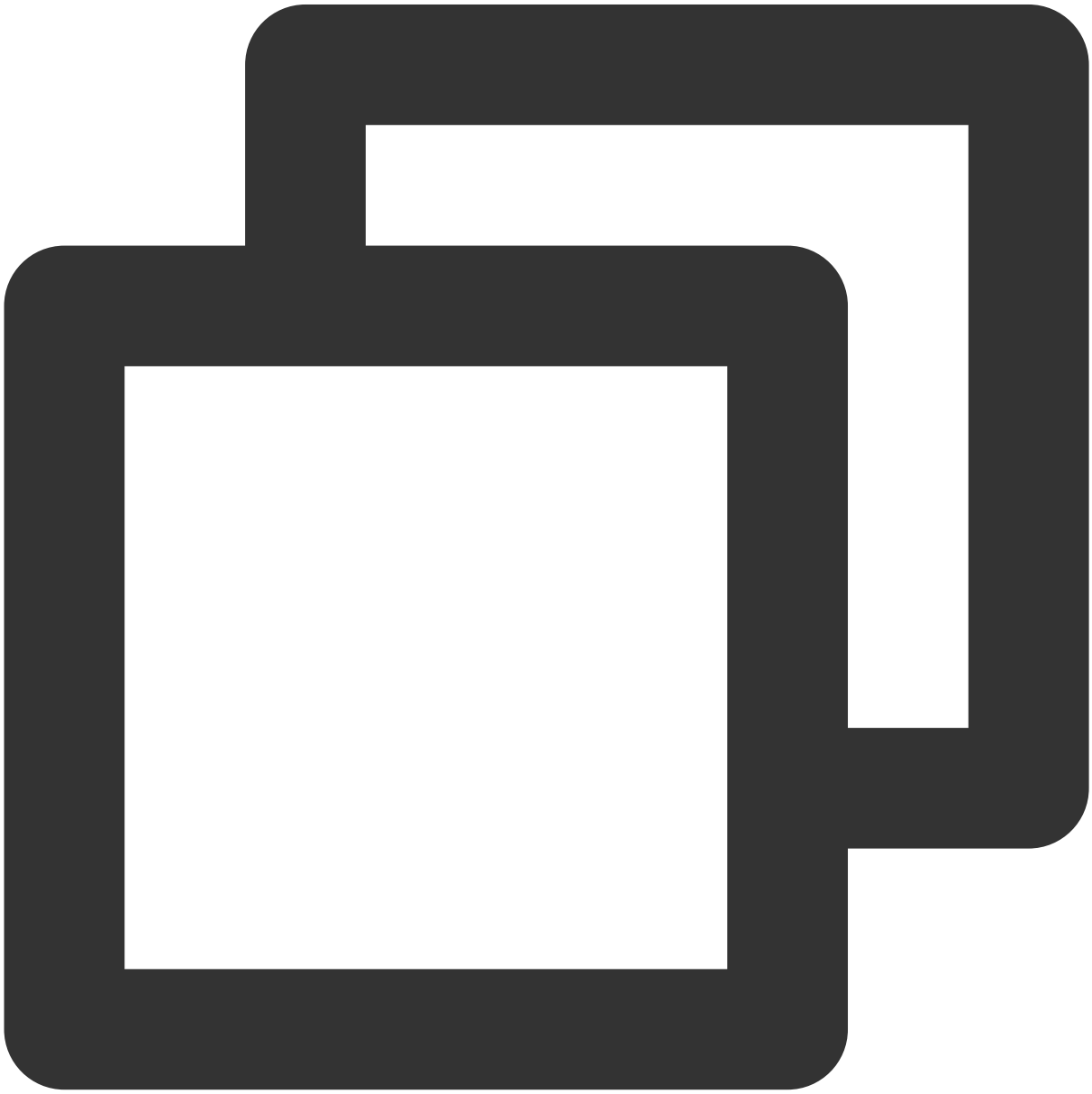
```
// 开启插件
const options = {
  sdkAppId: 0,
  userId: '',
  userSig: '',

  whiten: 0.5; // 美白 0-1
  dermabrasion: 0.5; // 磨皮 0-1
  lift: 0.5; // 窄脸 0-1
```

```
    shave: 0.5; // 削脸 0-1
    eye: 0.5; // 大眼 0-1
    chin: 0.5; // 下巴 0-1

    // 特效数组
    effect: [{
      id: '7A62218064EF7959', // 特效 ID
      intensity: 0.7 // 生效强度
    }]
}
try {
    await trtc.startPlugin('Beauty', options);
} catch (error) {
    console.error('Beauty start failed', error);
}
```



```
// 更新美颜参数
const options = {
  whiten: 0.5; // 美白 0-1
  dermabrasion: 0.5; // 磨皮 0-1
  lift: 0.5; // 窄脸 0-1
  shave: 0.5; // 削脸 0-1
  eye: 0.5; // 大眼 0-1
  chin: 0.5; // 下巴 0-1

  effect: [{
    id: '7A62218064EF7959',
```

```
    intensity: 0.7,
  },{
    id: 'D7C6418064B90F4C',
    intensity: 0.7
  }]
}
try {
  await trtc.updatePlugin('Beauty', options);
} catch (error) {
  console.error('Beauty update failed', error);
}
```



```
// 停止美颜
try {
  await trtc.stopPlugin('Beauty');
} catch (error) {
  console.error('Beauty stop failed', error);
}
```

API 说明

trtc.startPlugin('Beauty', options)

用于开启美颜。

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
whiten	`number`	选填	美白，取值范围 [0,1]
dermabrasion	`number`	选填	磨皮，取值范围 [0,1]
lift	`number`	选填	窄脸，取值范围 [0,1]
shave	`number`	选填	削脸，取值范围 [0,1]
eye	`number`	选填	大眼，取值范围 [0,1]
chin	`number`	选填	下巴，取值范围 [0,1]
effect	`Array<EffectListOptions>`	选填	特效列表
onError	`(error) => {}`	选填	运行过程中发生错误的回调 error.code=10000003 渲染耗时长 error.code=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文末常见问题

EffectListOptions:

Name	Type	Attributes	Description
id	`number`	必填	特效 ID
intensity	`string`	选填	生效强度

特效 ID	名称
EDE5D18065451445	奶瓶面膜
7A62218064EF7959	毛毡发箍
B4D63180653948C3	刘海发带
D7C6418064B90F4C	可爱喵
1D6EB18069D76A62	波点女孩
CBE7618065D9D76F	爱心烟花
9B86618065596018	可爱猪猪
428C518065369ACF	双麻花
B30E218064F7B397	元气贴纸
AE3C81806521B49B	兔兔酱

trtc.updatePlugin('Beauty', options)

可修改美颜参数

options

Name	Type	Attributes	Description
whiten	`number`	选填	美白，取值范围 [0,1]
dermabrasion	`number`	选填	磨皮，取值范围 [0,1]
lift	`number`	选填	窄脸，取值范围 [0,1]
shave	`number`	选填	削脸，取值范围 [0,1]
eye	`number`	选填	大眼，取值范围 [0,1]
chin	`number`	选填	下巴，取值范围 [0,1]

effect	`Array<EffectListOptions>`	选填	特效列表

EffectListOptions:

Name	Type	Attributes	Description
id	`number`	必填	特效 ID
intensity	`string`	选填	生效强度

特效 ID	名称
EDE5D18065451445	奶瓶面膜
7A62218064EF7959	毛毡发箍
B4D63180653948C3	刘海发带
D7C6418064B90F4C	可爱喵
1D6EB18069D76A62	波点女孩
CBE7618065D9D76F	爱心烟花
9B86618065596018	可爱猪猪
428C518065369ACF	双麻花
B30E218064F7B397	元气贴纸
AE3C81806521B49B	兔兔酱

trtc.stopPlugin('Beauty')

关闭美颜。

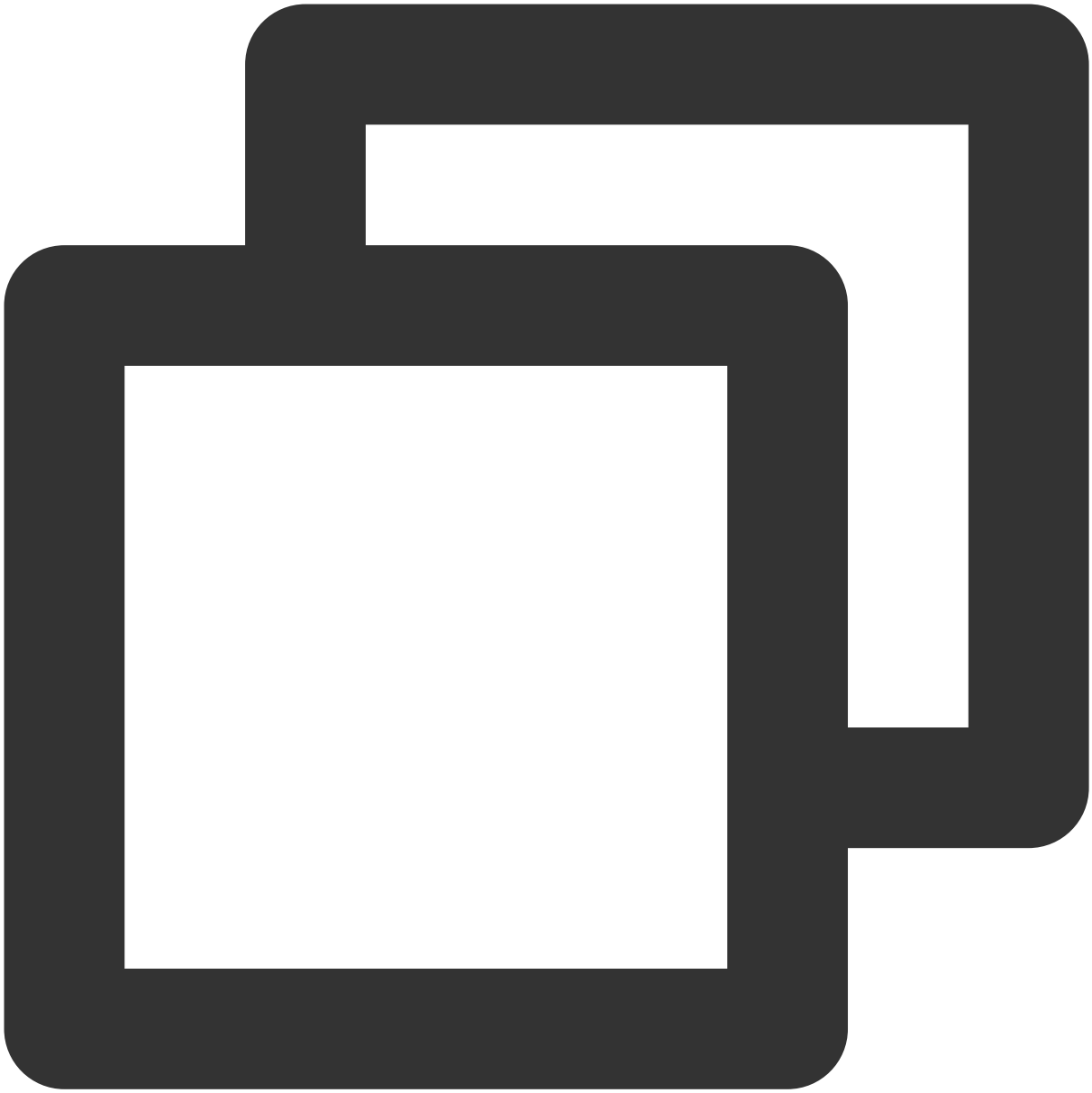
常见问题

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。



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

```
}

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

测试硬件设备
Android&iOS&Windows&Mac

最近更新时间：2022-06-22 15:14:12

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。

支持此功能的平台

iOS	Android	Mac OS	Windows	Electron	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 Objective-C
- Windows平台（C++） C++
- Windows平台（C#） c#

```
// 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];
}
```

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 Objective-C
- Windows平台（C++） C++
- Windows平台（C#） c#

```
// 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";
}
}
```

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 Objective-C
- Windows平台（C++） C++
- Windows平台（C#） C#

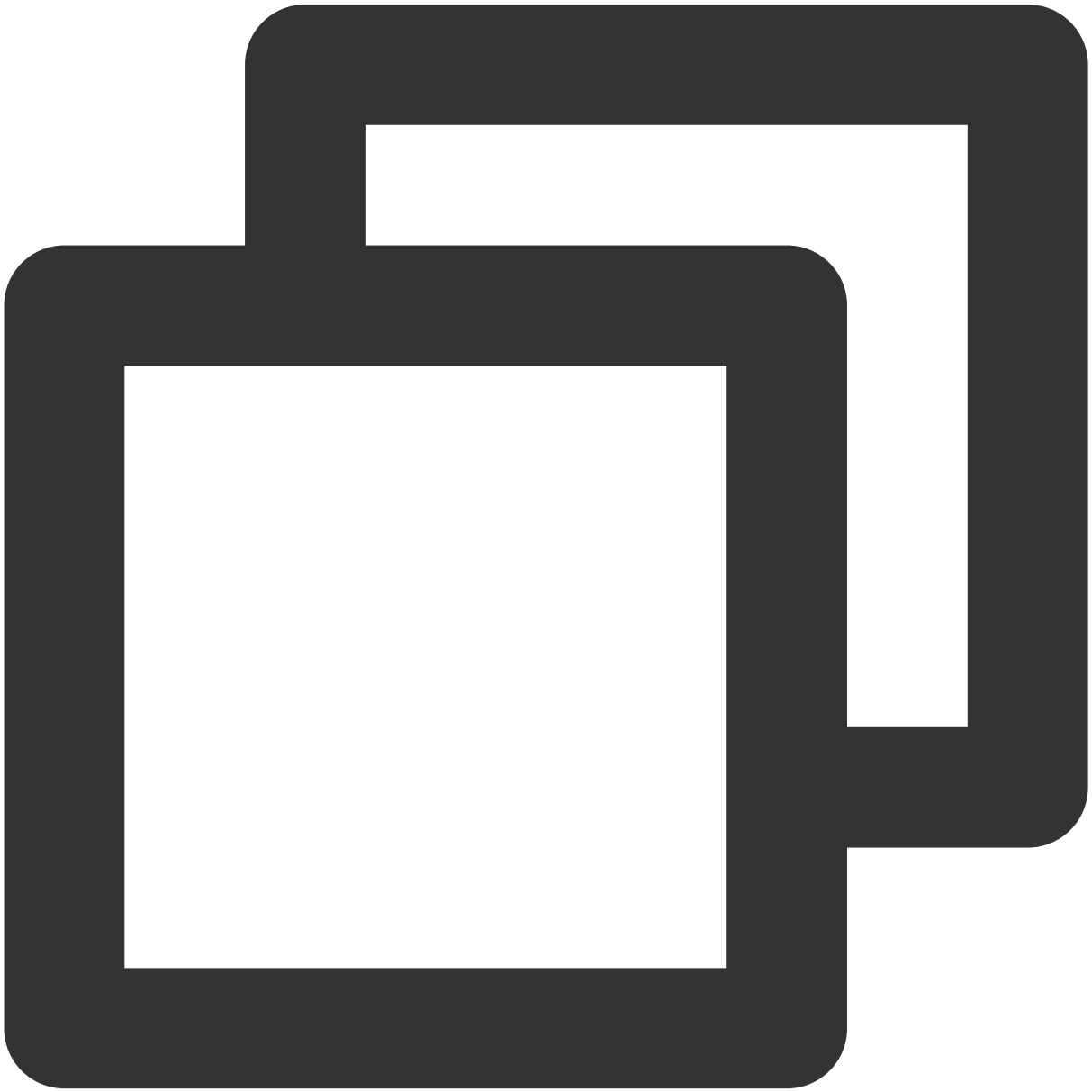
```
// 扬声器测试示例代码
// 以作为 NSButton 的点击事件为例，在 xib 中设置 Button 在 On 和 Off 下的标题分别为"结束测试"和"开始测试"
- (IBAction)speakerTest:(NSButton *)btn {
NSString *path = [[NSBundle mainBundle] pathForResource:@"test-32000-mono" ofType:@"mp3"];
if (btn.state == NSControlStateValueOn) {
// 单击"开始测试"
__weak __typeof(self) wself = 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;
}
```

Web

最近更新时间：2023-11-16 15:09:26

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) {}
})
```

`TRTC.isSupported()` 返回的检测结果为 false 时，可能是以下原因：

1. 网页使用了 http 协议。浏览器不允许 http 协议的网站采集摄像头和麦克风，您需要使用 https 协议部署您的网页。
2. 当前浏览器不支持 WebRTC 相关能力，您需要引导用户使用推荐的浏览器，参考：Supported Browsers。
3. 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。

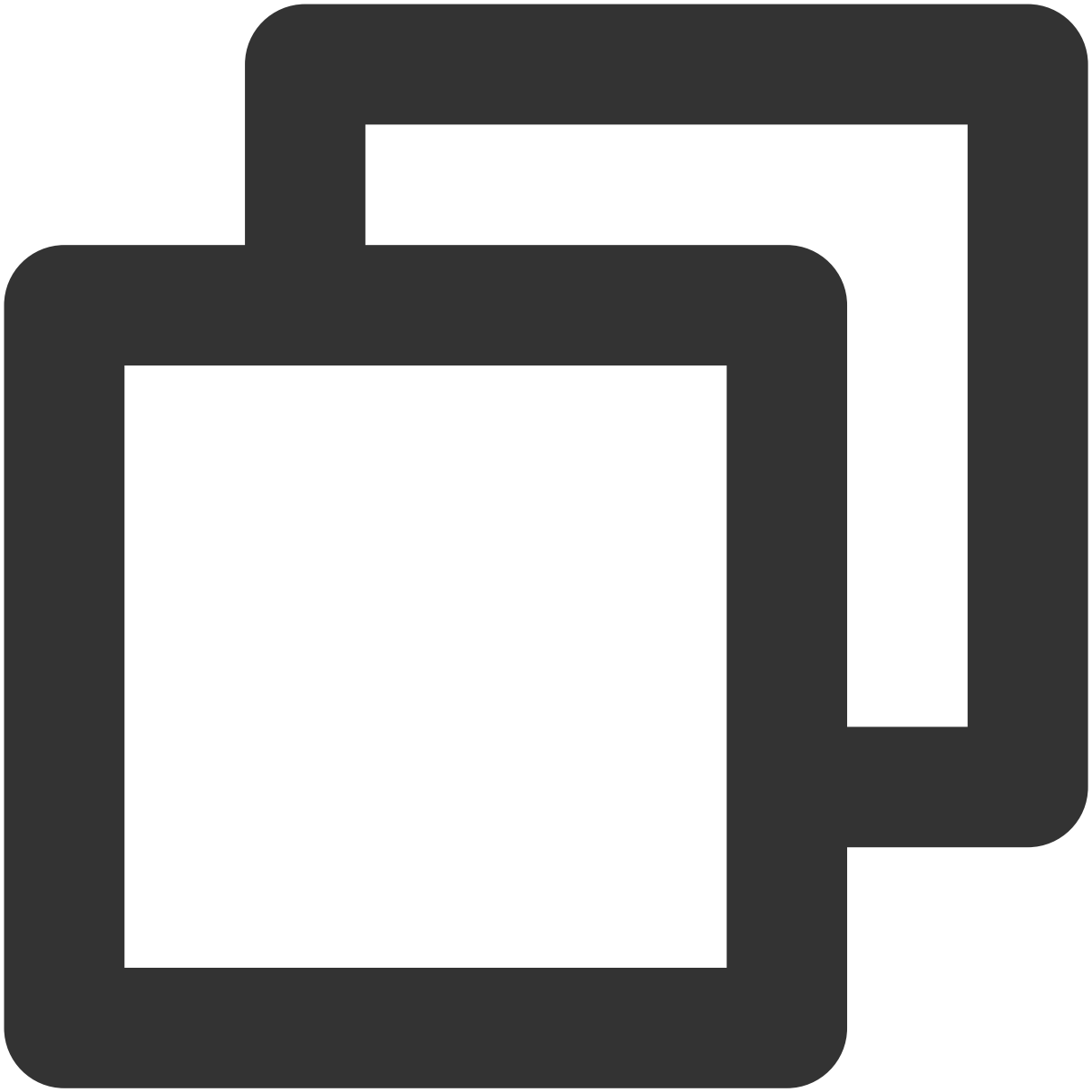
音视频设备测试

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

rtc-detect 库

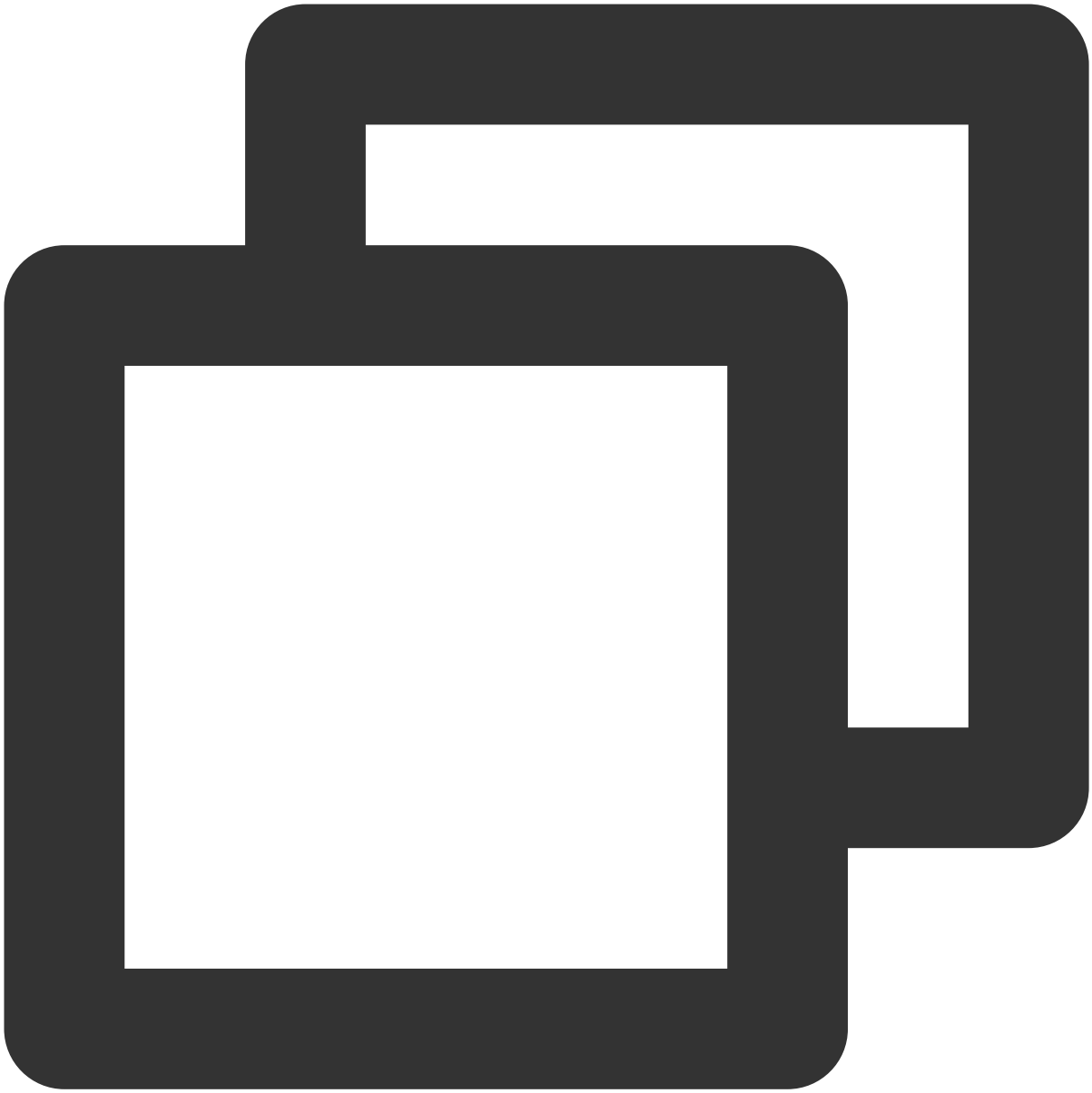
您可以使用 rtc-detect 用来检测当前环境对 TRTC SDK 的支持度，以及当前环境的详细信息。

安装



```
npm install rtc-detect
```

使用方法

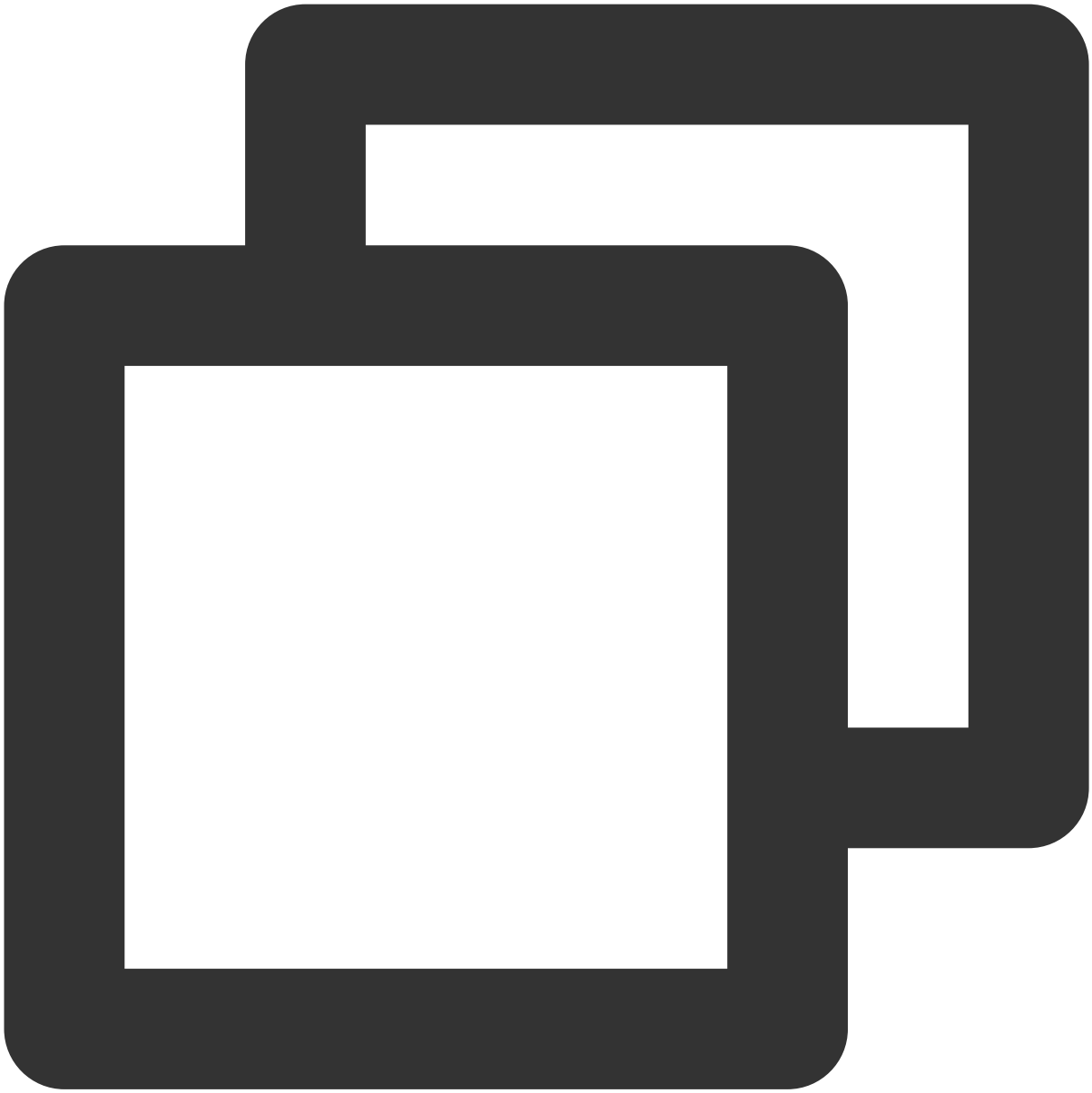


```
import RTCDetect from 'rtc-detect';
// 初始化监测模块
const detect = new RTCDetect();
// 获得当前环境监测结果
const result = await detect.getReportAsync();
// result 包含了当前环境系统的信息，API 支持度，编解码支持度，设备相关的信息
console.log('result is: ' + result);
```

API

(async) isTRTCSupported()

判断当前环境是否支持 TRTC。

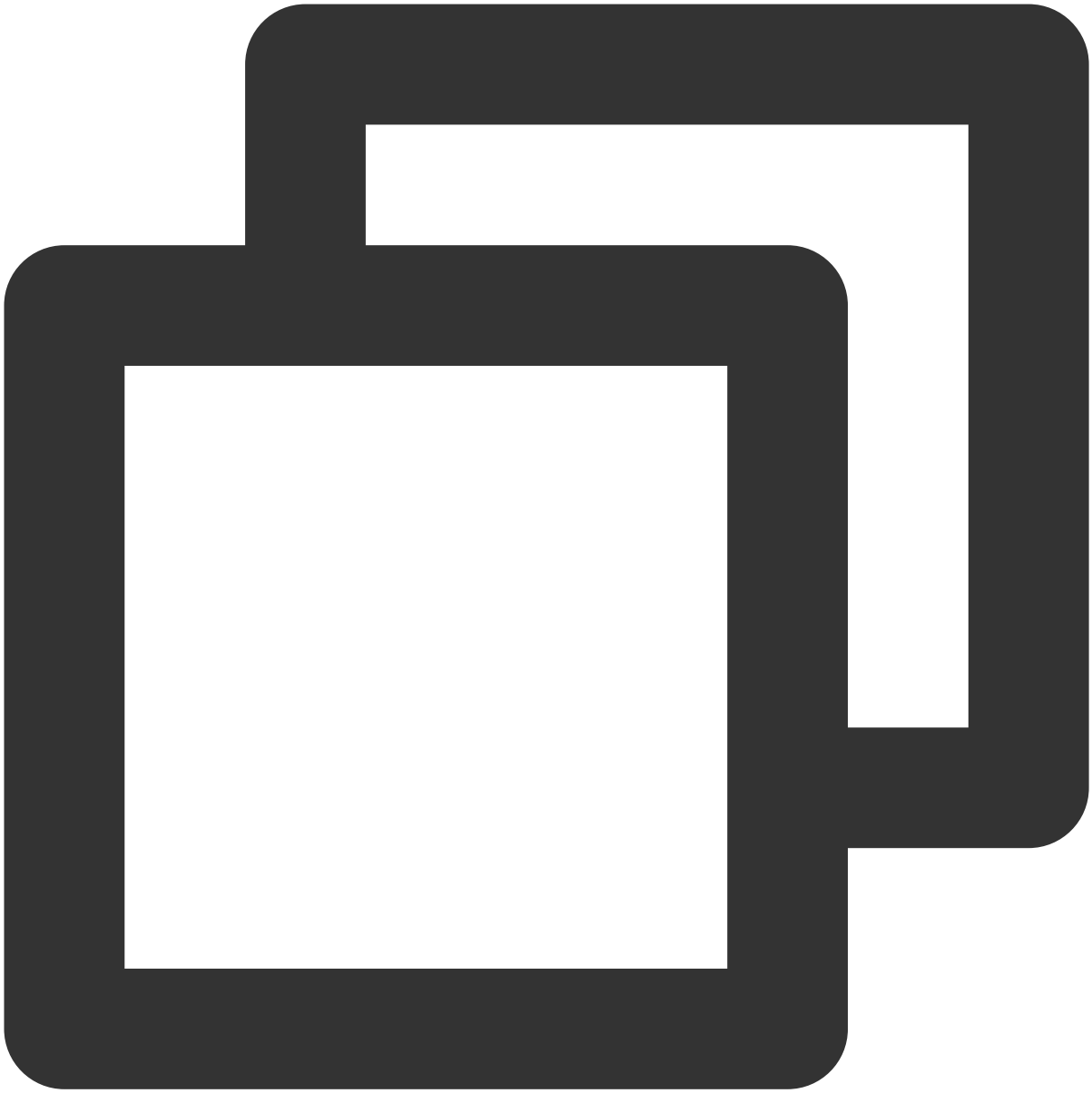


```
const detect = new RTCDetect();
const data = await detect.isTRTCSupported();

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

getSystem()

获取当前系统环境参数。

Item	Type	Description
UA	string	浏览器的 ua
OS	string	当前设备的系统型号
browser	object	当前浏览器信息{ name, version }
displayResolution	object	当前分辨率 { width, height }
getHardwareConcurrency	number	当前设备 CPU 核心数



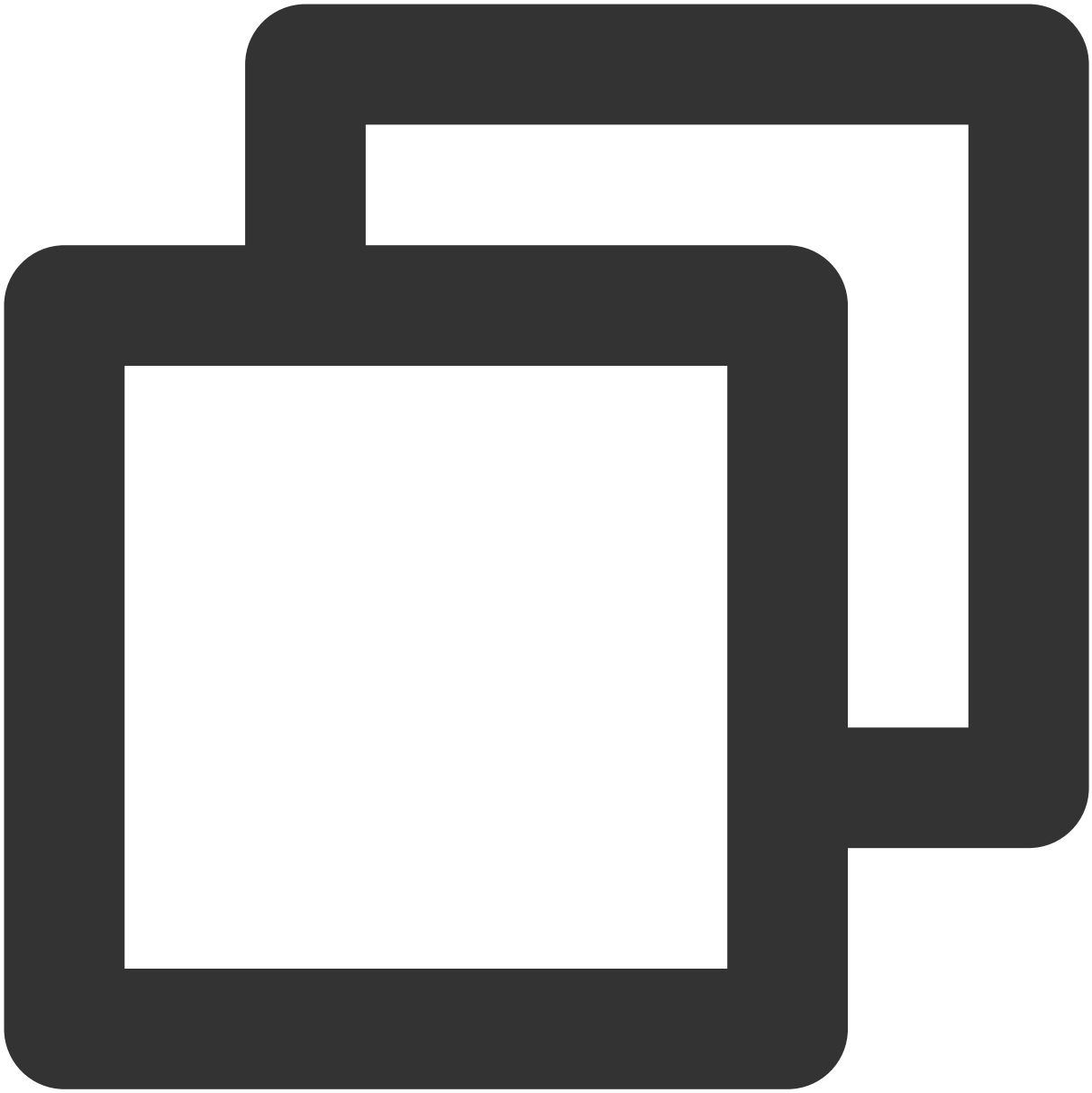
```
const detect = new RTCDetect();
const result = detect.getSystem();
```

getAPISupported()

获取当前环境 API 支持度。

Item	Type	Description
isUserMediaSupported	boolean	是否支持获取用户媒体数据流

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



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

(async) getDevicesAsync()

获取当前环境可用的设备。

Item	Type	Description
hasCameraPermission	boolean	是否支持获取用户摄像头数据

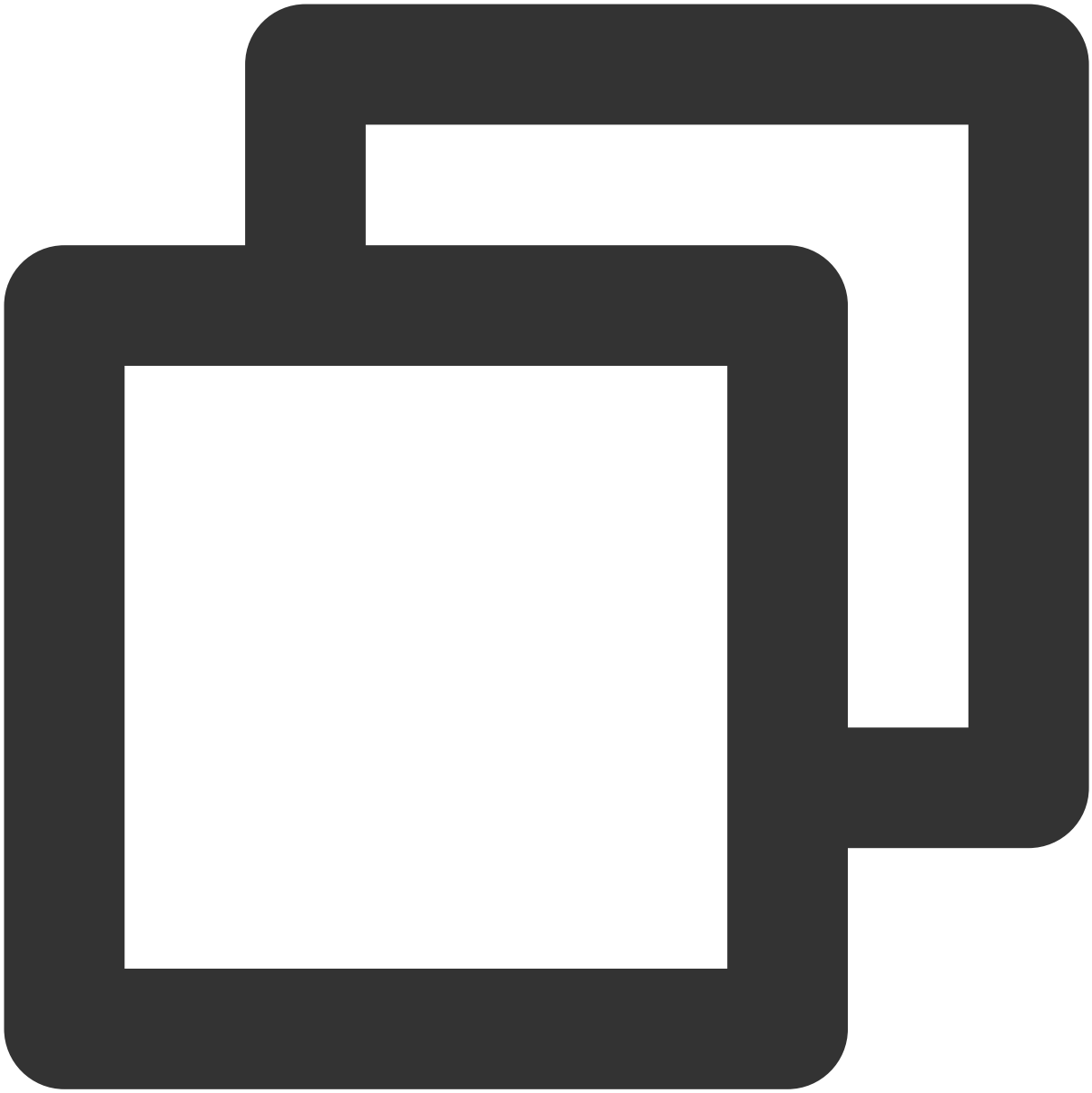
hasMicrophonePermission	boolean	是否支持获取用户麦克风数据
cameras	array<CameraItem>	用户的摄像头设备列表，包含支持视频流的分辨率信息，最大宽高以及最大帧率（最大帧率有部分浏览器不支持）
microphones	array<DeviceItem>	用户的麦克风设备列表
speakers	array<DeviceItem>	用户的扬声器设备列表

CameraItem

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

DeviceItem

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



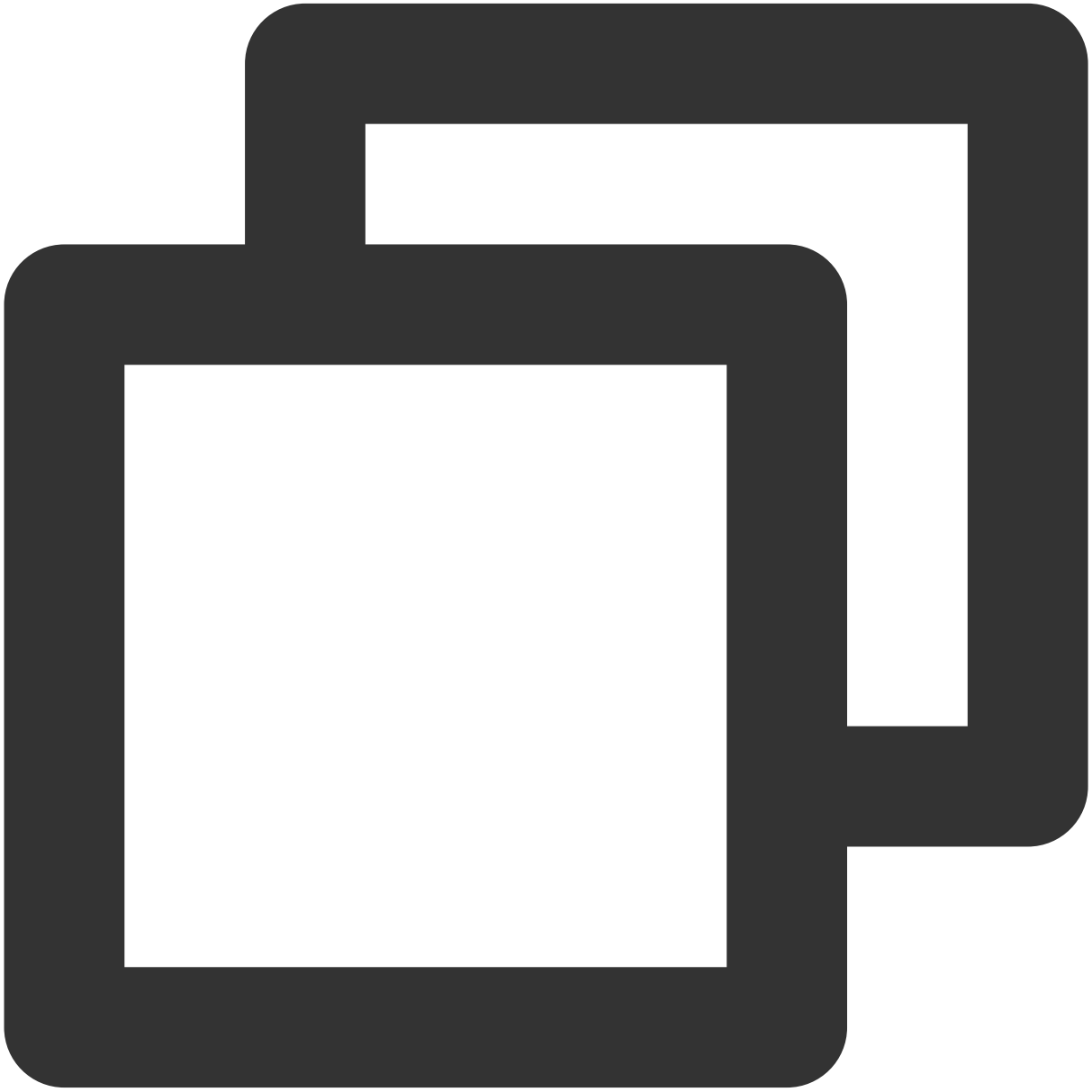
```
const detect = new RTCDetect();
const result = await detect.getDevicesAsync();
```

(async) getCodecAsync()

获取当前环境参数对编码的支持度。

Item	Type	Description
isH264EncodeSupported	boolean	是否支持 h264 编码

isH264DecodeSupported	boolean	是否支持 h264 解码
isVp8EncodeSupported	boolean	是否支持 vp8 编码
isVp8DecodeSupported	boolean	是否支持 vp8 解码
支持编码即支持发布音视频，支持解码即支持拉取音视频播放		

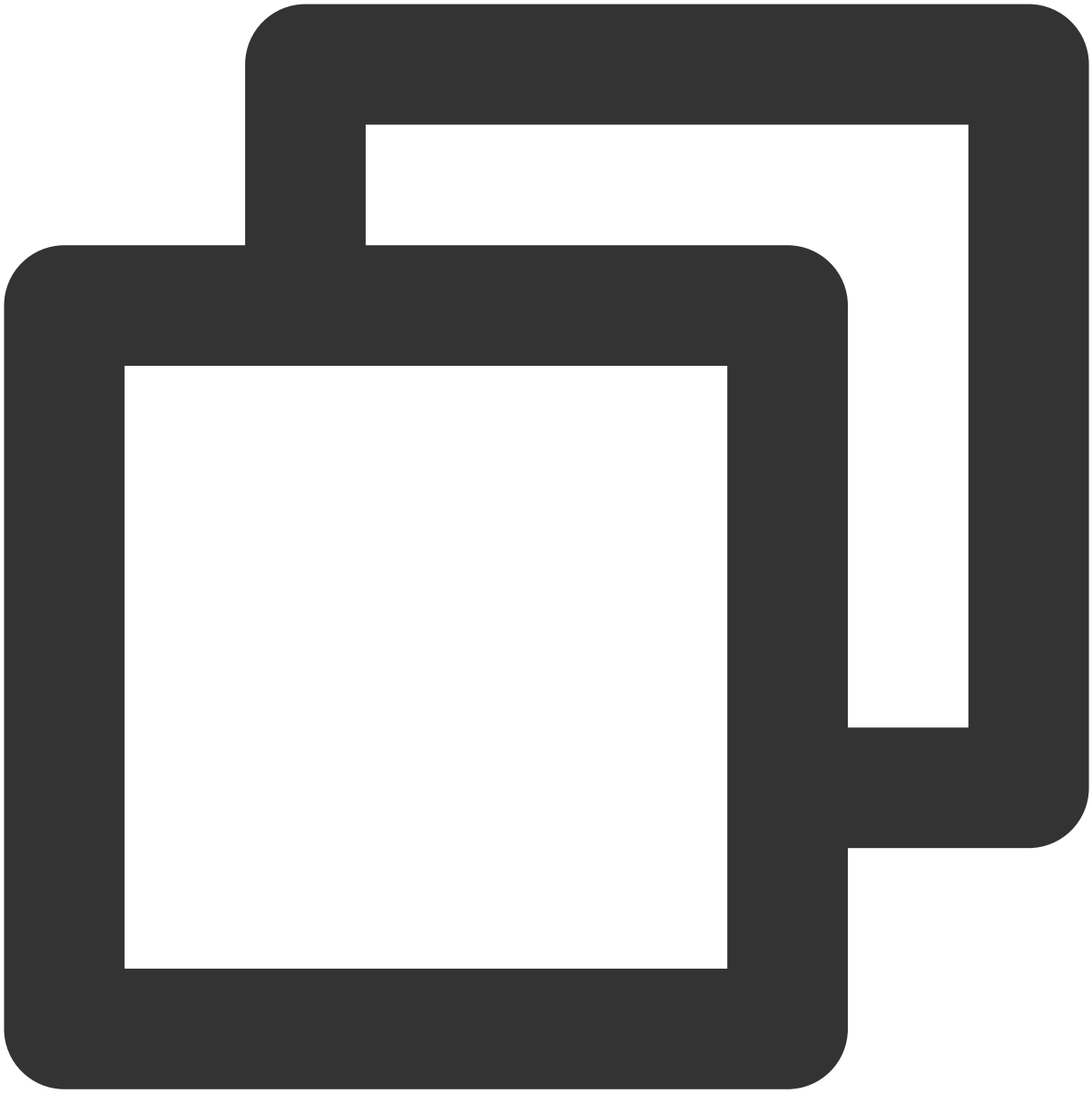


```
const detect = new RTCDetect();
const result = await detect.getCodecAsync();
```

(async) getReportAsync()

获取当前环境监测报告。

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



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

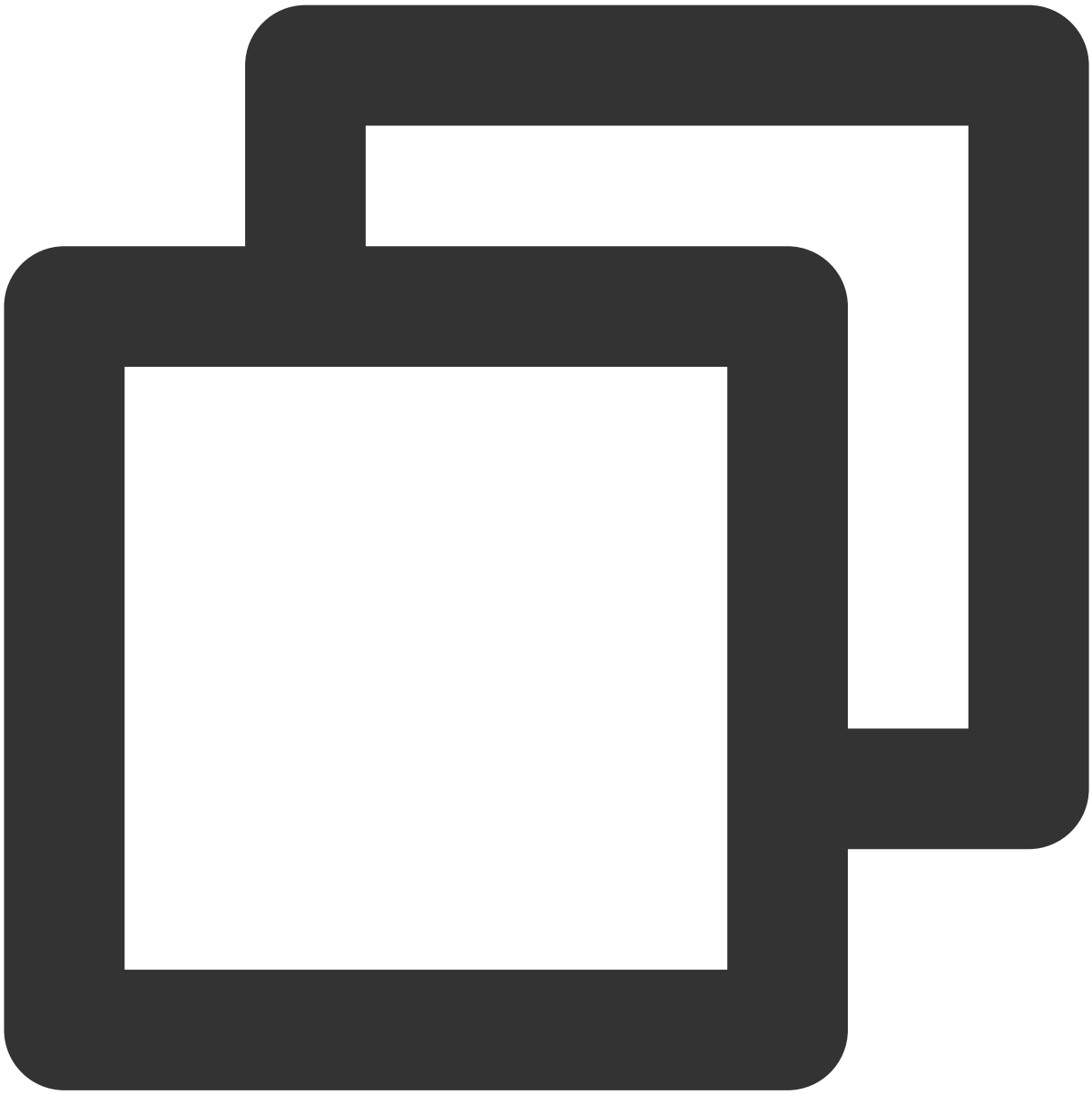
(async) isHardWareAccelerationEnabled()

检测 Chrome 浏览器是否开启硬件加速。

注意：

该接口的实现依赖于 WebRTC 原生接口，建议在 isTRTCSupported 检测支持后，再调用该接口进行检测。检测最长耗时 30s。经实测：

1. 开启硬件加速的情况下，该接口在 Windows 耗时 2s 左右， Mac 需耗时 10s 左右。
2. 关闭硬件加速的情况下，该接口在 Windows 和 Mac 耗时均为 30s。



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

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

```
}
```

设备检测的 React 组件

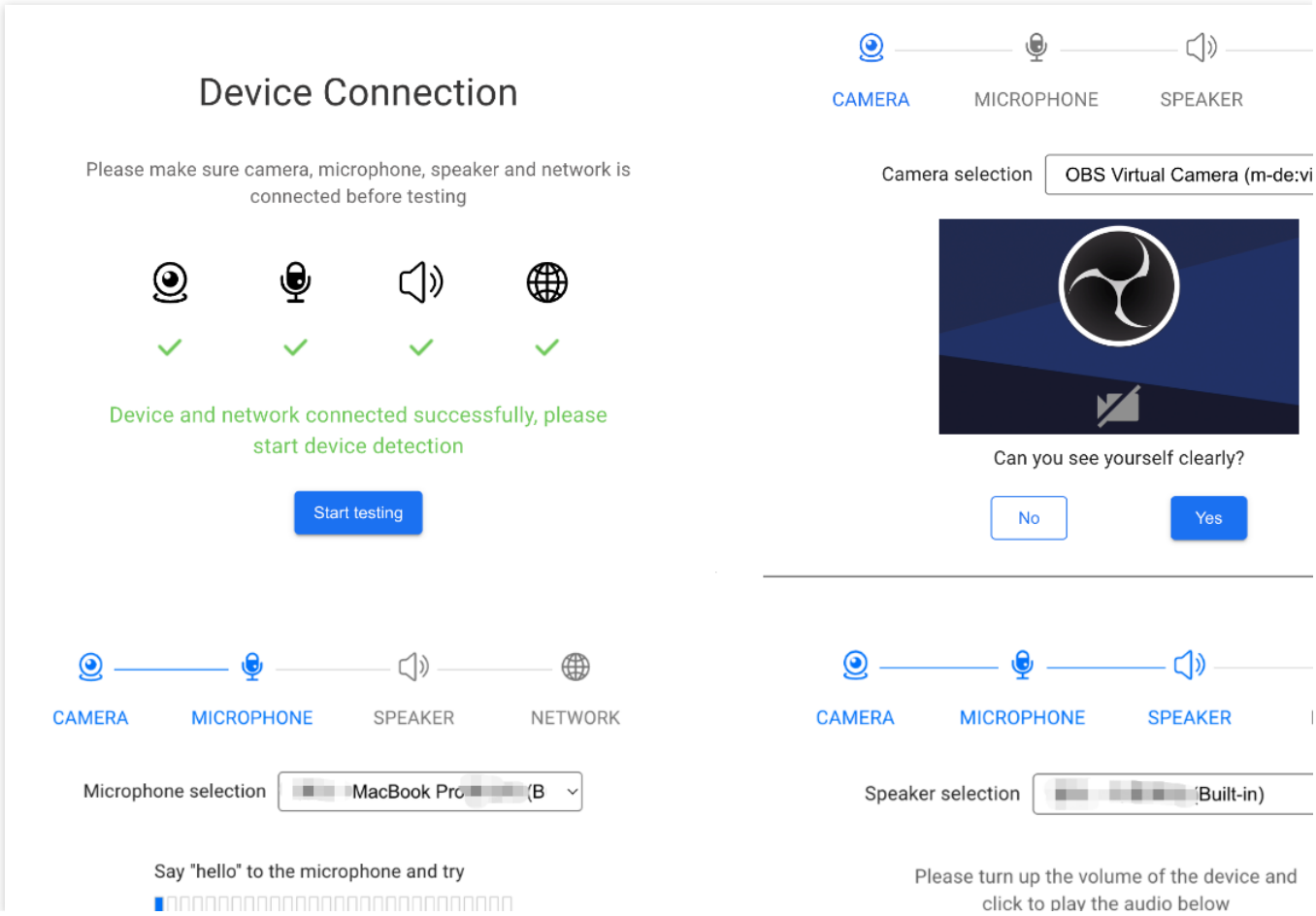
设备检测 UI 组件特点

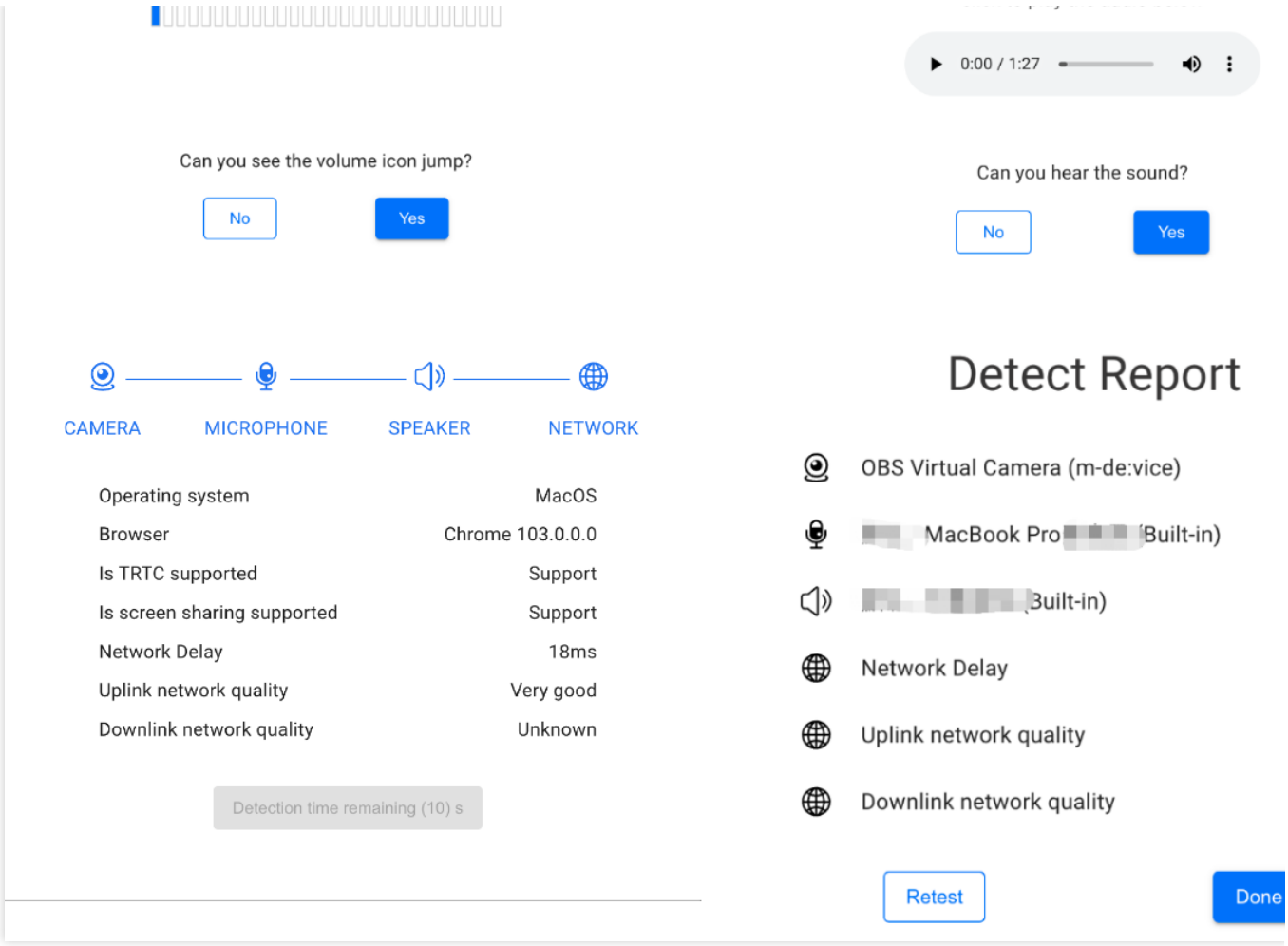
1. 处理了设备连接及设备检测逻辑
2. 处理了网络检测的逻辑
3. 网络检测 tab 页可选
4. 支持中、英文两种语言

设备检测 UI 组件相关链接

组件npm包使用说明请参考：rtc-device-detector-react
组件源码调试请参考：github/rtc-device-detector
组件引用示例请参考：WebRTC API Example

设备检测 UI 组件界面



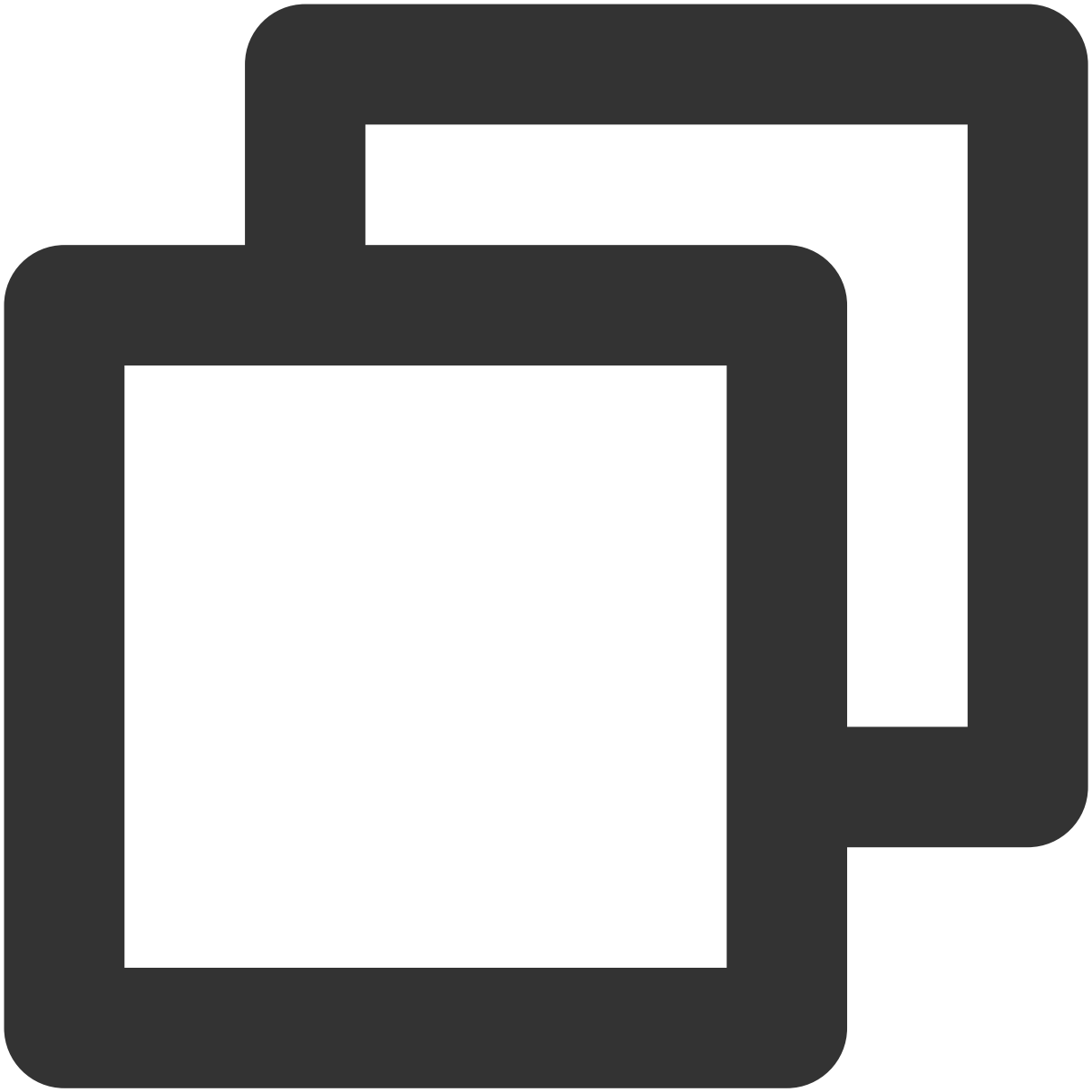


设备及网络检测逻辑

1）设备连接

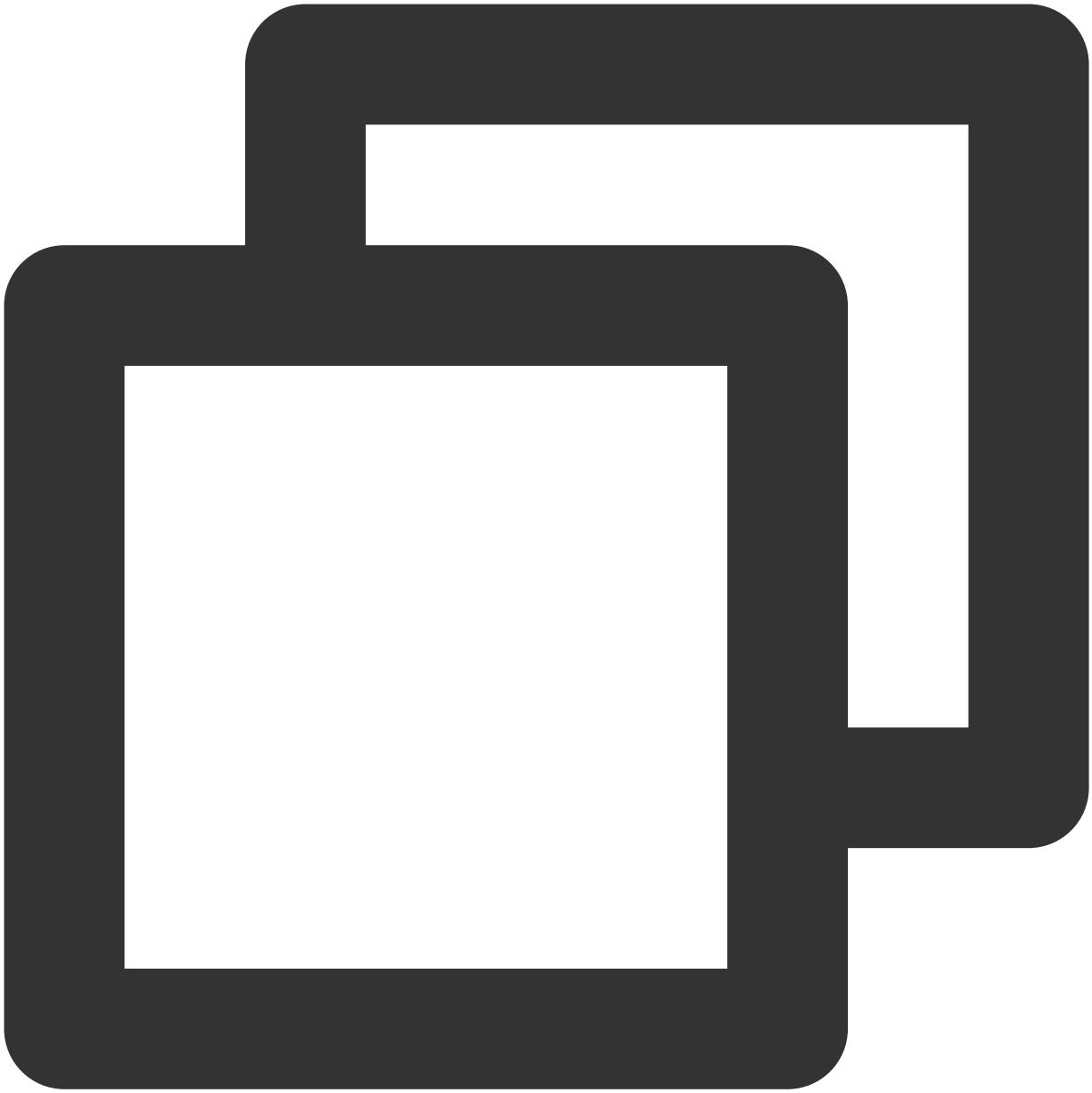
设备连接的目的是检测用户使用的机器是否有摄像头，麦克风，扬声器设备，是否在联网状态。如果有摄像头，麦克风设备，尝试获取音视频流并引导用户授予摄像头，麦克风的访问权限。

判断设备是否有摄像头，麦克风，扬声器设备



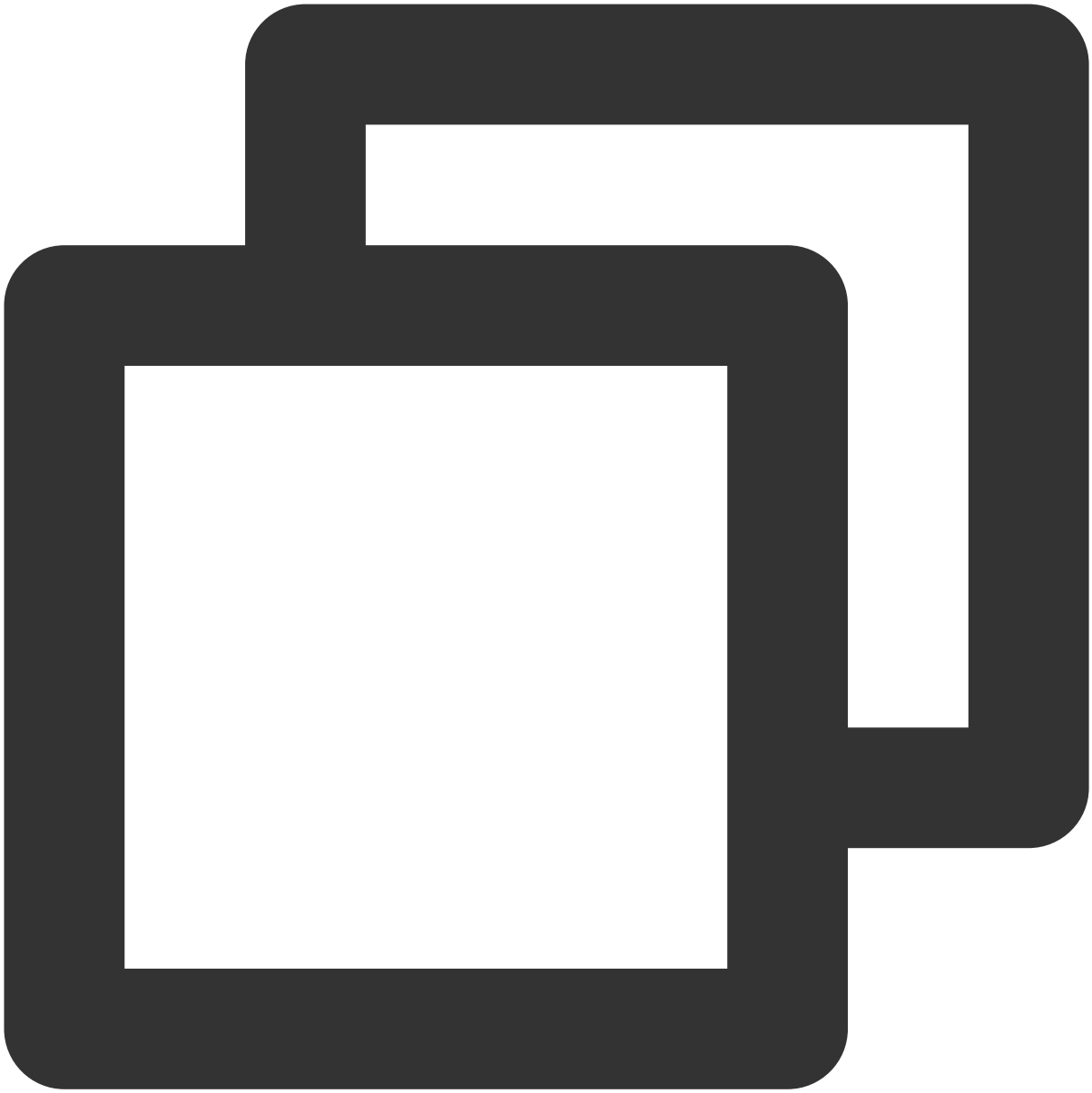
```
import TRTC from 'trtc-sdk-v5';

const cameraList = await TRTC.getCameraList();
const micList = await TRTC.getMicrophoneList();
const speakerList = await TRTC.getSpeakerList();
const hasCameraDevice = cameraList.length > 0;
const hasMicrophoneDevice = micList.length > 0;
const hasSpeakerDevice = speakerList.length > 0;
```

获取摄像头，麦克风的访问权限



```
await trtc.startLocalVideo({ publish: false });
await trtc.startLocalAudio({ publish: false });
```

判断设备是否联网



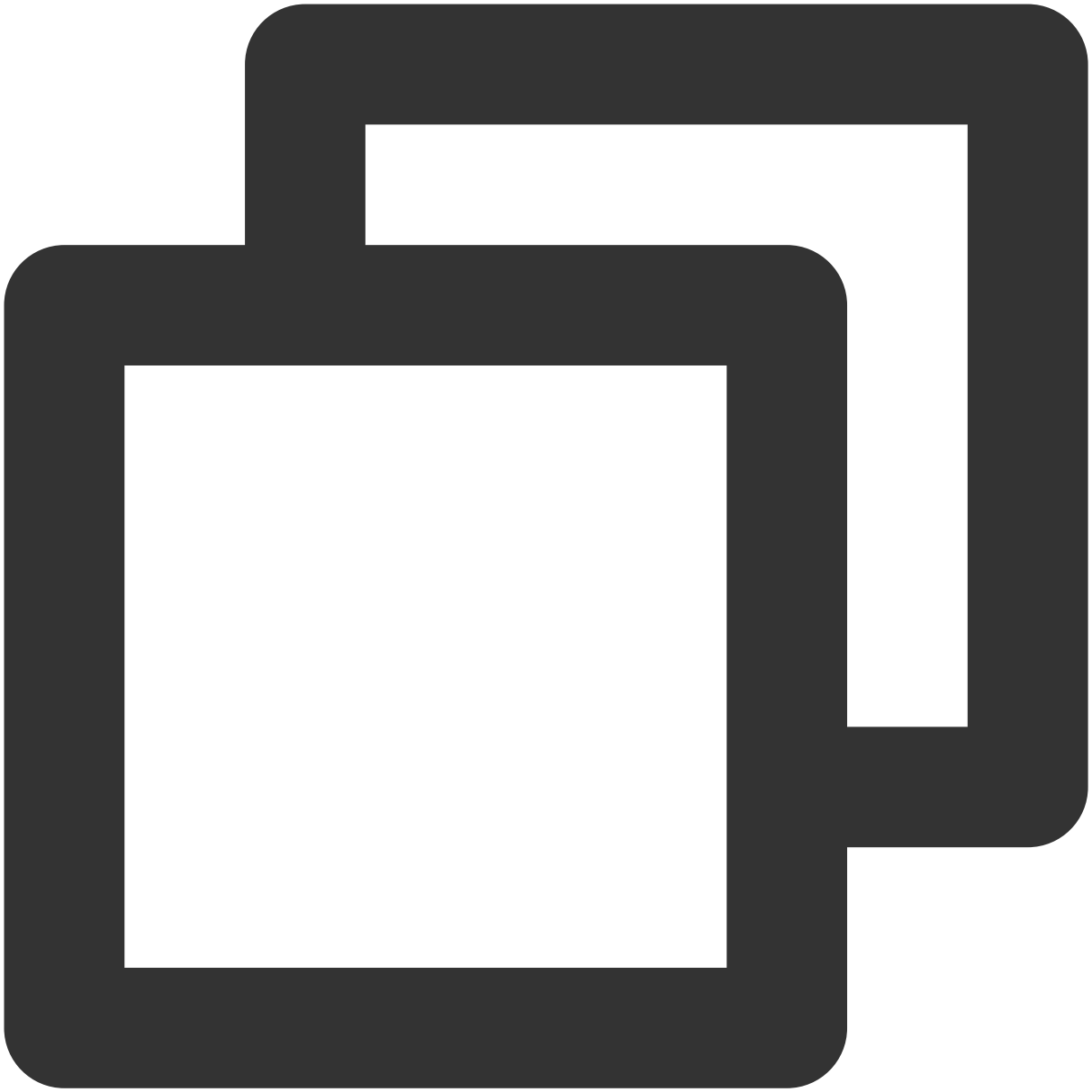
```
export function isOnline() {
  const url = 'https://web.sdk.qcloud.com/trtc/webrtc/assets/trtc-logo.png';
  return new Promise((resolve) => {
    try {
      const xhr = new XMLHttpRequest();
      xhr.onload = function () {
        resolve(true);
      };
      xhr.onerror = function () {
     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

```
        xhr.open('GET', url, true);
        xhr.send();
      } catch (err) {
        // console.log(err);
      }
    });
}
const isOnline = await isOnline();
```

