实时音视频

基础功能

产品文档

腾讯云

文档目录

基础功能
跑通通话模式
跑通通话模式(iOS&Mac)

最近更新时间：2022-03-09 16:42:59

适用场景

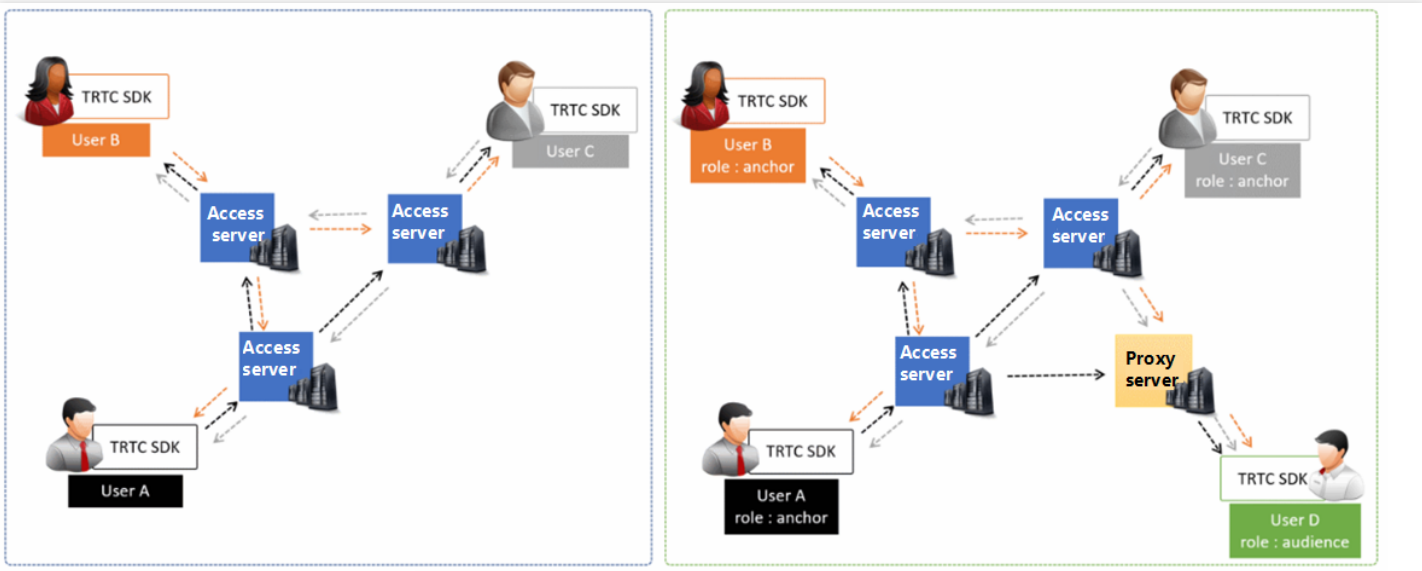
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（AudioCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

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

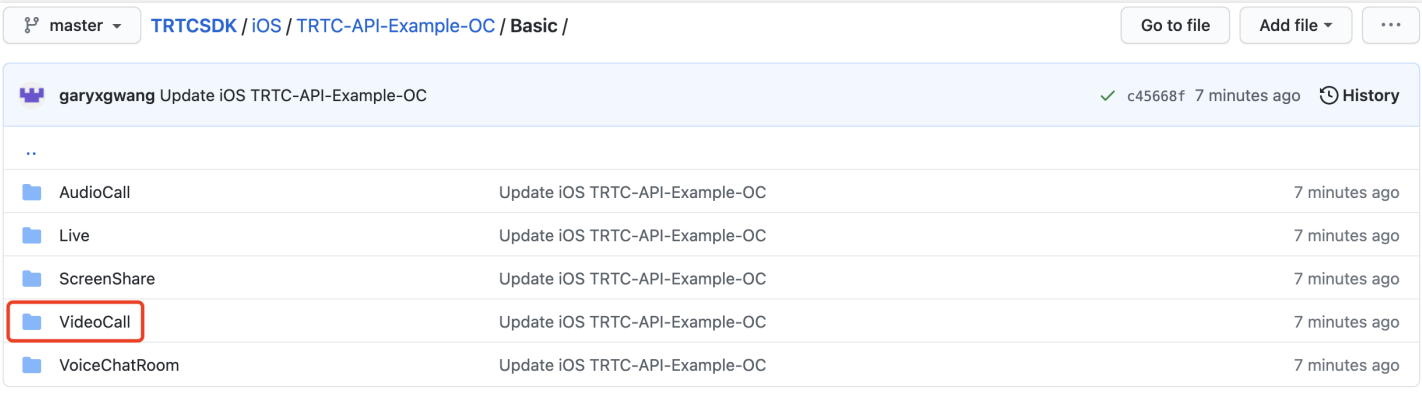
- **接口机**
  该类节点都采用最优质的线路和高性能的机器，善于处理端到端的低延时连麦通话，单位时长计费较高。
- **代理机**
  该类节点都采用普通的线路和性能一般的机器，善于处理高并发的拉流观看需求，单位时长计费较低。

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



示例代码

您可以登录 Github 获取本文档相关的示例代码。



说明：

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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 **TRTC SDK** 集成到项目中。

方式一：使用 CocoaPods 集成

1. 安装 **CocoaPods**，具体操作请参考 CocoaPods 官网安装说明。

2. 打开您当前项目根目录下的 `Podfile` 文件，添加以下内容：

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

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

3. 执行以下命令安装 **TRTC SDK**。

```
pod install
```

安装成功后当前项目根目录下会生成一个 **xcworkspace** 文件。
4. 打开新生成的 **xcworkspace** 文件即可。

方式二：下载 ZIP 包手动集成

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

步骤2：添加媒体设备权限

在 `Info.plist` 文件中添加摄像头和麦克风的申请权限：

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

步骤3：初始化 SDK 实例并监听事件回调

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

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

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

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

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1SsII...
roomId	数字	数字类型的房间号。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。	29834

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

步骤5：创建并进入房间

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

- 视频通话，请设置为 `TRTCAppScene.videoCall` 。
- 语音通话，请设置为 `TRTCAppScene.audioCall` 。

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

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

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

注意：

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

步骤6：订阅远端的音视频流

SDK 支持自动订阅和手动订阅。

自动订阅模式（默认）

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

1. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知，SDK 会自动播放这些远端用户的声音。
2. 您可以通过 muteRemoteAudio(userId, mute: true) 屏蔽某一个 userId 的音频数据，也可以通过 muteAllRemoteAudio(true) 屏蔽所有远端用户的音频数据，屏蔽后 SDK 不再继续拉取对应远端用户的音频数据。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知，但此时 SDK 未收到该如何展示视频数据的指令，因此不会自动处理视频数据。您需要通过调用 startRemoteView(userId, view: view) 方法将远端用户的视频数据和显示 `view` 关联起来。
4. 您可以通过 setRemoteViewFillMode() 指定视频画面的显示模式：

- Fill 模式：表示填充，画面可能会等比放大和裁剪，但不会有黑边。
- Fit 模式：表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

5. 您可以通过 stopRemoteView(userId) 可以屏蔽某一个 userId 的视频数据，也可以通过 stopAllRemoteView() 屏蔽所有远端用户的视频数据，屏蔽后 SDK 不再继续拉取对应远端用户的视频数据。

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

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

手动订阅模式

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

1. 在**进房前**调用 setDefaultStreamRecvMode(false, video: false) 接口将 SDK 设定为手动订阅模式。

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

步骤7：发布本地的音视频流

1. 调用 startLocalAudio() 可以开启本地的麦克风采集，并将采集到的声音编码并发送出去。
2. 调用 startLocalPreview() 可以开启本地的摄像头，并将采集到的画面编码并发送出去。
3. 调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：

- Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
- Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

4. 调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。

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

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

步骤8：退出当前房间

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

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

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

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

跑通通话模式(Android)

最近更新时间：2022-03-09 16:34:33

适用场景

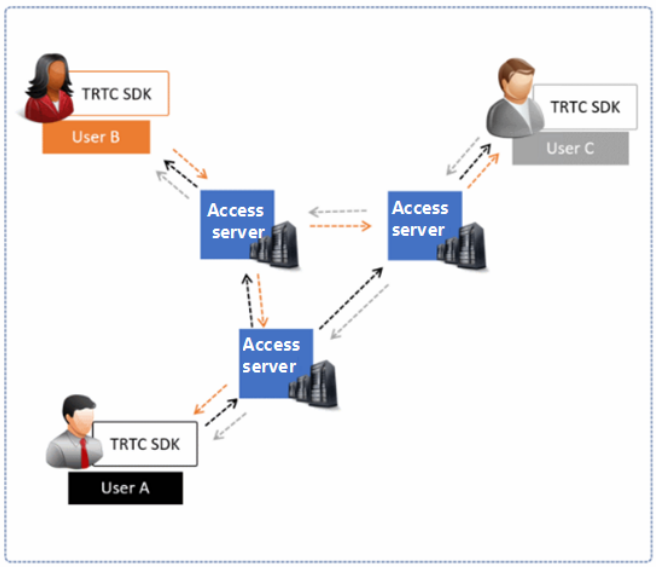
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（AudioCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

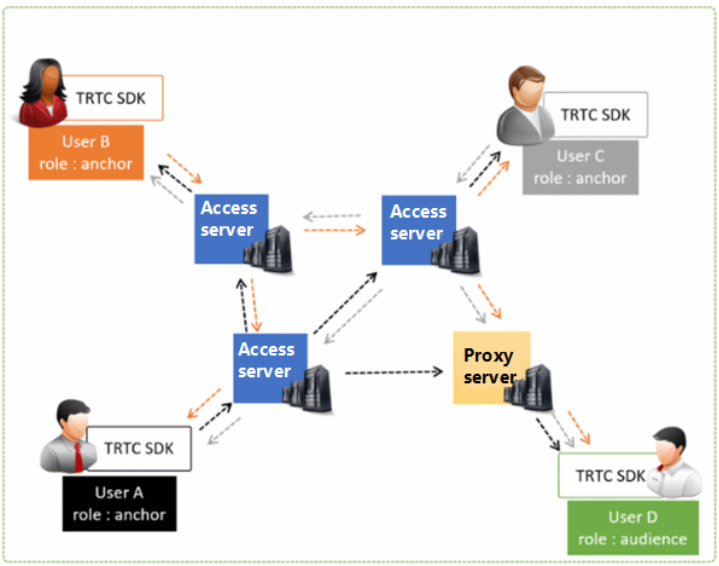
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

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

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



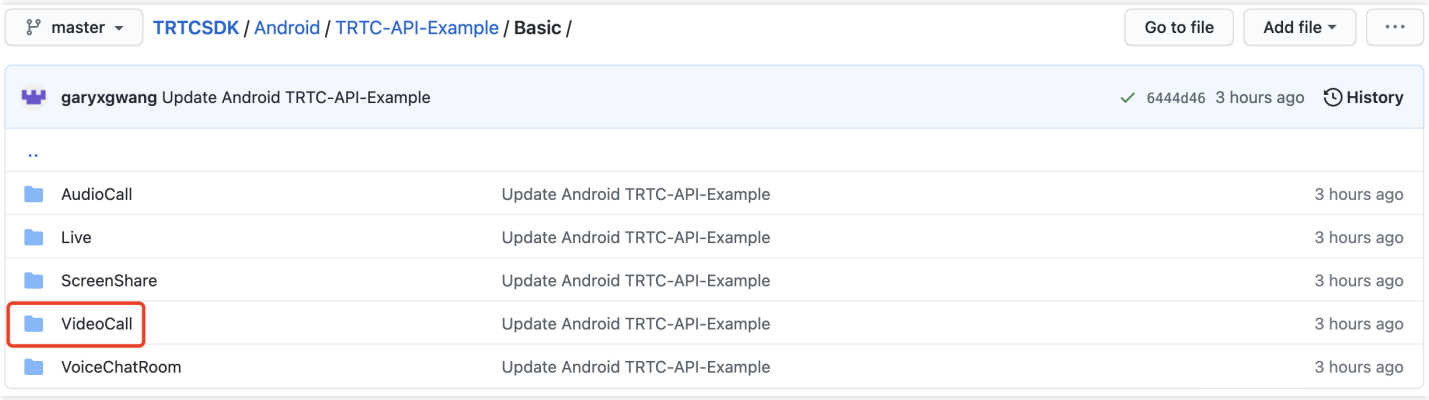
Call mode (no role differentiation)