2） 摄像头检测

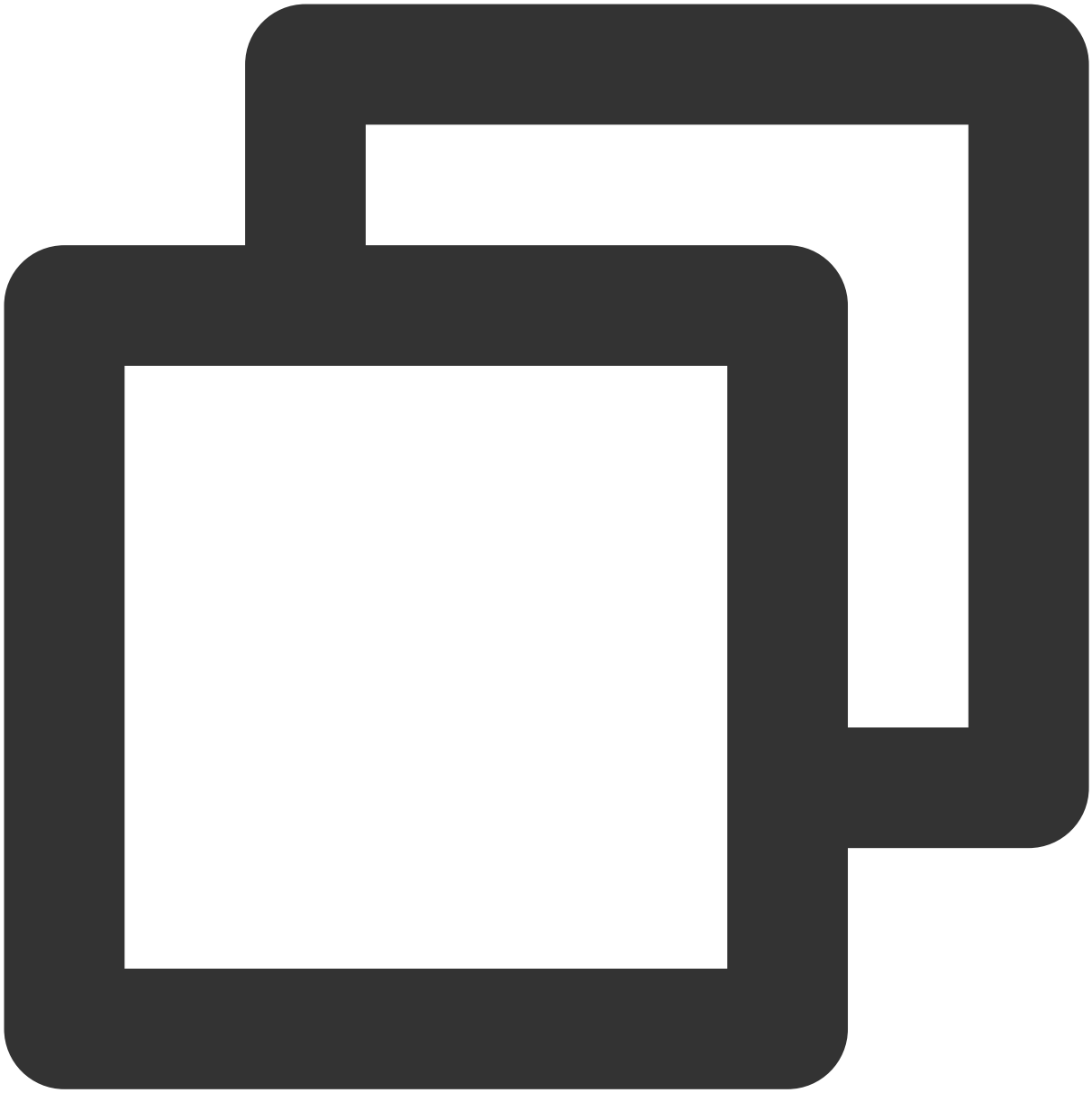
检测原理：打开摄像头，并在页面中渲染摄像头画面。

打开摄像头



```
trtc.startLocalVideo({ view: 'camera-video', publish: false });
```

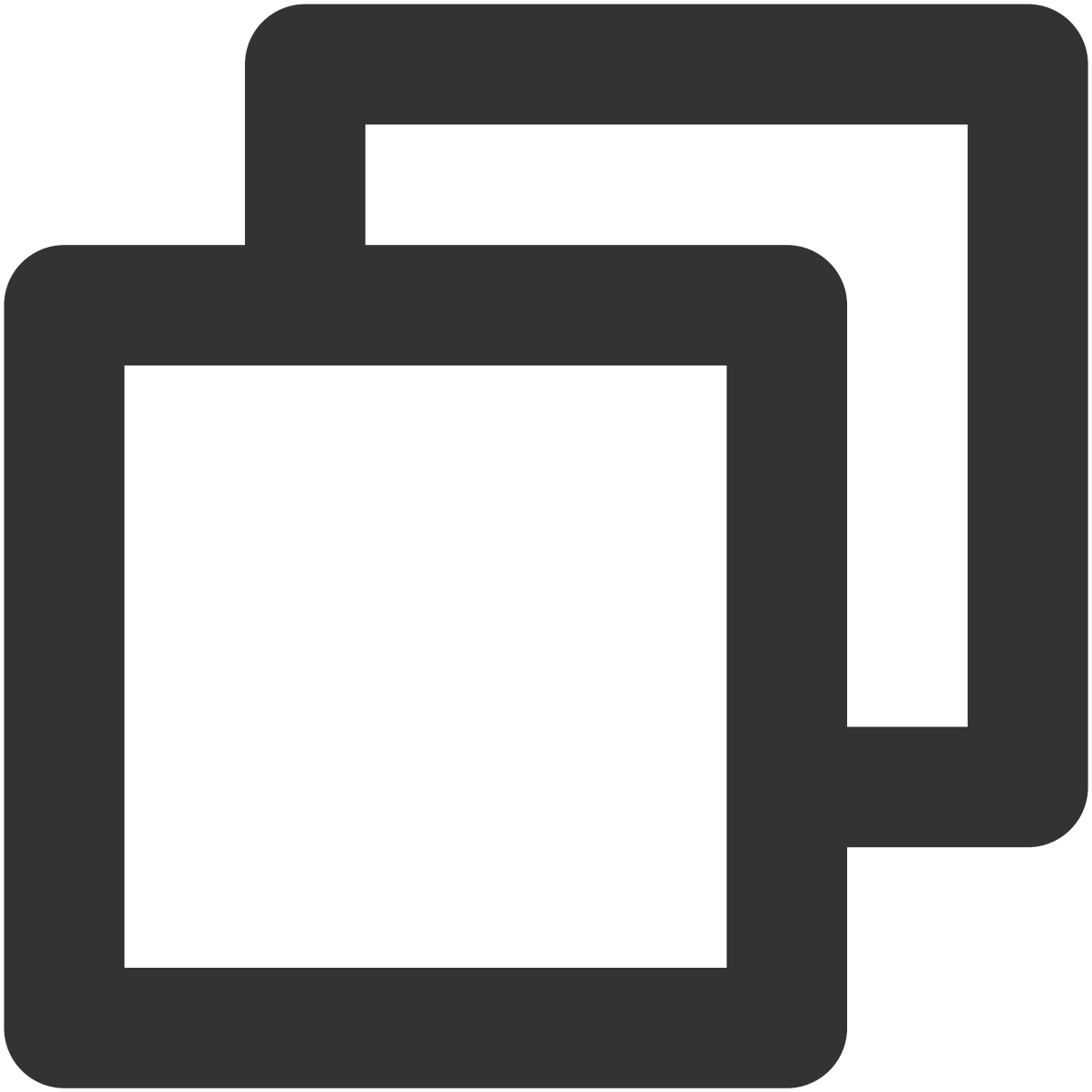
切换摄像头



```
trtc.updateLocalVideo({
  option: { cameraId }
});
```

设备插拔检测

检测完成后关闭摄像头

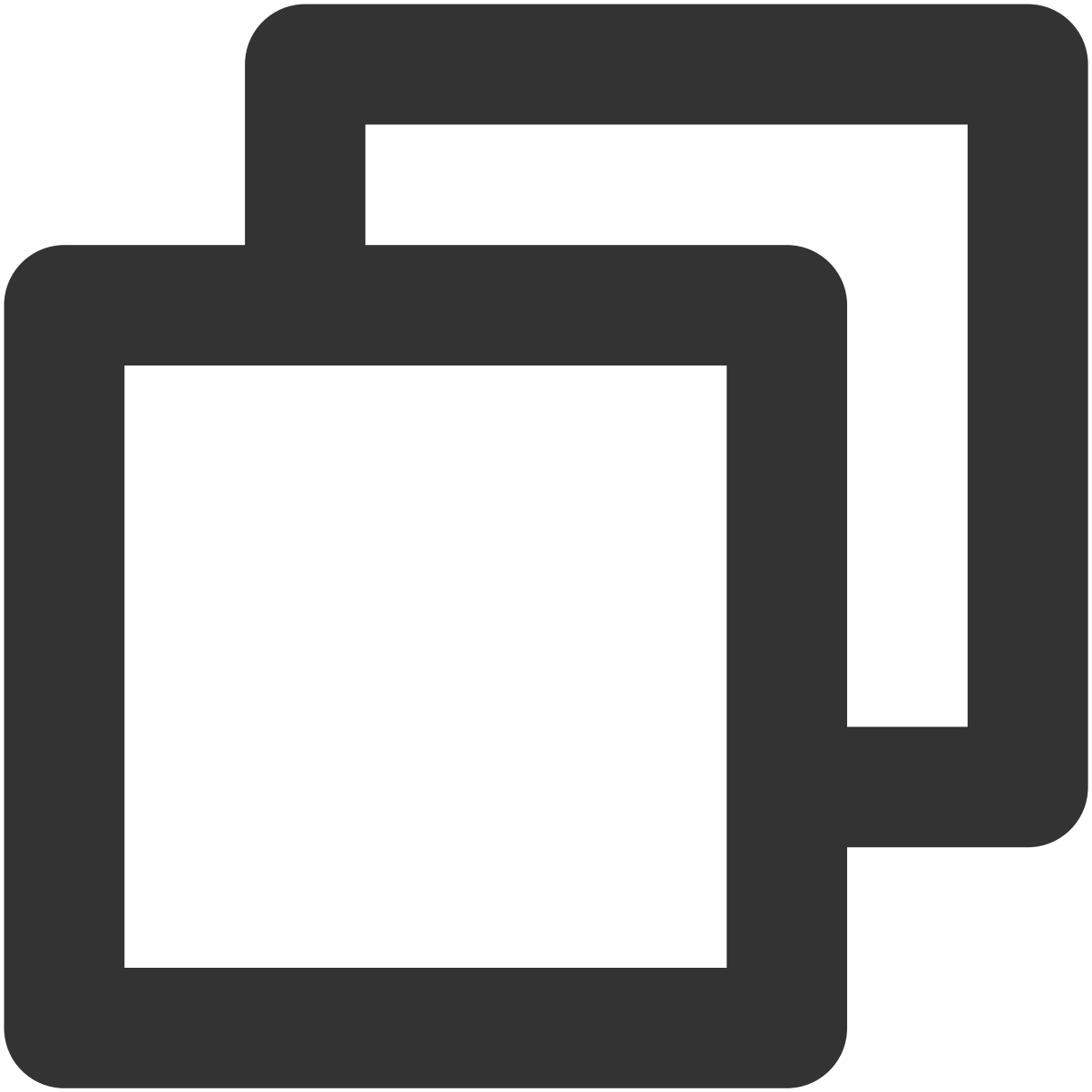


```
trtc.stopLocalVideo();
```

3）麦克风检测

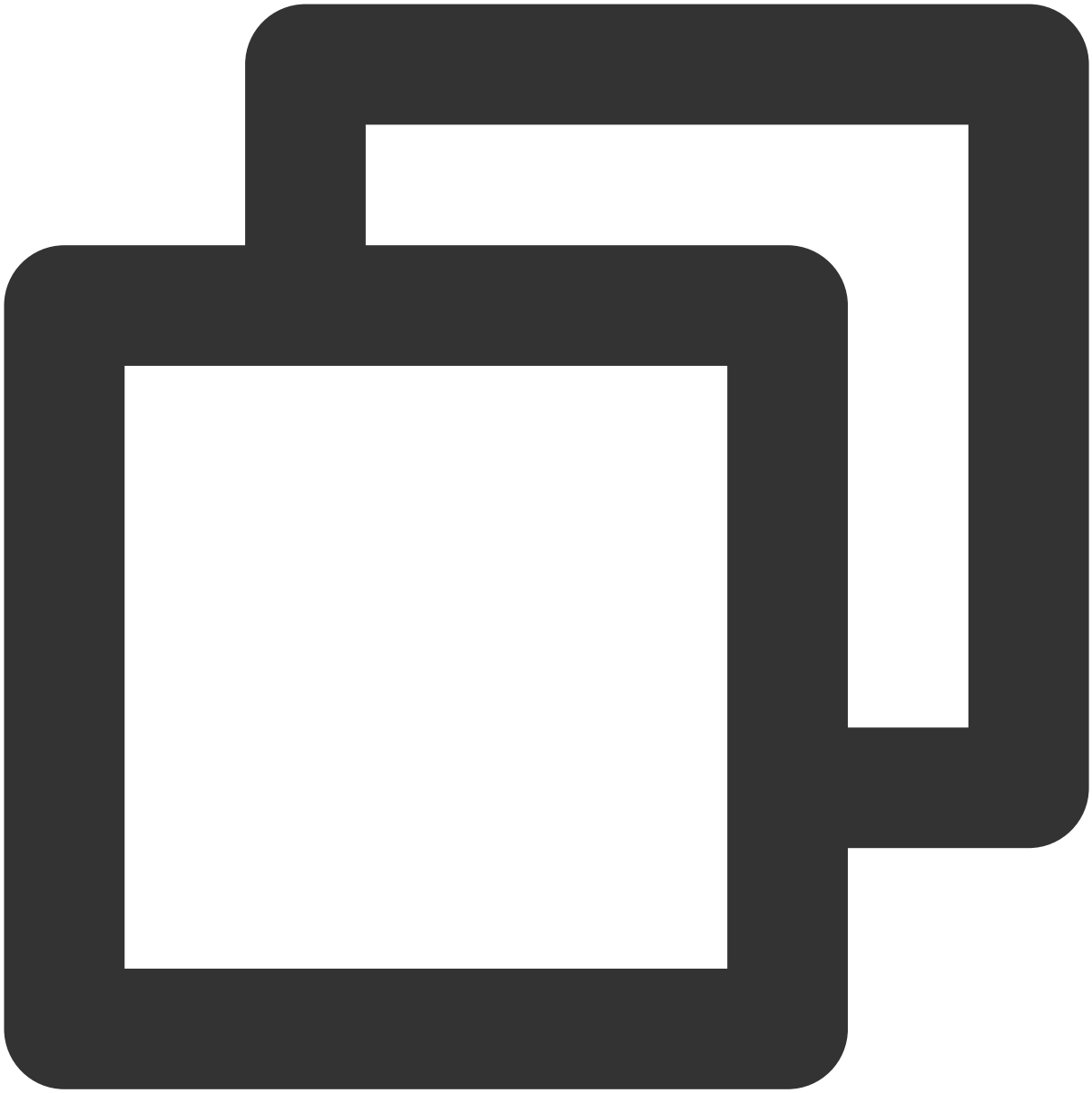
检测原理：打开麦克风，并获取麦克风音量。

打开麦克风



```
trtc.startLocalAudio({ publish: false });
```

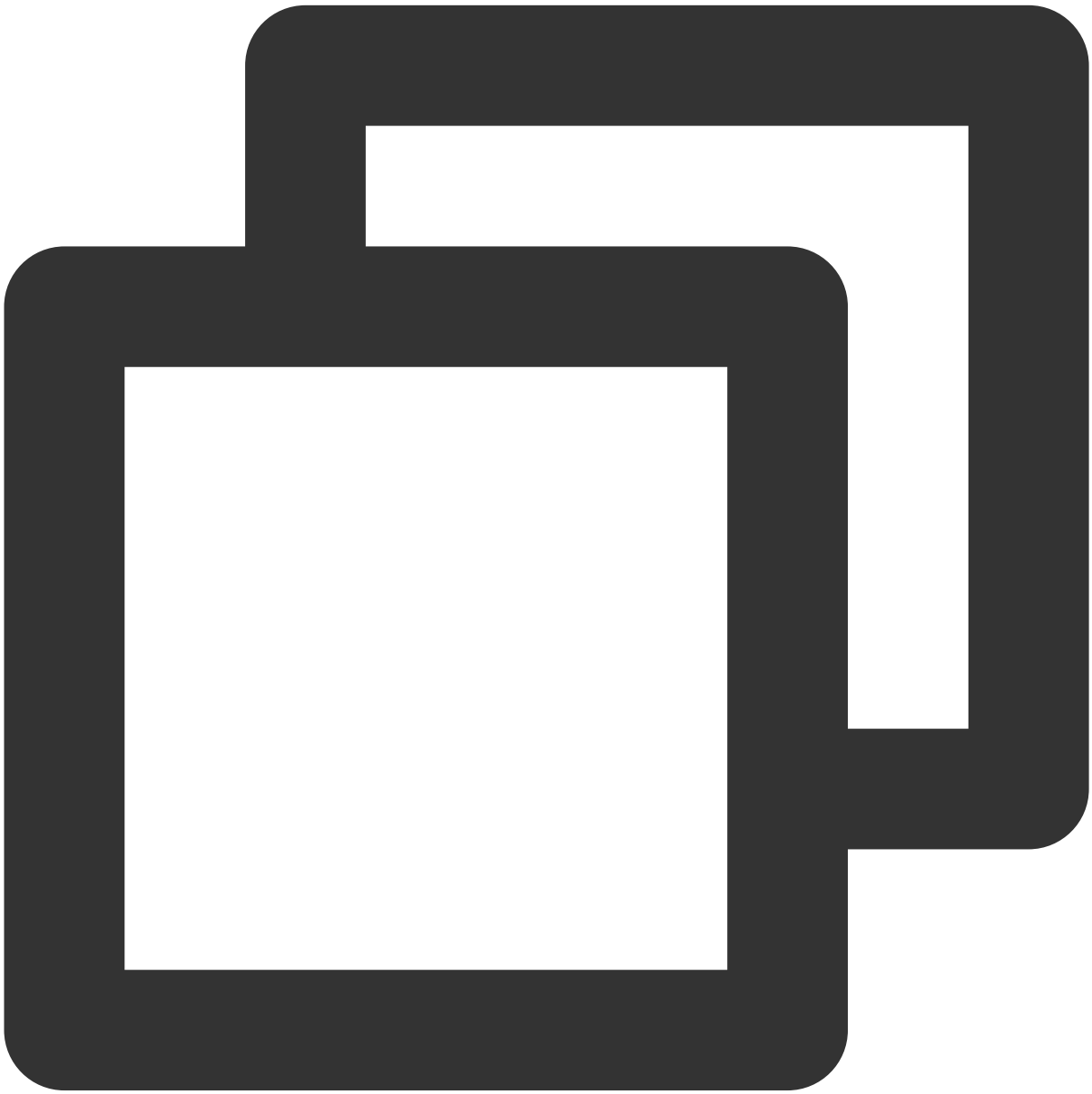
检测麦克风音量



```
trtc.on(TRTC.EVENT.AUDIO_VOLUME, event => {
    event.result.forEach(({ userId, volume }) => {
        const isMe = userId === ''; // 当 userId 为空串时，代表本地麦克风音量。
        if (isMe) {
            console.log(`my volume: ${volume}`);
        } else {
            console.log(`user: ${userId} volume: ${volume}`);
        }
    })
});
// 开启音量回调，并设置每 500ms 触发一次事件
```

```
trtc.enableAudioVolumeEvaluation(500);
// 出于性能的考虑，当页面切换到后台时，SDK 不会抛出音量回调事件。如需在页面切后台时接收音量回调事件
trtc.enableAudioVolumeEvaluation(500, true);
// 如需关闭音量回调，传入 interval 值小于等于0即可
trtc.enableAudioVolumeEvaluation(-1);
```

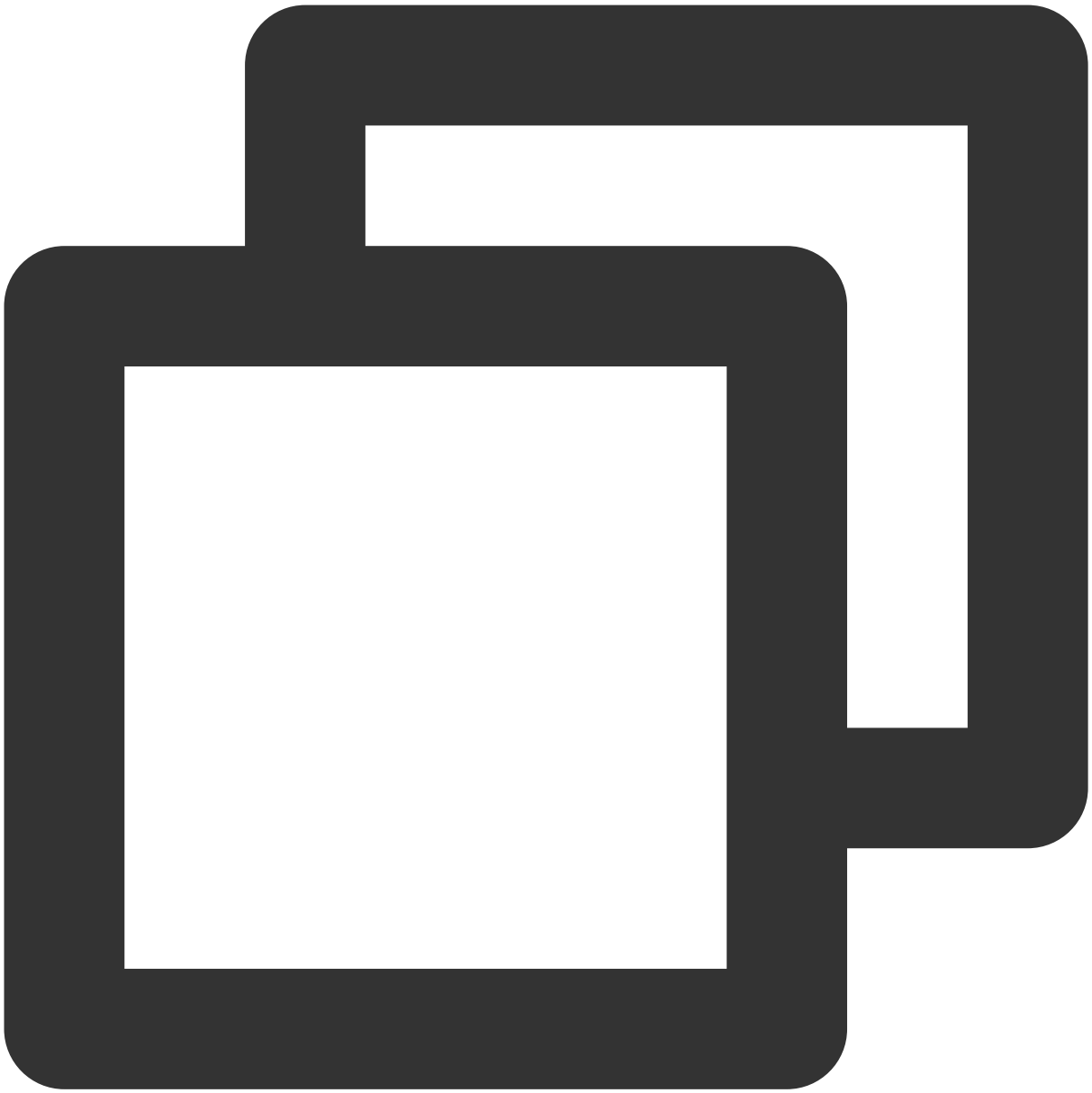
切换麦克风



```
trtc.updateLocalAudio({ option: { microphoneId }});
```

设备插拔检测

检测完成后释放麦克风占用

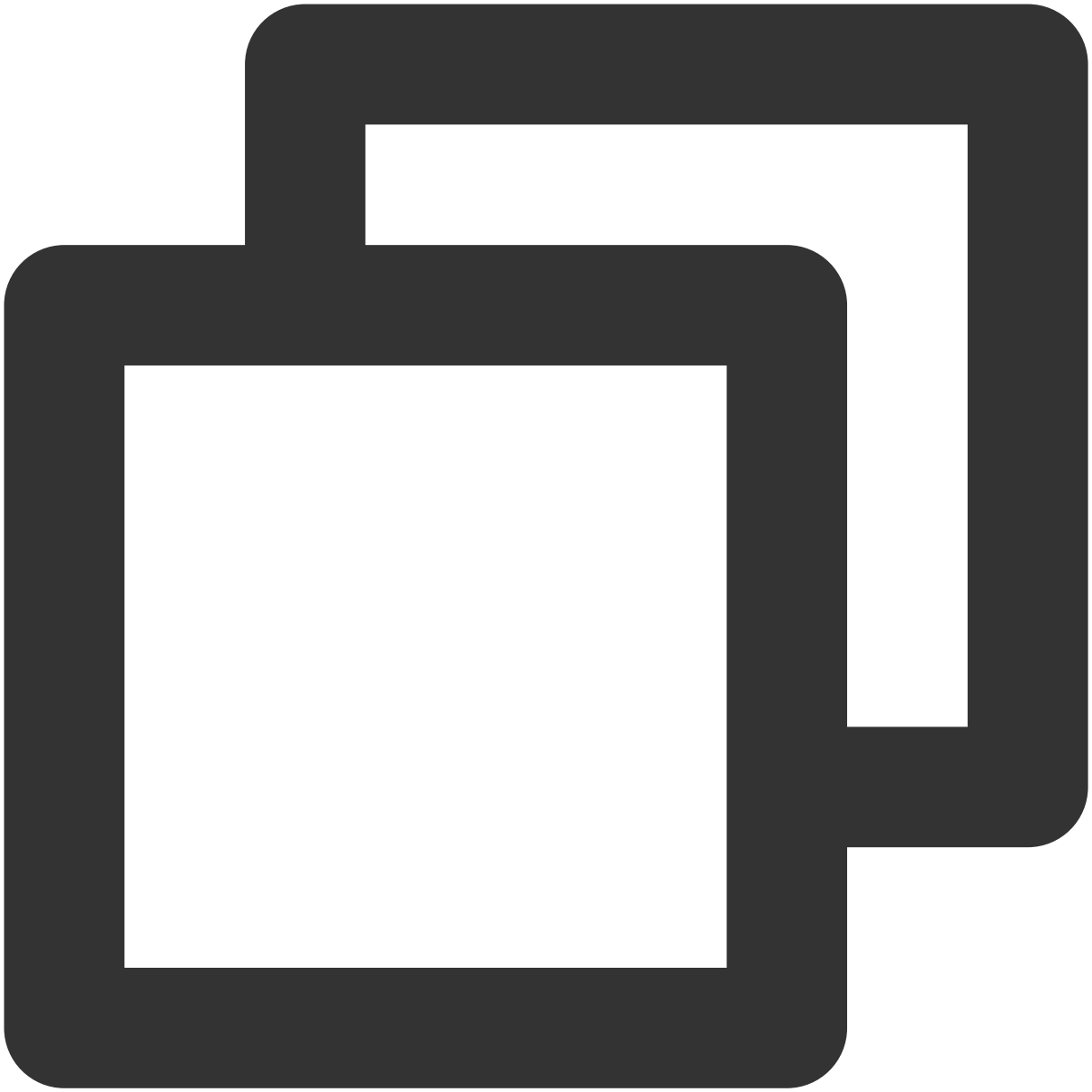


```
trtc.stopLocalAudio();
```

4） 扬声器检测

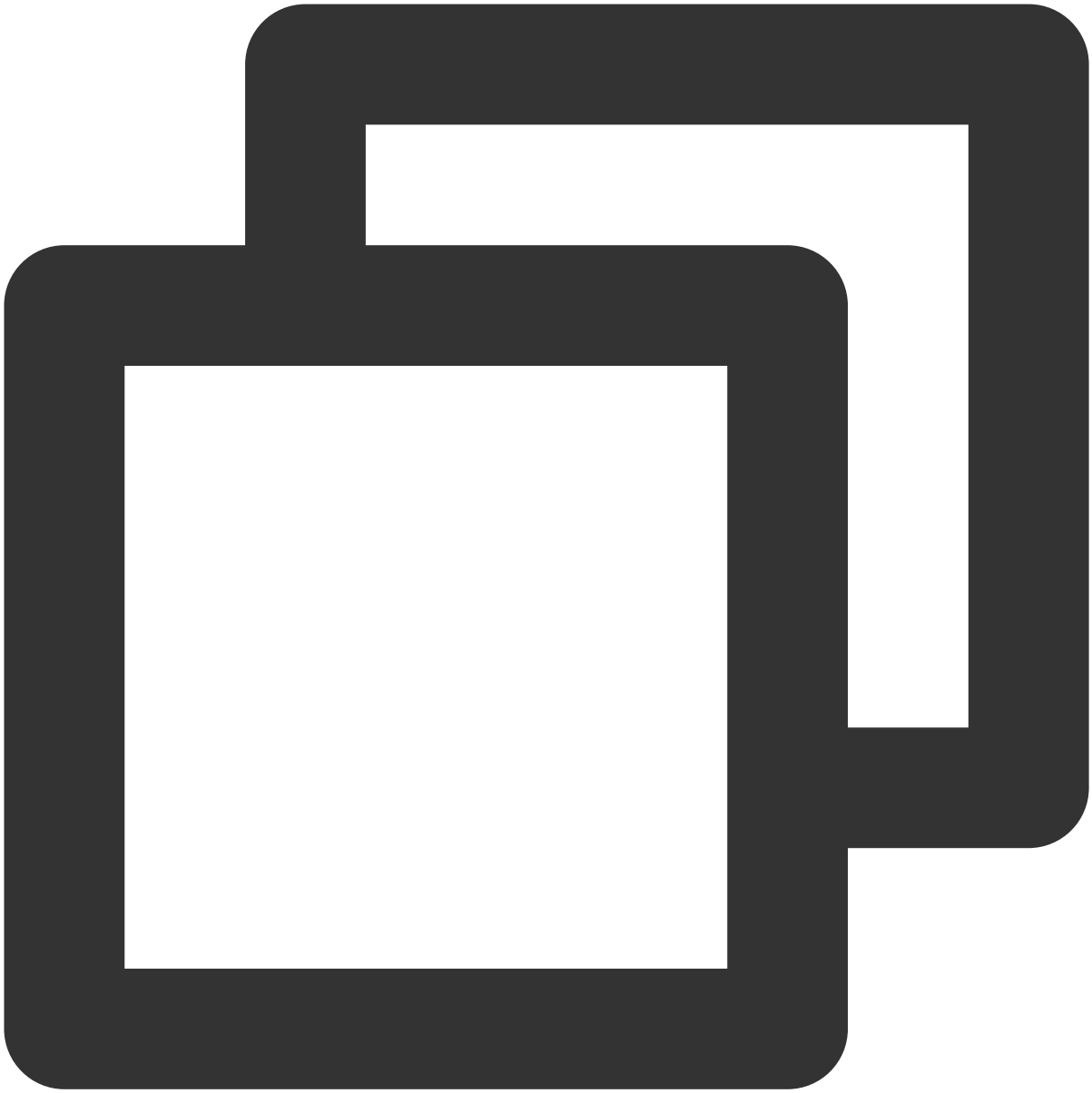
检测原理：通过 audio 标签播放一个 mp3 媒体文件

创建 audio 标签，提醒用户调高播放音量，播放 mp3 确认扬声器设备是否正常。



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

检测结束后停止播放



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

5） 网络检测

参考：通话前的网络质量检测。

TRTC 能力检测页面

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

测试网络质量
Android&iOS&Windows&Mac

最近更新时间：2023-05-31 11:30:58

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。

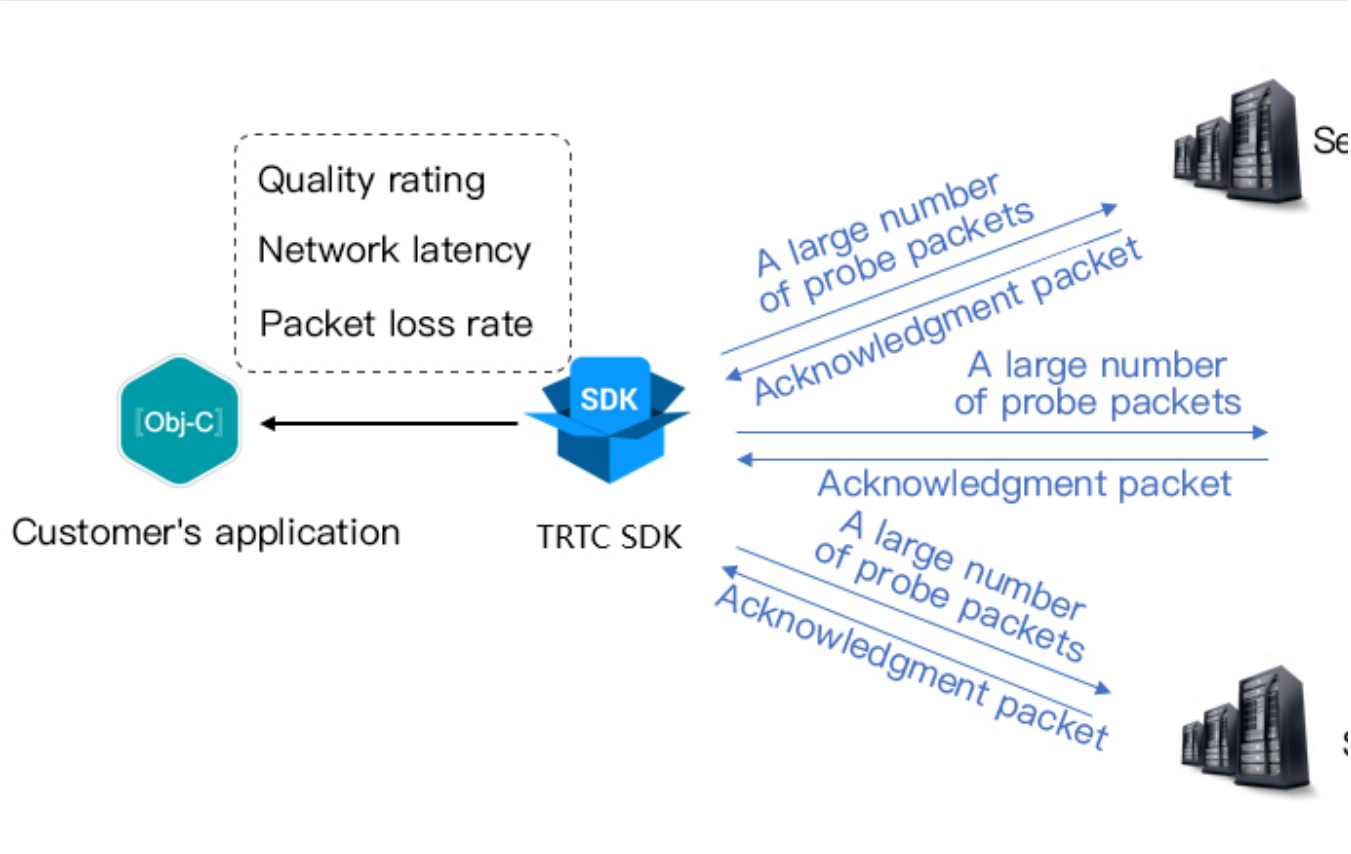
注意事项

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	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

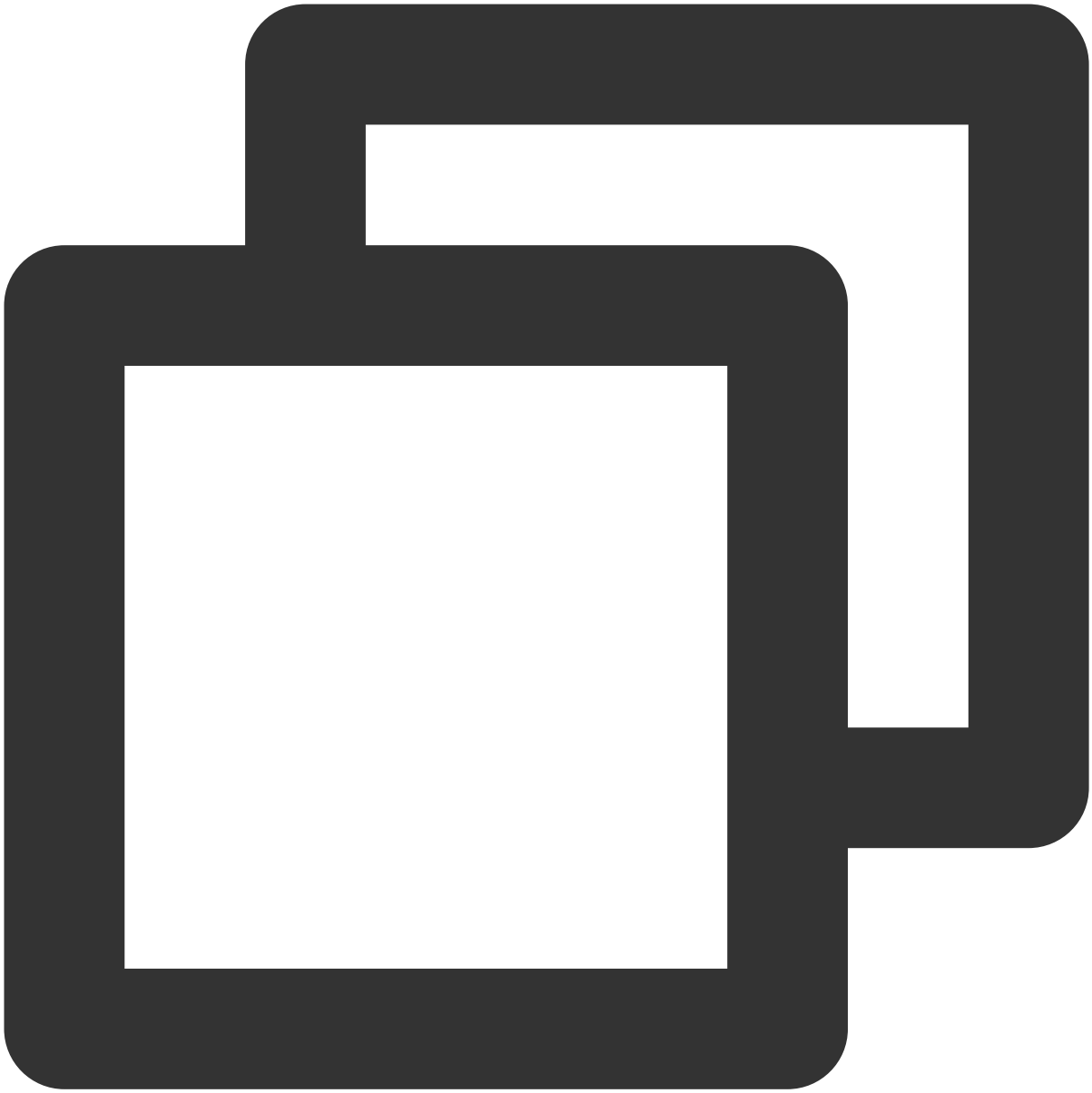
通过 TRTCCloud 的 `startSpeedTest` 功能可以启动测速功能，测速的结果会通过回调函数返回。

Objective-C

Java

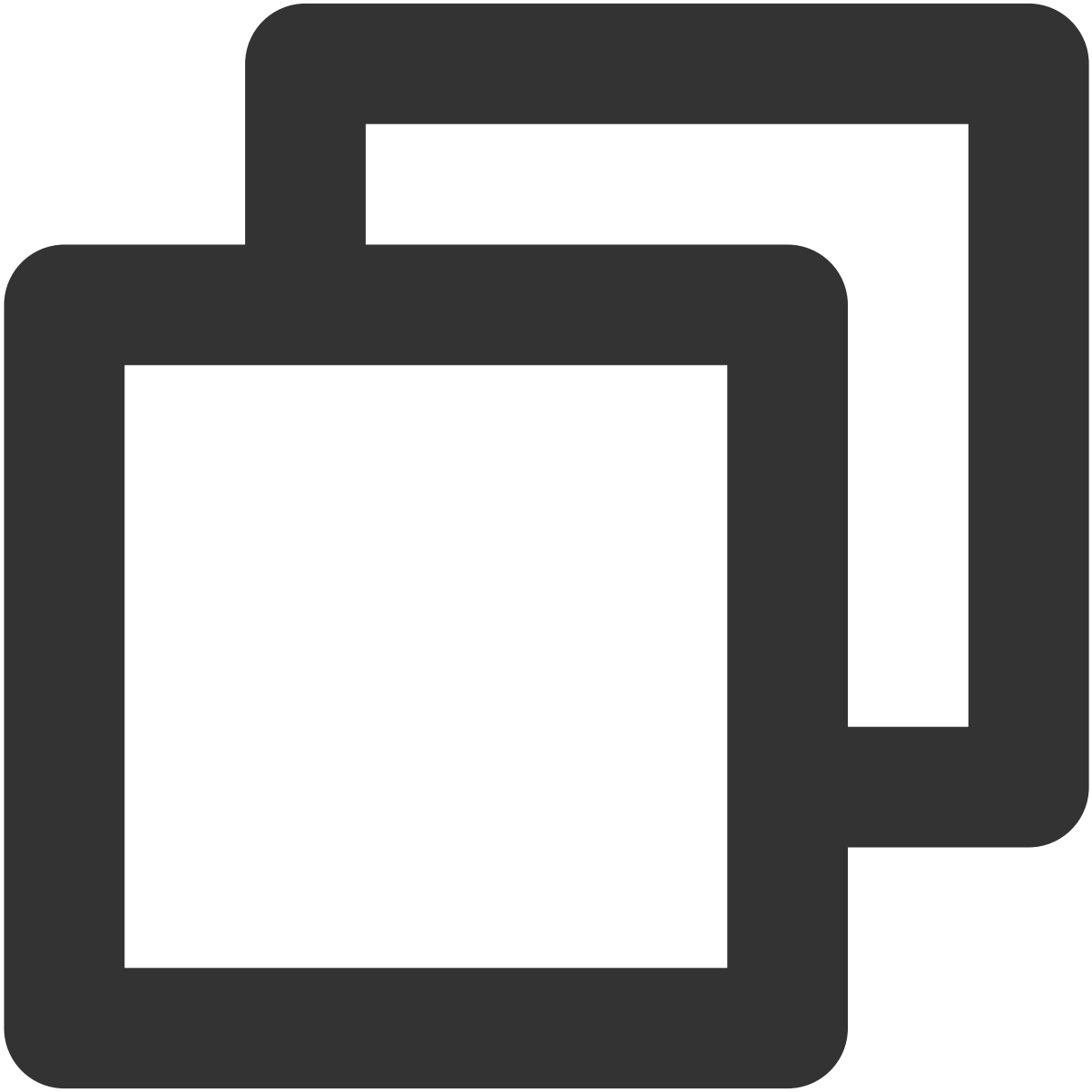
C++

C#



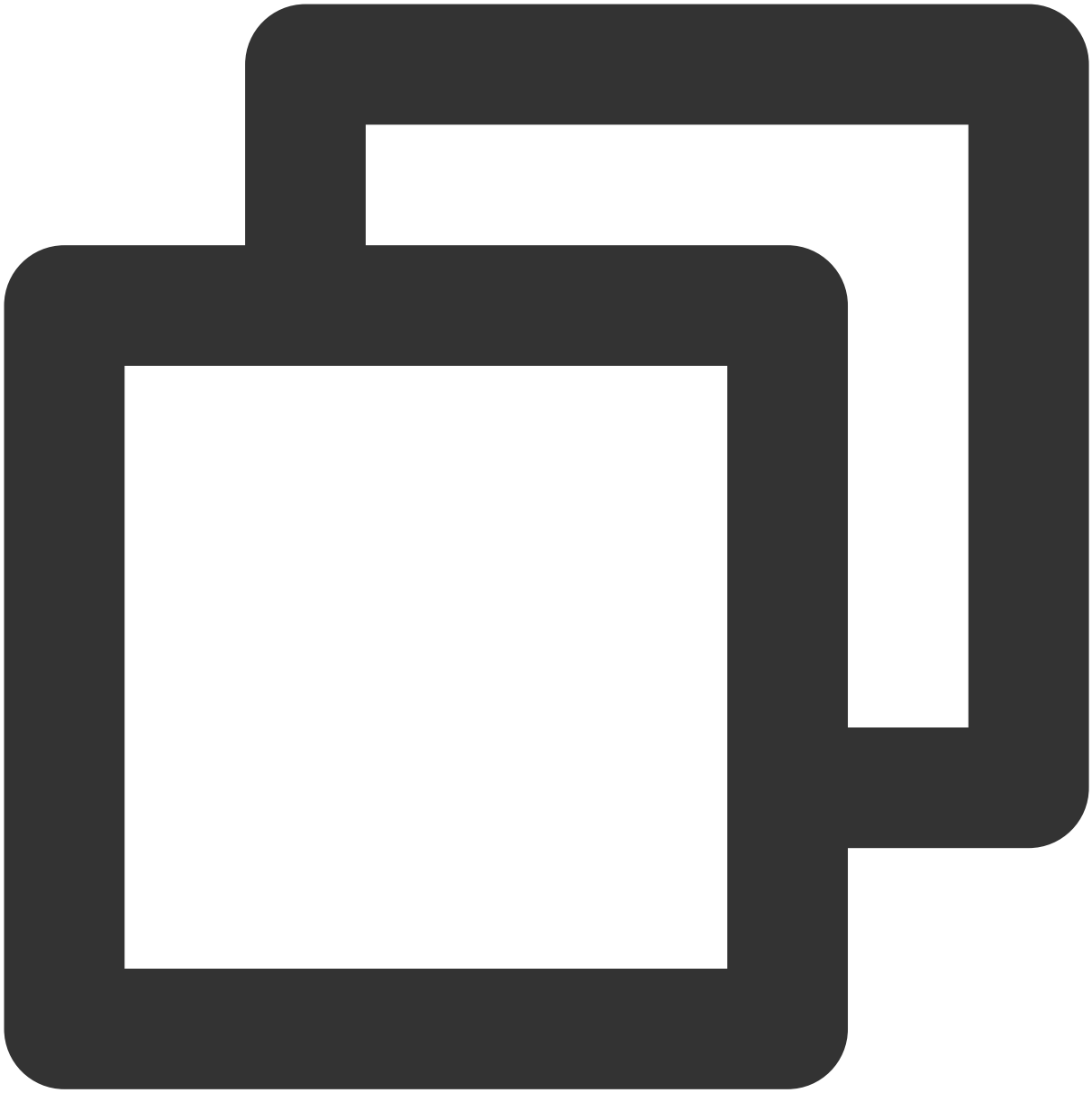
```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，（获取方式参考基本功能）
// 这里以登录后开始测试为例
- (void)onLogin:(NSString *)userId userSig:(NSString *)userSid
{
    TRTCSpeedTestParams *params;
    // sdkAppID 为控制台中获取的实际应用的 AppID
    params.sdkAppID = sdkAppId;
    params.userID = userId;
    params.userSig = userSig;
    // 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
}
```



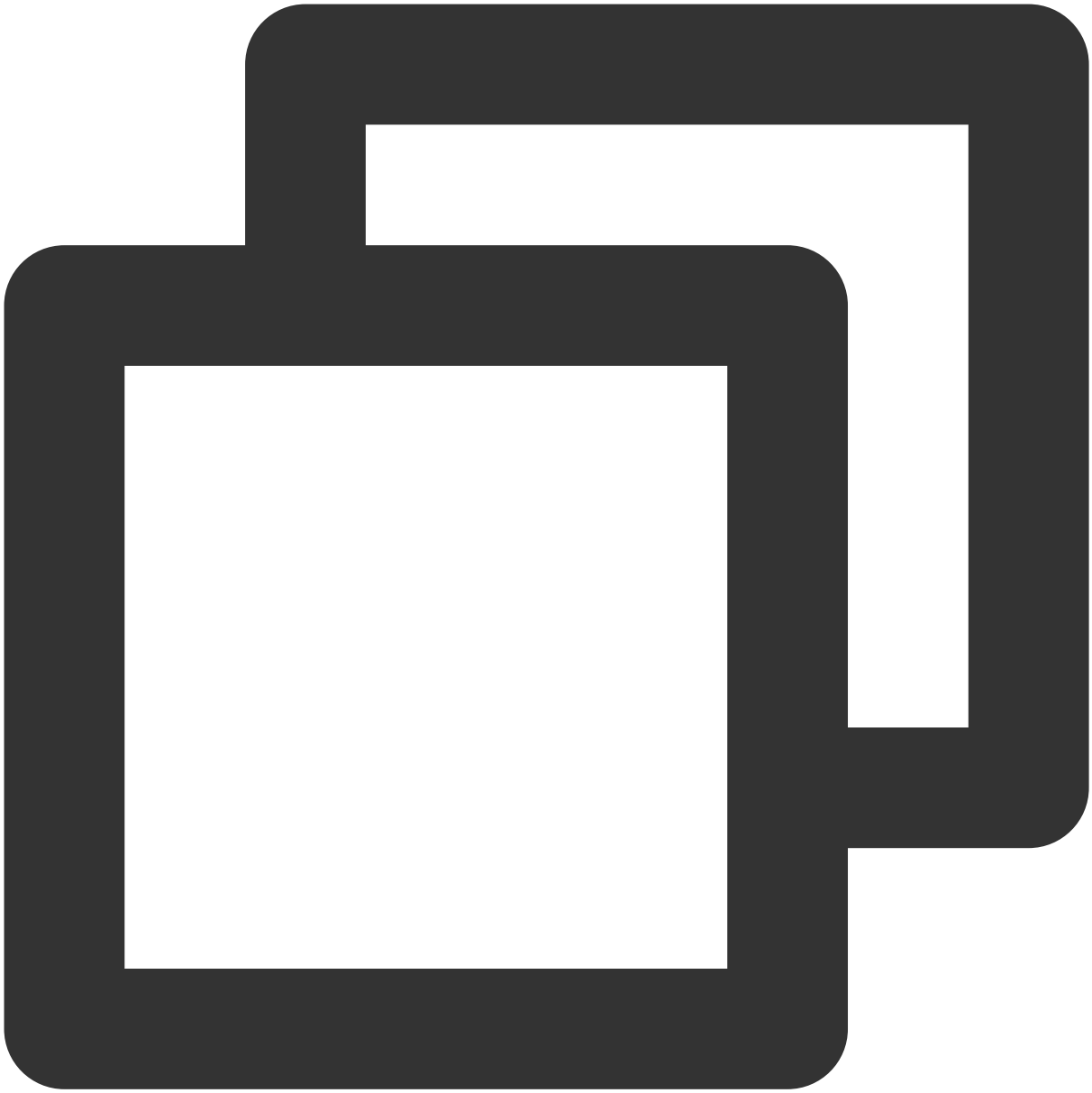
```
//启动网络测速的示例代码，需要 sdkAppId 和 UserSig，（获取方式参考基本功能）
// 这里以登录后开始测试为例
```

```
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
}
```



```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，（获取方式参考基本功能）
// 这里以登录后开始测试为例
void onLogin(const char* userId, const char* userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppID 为控制台中获取的实际应用的 AppID
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // 预期的上行带宽（kbps，取值范围： 10 ～ 5000，为 0 时不测试）
    param.expectedUpBandwidth = 5000;
```

```
    // 预期的下行带宽（kbps，取值范围： 10 ～ 5000，为 0 时不测试）
    param.expectedDownBandwidth = 5000;
    trtcCloud->startSpeedTest(params);
}

// 监听测速结果
void TRTCCloudCallbackImpl::onSpeedTestResult(
            const TRTCSpeedTestResult& result)
{
    // 测速完成后，会回调出测速结果
}
```



```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，（获取方式参考基本功能）
// 这里以登录后开始测试为例
private void onLogin(string userId, string userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppID 为控制台中获取的实际应用的 AppID
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // 预期的上行带宽（kbps，取值范围： 10 ～ 5000，为 0 时不测试）
    param.expectedUpBandwidth = 5000;
```

```
    // 预期的下行带宽（kbps，取值范围： 10 ～ 5000，为 0 时不测试）
    param.expectedDownBandwidth = 5000;
    mTRTCCloud.startSpeedTest(params);
}

// 监听测速结果
public void onSpeedTestResult(TRTCSpeedTestResult result)
{
    // 测速完成后，会回调出测速结果
}
```

测速工具

如果您不想通过调用接口的方式来进行网络测速，TRTC 还提供了桌面端的网络测速工具程序，帮助您快速获取详细的网络质量信息。

下载链接

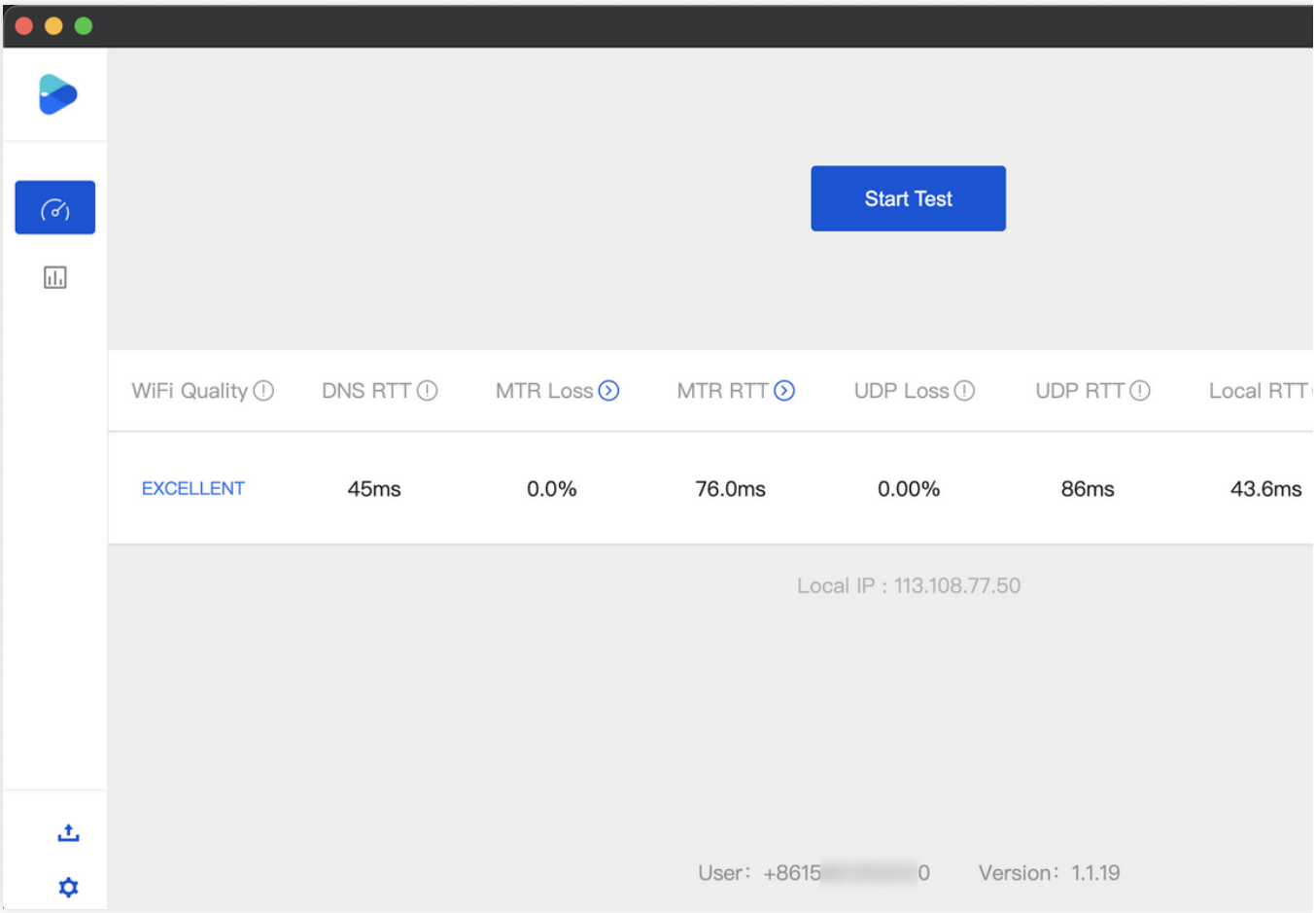
Mac | Windows

测试指标

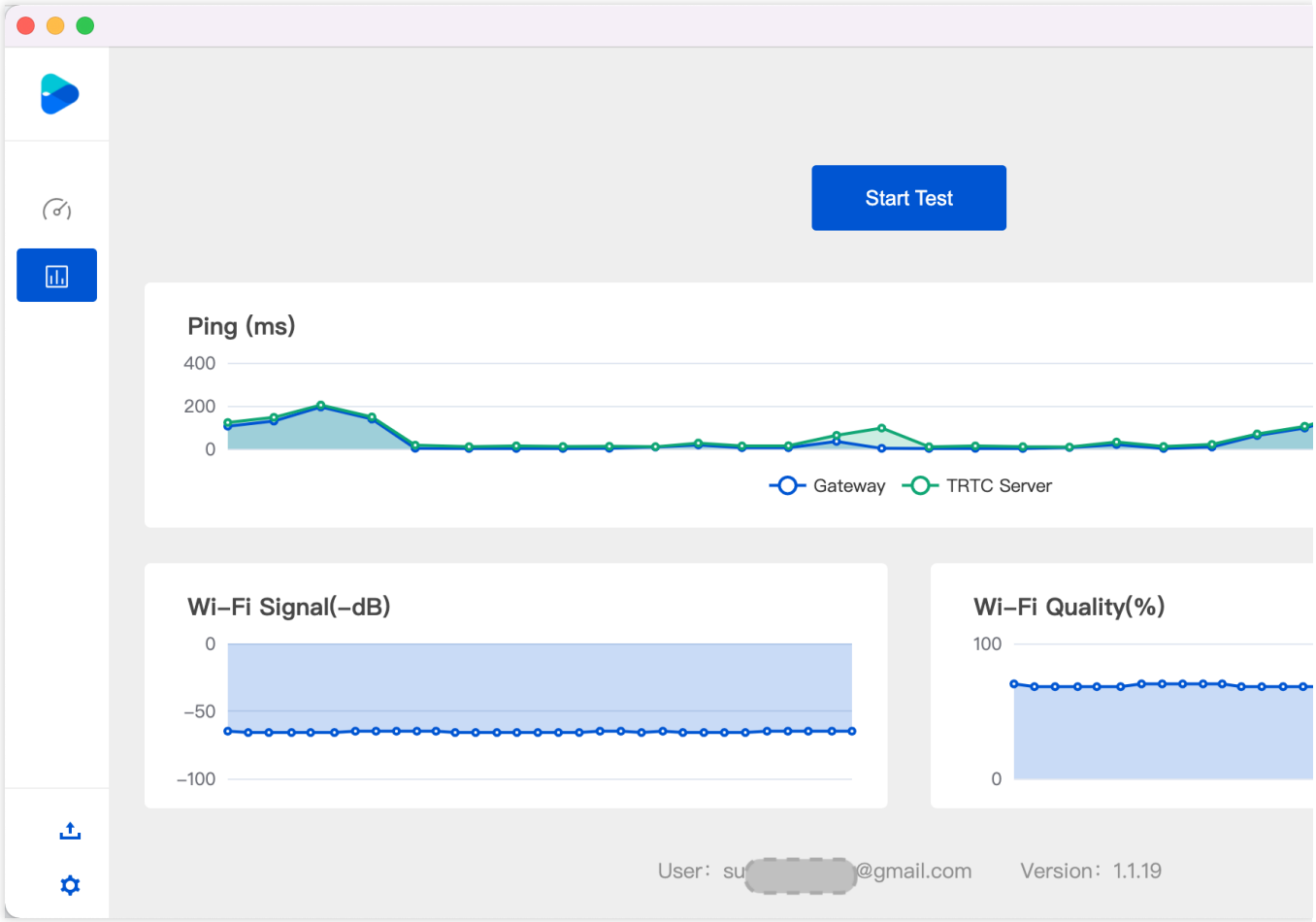
指标	含义
WiFi Quality	Wi-Fi 信号质量
DNS RTT	腾讯云的测速域名解析耗时
MTR	MTR 是一款网络测试工具，能探测客户端到 TRTC 节点的丢包率与延时，还可以查看路由中每一跳的具体信息
UDP Loss	客户端到 TRTC 节点的 UDP 丢包率
UDP RTT	客户端到 TRTC 节点的 UDP 延时
Local RTT	客户端到本地网关的延时
Upload	上行预估带宽
Download	下行预估带宽

工具截图

快速测试：


Start Test
WiFi Quality
DNS RTT
MTR Loss
MTR RTT
UDP Loss
UDP RTT
Local RTT
EXCELLENT
45ms
0.0%
76.0ms
0.00%
86ms
43.6ms
Local IP : 113.108.77.50
User: +8615 0
Version: 1.1.19

持续测试：

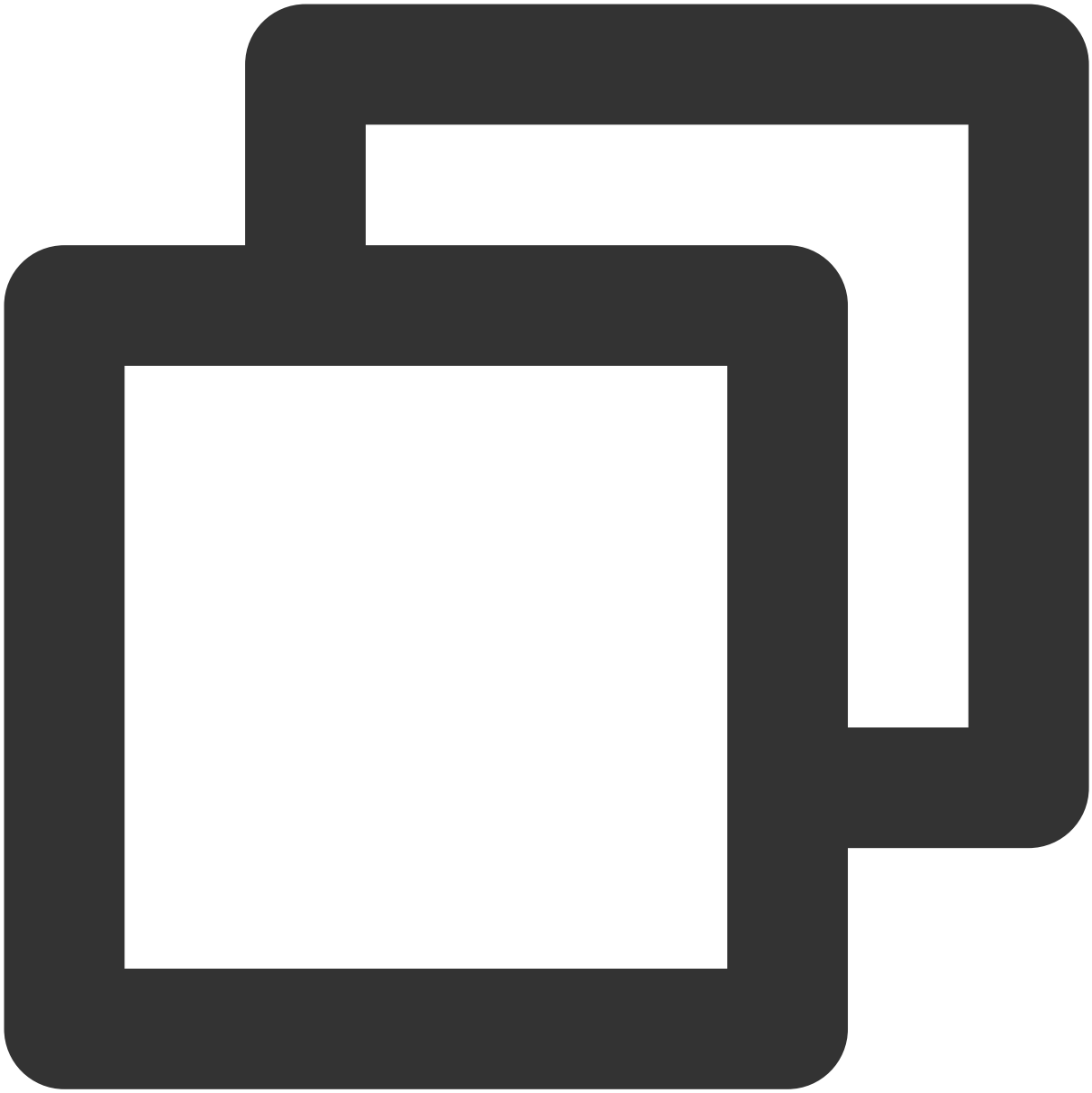

Start Test
Ping (ms)
400
200
0
Gateway
TRTC Server
Wi-Fi Signal(-dB)
0
-50
-100
Wi-Fi Quality(%)
100
0
User: su @gmail.com
Version: 1.1.19

Web

最近更新时间：2023-07-21 14:26:04

在进房之前或在通话过程中，检测用户的网络质量，可以判断用户当下的网络质量情况。若用户网络质量太差，应建议用户检查网络或尝试更换网络，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。

通话过程中的网络质量检测



```
const trtc = TRTC.create();
trtc.on(TRTC.EVENT.NETWORK_QUALITY, event => {
    console.log(`network-quality, uplinkNetworkQuality:${event.uplinkNetworkQuality}
    console.log(`uplink rtt:${event.uplinkRTT} loss:${event.uplinkLoss}`)
    console.log(`downlink rtt:${event.downlinkRTT} loss:${event.downlinkLoss}`)
})
```

通话前的网络质量检测

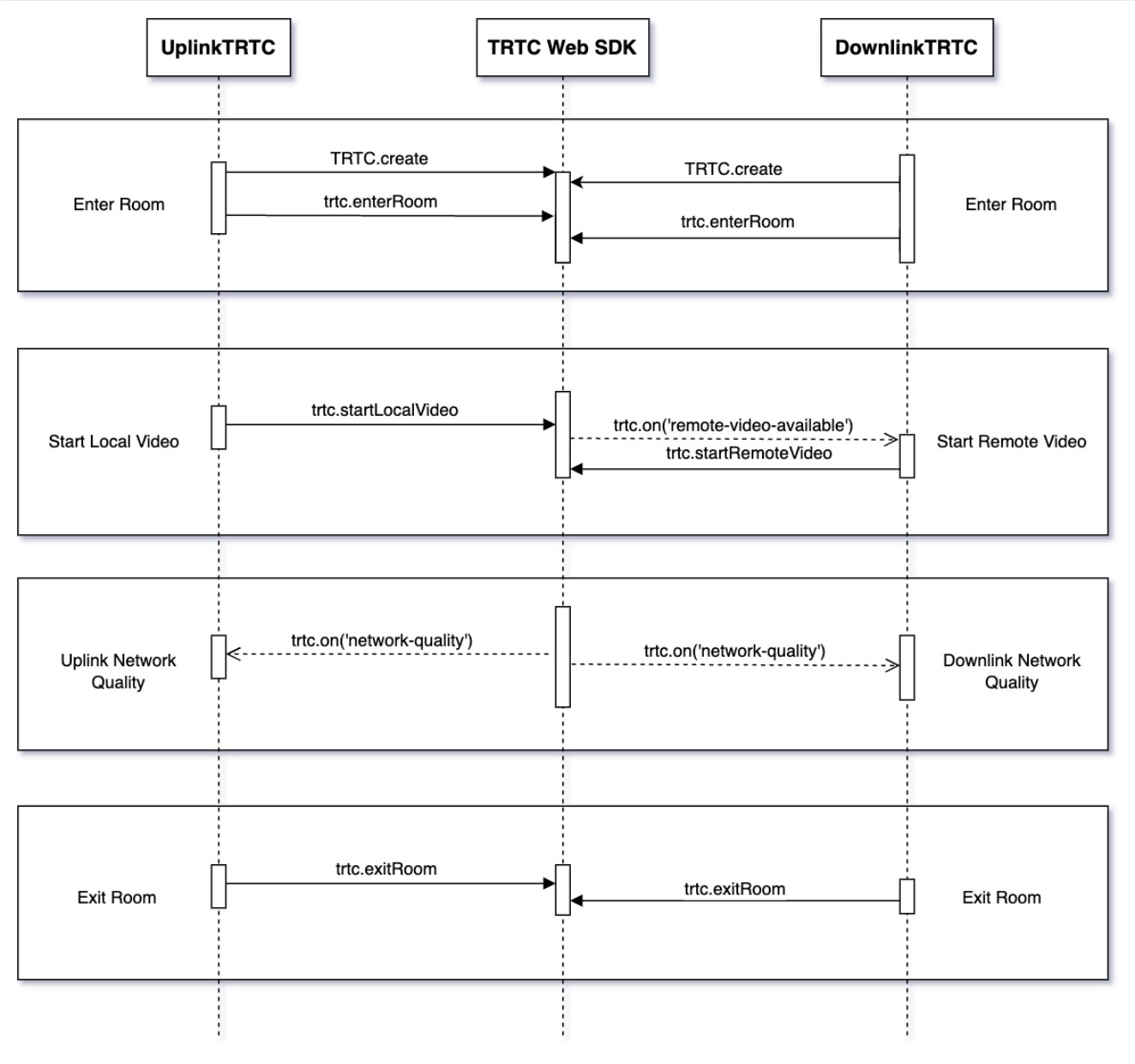
实现流程

1. 调用 `TRTC.create()` 方法创建两个 `TRTC`，分别称为 `uplinkTRTC` 和 `downlinkTRTC`。
2. 这两个 `TRTC` 都进入同一个房间。
3. 使用 `uplinkTRTC` 进行推流，监听 `TRTC.EVENT.NETWORK_QUALITY` 事件来检测上行网络质量。
4. 使用 `downlinkTRTC` 进行拉流，监听 `TRTC.EVENT.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。

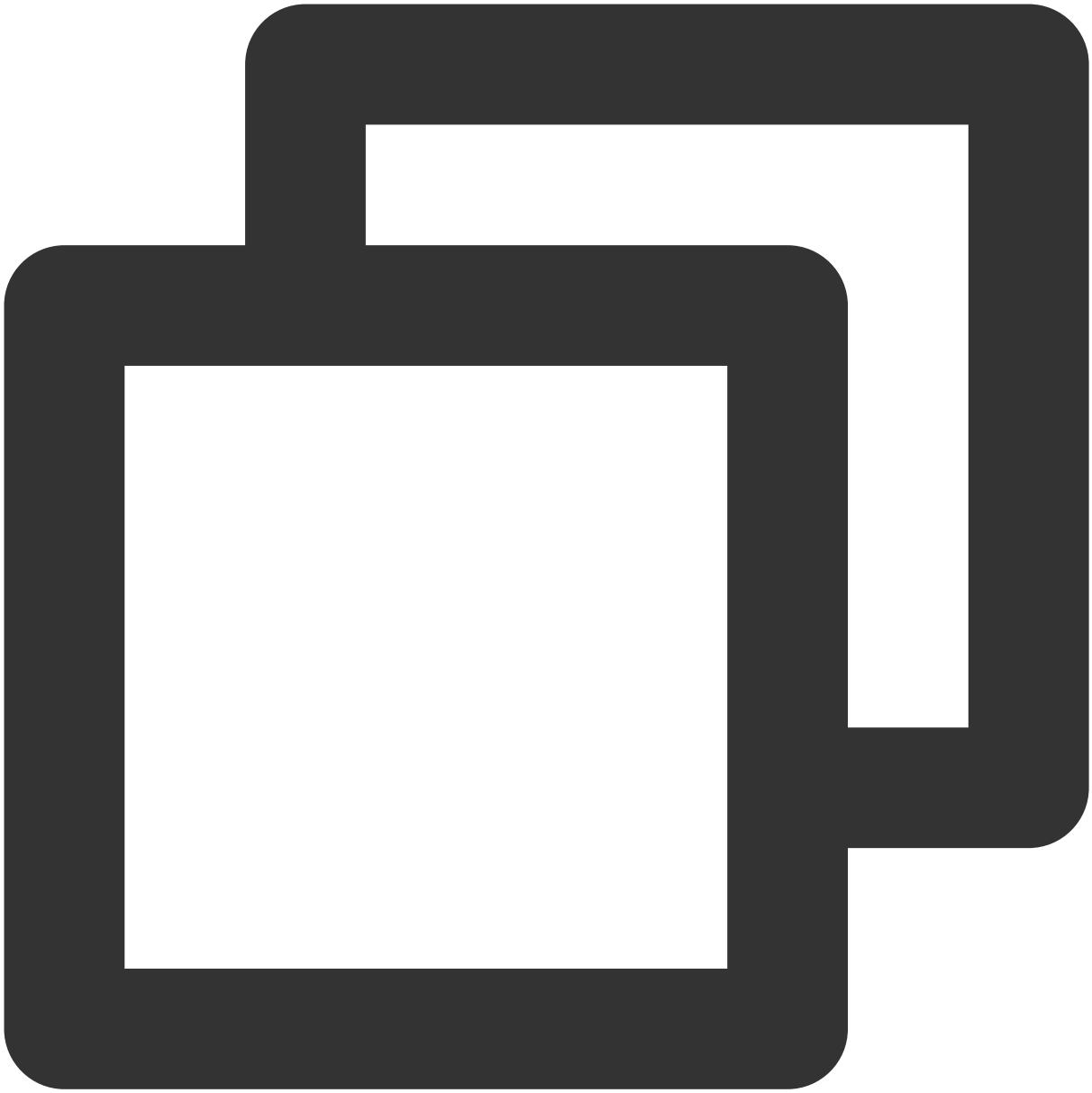
注意：

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。

API 调用时序


UplinkTRTC
TRTC Web SDK
DownlinkTRTC
Enter Room
TRTC.create
trtc.enterRoom
TRTC.create
trtc.enterRoom
Enter Room
Start Local Video
trtc.startLocalVideo
trtc.on('remote-video-available')
trtc.startRemoteVideo
Start Remote Video
Uplink Network Quality
trtc.on('network-quality')
trtc.on('network-quality')
Downlink Network Quality
Exit Room
trtc.exitRoom
trtc.exitRoom
Exit Room

代码示例



```
let uplinkTRTC = null; // 用于检测上行网络质量
let downlinkTRTC = 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() {
  uplinkTRTC = TRTC.create();

  uplinkTRTC.enterRoom({
    roomId: 8080,
    sdkAppId: 0, // 填写 sdkAppId
    userId: 'user_uplink_test',
    userSig: '', // uplink_test 的 userSig
    scene: 'rtc'
  })

  uplinkTRTC.on(TRTC.EVENT.NETWORK_QUALITY, event => {
    const { uplinkNetworkQuality } = event;
    testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
  });

}

// 2. 检测下行网络质量
async function testDownlinkNetworkQuality() {
  downlinkTRTC = TRTC.create();
  downlinkTRTC.enterRoom({
    roomId: 8080,
    sdkAppId: 0, // 填写 sdkAppId
    userId: 'user_downlink_test',
    userSig: '', // userSig
    scene: 'rtc'
  });

  downlinkTRTC.on(TRTC.EVENT.NETWORK_QUALITY, event => {
      const { downlinkNetworkQuality } = event;
      testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
  })
}

// 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,
    );
  }

  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 + curren
    );
  }

  // 检测结束，清理相关状态。
  uplinkTRTC.exitRoom();
  downlinkTRTC.exitRoom();
}, 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 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

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

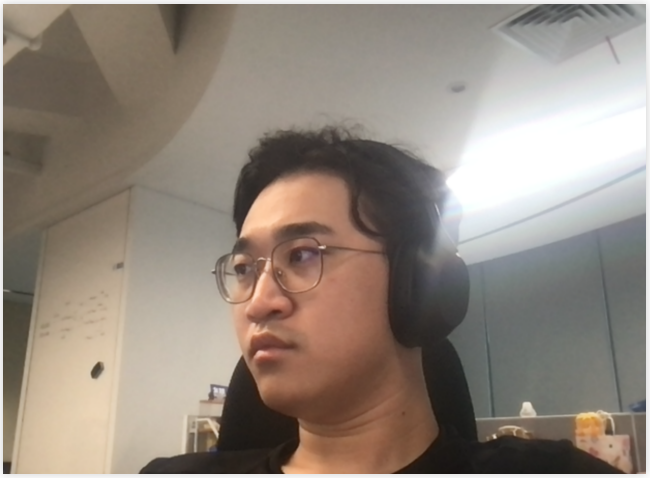
使用虚拟背景

Web

最近更新时间：2024-07-03 16:16:10

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。

原始摄像头	背景虚化
	

前提条件

需要使用虚拟背景功能的 SdkAppId 已开通 RTC Engine 包月套餐包 Pro版（1499美元/月）。

TRTC Web SDK 版本 ≥ 5.5.0

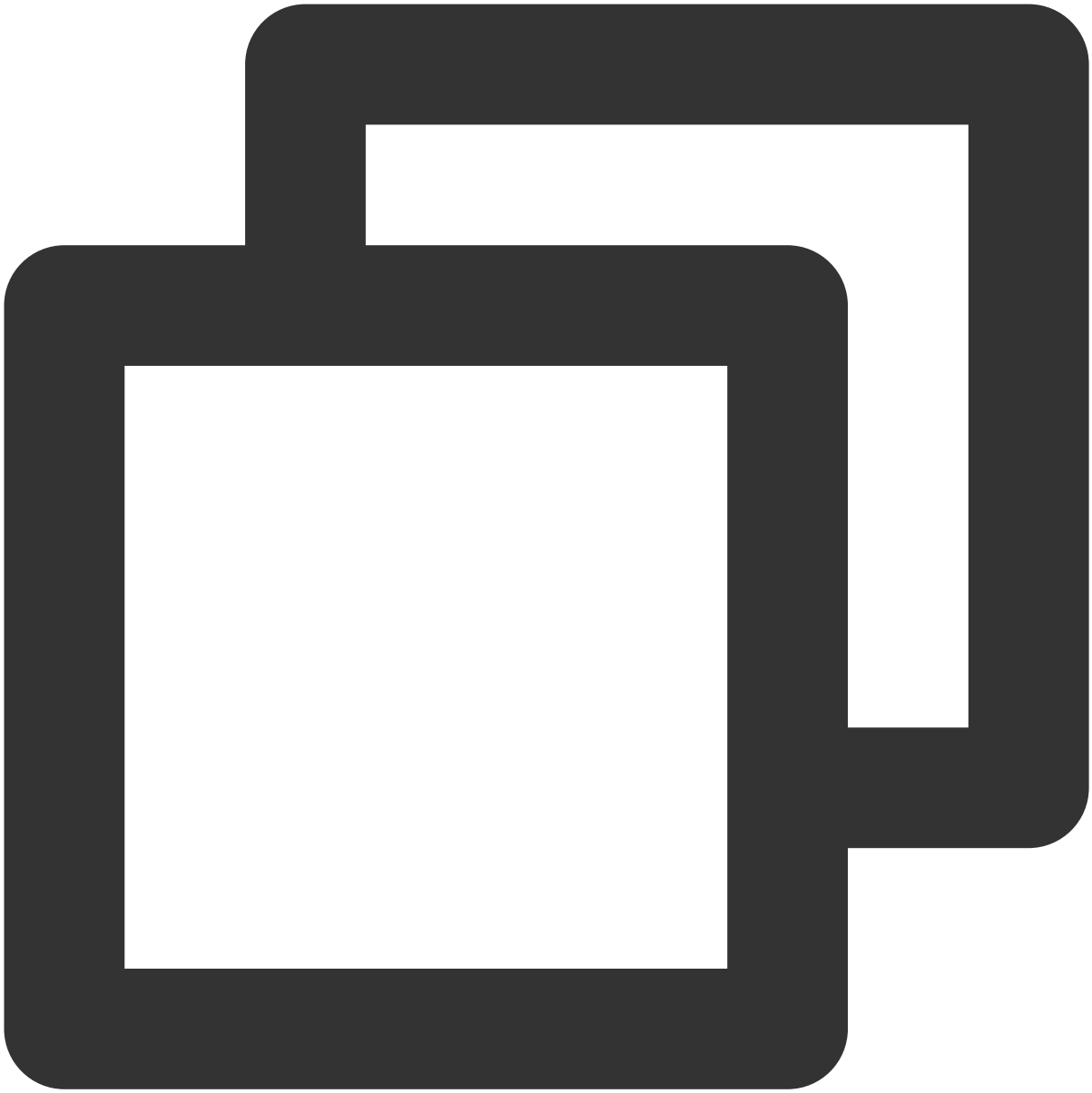
各平台系统及配置要求如下表：

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：16GBCPU：i5-10500GPU：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：8GBCPU：i3-8300GPU：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	
	Android	ChromeFirefox Browser	30	High-end devices (e.g., Qualcomm Snapdragon 8 Gen1)	建议使用 Chrome、火狐浏览器等主流浏览器
			20	Mid-range devices (e.g., MediaTek Dimensity 8000-MAX)	
			10	Low-end devices (e.g., Qualcomm Snapdragon 660)	
	iOS	ChromeSafariFirefox	30	iPhone 13	- 需要 iOS 14.4 或以上版本 - 建议使用 Chrome 或 Safari 浏览器
			20	iPhone XR	

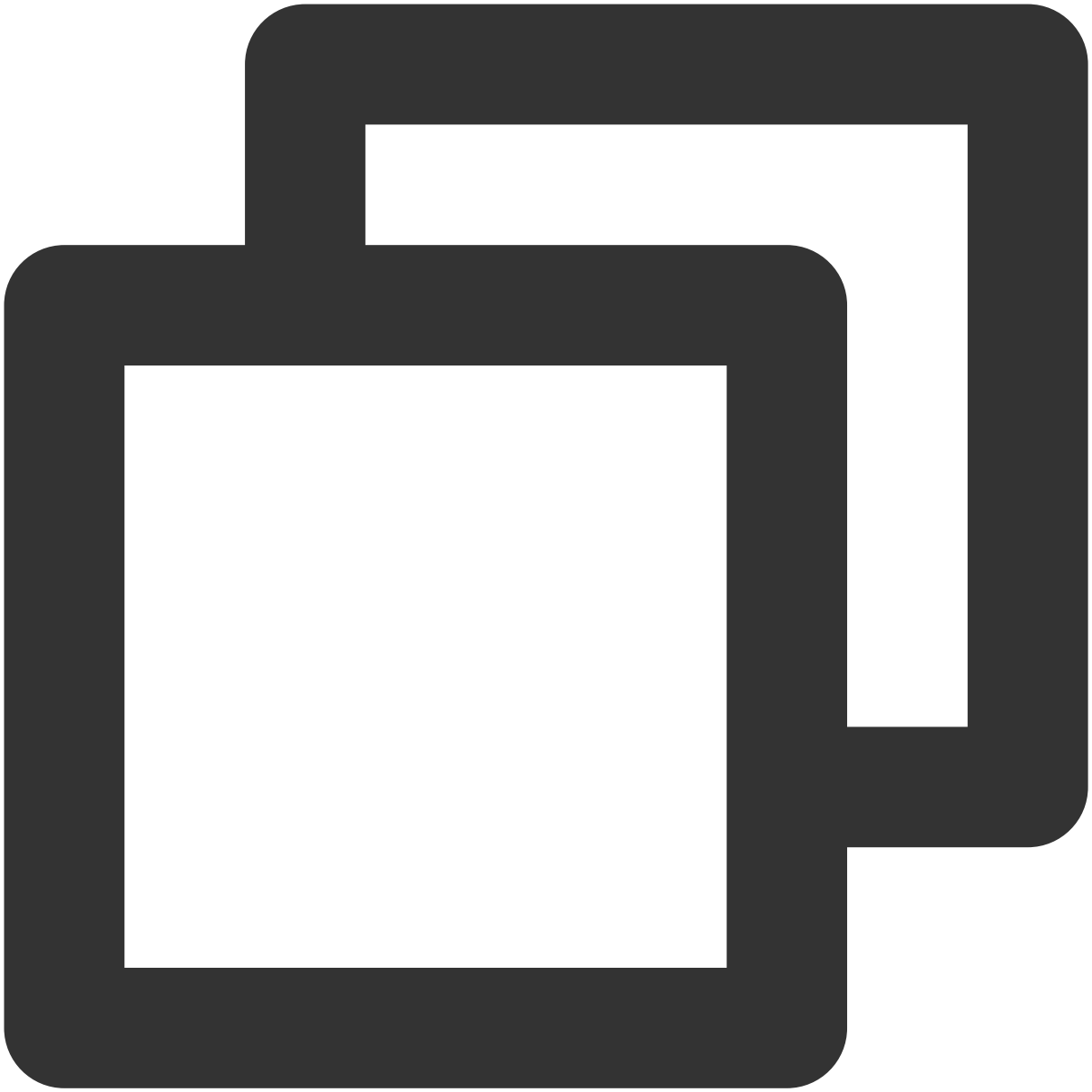
实现流程

引入并注册插件



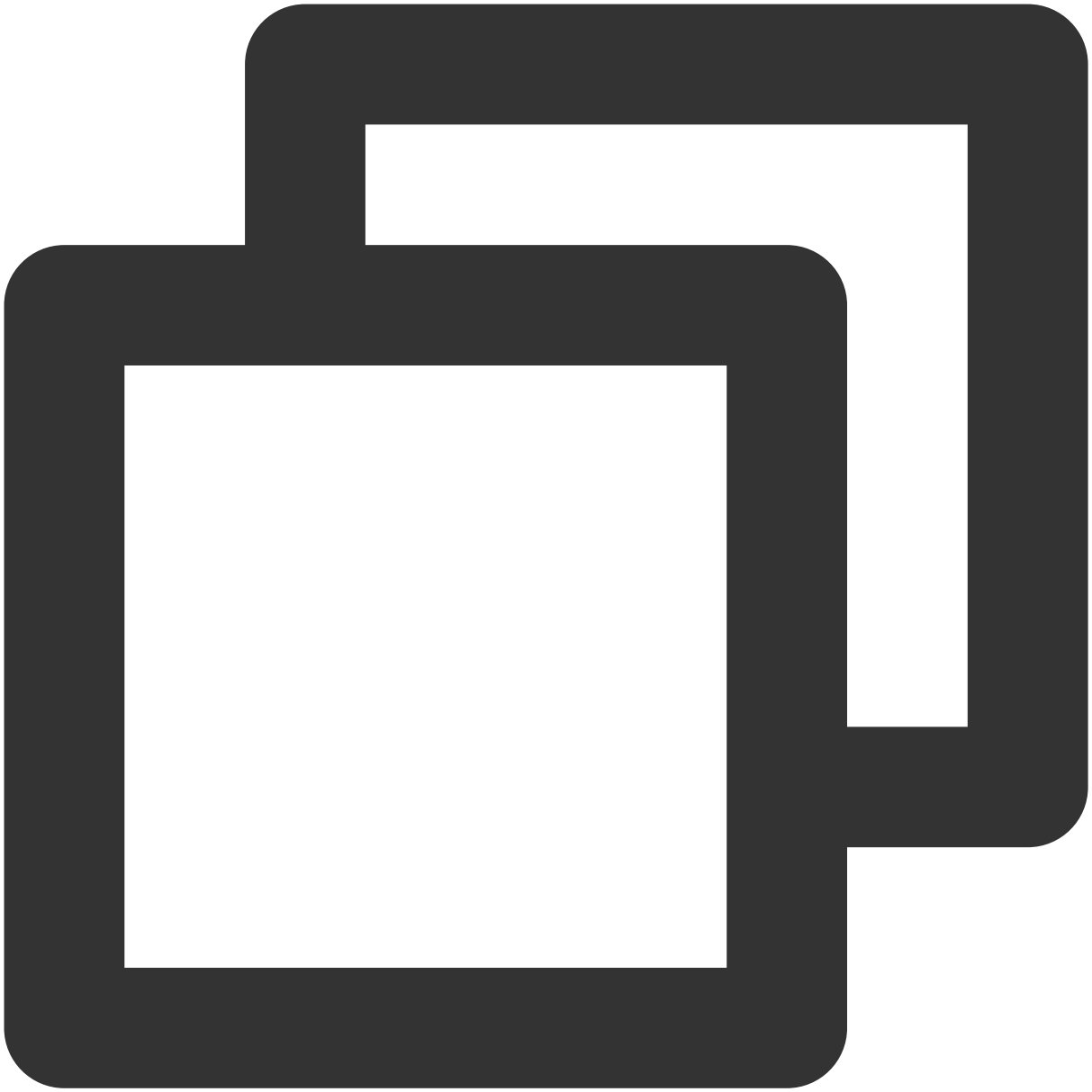
```
import { VirtualBackground } from 'trtc-sdk-v5/plugins/video-effect/virtual-backgro

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

开启本地摄像头



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

开启虚拟背景插件

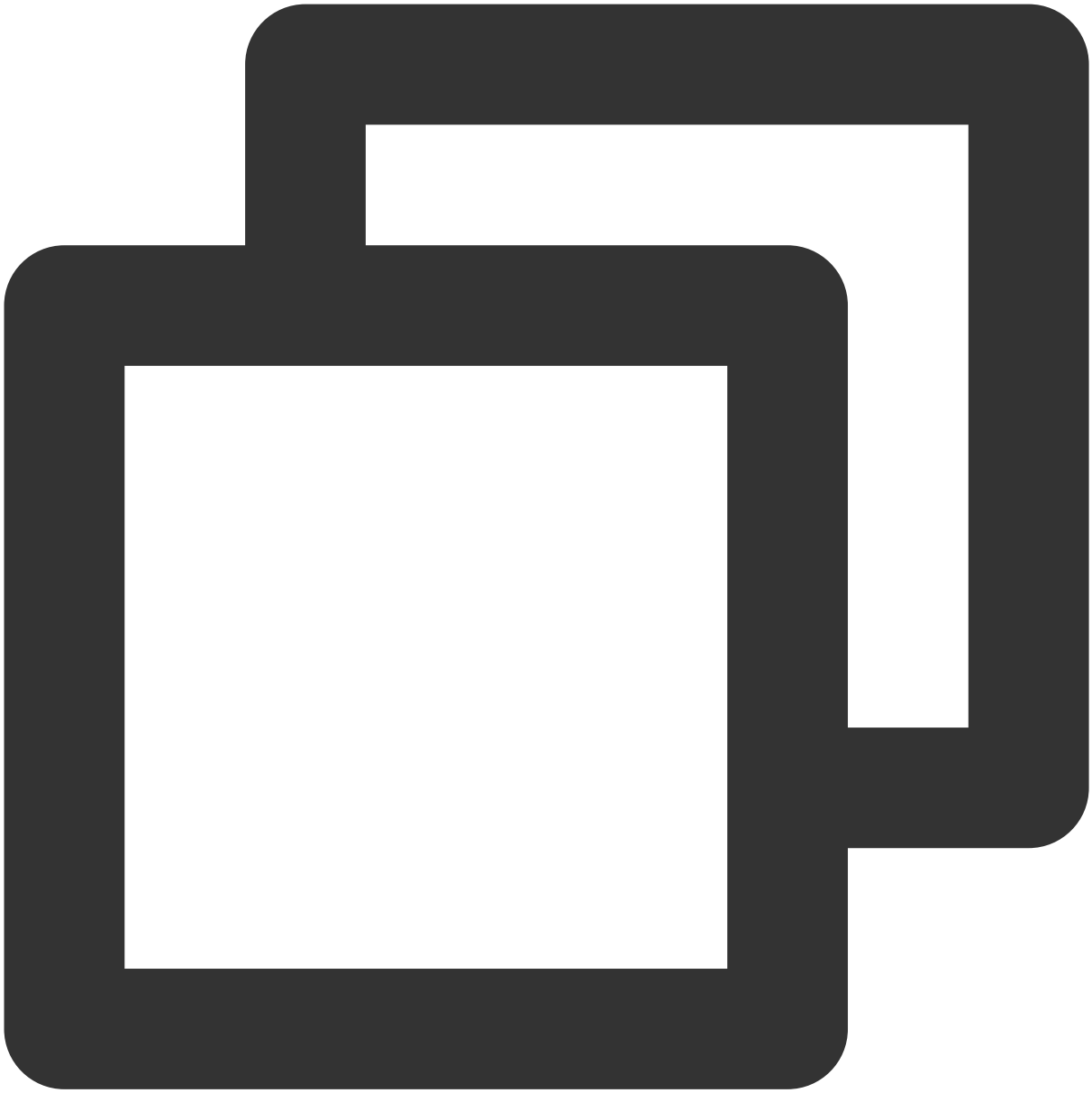


```
await trtc.startPlugin('VirtualBackground', {
  sdkAppId: 123123,
  userId: 'userId_123',
  userSig: 'your_userSig'
});
```

注意：

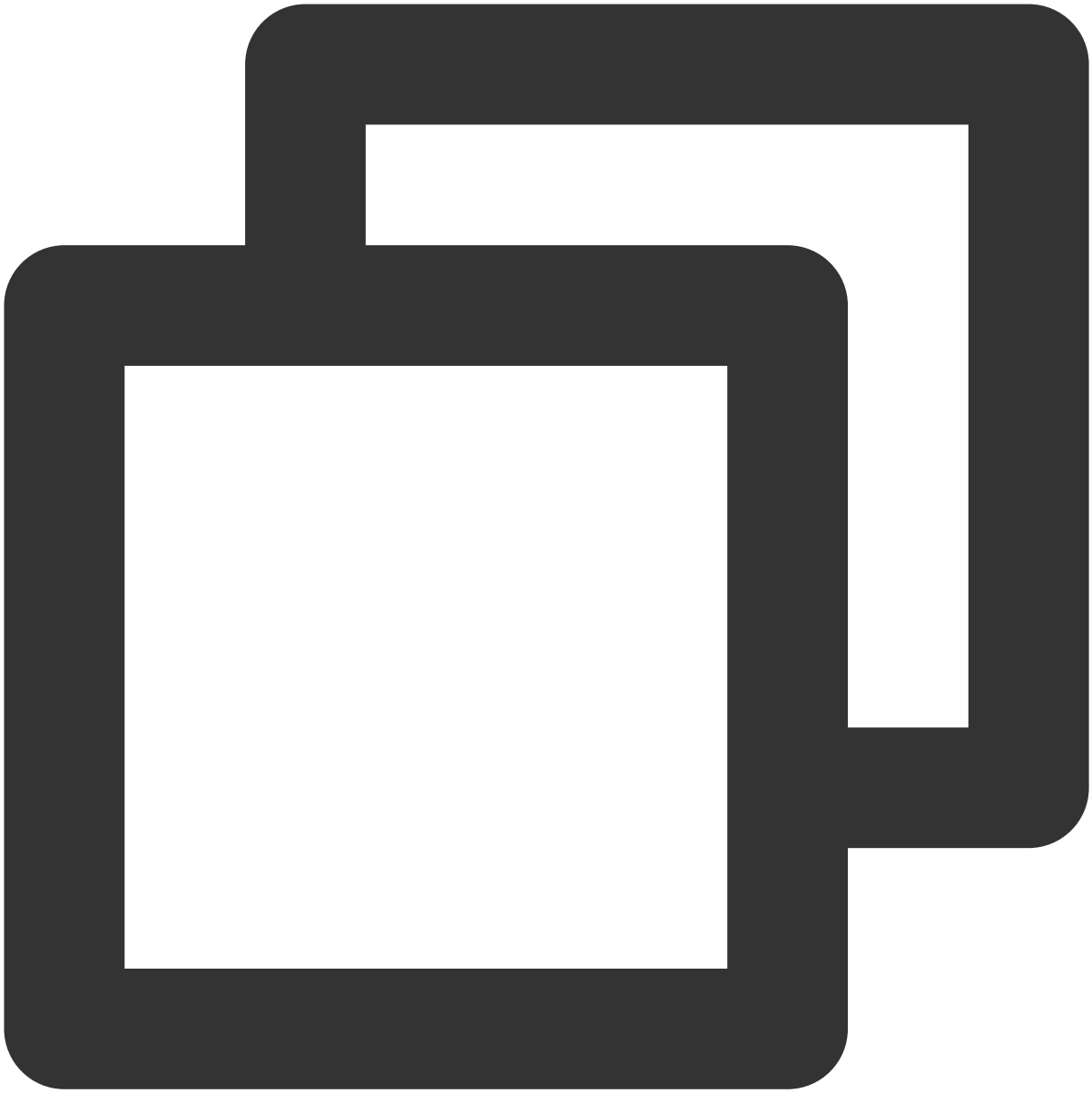
第一次开启本功能需要加载计算资源，已经帮您自动处理。如需获得更极致的启动体验，请参见文末：常见问题 > 如何加快启动速度？

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'
});
```

关闭虚拟背景



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

API 说明

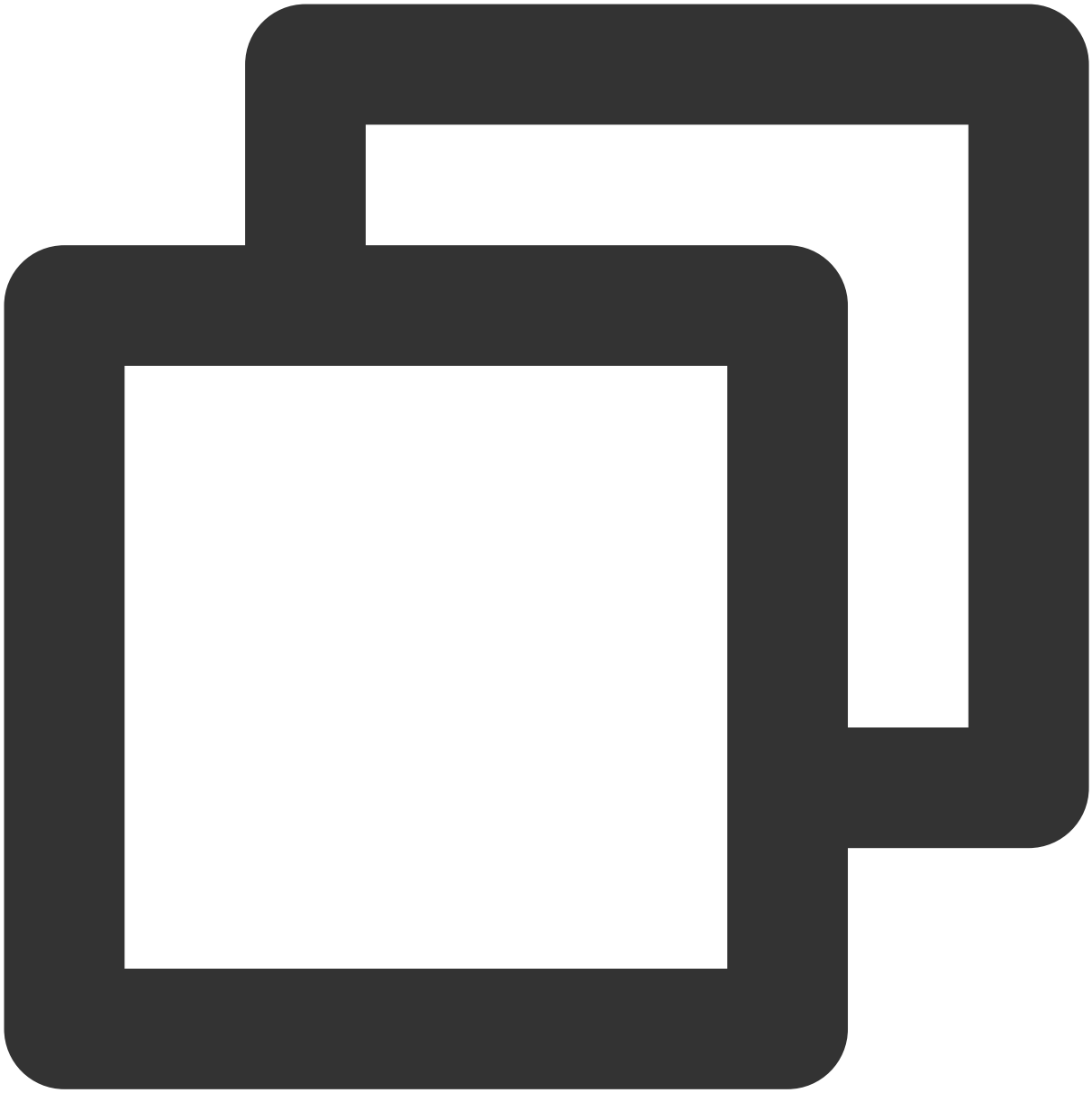
trtc.startPlugin('VirtualBackground', options)

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	`(error) => {}`	选填	运行过程中发生错误的回调 error.code=10000003 渲染耗时长 error.code=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文末常见问题

Example:



```
await trtc.startPlugin('VirtualBackground', {
  sdkAppId: 123123,
  userId: 'userId_123',
  userSig: 'your_userSig',
  type: 'image',
  src: 'https://picsum.photos/seed/picsum/200/300'
});
```

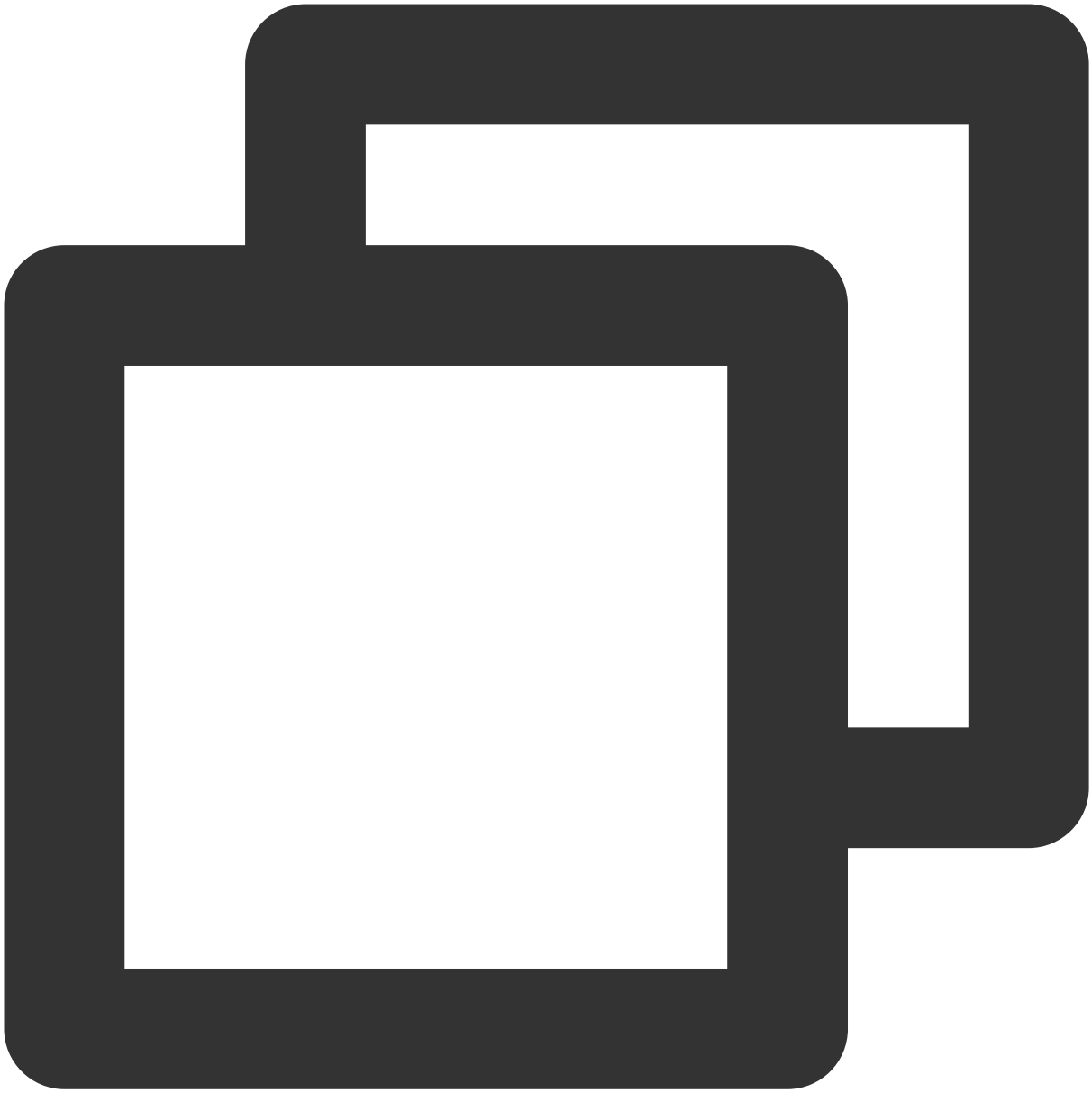
trtc.updatePlugin('VirtualBackground', options)

可修改虚拟背景参数

options

Name	Type	Attributes	Description
type	`string`	必填	`image` 图片背景 `blur` 虚化背景
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

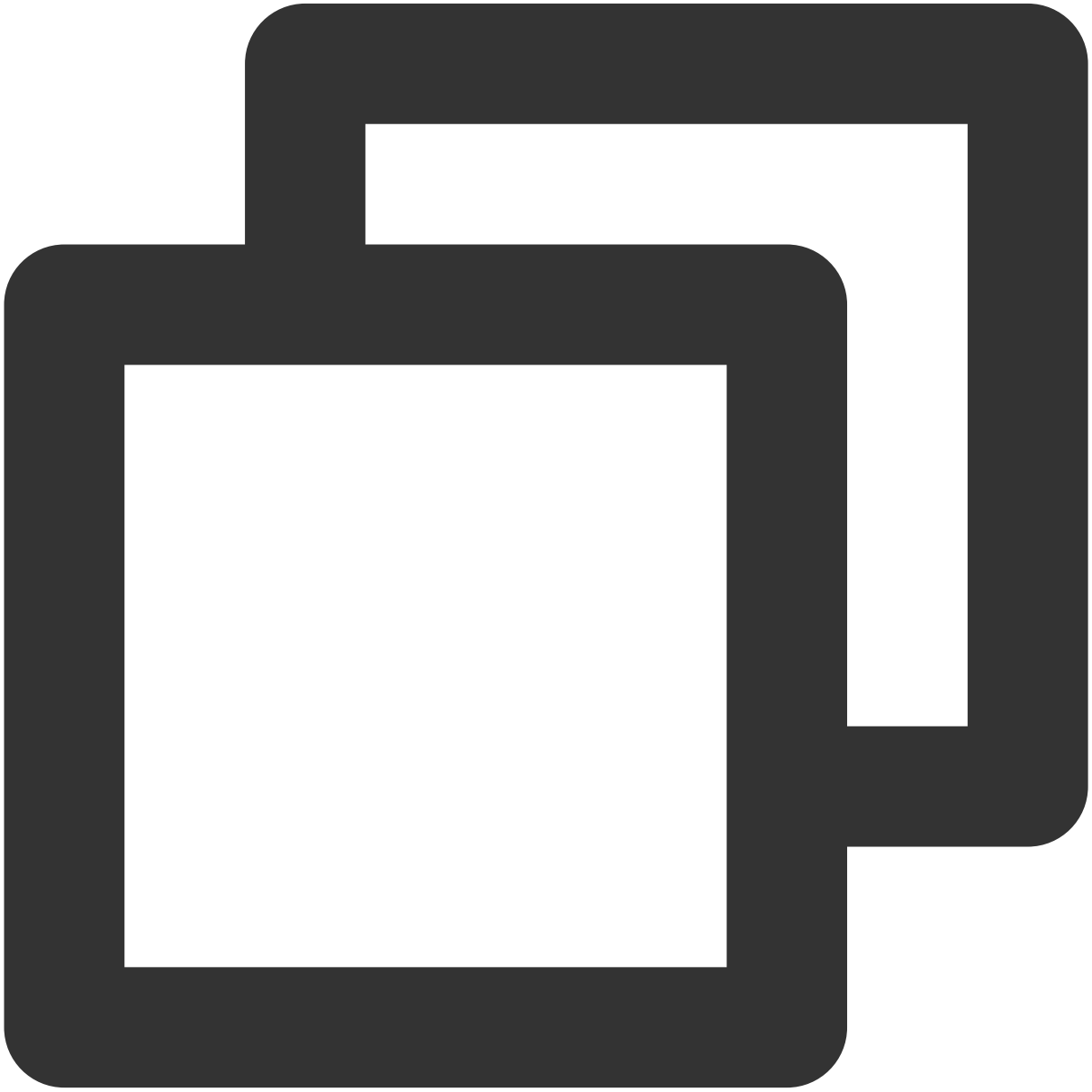
常见问题

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。

当设备性能不足造成延迟高，提示渲染耗时长？

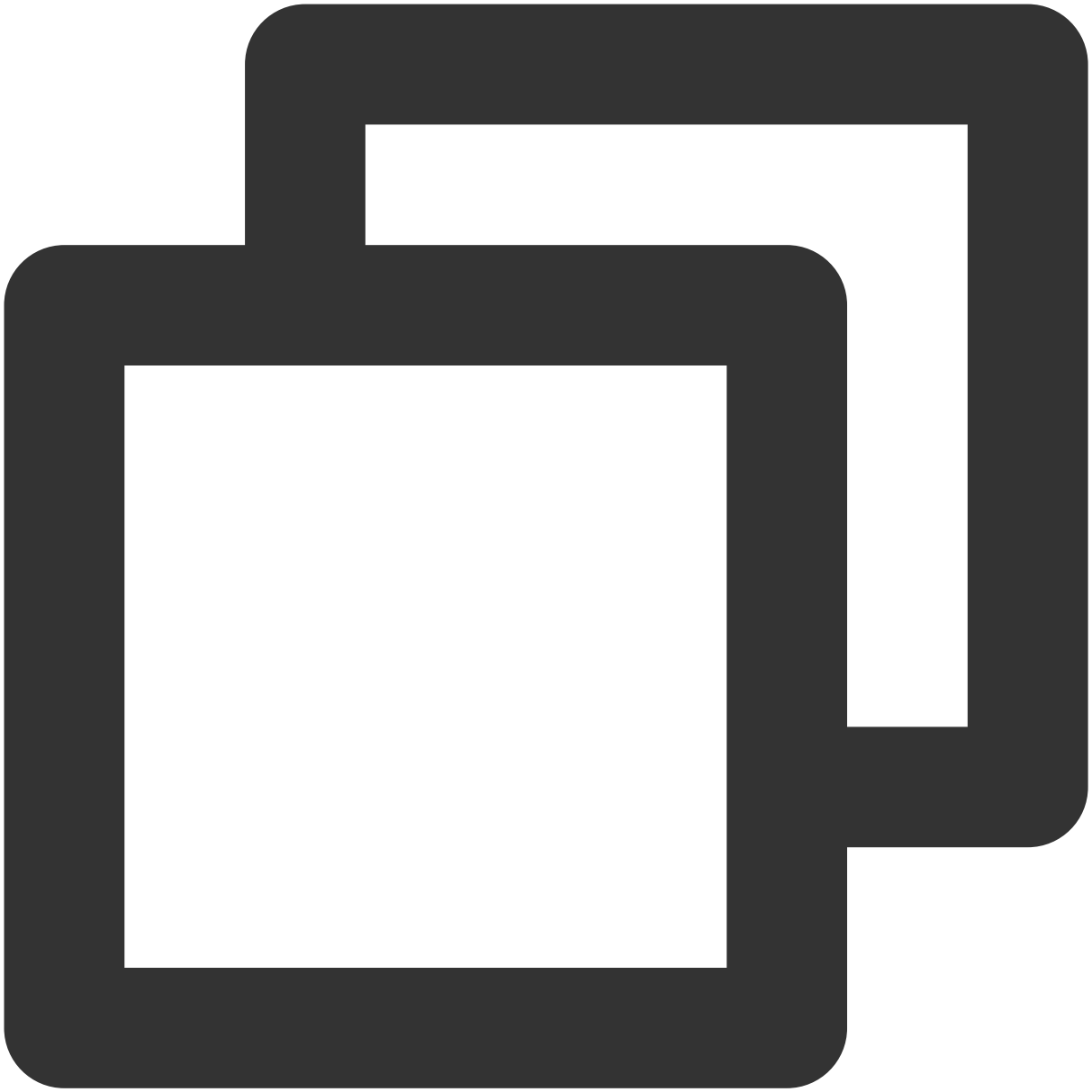
可通过监听事件，降低视频分辨率或者帧率。



```
async function onError(error) {
  const { code } = error;
  if (code === 10000003 || code === 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,
});
```

如何加快启动速度？（手动部署虚拟背景模型文件）

本插件的原理为：采集到摄像头后，在本地通过**机器学习模型**进行人像分割，然后将结果上行到后台传输。
第一次启用时，会自动从腾讯云 CDN 下载机器学习模型，但腾讯云 CDN 与您网页的静态文件地址是非同源的，会降低部分加载速度。**最佳方案是将模型文件部署在您自己的静态资源服务器上，比如跟网页服务部署在一起。**
加速方法很简单，将 assets.zip 解压到您的项目中，如 `./src/assets/` ，并且在创建 TRTC 实例时，指定这个地址：



```
const trtc = TRTC.create({ plugins: [VirtualBackground], assetsPath: './src/assets/
```

TRTC 云端录制说明

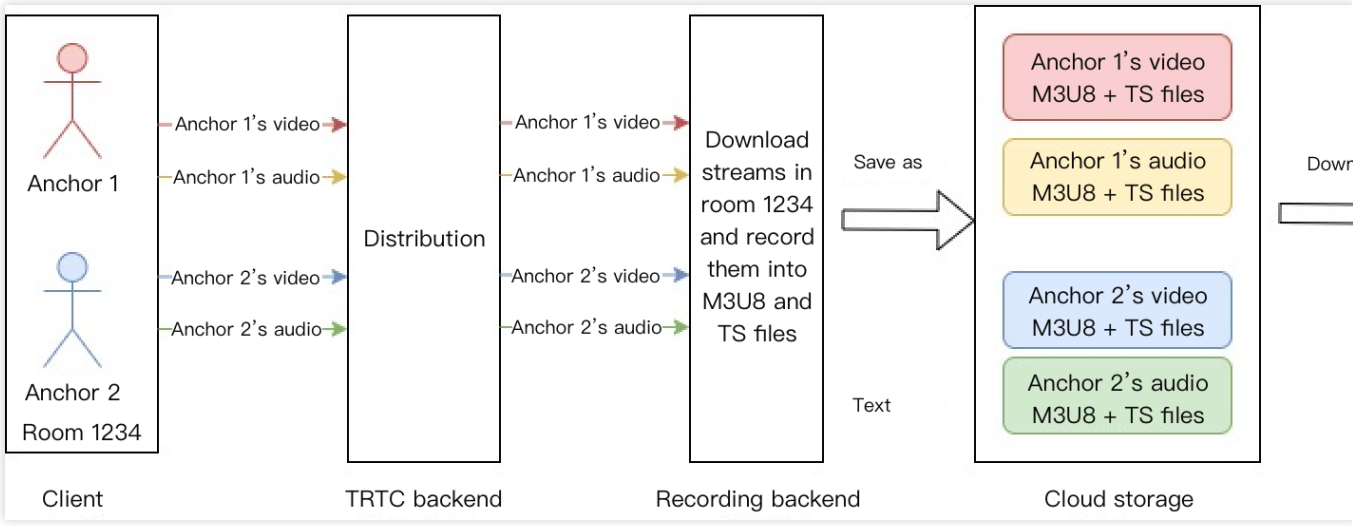
最近更新时间：2024-04-26 17:43:17

在线教育、秀场直播、视频会议、在线医疗、远程银行等应用场景中，考虑内容审核、录像存档和视频回放等需求，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，可以通过云端录制功能实现。

功能概述

通过 TRTC 的云端录制功能，您可以通过调用 REST API 接口，启动云端录制任务订阅需要录制的音视频流，实时灵活控制。并且无需开发者自行部署服务器和录制相关模块，更轻量便捷易用。
录制模式：单流录制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；混流录制可以把同一个房间的音视频媒体流混流录制成一个文件。
订阅流：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。
转码参数：混流的场景下，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。
混流参数：混流的场景下，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：支持指定录制的文件保存在云存储/云点播，当前云存储厂商支持腾讯云的 COS 存储，云点播厂商支持腾讯云点播
（未来会支持其他云厂商的存储和点播服务，届时需要您提供云服务账号，云存储服务需要提供存储参数）
回调通知：支持回调通知的能力，通过配置回调域名，云端录制的事件状态会通知到您的回调服务器。

单流录制



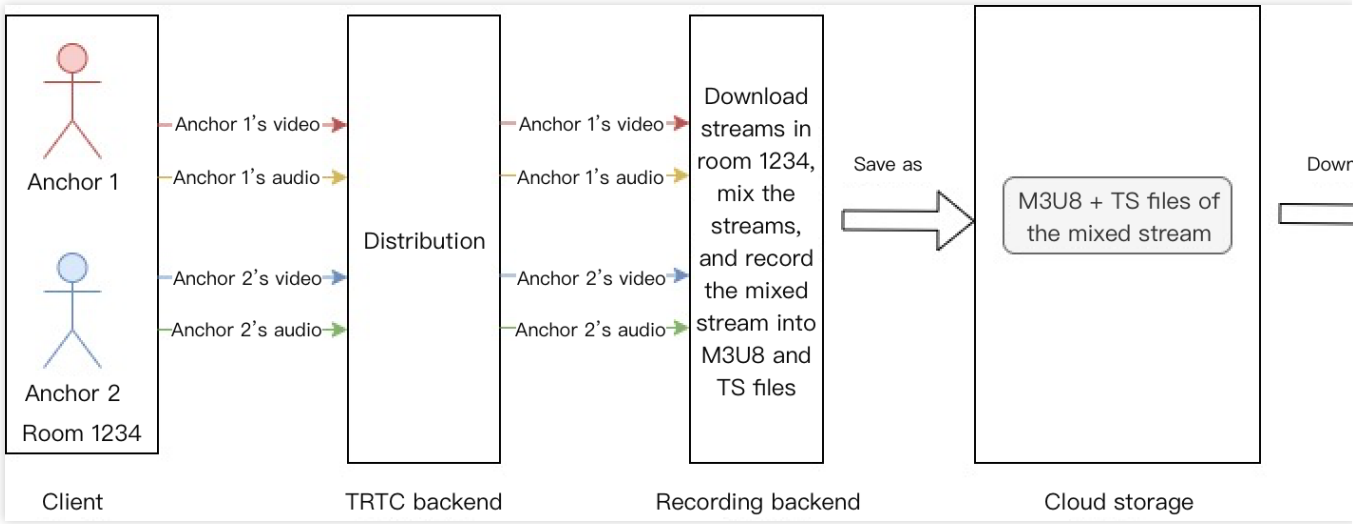
如图所示为单流录制的场景，房间1234里面主播1和主播2都上行了音视频流，假设您订阅了主播1和主播2的音视频流，并设置录制模式为单流录制，录制后台会分别拉取主播1和主播2的音视频流，并把他们录制成独立的媒体文

件，包含：

1. 主播1的一个视频M3U8索引文件；
2. 主播1的若干个视频TS切片文件；
3. 主播1的一个音频M3U8索引文件；
4. 主播1的若干个音频TS切片文件；
5. 主播2的一个视频M3U8索引文件；
6. 主播2的若干个视频TS切片文件；
7. 主播2的一个音频M3U8索引文件；
8. 主播2的若干个音频TS切片文件；

录制后台会把这些文件上传到您指定的云存储服务器，您的业务后台需要把这些录制文件拉取下来，并根据业务的需求对这些文件进行合并转码操作，我们会提供音视频合并转码脚本。

混流录制



如图所示为混流录制的场景，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，设置录制模式为混流录制，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混流，音频流进行混音，最后把媒体流混合成一路媒体文件，包含：

1. 混流后的的一个视频M3U8索引文件；
2. 混流后的若干个视频TS切片文件；

录制后台会把这些文件上传到您指定的云存储服务器，您的业务后台需要把这些录制文件拉取下来，并根据业务的需求对这些文件进行合并转码操作，我们会提供音视频合并转码脚本。

注意：

录制接口的调用频率限制为20qps

单个接口超时时间为6秒

默认并发录制支持100路，如果需要更多路数，请提交工单联系我们。

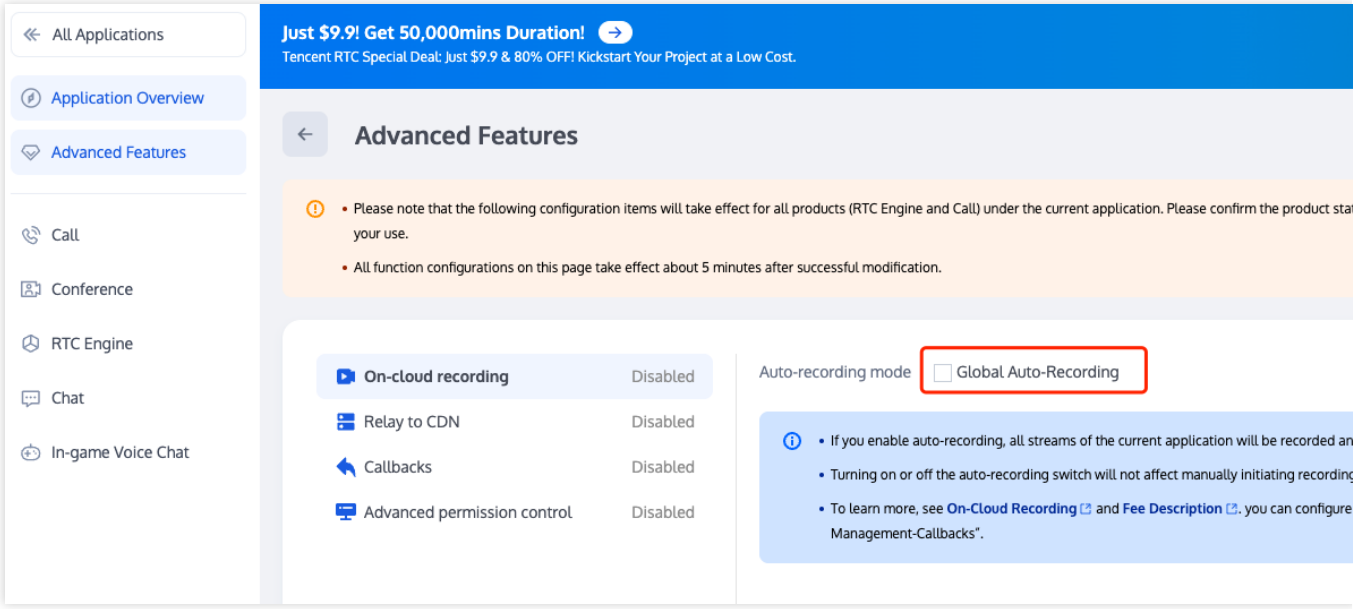
单录制任务最大支持同时订阅的单房间内音视频流为25路。

自动录制

TencentRTC 提供自动录制功能，无需手动管理录制任务。要使用该种录制方案，请前往**控制台** > 应用管理，选择所需应用并点击右侧

，进入应用信息界面。然后点击**高级功能**，在配置中开启云端录制自动录制，完成全局自动录制模板配置并启用即可。

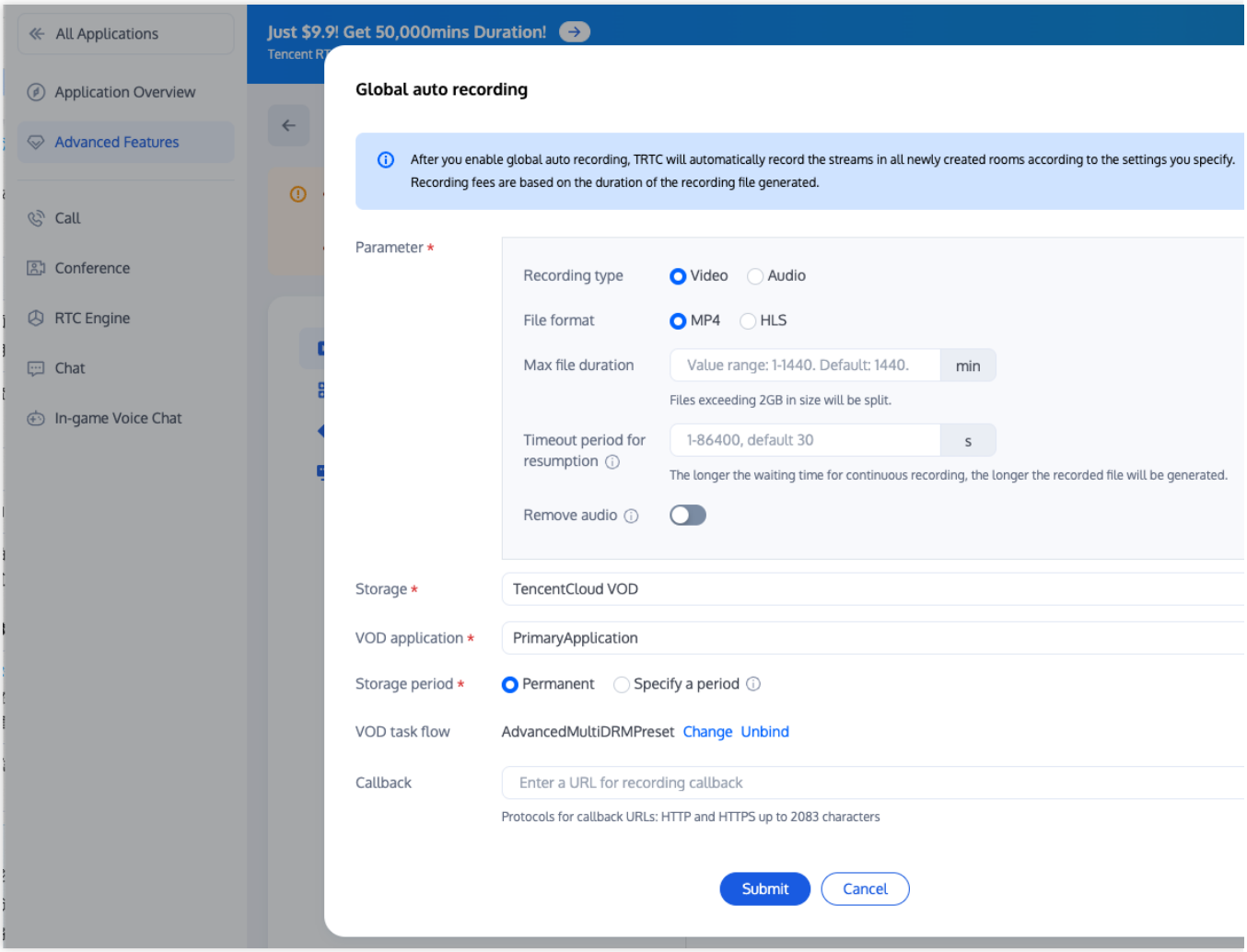
生效后（生效等待5-10分钟）TRTC 房间中的主播上行音视频后将触发启动录制任务，房间内主播都退房且超过设置的等待续录时间后将触发停止录制任务。



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）。

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 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 腾讯云云点播 VOD 、 腾讯云对象存储 COS 和AWS S3存储 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&AWS：需要完成对应存储桶的配置，请确保存储桶具备可写权限。
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）。

手动录制流程

1. 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，需要重点关注的参数如下：

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。

录制的模式（RecordMode）

单流录制，分别录制房间中的您所订阅的主播的音频和视频文件并将录制文件（M3U8/TS）上传至云存储；
混流录制，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并将录制文件[M3U8/TS]上传至云存储；

录制用户的黑白名单（SubscribeStreamUserIds）

默认情况下，云端录制会订阅房间内所有的媒体流（最多25路），您也可以通过该参数指定订阅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。

云存储参数（StorageParams）

通过指定云存储参数，我们会为您把录制后的文件上传到您所开通的指定云存储/云点播服务，请关注云存储/云点播空间的参数的有效性和保持非欠费状态。这里需要注意的是录制文件名称是有固定的规则。

录制文件名命名规则

单流录制M3U8文件名规则：
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8
单流录制TS文件名规则为：
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UTC>.ts
单流录制mp4文件名规则为：
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Index>.mp4
混流录制M3U8文件名规则：
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>.m3u8
混流录制TS文件名规则：
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<UTC>.ts
混流录制mp4文件名规则为：
<Prefix>/<TaskId>/<SdkAppId>_<RoomId>_<Index>.mp4
高可用拉起后的文件名规则
云录制服务在机房故障的时候会通过高可用的方案对录制任务进行恢复，这种情况下为了不覆盖原有的录制文件，拉起后会加上一个前缀ha<1/2/3>，表示发生高可用的次数。一个录制任务最大允许拉起的次数为3次。
单流录制M3U8 文件名规则：
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>.m3u8
单流录制TS文件名规则为：
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>__UserId_s_<UserId>__UserId_e_<MediaId>_<Type>_<UTC>.ts
混流录制M3U8 文件名规则：
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>.m3u8
混流录制TS文件名规则：
<Prefix>/<TaskId>/ha<1/2/3>_<SdkAppId>_<RoomId>_<UTC>.ts

字段含义说明：

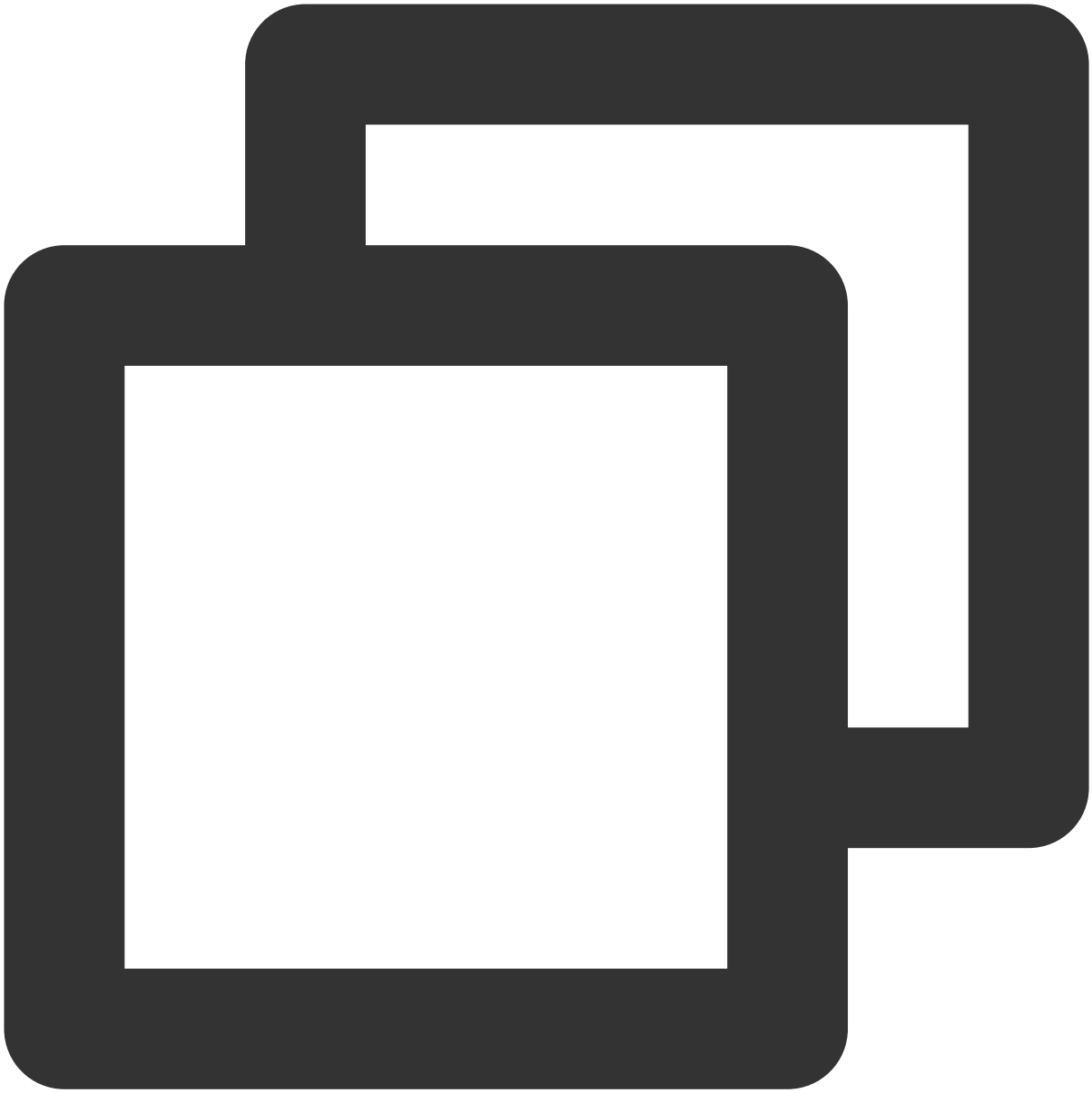
<Prefix>: 录制参数中设置的文件名前缀，如果没有设置那么就不存在；
<TaskId>: 录制的任务ID，全局唯一，启动录制后返回参数中有携带；
<SdkAppId>: 录制任务的SdkAppId；
<RoomId>: 录制的房间号；
<UserId>: 特殊的base64处理后的录制流的用户ID，见下面注意事项；
<MediaId>: 主辅流标识，main或者aux；
<Type>: 录制流的类型，audio或者video；
<UTC>: 该文件开始的UTC录制时间，时区UTC+0，由年、月、日、小时、分钟、秒和毫秒组成；
<Index>: 如果没有触发mp4切片逻辑（大小超过2GB或时长超过24小时）则无该字段，否则为切片的索引号，从1开始递增;
ha<1/2/3>：高可用的拉起的前缀，比如第一次拉起会变成<Prefix>/<TaskId>/ha1_<SdkAppId>_<RoomId>.m3u8。

注意：

如果这里<RoomId>如果是字符串房间ID，我们会对房间ID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 '.' ；
录制流的<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 '.' 。

录制开始的时间的获取

录制开始的时间定义为第一次收到主播的音视频数据，并启动录制第一个文件的服务器unix时间。
您可以通过以下三个方法获取到录制开始的时间戳：
查询录制文件接口（DescribeCloudRecording）中的BeginTimeStamp字段
比如下面这个查询接口返回的信息看到BeginTimeStamp为1622186279144ms：

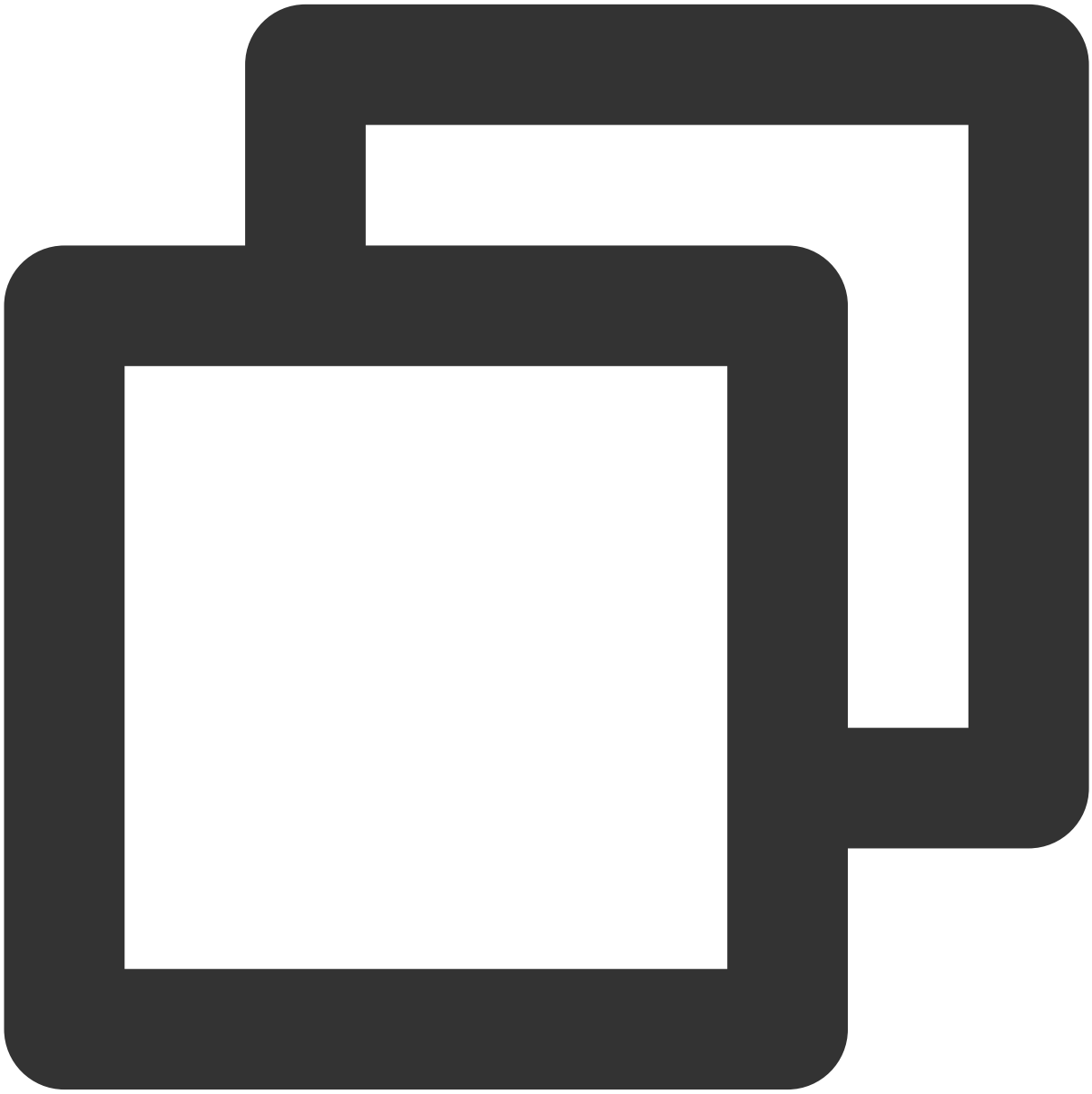


```
{
  "Response": {
    "Status": "xx",
    "StorageFileList": [
      {
        "TrackType": "xx",
        "BeginTimeStamp": 1622186279144,
        "UserId": "xx",
        "FileName": "xx"
      }
    ],
```

```
    "RequestId": "xx",
    "TaskId": "xx"
  }
}
```

读取M3U8文件中对应的标签项（#EXT-X-TRTC-START-REC-TIME）

比如下面这个M3U8文件表示起始录制的unix时间戳为1622425551884ms：



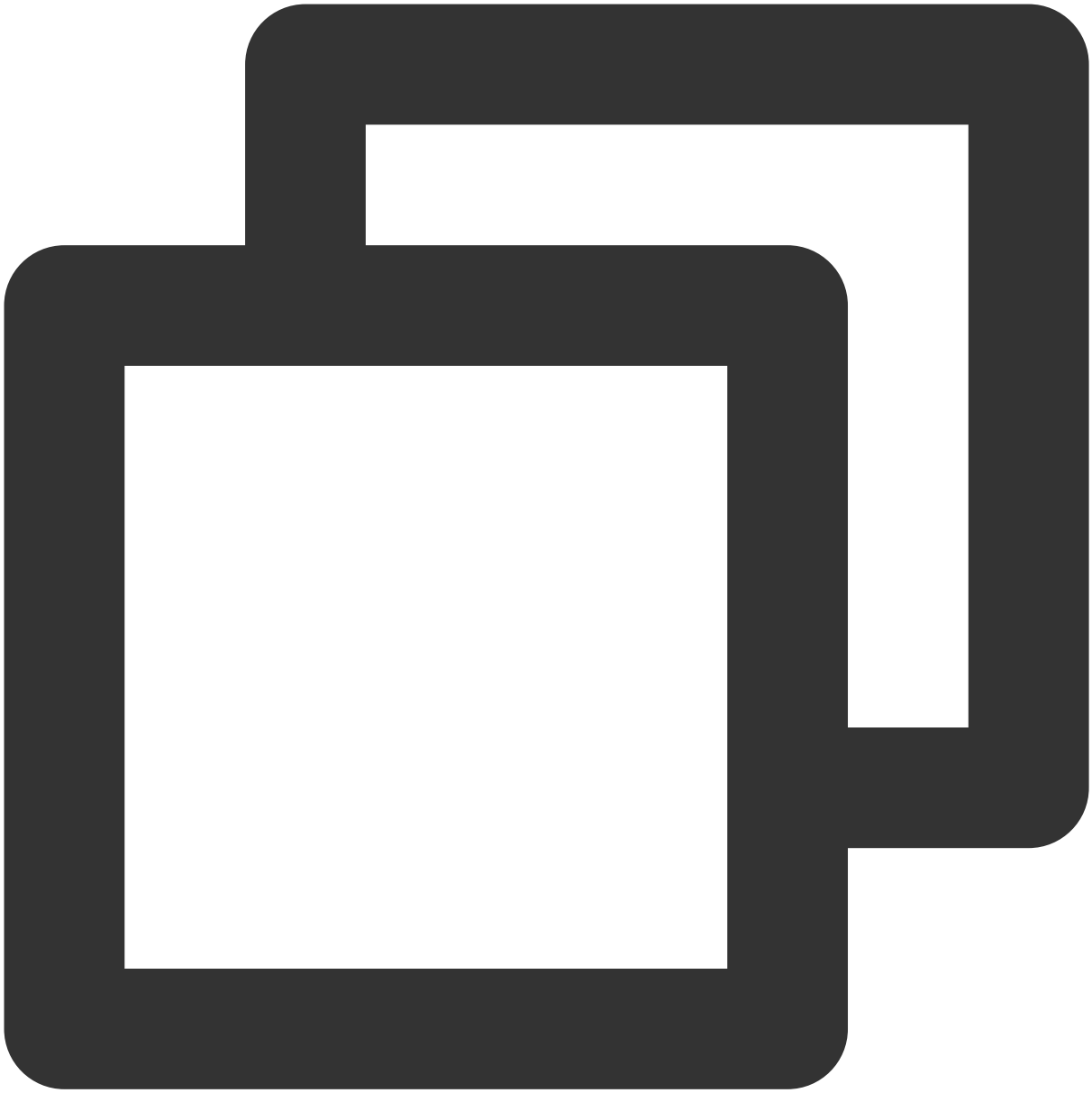
```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
```

```
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:70
#EXT-X-TRTC-START-REC-TIME:1622425551884
#EXT-X-TRTC-VIDEO-METADATA:WIDTH:1920 HEIGHT:1080
#EXTINF:12.074
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094551825.ts
#EXTINF:11.901
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094603825.ts
#EXTINF:12.076
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094615764.ts
#EXT-X-ENDLIST
```

监听录制回调事件

通过订阅回调，在事件类型307中的BeginTimeStamp字段您可以获取到录制文件对应的录制起始时间戳

比如下面这个回调事件，您可以读取到这个文件的录制开始的unix时间戳是1622186279144ms：



```
{
        "EventGroupId": 3,
        "EventType":    307,
        "CallbackTs":   1622186289148,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186289",
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "FileName":     "xx.m3u8",
```

```
                        "UserId":       "xx",
                        "TrackType":    "audio",
                        "BeginTimeStamp":       1622186279144
                }
        }
}
```

混流录制的水印参数 （MixWatermark）

我们支持在混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。

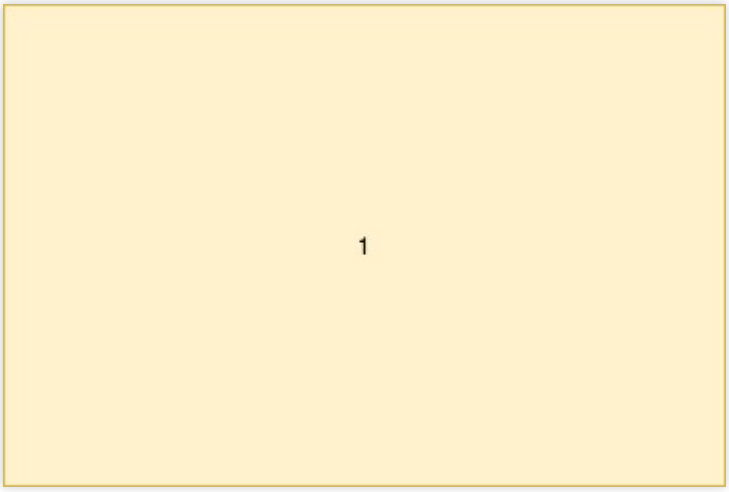
字段名	解释
Top	水印相对左上角的垂直位移
Left	水印相对左上角的水平位移
Width	水印显示的宽度
Height	水印显示的高度
url	水印文件的存储url

混流录制的布局模式参数（MixLayoutMode）

我们支持九宫格布局（默认），悬浮布局，屏幕分享布局和自定义布局四种布局。

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。
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;
九宫格布局随着订阅的子画面增加按照下图进行变化：



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 x 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 x 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。

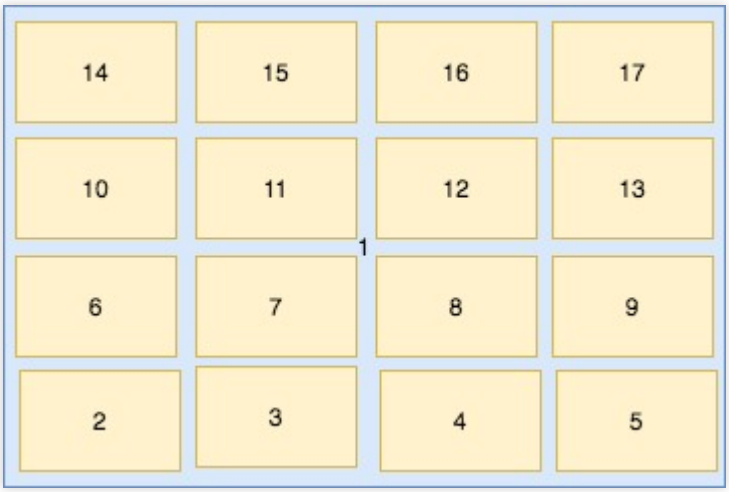
子画面小于等于17个时：

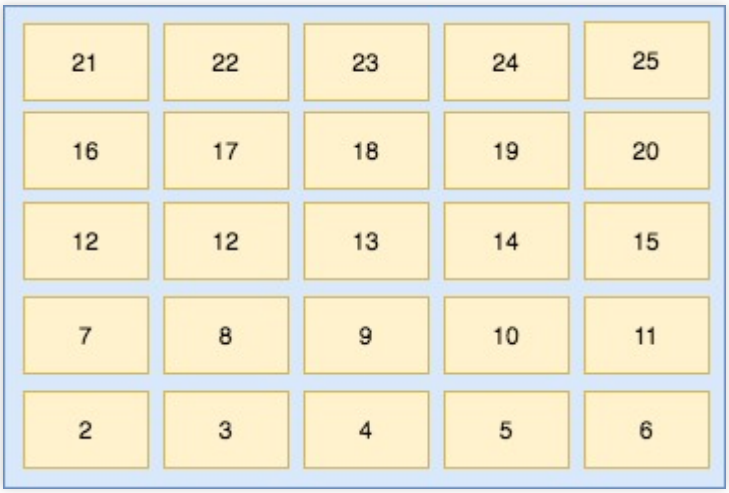
- 每个小画面的宽和高分别为整个画布宽和高的 0.235；
- 相邻小画面的左右和上下间距分别为整个画布宽和高的 0.012；
- 小画面距离画布的水平和垂直边距也分别为整个画布宽和高的 0.012；

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；

悬浮布局随着订阅的子画面增加按照下图进行变化：





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。

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；

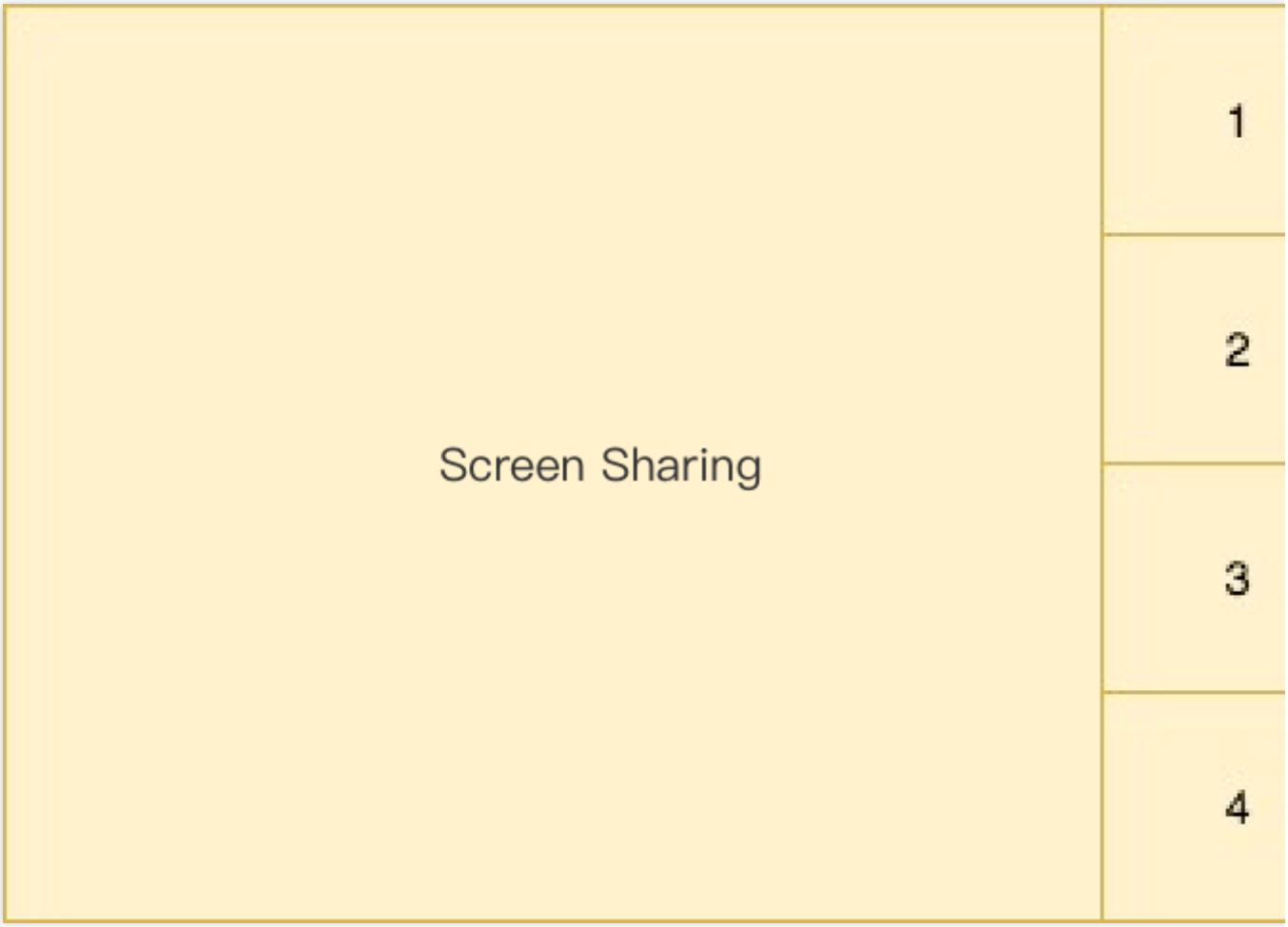
左侧大画面的宽为整个画布宽的4/5,左侧大画面的高为整个画布高;

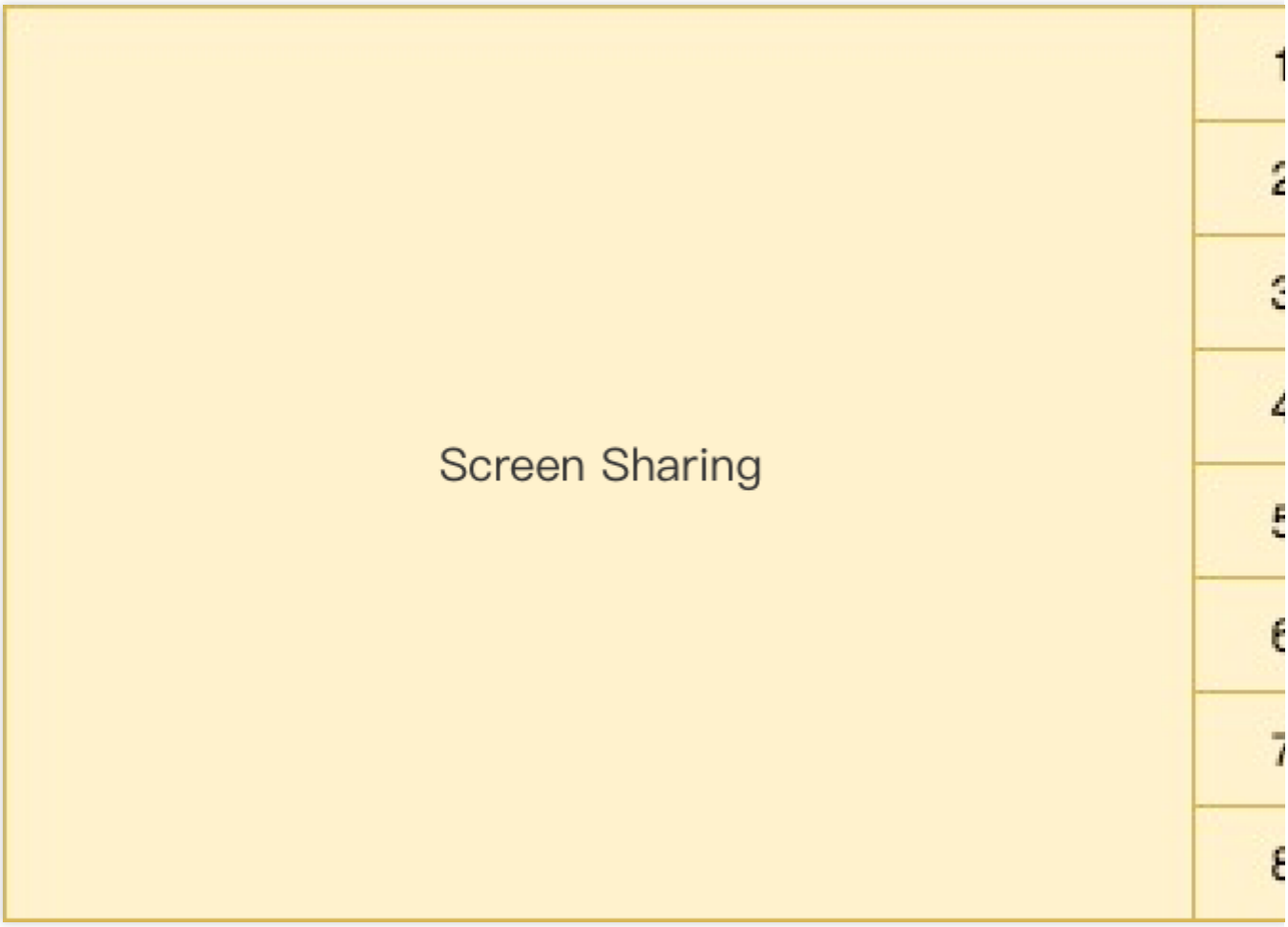
子画面大于17个时:

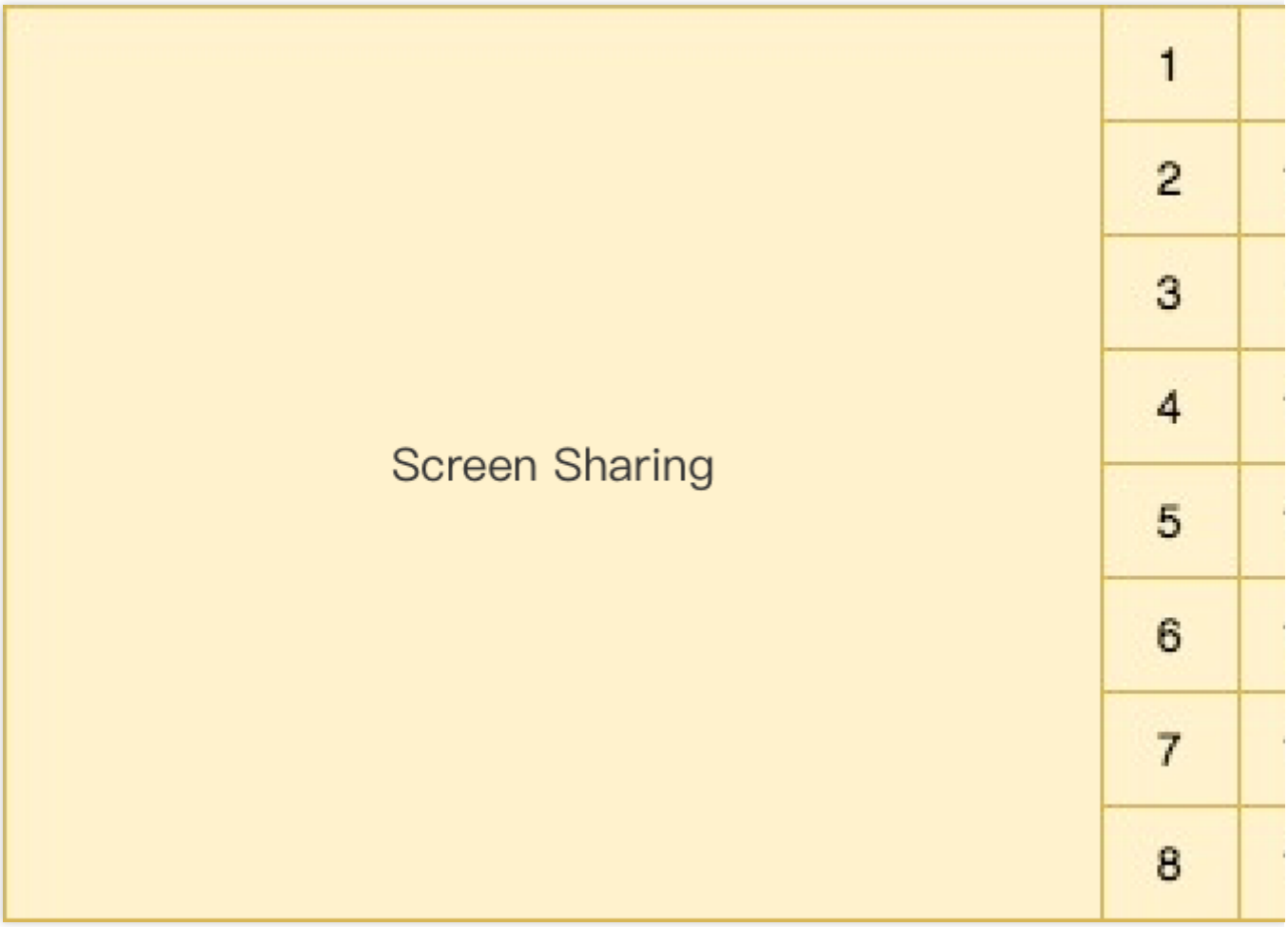
右（下）侧小画面的宽为整个画布宽的1/10,右（下）侧小画面的高为整个画布高的1/8；

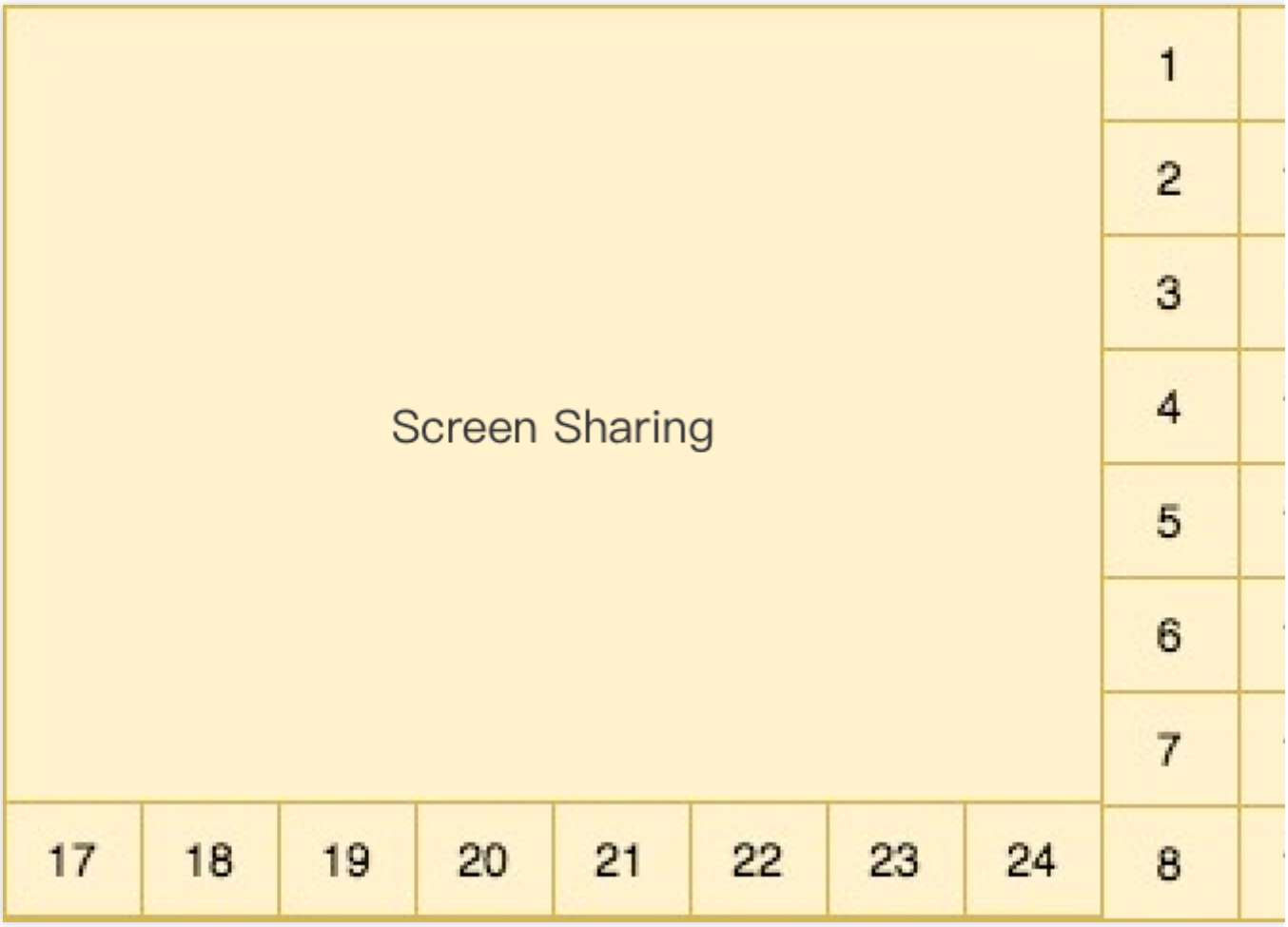
左侧大画面的宽为整个画布宽的4/5,左侧大画面的高为整个画布高的7/8;

屏幕分享布局随着订阅的子画面增加按照下图进行变化：






Screen Sharing
1
2
3
4
5
6
7
8



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。

2. 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。
录制的文件列表（StorageFile）会包含本次录制的所有M3U8索引文件和录制的起始Unix时间戳信息。注意，如果是上传云点播任务，该接口返回的StorageFile为空。

3. 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（单流和混流录制有效），录制的模板参数MixLayoutParams（混流录制有效），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。

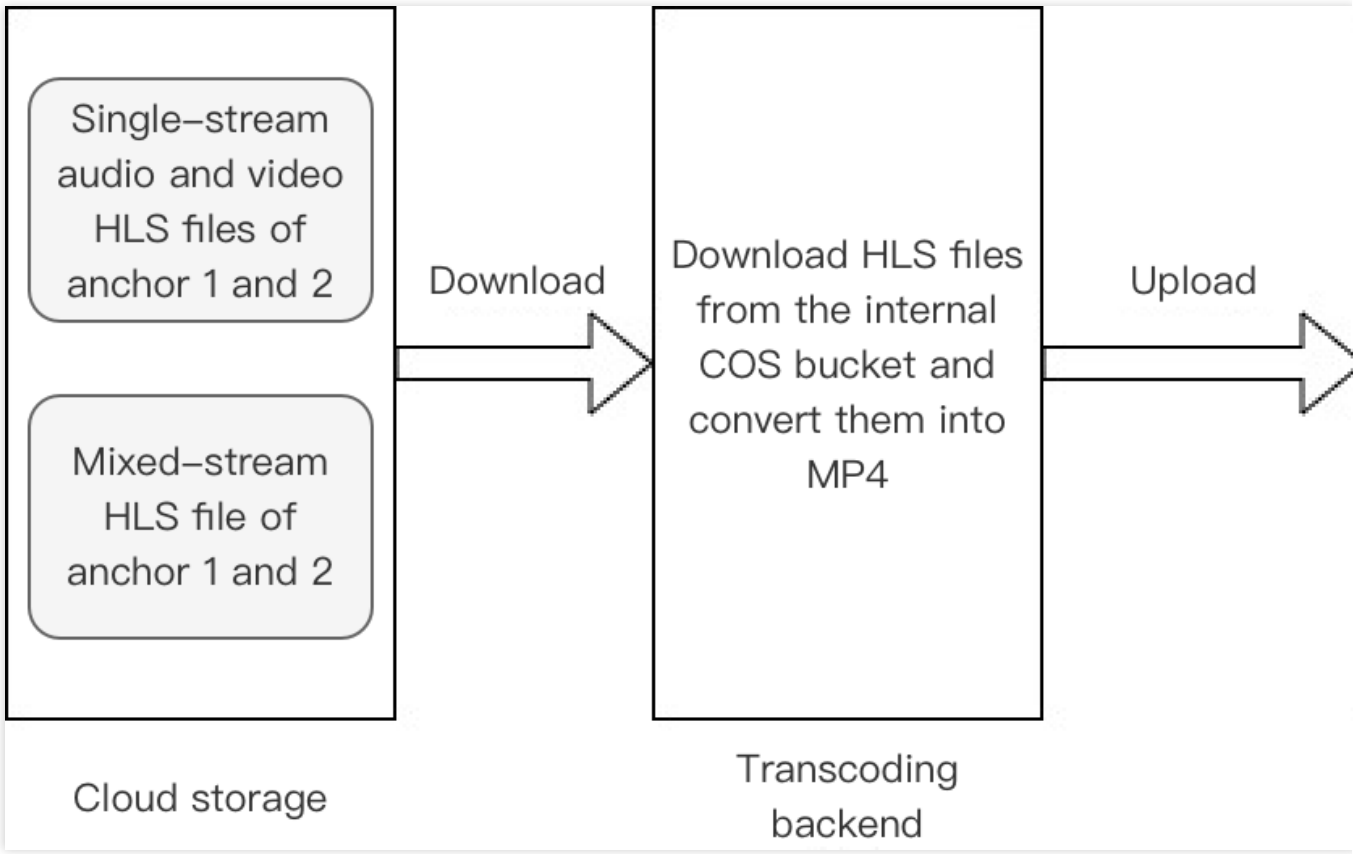
4. 停止录制（DeleteCloudRecording）

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。需要注意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。建议业务在录制结束的时候调用此接口结束录制任务。

高级功能

录制mp4文件

如果希望录制的输出文件格式是mp4格式，可以在启动录制的参数(OutputFormat)指定录制文件输出格式为hls+mp4。默认依然会录制hls文件，在hls录制完成后会立即触发转mp4任务，从hls所在的cos内拉流进行mp4封装，再上传到客户的cos中。
注意这里需要提供COS的文件下拉权限，否则转mp4任务会因为下拉hls文件失败而终止。
mp4文件将在以下情况下强制分片。
1. 录制时长超过24小时;
2. 单个mp4文件大小达到2GB;
具体的流程如图所示

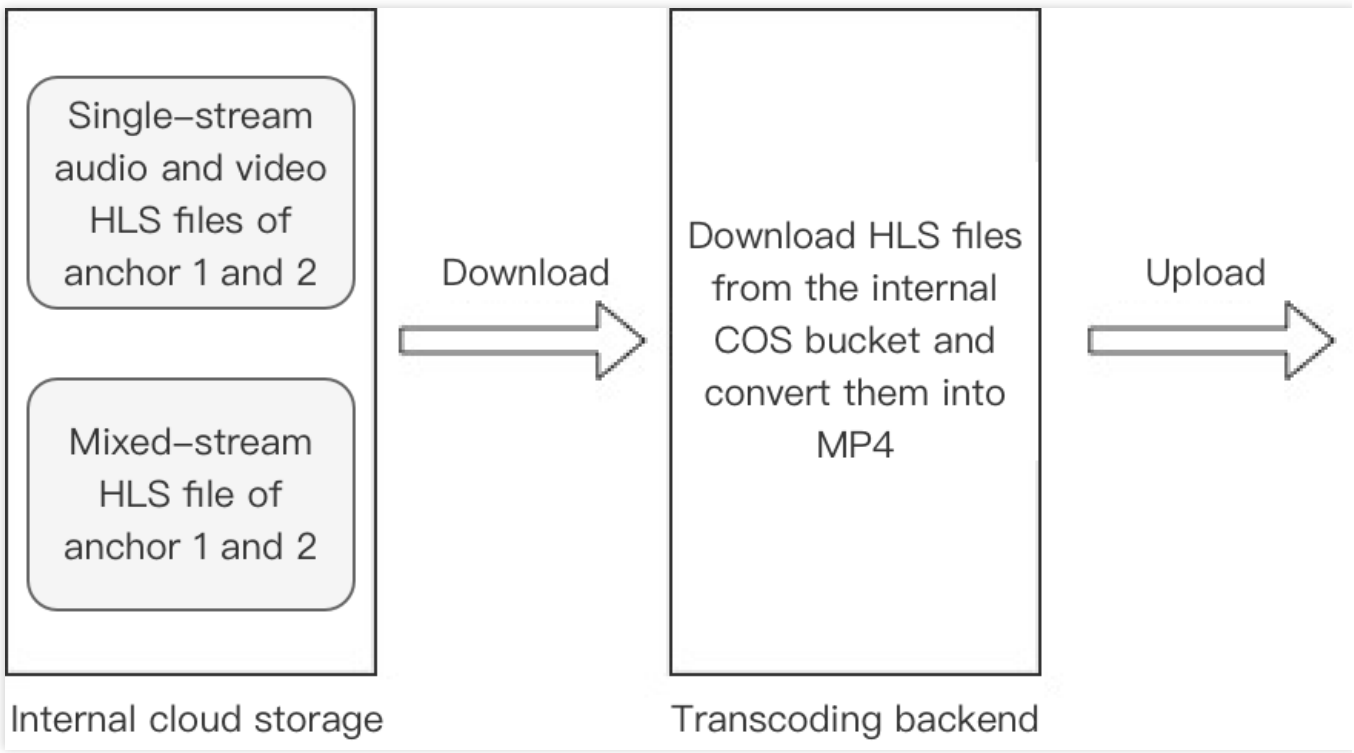


录制上传点播

如果希望将录制的输出文件上传至点播平台，可以在启动录制的存储参数(StorageParams)中指定CloudVod成员参数。录制后台会在录制结束后将录制的mp4文件通过您指定的方式上传到点播平台，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；如果录制模式为混流录制模式，只有一个混流后媒体的播放地址。上传点播任务有以下几个注意事项。

1. 存储参数中的CloudVod和CloudStorage只能同时指定一个，否则发起录制将失败；
2. 上传点播任务的过程中使用DescribeCloudRecording查询到的任务状态，不会携带录制文件的信息；

具体的流程如图所示，这里的云存储为录制内部云存储，客户不用填写相关参数。



单流文件合并脚本

我们提供了合并单流音视频文件的脚本，用于把单流录制的音视频文件合并成MP4文件。

说明：

如果两个切片之间的时间间隔超过15秒，间隔时间内没有任何音频/视频信息（如果禁用了辅流，则会忽略辅流信息），我们把这两个切片看作两个不同的分段。其中录制时间较早的切片看作前一个分段的结束切片，录制时间较晚的切片看作后一个分段的开始切片。

分段模式（-m 0）

此模式下，脚本将每个UserId下的录制文件按分段合并，一个UserId下的分段被独立合并为一个个文件。

合并模式（-m 1）

把同一个UserId下的所有分段合并为一个音视频文件。可利用 -s 选项选择是否填充各个分段之间的间隔。

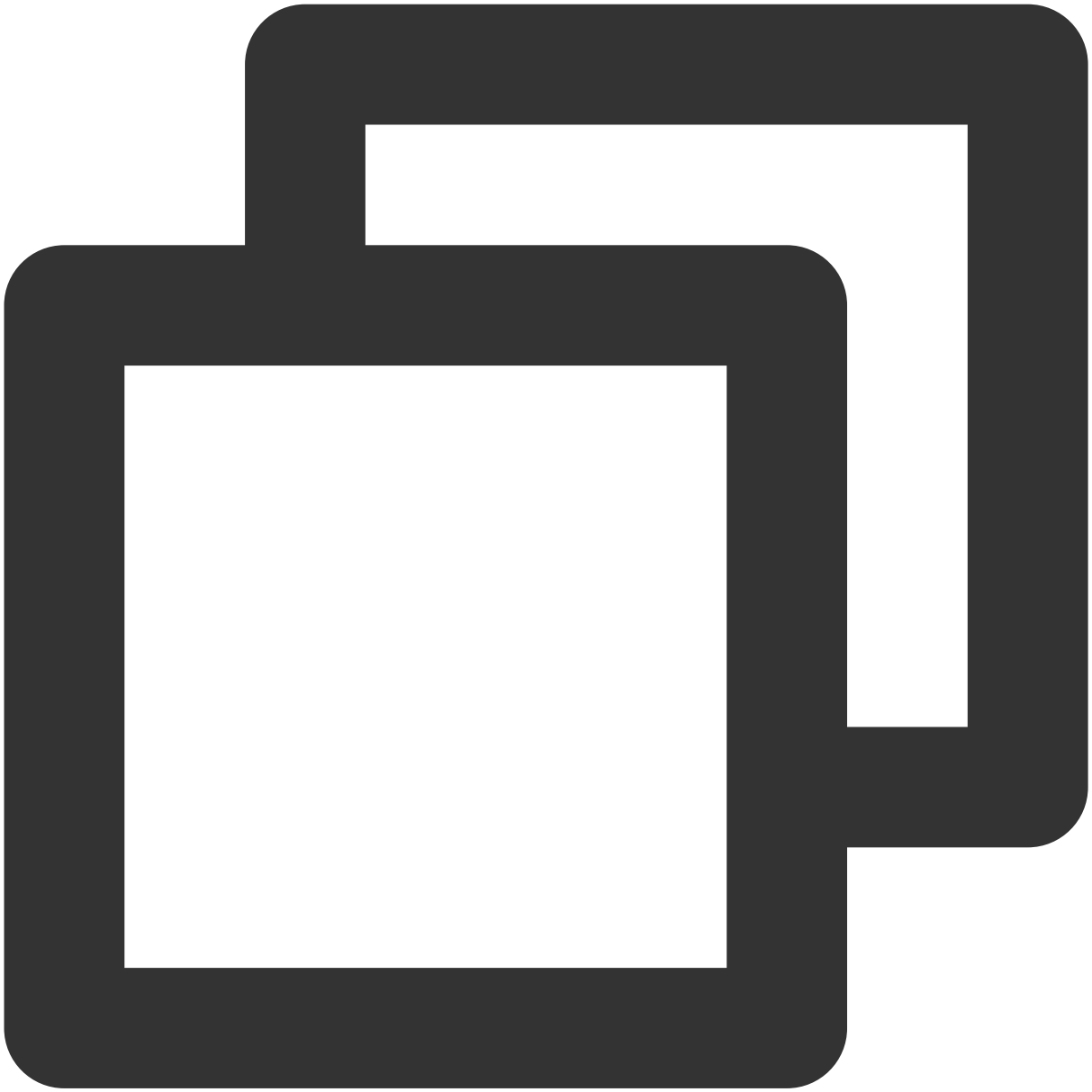
依赖环境

Python3

Centos：sudo yum install python3

Ubuntu：sudo apt-get install python3

Python3 依赖包