Live streaming mode (two roles: anchor and viewer)

示例代码

您可以登录 Github 获取本文档相关的示例代码。



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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 **TRTC SDK** 集成到项目中。

方式一：自动加载（aar）

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

1. 在 dependencies 中添加 TRTCSDK 的依赖。

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

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

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

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

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

方式二：下载 ZIP 包手动集成

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

步骤2：配置 App 权限

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

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

步骤3：初始化 SDK 实例并监听事件回调

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

```
// 创建 trtcCloud 实例
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener(){
```

```
// 回调处理
...
});
```

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

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

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	140000
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_us
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 [如何计算及使用 UserSig]](https://intl.cloud.tencent.com/document/product/647/35166)。	eJyrVar
roomId	数字	数字类型的房间号。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。	29834

注意：

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

步骤5：创建并进入房间

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

- 视频通话，请设置为 `TRTC_APP_SCENE_VIDEOCALL` 。
- 语音通话，请设置为 `TRTC_APP_SCENE_AUDIOCALL` 。

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

```
public void enterRoom() {
TRTCCloudDef.TRTCParams trtcParams = new TRTCCloudDef.TRTCParams();
trtcParams.sdkAppId = sdkappid;
trtcParams.userId = userid;
trtcParams.roomId = 908;
trtcParams.userSig = usersig;
mTRTCCloud.enterRoom(trtcParams, TRTC_APP_SCENE_VIDEOCALL);
}
@Override
public void onEnterRoom(long result) {
if (result > 0) {
toastTip("进房成功，总计耗时[¥(result)]ms")
} else {
toastTip("进房失败，错误码[¥(result)]")
}
}
```

注意：

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

步骤6：订阅远端的音视频流

SDK 支持自动订阅和手动订阅。

自动订阅模式（默认）

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

1. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知，SDK 会自动播放远端用户的声音。
2. 您可以通过 muteRemoteAudio(userId, true) 屏蔽某一个 userId 的音频数据，也可以通过 muteAllRemoteAudio(true) 屏蔽所有远端用户的音频数据，屏蔽后 SDK 不再继续拉取对应远端用户的音频数据。
3. 当房间中有其他用户在上行视频数据时，您会收到 onUserVideoAvailable() 事件通知，但此时 SDK 未收到该如何展示视频数据的指令，因此不会自动处理视频数据。您需要通过调用 startRemoteView(userId, view) 方法将远端用户的视频数据和显示 `view` 关联起来。
4. 您可以通过 setRemoteViewFillMode() 指定视频画面的显示模式：

- Fill 模式：表示填充，画面可能会等比放大和裁剪，但不会有黑边。
- Fit 模式：表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

5. 您可以通过 stopRemoteView(userId) 屏蔽某一个 userId 的视频数据，也可以通过 stopAllRemoteView() 屏蔽所有远端用户的视频数据，屏蔽后 SDK 不再继续拉取对应远端用户的视频数据。

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

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

手动订阅模式

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

1. 在**进房前**调用 setDefaultStreamRecvMode(false, false) 接口将 SDK 设定为手动订阅模式。

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

步骤7：发布本地的音视频流

1. 调用 startLocalAudio() 可以开启本地的麦克风采集，并将采集到的声音编码并发送出去。
2. 调用 startLocalPreview() 可以开启本地的摄像头，并将采集到的画面编码并发送出去。
3. 调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：

- Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
- Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

4. 调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。

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

步骤8：退出当前房间

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

```
// 调用退房后请等待 onExitRoom 事件回调
mTRTCCloud.exitRoom()
@Override
public void onExitRoom(int reason) {
Log.i(TAG, "onExitRoom: reason = " + reason);
}
```

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

跑通通话模式(Windows)

最近更新时间：2022-01-10 10:42:47

文档导读

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

示例代码

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

视频通话

1. 初始化 SDK

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

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

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

- C++
- C#

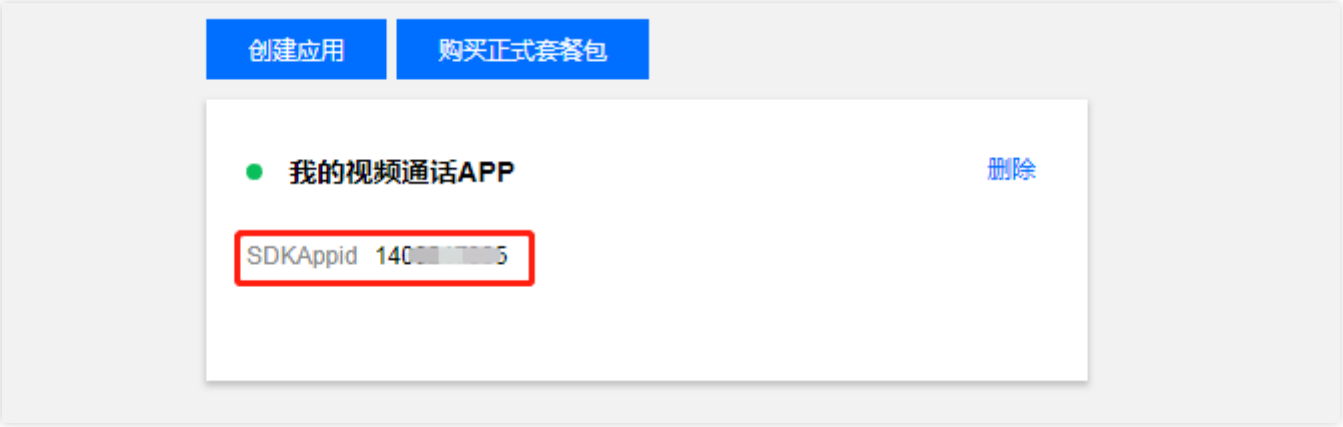
```
// TRTCMainViewController.h

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

virtual void onError(TXLiteAVError errCode, const char* errMsg, void* arg);
virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void* arg);
virtual void onEnterRoom(int result);
virtual void onExitRoom(int reason);
virtual void onRemoteUserEnterRoom(const char* userId);
virtual void onRemoteUserLeaveRoom(const char* userId, int reason);
virtual void onUserVideoAvailable(const char* userId, bool available);
virtual void onUserAudioAvailable(const char* userId, bool available);
...
private:
ITRTCCloud * m_pTRTCSDK = NULL ;
...
}

// TRTCMainViewController.cpp

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

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

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

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

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

2. 组装 TRTCParams

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

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



- **userId**
  您可以随意指定，由于是字符串类型，可以直接跟您现有的账号体系保持一致，但请注意，**同一个音视频房间里不应该有两个同名的 userId**。

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

- **roomId**
  房间号是数字类型，您可以随意指定，但请注意，**同一个应用里的两个音视频房间不能分配同一个 roomId**。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。

3. 进入（或创建）房间

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

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

- 如进房成功，SDK 会回调 `onEnterRoom` 接口，参数：当 `result` 大于0时，进房成功，数值表示加入房间所消耗的时间，单位为毫秒（ms）；当 `result` 小于0时，进房失败，数值表示进房失败的错误码。
- 如进房失败，SDK 同时会回调 `onError` 接口，参数：`errCode`（错误码 `ERR_ROOM_ENTER_FAIL`，错误码可参考 `TXLiteAVCode.h`）、`errMsg`（错误原因）、`extraInfo`（保留参数）。
- 如果已在房间中，则必须调用 `exitRoom` 方法退出当前房间，才能进入下一个房间。

- C++
- C#

```
// TRTCMainViewController.cpp

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

...

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

...
```

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

注意：

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

4. 收听远端音频流

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

5. 观看远端视频流

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

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

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

- C++
- C#

```
// TRTCMainViewController.cpp
void TRTCMainViewController::onUserVideoAvailable(const char* userId, bool available){
```

```
if (available) {
// 获取渲染窗口的句柄。
CWnd *pRemoteVideoView = GetDlgItem(IDC_REMOTE_VIDEO_VIEW);
HWND hwnd = pRemoteVideoView-&gt;GetSafeHwnd();

// 设置远端用户视频的渲染模式。
m_pTRTCSDK-&gt;setRemoteViewFillMode(TRTCVideoFillMode_Fill);
// 调用 SDK 接口播放远端用户流。
m_pTRTCSDK-&gt;startRemoteView(userId,  hwnd);
} else {
m_pTRTCSDK-&gt;stopRemoteView(userId);
}
}
```

6. 开关本地声音采集

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

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

7. 开关本地视频采集

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

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

- C++
- C#

```
// TRTCMainViewController.cpp

void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
```

```
{
...

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

if(m_pTRTCSDK)
{
// 调用 SDK 接口设置渲染模式和渲染窗口。
m_pTRTCSDK-&gt;setLocalViewFillMode(TRTCVideoFillMode_Fit);
m_pTRTCSDK-&gt;startLocalPreview(hwnd);
}

...
}
```

8. 屏蔽音视频数据流

- **屏蔽本地视频数据**
  如果用户在通话过程中，出于隐私目的希望屏蔽本地的视频数据，让房间里的其他用户暂时无法看到您的画面，可以调用 `muteLocalVideo` 。

- **屏蔽本地音频数据**
  如果用户在通话过程中，出于隐私目的希望屏蔽本地的音频数据，让房间里的其他用户暂时无法听到您的声音，可以调用 `muteLocalAudio` 。

- **屏蔽远程视频数据**
  通过 `stopRemoteView` 可以屏蔽某一个 userid 的视频数据。
  通过 `stopAllRemoteView` 可以屏蔽所有远端用户的视频数据。

- **屏蔽远程音频数据**
  通过 `muteRemoteAudio` 可以屏蔽某一个 userid 的音频数据。
  通过 `muteAllRemoteAudio` 可以屏蔽所有远端用户的音频数据。

9. 退出房间

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

> 说明：

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

- C++
- C#

```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;exitRoom();
}
}
....
void TRTCMainViewController::onExitRoom(int reason)
{
// 退房成功，reason 参数保留，暂未使用。

...
}
```

跑通通话模式(Electron)

最近更新时间：2022-01-10 10:44:40

适用场景

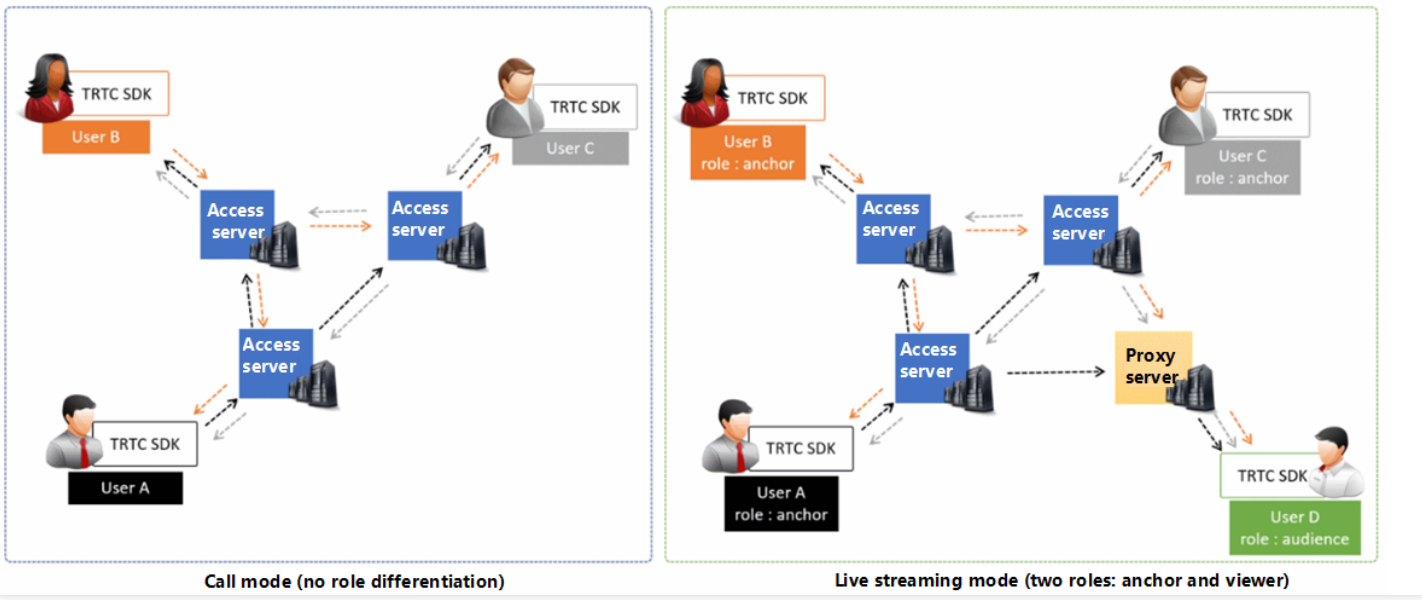
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

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

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

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



示例代码

您可以登录 Github 获取本文档相关的示例代码。

操作步骤

步骤1：尝试跑通官网 SimpleDemo

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

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

步骤2：为您的项目集成 trtc-electron-sdk

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

- 您可以在我们的官方 Demo 的基础上进行二次开发，项目的起步阶段会比较顺利。
- 您也可以执行以下指令，把 `trtc-electron-sdk` 安装到您现有的项目中：

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

步骤3：初始化 SDK 实例并监听事件回调

1. 创建 `trtc-electron-sdk` 实例：

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

2. 监听 `onError` 事件：

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

步骤4： 组装进房参数 TRTCParams

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

参数	类型	说明	示例
sdkAppId	数字	应用 ID，您可以在 控制台 >【应用管理】>【应用信息】中查找到。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1SsII...
roomId	数字	数字类型的房间号。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。	29834

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

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
```