```
sortedcontainers：pip3 install sortedcontainers
```

使用方法

1.运行合并脚本：python3 TRTC_Merge.py [option]

2.会在录制文件目录下生成合并后的mp4文件

例如：python3 TRTC_Merge.py -f /xxx/file -m 0

可选参数和对应功能见下表:

参数	功能
-f	指定待合并文件的存储路径。如果有多个UserId的录制文件，脚本会分别进行合并操作。
-m	0：分段模式(默认设置)，此模式下，脚本将每个UserId下的录制文件按分段合并，每个UserId有可能产生多个文件。 1：合并模式，一个UserId下的所有音视频文件合并为一个文件。
-s	保存模式。如果设置了该参数，则合并模式下的分段之间的空白部分被删除，文件的实际时常小于物理时常。
-a	0: 主流合并(默认设置)，同一个UserId的主流和音频做合并，辅流不会和音频做合并。 1: 自动合并，如果主流存在则主流和音频合并，如果主流不存在辅流存在则辅流和音频做合并。 2: 辅流合并，同一个UserId的辅流和音频做合并，主流不会和音频做合并。
-p	指定输出视频的fps。默认为15 fps，有效范围5-120 fps，低于5 fps计为5 fps，高于120 fps计为120 fps。
-r	指定输出视频的分辨率。如 -r 640 360，表示输出视频的宽为640，高为360。

文件名规则

音视频文件名规则：UserId_timestamp_av.mp4

纯音频文件名规则：UserId_timestamp.m4a

说明：

这里的UserId为录制流的用户ID特殊的base64值，详见启动录制中文件名命名规则相关说明。timestamp为每个分段的首个ts切片启动录制的时间。

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。

和事件回调消息格式

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}。

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
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 录制模块启动
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	云端录制 录制模块退出
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	云端录制 上传模块启动
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
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	云端录制 上传结束
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文件(切出第一个ts分片) 生成后回调
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR	308	云端录制 上传模块发生错

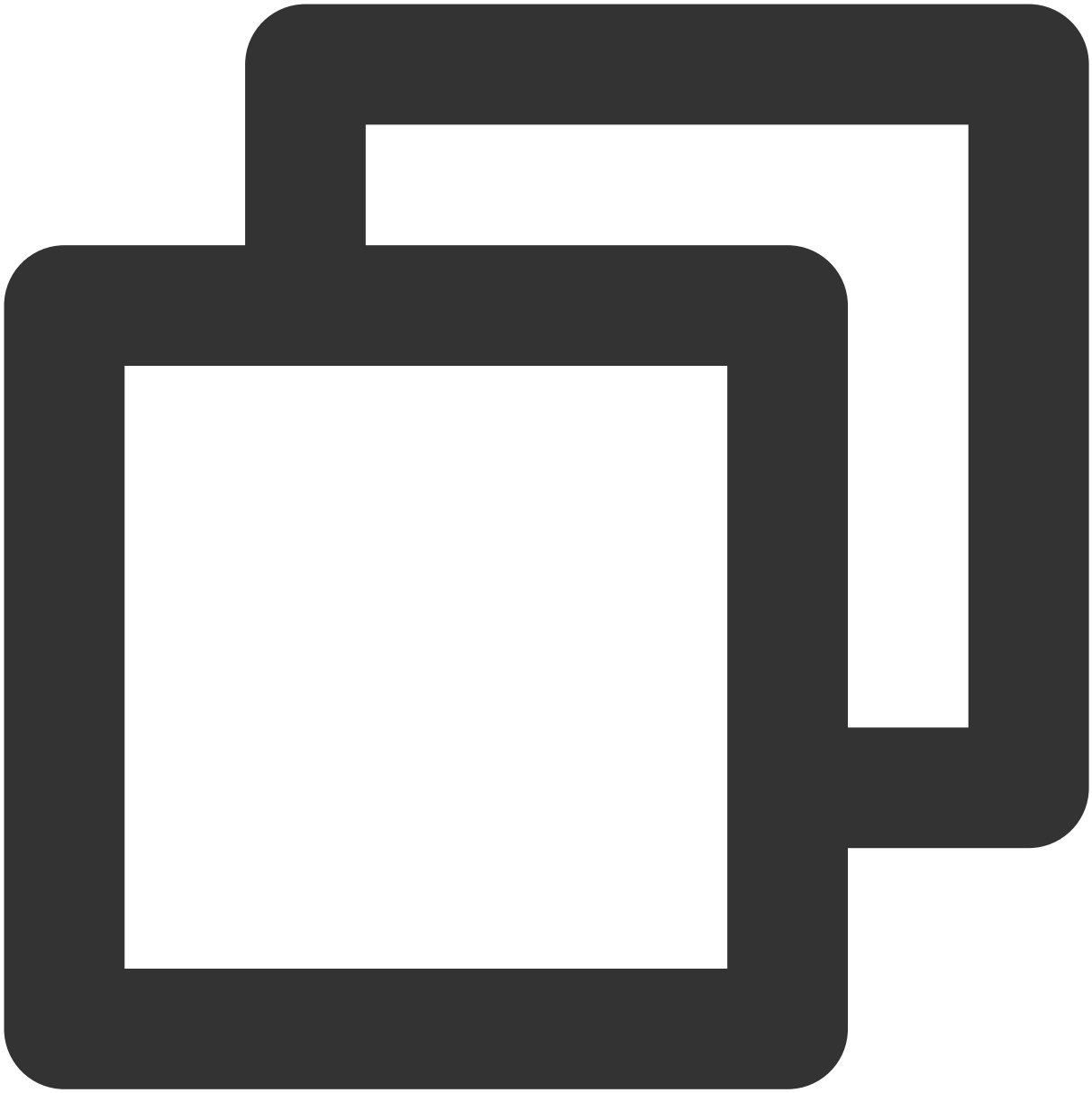
		误
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	云端录制 下载解码图片文件发生错误
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	云端录制 mp4录制任务结束，回调包含录制的mp4文件名和详细信息
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	云端录制 vod录制任务上传媒体资源完成
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	云端录制 vod录制任务结束

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	时间发生的 Unix 时间戳，单位为秒
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。

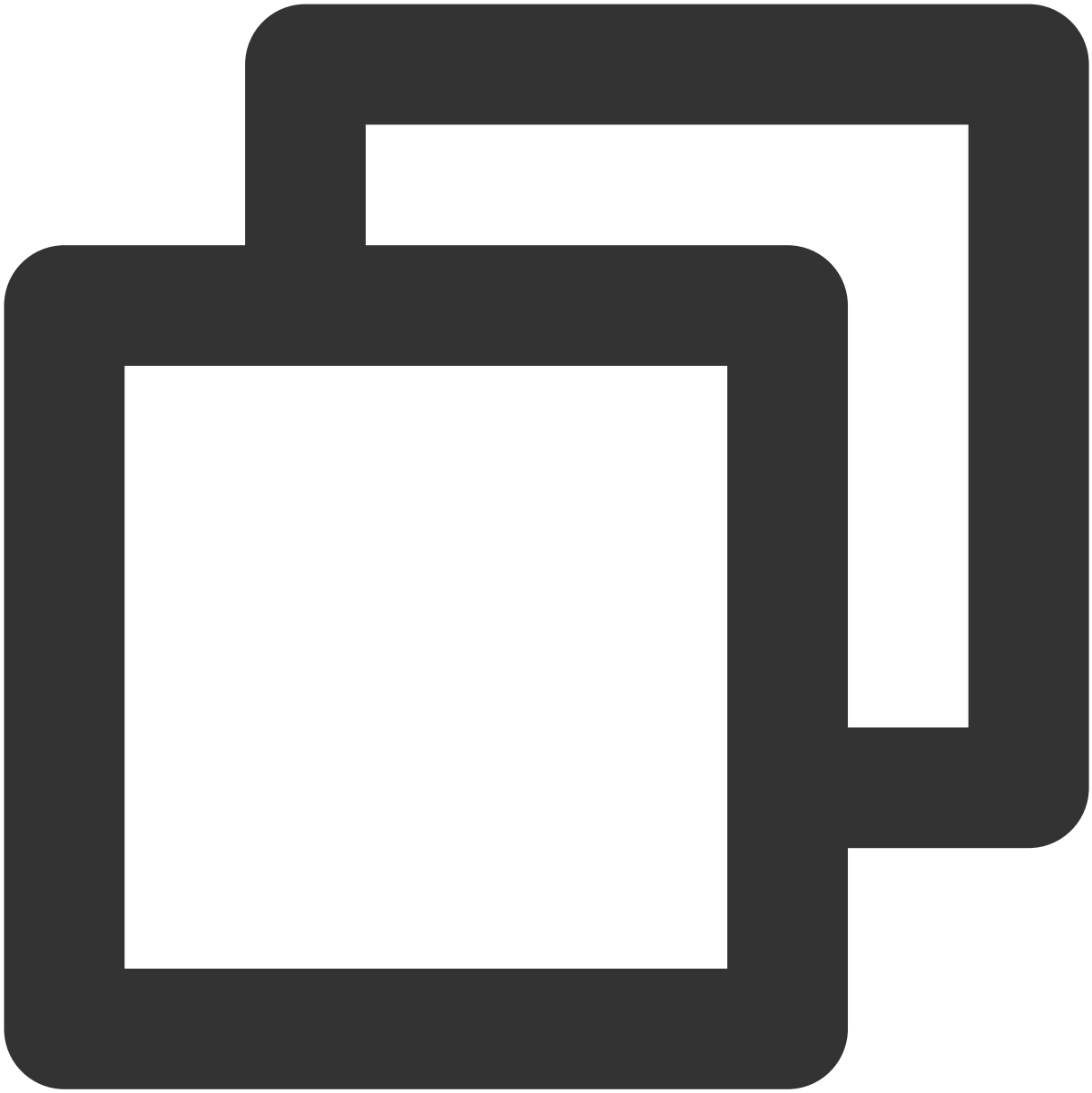


```
{
        "EventGroupId": 3,
        "EventType":    301,
        "CallbackTs":   1622186275913,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186275",
                "UserId":       "xx",
                "TaskId":       "xx",
                "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；

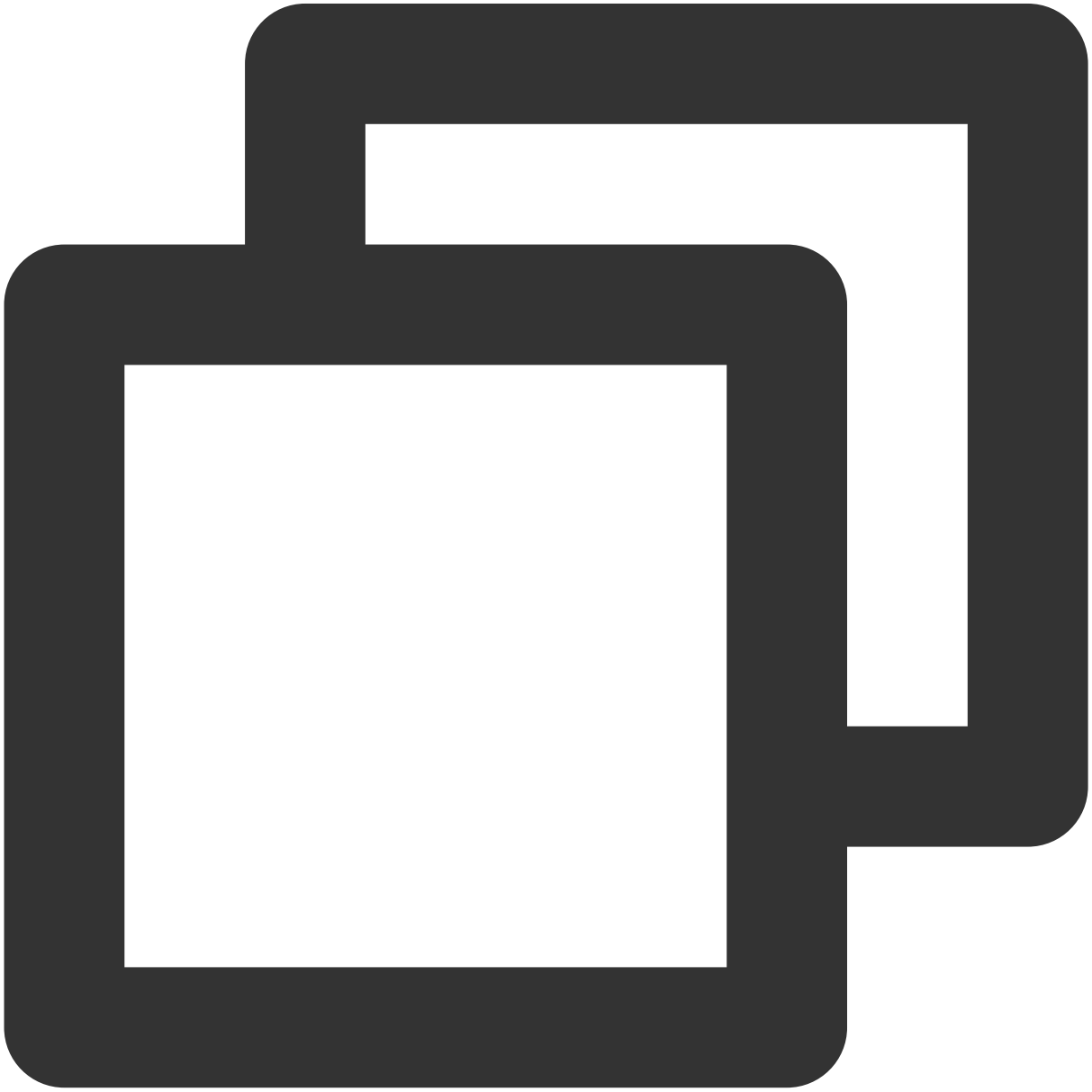


```
{
        "EventGroupId": 3,
        "EventType":    302,
        "CallbackTs":   1622186354806,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186354",
                "UserId":       "xx",
                "TaskId":       "xx",
             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

```
                    }
                }
            }
```

事件类型为303 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START 时Payload的定义

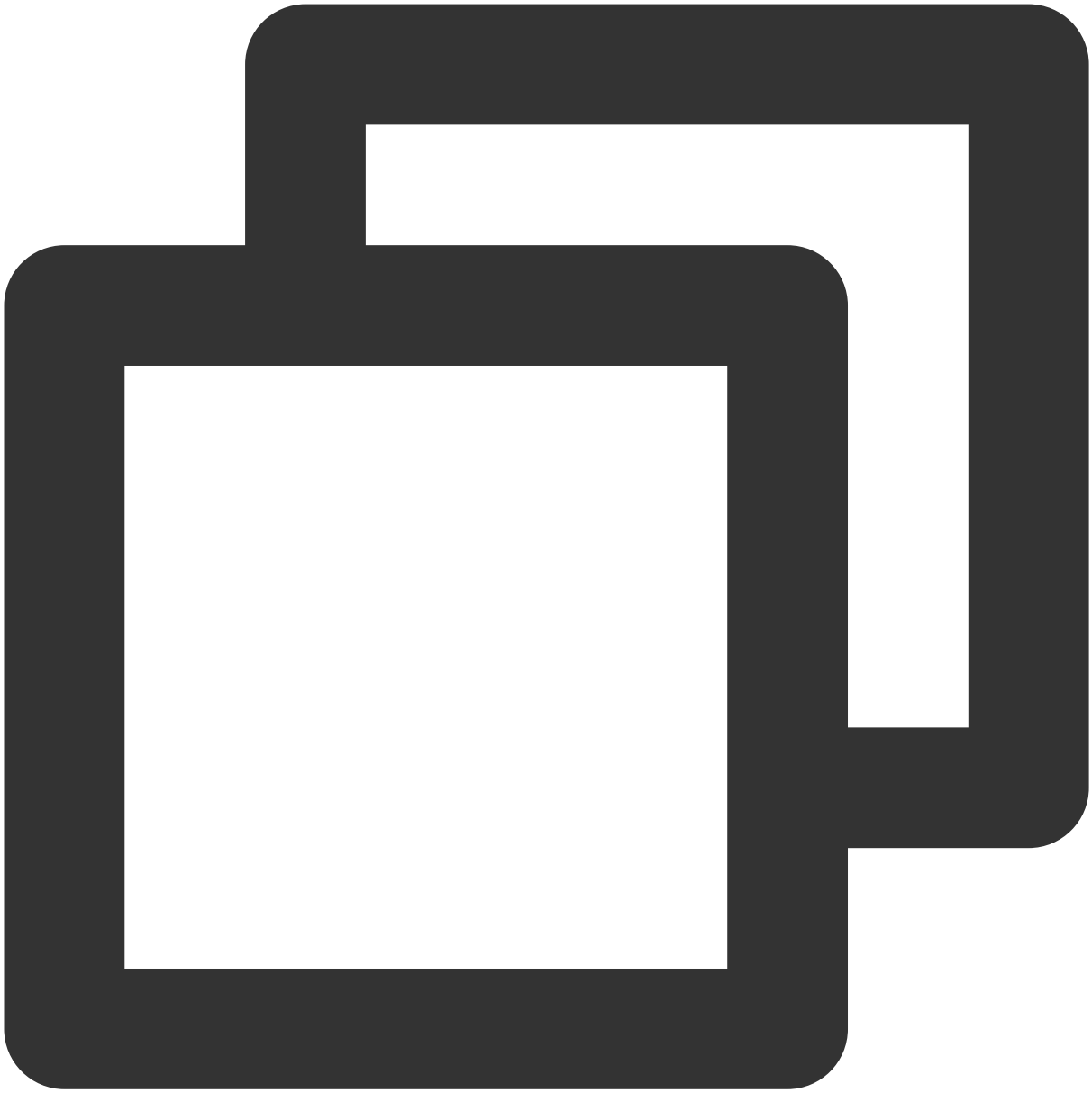
字段名	类型	含义
Status	Number	0：代表上传模块正常启动。 1：代表上传模块初始化失败。



```
{
        "EventGroupId": 3,
        "EventType":    303,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
                }
        }
}
```

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

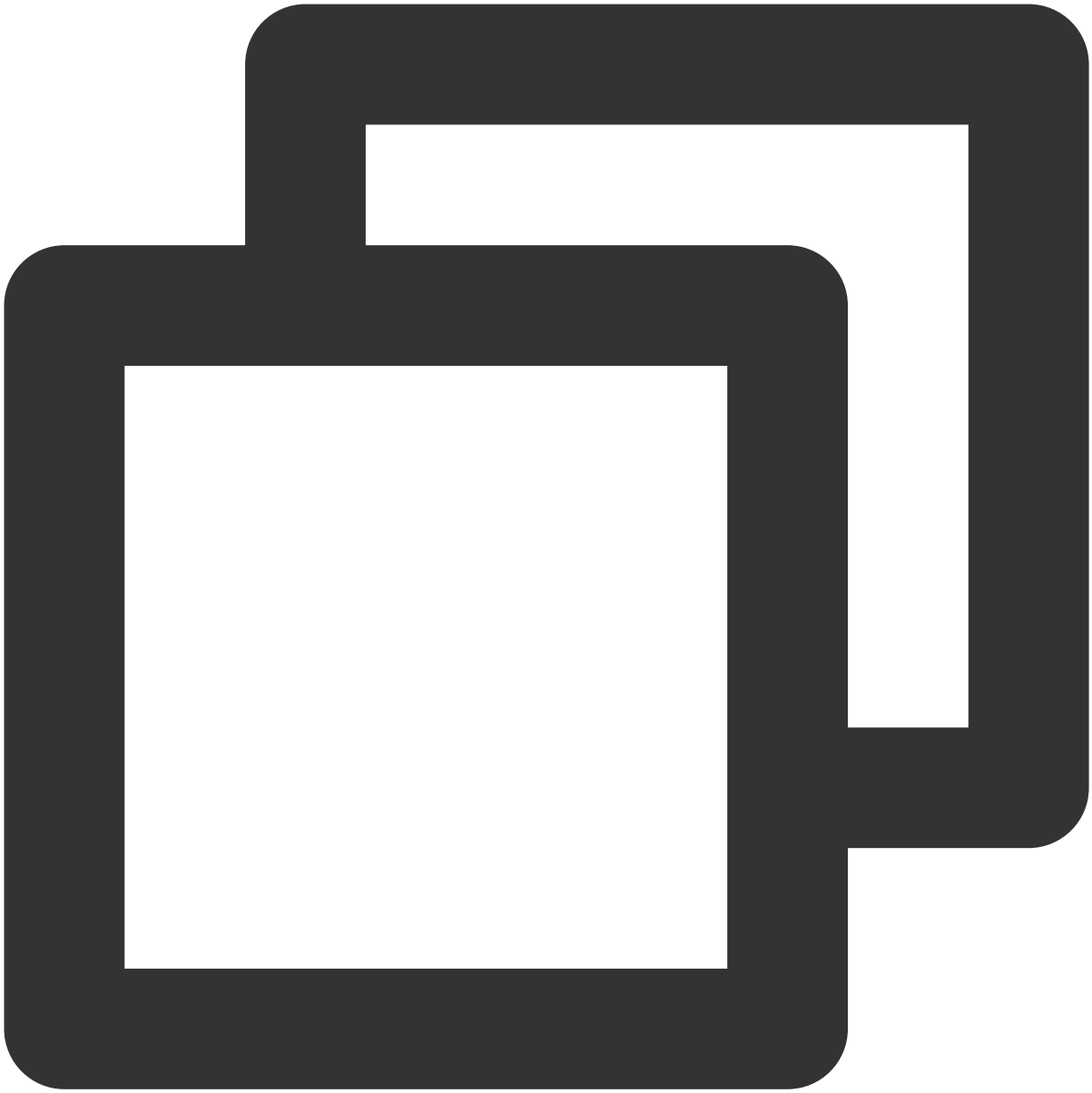


```
{
        "EventGroupId": 3,
        "EventType":    304,
        "CallbackTs":   1622191965350,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "FileList":     "xx.m3u8"
```

```
                    }
                }
            }
```

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

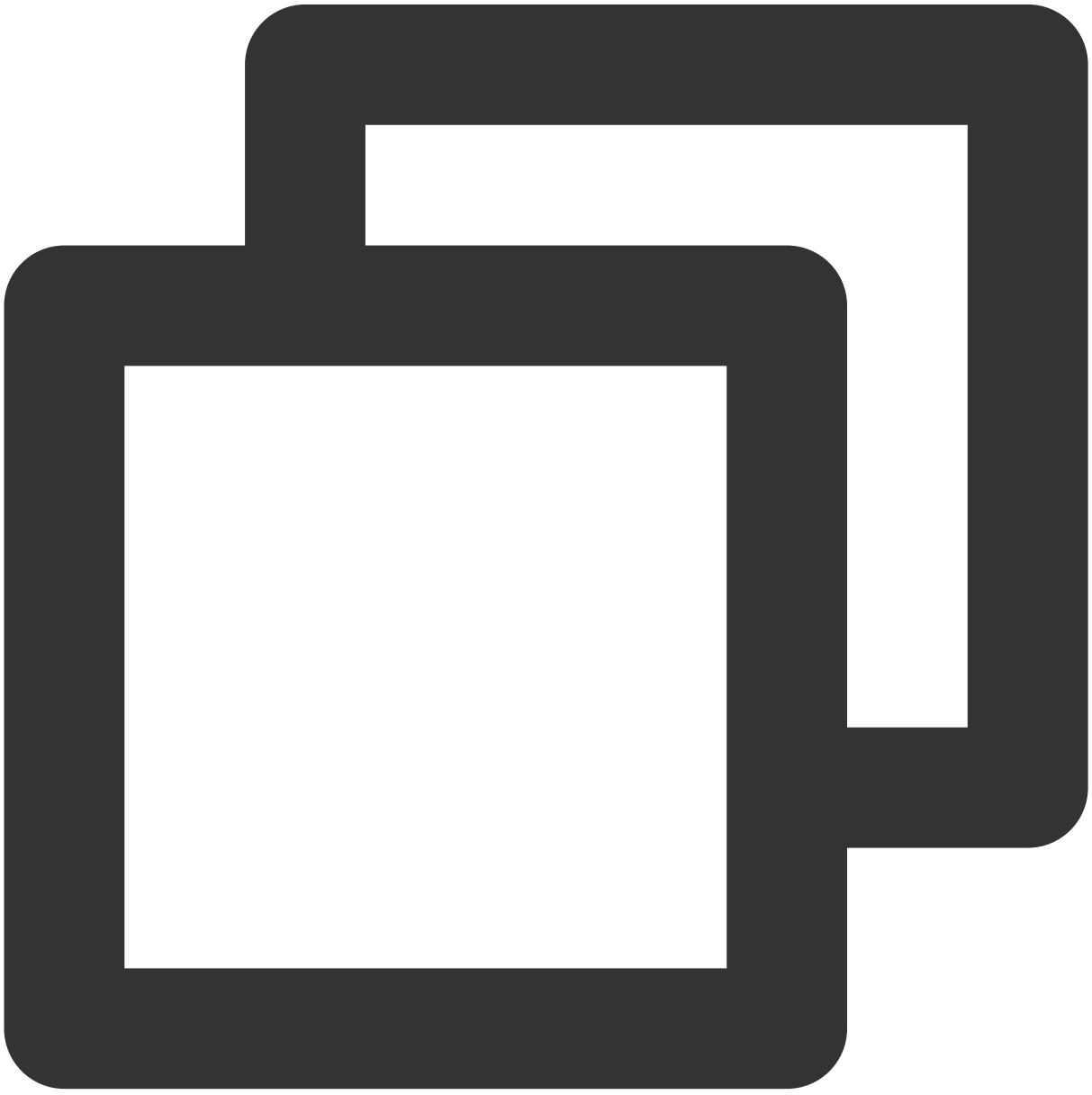


```
{
        "EventGroupId": 3,
        "EventType":    305,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
```

```
                }
            }
}
```

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

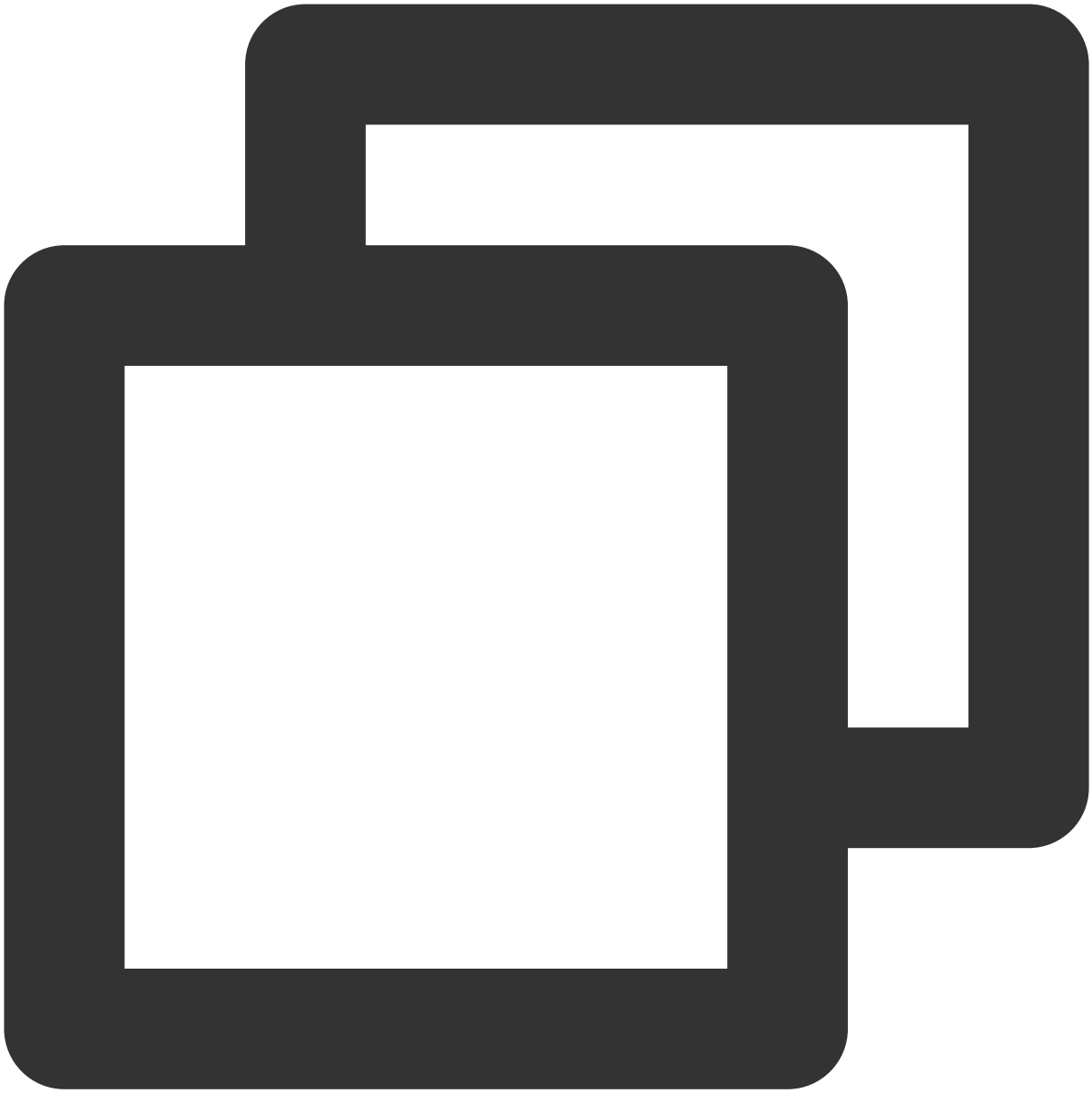


```
{
```

```
        "EventGroupId": 3,
        "EventType":    306,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "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	本录制文件对应的用户ID
TrackType	String	audio/video/audio_video
BeginTimeStamp	Number	录制开始时，服务器Unix时间戳（毫秒）

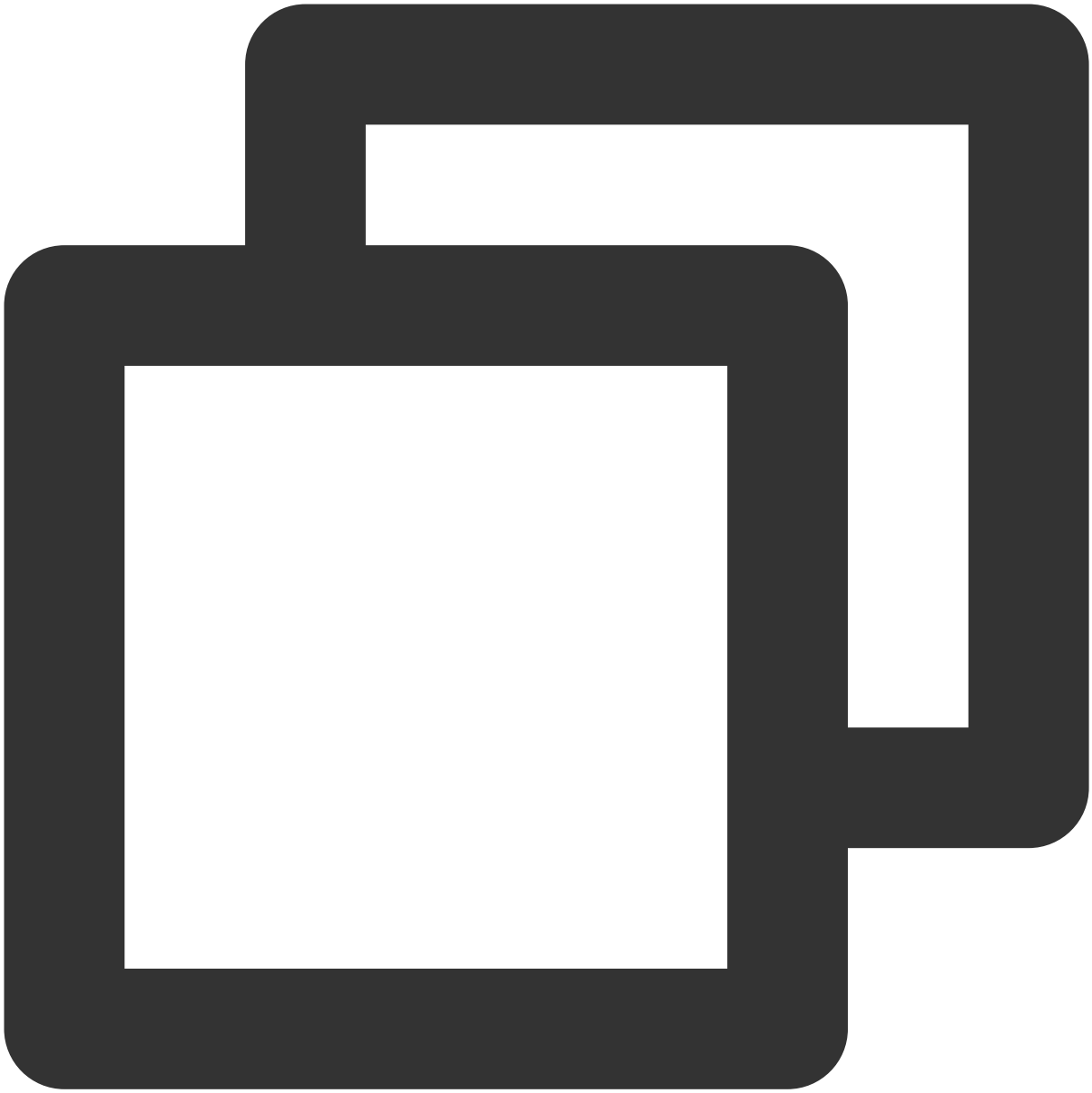


```
{
        "EventGroupId": 3,
        "EventType":    307,
        "CallbackTs":   1622186289148,
        "EventInfo":    {
                "RoomId":       "xx",
                "EventTs":      "1622186289",
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "FileName":     "xx.m3u8",
```

```
                        "UserId":       "xx",
                        "TrackType":    "audio",
                        "BeginTimeStamp":       1622186279144
                }
        }
}
```

事件类型为308 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR 时Payload的定义

字段名	类型	含义
Code	String	第三方云存储的返回code
Message	String	第三方云存储的返回消息内容

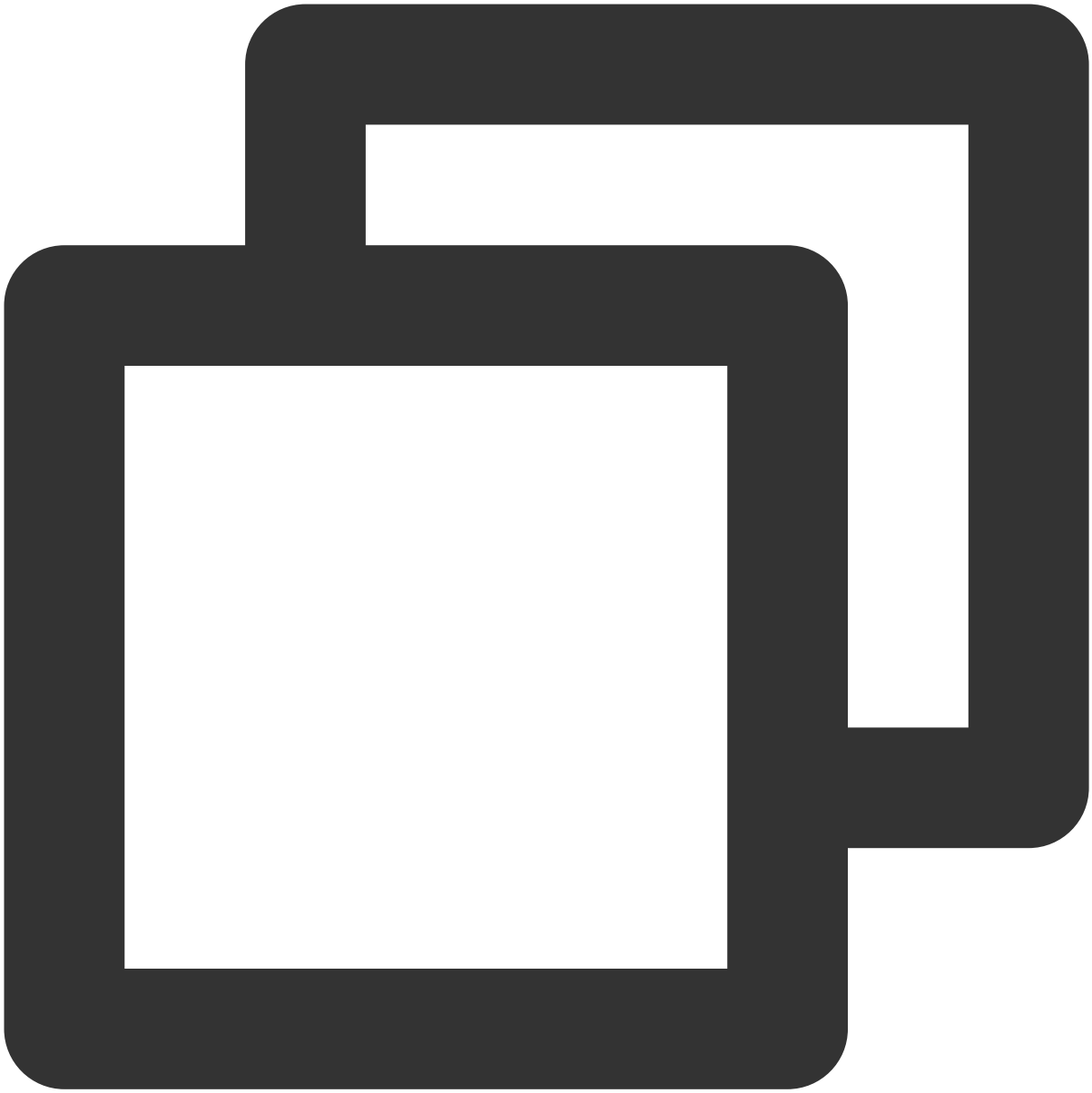


```
{
  "Code": "InvalidParameter",
  "Message": "AccessKey invalid"
}

{
        "EventGroupId": 3,
        "EventType":    308,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
```

```
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Code":         "xx",
                        "Message":      "xx"
                }
        }
}
```

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

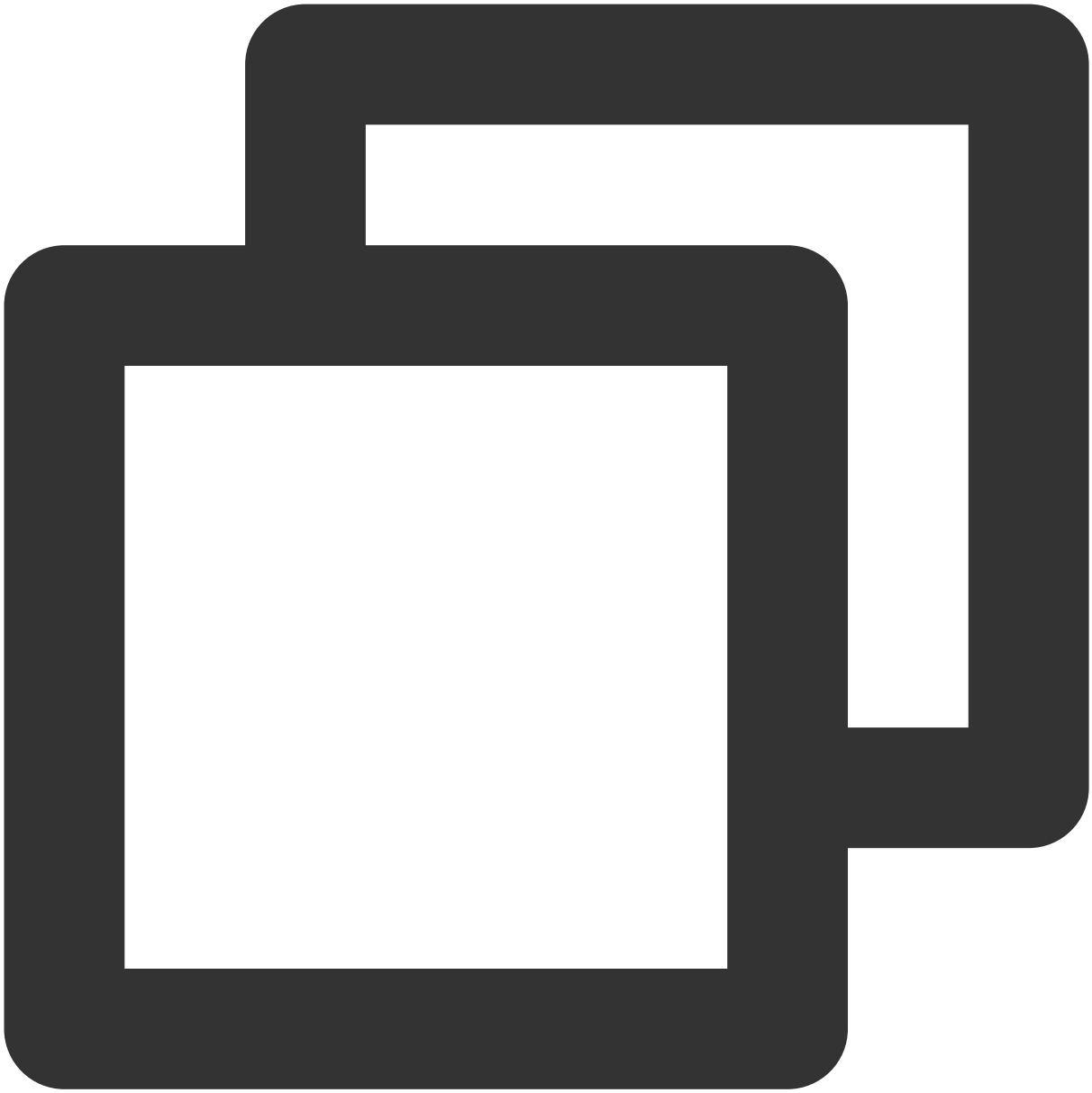


```
{
        "EventGroupId": 3,
        "EventType":    309,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
            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

```
                    }
                }
            }
```

事件类型为310 EVENT_TYPE_CLOUD_RECORDING_MP4_STOP 时Payload的定义

字段名	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/video/audio_video
MediaId	String	main/aux
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）



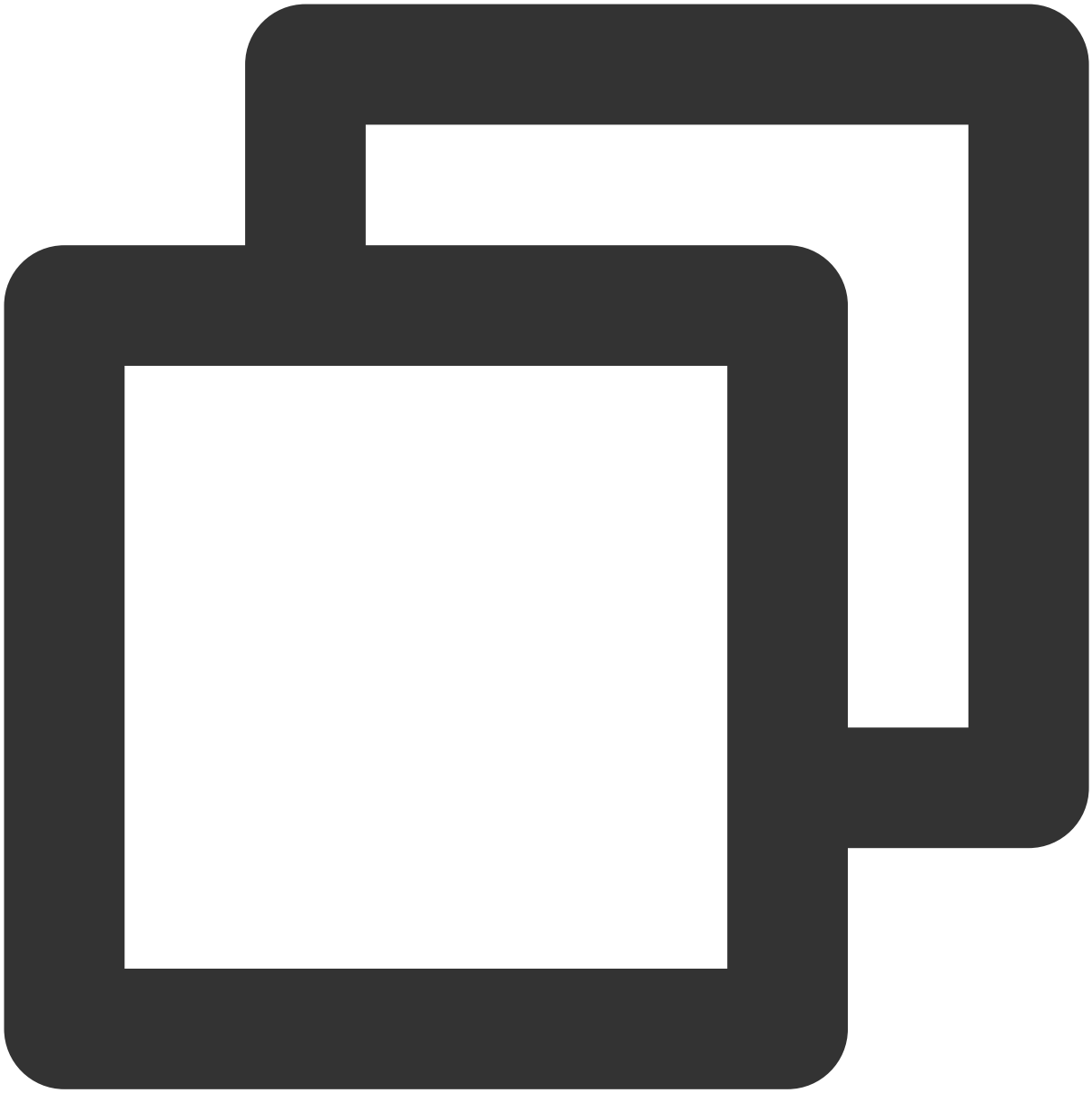
```
{
        "EventGroupId": 3,
        "EventType":    310,
        "CallbackTs":   1622191989674,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191989,
                "UserId":       "xx",
                "TaskId":       "xx",
                "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（当录制模式为混流模式时，此字段为空）
TrackType	String	audio/video/audio_video
MediaId	String	main/aux
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文件名（未上传点播之前）
Errmsg	String	statue不为0时，对应的错误信息

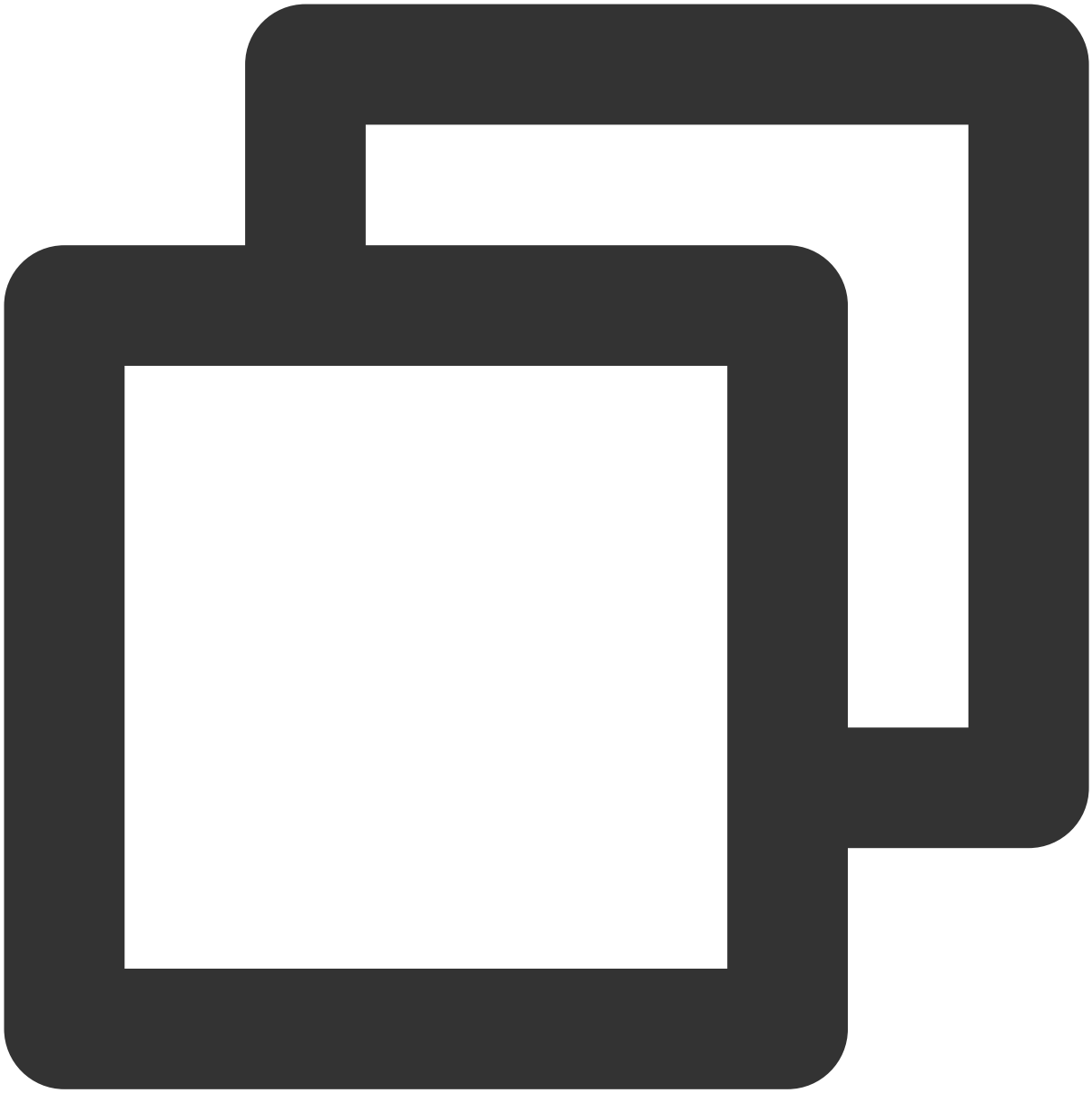
上传成功的回调：



```
{
        "EventGroupId": 3,
        "EventType":    311,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0,
```

```
                        "TencentVod":   {
                            "UserId":       "xx",
                            "TrackType":    "audio_video",
                            "MediaId":      "main",
                            "FileId":       "xxxx",
                            "VideoUrl":     "http://xxxx"
                        }
                }
        }
}
```

上传失败的回调：

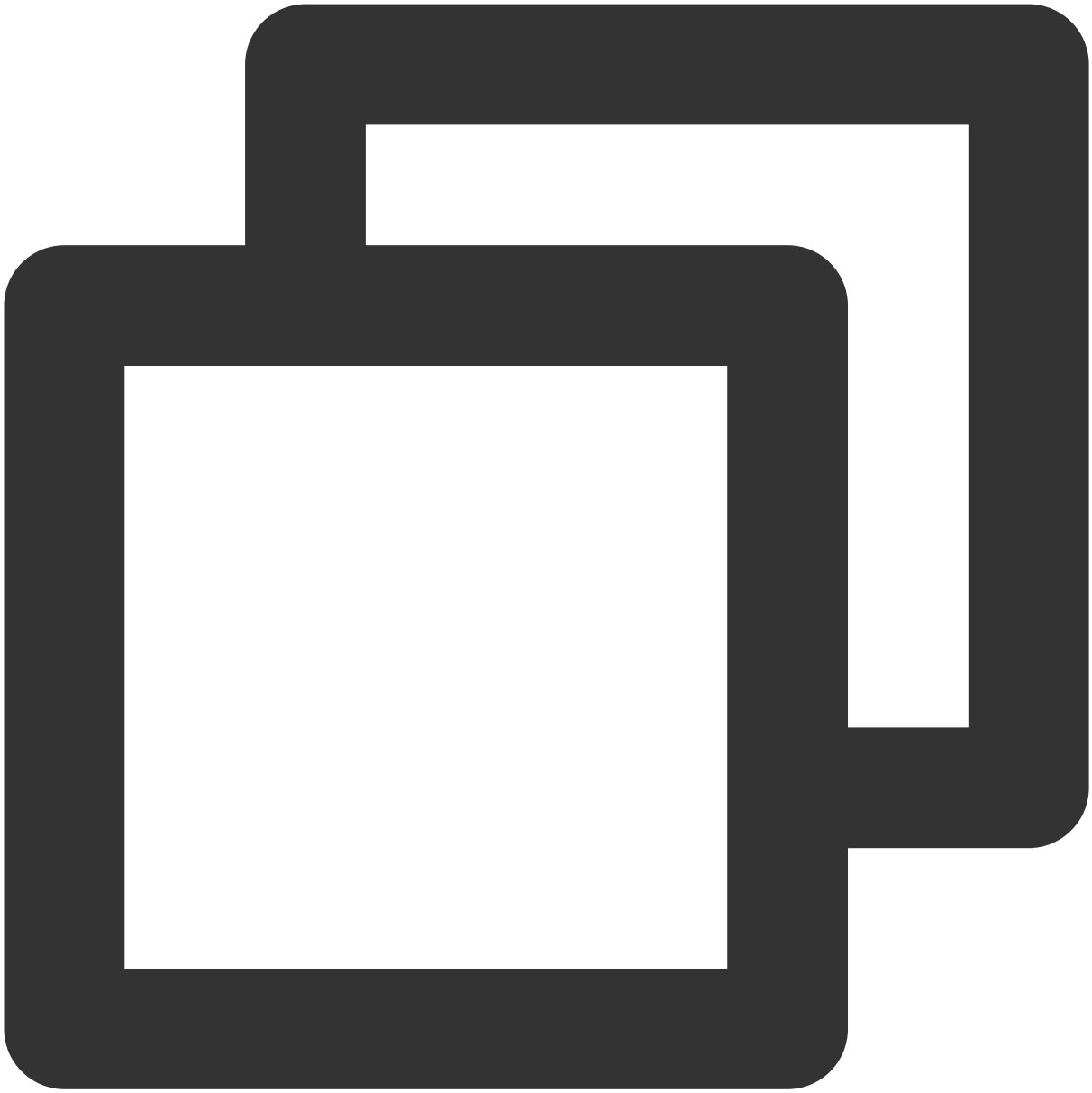


```
{
        "EventGroupId": 3,
        "EventType":    311,
        "CallbackTs":   1622191965320,
        "EventInfo":    {
                "RoomId":       "20015",
                "EventTs":      1622191965,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       1,
```

```
                        "Errmsg":       "xxx",
                        "TencentVod":   {
                            "UserId":       "123",
                            "TrackType":    "audio_video",
                            "CacheFile":    "xxx.mp4"
                        }
                    }
            }
    }
```

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,
                "UserId":       "xx",
                "TaskId":       "xx",
                "Payload":      {
                        "Status":       0
```

```
                    }
                }
            }
```

最佳实践

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

1. 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100路)，请联系腾讯云技术支持来调整最高并发路数限制。

2. 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直，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。

3. 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。具体文件名规则请看上面(这里链接到文件名规则)。

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。（这里链接到回调页面）

通过Cos来查询录制文件，发起云端录制的时候可以指定存储在cos的目录，录制任务结束以后，可以找到对应的目录来找到录制文件。

4. 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。

自定义视频采集和渲染 Android&iOS&Windows&Mac

最近更新时间：2022-07-20 14:26:08

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

自定义视频采集

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

- Android
- iOS
- Windows

开启自定义视频采集功能

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

- Android Java
- iOS&Mac ObjC
- Windows C++

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

发送自定义视频帧

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

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

- Android Java
- iOS&Mac ObjC
- Windows C++

```
// 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, videoFra
me);
```

自定义视频渲染

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

- Android：
- iOS
- Windows

设置本地预览画面的渲染回调

- Android Java
- iOS&Mac ObjC
- Windows C++

```
mTRTCCloud.setLocalVideoRenderListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Text
ure_2D, TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCV
ideoRenderListener() {
@Override
public void onRenderVideoFrame(String userId int streamType, TRTCCloudDef.TRTCV
ideoFrame frame) {
```

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

设置远端用户画面的渲染回调

- Android Java
- iOS&Mac ObjC
- Windows C++

```
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
}
});
```

Web

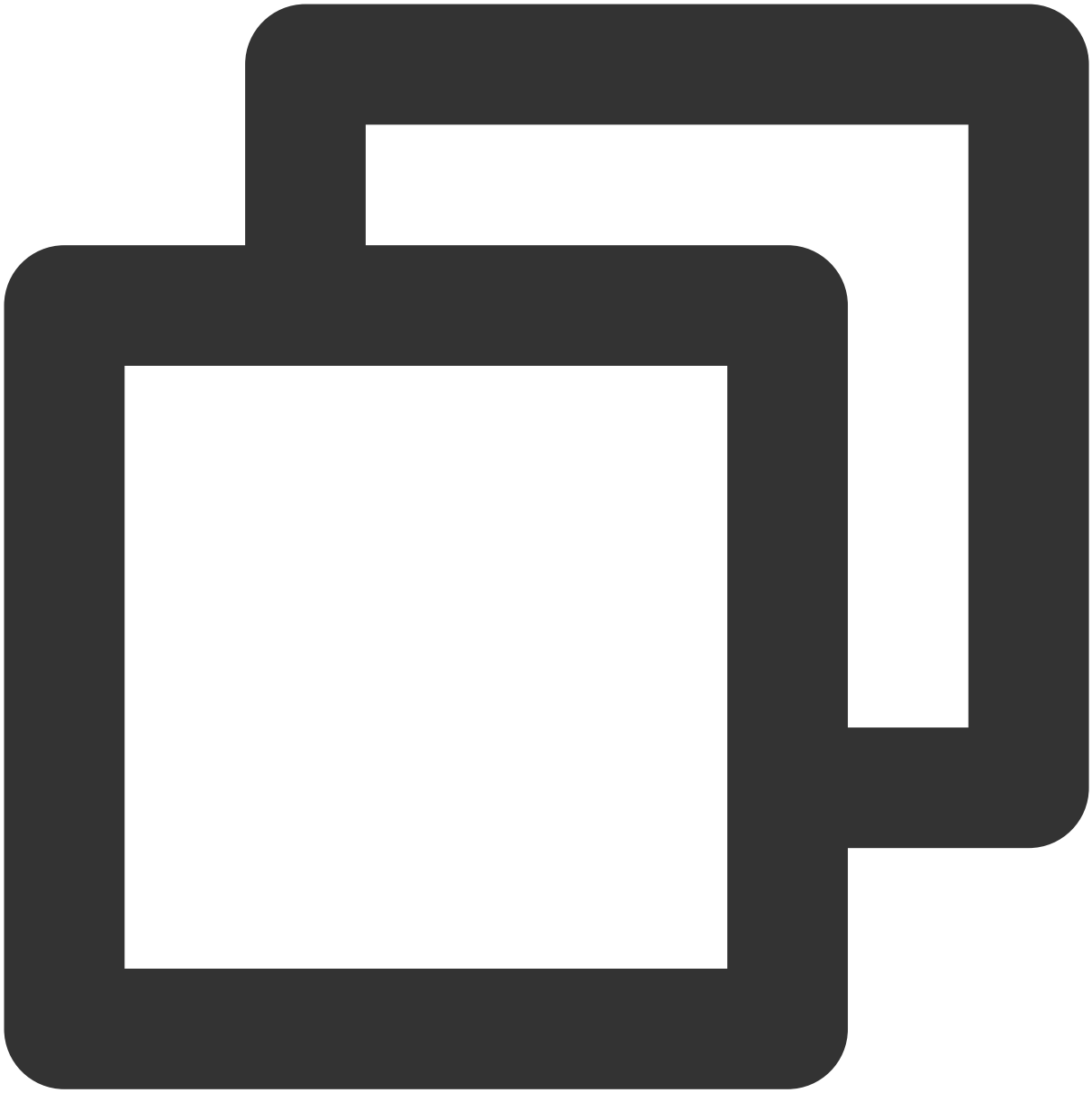
最近更新时间：2023-05-31 10:58:30

本文主要介绍本地流的自定义采集和音视频流的自定义播放渲染等高阶用法。

自定义采集

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。

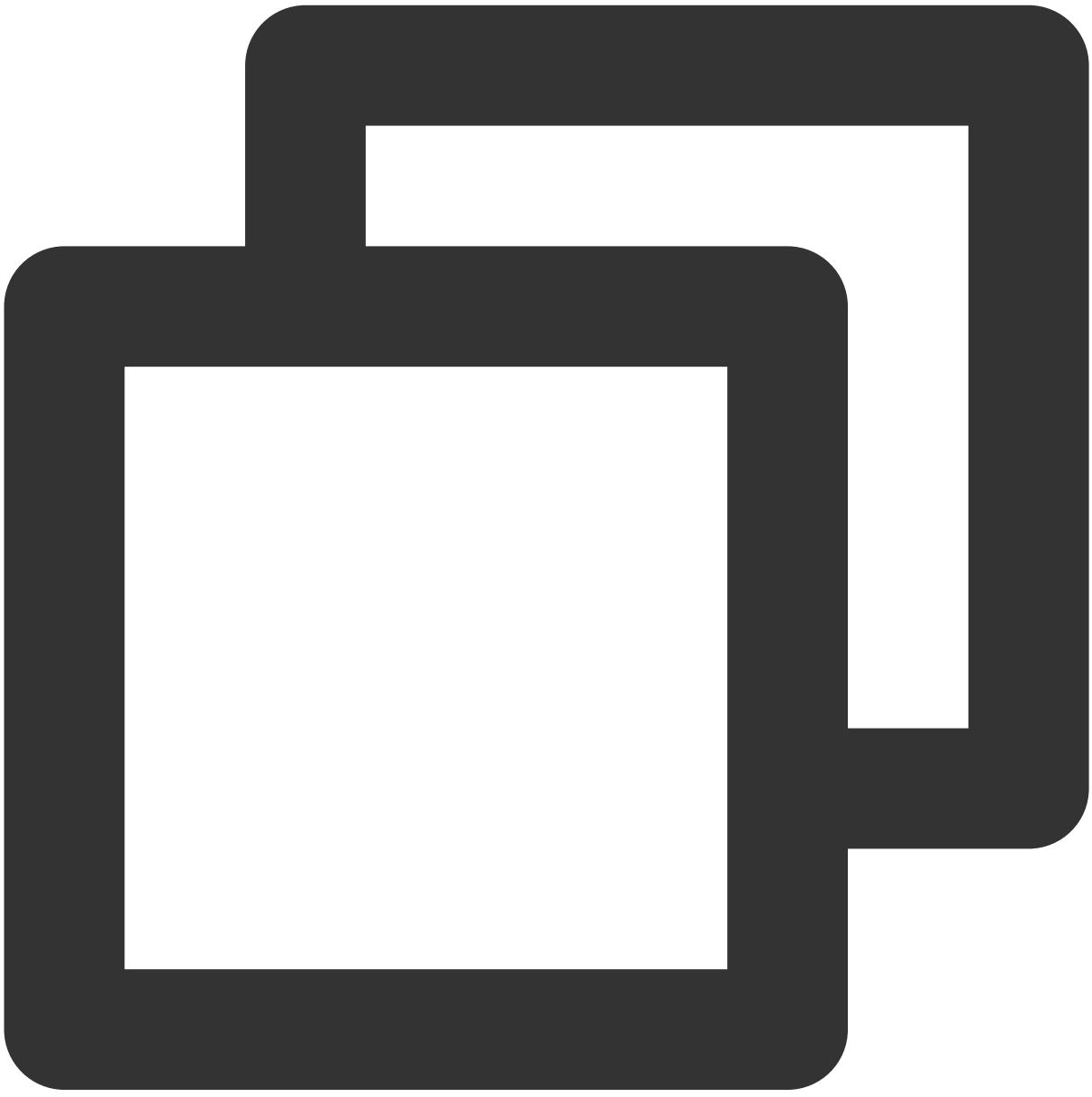
采集 video 标签中正在播放的视频



```
// 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

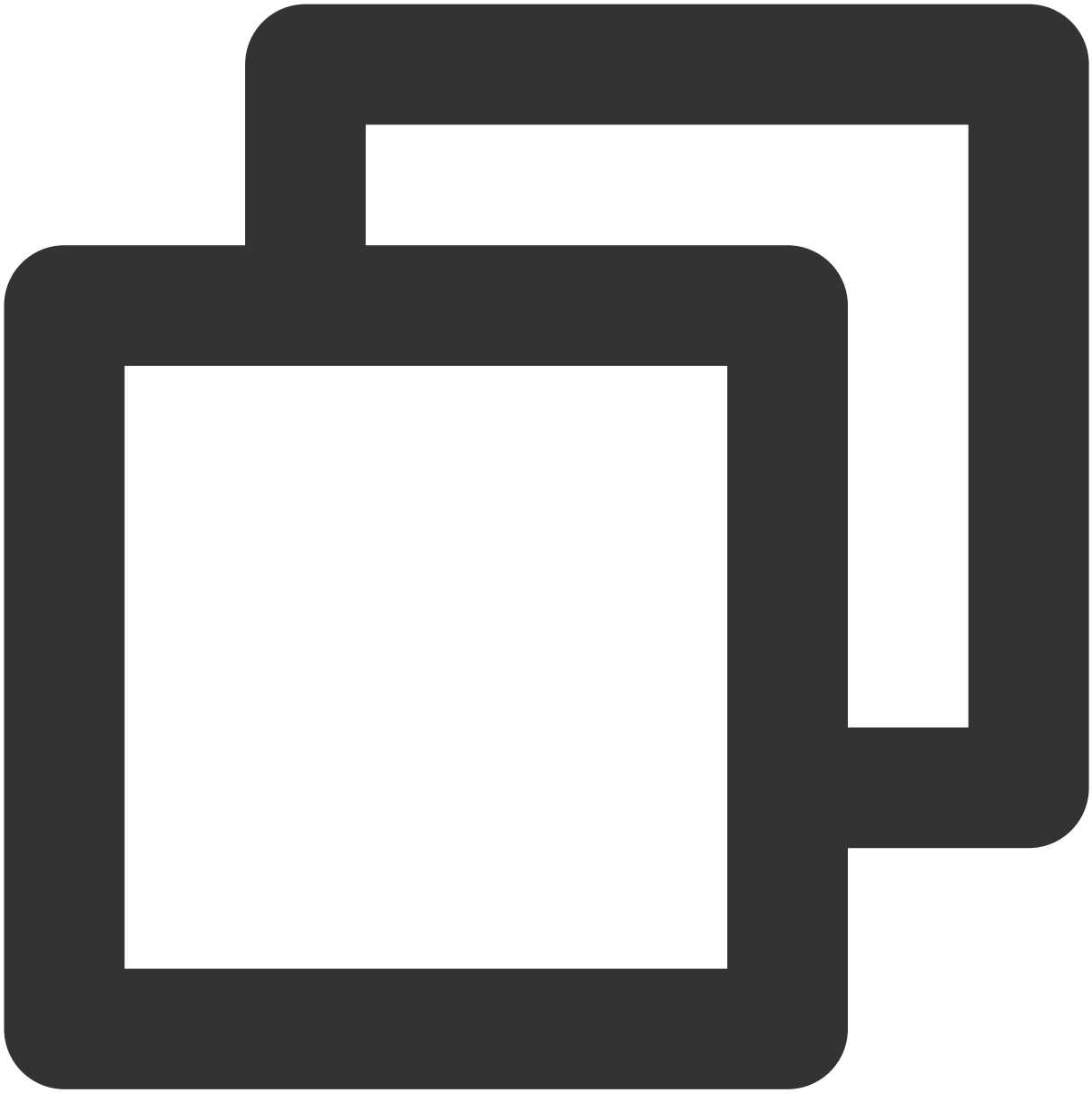
通过 TRTC.getVideoTrack() 方法获取相应的 videoTrack