注意：

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

步骤5：创建并进入房间

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

> 说明：
> 关于 `TRTCAppScene` 的详细介绍，请参见 TRTCAppScene 。

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

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

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

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

// 进房, 如果房间不存在, TRTC 后台会自动创建一个新房间
let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneVideoCall);
```

步骤6: 订阅远端的音视频流

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

自动订阅（推荐）

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

1. 当房间中有其他用户在上行音频数据时，您会收到 onUserAudioAvailable() 事件通知，SDK 会自动播放这些远端用户的声音。

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

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

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

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

let onUserVideoAvailable = function (uid, available) {

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

```
let view = document.getElementById(id);
if (view) {
videoContainer.removeChild(view);
}
}
};

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

</script>
```

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

手动订阅

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

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

步骤7：发布本地的音视频流

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

```
//示例代码：发布本地的音视频流
trtcCloud.startLocalPreview(view);
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);
trtcCloud.startLocalAudio();
//设置本地视频编码参数
let encParam = new TRTCVideoEncParam();
encParam.videoResolution = TRTCVideoResolution.TRTCVideoResolution_640_360;
encParam.resMode = TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape;
encParam.videoFps = 25;
encParam.videoBitrate = 600;
encParam.enableAdjustRes = true;
trtcCloud.setVideoEncoderParam(encParam);
```

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

步骤8：退出当前房间

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

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

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

跑通通话模式(Web)

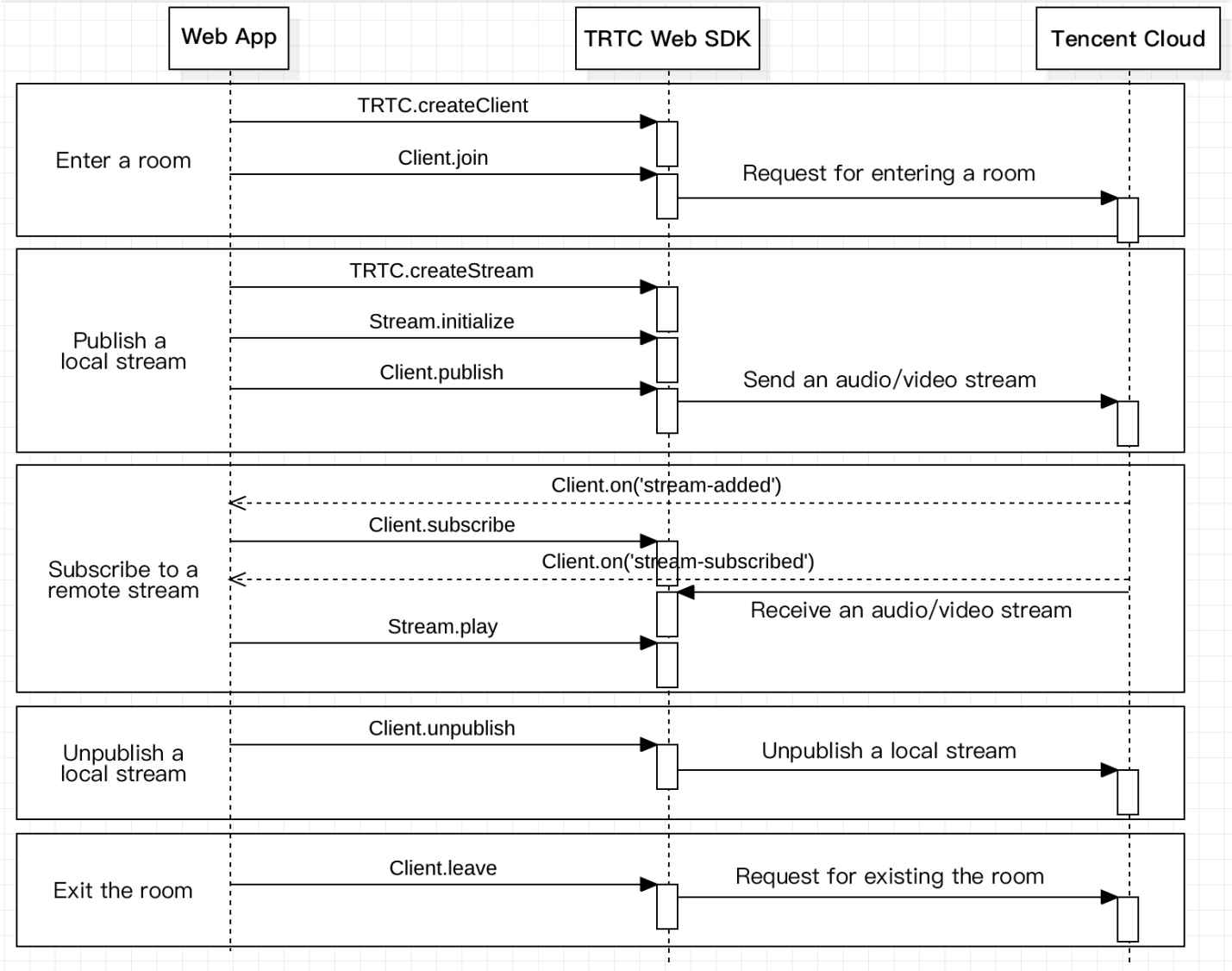
最近更新时间：2022-03-10 09:56:42

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

在使用 TRTC Web SDK 中，经常会接触到以下对象：

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

基本音视频通话的 API 调用流程如下图所示：



步骤1：创建 Client 对象

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

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

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

步骤2：进入音视频通话房间

调用 Client.join() 进入音视频通话房间。

参数 `roomId` ：房间 ID

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

步骤3：发布本地流和订阅远端流

1. 使用 TRTC.createStream() 方法创建本地音视频流。
   以下实例从摄像头及麦克风中采集音视频流，参数设置如下：

- `userId` ：本地流所属的用户 ID
- `audio` ：是否开启音频

- `video` ：是否开启视频

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

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

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

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

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

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

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

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

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

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

- 初始化本地流成功时播放本地流

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

- 订阅远端流成功时播放远端流

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

步骤4：退出音视频通话房间

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

```
client
.leave()
.then(() => {
// 退房成功，可再次调用client.join重新进房开启新的通话。
```

```
})
.catch(error => {
console.error('退房失败 ' + error);
// 错误不可恢复，需要刷新页面。
});
```

注意：

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

实时屏幕分享
实时屏幕分享(iOS)

最近更新时间：2022-03-09 16:24:00

腾讯云 TRTC 在 iOS 平台下支持两种不同的屏幕分享方案：

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

支持的平台

iOS	Android	Mac OS	Windows	Electron	Chrome 浏览器
✓	✓	✓	✓	✓	✓

应用内分享

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

我们推荐的用于 iOS 屏幕分享的编码参数是：

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

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。

- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

跨应用分享

示例代码

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

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

您可以通过 README 中的指引跑通该示例 Demo。

对接步骤

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

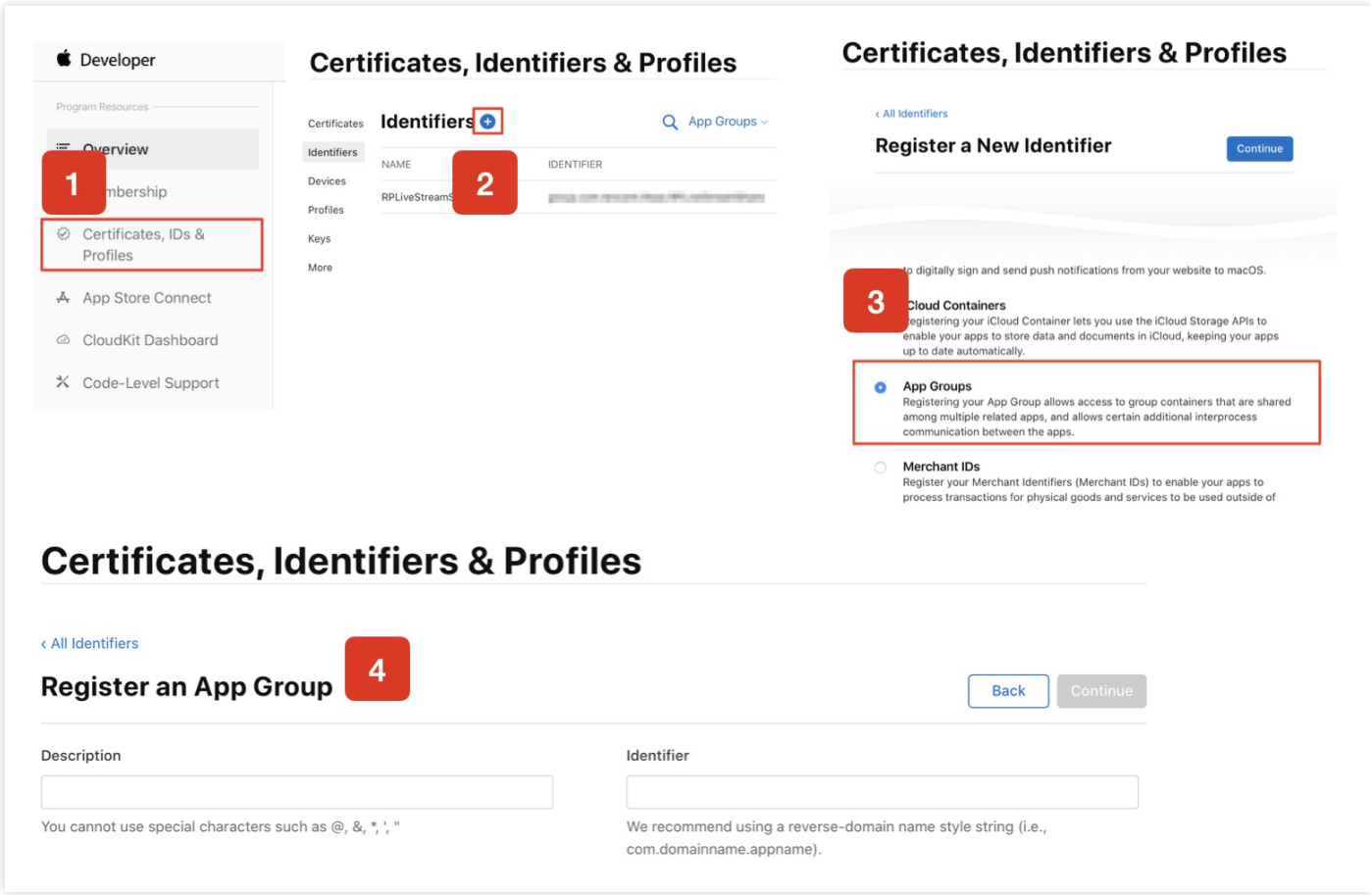
1. 创建 App Group，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 Broadcast Upload Extension 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 `TXLiteAVSDK_ReplayKitExt.framework` 。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 Broadcast Upload Extension 的录屏数据。
4. 使用 Demo 中预先实现的一个 helper class (`RPSystemBroadcastPickerView`)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

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

步骤1：创建 App Group

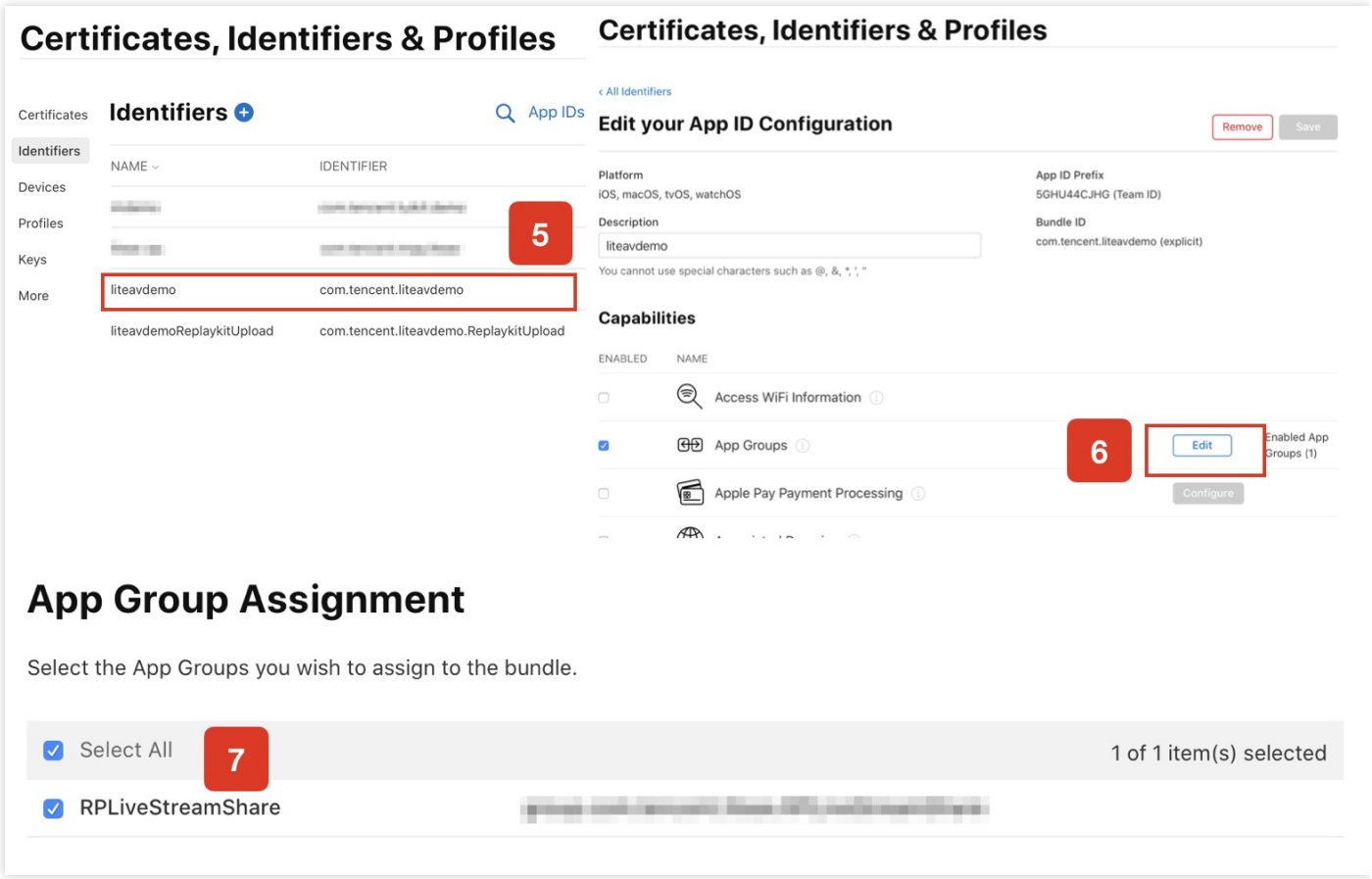
使用您的帐号登录 **https://developer.apple.com/**，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

1. 单击【Certificates, IDs & Profiles】。
2. 在右侧的界面中单击加号。
3. 选择【App Groups】，单击【Continue】。
4. 在弹出的表单中填写 Description 和 Identifier, 其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击【Continue】。



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

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

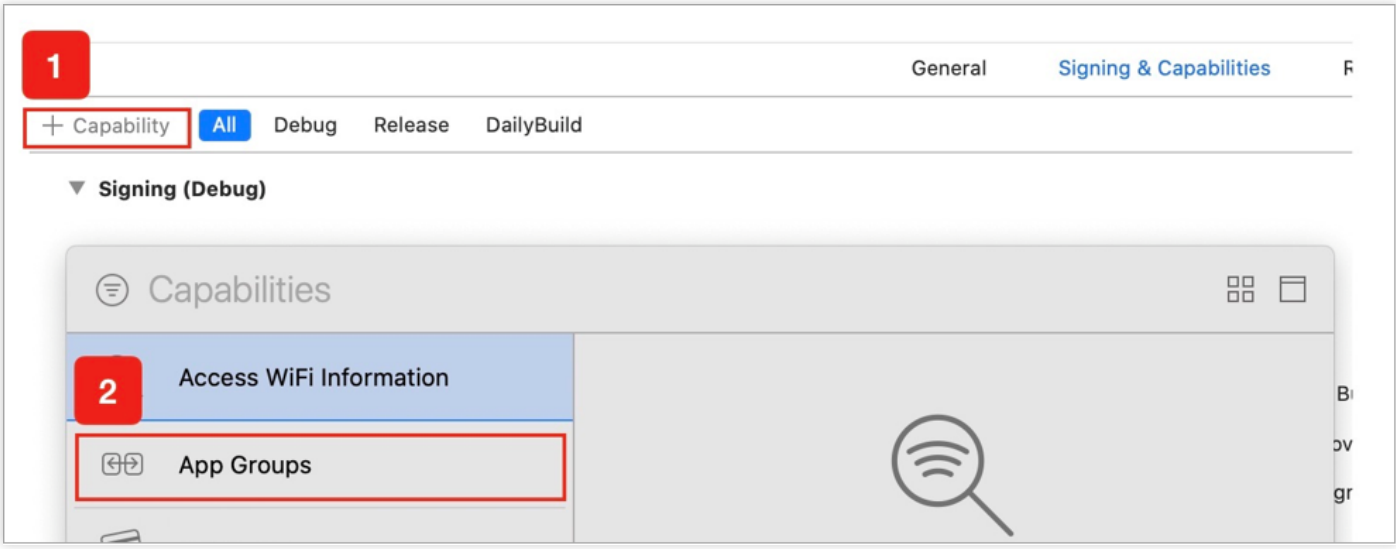


8. 重新下载 Provisioning Profile 并配置到 XCode 中。

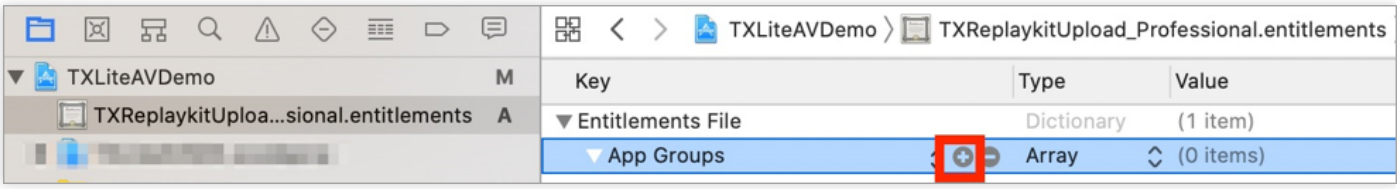
步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击【File】、【New】、【Target...】，选择【Broadcast Upload Extension】。
2. 在弹出的对话框中填写相关信息，**不用**勾选"【Include UI Extension】，单击【Finish】完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。

4. 选中新增加的 Target，依次单击【+ Capability】，双击【App Groups】，如下图：



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



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

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

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

@interface SampleHandler() <txreplaykitextdelegate>
@end

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

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

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

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

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

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

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampl
eBufferType {
switch (sampleBufferType) {
case RPSampleBufferTypeVideo:
[[TXReplayKitExt sharedInstance] sendVideoSampleBuffer:sampleBuffer];
break;
case RPSampleBufferTypeAudioApp:
// Handle audio sample buffer for app audio
break;
case RPSampleBufferTypeAudioMic:
// Handle audio sample buffer for mic audio
break;

default:
```

```
break;
}
}
@end
```

步骤3：对接主 App 端的接收逻辑

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

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

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

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

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

步骤4：增加屏幕分享的触发按钮（可选）

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

1. 在 Demo 中寻找 `TRTCBroadcastExtensionLauncher` 这个类，并将其加入到您的工程中。

2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 `TRTCBroadcastExtensionLauncher` 中的 `launch` 函数，就可以唤起屏幕分享功能了。

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

注意：

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

观看屏幕分享

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

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

实时屏幕分享(Android)

最近更新时间：2022-03-09 15:46:40

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

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

支持的平台

iOS	Android	Mac OS	Windows	Electron	Chrome 浏览器
✓	✓	✓	✓	✓	✓

启动屏幕分享

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

添加 Activity

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

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

设定视频编码参数

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

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

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

弹出悬浮窗以避免被强杀

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

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

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

- **方案2：弹出摄像头预览窗**
  由于 TRTC Android 8.6之前的版本屏幕分享并不像桌面端版本一样支持“辅路分享”，因此在启动屏幕分享时，

摄像头这一路的视频数据无法上行，否则会相互冲突。8.6及之后版本支持“辅路分享”，则无此冲突，但一样可以使用以下方法。

那要如何才能做到同时分享屏幕和摄像头画面呢？

答案很简单：只需要在屏幕上悬浮一个摄像头画面即可，这样一来，TRTC 在采集屏幕画面的同时也会将摄像头画面一并分享出去。

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**

  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserSubStreamAvailable 事件获得这个通知。

  希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。

- **观看 Android / iOS 屏幕分享**

  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudListener 中的 onUserVideoAvailable 事件获得这个通知。

  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

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

常见问题

一个房间里可以同时有多路屏幕分享吗？

目前一个 TRTC 音视频房间只能有一路屏幕分享。

实时屏幕分享(Windows)

最近更新时间：2021-11-08 11:39:29

腾讯云 TRTC 支持屏幕分享功能，Windows 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（**substream**）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeSub` 来启用该模式。观看该路画面需要使用专门的 `startRemoteSubStreamView` 接口。

- **主路分享**
  在 TRTC 中，我们一般把摄像头走的通道叫做“主路（**bigstream**）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeBig` 来启用该模式。

支持的平台

iOS	Android	Mac OS	Windows	Electron	Chrome 浏览器
✓	✓	✓	✓	✓	✓

依赖的 API

API 功能	C++ 版本	C# 版本	Electron 版本
选择分享目标	selectScreenCaptureTarget	selectScreenCaptureTarget	selectScreenCaptureTarget
开始屏幕分享	startScreenCapture	startScreenCapture	startScreenCapture

API 功能	C++ 版本	C# 版本	Electron 版本
暂停屏幕分享	pauseScreenCapture	pauseScreenCapture	pauseScreenCapture
恢复屏幕分享	resumeScreenCapture	resumeScreenCapture	resumeScreenCapture
结束屏幕分享	stopScreenCapture	stopScreenCapture	stopScreenCapture

获取分享目标

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

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

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

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

字段	类型	含义
iconHeight	int32	窗口图标的高度
iconBGRA	buffer	窗口图标的二进制 buffer

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

选择分享目标

TRTC SDK 支持三种分享模式，您可以通过 `selectScreenCaptureTarget` 来指定：

- **整个屏幕分享**：
  即分享整个屏幕窗口，支持多显示器分屏的情况。需要指定一个 sourceInfoList 中 type 为 `TRTCScreenCaptureSourceTypeScreen` 的 source 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

- **指定区域分享**：
  即分享屏幕的某个区域，需要用户圈定区域的位置坐标。需要指定一个 sourceInfoList 中 type 为 `TRTCScreenCaptureSourceTypeScreen` 的 source 参数，并将 captureRect 设为非 NULL，例如 { 100, 100, 300, 300 }。

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

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

开始屏幕分享

- 选取分享目标后，使用 `startScreenCapture` 接口可以启动屏幕分享。
- 分享过程中，您依然可以通过调用 `selectScreenCaptureTarget` 更换分享目标。