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。

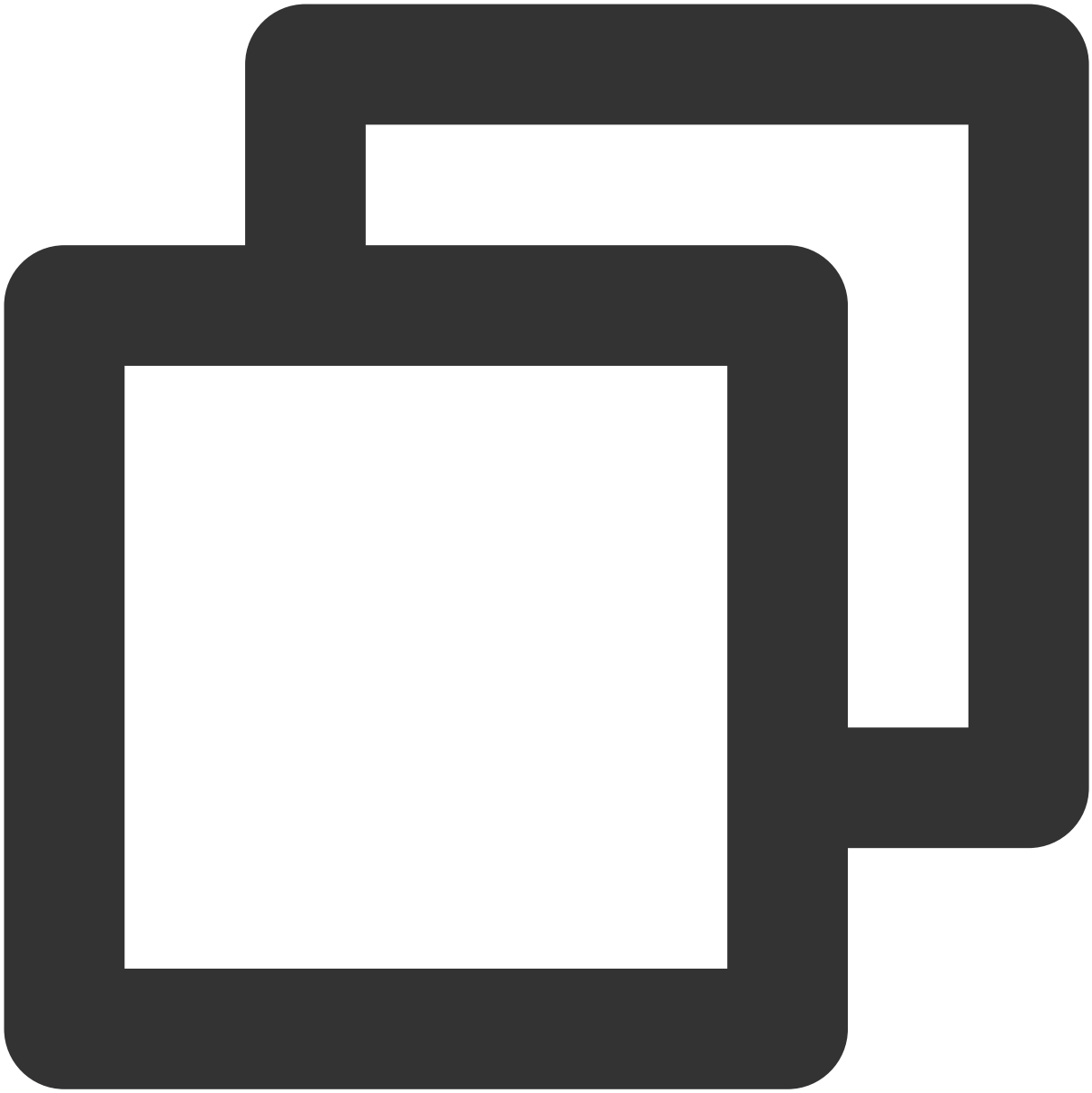
对于远端视频，还需要监听 REMOTE_VIDEO_AVAILABLE，REMOTE_VIDEO_UNAVAILABLE 事件来处理远端视频的生命周期。

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();
});
```

Flutter

最近更新时间：2024-06-07 11:20:49

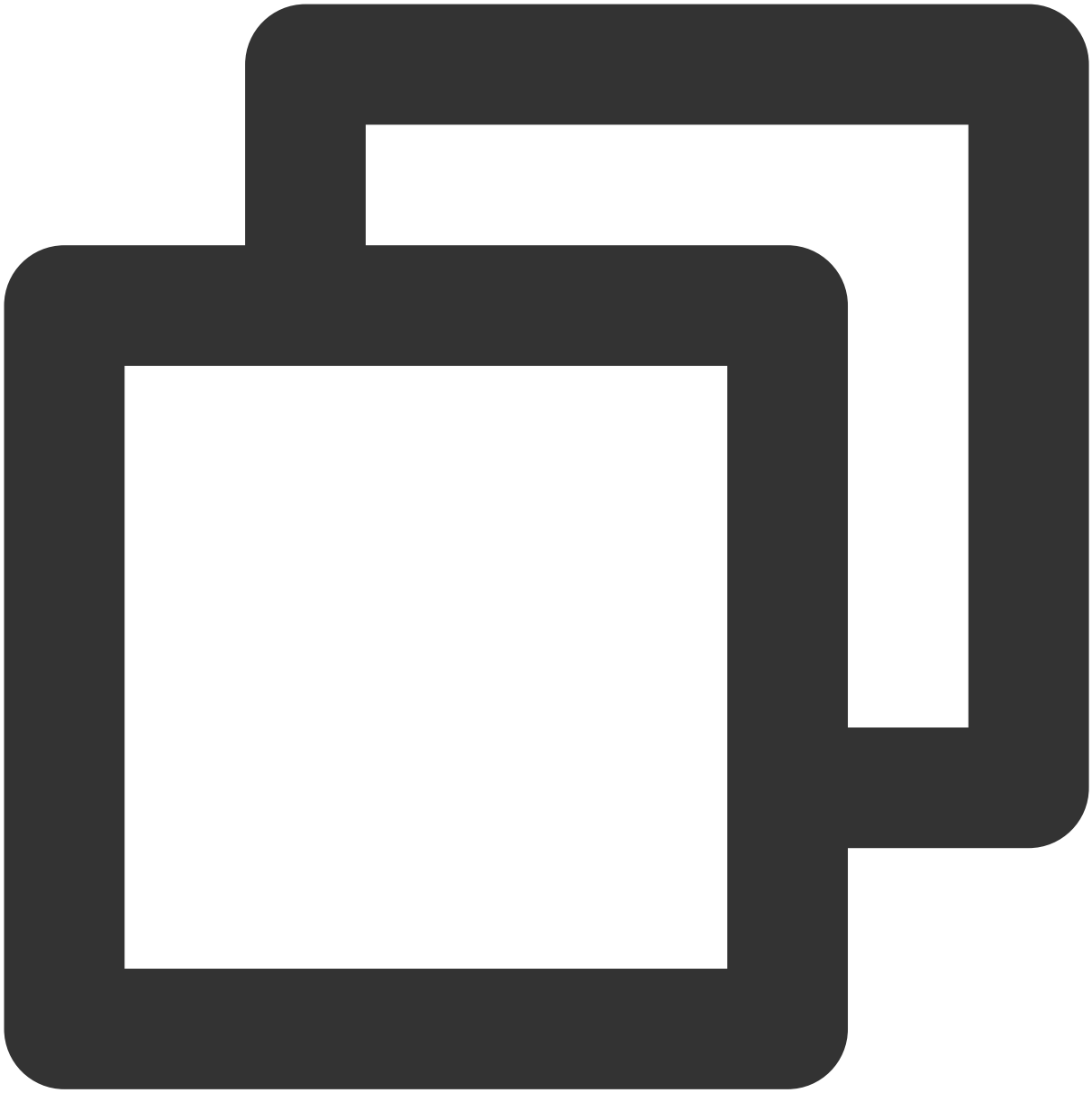
Flutter本文档主要介绍如何使用 TRTC Flutter SDK 实现自定义音频原数据获取。

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。

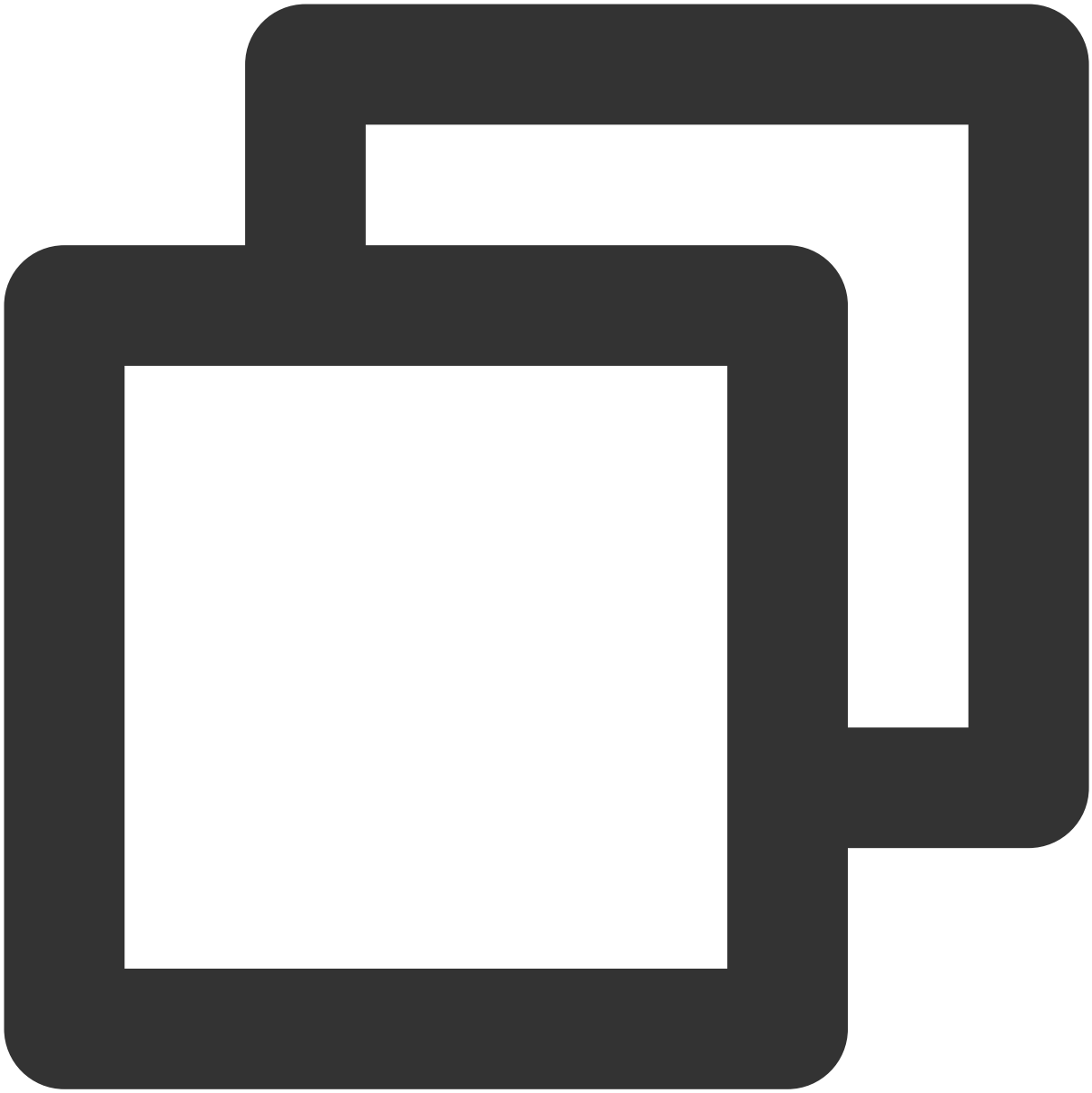
Native 接入

具体接入过程和接入效果可使用 demo 体验。
1. 在 Native 层监听音频原数据，获取音频原数据。
java
swift



```
void enableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance(getApplicationContext()).setAudioFrameListener(new Aud
    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 trtcAudioFra
        // TODO
    }

    @Override
    public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame,
        // TODO
    }

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

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

    @Override
    public void onVoiceEarMonitorAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFr
        // TODO
    }
}
```

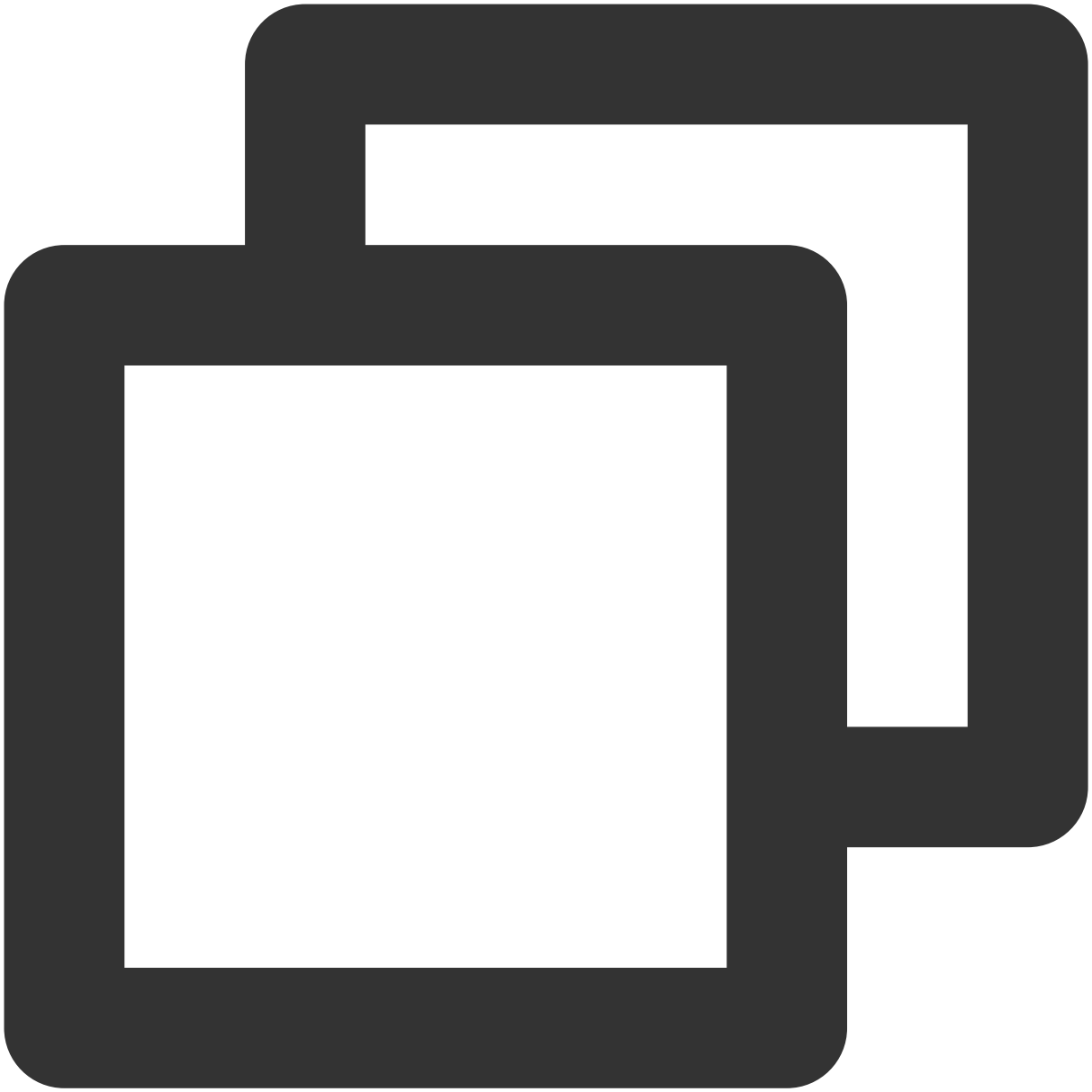


```
let listener = AudioFrameProcessListener()
func enableTRTCAudioFrameDelegate() {
    TRTCCloud.sharedInstance().setAudioFrameDelegate(listener)
    result(nil)
}

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
    }
}
```

2. 使用 Method Channel 实现开始/停止获取音频原数据。

2.1 在 Dart 层实现获取音频原数据的开始/停止接口。

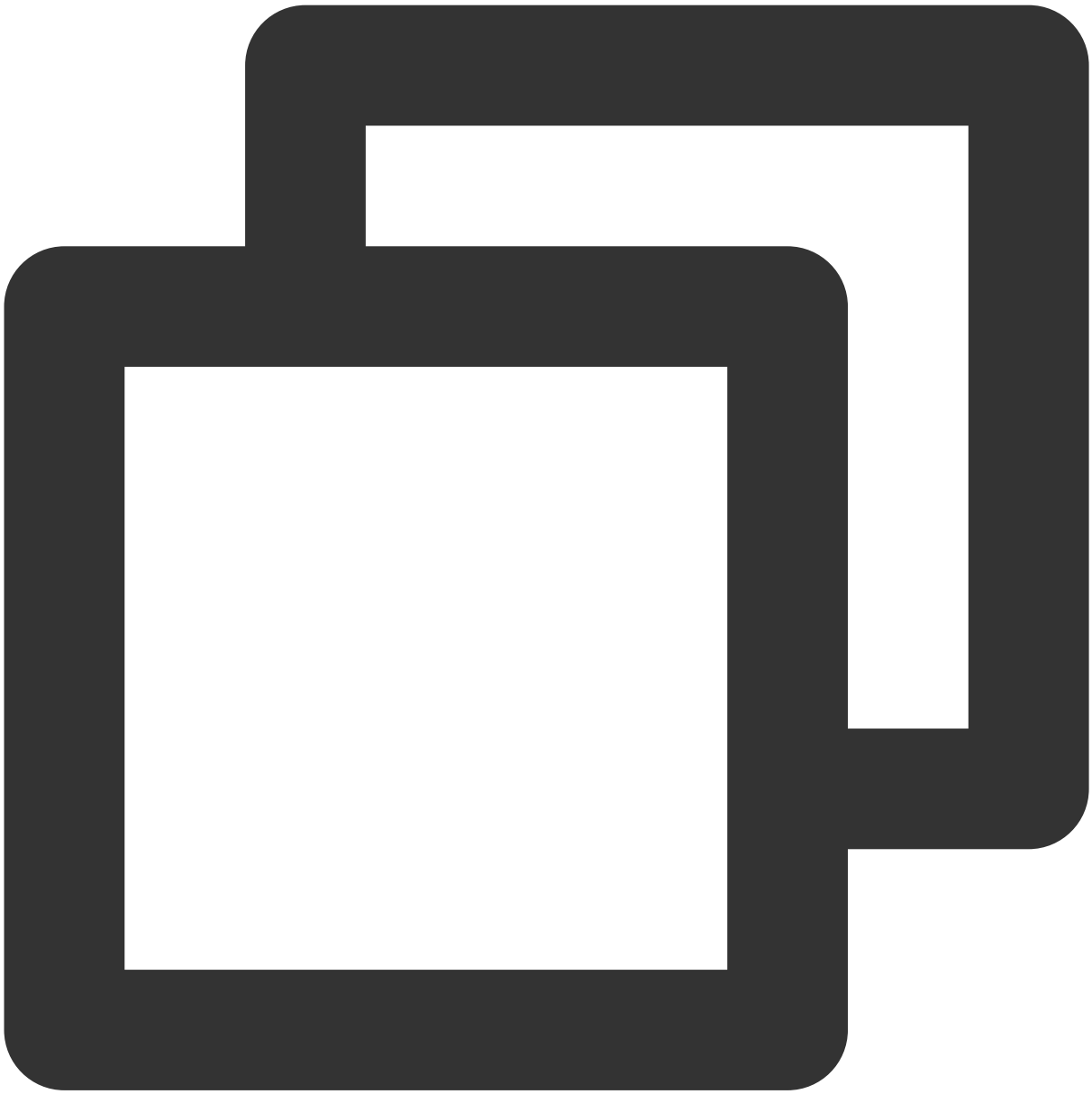


```
final channel = MethodChannel('TRCT_FLUTTER_EXAMPLE');

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

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

2.2 在 Native 层实现获取音频原数据的开始/停止接口。

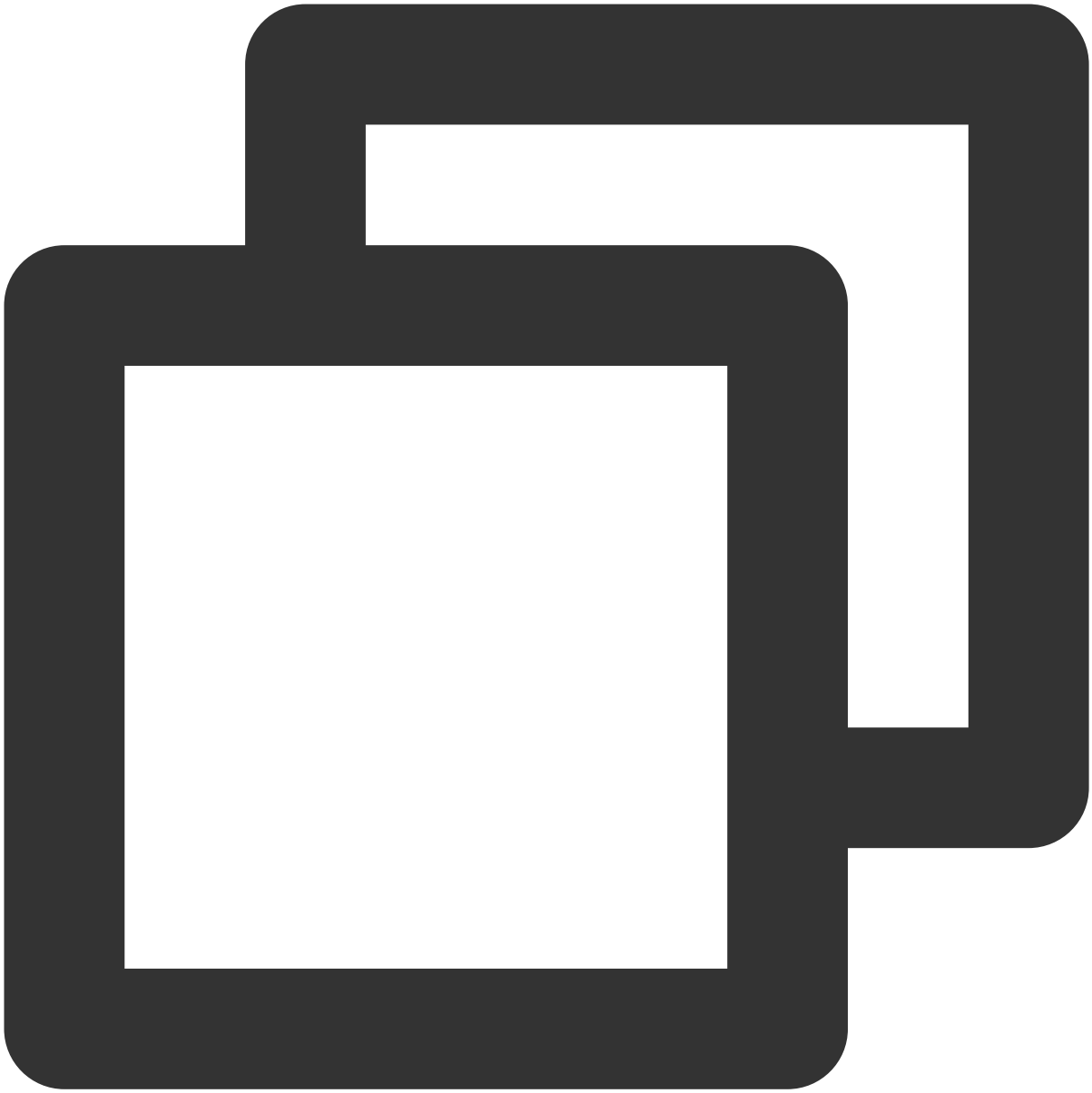
java

swift



```
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().getBinaryMessen
        channel.setMethodCallHandler(((call, result) -> {
            switch (call.method) {
                case "enableTRTCAudioFrameDelegate":
                    enableTRTCAudioFrameDelegate();
                    break;
                case "disableTRTCAudioFrameDelegate":
                    disableTRTCAudioFrameDelegate();
                    break;
                default:
                    break;
            }
        }));
    }
}
```

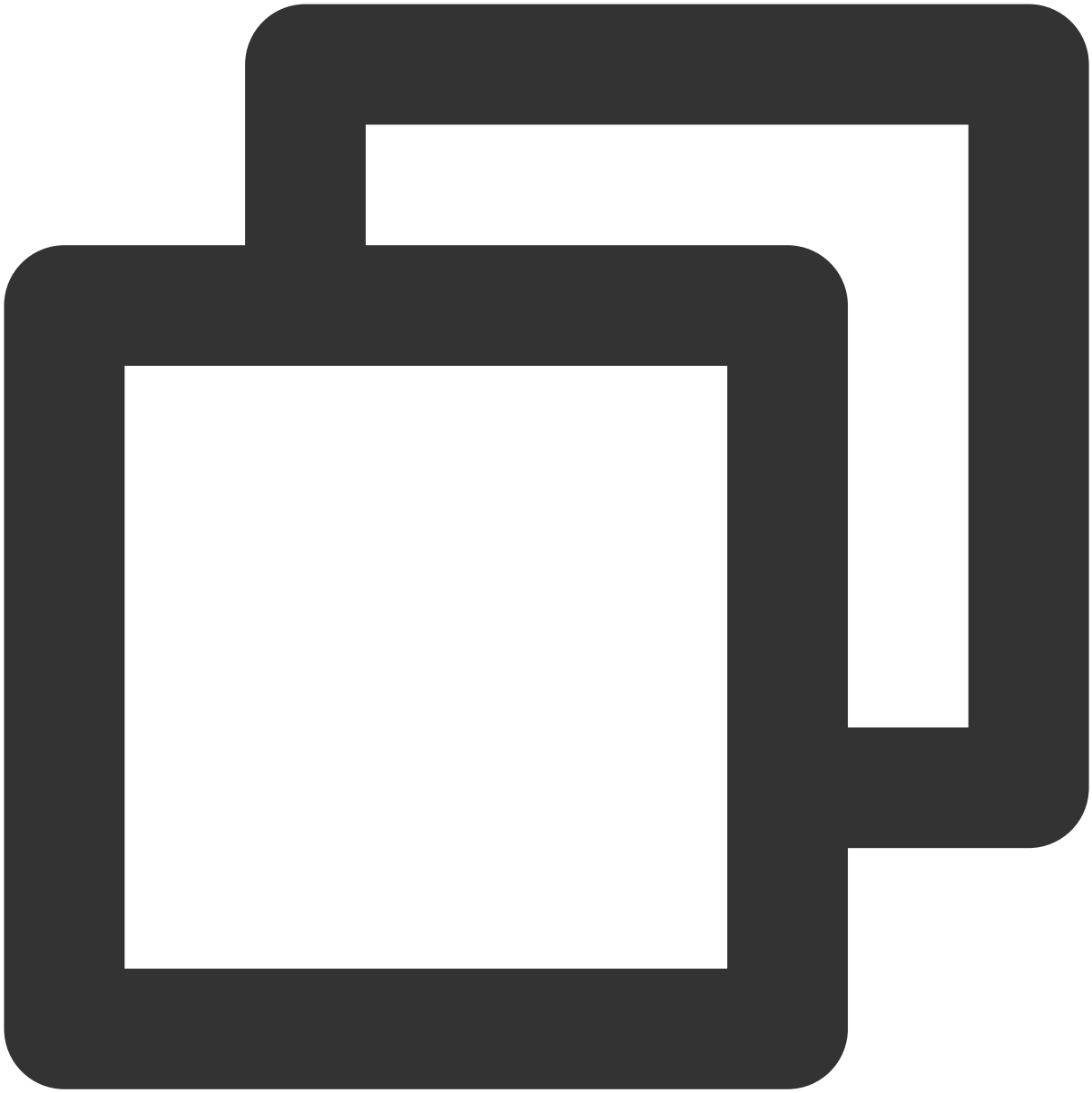


```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
    var channel: FlutterMethodChannel?

    override func application(_ application: UIApplication,
                             didFinishLaunchingWithOptions launchOptions: [UIAppli
        GeneratedPluginRegistrant.register(with: self)

        guard let controller = window?.rootViewController as? FlutterViewController
            fatalError("Invalid root view controller")
        }
```

```
        channel = FlutterMethodChannel(name: "TRCT_FLUTTER_EXAMPLE", binaryMessenge
        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: launch
    }
}
```

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);
```

自定义音频采集和播放 Android&iOS&Windows&Mac

最近更新时间：2022-07-20 14:51:36

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

自定义音频采集

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

- Android
- iOS
- Windows

开启自定义音频采集功能

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

- Android Java
- iOS&Mac ObjC
- Windows C++

```
TRTCCloud mTRTCCloud = TRTCCloud.shareInstance();
mTRTCCloud.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 Java
- iOS&Mac ObjC
- Windows C++

```
TRTCCloudDef.TRTCAudioFrame trtcAudioFrame = new TRTCCloudDef.TRTCAudioFrame();
trtcAudioFrame.data = data;
```

```
trtcAudioFrame.sampleRate = sampleRate;
trtcAudioFrame.channel = channel;
trtcAudioFrame.timestamp = timestamp;
mTRTCCloud.sendCustomAudioData(trtcAudioFrame);
```

注意：
使用 `sendCustomAudioData` 有可能会导致回声抵消（AEC）的功能失效。

获取音频原数据

声音模块是一个高复杂度的模块，SDK 需要严格控制声音设备的采集和播放逻辑。在某些场景下，当您需要获取远程用户的音频数据或者需要获取本地麦克风采集到的音频数据时，可以通过 TRTCCloud 对应的不同平台的接口，我们也提供有对应平台的的 API-Example：

- Android：
- iOS
- Windows

设置音频回调函数

- Android Java
- iOS&Mac ObjC
- Windows C++

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

}

@Override
public void onLocalProcessedAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFram
e) {

}

@Override
```

```
public void onRemoteUserAudioFrame(TRTCCloudDef.TRTCAudioFrame trtcAudioFrame, S
tring 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.Cust
omFrameRender
}
});
```

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

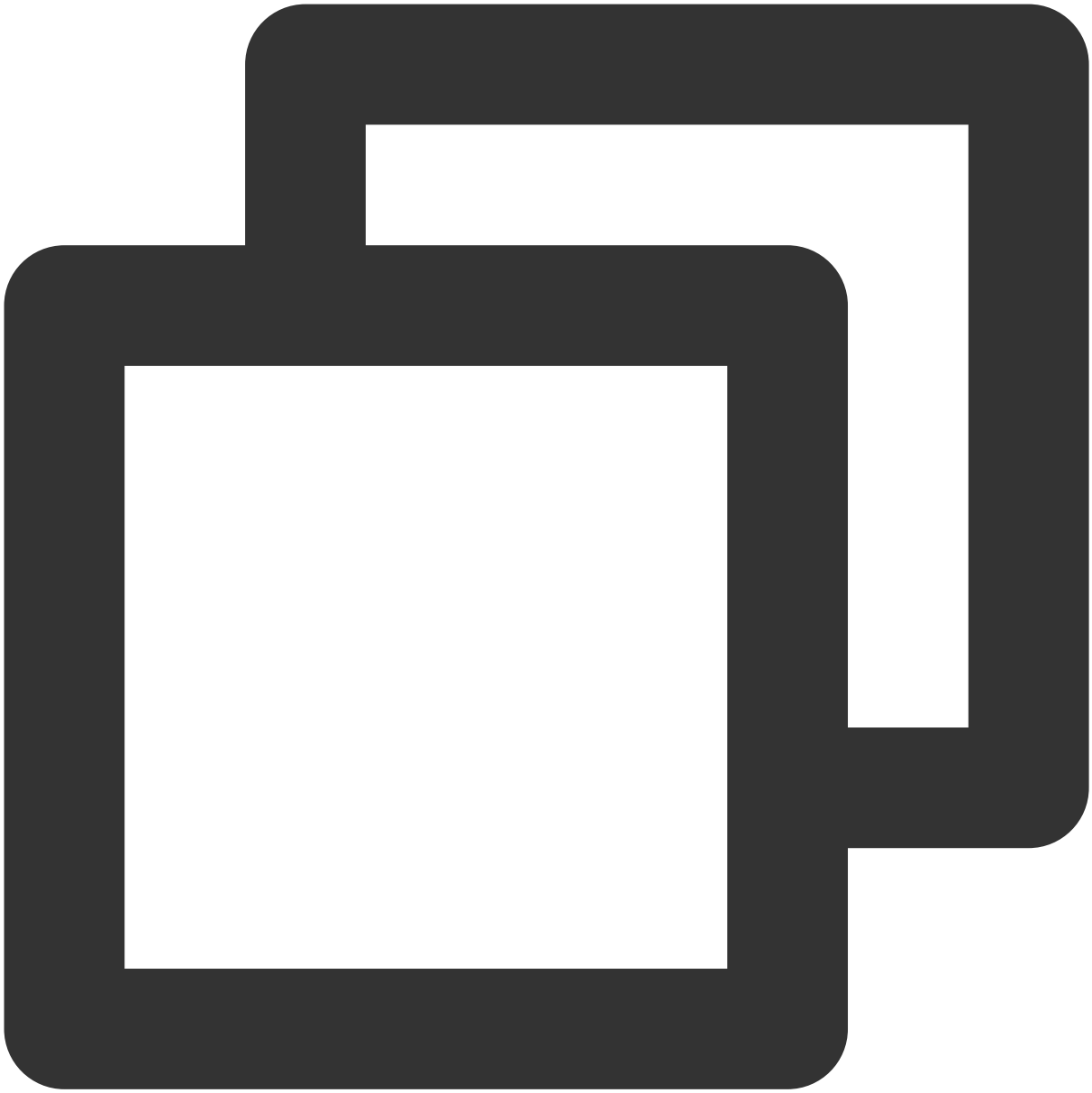
最近更新时间：2023-05-31 10:57:20

本文主要介绍本地流的自定义采集和音视频流的自定义播放渲染等高阶用法。

自定义采集

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。

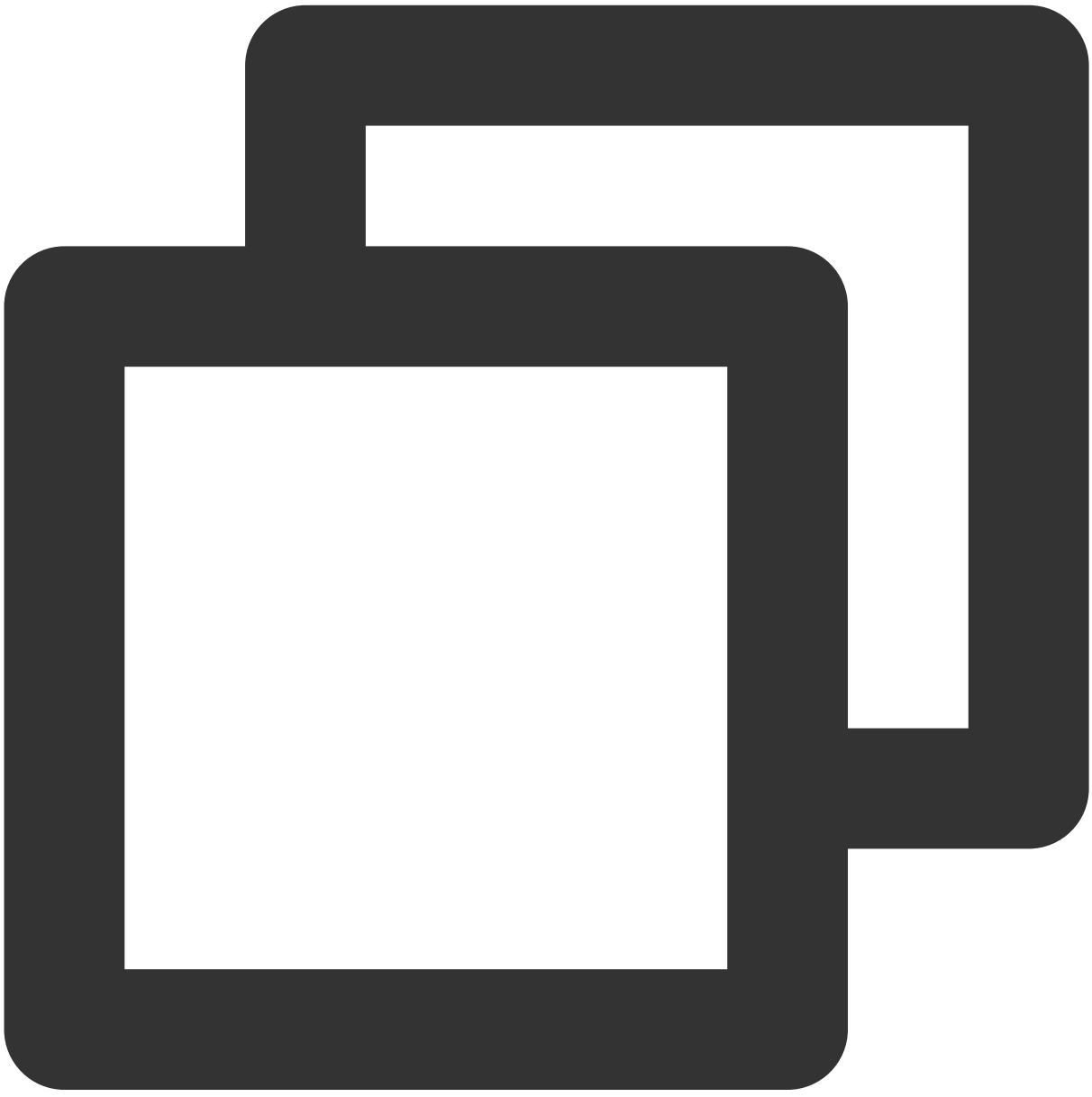
采集 video 标签中正在播放的视频



```
// 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

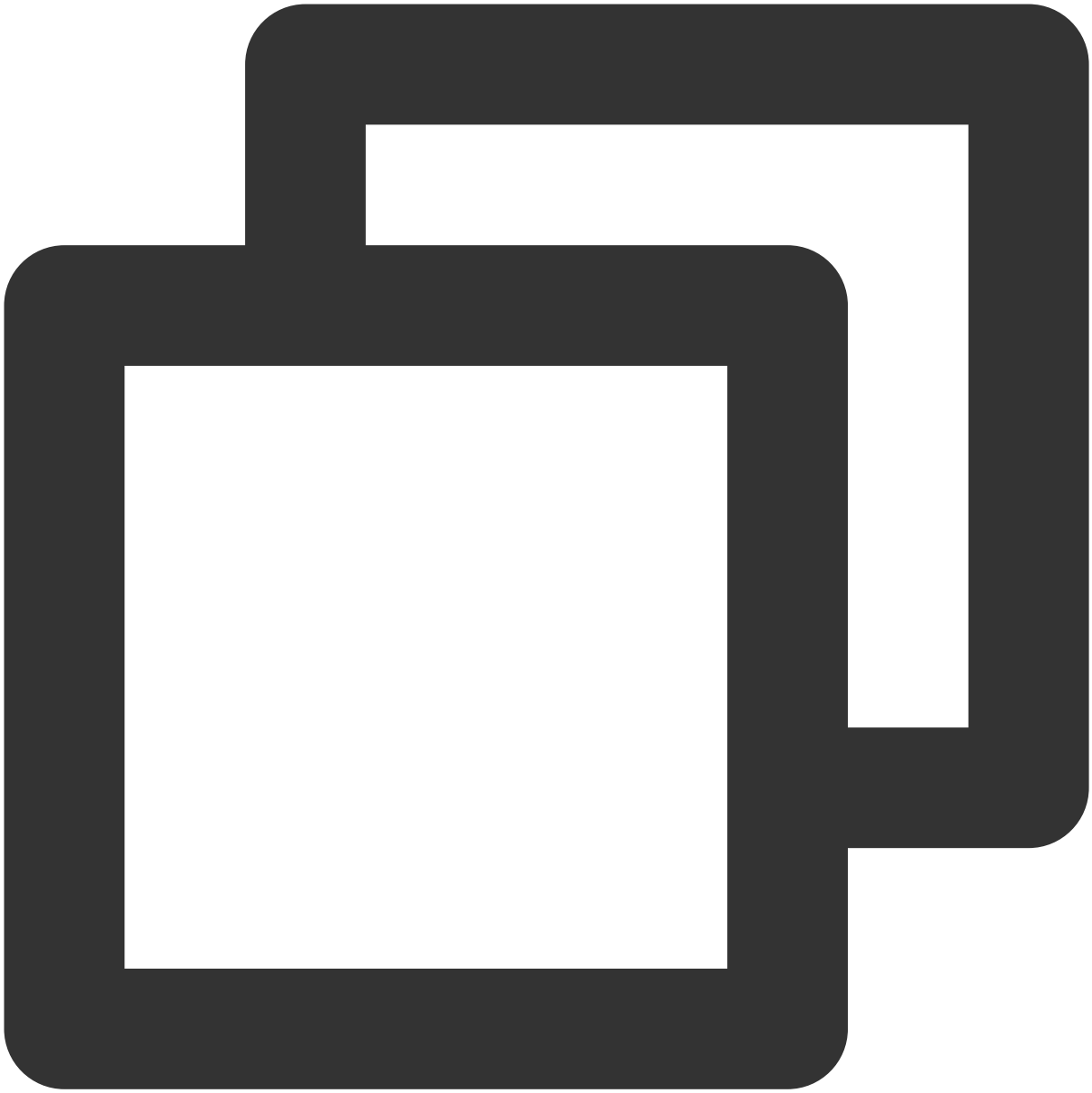
通过 TRTC.getVideoTrack() 方法获取相应的 videoTrack

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。

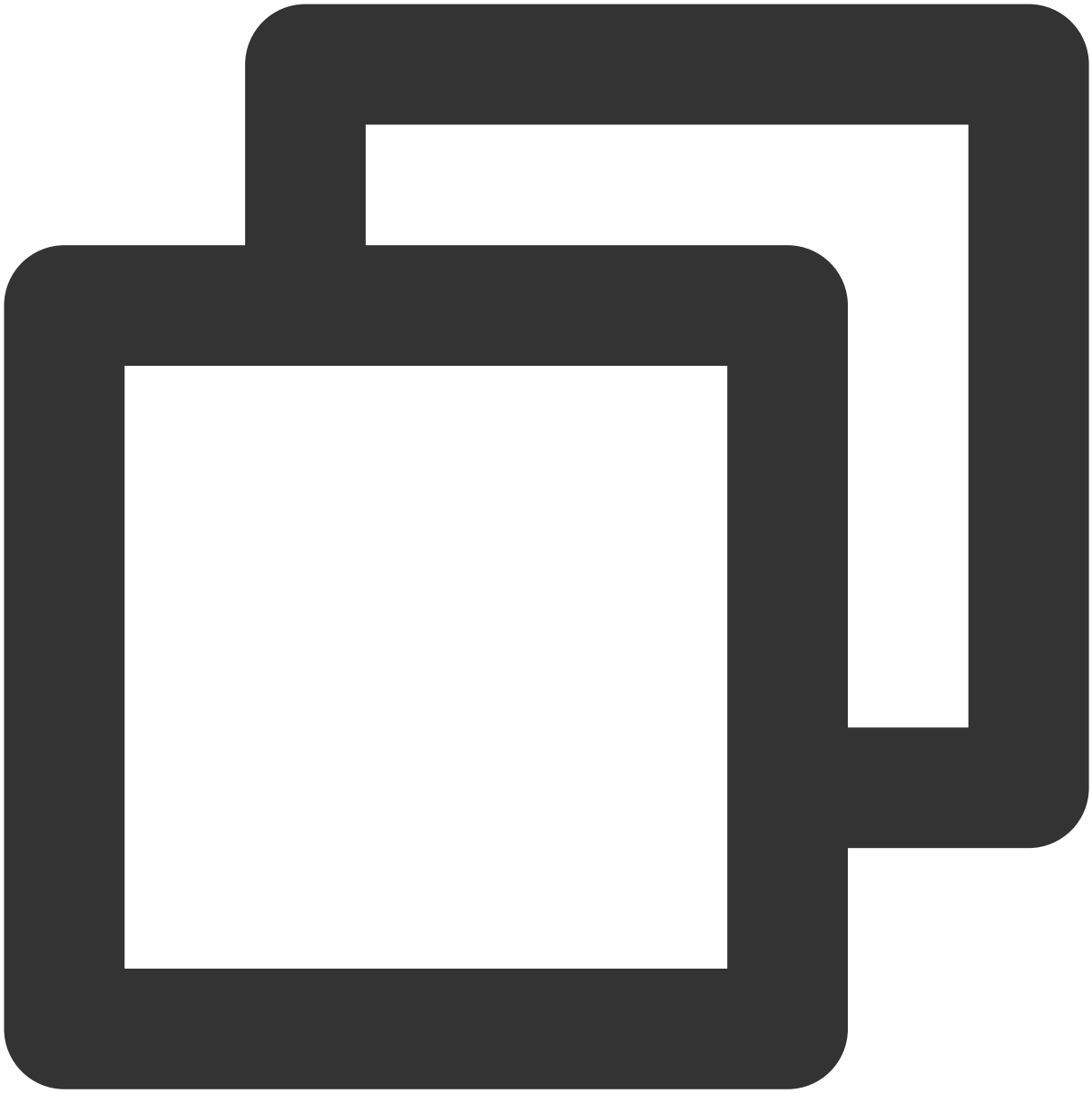
对于远端视频，还需要监听 REMOTE_VIDEO_AVAILABLE，REMOTE_VIDEO_UNAVAILABLE 事件来处理远端视频的生命周期。

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();
});
```

发送和接收消息

最近更新时间：2024-07-17 11:44:34

内容介绍

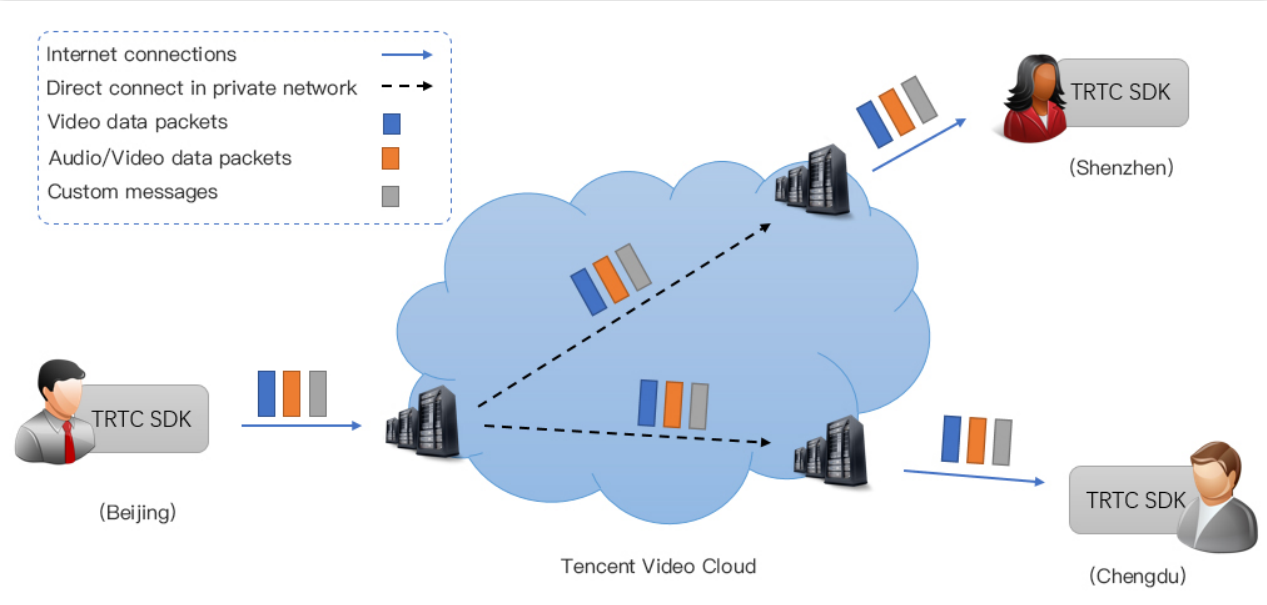
TRTC SDK 提供了发送自定义消息的功能，通过该功能，角色为主播的用户都可以向同一个视频房间里的其他用户广播自己的定制消息。

支持的平台

iOS	Android	Mac OS	Windows	Electron	Web 端
✓	✓	✓	✓	✓	×

发送接收原理

某一个用户的自定义消息会被夹在音视频数据流中，随着音视频数据一起传输给房间里的其他用户。由于音视频线路本身并不是100%可靠的，为了提高可靠性，TRTC SDK 内部本身实现了一些可靠性保护机制。



消息发送

通过调用 TRTCCloud 的 `sendCustomCmdMsg` 接口发送的，发送时需要指定四个参数：

参数名	参数说明
cmdID	消息ID，取值范围为 1 ~ 10，不同业务类型的消息应当使用不同的 cmdID。
data	待发送的消息，最大支持 1KB（1000字节）的数据大小。
reliable	是否可靠发送，可靠发送的代价是会引入一定的延时，因为接收端要暂存一段时间的数据来等待重传。
ordered	是否要求有序，即是否要求接收端接收的数据顺序和发送端发送的顺序一致，这会带来一定的接收延时，因为在接收端需要暂存并排序这些消息。

注意：

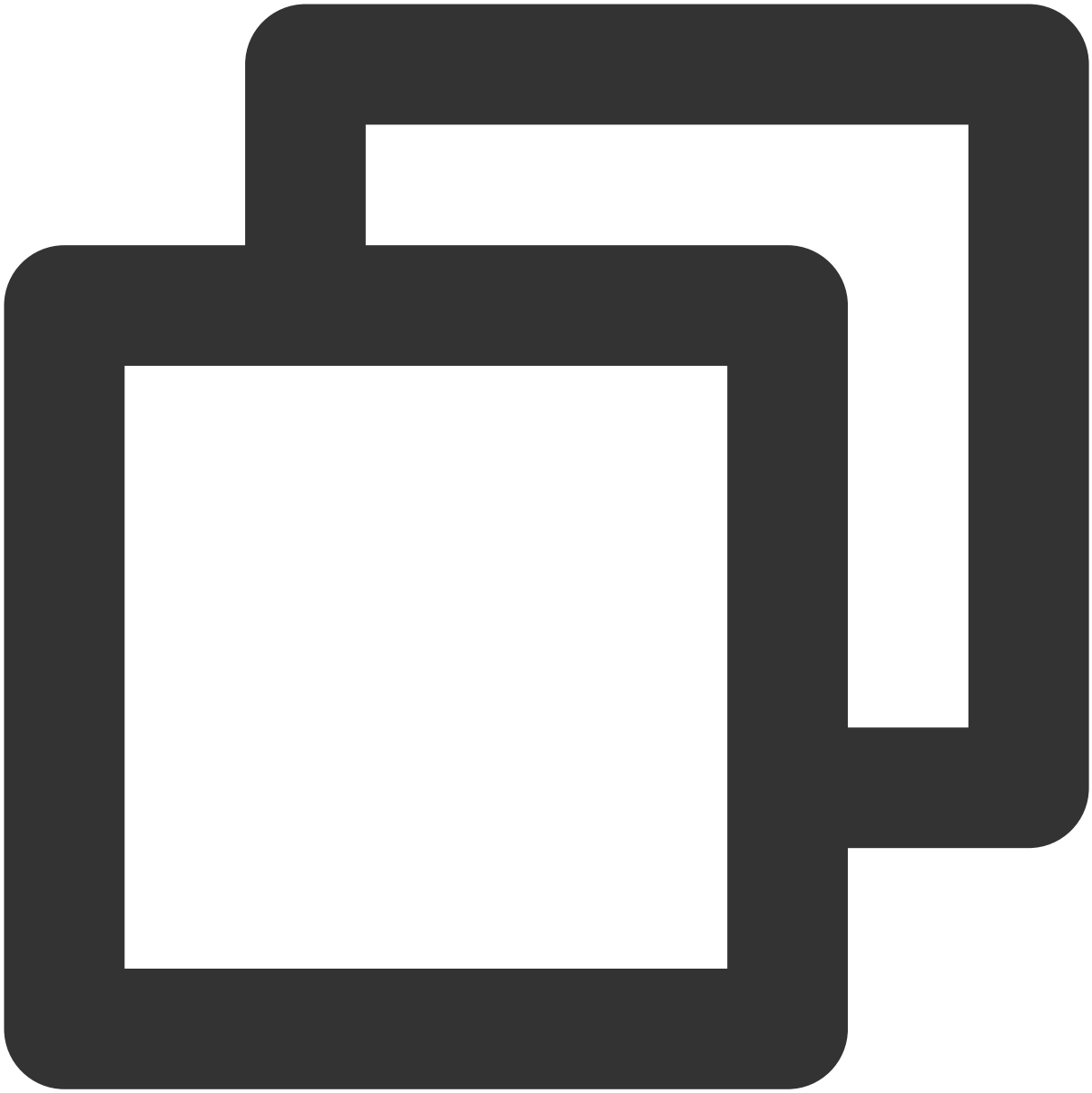
请将 reliable 和 ordered 同时设置为 YES 或 NO, 暂不支持交叉设置。

Objective-C

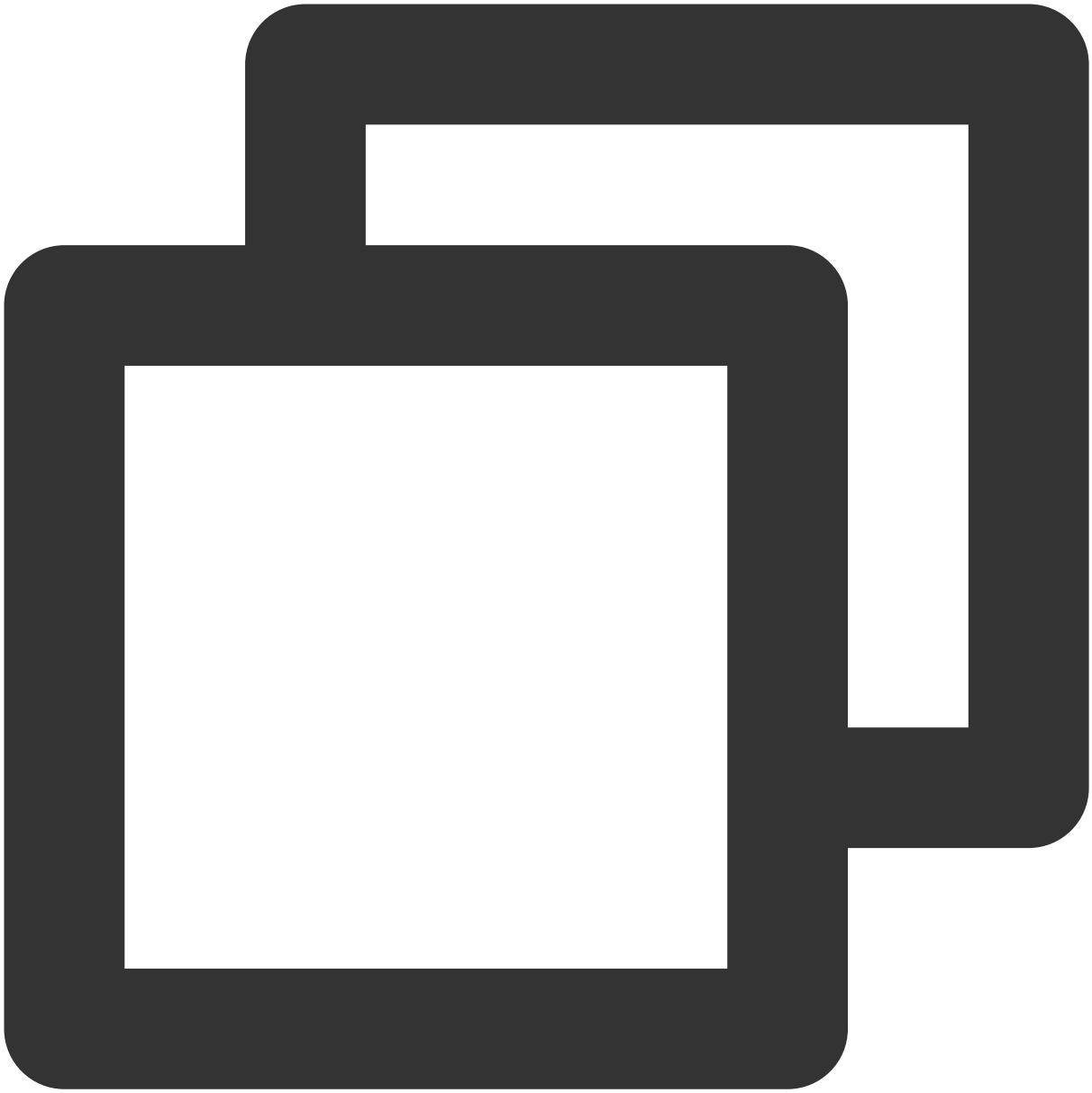
Java

C++

C#

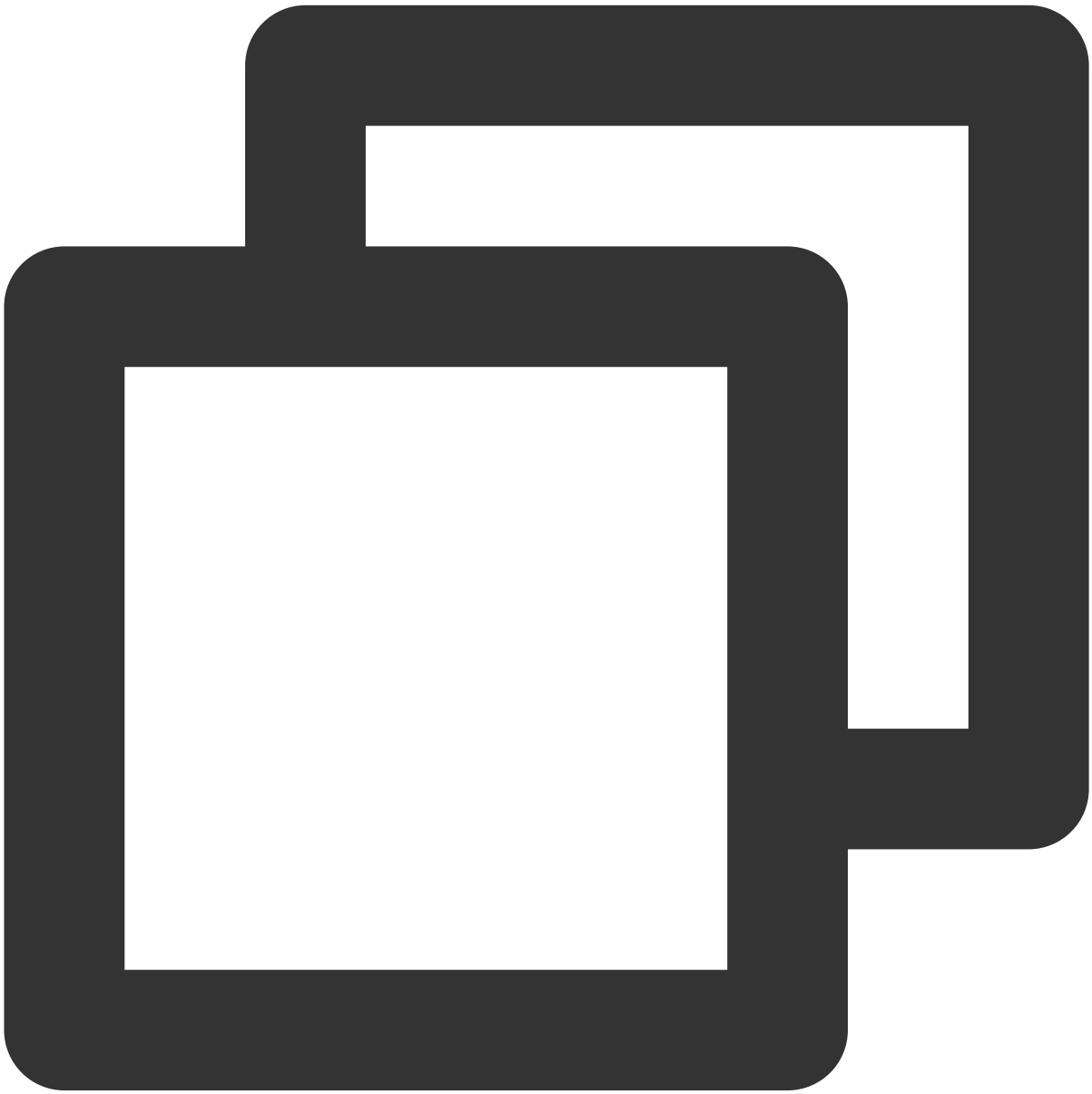


```
//发送自定义消息的示例代码
- (void)sendHello {
    // 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例
    NSInteger cmdID = 0x1;
    NSData *data = [@"Hello" dataUsingEncoding:NSUTF8StringEncoding];
    // reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
    [trtcCloud sendCustomCmdMsg:cmdID data:data reliable:YES ordered:YES];
}
```



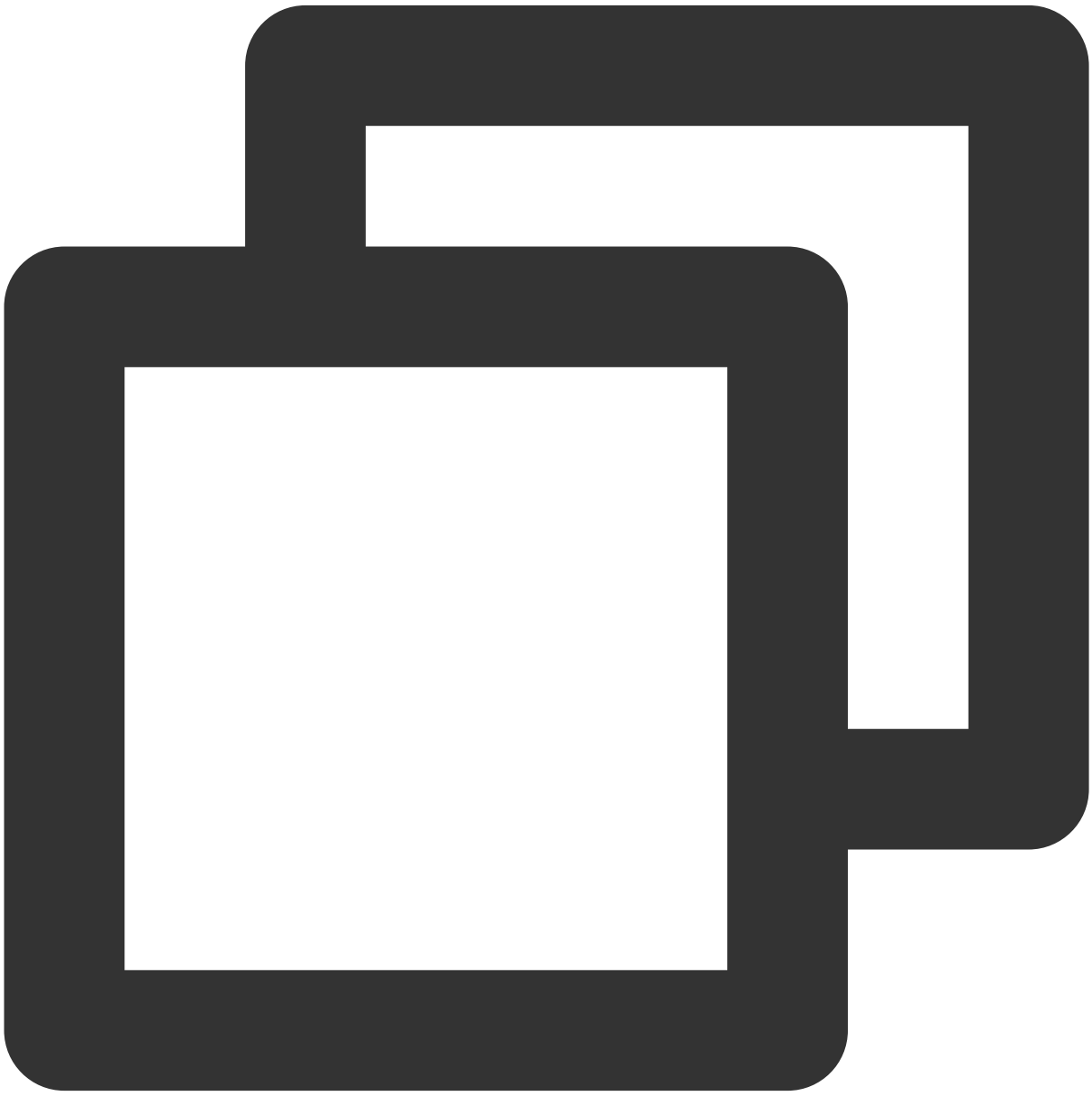
```
//发送自定义消息的示例代码
public void sendHello() {
    try {
        // 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例
        int cmdID = 0x1;
        String hello = "Hello";
        byte[] data = hello.getBytes("UTF-8");
        // reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
        trtcCloud.sendCustomCmdMsg(cmdID, data, true, true);

    } catch (UnsupportedEncodingException e) {
```

```
            e.printStackTrace();
        }
    }
```



```
// 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);
}
```



```
// 发送自定义消息的示例代码
```

```
private void sendHello()
{
    // 自定义消息命令字，这里需要根据业务定制一套规则，这里以0x1代表发送文字广播消息为例

    uint cmdID = 0x1;
    byte[] data = { '1', '2', '3' };
    uint dataSize = 3;  // data的长度

    // reliable 和 ordered 目前需要一致，这里以需要保证消息按发送顺序到达为例
    mTRTCCloud.sendCustomCmdMsg(cmdID, data, dataSize, true, true);
}
```

消息接收

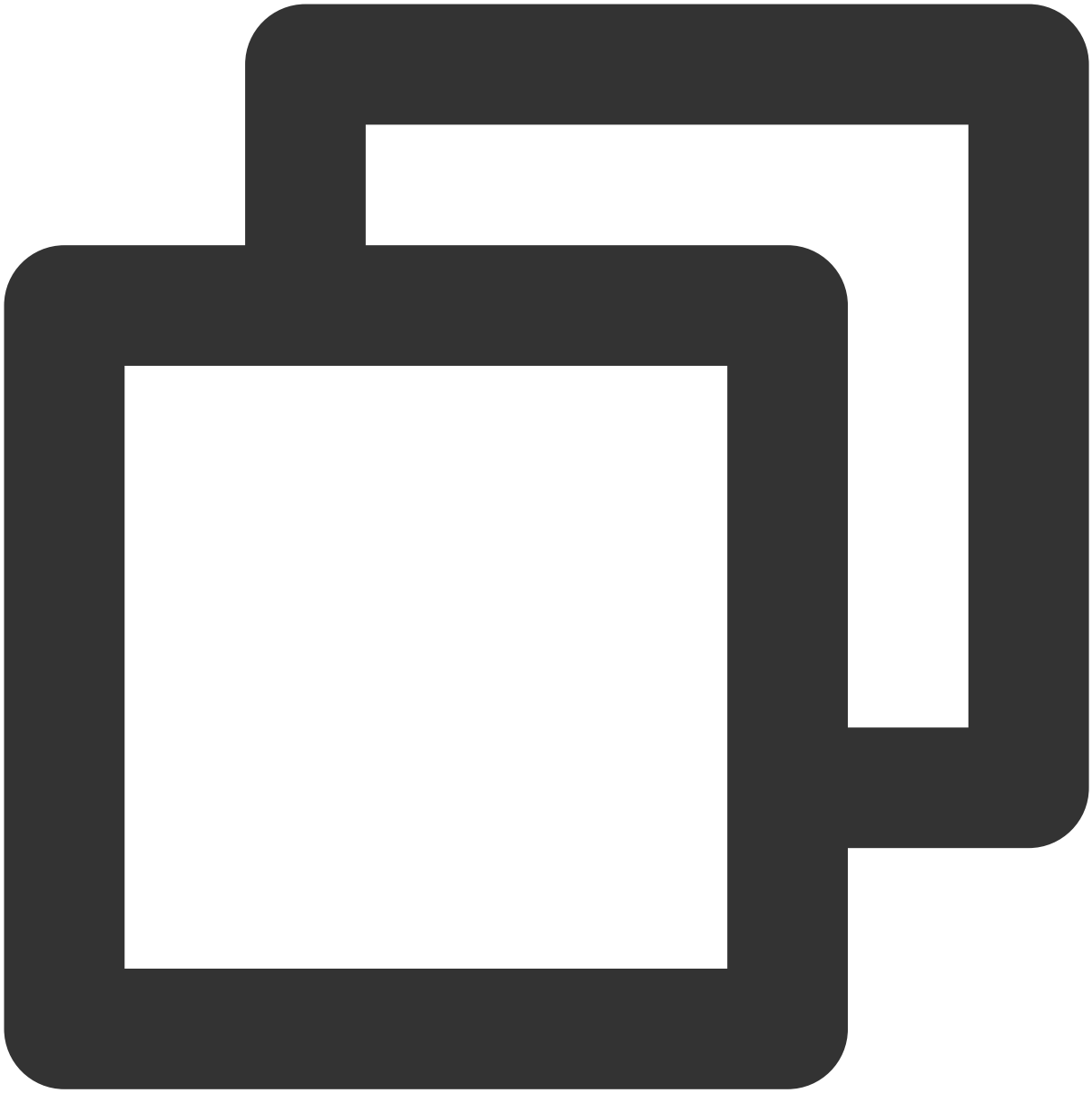
当房间中的一个用户通过 `sendCustomCmdMsg` 发出自定义消息后，房间中其他的用户可以通过 SDK 回调中的 `onRecvCustomCmdMsg` 接口来接收这些消息。

Objective-C

Java

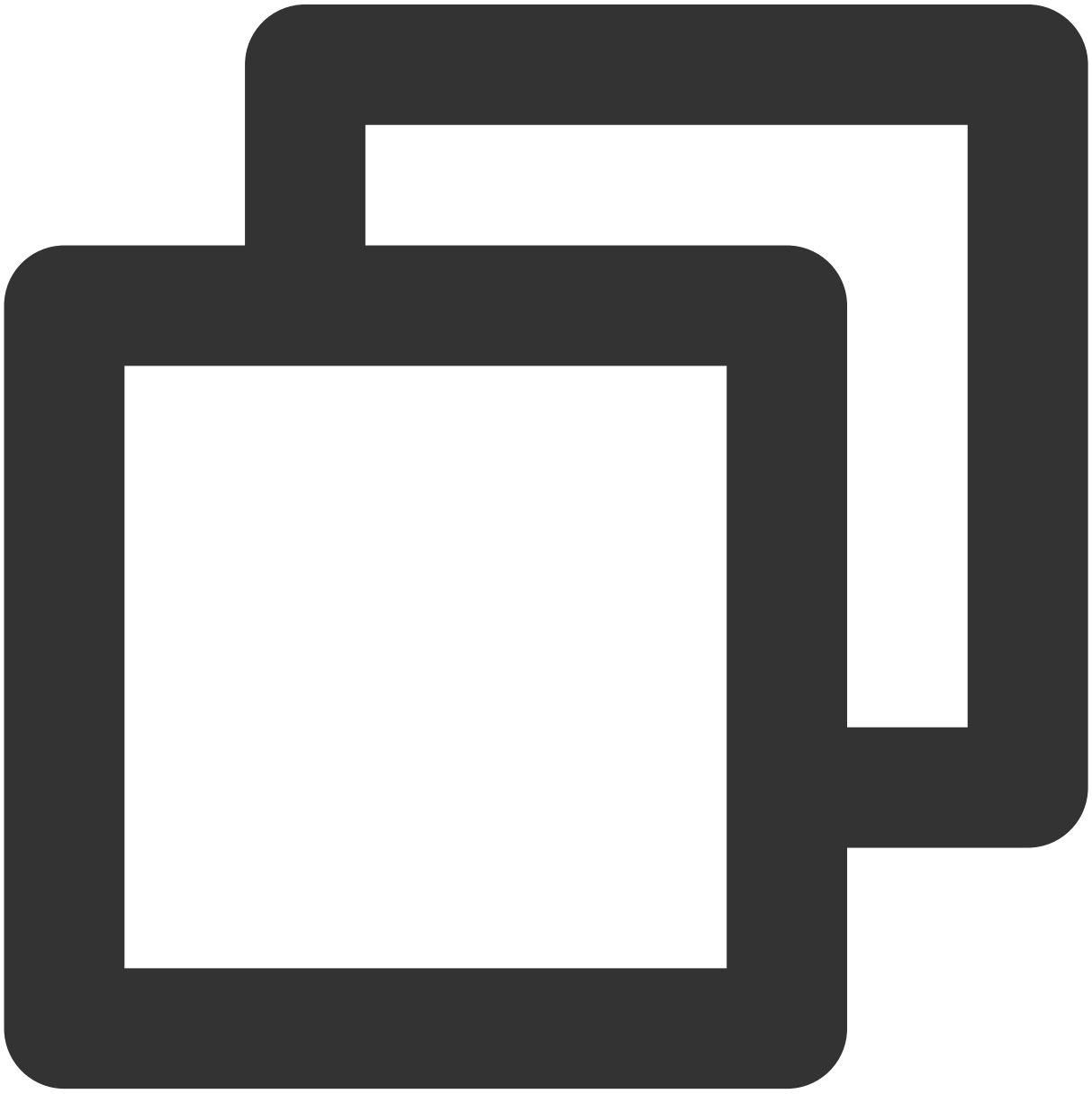
C++

C#

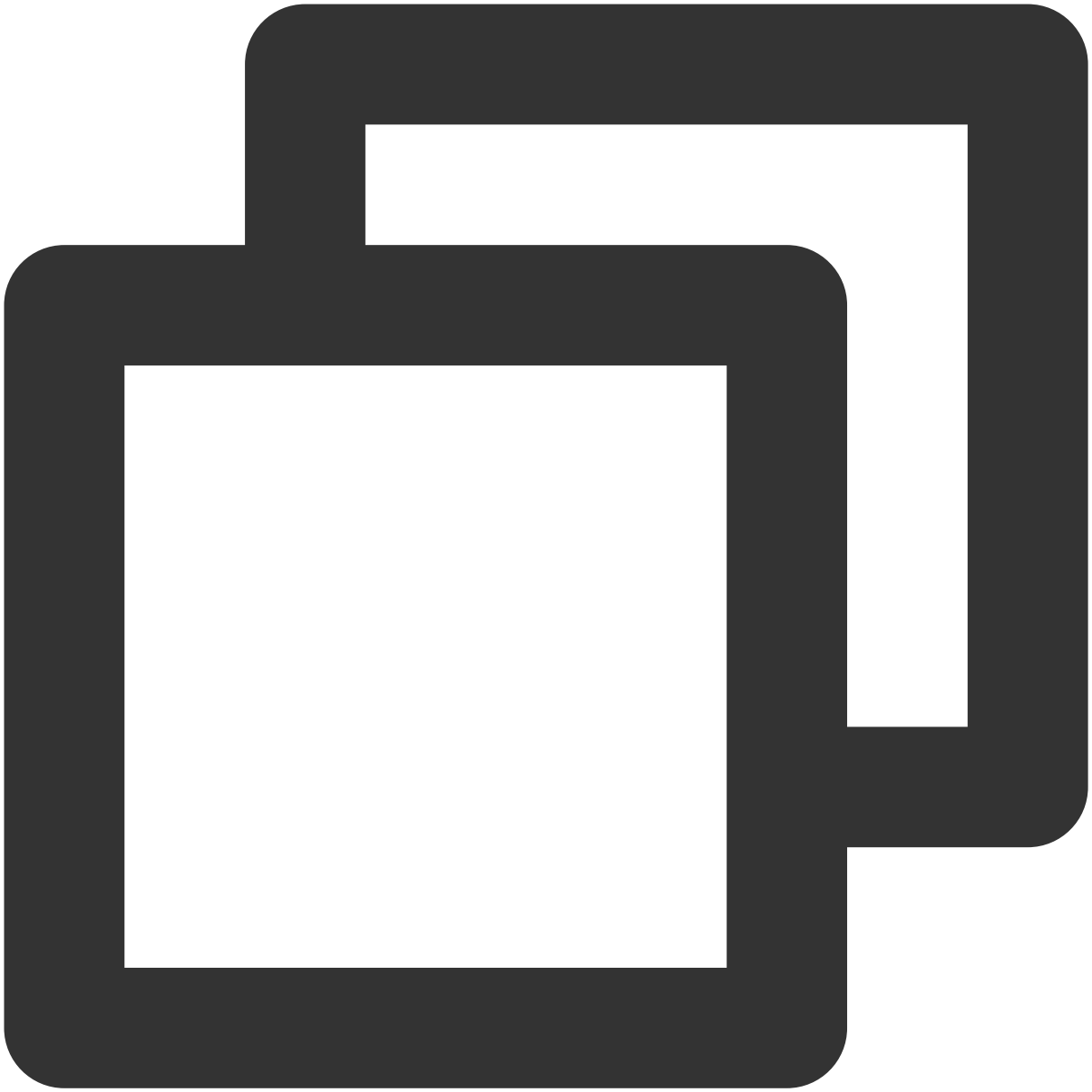


```
//接收和处理房间内其他人发送的消息
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:(NSInteger)cmdId seq:(UIn
{
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId = 1消息
```

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

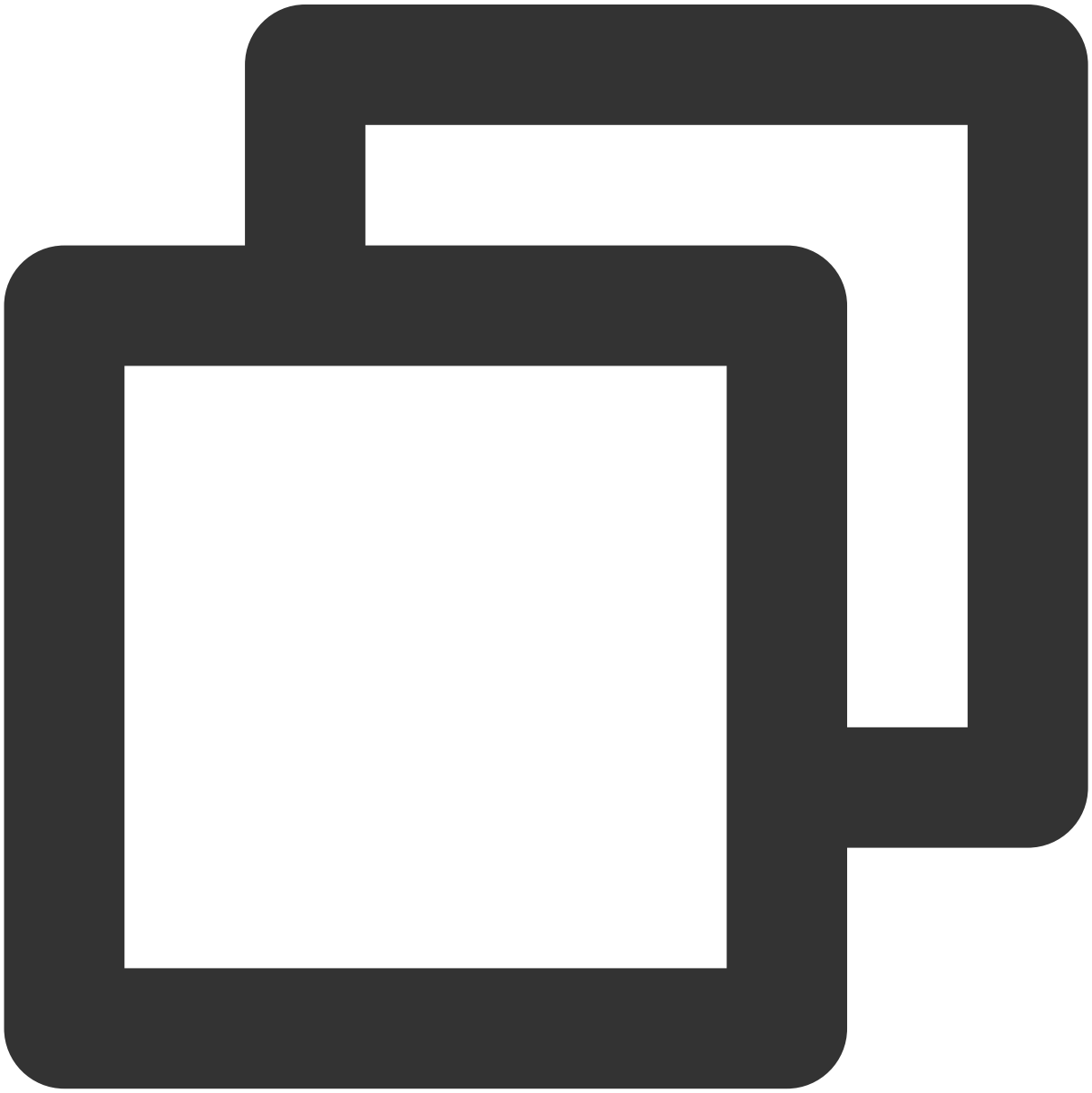


```
//继承 TRTCCloudListener，实现 onRecvCustomCmdMsg 方法接收和处理房间内其他人发送的消息
public void onRecvCustomCmdMsg(String userId, int cmdId, int seq, byte[] message) {
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId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

}
```



```
// 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
{
    // 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;
        }
    }
```



```
// 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
{
    // 接收到 userId 发送的消息
    switch (cmdId)  // 发送方和接收方协商好的cmdId
    {
    case 0:
        // 处理cmdId = 0消息
        break;
    case 1:
        // 处理cmdId = 1消息
        break;
    case 2:
        // 处理cmdId = 2消息
        break;
    default:
        break;
    }
}
```

使用限制

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

自定义消息会被云广播给房间内所有用户，所以每秒最多能发送 30 条消息。

每个消息包（即 data 的大小）最大为 1 KB，超过则很有可能会被中间路由器或者服务器丢弃。

每个客户端每秒最多能发送总计 8 KB 数据，也就是如果每个数据包都是 1KB，那么每秒钟您最多只能发送 8 个数据包。

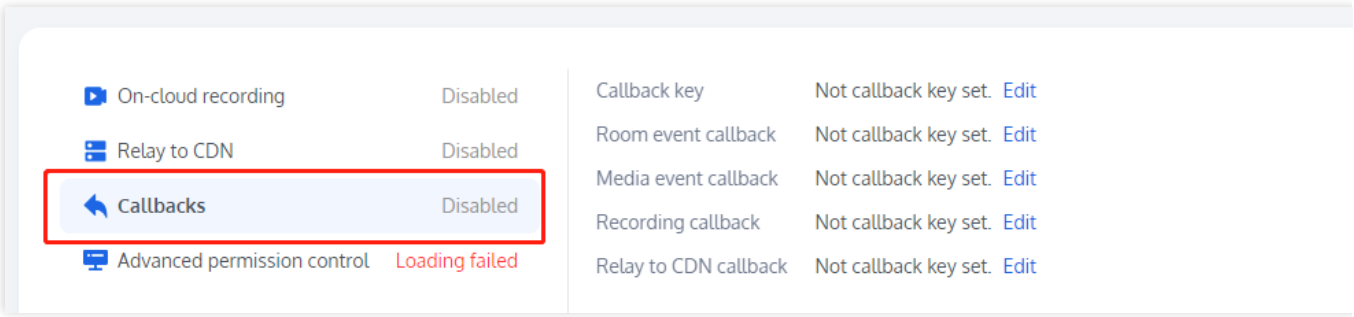
监听服务端事件回调
房间与媒体回调

最近更新时间：2024-08-07 10:53:53

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）以及录制事件组的一些事件（云端录制功能回调事件说明请参见 实现云端录制与回放），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

Tencent RTC 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。



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}。

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

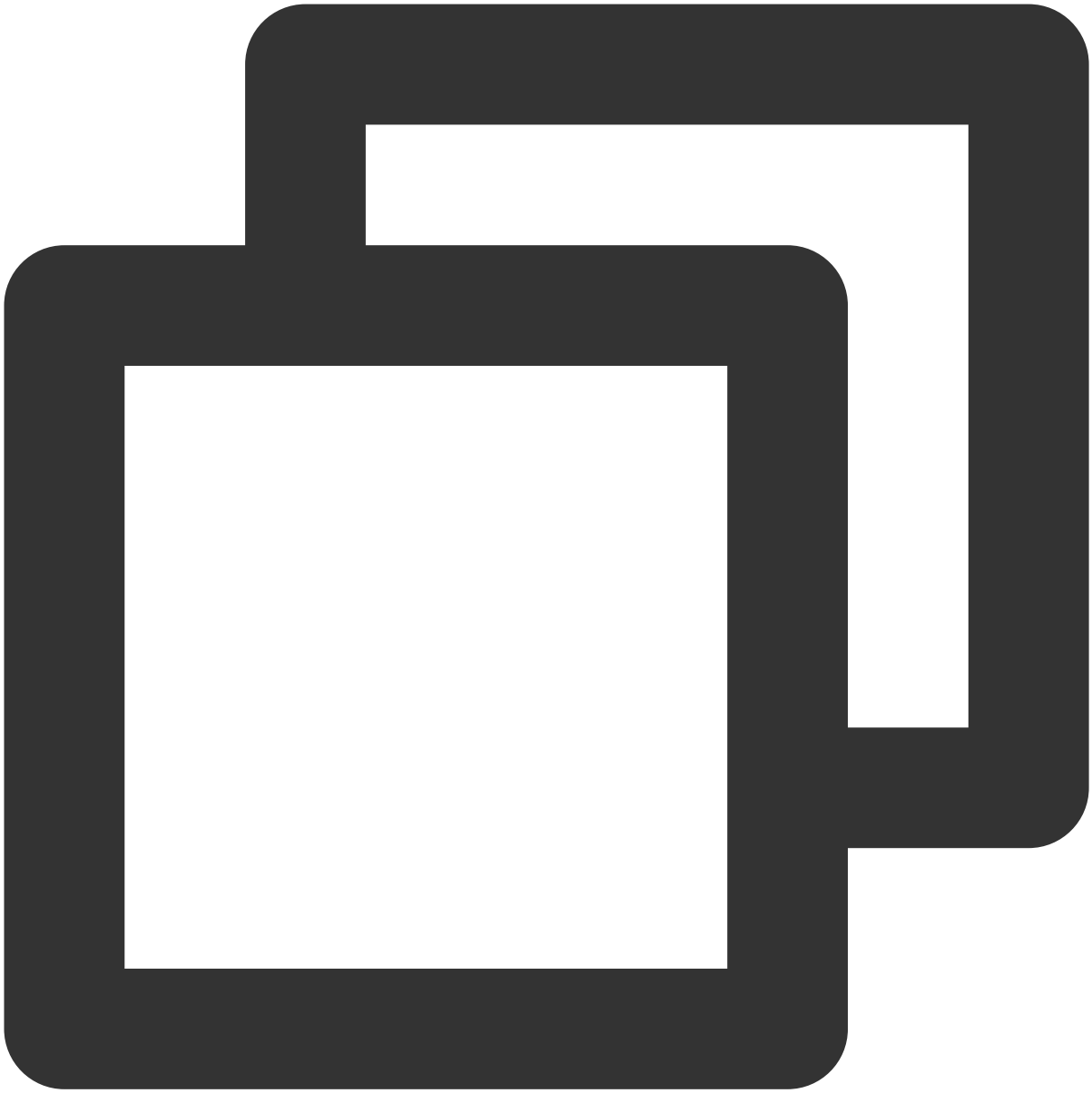
字段名	值	含义
EVENT_TYPE_CREATE_ROOM	101	创建房间
EVENT_TYPE_DISMISS_ROOM	102	解散房间
EVENT_TYPE_ENTER_ROOM	103	进入房间
EVENT_TYPE_EXIT_ROOM	104	退出房间
EVENT_TYPE_CHANGE_ROLE	105	切换角色
EVENT_TYPE_START_VIDEO	201	开始推送视频数据
EVENT_TYPE_STOP_VIDEO	202	停止推送视频数据
EVENT_TYPE_START_AUDIO	203	开始推送音频数据
EVENT_TYPE_STOP_AUDIO	204	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

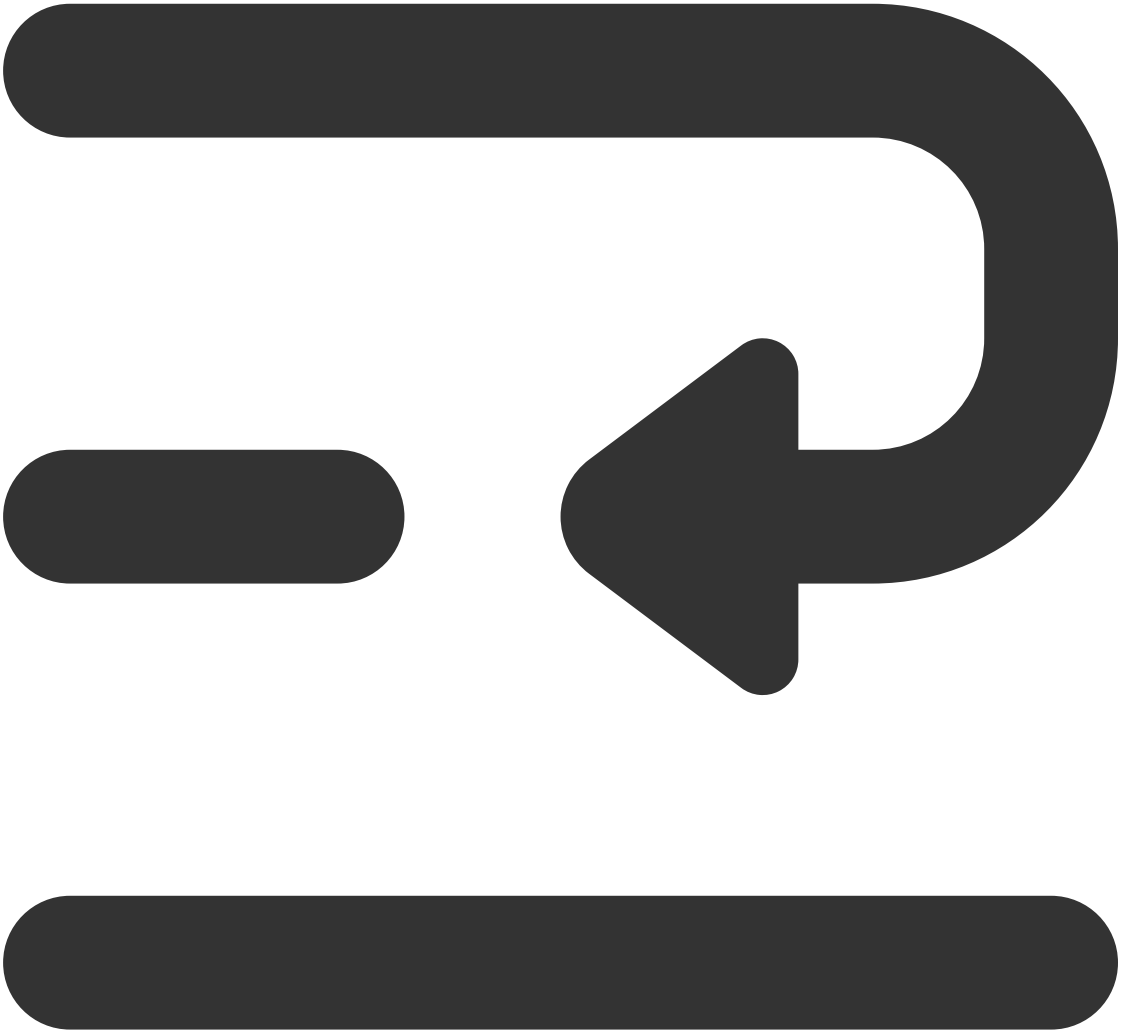
退出房间只会回调104事件，不会回调202跟204事件。104事件相当于包含了202和204事件。手动关闭视频/音频，才会回调202/204事件。

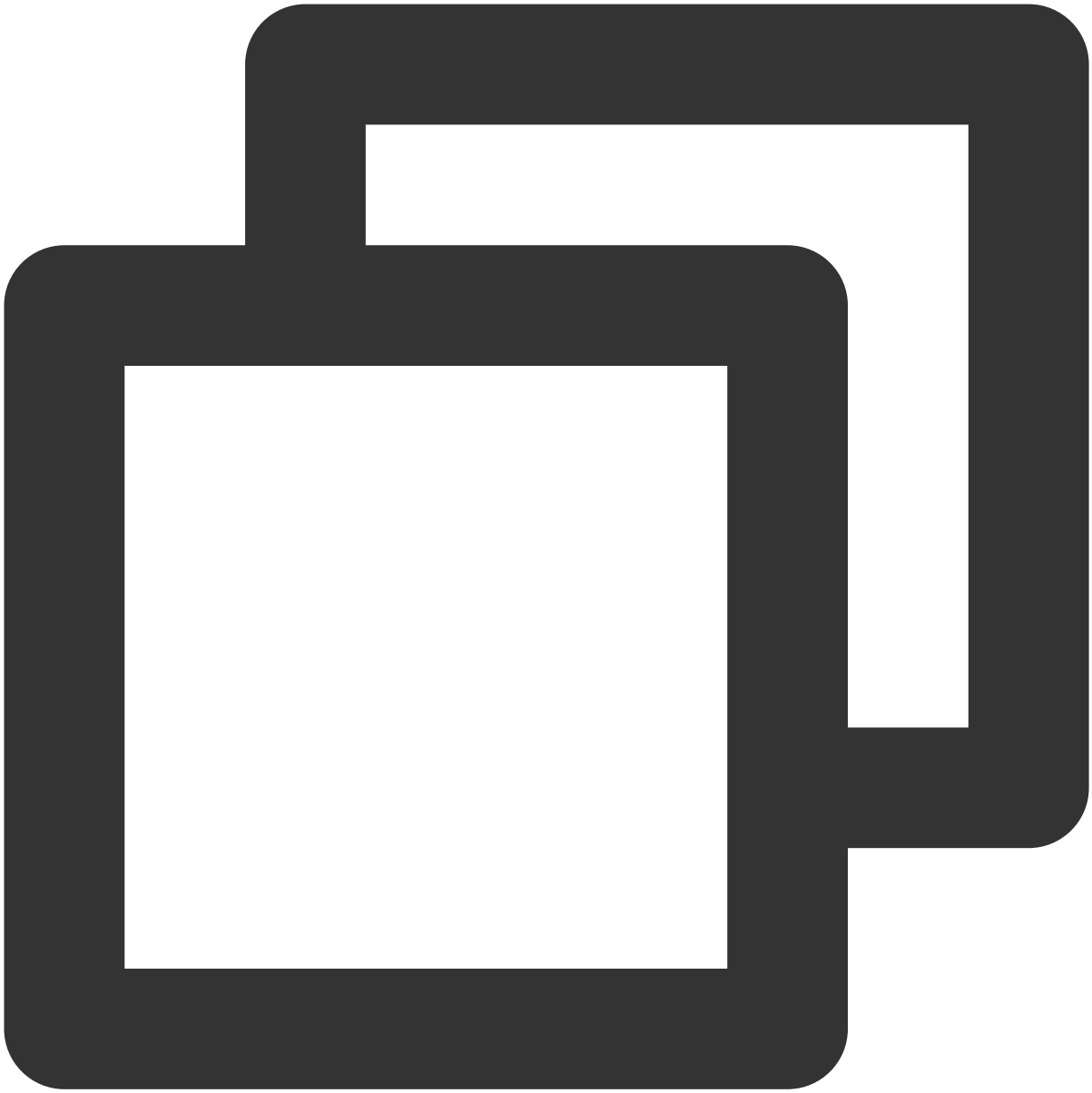
事件回调示例

101
102
103
104
105
201
202
203
204
205
206

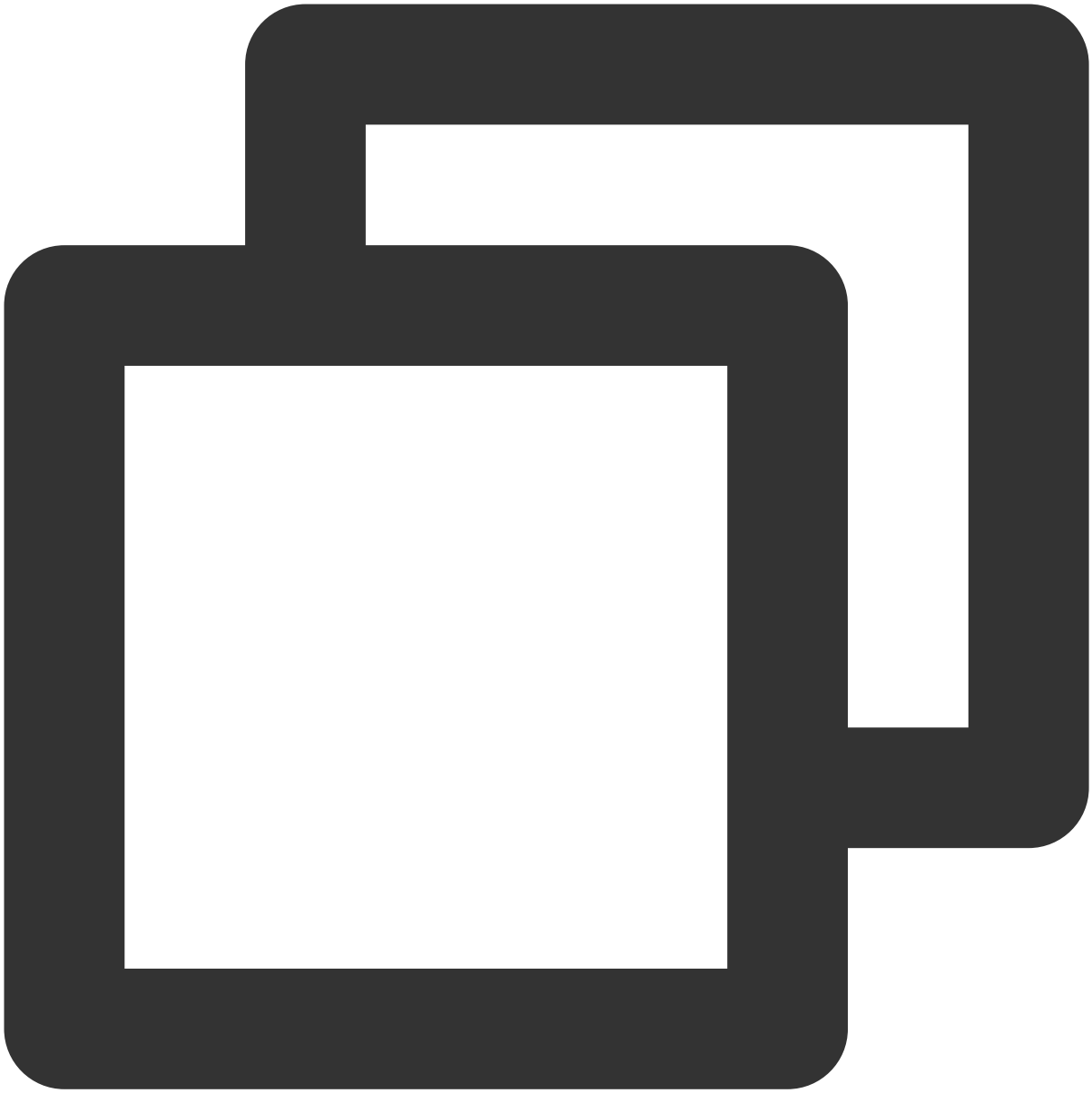


```
{       "EventGroupId": 1,
        "EventType":    101,
        "CallbackTs":   1687770730166,
        "EventInfo":    {
                "RoomId":       12345,
        "EventTs":      1687770730,
        "EventMsTs":    1687770730160,
        "UserId":       "test"
        }
}
```



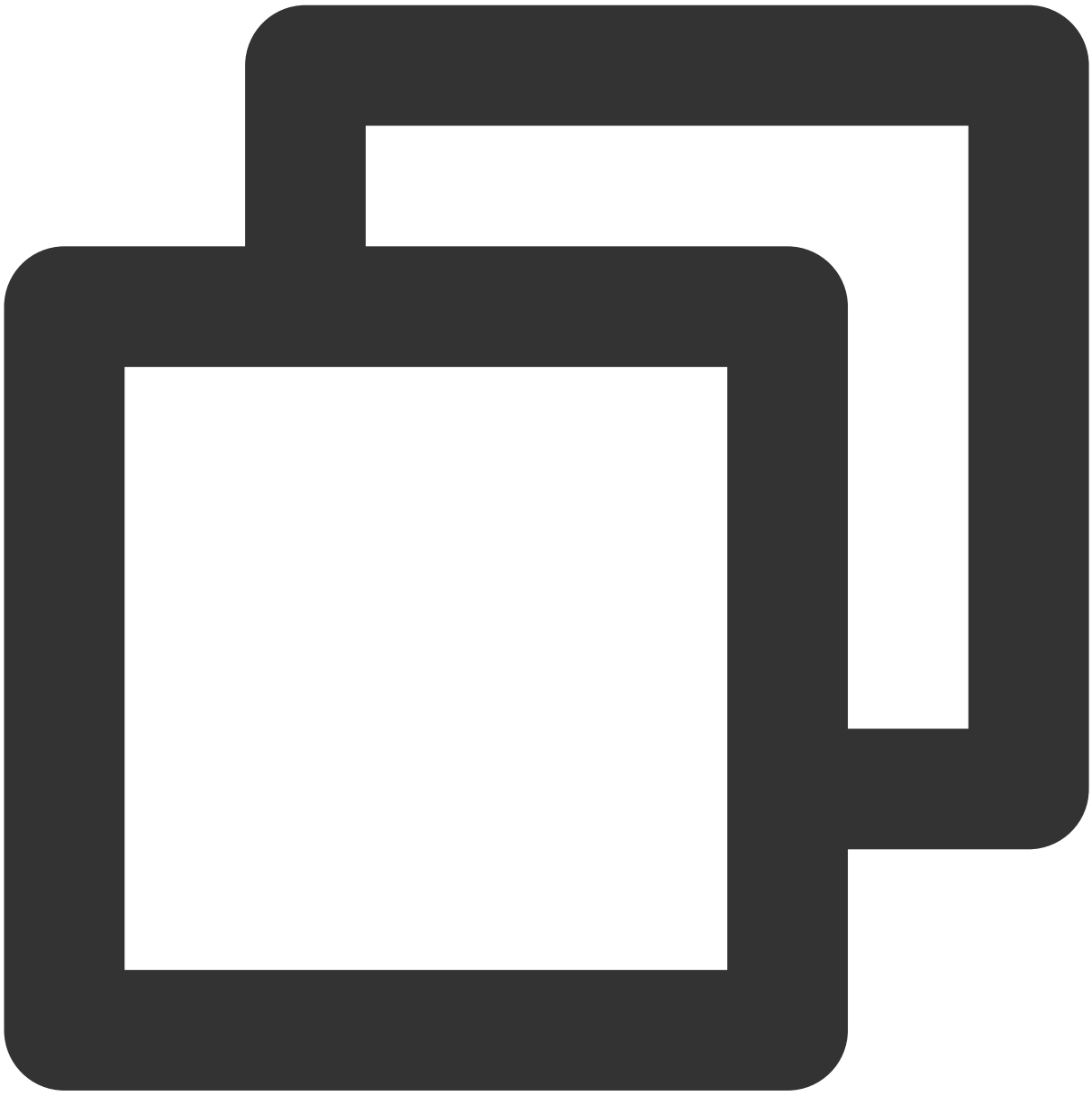


```
{
    "EventGroupId":     1,
    "EventType":        102,
    "CallbackTs":       1687771618531,
    "EventInfo":        {
        "RoomId":       "12345",
        "EventTs":      1687771618,
        "EventMsTs":    1687771618457
     }
}
```



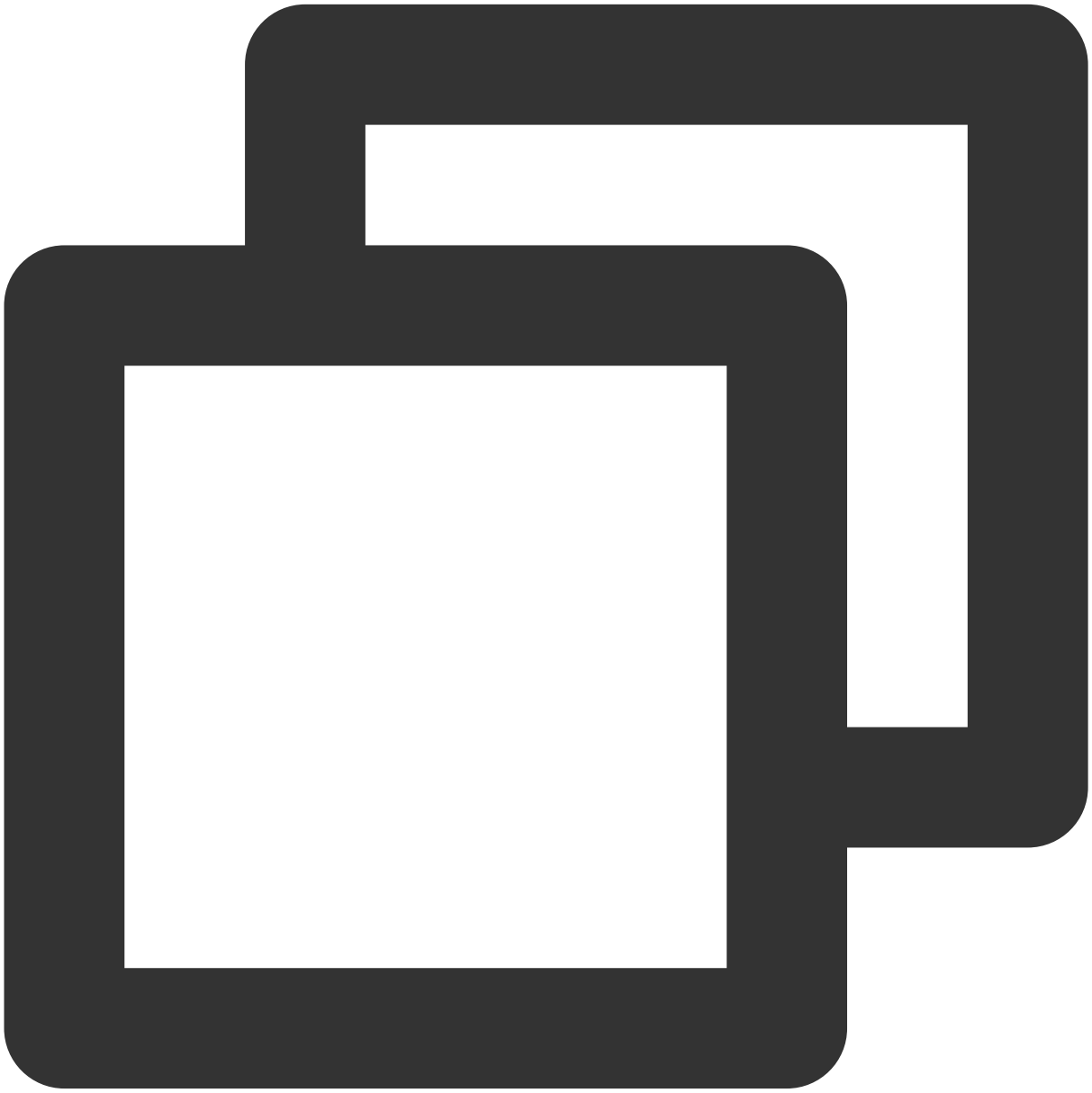
```
{
        "EventGroupId": 1,
        "EventType":    103,
        "CallbackTs":   1687770731932,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687770731,
                "EventMsTs":    1687770731831,
                "UserId":       "test",
   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

```
                "UserType":     3,
                "Reason":       1
        }
}
```



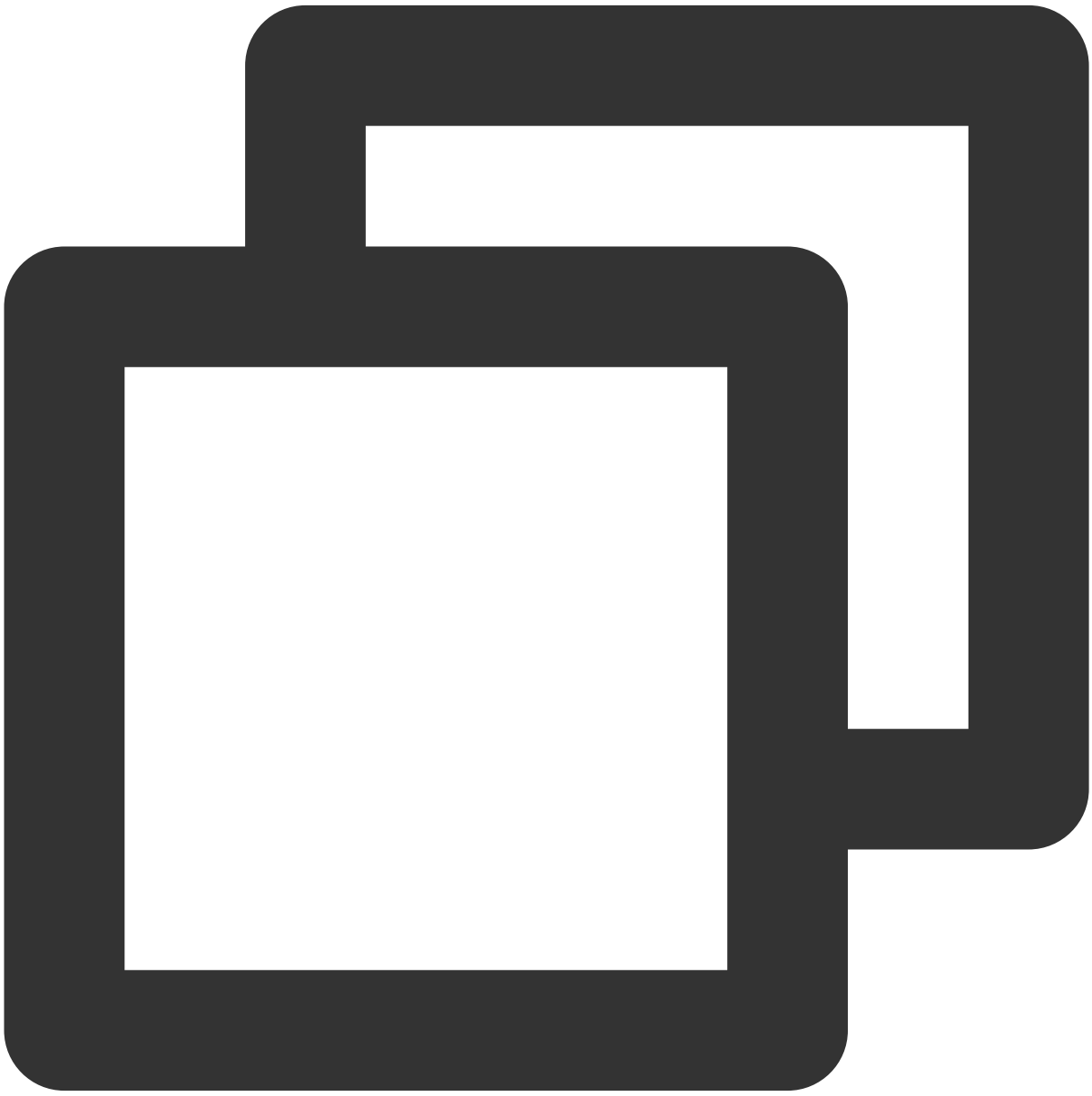
```
{
        "EventGroupId": 1,
        "EventType":    104,
        "CallbackTs":   1687770731922,
        "EventInfo":    {
                "RoomId":       12345,
```

```
                "EventTs":      1687770731,
                "EventMsTs":    1687770731898,
                "UserId":       "test",
                "Role": 20,
                "Reason":       1
        }
}
```



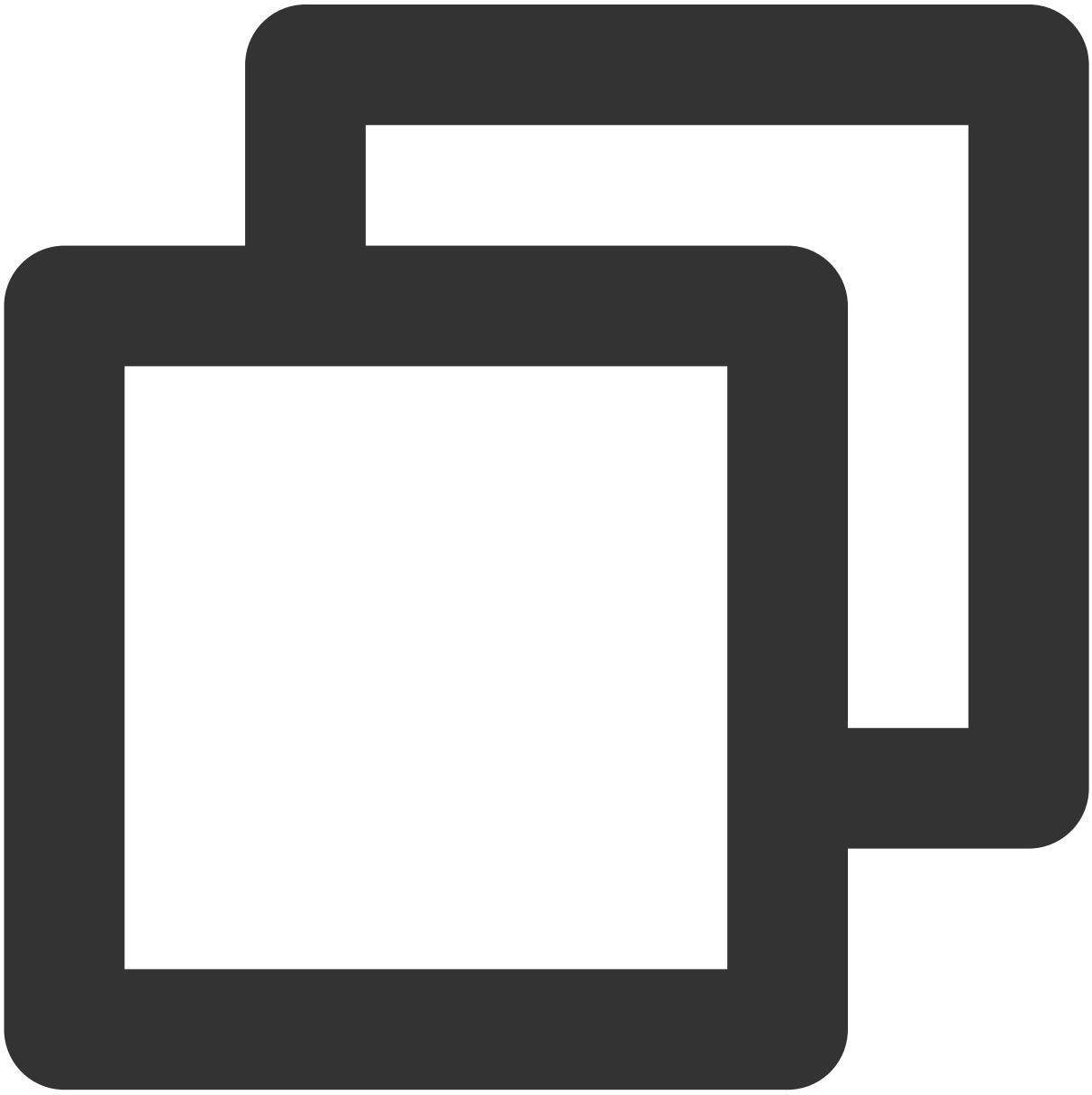
```
{
        "EventGroupId": 1,
        "EventType":    105,
```

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

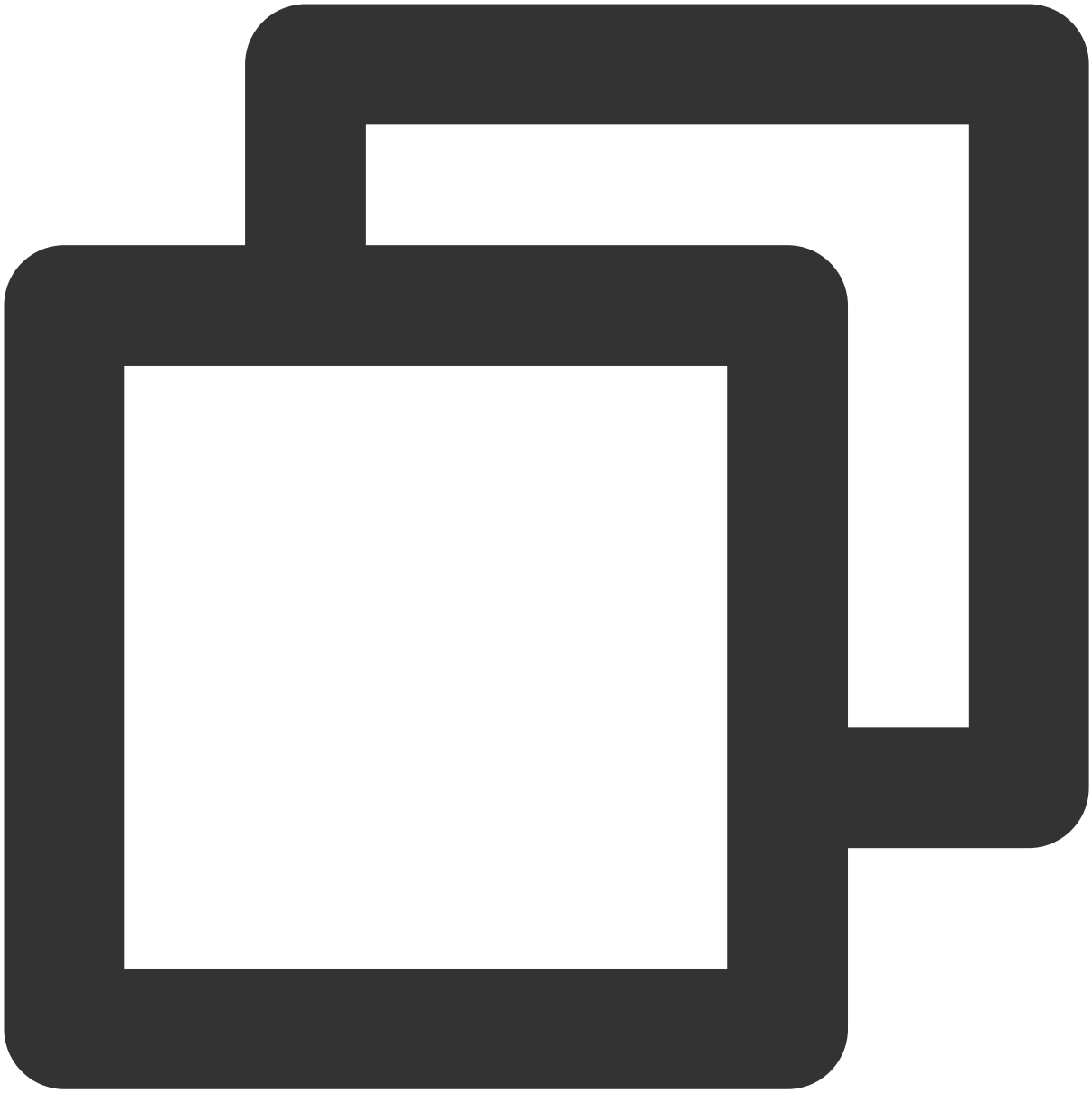


```
{
```

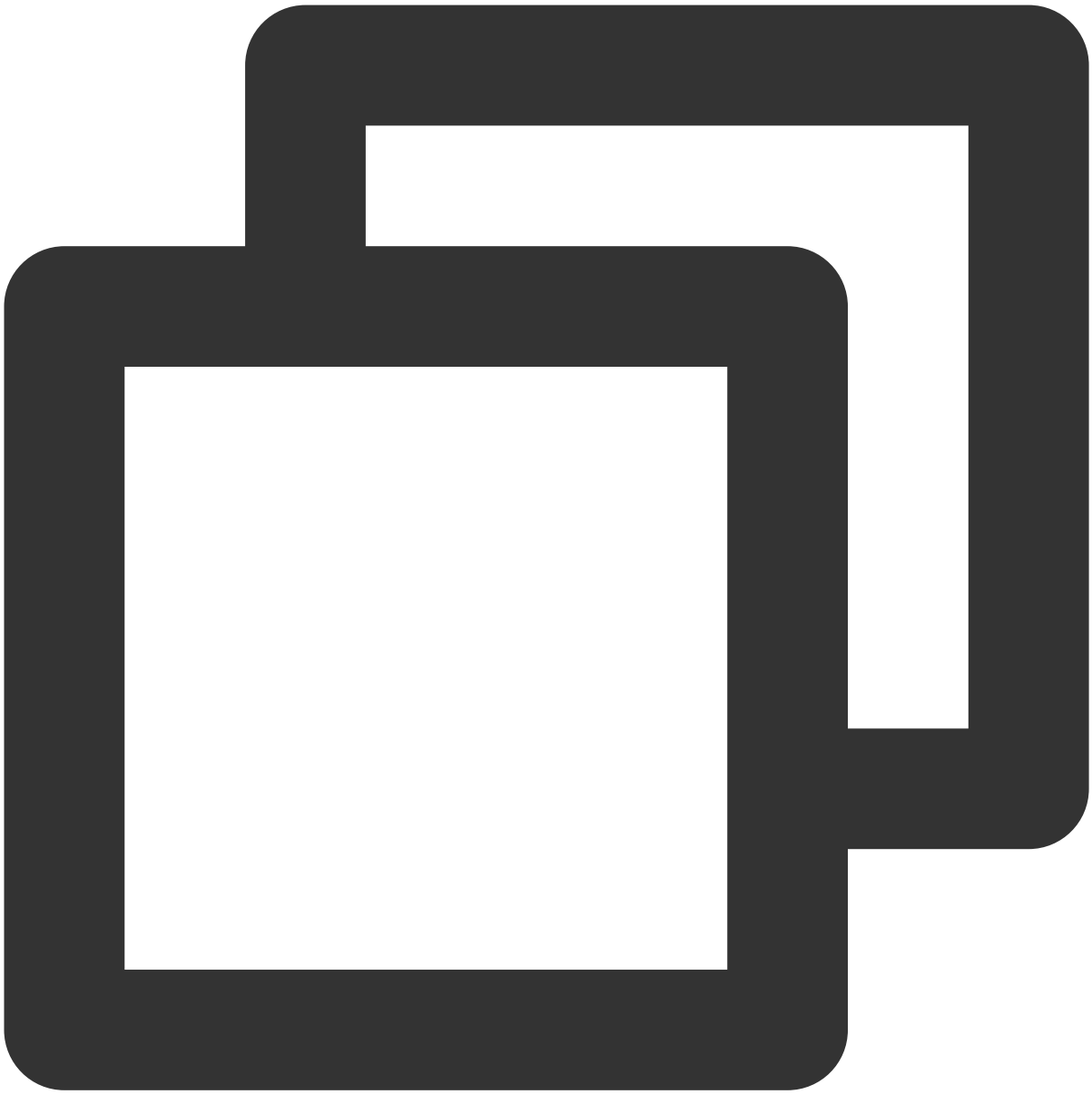
```
        "EventGroupId": 2,
        "EventType":    201,
        "CallbackTs":   1687771803198,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687771803,
                "EventMsTs":    1687771803192,
                "UserId":       "test"
      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



```
{
        "EventGroupId": 2,
        "EventType":    202,
        "CallbackTs":   1687771919458,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687771919,
                "EventMsTs":    1687771919447,
                "UserId":       "test",
                "Reason":       0
        }
}
```

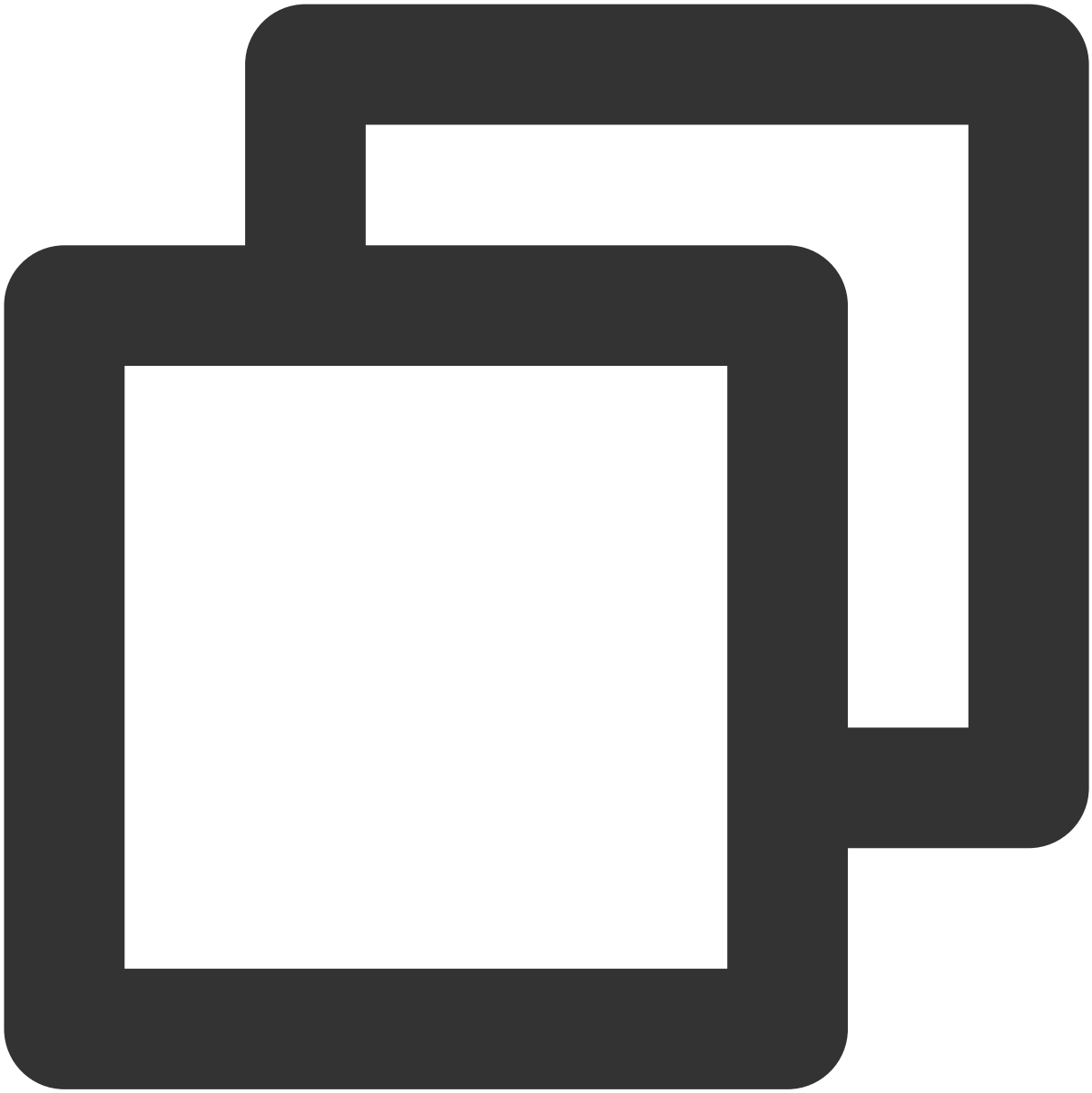


```
{
        "EventGroupId": 2,
        "EventType":    203,
        "CallbackTs":   1687771869377,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687771869,
                "EventMsTs":    1687771869365,
                "UserId":       "test"
    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



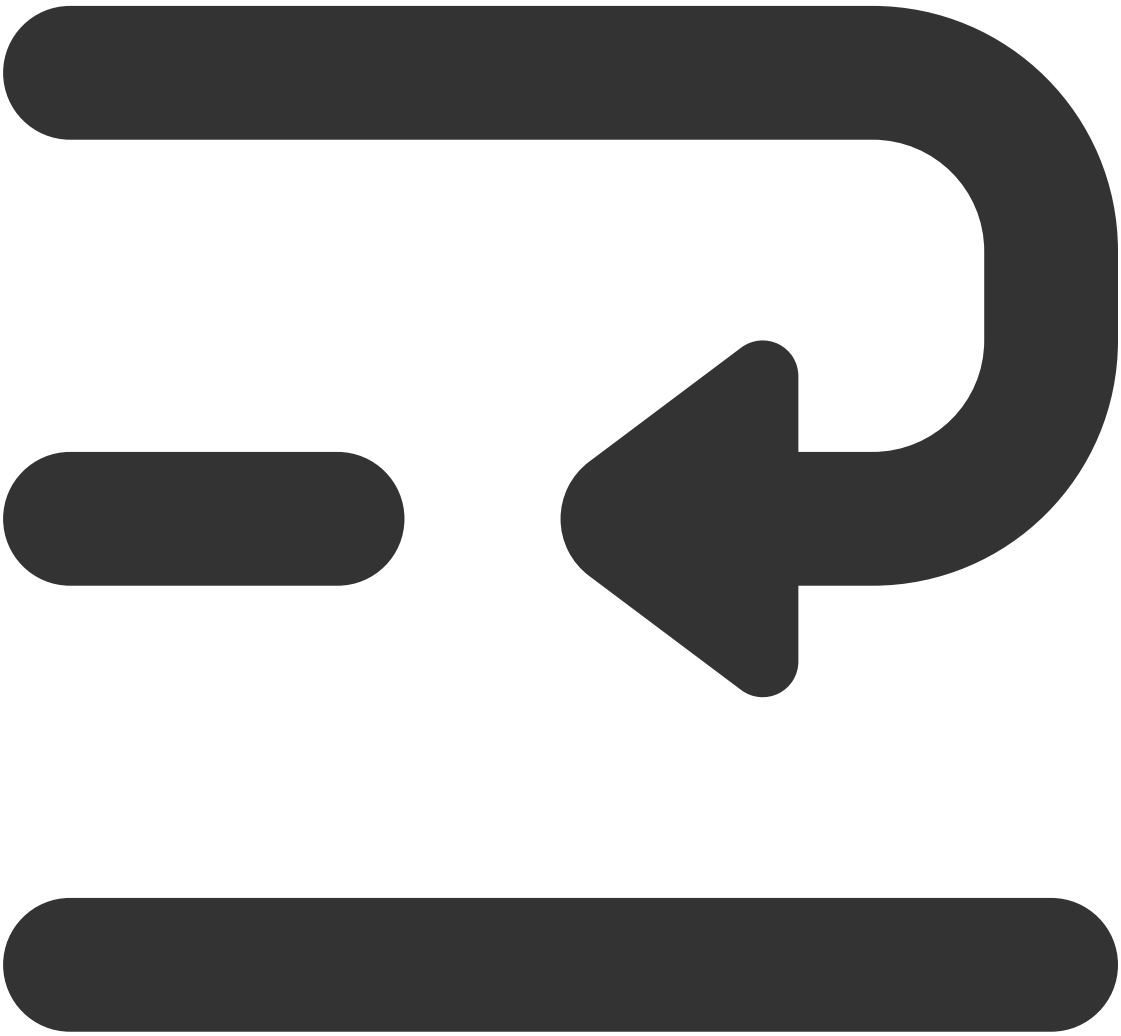
```
{
        "EventGroupId": 2,
        "EventType":    204,
        "CallbackTs":   1687770732498,
        "EventInfo":    {
                "RoomId":       12345,
                "EventTs":      1687770732,
                "EventMsTs":    1687770732383,
                "UserId":       "test",
   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

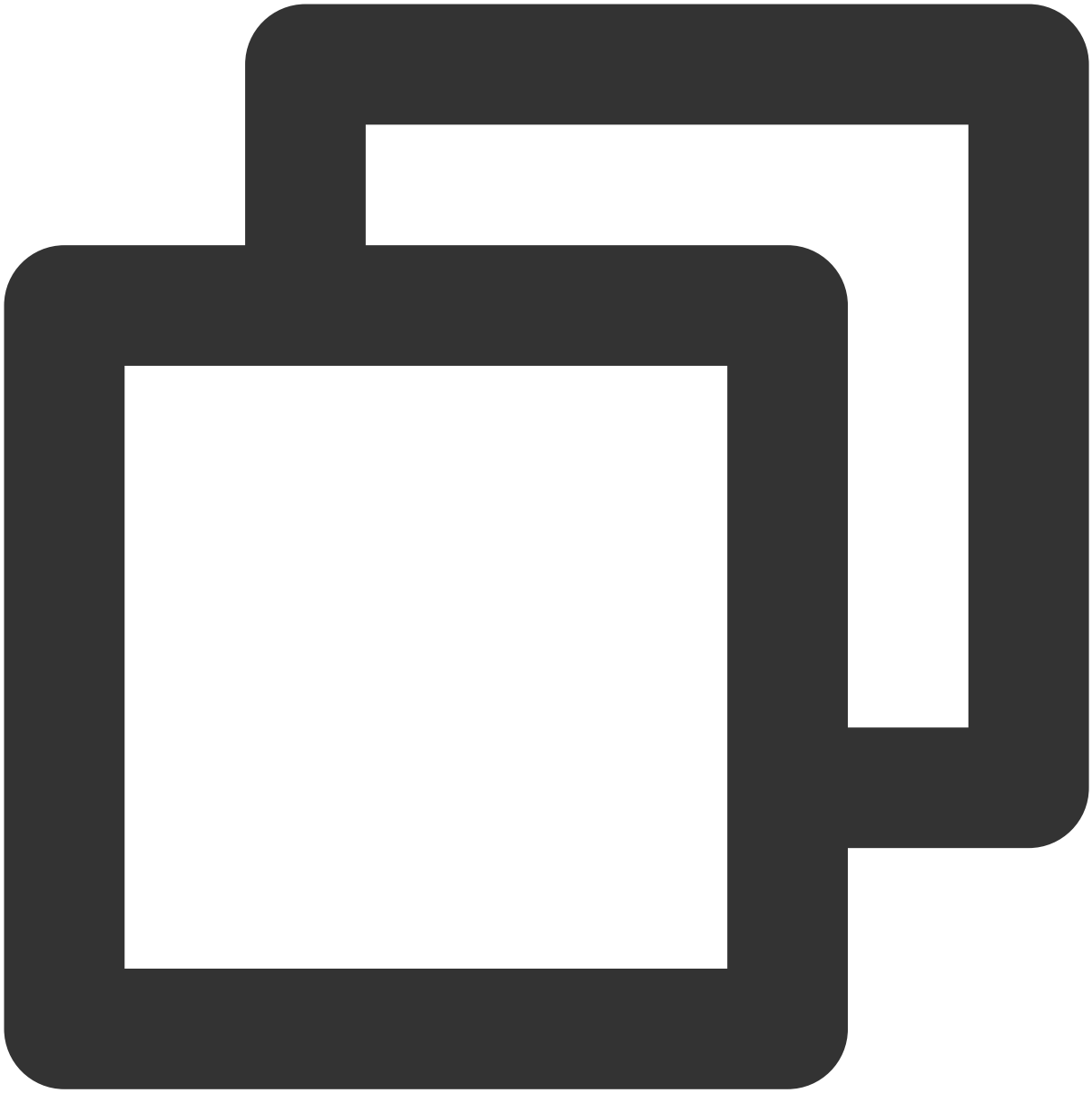
```
        }
}
```



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

```
                "UserId":       "test"
        }
}
```





```
{
        "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：进退房时携带）

注意：

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

角色类型

字段名	值	含义
MEMBER_TRTC_ANCHOR	20	主播
MEMBER_TRTC_VIEWER	21	观众

终端类型

字段名	值	含义
TERMINAL_TYPE_WINDOWS	1	Windows 端
TERMINAL_TYPE_ANDROID	2	Android 端

TERMINAL_TYPE_IOS	3	iOS 端
TERMINAL_TYPE_LINUX	4	Linux 端
TERMINAL_TYPE_OTHER	100	其他

用户类型

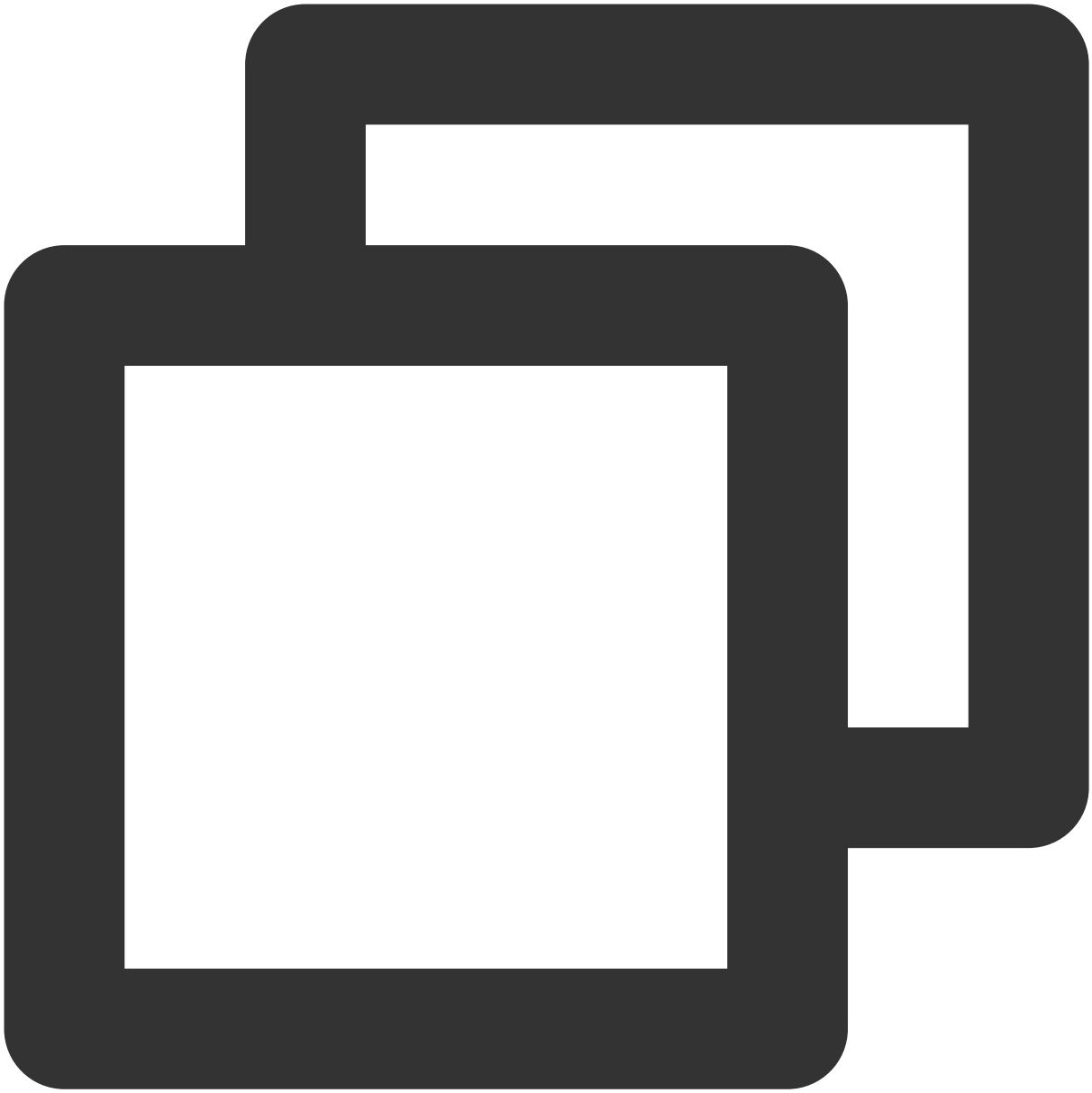
字段名	值	含义
USER_TYPE_WEBRTC	1	webrtc
USER_TYPE_APPLET	2	小程序
USER_TYPE_NATIVE_SDK	3	Native SDK

具体原因

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

计算签名

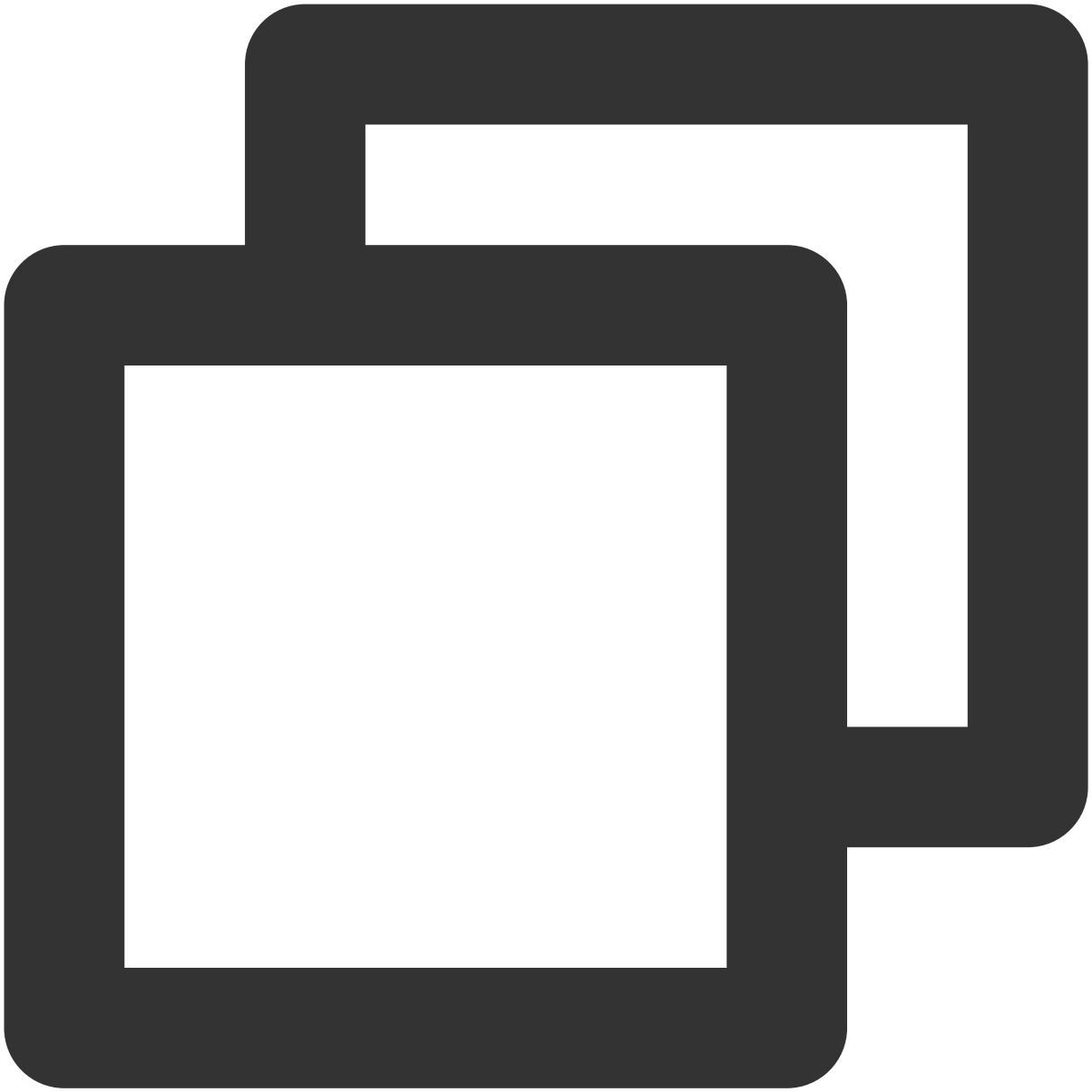
签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：



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

注意：

body 为您收到回调请求的原始包体，不要做任何转化，示例如下：



```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

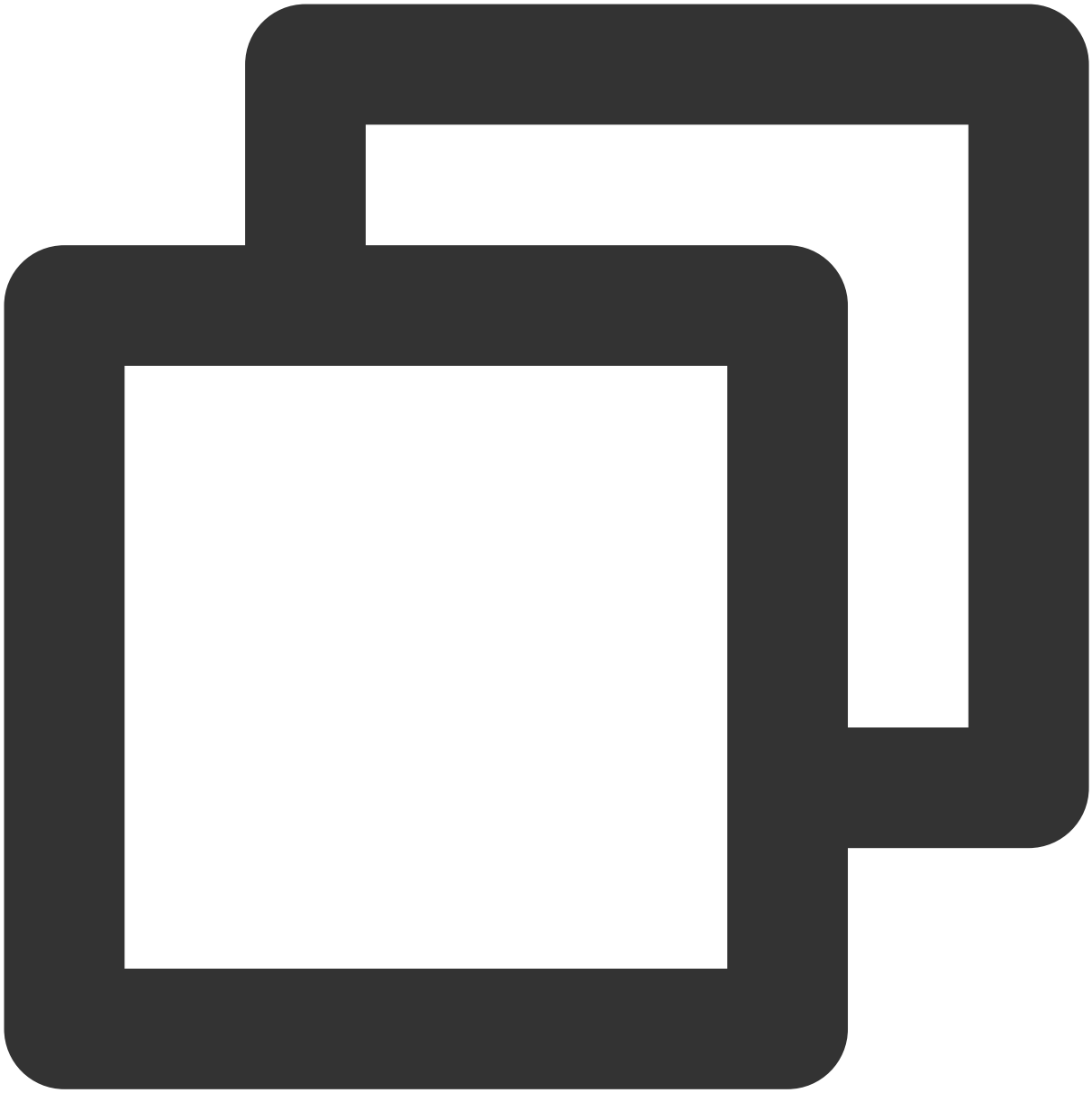
签名校验示例

Java

Python

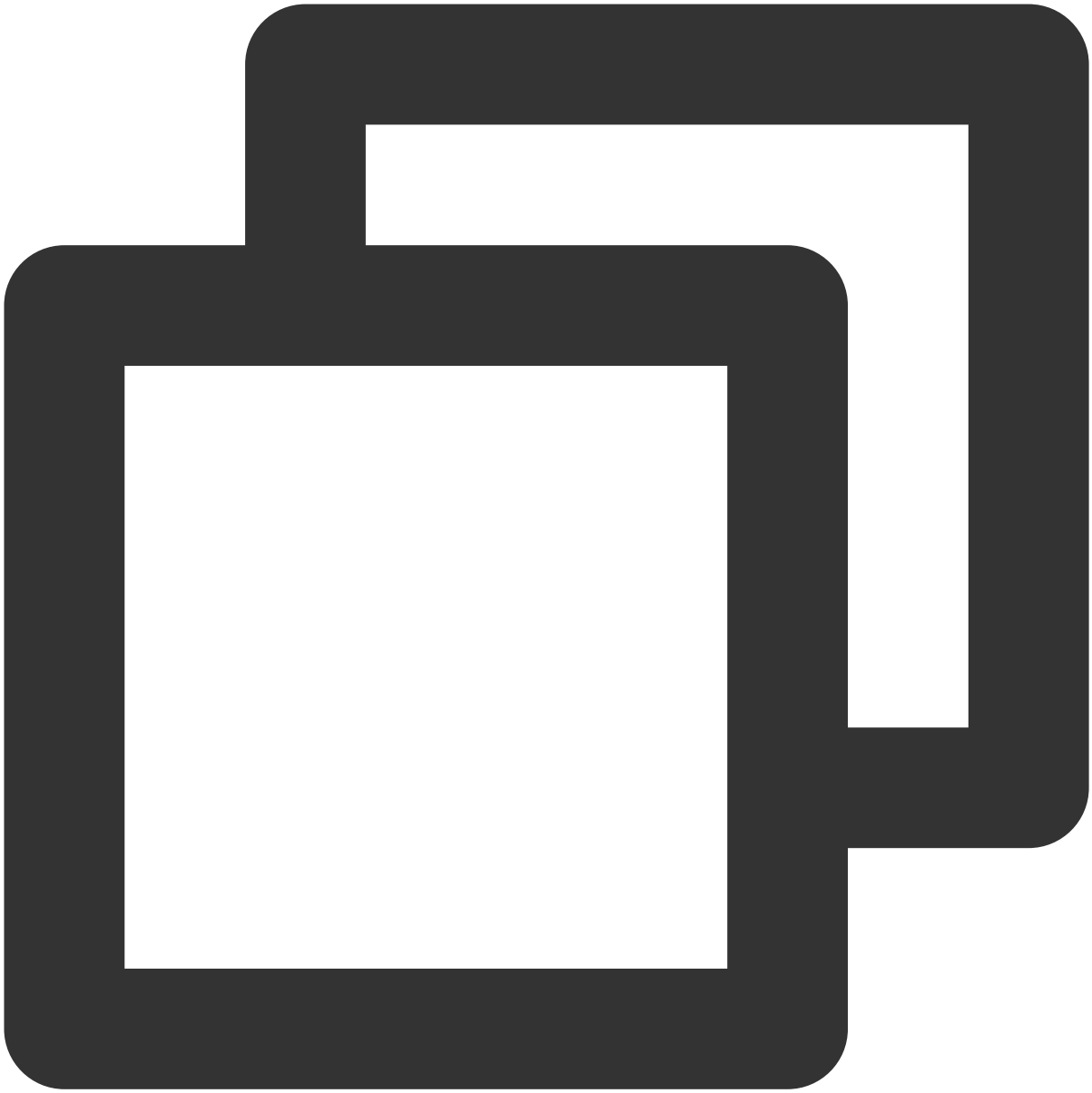
PHP

Golang



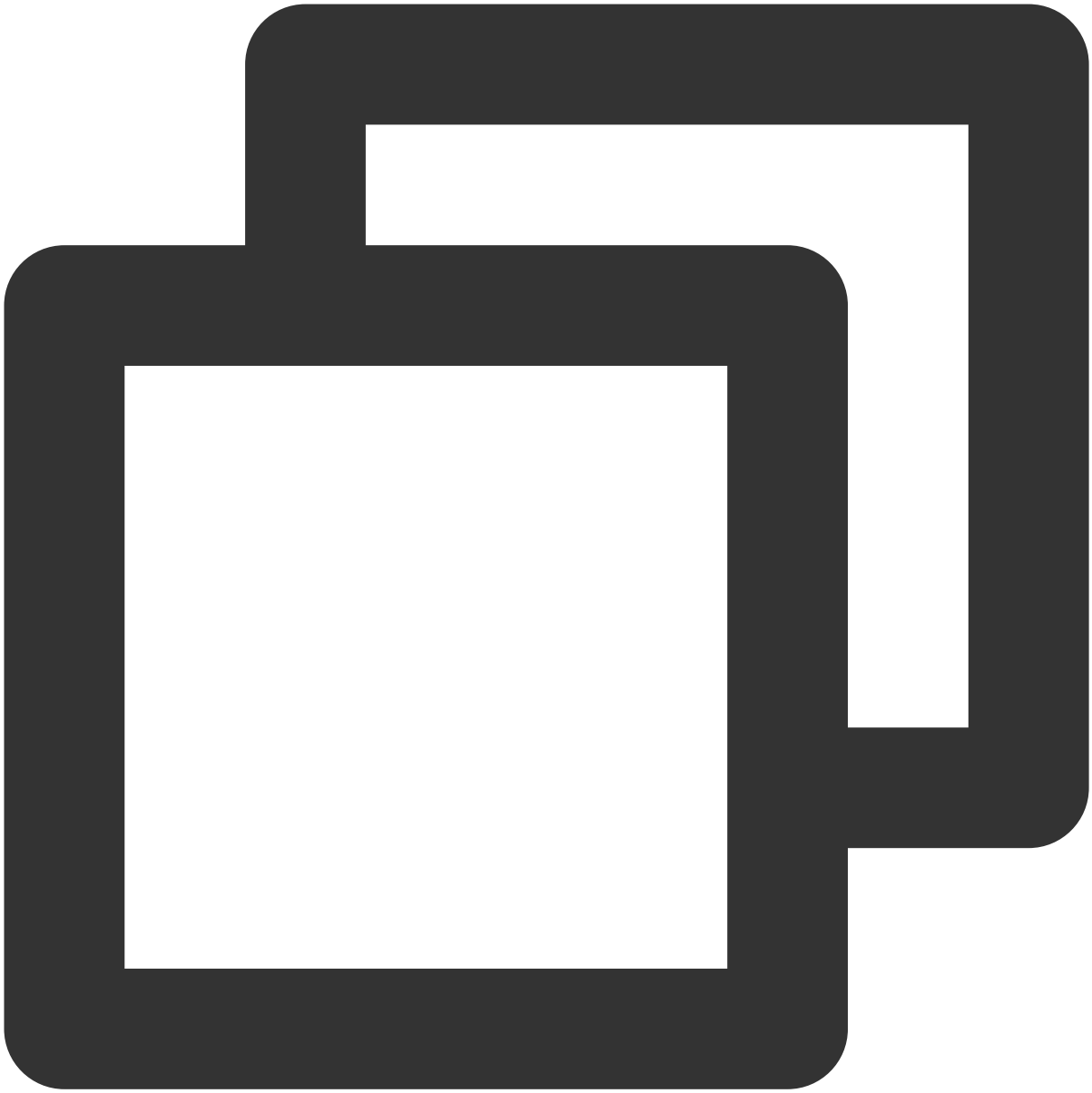
```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#    key：控制台配置的密钥key
//#    body：腾讯云回调返回的body体
//#    sign：腾讯云回调返回的签名值sign
//# 返回值：
//#    Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#    Info：成功/失败信息
```

```
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\
        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'}");
        }
    }
}
```



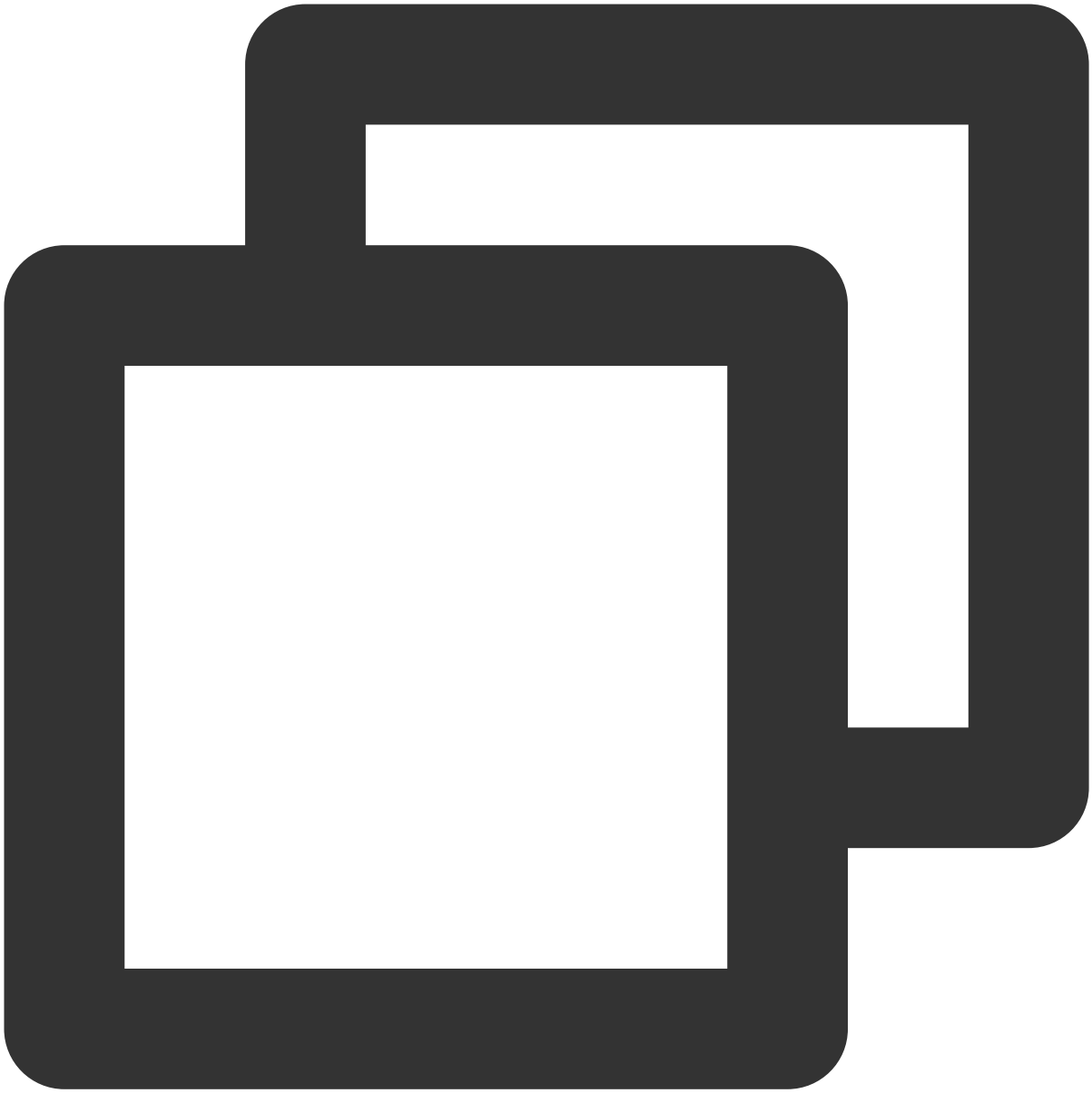
```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 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'
    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,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```



```
<?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\\"Callbac

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