- `pauseScreenCapture` 和 `stopScreenCapture` 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resume。

```
/**
* ¥brief 7.5 【屏幕共享】启动屏幕分享
* ¥param：rendHwnd - 承载预览画面的 HWND
*/
void startScreenCapture(HWND rendHwnd);
/**
* ¥brief 7.6 【屏幕共享】暂停屏幕分享
*/
void pauseScreenCapture();
/**
* ¥brief 7.7 【屏幕共享】恢复屏幕分享
*/
void resumeScreenCapture();
/**
* ¥brief 7.8 【屏幕共享】关闭屏幕分享
*/
void stopScreenCapture();
```

设定画面质量

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

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

观看屏幕分享

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

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

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

常见问题

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

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

实时屏幕分享(Mac)

最近更新时间：2021-11-08 11:39:29

腾讯云 TRTC 支持屏幕分享功能，Mac 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（**substream**）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeSub` 来启用该模式。观看该路画面需要使用专门的 `startRemoteSubStreamView` 接口。

- **主路分享**
  在 TRTC 中，我们一般把摄像头走的通道叫做“主路（**bigstream**）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeBig` 来启用该模式。

支持的平台

iOS	Android	Mac OS	Windows	Electron	Chrome 浏览器
✓	✓	✓	✓	✓	✓

获取分享目标

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

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

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

字段	类型	含义
type	TRTCScreenCaptureSourceType	采集源类型：指定类型为窗口或屏幕

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

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

选择分享目标

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

- **整个屏幕分享**：
  即把整个屏幕窗口分享出去，支持多显示器分屏的情况。需要指定一个 type 为 `TRTCScreenCaptureSourceTypeScreen` 的 screenSource 参数，并将 rect 设为 { 0, 0, 0, 0 }。

- **指定区域分享**：
  即把屏幕的某个区域分享出去，需要用户圈定区域的位置坐标。需要指定一个 type 为 `TRTCScreenCaptureSourceTypeScreen` 的 screenSource 参数，并将 captureRect 设为非 NULL，例如 { 100, 100, 300, 300 }。

- **指定窗口分享**：
  即把目标窗口的内容分享出去，需要用户选择要分享的是哪一个窗口。需要指定一个 type 为 `TRTCScreenCaptureSourceTypeWindow` 的 screenSource 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

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

开始屏幕分享

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

```
/**
* 7.6 【屏幕共享】启动屏幕分享
* @param view 渲染控件所在的父控件
*/
- (void)startScreenCapture:(NSView *)view;
/**
* 7.7 【屏幕共享】停止屏幕采集
* @return 0：成功 <0:失败
*/
- (int)stopScreenCapture;
/**
* 7.8 【屏幕共享】暂停屏幕分享
* @return 0：成功 <0:失败
*/
- (int)pauseScreenCapture;
/**
* 7.9 【屏幕共享】恢复屏幕分享
*
* @return 0：成功 <0:失败
*/
- (int)resumeScreenCapture;
```

设定画面质量

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

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

观看屏幕分享

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

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

```
//示例代码：观看屏幕分享的画面
- (void)onUserSubStreamAvailable:(NSString *)userId available:(BOOL)available {
if (available) {
[self.trtcCloud startRemoteSubStreamView:userId view:self.capturePreviewWindow.contentView];
} else {
[self.trtcCloud stopRemoteSubStreamView:userId];
}
}
```

常见问题

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

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

实时屏幕分享(Web)

最近更新时间：2022-04-15 15:38:57

TRTC Web SDK 屏幕分享支持度请查看 浏览器支持情况。同时 SDK 提供 TRTC.isScreenShareSupported 接口判断当前浏览器是否支持屏幕分享。
本文通过以下不同的场景介绍实现过程。

注意：

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

创建和发布屏幕分享流

注意：
请严格按照以下顺序执行创建和发布屏幕分享流的代码。

步骤1：创建屏幕分享流

屏幕分享流包含视频流和音频流。其中音频流分为麦克风音频或者系统音频。

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

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

注意：

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

步骤2：初始化屏幕分享流

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

注意：
由于 Safari 的限制，屏幕分享流的初始化操作，必须在点击事件的回调中完成，该问题详细介绍请参见本文常见问题。

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

步骤3：创建负责进行屏幕分享的客户端对象

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

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

步骤4：发布屏幕分享流

通过第一步创建的客户端对象进行发布。发布成功后，远端就能收到屏幕分享流。

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

完整代码

```
// 通常情况下，建议给 userId 加上前缀 `share_`，用来标识这是用于屏幕分享的客户端对象。
const userId = 'share_userId';
const roomId = 'roomId';
// 仅采集屏幕视频流
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// or 采集麦克风音频及屏幕视频流
// const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// or 采集系统音频及屏幕视频流
// const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
try {
await shareStream.initialize();
} catch (error) {
// 当屏幕分享流初始化失败时，提醒用户并停止后续进房发布流程
switch (error.name) {
case 'NotReadableError':
// 提醒用户确保系统允许当前浏览器获取屏幕内容
return;
case 'NotAllowedError':
if (error.message.includes('Permission denied by system')) {
```

```
// 提醒用户确保系统允许当前浏览器获取屏幕内容
} else {
// 用户拒绝/取消屏幕分享
}
return;
default:
// 初始化屏幕分享流时遇到了未知错误，提醒用户重试
return;
}
}
const shareClient = TRTC.createClient({
mode: 'rtc',
sdkAppId,
userId, // 例如：‘share_teacher’
userSig
});
// 客户端对象进入房间
try {
await shareClient.join({ roomId });
// ShareClient join room success
} catch (error) {
// ShareClient join room failed
}
try {
await shareClient.publish(shareStream);
} catch (error) {
// ShareClient failed to publish local stream
```

屏幕分享参数配置

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

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

或者使用自定义分辨率、帧率和码率：

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

```
});
await shareStream.initialize();
```

屏幕分享属性推荐列表：

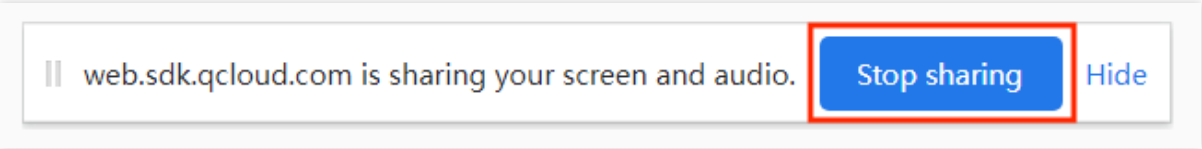
profile	分辨率（宽 x 高）	帧率（fps）	码率 (kbps)
480p	640 x 480	5	900
480p_2	640 x 480	30	1000
720p	1280 x 720	5	1200
720p_2	1280 x 720	30	3000
1080p	1920 x 1080	5	1600
1080p_2	1920 x 1080	30	4000

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

停止屏幕分享

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

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

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




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

同时发布摄像头视频和屏幕分享

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

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

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

示例代码：

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

// 负责本地音视频流发布的 client 需要取消订阅 shareClient 发布的流。
client.on('stream-added', event => {
const remoteStream = event.stream;
```

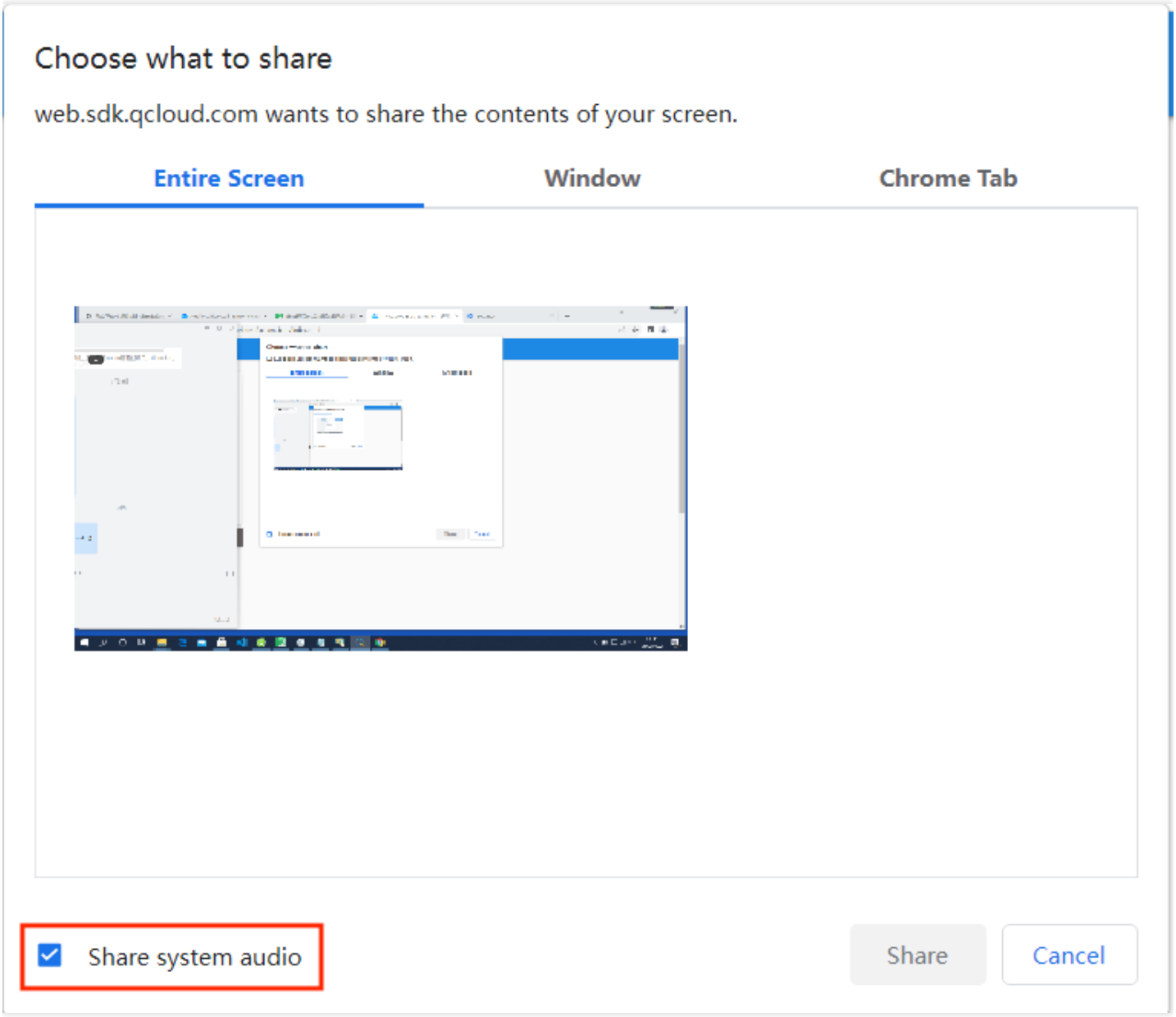
```
const remoteUserId = remoteStream.getUserId();
if (remoteUserId === `share_${userId}`) {
// 取消订阅自己的屏幕分享流
client.unsubscribe(remoteStream);
} else {
// 订阅其他一般远端流
client.subscribe(remoteStream);
}
});

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

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

// ... 省略初始化和发布相关代码，按需发布流即可。
```

屏幕分享采集系统声音

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

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

在弹出的对话框中勾选 `分享音频`，发布的 stream 将会带上系统声音。



常见问题

1. **Safari 屏幕分享出现报错** `getDisplayMedia must be called from a user gesture handler`
   因为 Safari 限制了 `getDisplayMedia` 屏幕采集的接口，必须在用户点击事件的回调函数执行的 1 秒内才可以调用。请参见 webkit issue。

```
// good
async function onClick() {
// 建议在 onClick 执行时，先执行采集逻辑
const screenStream = TRTC.createStream({ screen: true });
await screenStream.initialize();
```

```
await client.join({ roomId: 123123 });
}
// bad
async function onClick() {
await client.join({ roomId: 123123 });
// 进房可能耗时超过 1s，可能会采集失败
const screenStream = TRTC.createStream({ screen: true });
await screenStream.initialize();
}
```

2. WebRTC 屏幕分享已知问题及规避方案

实时屏幕分享(Flutter)

最近更新时间：2021-11-10 15:30:25

基于 Android 平台

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

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

启动屏幕分享

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

添加 Activity

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

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

设定视频编码参数

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

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps

参数项	参数名称	常规推荐值	文字教学场景
分辨率自适应	enableAdjustRes	NO	NO

说明：

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

弹出悬浮窗以避免被强杀

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

方案：弹出普通的悬浮窗

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

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

基于 iOS 平台

- **应用内分享**
  即只能分享当前 App 的画面，该特性需要 iOS 13 及以上版本的操作系统才能支持。由于无法分享当前 App 之

外的屏幕内容，因此适用于对隐私保护要求高的场景。

- **跨应用分享**
  基于苹果的 Replaykit 方案，能够分享整个系统的屏幕内容，但需要当前 App 额外提供一个 Extension 扩展组件，因此对接步骤也相对应用内分享要多一点。

方案1：iOS 平台应用内分享

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

我们推荐的用于 iOS 屏幕分享的编码参数是：

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

说明：

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

方案2：iOS 平台跨应用分享

示例代码

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

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

```
├──── Runner // TRTC 精简化 Demo
├──── TXLiteAVSDK_ReplayKitExt.framework //TXLiteAVSDK_ReplayKitExt SDK
```

您可以通过 README 中的指引跑通该示例 Demo。

对接步骤

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

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

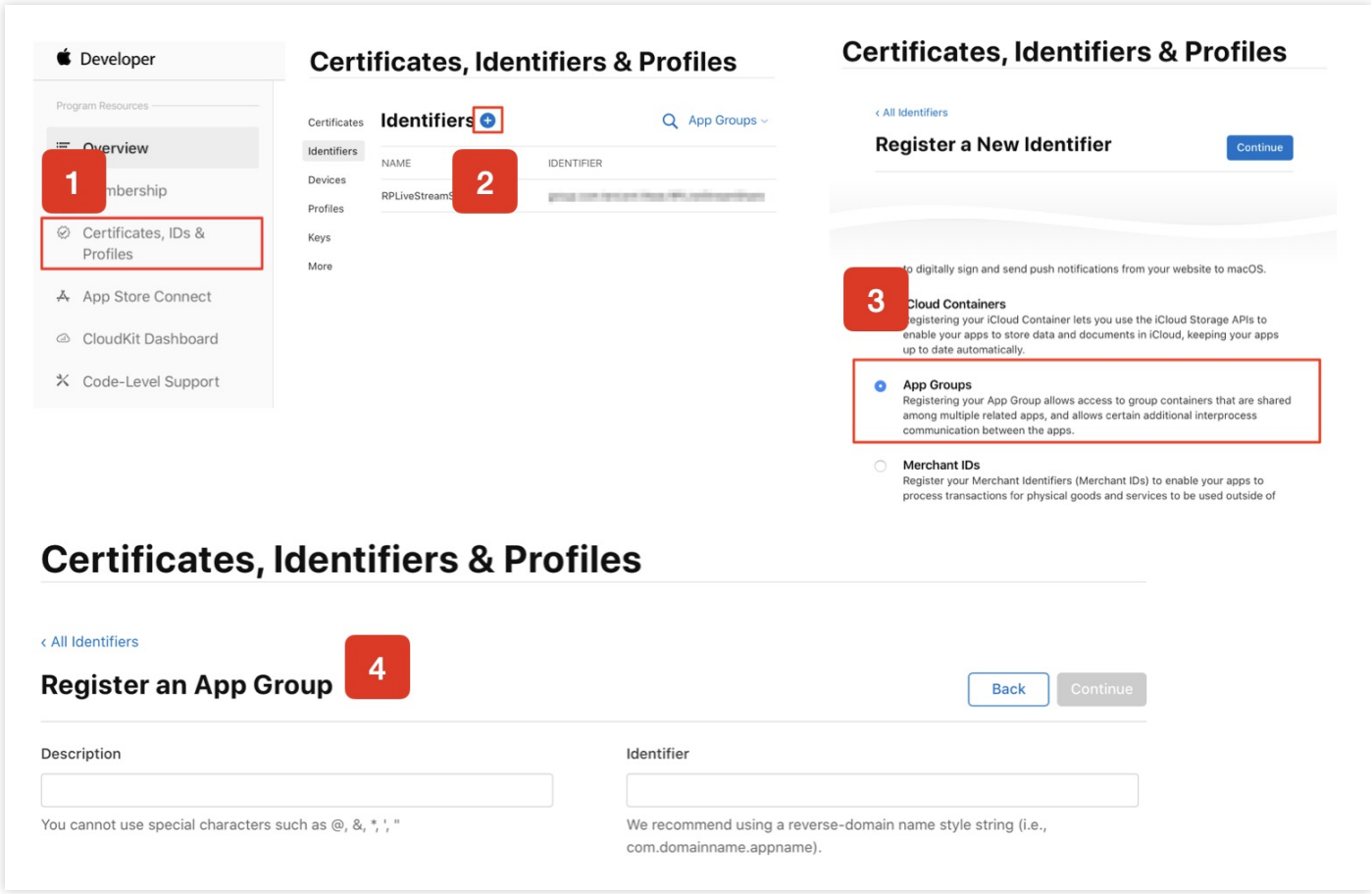
```
# 引入 trtc sdk和replay_kit_launcher
dependencies:
tencent_trtc_cloud: ^0.2.1
replay_kit_launcher: ^0.2.0+1
```

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

步骤1：创建 App Group

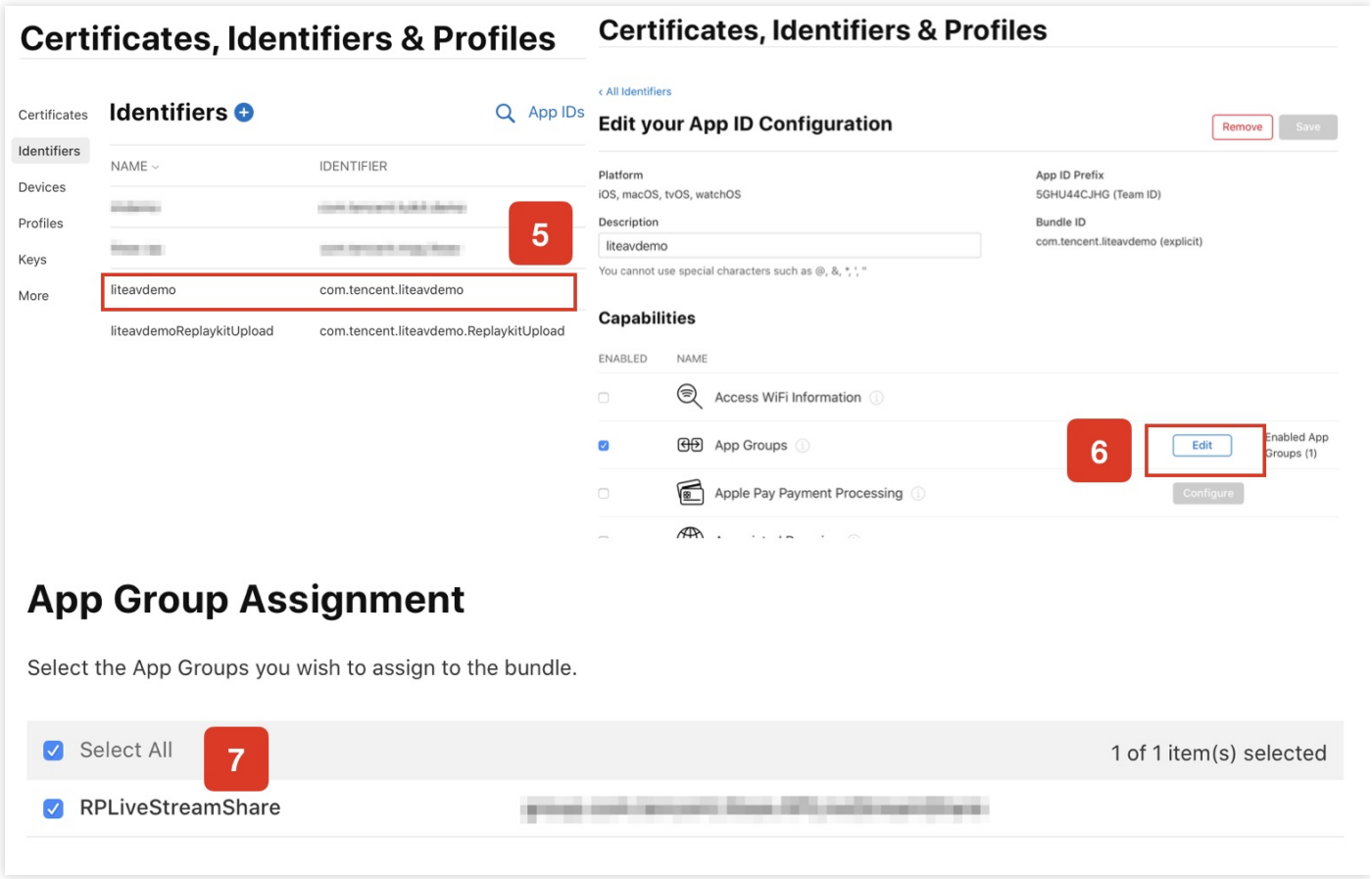
使用您的帐号登录 **https://developer.apple.com/**，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

1. 单击【Certificates, IDs & Profiles】。
2. 在右侧的界面中单击加号。
3. 选择【App Groups】，单击【Continue】。
4. 在弹出的表单中填写 Description 和 Identifier，其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击【Continue】。



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

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

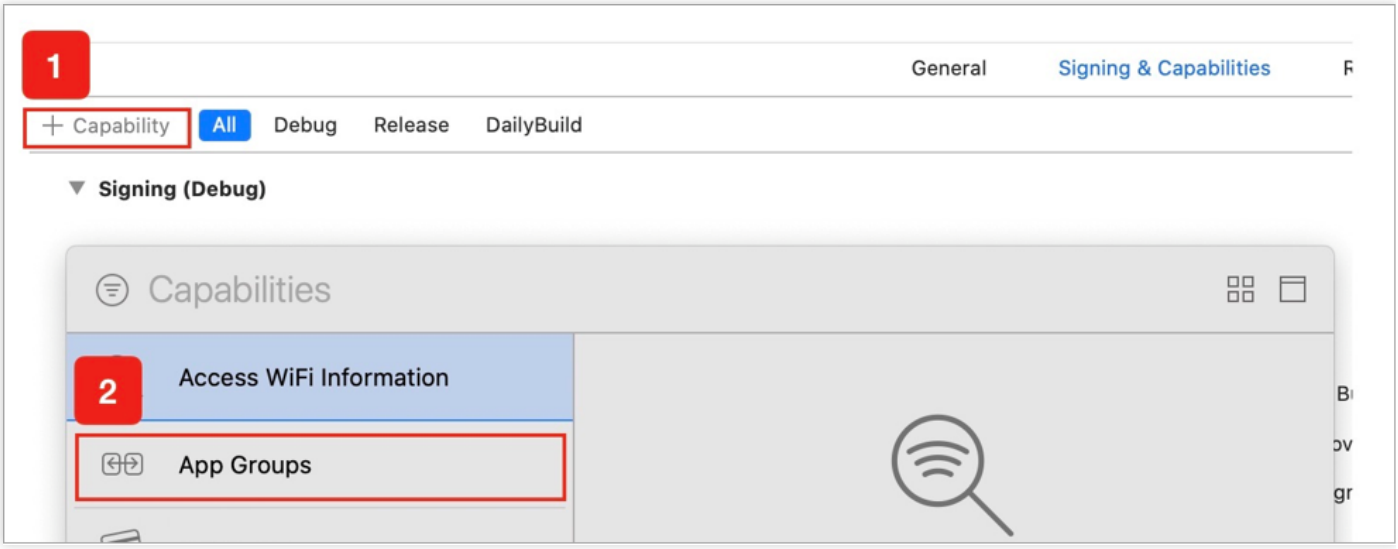


8. 重新下载 Provisioning Profile 并配置到 XCode 中。

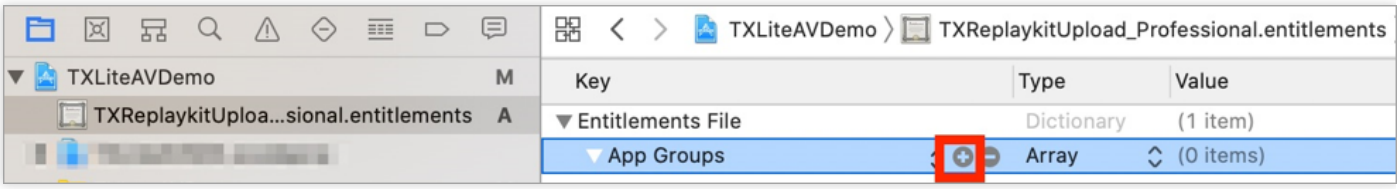
步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击【File】>【New】 >【Target...】，选择【Broadcast Upload Extension】。
2. 在弹出的对话框中填写相关信息，**不用**勾选"【Include UI Extension】，单击【Finish】完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。

4. 选中新增加的 Target，依次单击【+ Capability】，双击【App Groups】，如下图：



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



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