```
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\
    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))
}
```

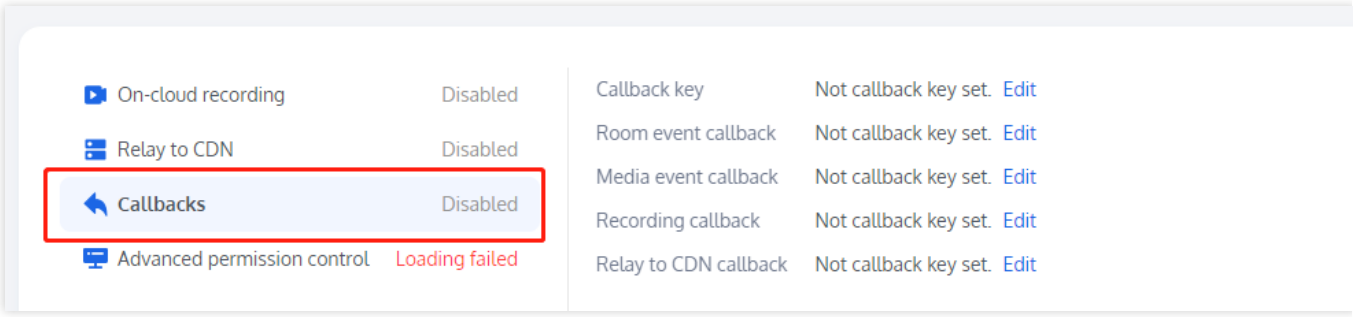
旁路转推回调

最近更新时间：2024-08-07 10:53:53

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

Tencent RTC 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。



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

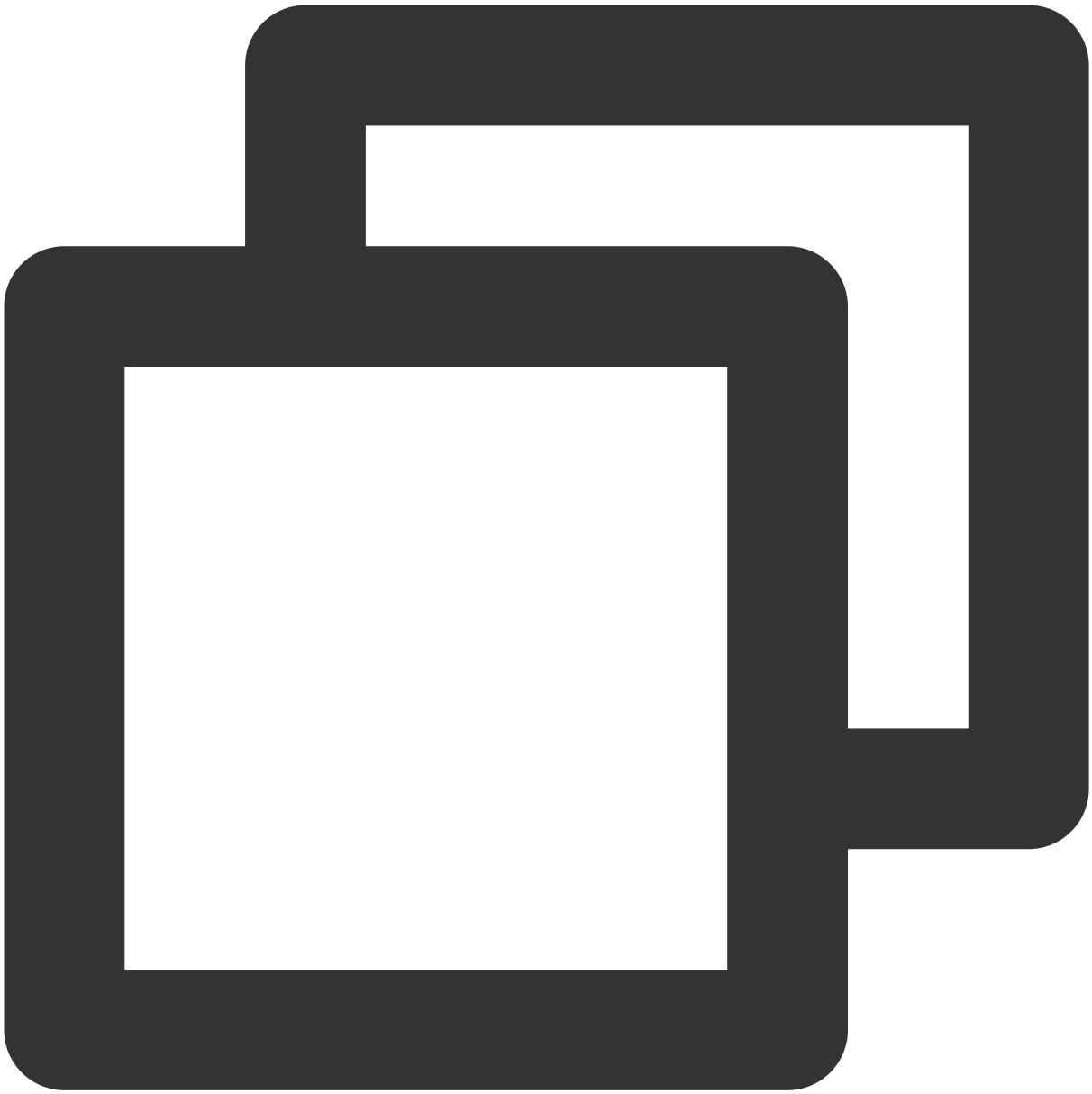
事件回调消息以 HTTP/HTTPS POST 请求发送给您的服务器，其中：

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/Number	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	Number	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	Number	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 调 `PUBLISH_CDN_STREAM_STATE_RUNNING` ，或 调 `PUBLISH_CDN_STREAM_STATE_FAILURE` 。您 到 `PUBLISH_CDN_STREAM_STATE_FAILURE` 的时 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 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 调 `PUBLISH_CDN_STREAM_STATE_RUNNING` ，或 调 `PUBLISH_CDN_STREAM_STATE_FAILURE` 。通 如 果 `PUBLISH_CDN_STREAM_STATE_RECOVERING` 和 时间内交替出现，您需要检查下是否存在多任务使用相
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， 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PUBL

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

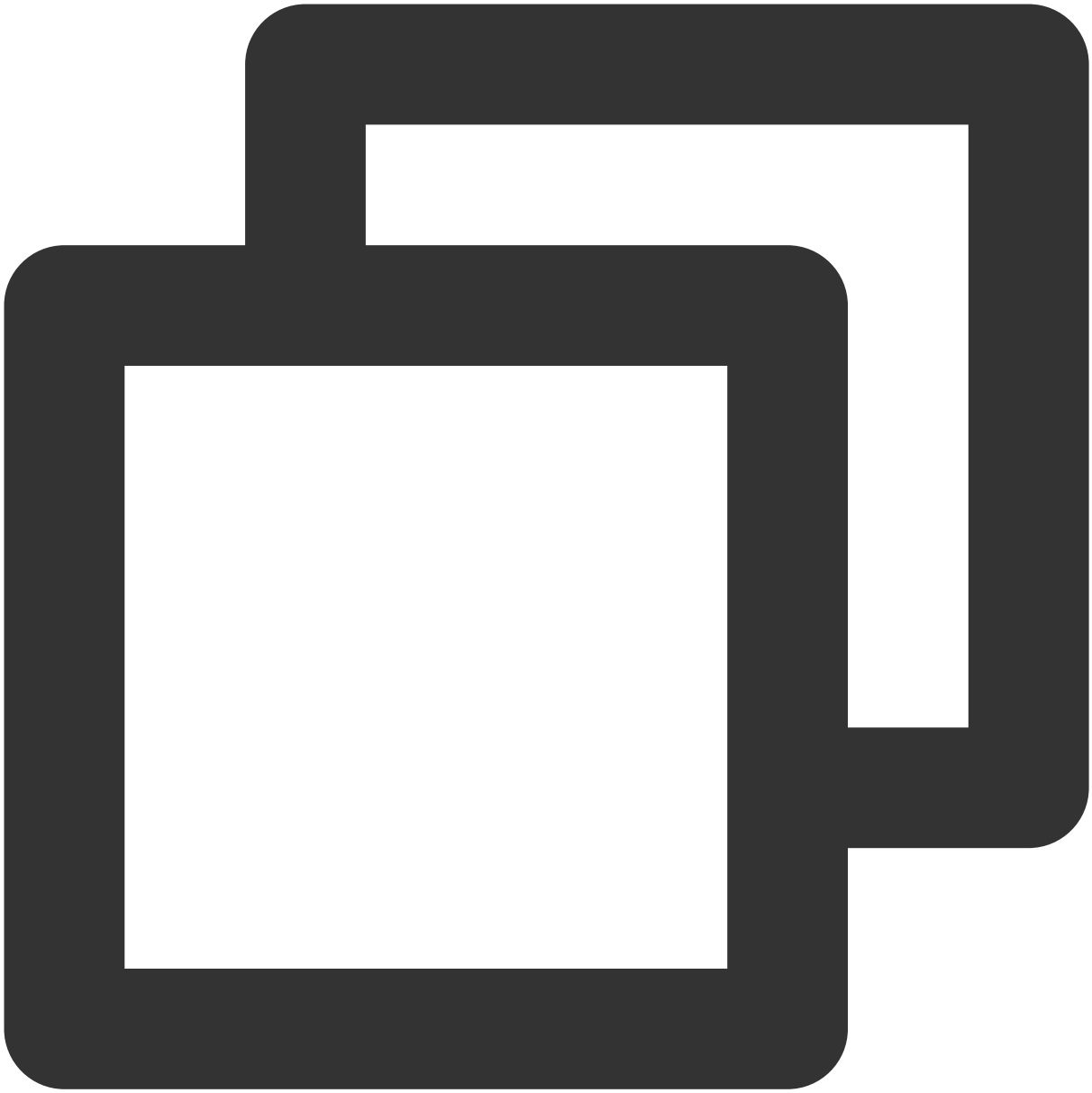
`PUBLISH_CDN_STREAM_STATE_RUNNING` -> `PUBLISH_CDN_STREAM_STATE_RECOVERING` -> `PUBLISH_CDN_STREAM_STATE_FAILURE` -> `PUBLISH_CDN_STREAM_STATE_IDLE`

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

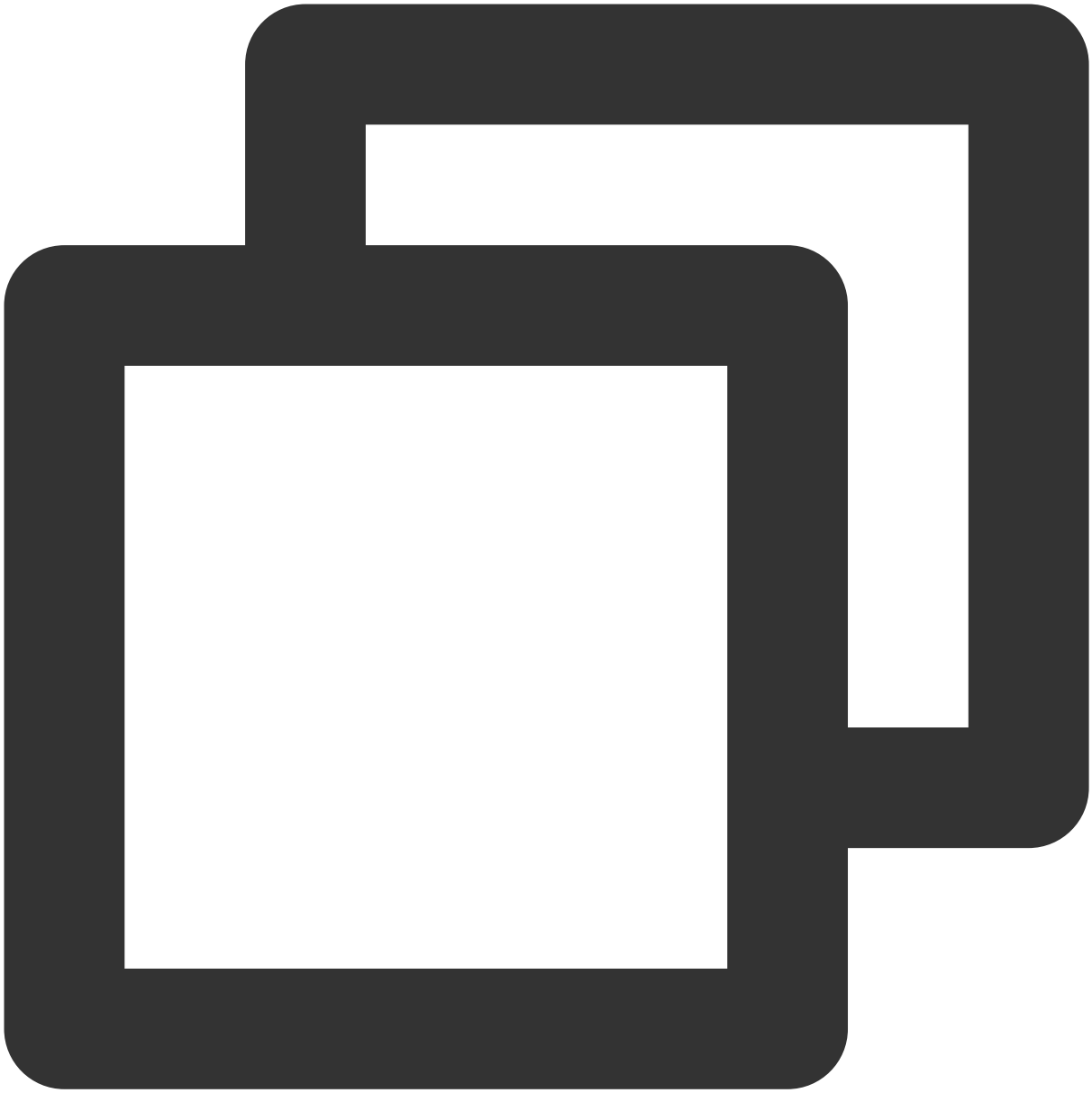
签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：



```
//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\\"Callback
```

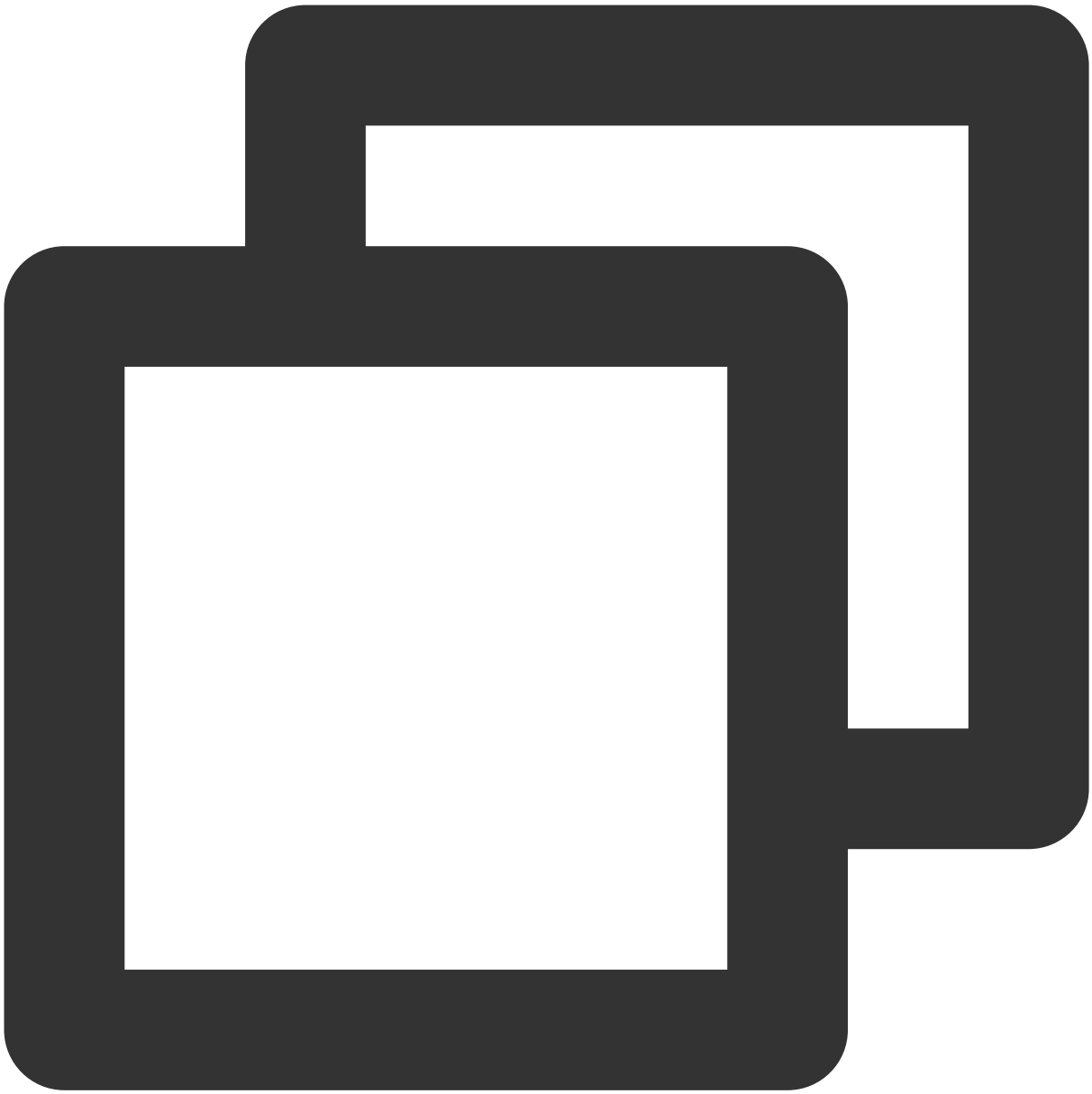
签名校验示例

Java

Python

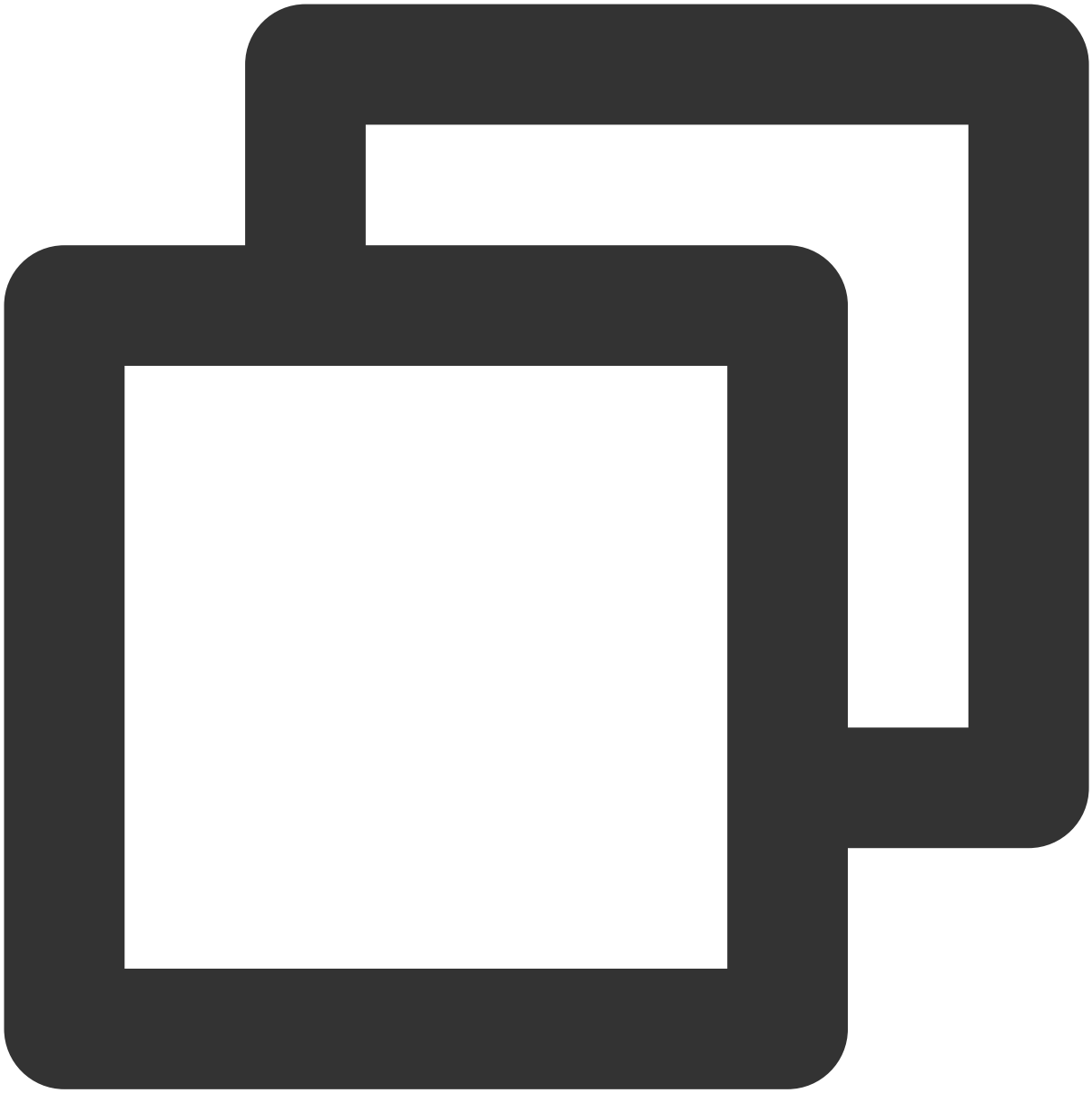
PHP

Golang



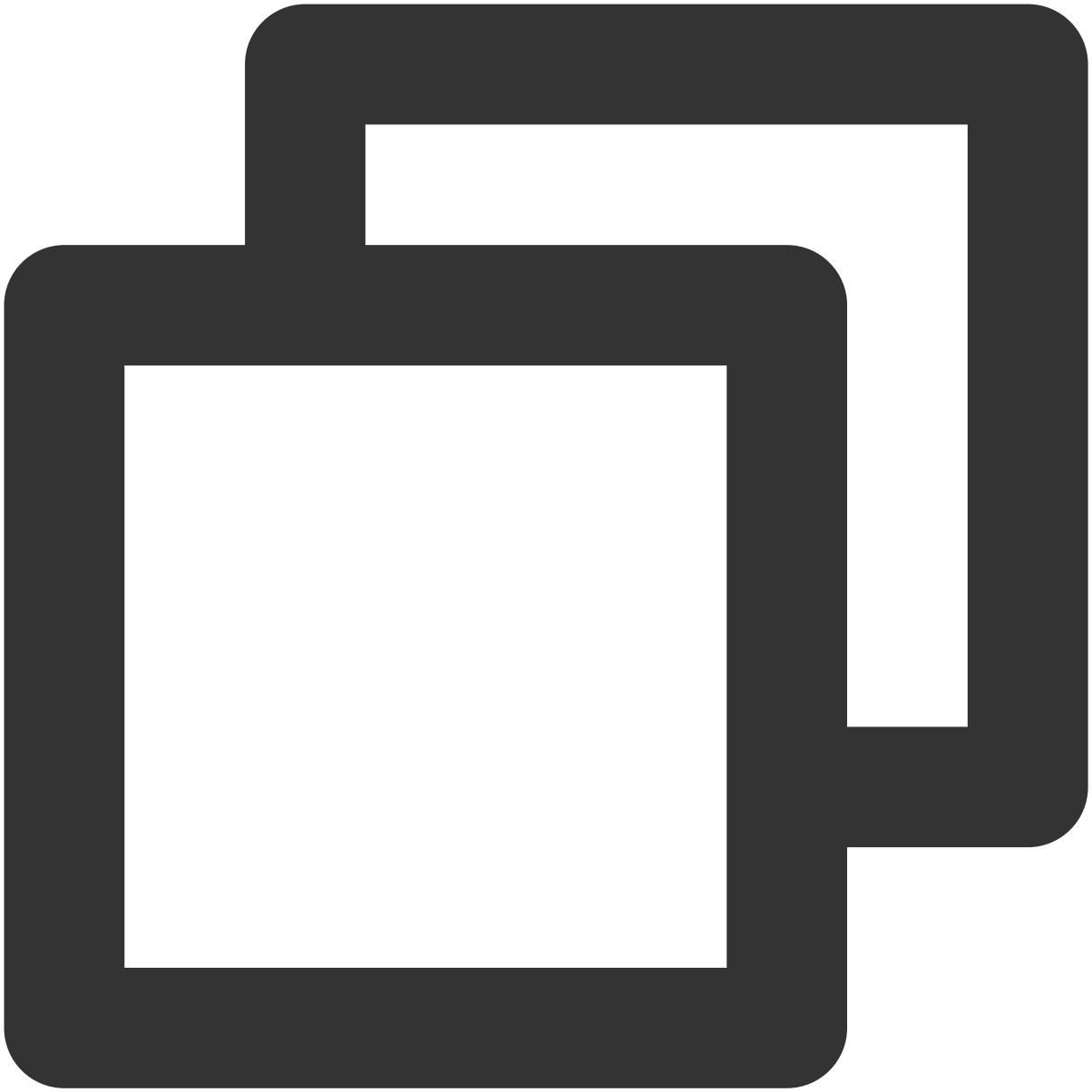
```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#    key：控制台配置的密钥key
//#    body：腾讯云回调返回的body体
//#    sign：腾讯云回调返回的签名值sign
//# 返回值：
```

```
//#   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\
        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'}");
        }
    }
}
```



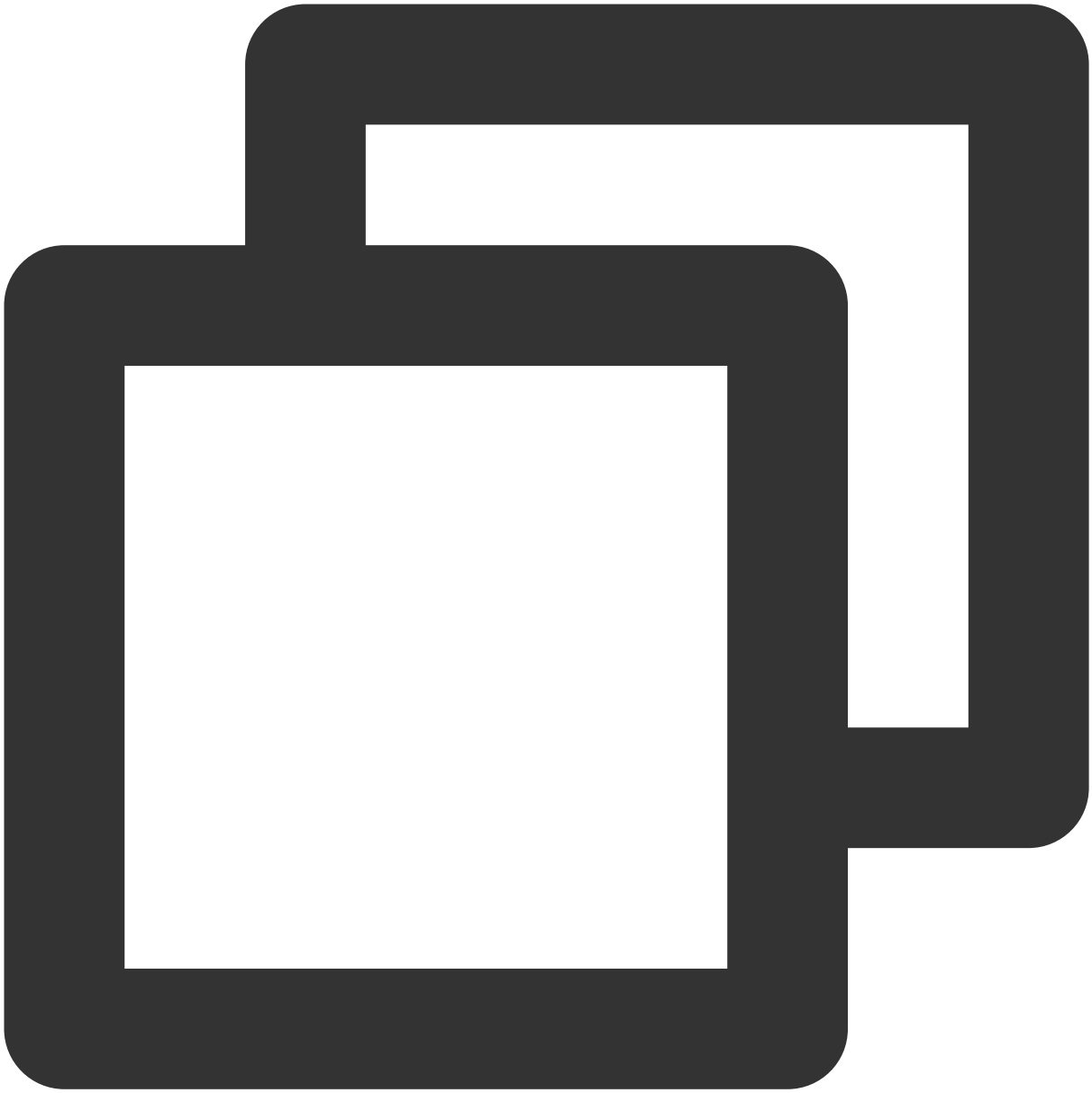
```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 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'
    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,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```



```
<?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\\"Callbac

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



```
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\
    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))
}
```

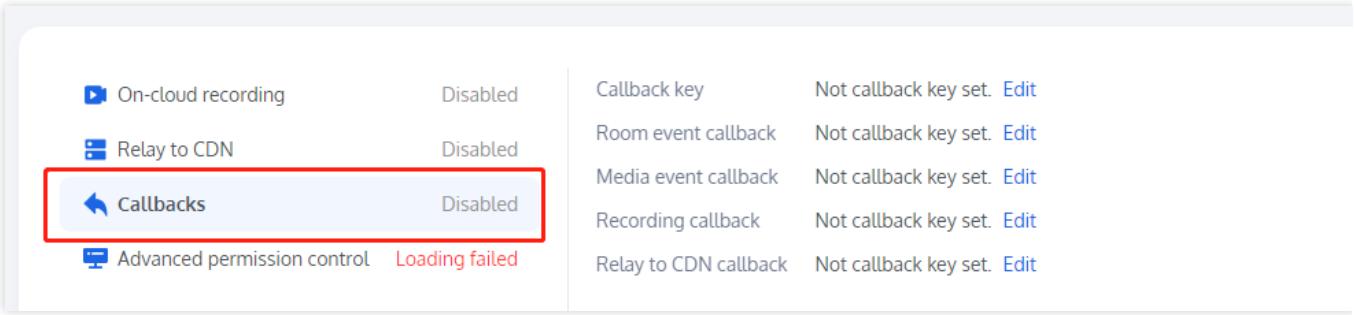
云端录制回调

最近更新时间：2024-08-07 10:53:53

本文针对 新版云端录制功能 的回调事件进行具体说明。

配置信息

Tencent RTC 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。



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}。

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
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	云端录制录制模块启动
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	云端录制录制模块退出
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：

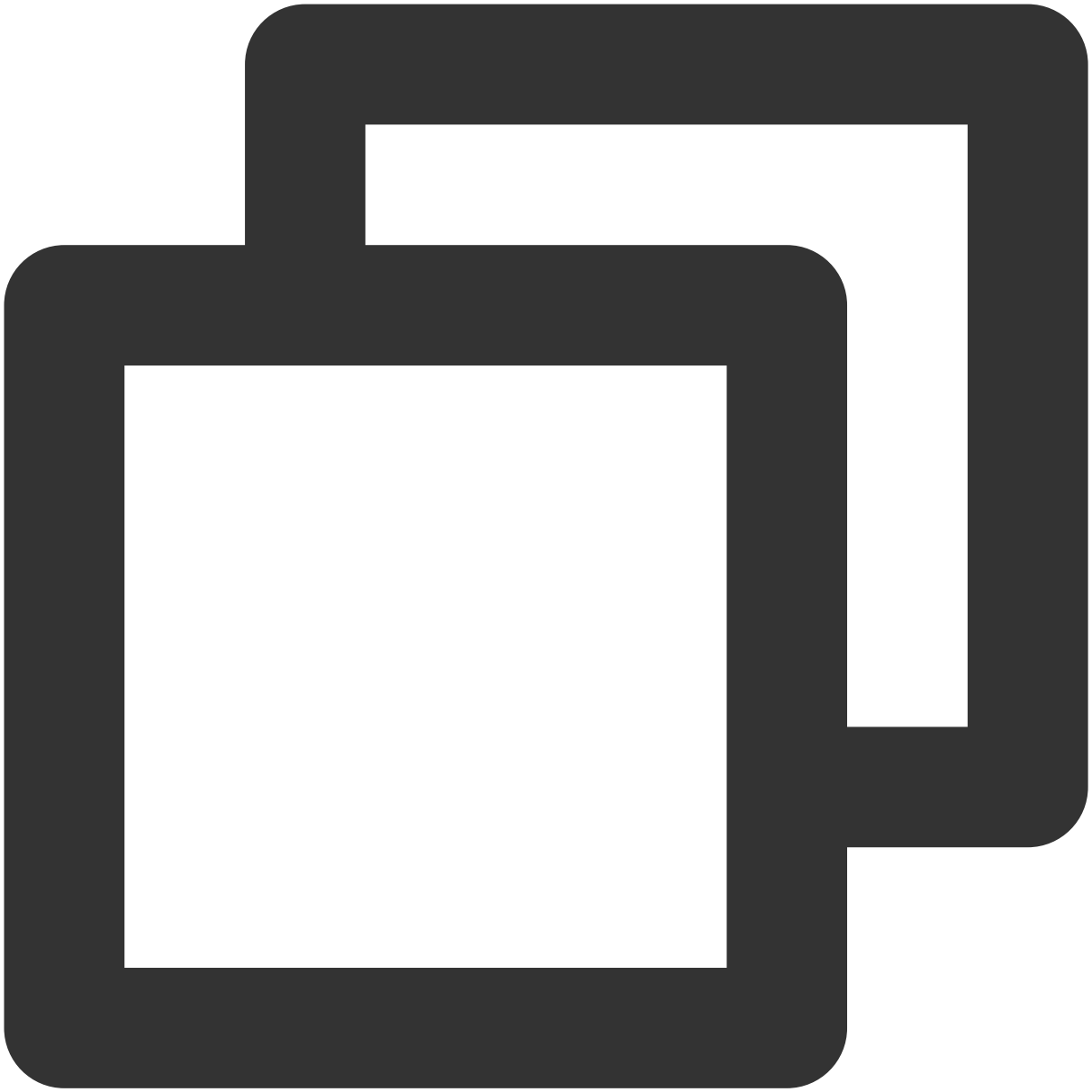
字段名	类型	含义
RoomId	String/Number	房间名（类型与客户端房间号类型一致）
EventTs	Number	时间发生的 Unix 时间戳，单位为秒 (不建议使用该字段，建议使用EventMsTs)

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



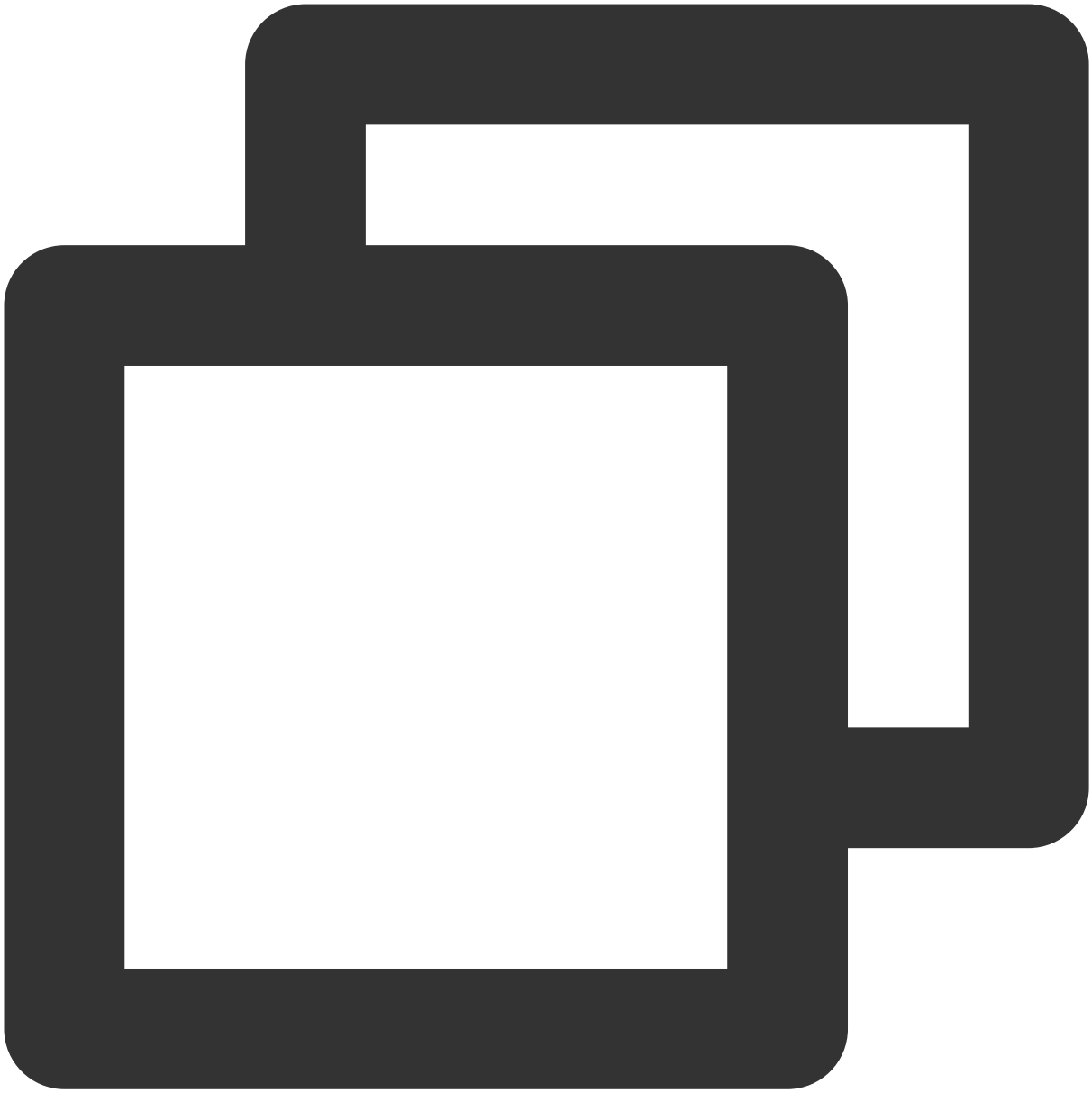
```
{
    "EventGroupId": 3,
    "EventType": 301,
    "CallbackTs": 1622186275913,
    "EventInfo": {
        "RoomId": "xx",
        "EventTs": "1622186275",
        "EventMsTs": 1622186275757,
        "UserId": "xx",
        "TaskId": "xx",
        "Payload": {
```

```
            "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

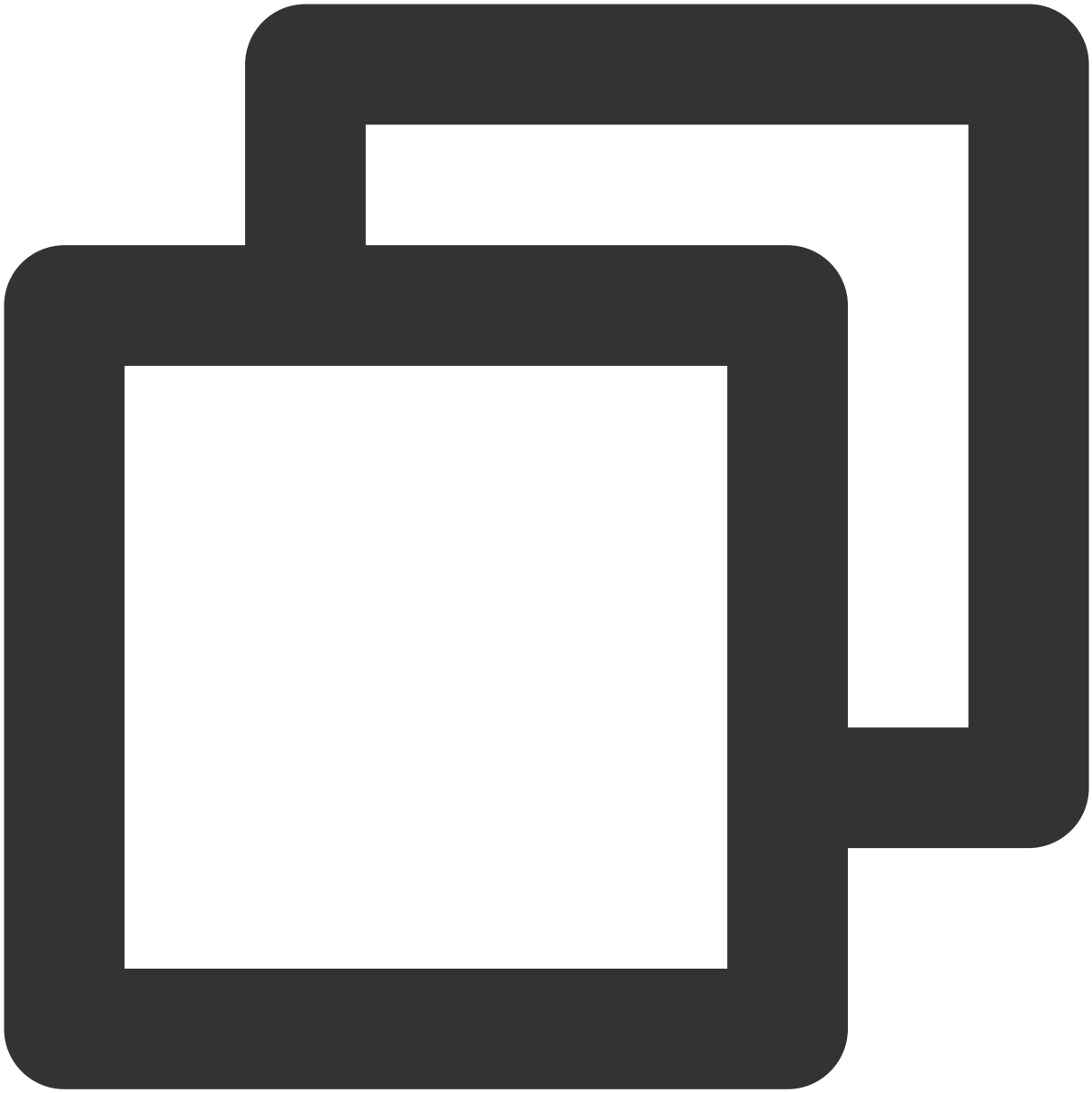
（EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP）时 Payload 的定义：

字段名	类型	含义
LeaveCode	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

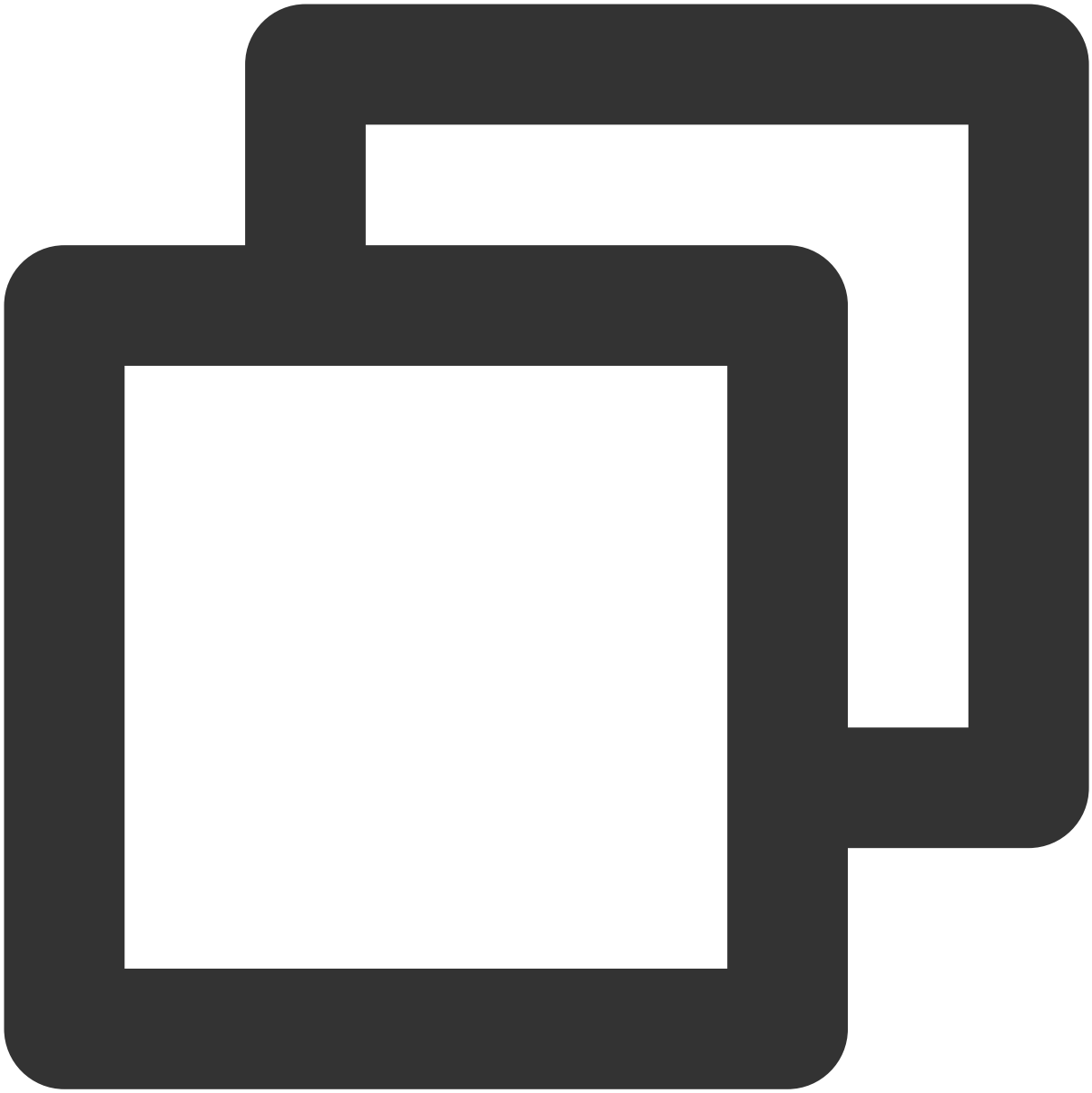
（EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE）时 Payload 的定义：

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"
      }
    }
}
```

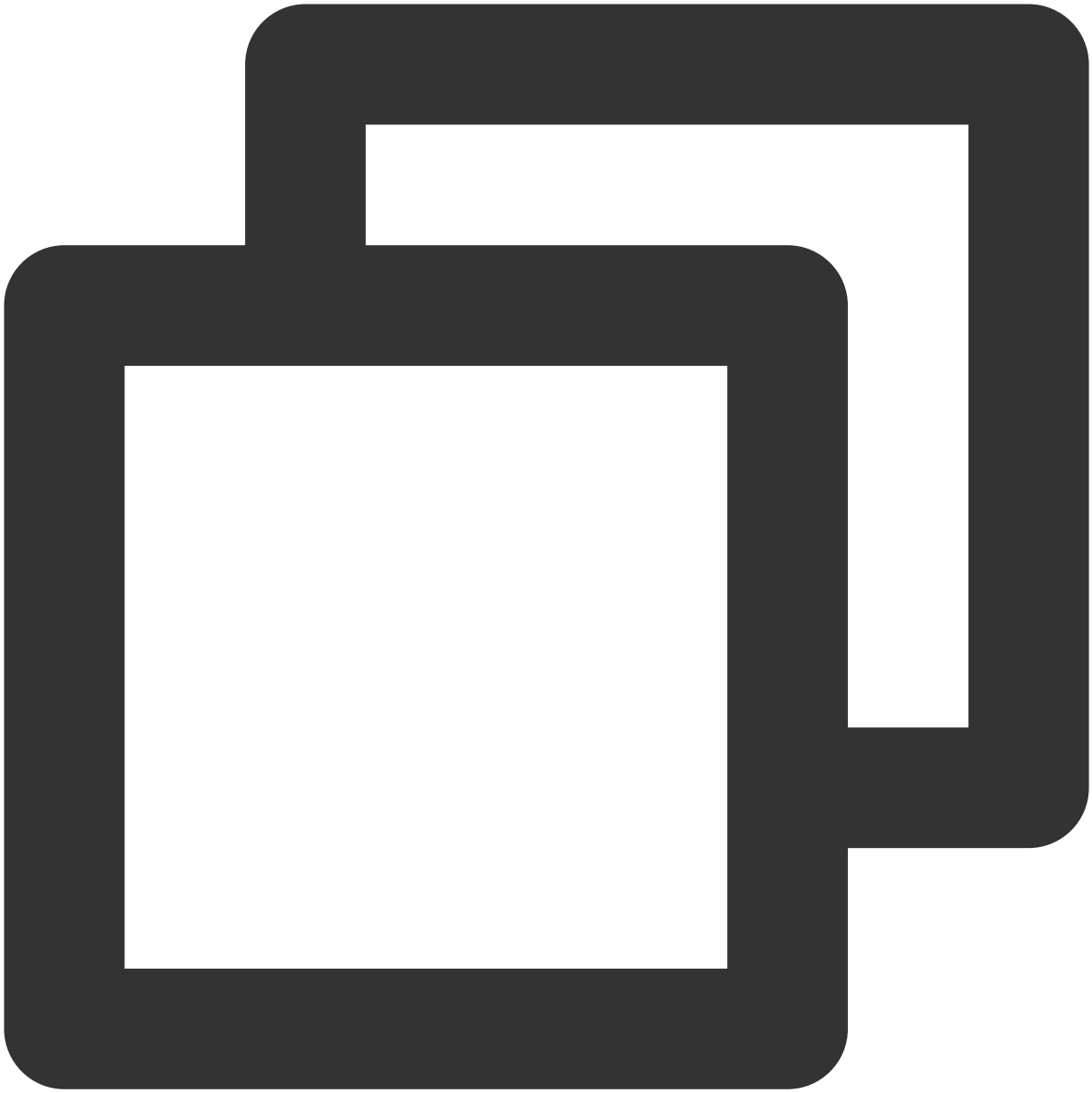
事件类型为310

（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，一个录制任务可能会回调多个310事件(每个事件对应一个录制文件信息)

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
          }
       ]

      }
    }
}
```

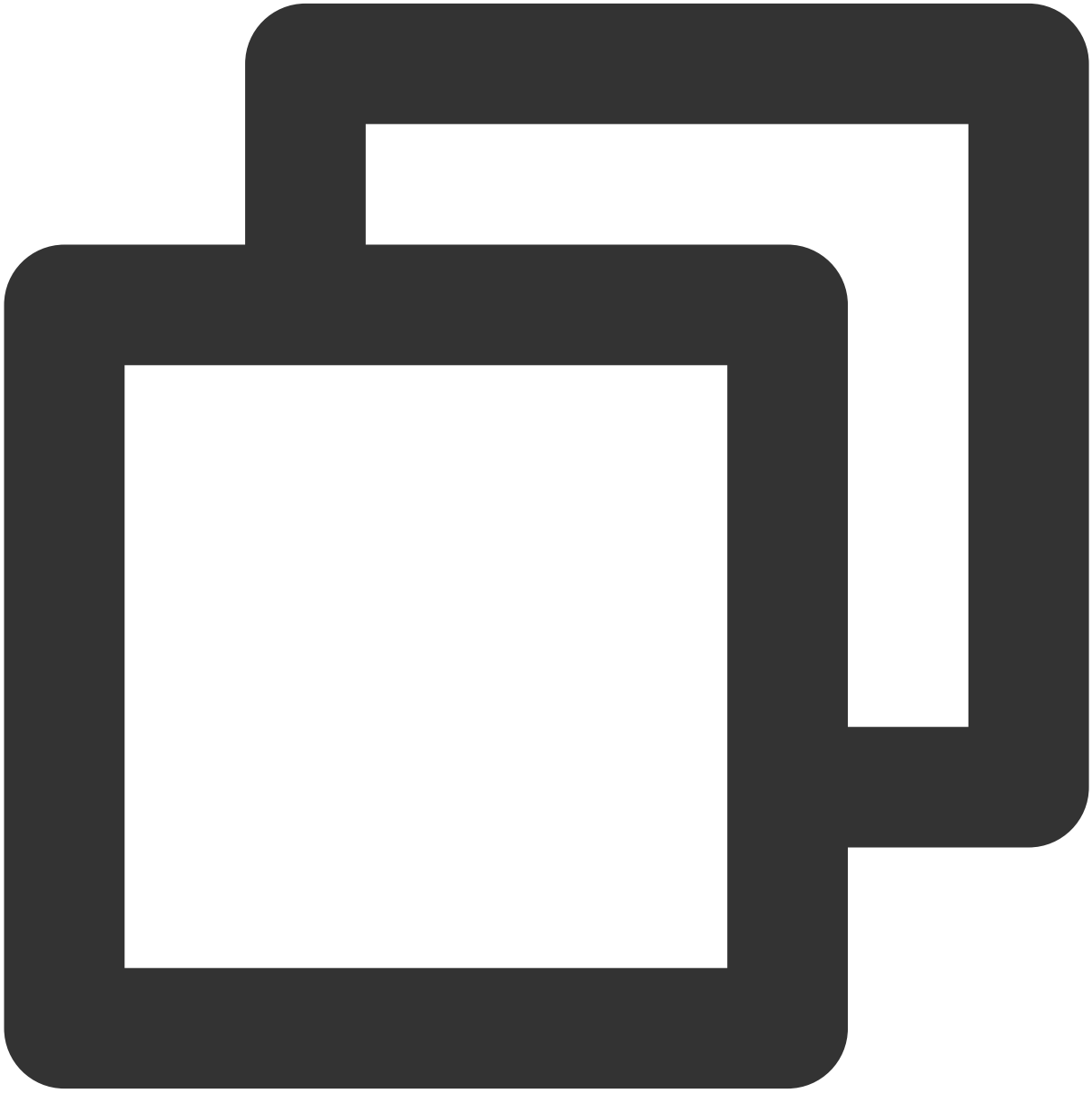
事件类型为311

（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	statue 不为0时，对应的错误信息

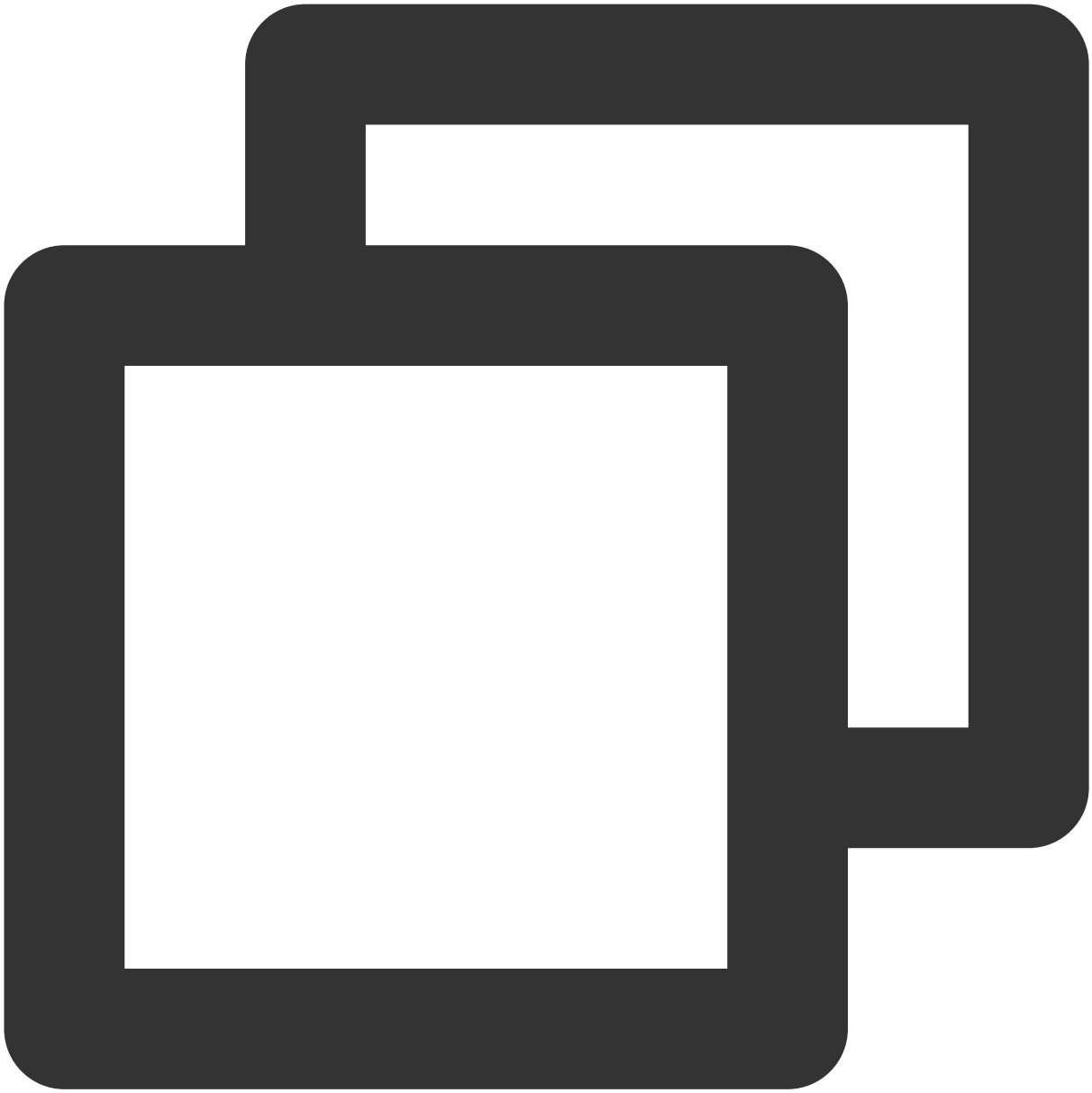
上传成功的回调：



```
{
  "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"
        }
      }
    }
  }
```

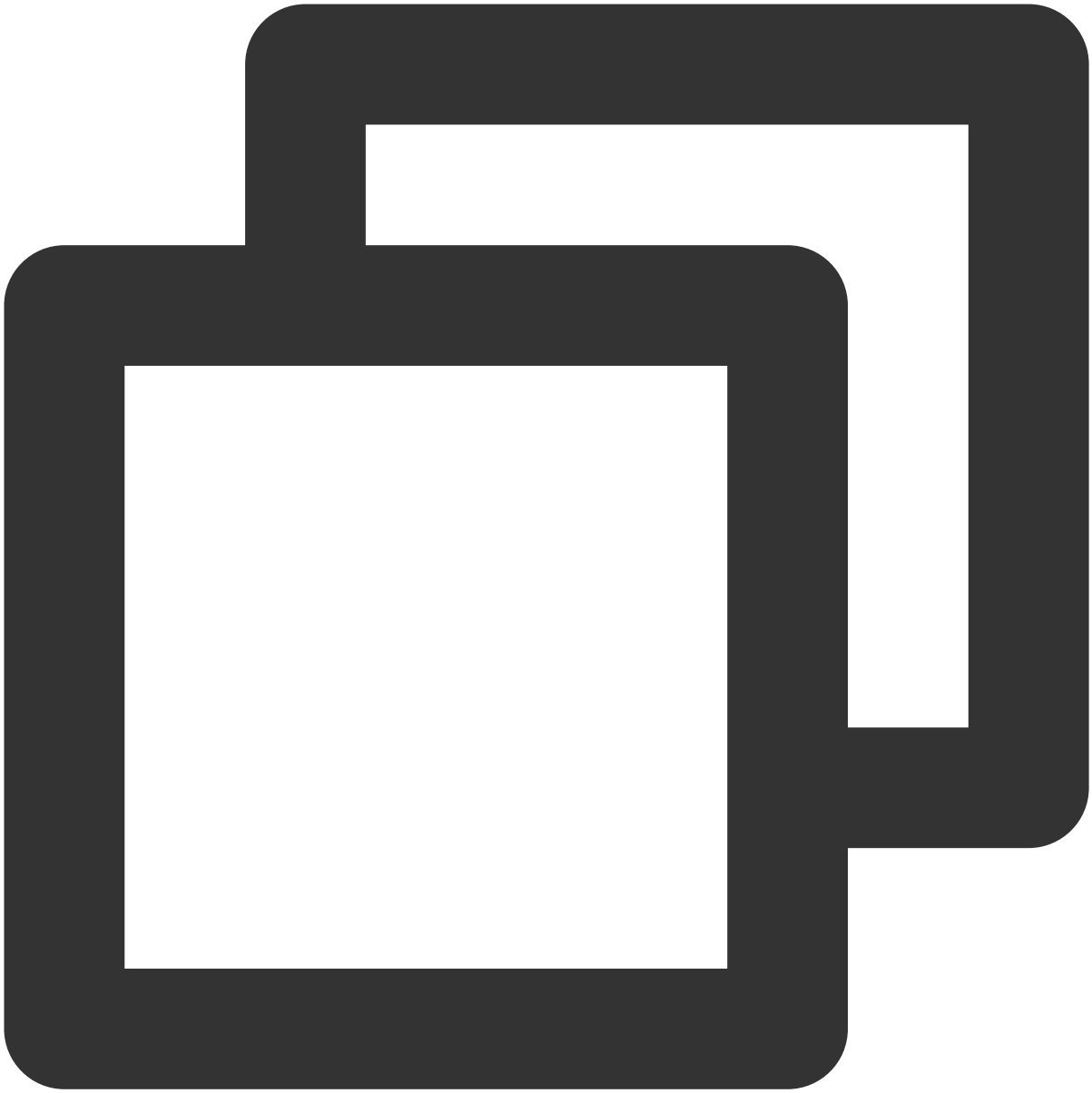
说明：

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

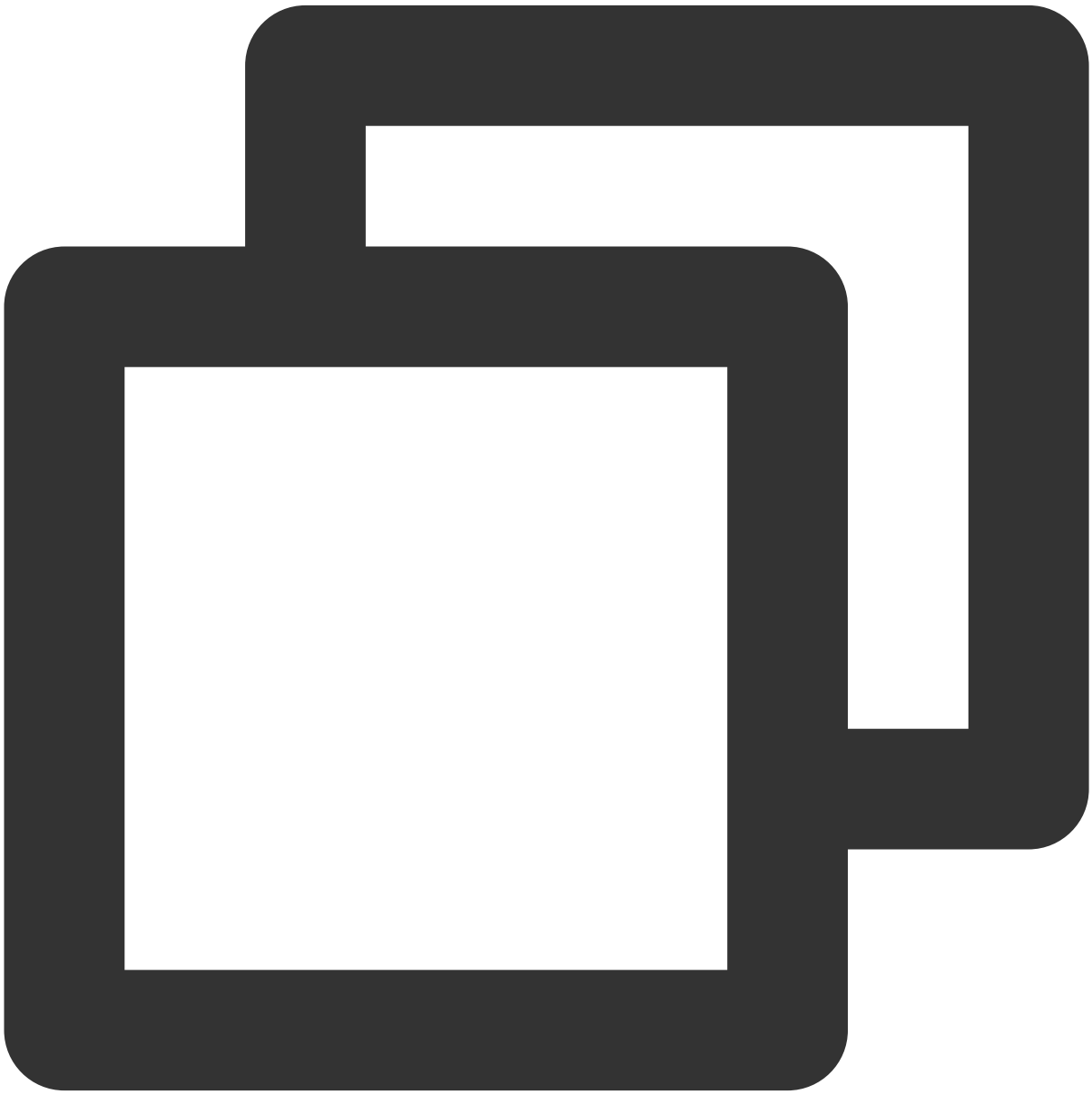


```
{
  "EventGroupId": 3,
  "EventType": 312,
  "CallbackTs": 1622191965320,
  "EventInfo": {
    "RoomId": "20015",
    "EventTs": 1622191965,
    "EventMsTs": 1622186275757,
    "UserId": "xx",
    "TaskId": "xx",
    "Payload": {
```

```
        "Status": 0
      }
    }
}
```

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：

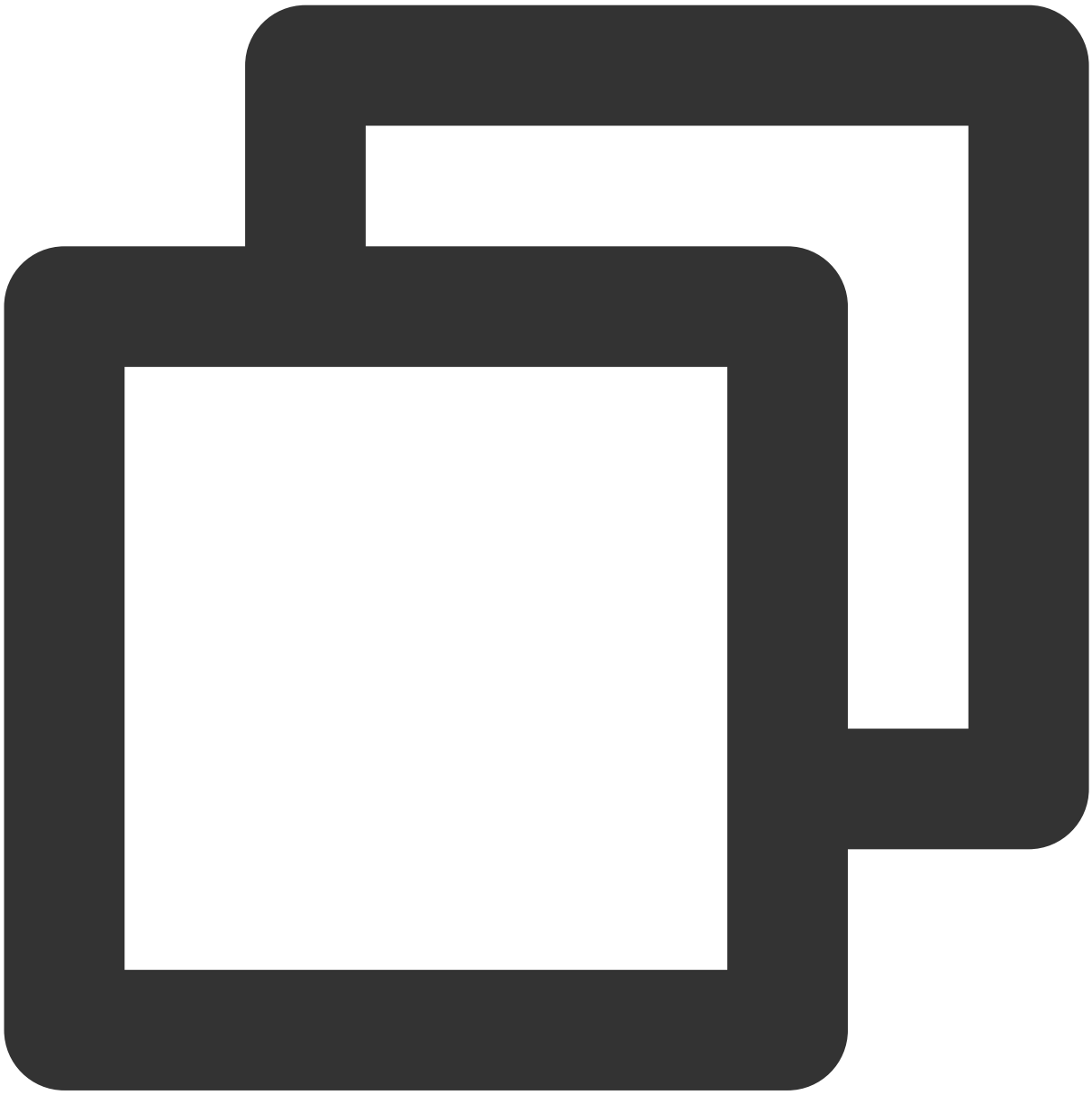


```
//签名 Sign 计算公式中 key 为计算签名 Sign 用的加密密钥。
```

```
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\\"Callback
```

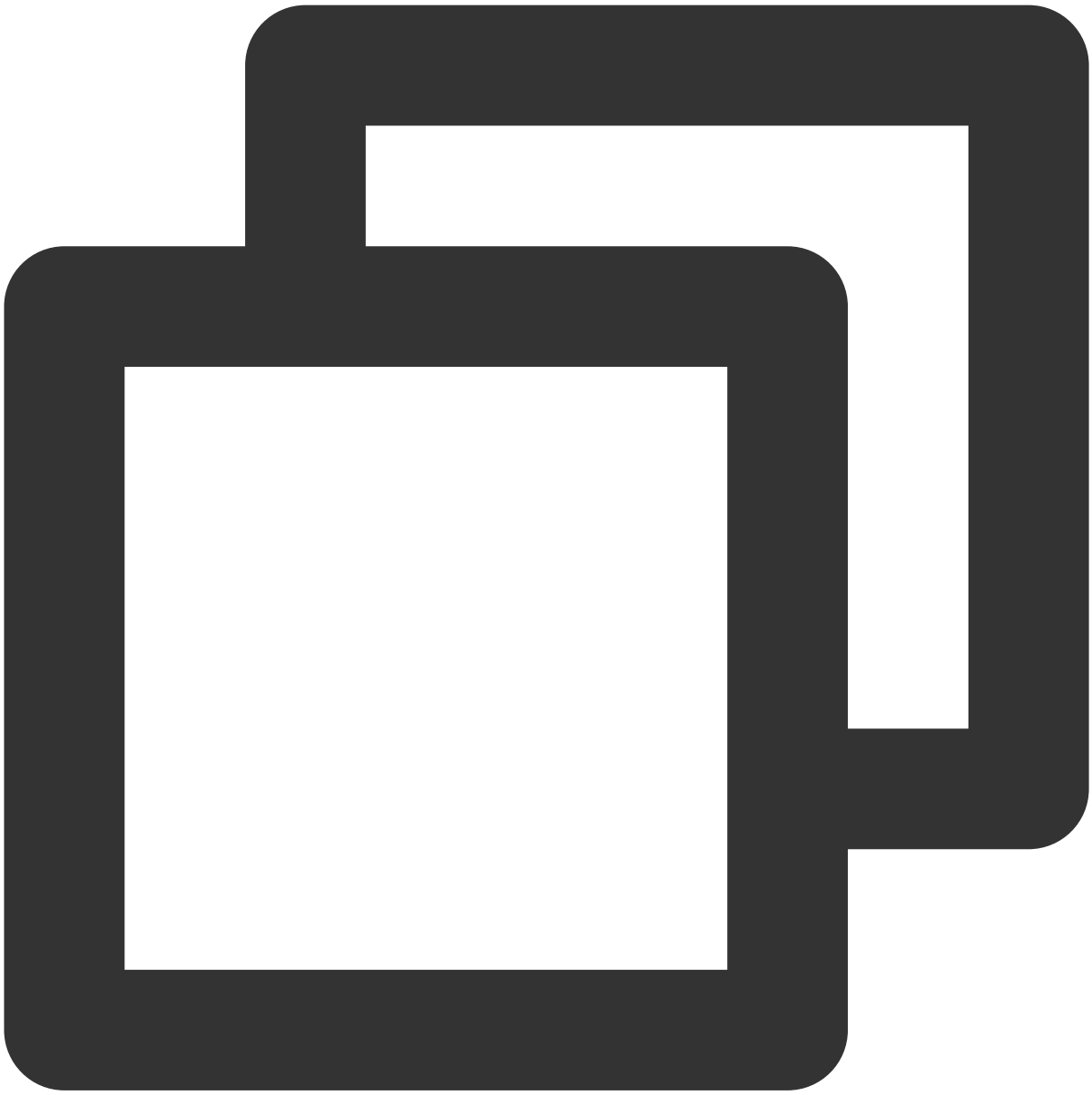
签名校验示例

Java

Python

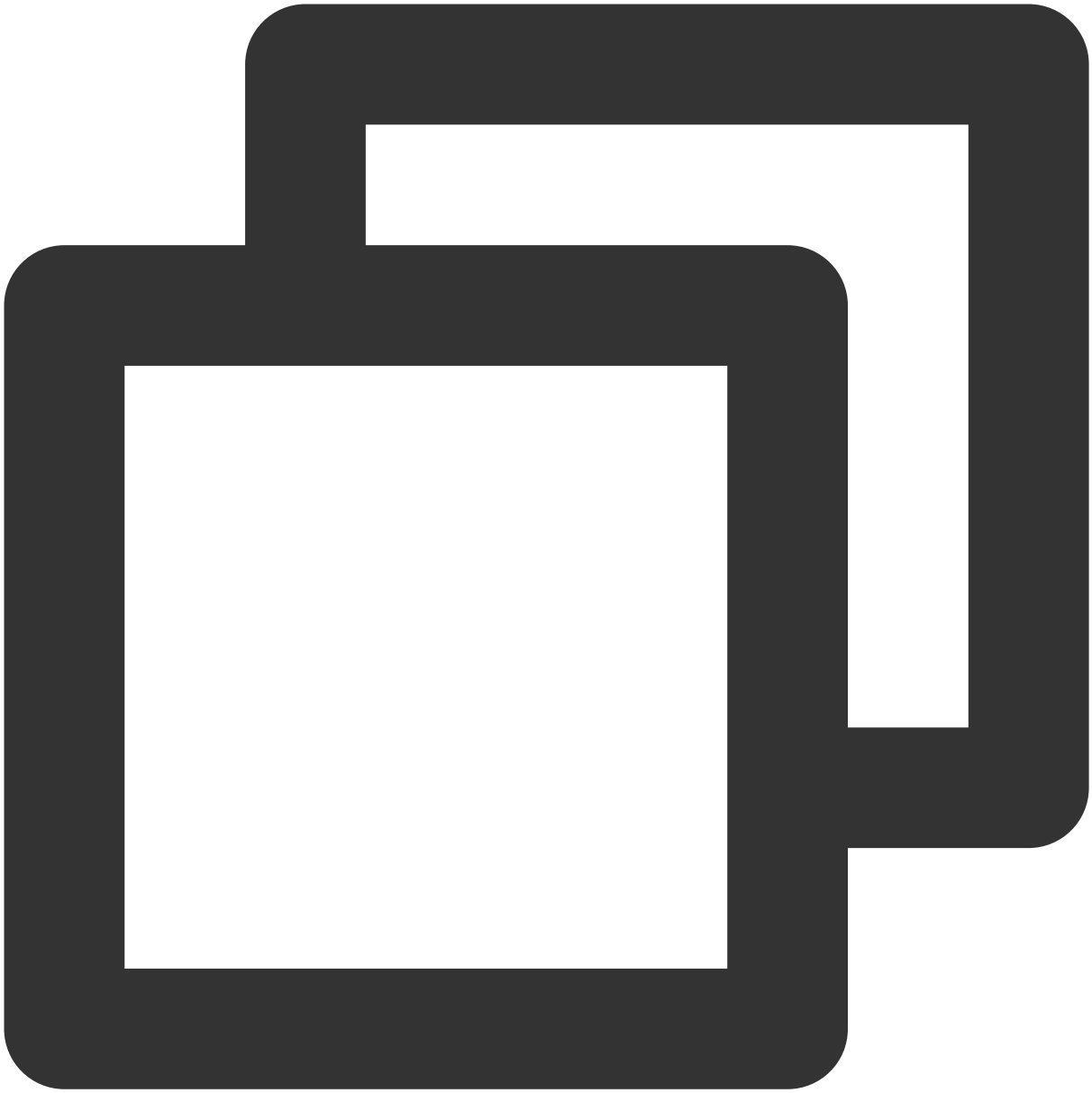
PHP

Golang



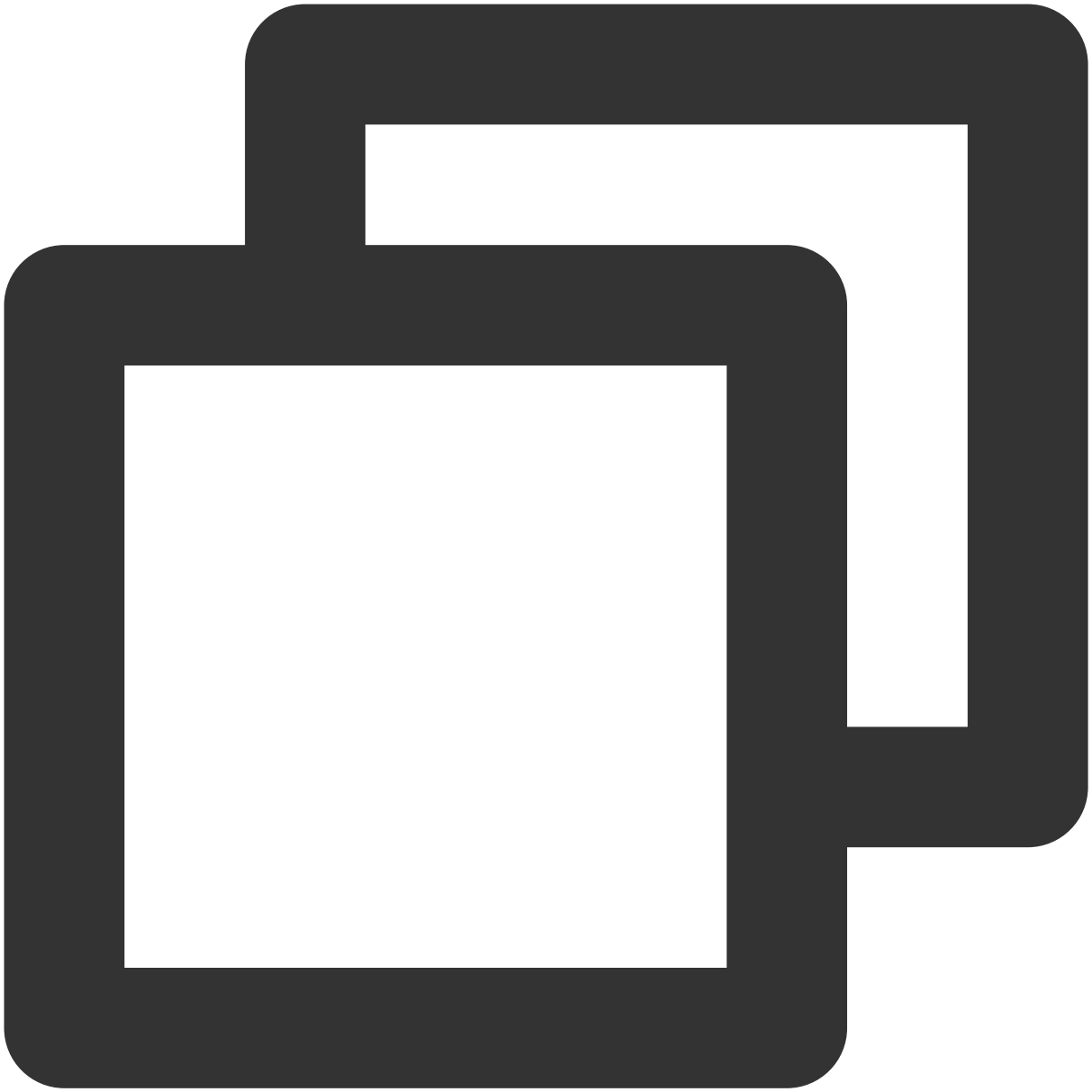
```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#    key：控制台配置的密钥key
//#    body：腾讯云回调返回的body体
//#    sign：腾讯云回调返回的签名值sign
```

```
//# 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\
        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'}");
        }
    }
}
```