```
import ReplayKit
import TXLiteAVSDK_ReplayKitExt


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

class SampleHandler: RPBroadcastSampleHandler, TXReplayKitExtDelegate {

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

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

override func broadcastPaused() {
```

```
// User has requested to pause the broadcast. Samples will stop being delivered.
}

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

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

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

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

override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType: RPSam
pleBufferType) {
switch sampleBufferType {
case RPSampleBufferType.video:
// Handle video sample buffer
TXReplayKitExt.sharedInstance() .sendVideoSampleBuffer(sampleBuffer)
break
case RPSampleBufferType.audioApp:
// Handle audio sample buffer for app audio
break
case RPSampleBufferType.audioMic:
// Handle audio sample buffer for mic audio
break
@unknown default:
// Handle other sample buffer types
```

```
fatalError("Unknown type of sample buffer")
}
}
}
```

步骤3：对接主 App 端的接收逻辑

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

1. 确保 TRTCCloud 已经关闭了摄像头采集，如果尚未关闭，请调用 stopLocalPreview 关闭摄像头采集。
2. 调用 startScreenCapture 方法，并传入 步骤1 中设置的 AppGroup，让 SDK 进入“等待”状态。
3. 等待用户触发屏幕分享。如果不实现 步骤4 中的“触发按钮”，屏幕分享就需要用户在 iOS 系统的控制中心，通过长按录屏按钮来触发。
4. 通过调用 stopScreenCapture 接口可以随时中止屏幕分享。

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

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


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

步骤4：增加屏幕分享的触发按钮（可选）

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

1. 将 `replay_kit_launcher` 插件加入到您的工程中。

2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 `ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);` 函数，就可以唤起屏幕分享功能了。

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

观看屏幕分享

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

常见问题

一个房间里可以同时有多路屏幕分享吗？

目前一个 TRTC 音视频房间只能有一路屏幕分享。

跑通直播模式
跑通直播模式(iOS&Mac)

最近更新时间：2022-03-09 18:04:05

适用场景

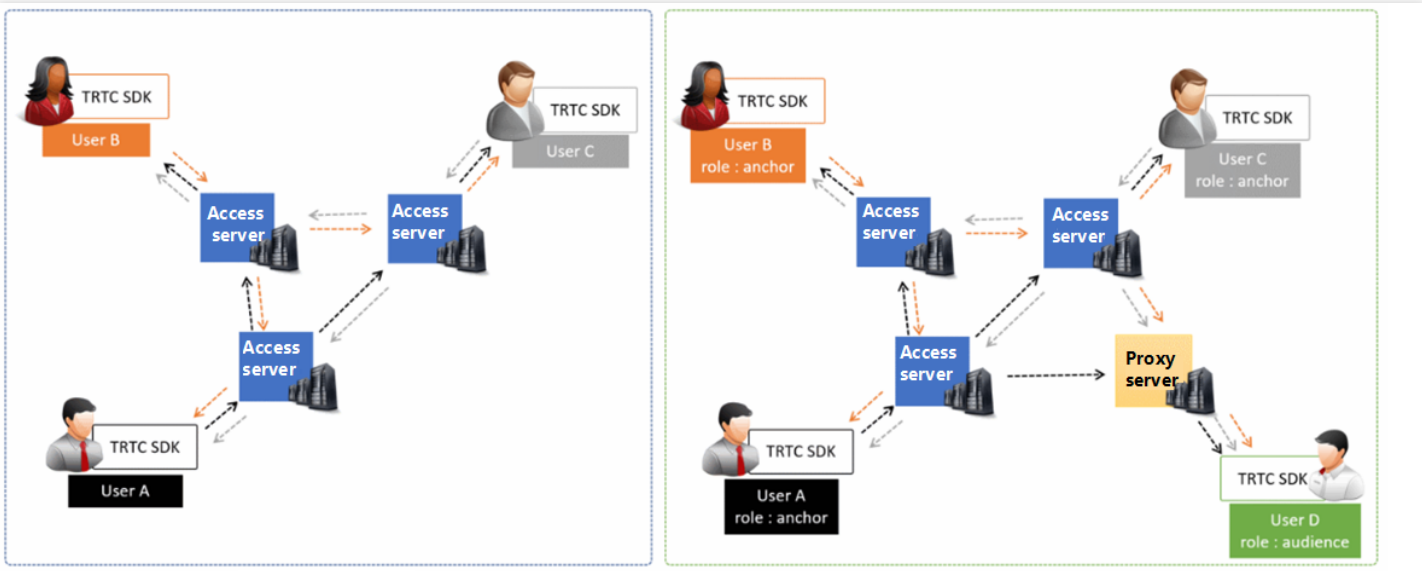
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为 通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为直播模式。
直播模式下的 TRTC，支持单个房间最多10万人同时在线，具备小于300ms的连麦延迟和小于1000ms的观看延迟，以及平滑上下麦切换技术。适用低延时互动直播、十万人互动课堂、视频相亲、在线教育、远程培训、超大型会议等应用场景。

原理解析

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

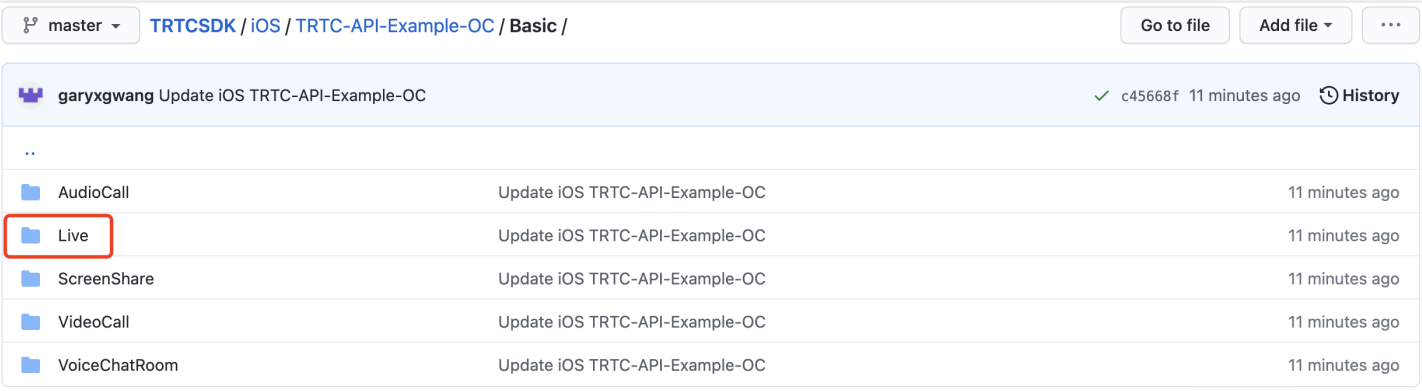
- **接口机**
  该类节点都采用最优质的线路和高性能的机器，善于处理端到端的低延时连麦通话。
- **代理机**
  该类节点都采用普通的线路和性能一般的机器，善于处理高并发的拉流观看需求。

在直播模式下，TRTC 引入了角色的概念，用户被分成“主播”和“观众”两种角色，“主播”会被分配到接口机上，“观众”则被分配在代理机，同一个房间的观众人数上限为10万人。
如果“观众”要上麦，需要先切换角色（switchRole）为“主播”才能发言。切换角色的过程也伴随着用户从代理机到接口机的迁移，TRTC 特有的低延时观看技术和平滑上下麦切换技术，可以让整个切换时间变得非常短暂。



示例代码

您可以登录 Github 获取本文档相关的示例代码。



说明：

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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 **TRTC SDK** 集成到项目中。

方式一：使用 CocoaPods 集成

1. 安装 **CocoaPods**，具体操作请参见 CocoaPods 官网安装说明。

2. 打开您当前项目根目录下的 `Podfile` 文件，添加以下内容：

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

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

3. 执行以下命令安装 **TRTC SDK**。

```
pod install
```

安装成功后当前项目根目录下会生成一个 **xcworkspace** 文件。
4. 打开新生成的 **xcworkspace** 文件即可。

方式二：下载 ZIP 包手动集成

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

步骤2：添加媒体设备权限

在 `Info.plist` 文件中添加摄像头和麦克风的申请权限：

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

步骤3：初始化 SDK 实例并监听事件回调

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

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

2. 设置 delegate 属性注册事件回调，并监听相关事件和错误通知。

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

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1SsII...
roomId	数字	数字类型的房间号。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。	29834

注意：

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

步骤5：主播端开启摄像头预览和麦克风采音

1. 主播端调用 startLocalPreview() 可以开启本地的摄像头预览，SDK 会向系统请求摄像头使用权限。

2. 主播端调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：

- Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
- Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

3. 主播端调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。
4. 主播端调用 startLocalAudio() 开启麦克风，SDK 会向系统请求麦克风使用权限。

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

//设置本地视频编码参数
TRTCVideoEncParam *encParams = [TRTCVideoEncParam new];
encParams.videoResolution = TRTCVideoResolution_640_360;
encParams.videoBitrate = 550;
encParams.videoFps = 15;

[self.trtcCloud setVideoEncoderParam:encParams];
```

步骤6：主播端设置美颜效果

1. 主播端调用 getBeautyManager() 可以获取美颜设置接口 TXBeautyManager。
2. 主播端调用 setBeautyStyle() 可以设置美颜风格：

- Smooth：光滑，效果比较明显，类似网红风格。
- Nature：自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然。
- Pitu ：仅 企业版 才支持。

3. 主播端调用 setBeautyLevel() 可以设置磨皮的级别，一般设置为5即可。
4. 主播端调用 setWhitenessLevel() 可以设置美白级别，一般设置为5即可。
5. 由于 iPhone 的摄像头调色默认偏黄，建议调用 setFilter() 为主播增加美白特效，美白特效所对应的滤镜文件的下载地址：滤镜文件。

步骤7：主播端创建房间并开始推流

1. 主播端设置 TRTCParams 中的字段 `role` 为 ** `TRTCRoleType.anchor` **，表示当前用户的角色为主播。
2. 主播端调用 enterRoom() 即可创建 TRTCParams 参数字段 `roomId` 的值为房间号的音视频房间，并指定 ** `appScene` **参数：

- TRTCAppScene.LIVE：视频互动直播模式，本文以该模式为例。

- TRTCAppScene.voiceChatRoom：语音互动直播模式。

3. 房间创建成功后，主播端开始音视频数据的编码和传输流程。同时，SDK 会回调 onEnterRoom(result) 事件，参数 `result` 大于0时表示进房成功，具体数值为加入房间所消耗的时间，单位为毫秒（ms）；当 `result` 小于0时表示进房失败，具体数值为进房失败的错误码。

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

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

步骤8：观众端进入房间观看直播

1. 观众端设置 TRTCParams 中的字段 `role` 为** `TRTCRoleType.audience` **，表示当前用户的角色为观众。
2. 观众端调用 enterRoom() 即可进入 TRTCParams 参数中 `roomId` 代指的音视频房间，并指定** `appScene` ** 参数：

- TRTCAppScene.LIVE：视频互动直播模式，本文以该模式为例。
- TRTCAppScene.voiceChatRoom：语音互动直播模式。

3. 观看主播的画面：

- 如果观众端事先知道主播的 userId，直接在进房成功后使用主播 `userId` 调用 startRemoteView(userId, view: view) 即可显示主播的画面。
- 如果观众端不知道主播的 userId，观众端在进房成功后会收到 onUserVideoAvailable() 事件通知，使用回调中获取的主播 `userId` 调用 startRemoteView(userId, view: view) 便可显示主播的画面。

步骤9：观众跟主播连麦

1. 观众端调用 switch(TRTCRoleType.TRTCRoleAnchor) 将角色切换为主播（TRTCRoleType.TRTCRoleAnchor）。
2. 观众端调用 startLocalPreview() 可以开启本地的画面。
3. 观众端调用 startLocalAudio() 开启麦克风采音。

```
//示例代码：观众上麦
[self.trtcCloud switchRole:TRTCRoleAnchor];
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];

//示例代码：观众下麦
[self.trtcCloud switchRole:TRTCRoleAudience];
[self.trtcCloud stopLocalAudio];
[self.trtcCloud stopLocalPreview];
```