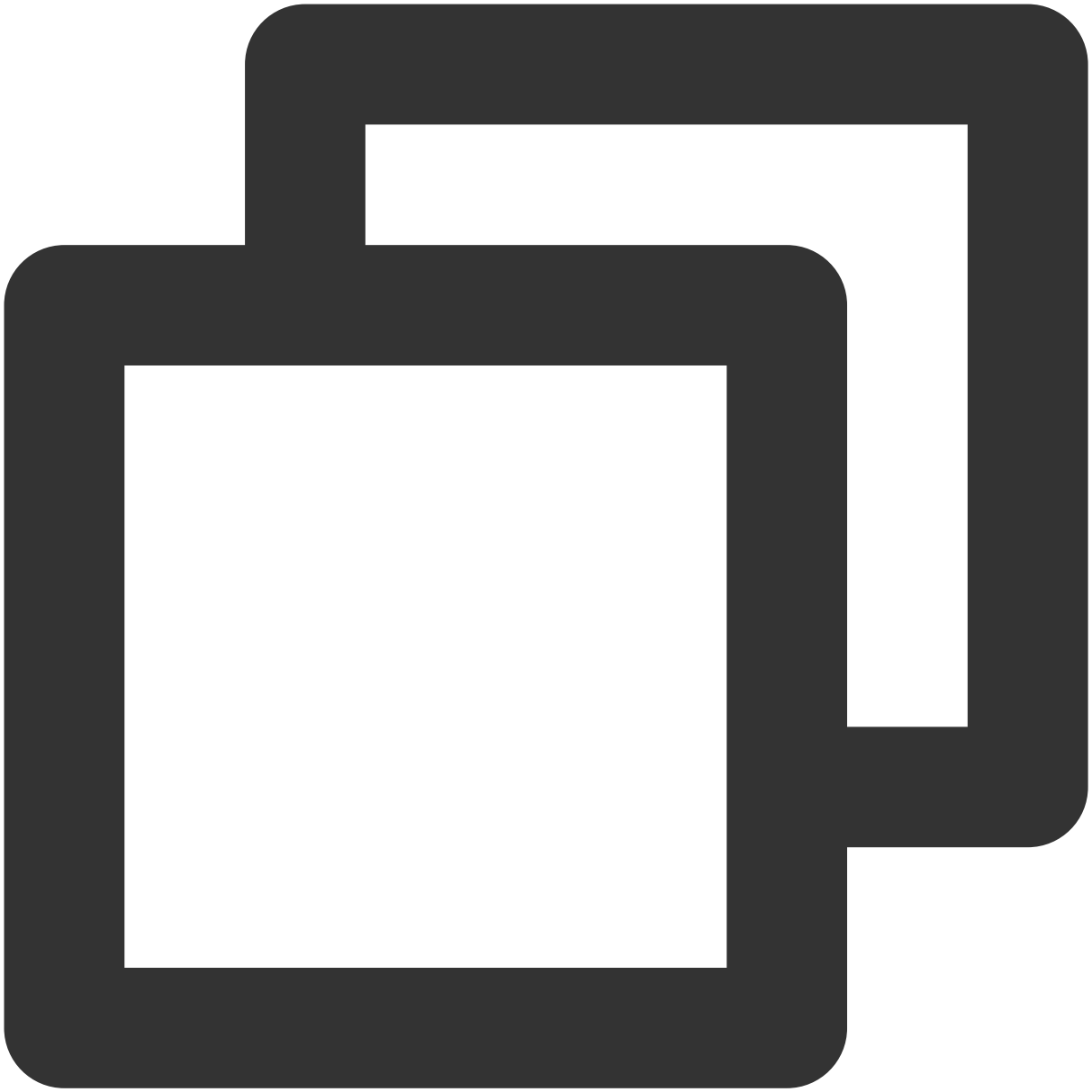
```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 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'
    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,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```



```
<?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\\"Callbac

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



```
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\
    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：2024-08-01 16:20:07

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

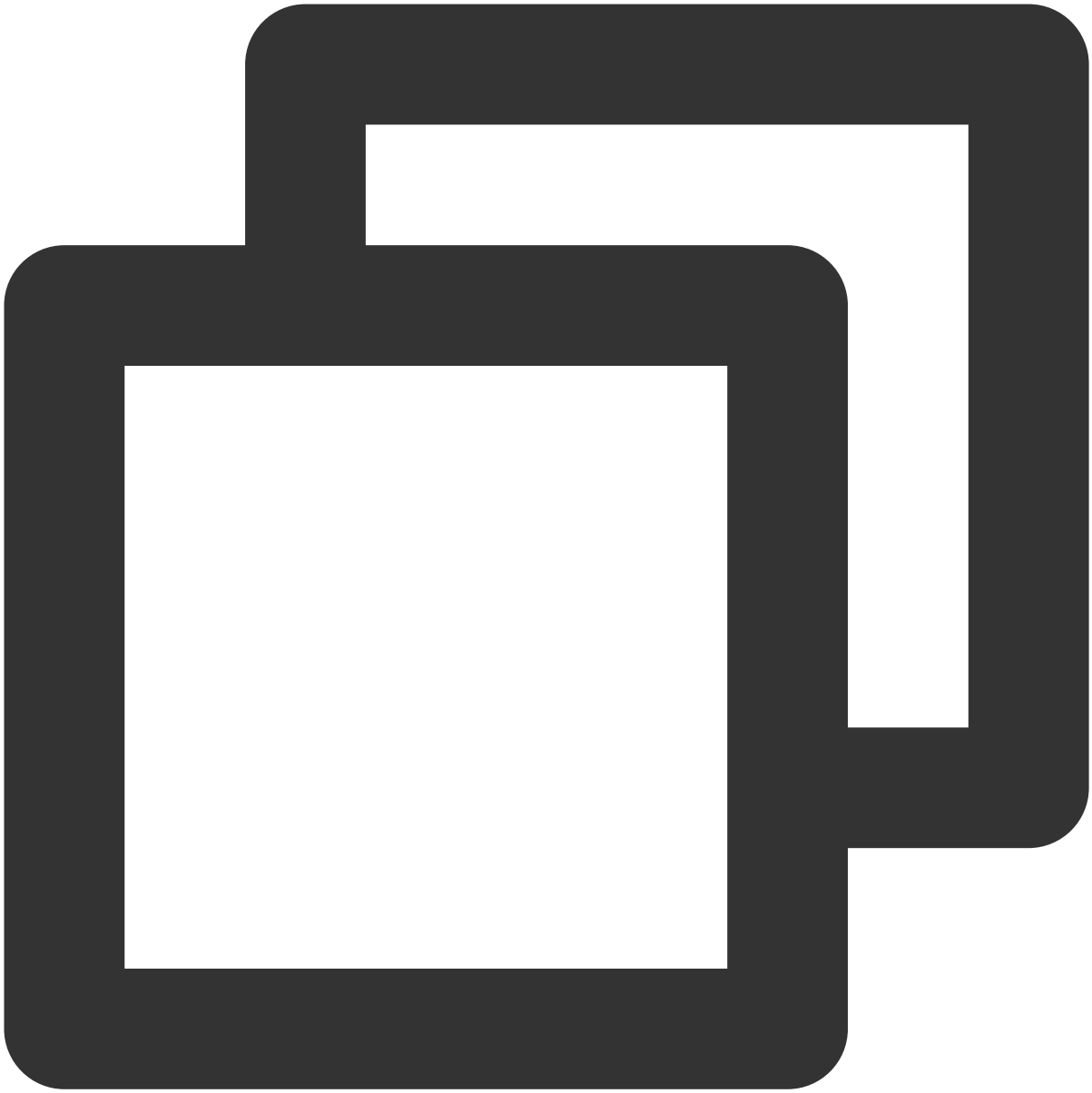
事件回调消息以 HTTP/HTTPS POST 请求发送给您的服务器，其中：

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

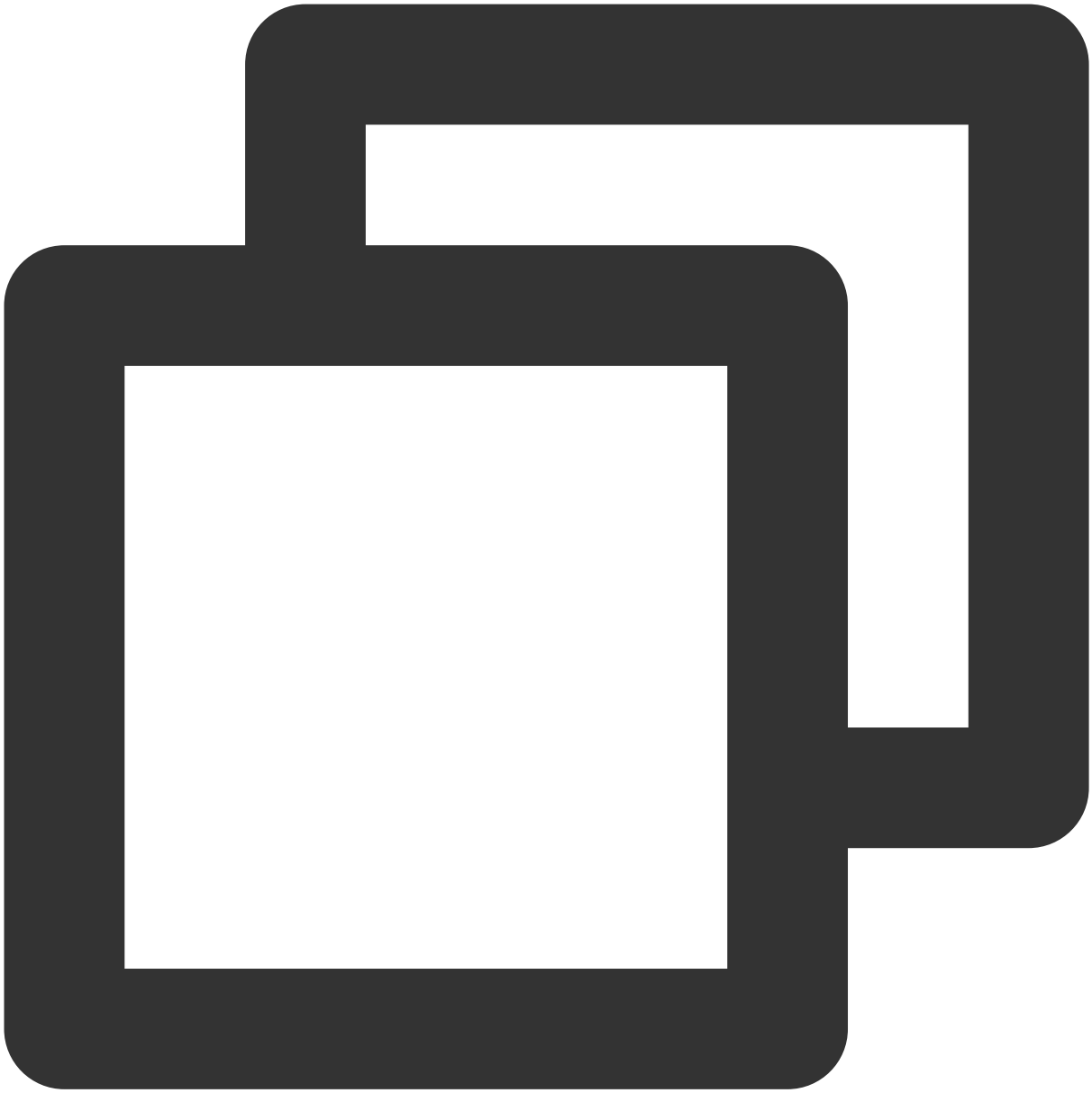
TaskId	String	输入在线媒体流任务 ID
Status	Number	输入在线媒体流停止状态

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

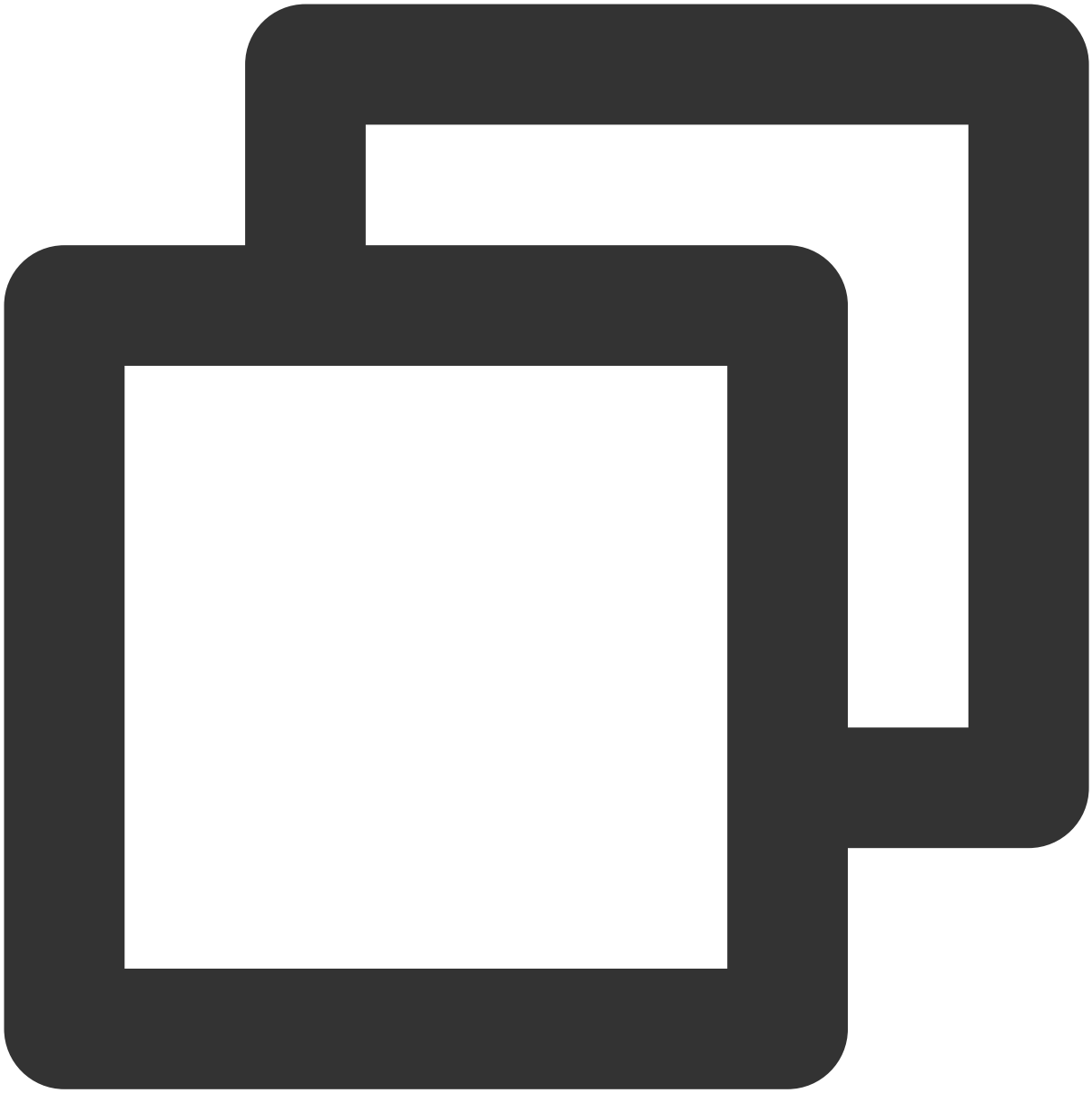
签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：



```
//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\\"
```

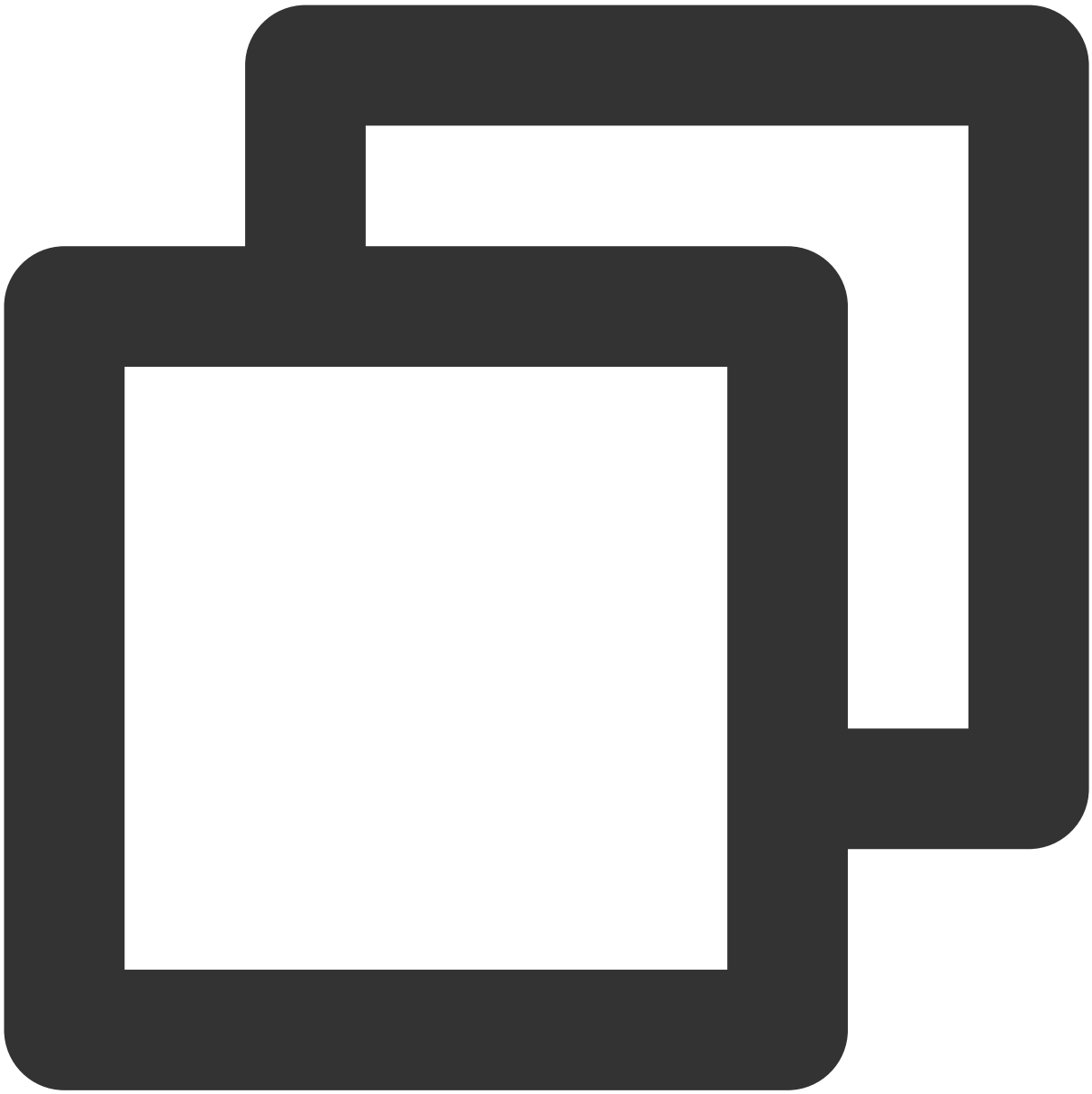
签名校验示例

Java

Python

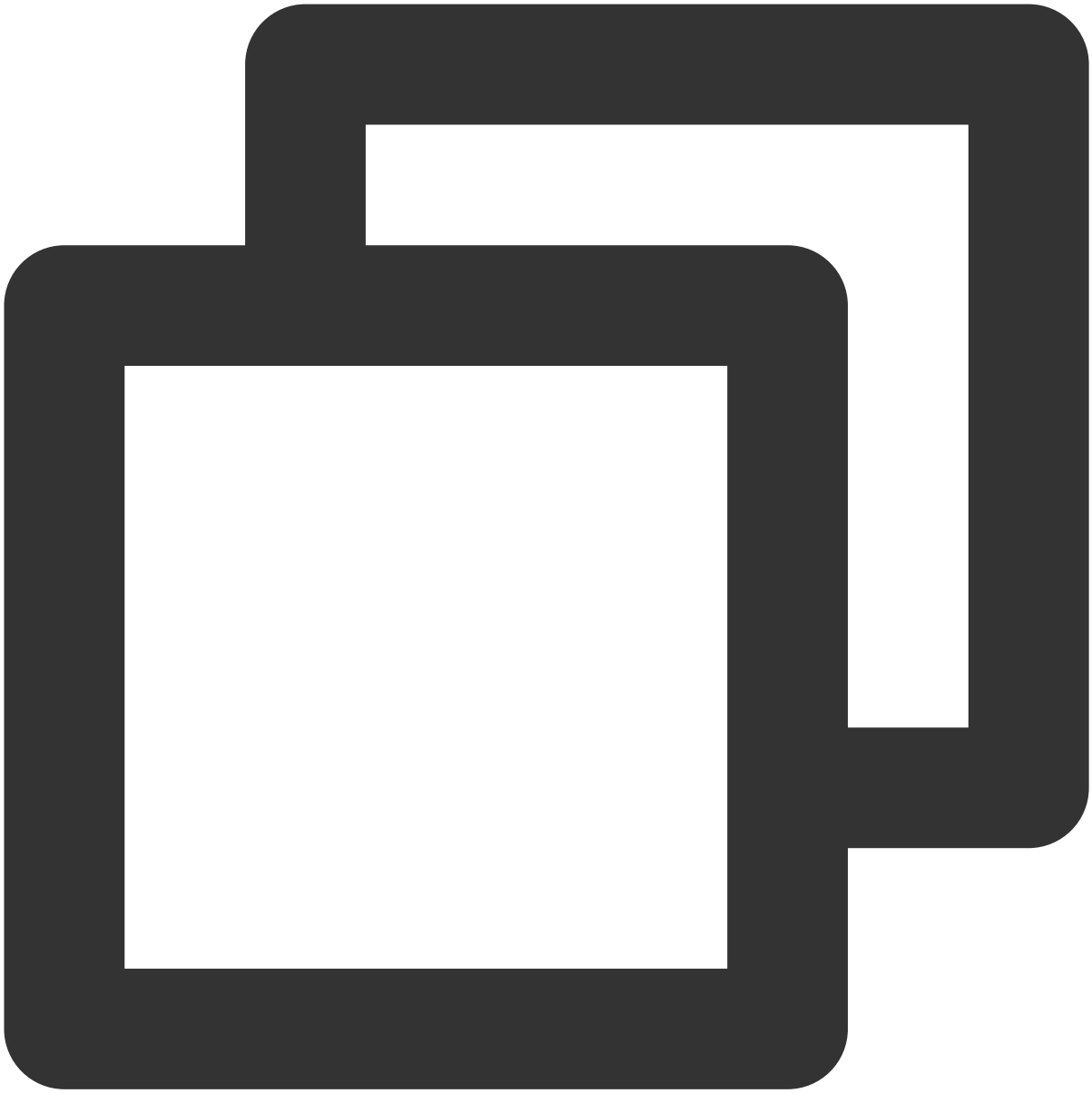
PHP

Golang



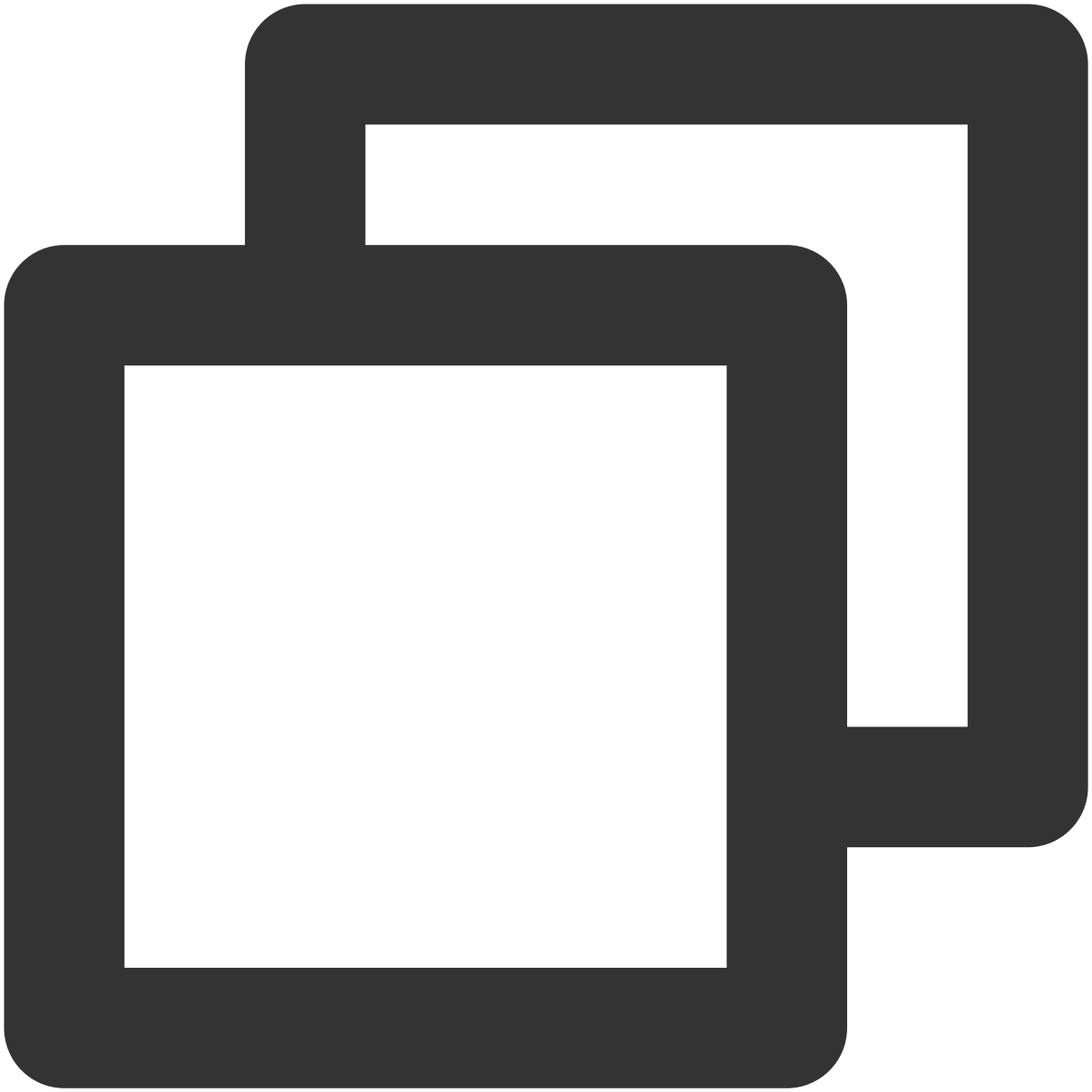
```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#    key：控制台配置的密钥key
//#    body：腾讯云回调返回的body体
//#    sign：腾讯云回调返回的签名值sign
//# 返回值：
```

```
//#   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\
        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'}");
        }
    }
}
```



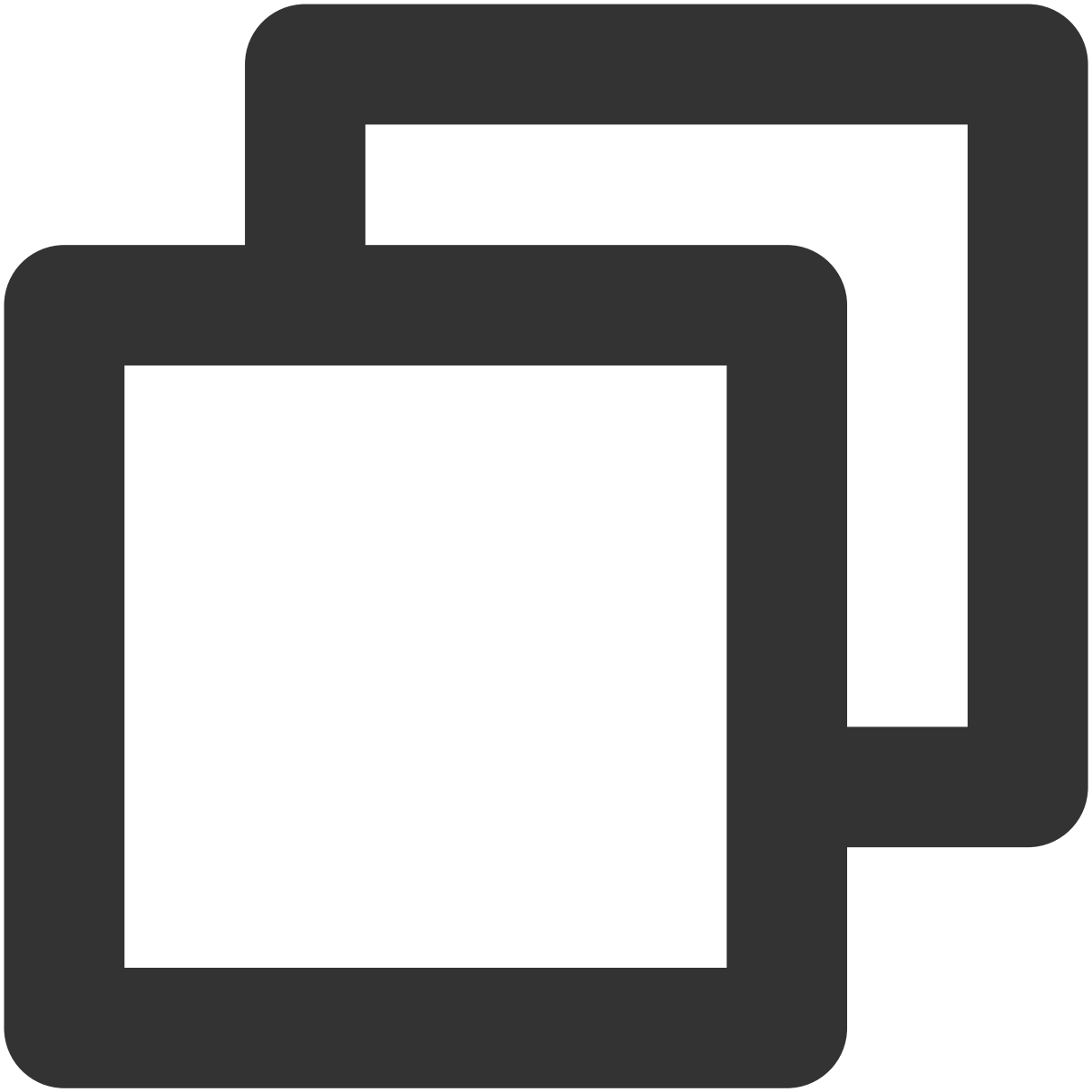
```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 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'
    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,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```



```
<?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\\"Callbac

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



```
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\
    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))
}
```

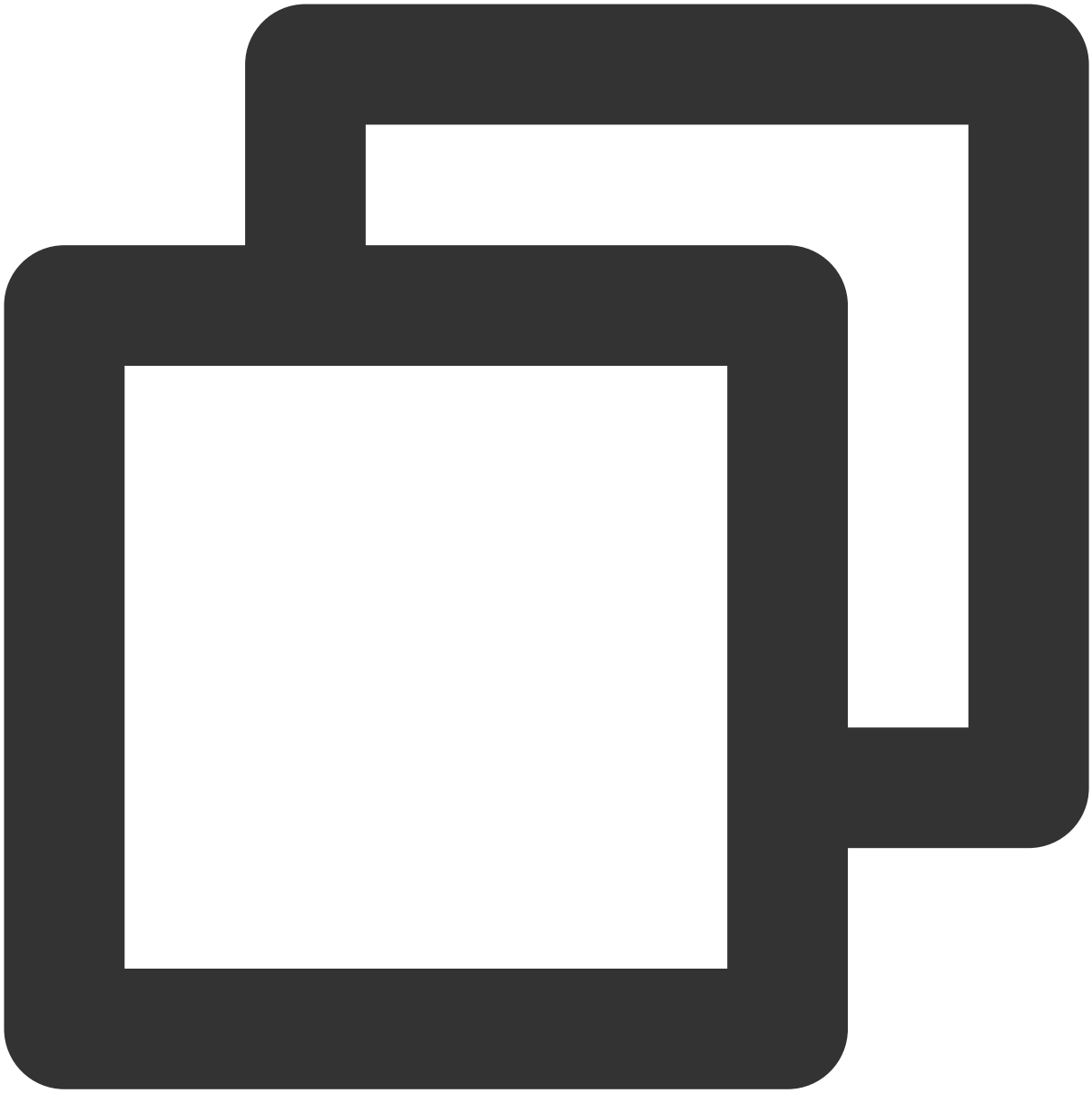
签名校验示例

最近更新时间：2023-08-09 14:42:49

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。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

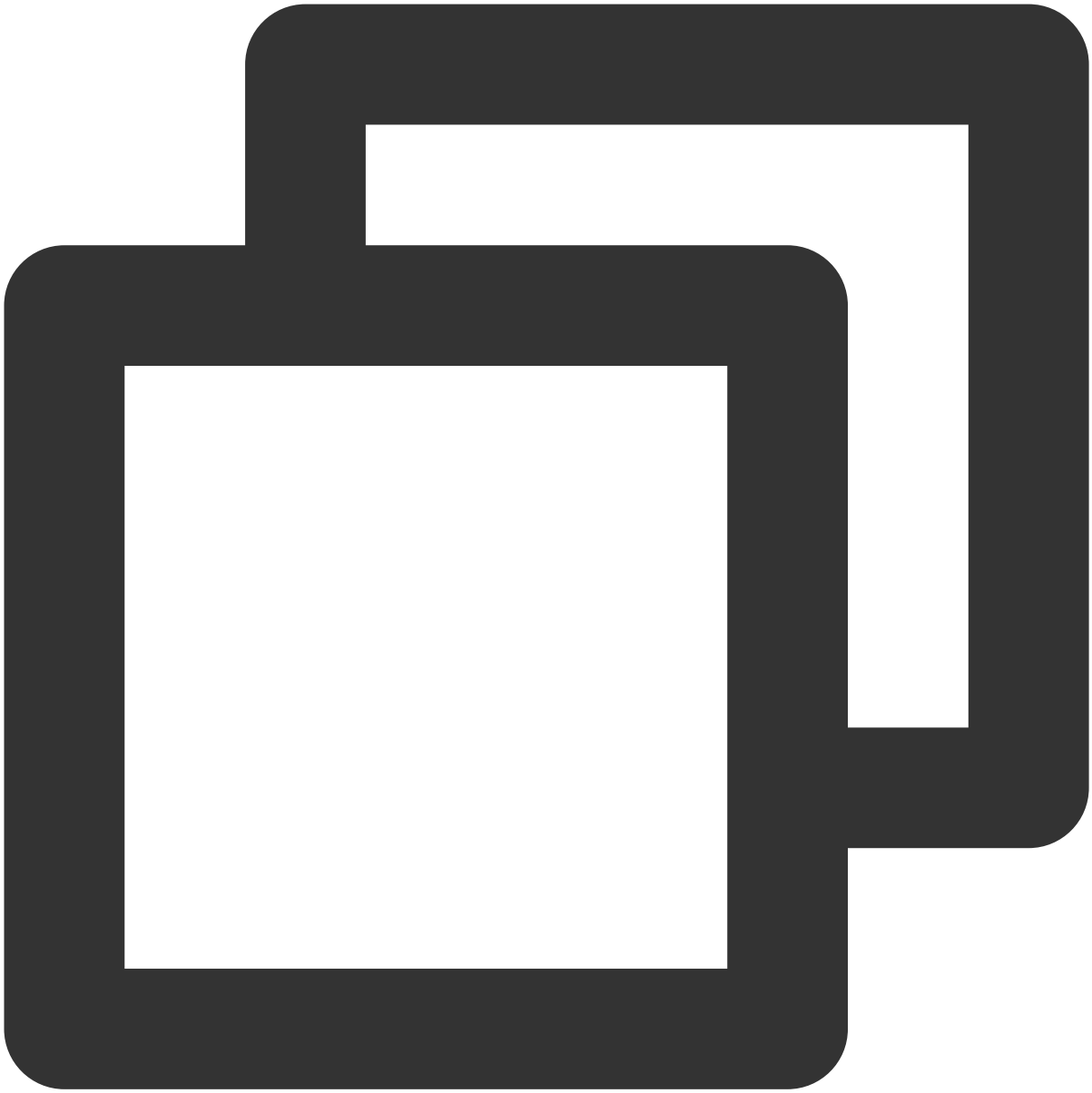
签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：



```
//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，示例如下：



```
body="{\\n\\t\\"EventGroupId\\":\\t1,\\n\\t\\"EventType\\":\\t103,\\n\\t\\"Callback
```

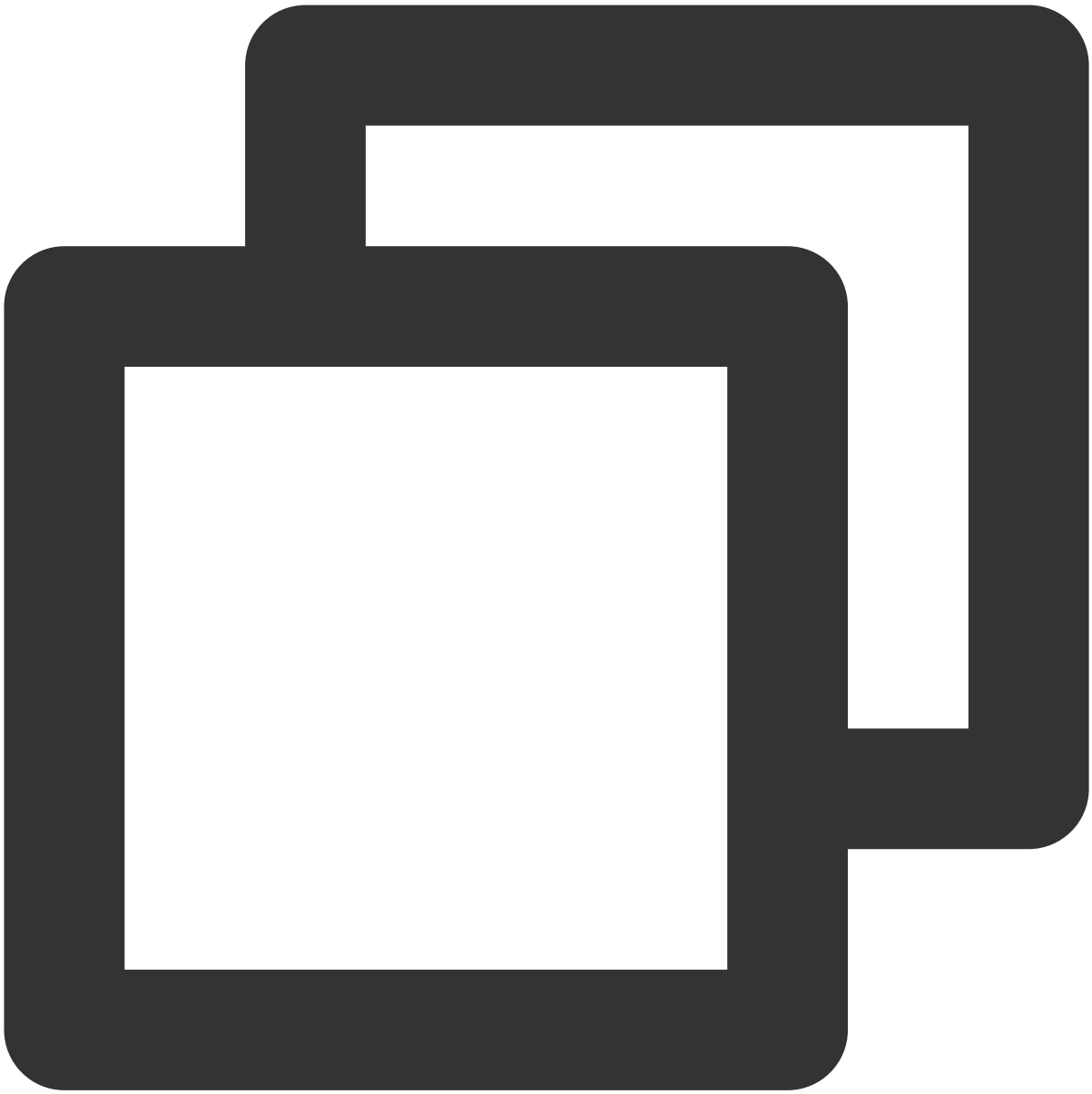
签名校验示例

Java

Python

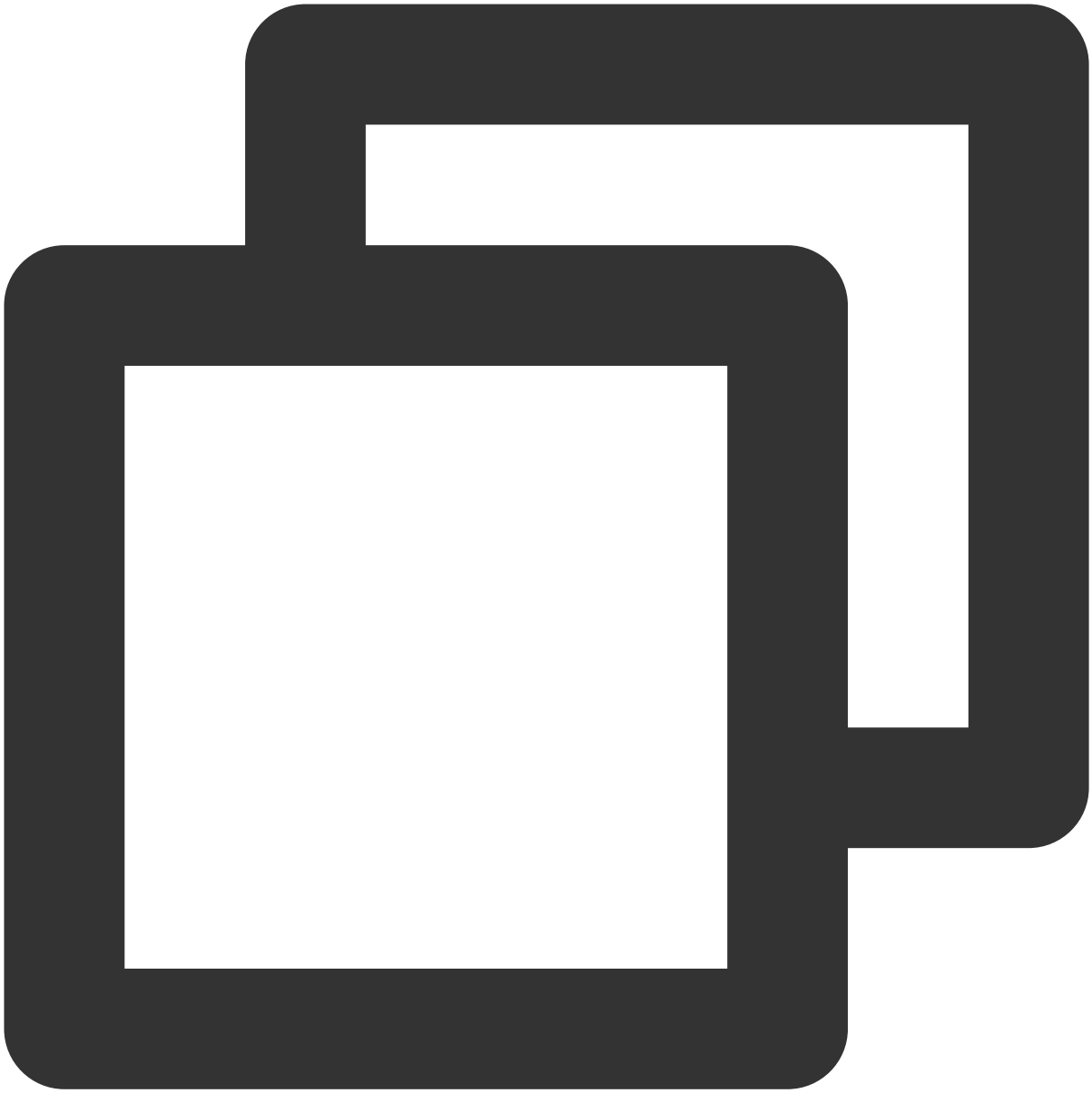
PHP

Golang



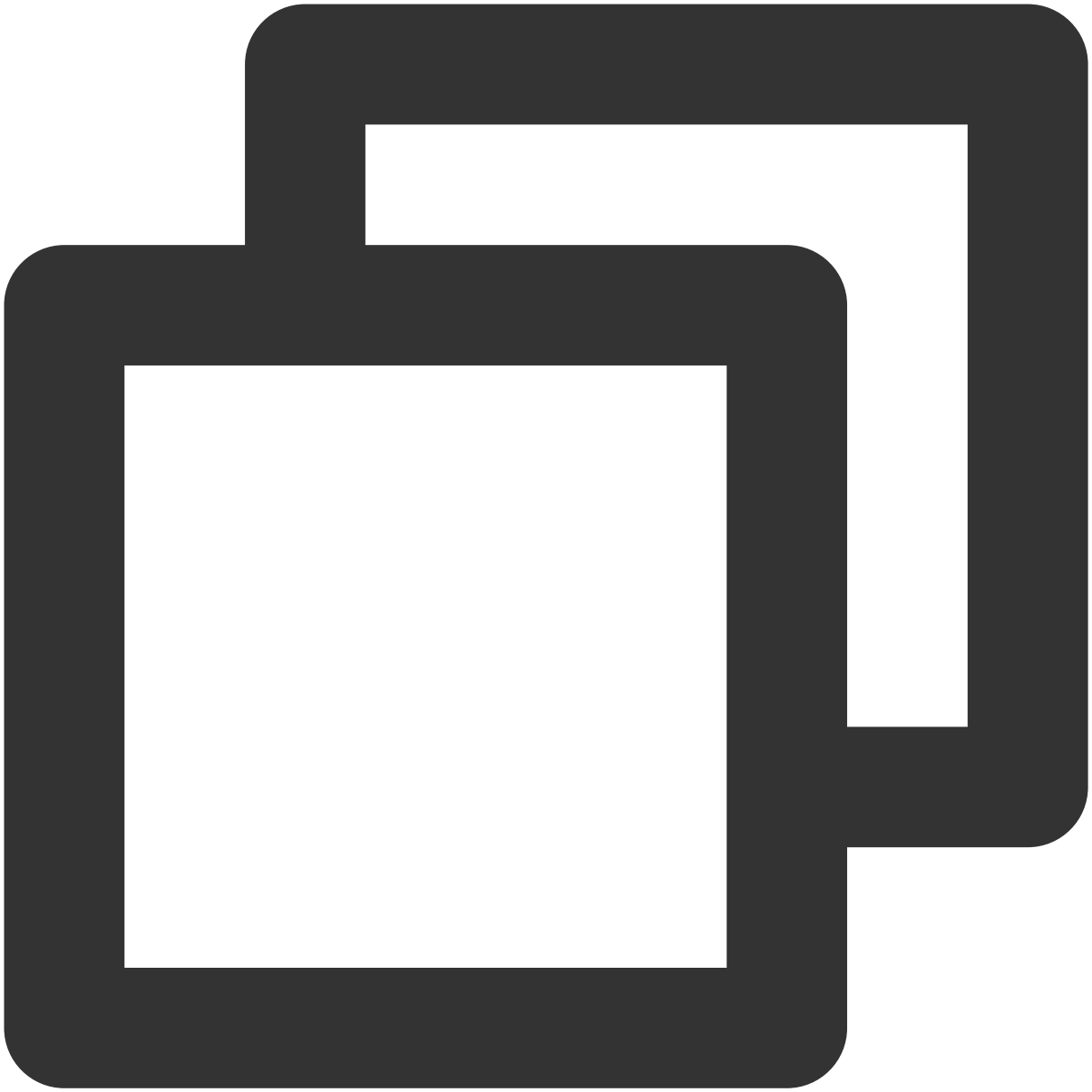
```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#    key：控制台配置的密钥key
//#    body：腾讯云回调返回的body体
//#    sign：腾讯云回调返回的签名值sign
//# 返回值：
```

```
//#   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\
        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'}");
        }
    }
}
```



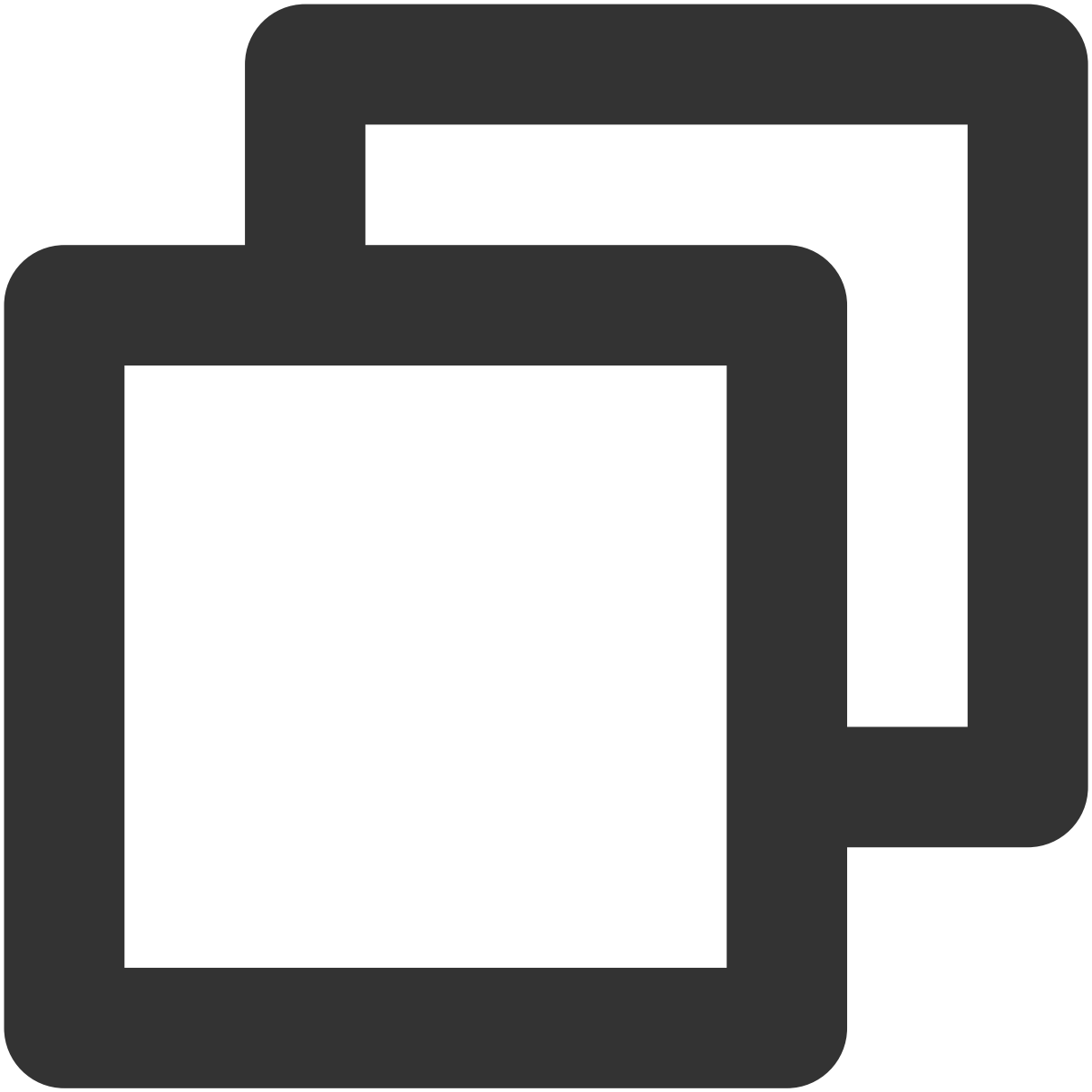
```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 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'
    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,\
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```



```
<?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\\"Callbac

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



```
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\
    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))
}
```

资源访问管理
访问管理综述

最近更新时间：2022-03-22 18:19:12

> 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：2022-06-10 14:44:08

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

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

接口名称	接口分类	功能描述
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 • 监控仪表盘 • 开发辅助 > UserSig 生成&校验 • 应用管理	获取应用列表
DescribeSdkAppInfo	TRTC 控制台 应用管理 > 应用信息	获取应用信息
ModifyAppInfo	TRTC 控制台 应用管理 > 应用信息	编辑应用信息
ChangeSecretKeyFlag	TRTC 控制台 应用管理 > 应用信息	修改权限密钥状态
CreateWatermark	TRTC 控制台 应用管理 > 素材管理	上传图片
DeleteWatermark	TRTC 控制台 应用管理 > 素材管理	删除图片

接口名称	使用模块	功能描述
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

接口名称	使用模块	功能描述
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

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

接口名称	使用模块	功能描述	特殊限制说明
ModifyMixConf	TRTC 控制台 应用管理 > 功能配置	开启/关闭自动旁路推流	输入参数无 SDKAppID，无法进行资源级授权。如有需要，可通过 DescribeAppStatList 接口来限制可查询的应用列表
RemindBalance	TRTC 控制台 套餐包管理	获取预付费套餐包余额告警信息	预付费套餐包为单个腾讯云账号下的所有 TRTC 应用共享，套餐包信息中无 SDKAppID 参数，无法进行资源级授权

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

预设策略

最近更新时间：2021-02-25 16:04:41

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

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

TRTC 目前提供了以下预设策略：

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

预设策略使用示例

新建拥有 TRTC 全读写访问权限的子帐号

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【用户列表】，单击【新建用户】。

2. 在“新建用户”页面选择【自定义创建】，进入“新建子用户”页面。

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

3. 在“设置用户权限”页面：

   i. 搜索并勾选预设策略 `QcloudTRTCFullAccess` 。
   ii. 单击【下一步】。

4. 在“审阅信息和权限”分栏下单击【完成】，完成子用户的创建，在成功页面下载并保管好该子用户的登录链接和安全凭证，其中包含的信息如下表：

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

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

将 TRTC 全读写访问权限授予已存在的子帐号

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

解除子帐号的 TRTC 全读写访问权限

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

自定义策略

最近更新时间：2021-02-25 16:04:41

⚠ 注意：

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

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

自定义策略创建方法

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

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

说明：

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

授权策略语法

资源语法描述

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

- 实时音视频所有应用的策略语法描述

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

- 单个应用的策略语法描述

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

- 多个应用的策略语法描述

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

操作语法描述

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

- 实时音视频所有云 API 的策略语法描述

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

- 单个云 API 操作的策略语法描述

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

- 多个云 API 操作的策略语法描述

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

自定义策略使用示例

使用策略生成器

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

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【策略】，单击【新建自定义策略】。
2. 选择【按策略生成器创建】，进入策略创建页面。
3. 选择服务和操作。
   - 【效果(Effect)】配置项选择【允许】。
   - 【服务(Service)】配置项选择【实时音视频】。
   - 【操作(Action)】配置项勾选所有项。
   - 【资源(Resource)】配置项按照 资源语法描述 说明填写 `qcs::trtc::uin/12345678:sdkappid/1400000001` 。
   - 【条件(Condition)】配置项无需配置。
   - 单击【添加声明】，页面最下方会出现一条“允许对实时音视频应用1400000001进行任何操作”的声明。
4. 在同个页面中继续添加另一条声明。
   - 【效果(Effect)】配置项选择【拒绝】。
   - 【服务(Service)】配置项选择【实时音视频】。
   - 【操作(Action)】配置项勾选 `RemoveUser` （可通过搜索功能快速查找）。
   - 【资源(Resource)】配置项按照 资源语法描述 说明填写 `qcs::trtc::uin/12345678:sdkappid/1400000001` 。
   - 【条件(Condition)】配置项无需配置。
   - 单击【添加声明】，页面最下方会出现一条“拒绝对实时音视频应用1400000001进行 `RemoveUser` 操作”的声明。
5. 单击【下一步】，按需修改策略名称（也可以不修改）。
6. 单击【完成】完成自定义策略的创建。

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

使用策略语法

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

1. 以腾讯云 主账号 的身份访问 CAM 控制台的【策略】，单击【新建自定义策略】。

2. 选择【按策略语法创建】，进入策略创建页面。

3. 在【选择模板类型】框下选择【空白模板】。

说明：

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

4. 单击【下一步】，按需修改策略名称（也可以不修改）。

5. 在【编辑策略内容】编辑框中填写策略内容。本示例的策略内容为：

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

说明：

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

6. 单击【创建策略】完成自定义策略的创建。

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

使用 CAM 提供的服务端 API

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

- 创建策略
- 删除策略
- 绑定策略到用户
- 解除绑定到用户的策略

如何用 OBS WHIP 推流到 TRTC 房间

最近更新时间：2024-08-07 10:53:53

概览

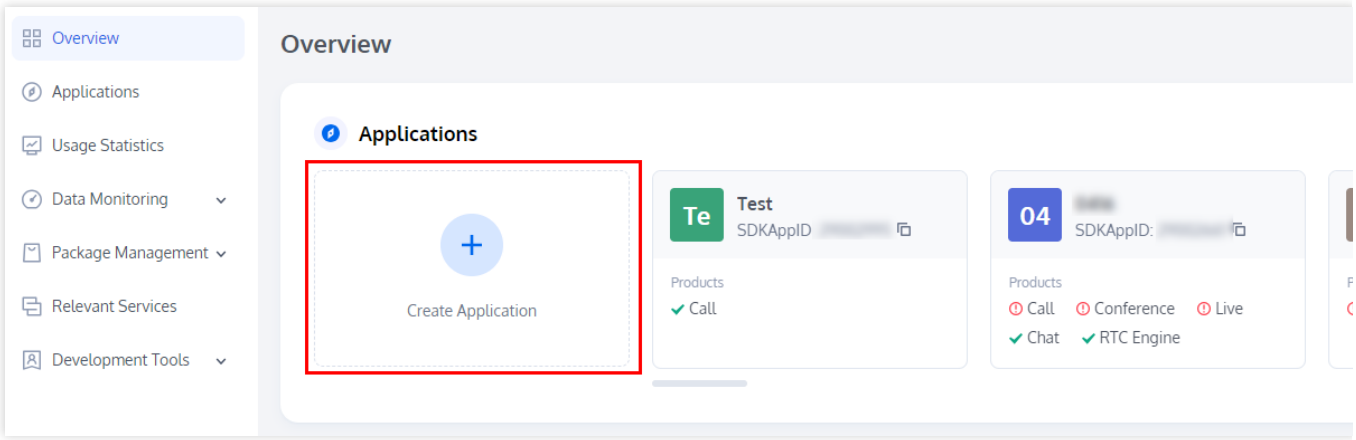
OBS 已经支持 WHIP 推流，这使您可以通过结合 OBS 和 WHIP 的功能来做许多有趣的事情。
WHIP 是一种标准协议，允许您使用 HTML5 和不同的客户端发布和播放实时流。此外，您可以使用开源工具构建自己的实时流媒体平台。
您还可以使用支持 OBS WHIP 的 TRTC（腾讯实时音视频）云服务作为流媒体平台。如果您不想构建自己的平台，或需要一个更可靠、稳定的平台，或者需要专业的技术支持，这是一个很好的选择。
此外，TRTC（腾讯实时音视频）提供免费试用，包括一定量的免费额度，使您可以轻松尝试。
如果您需要帮助或遇到任何问题，请随时在 Discord 上与我们联系。

前提条件

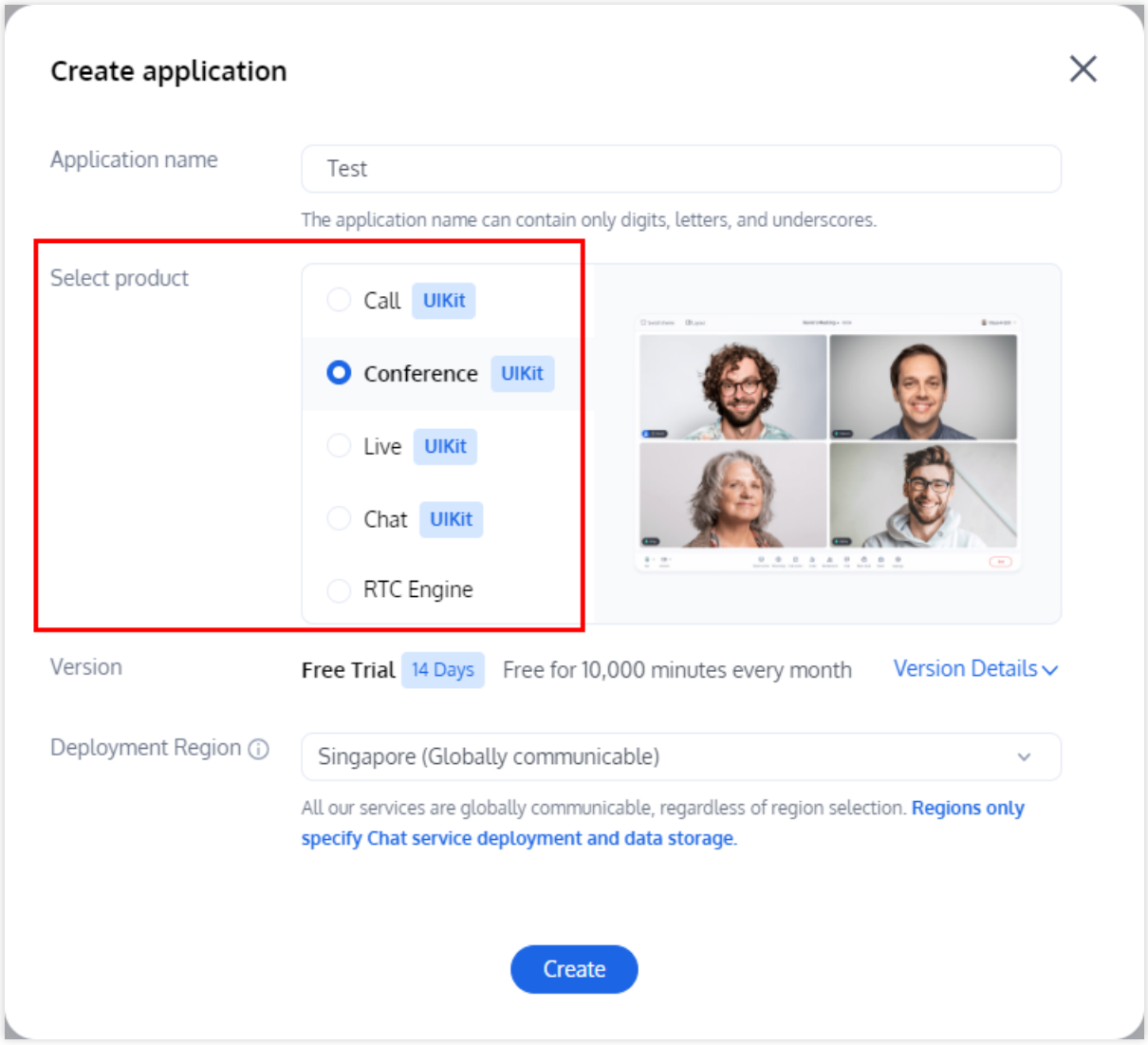
在您继续之前，请务必检查您已准备好以下必需的项目：
具有 WHIP 支持的 OBS，请从 OBS 下载。
腾讯实时音视频（TRTC）帐户，请在 此处 注册。
接下来，您需要创建一个 TRTC 应用程序，并为 WHIP 生成一个 Bearer 令牌。

第一步：创建一个 TRTC 应用

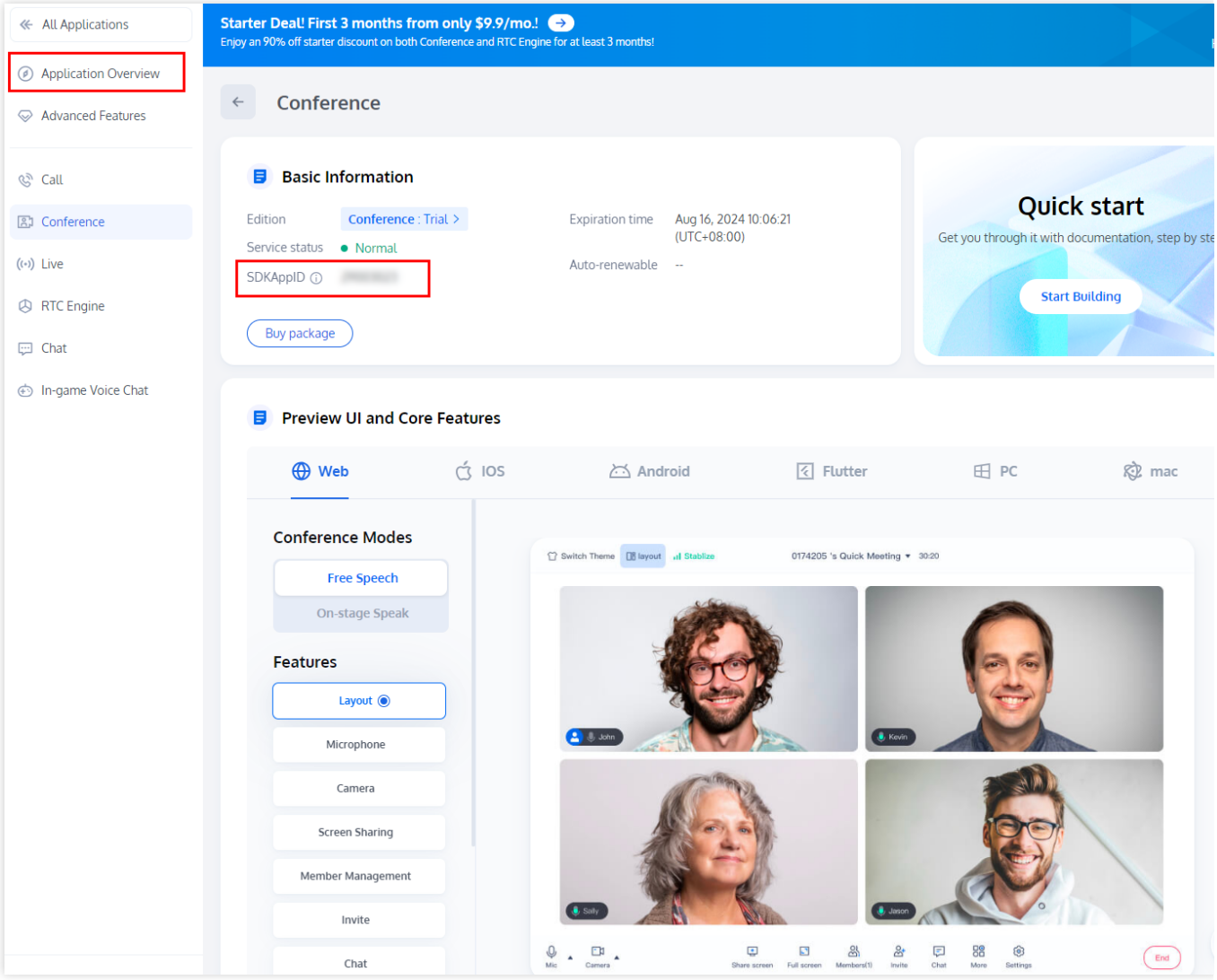
请按照以下步骤创建 TRTC 应用程序：
1. 登录 Tencent RTC 控制台，点击 **Create Application**。



2. 在创建弹窗中实际业务需求选择产品并输入应用名称，选择数据存储**地区**，点击 **Create**。



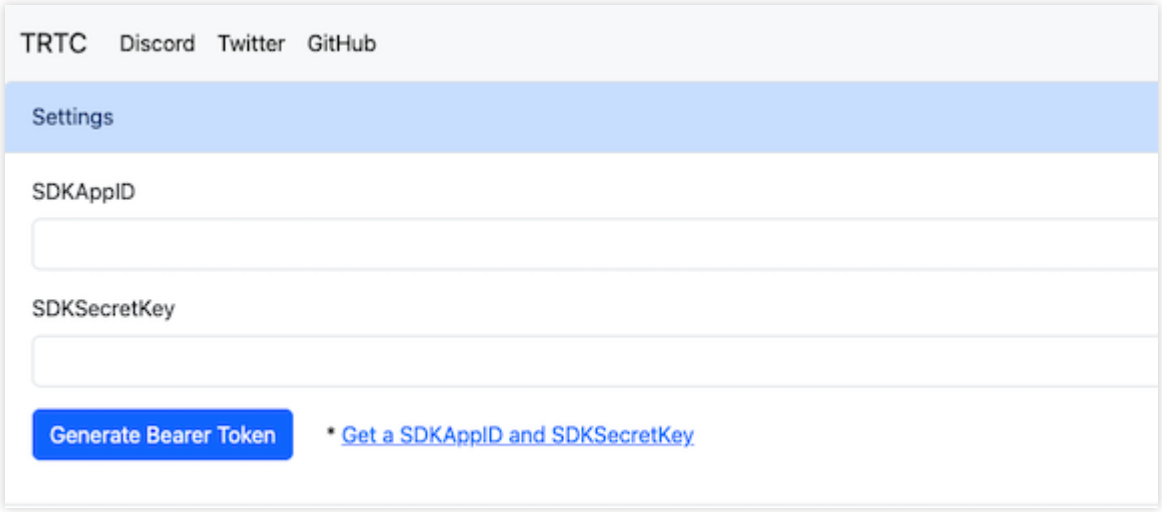
3. 应用程序创建后，默认进入所选产品详情页。您可在 **Application Overview** 查看 `SDKAppID` 和 `SDKSecretKey`， 会在后续步骤中使用到。



第二步：创建一个 WHIP Bearer Token

接下来，您必须为 WHIP 生成一个 Bearer Token，该 Token 将在 OBS 中使用。

您可以直接访问 https://tencent-rtc.github.io/obs-trtc/bearer.html 来创建 WHIP Bearer Token。确保使用您自己的 **SDKAppID** 和 **SDKSecretKey** 作为 appid 和 secret，然后点击 `Generate Bearer Token` 按钮。



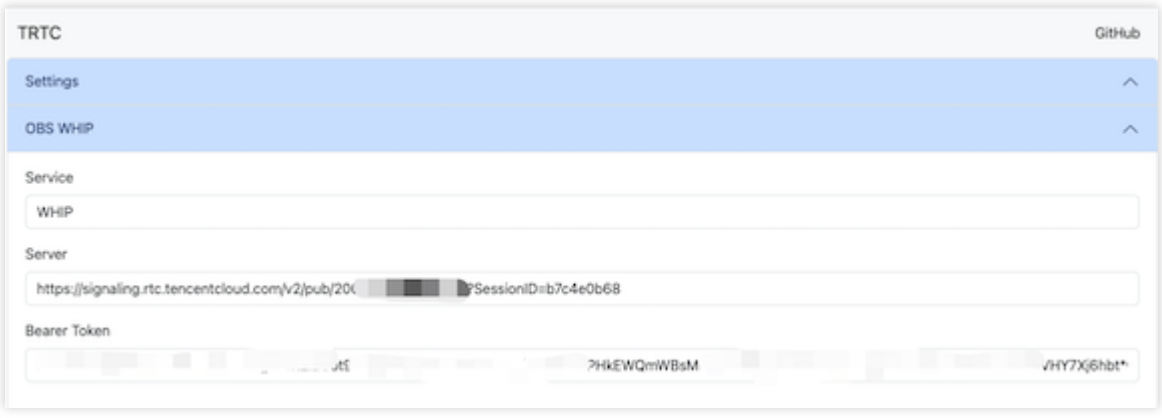
说明：

您还可以访问网址 `https://tencent-rtc.github.io/obs-trtc/bearer.html?appid=2000xxx&secret=yyyyyy` 来设置参数。

接下来，使用生成的 WHIP Bearer Token 来配置 OBS。

第三步：配置 OBS

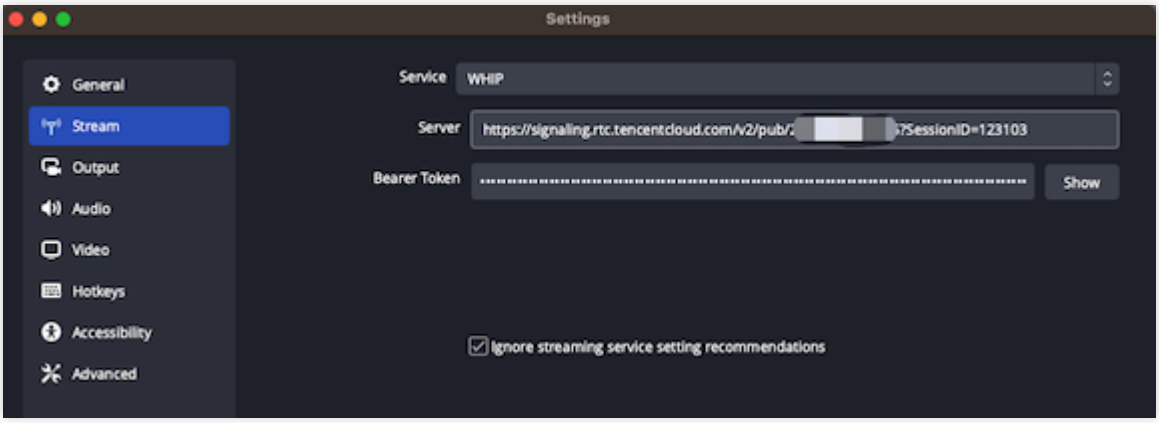
在 `OBS WHIP` 部分，您将找到生成的 WHIP `Server（服务器）` 和 `Bearer Token`，用于配置 OBS。



请按照以下步骤配置 OBS：

1. 打开 OBS 并点击**设置**。
2. 点击左侧边栏的**直播**。
3. 选择 **WHIP** 作为服务。
4. 确保准确输入**服务器**和 **Bearer Token**。
5. 点击**确定**以保存设置。

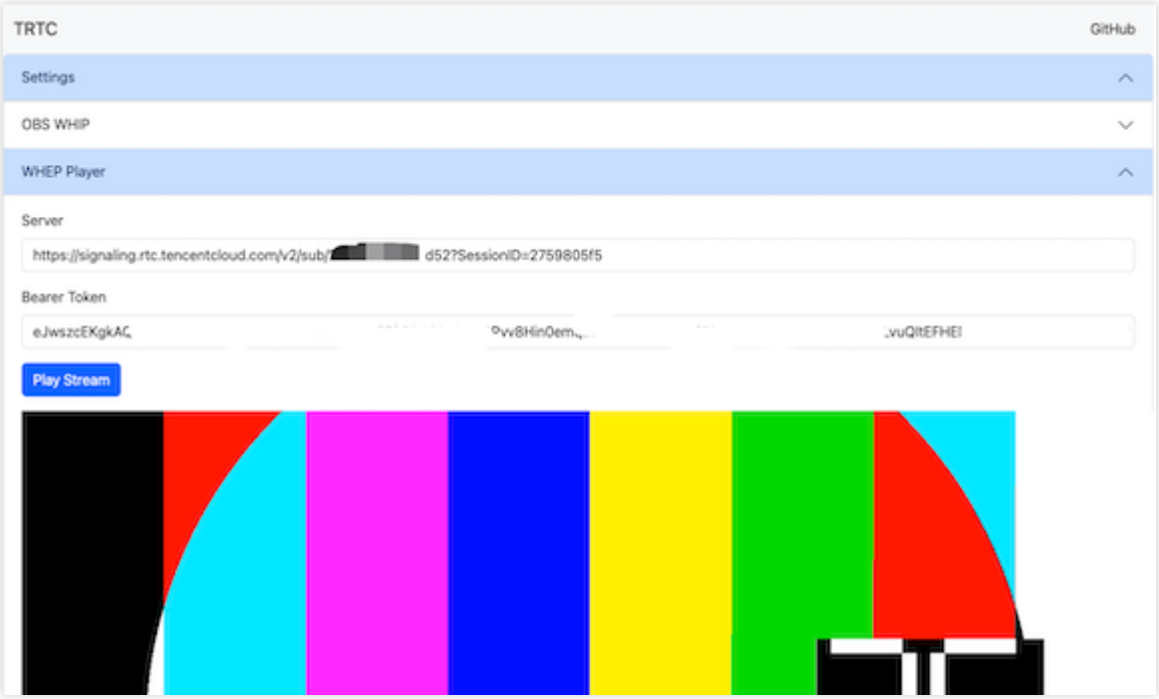
6. 点击**开始直播**。



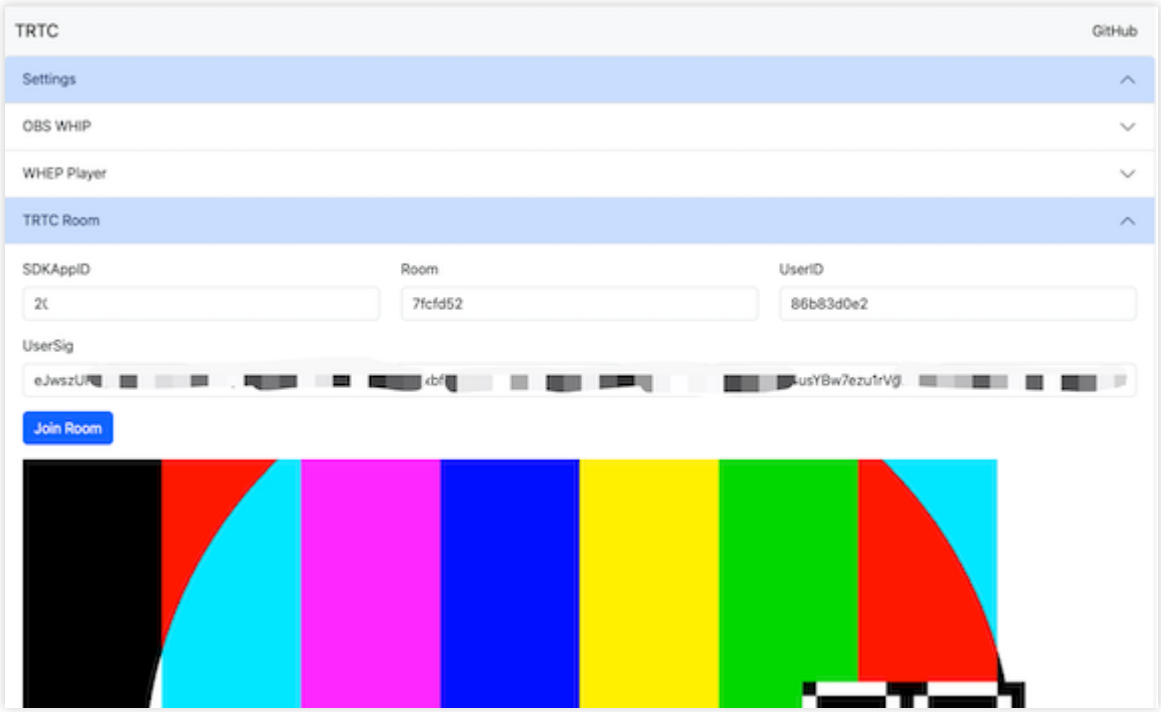
现在，流就已经推到了 TRTC 服务了。

第四步：播放流

打开之前的网页，转到 `WHEP Player` 部分，然后点击**Play Stream**就可以通过WHEP播放流。



另一个选择是在 `TRTC Room` 部分，然后点击**Join Room**，就可以通过TRTC观看流，或者您可以使用TRTC移动SDK加入房间并查看流。



由于WHIP和WHEP都是标准协议，您可以使用支持它们的任何客户端来播放流。

总结

我们展示了使用OBS WHIP 推流到TRTC（腾讯实时音视频）云服务，来构建更强大的流媒体应用。可以使用这项技术，在不同场景下实现超低延迟的流媒体能力，可以将 OBS 的能力拓展到不同的新的领域。
如果您需要帮助或遇到任何问题，请随时在 Discord 上与我们联系。