步骤10：主播间进行跨房连麦 PK

TRTC 中两个不同音视频房间中的主播，可以在不退出原来的直播间的场景下，通过“跨房通话”功能拉通连麦通话功能进行“跨房连麦 PK”。

1. 主播 A 调用 connectOtherRoom() 接口，接口参数目前采用 JSON 格式，需要将主播 B 的 `roomId` 和 `userId` 拼装成格式为 `{”roomId”: ”978”,”userId”: ”userB”}` 的参数传递给接口函数。
2. 跨房成功后，主播 A 会收到 onConnectOtherRoom() 事件回调。同时，两个直播房间里的所有用户均会收到 onUserVideoAvailable() 和 onUserAudioAvailable() 事件通知。
   例如，当房间“001”中的主播 A 通过 `connectOtherRoom()` 与房间“002”中的主播 B 拉通跨房通话后， 房间“001”中的用户会收到主播 B 的 `onUserVideoAvailable(B, available: true)` 回调和 `onUserAudioAvailable(B, available: true)` 回调。 房间“002”中的用户会收到主播 A 的 `onUserVideoAvailable(A, available: true)` 回调和 `onUserAudioAvailable(A, available: true)` 回调。
3. 两个房间里的用户通过调用 startRemoteView(userId, view: view) 即可显示另一房间里主播的画面，声音会自动播放。

```
//示例代码：跨房连麦 PK
NSMutableDictionary * jsonDict = [[NSMutableDictionary alloc] init];
[jsonDict setObject:@([_otherRoomIdTextField.text intValue]) forKey:@”roomId”];
[jsonDict setObject:_otherUserIdTextField.text forKey:@”userId”];
NSData* jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:NSJSONWritingPrettyPr
inted error:nil];
NSString* jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
[self.trtcCloud connectOtherRoom:jsonString];
```

步骤11：退出当前房间

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

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

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

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

跑通直播模式(Android)

最近更新时间：2022-03-09 16:56:52

适用场景

TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为 通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为直播模式。
直播模式下的 TRTC，支持单个房间最多10万人同时在线，具备小于300ms的连麦延迟和小于1000ms的观看延迟，以及平滑上下麦切换技术。适用低延时互动直播、十万人互动课堂、视频相亲、在线教育、远程培训、超大型会议等应用场景。

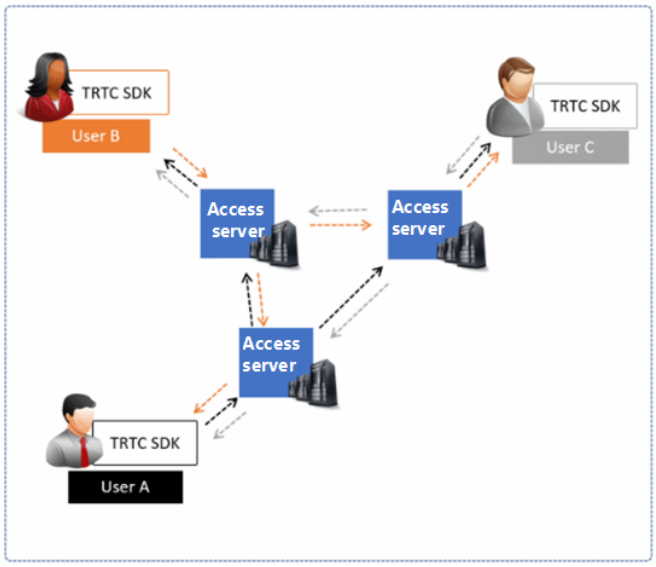
原理解析

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

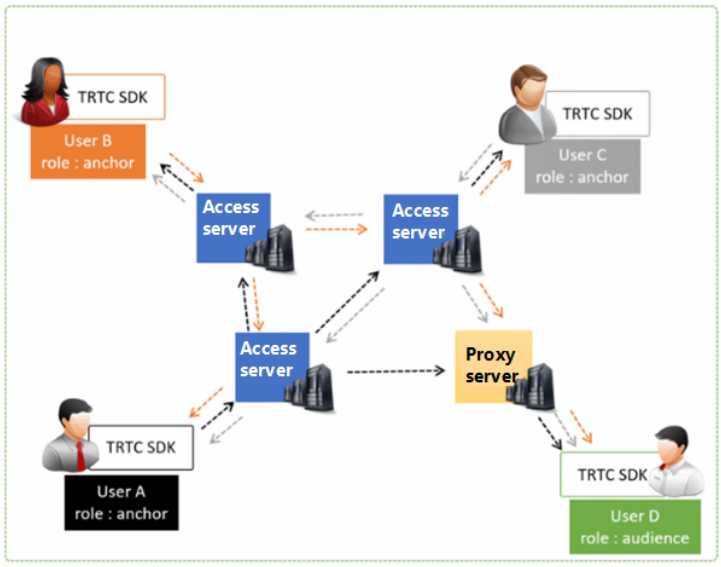
- **接口机**
  该类节点都采用最优质的线路和高性能的机器，善于处理端到端的低延时连麦通话。
- **代理机**
  该类节点都采用普通的线路和性能一般的机器，善于处理高并发的拉流观看需求。

在直播模式下，TRTC 引入了角色的概念，用户被分成“主播”和“观众”两种角色，“主播”会被分配到接口机上，“观众”则被分配在代理机，同一个房间的观众人数上限为10万人。
如果“观众”要上麦，需要先切换角色（switchRole）为“主播”才能发言。切换角色的过程也伴随着用户从代理机到

接口机的迁移，TRTC 特有的低延时观看技术和平滑上下麦切换技术，可以让整个切换时间变得非常短暂。



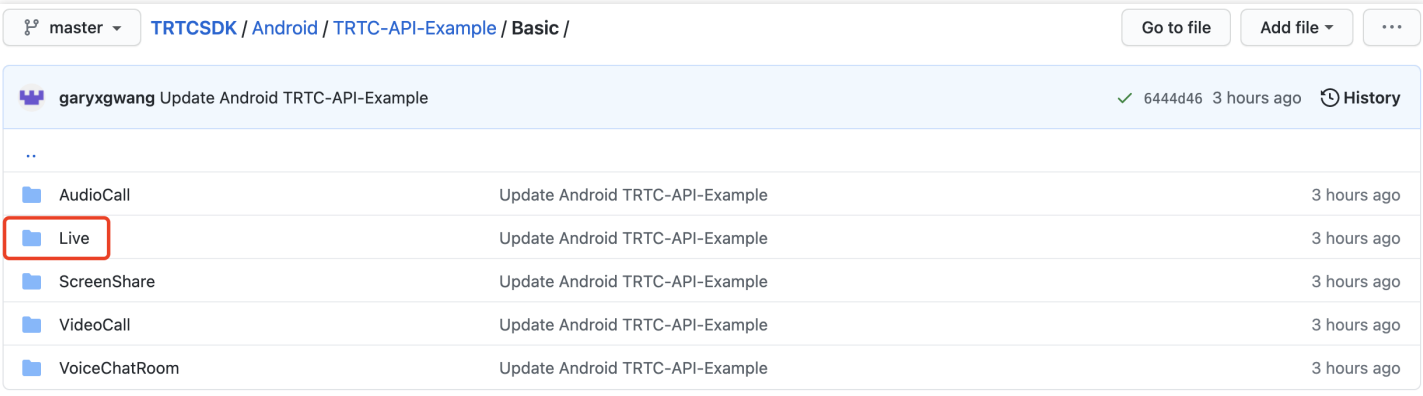
Call mode (no role differentiation)



Live streaming mode (two roles: anchor and viewer)

示例代码

您可以登录 Github 获取本文档相关的示例代码。



说明：

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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 **TRTC SDK** 集成到项目中。

方式一：自动加载（aar）

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

1. 在 dependencies 中添加 TRTCSDK 的依赖。

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

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

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

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

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

方式二：下载 ZIP 包手动集成

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

步骤2：配置 App 权限

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

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

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

步骤3：初始化 SDK 实例并监听事件回调

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

```
// 创建 trtcCloud 实例
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener());
```

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

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

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123

参数名称	字段类型	补充说明	填写示例
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig。	eJyrVareCeYrSy1SsII...
roomId	数字	数字类型的房间号。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。	29834

注意：

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

步骤5：主播端开启摄像头预览和麦克风采音

1. 主播端调用 startLocalPreview() 可以开启本地的摄像头预览，SDK 会向系统请求摄像头使用权限。
2. 主播端调用 setLocalViewFillMode() 可以设定本地视频画面的显示模式：

- Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
- Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

3. 主播端调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。
4. 主播端调用 startLocalAudio() 开启麦克风，SDK 会向系统请求麦克风使用权限。

```
//示例代码：发布本地的音视频流
mTRTCCloud.setLocalViewFillMode(TRTC_VIDEO_RENDER_MODE_FIT);
mTRTCCloud.startLocalPreview(mIsFrontCamera, localView);
//设置本地视频编码参数
TRTCCloudDef.TRTCVideoEncParam encParam = new TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_960_540;
encParam.videoFps = 15;
encParam.videoBitrate = 1200;
encParam.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT;
mTRTCCloud.setVideoEncoderParam(encParam);
mTRTCCloud.startLocalAudio();
```

步骤6：主播端设置美颜效果

1. 主播端调用 getBeautyManager() 可以获取美颜设置接口 TXBeautyManager。
2. 主播端调用 setBeautyStyle() 可以设置美颜风格：

- Smooth：光滑，效果比较明显，类似网红风格。
- Nature：自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然。
- Pitu ：仅 企业版 才支持。

3. 主播端调用 setBeautyLevel() 可以设置磨皮的级别，一般设置为5即可。
4. 主播端调用 setWhitenessLevel() 可以设置美白级别，一般设置为5即可。

步骤7：主播端创建房间并开始推流

1. 主播端设置 TRTCParams 中的字段 `role` 为 ** `TRTCCloudDef.TRTCRoleAnchor` **，表示当前用户的角色为主播。
2. 主播端调用 enterRoom() 即可创建 TRTCParams 参数中字段 `roomId` 的值为房间号的音视频房间，并指定 ** `appScene` **参数：

- TRTCCloudDef.TRTC_APP_SCENE_LIVE：视频互动直播模式，本文以该模式为例。
- TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM：语音互动直播模式。

3. 房间创建成功后，主播端开始音视频数据的编码和传输流程。同时，SDK 会回调 onEnterRoom(result) 事件，参数 `result` 大于0时表示进房成功，具体数值为加入房间所消耗的时间，单位为毫秒（ms）；当 `result` 小于0时表示进房失败，具体数值为进房失败的错误码。

```
public void enterRoom() {
TRTCCloudDef.TRTCParams trtcParams = new TRTCCloudDef.TRTCParams();
trtcParams.sdkAppId = sdkappid;
trtcParams.userId = userid;
trtcParams.roomId = 908;
trtcParams.userSig = usersig;
mTRTCCloud.enterRoom(trtcParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
}
@Override
public void onEnterRoom(long result) {
if (result > 0) {
toastTip("进房成功，总计耗时[¥(result)]ms")
} else {
toastTip("进房失败，错误码[¥(result)]")
}
}
```

步骤8：观众端进入房间观看直播

1. 观众端设置 TRTCParams 中的字段 `role` 为** `TRTCCloudDef.TRTCRoleAudience` **，表示当前用户的角色为观众。
2. 观众端调用 enterRoom() 即可进入 TRTCParams 参数中 `roomId` 代指的音视频房间，并指定** `appScene` **参数：

- TRTCCloudDef.TRTC_APP_SCENE_LIVE：视频互动直播模式，本文以该模式为例。
- TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM：语音互动直播模式。

3. 观看主播的画面：

- 如果观众端事先知道主播的 userId，直接在进房成功后使用主播 `userId` 调用 startRemoteView(userId, view) 即可显示主播的画面。
- 如果观众端不知道主播的 userId，观众端在进房成功后会收到 onUserVideoAvailable() 事件通知，使用回调中获取的主播 `userId` 调用 startRemoteView(userId, view) 便可显示主播的画面。

步骤9：观众跟主播连麦

1. 观众端调用 switchRole(TRTCCloudDef.TRTCRoleAnchor) 将角色切换为主播（TRTCCloudDef.TRTCRoleAnchor）。
2. 观众端调用 startLocalPreview() 可以开启本地的画面。
3. 观众端调用 startLocalAudio() 开启麦克风采音。

```
//示例代码：观众上麦
mTrtcCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);
mTrtcCloud.startLocalAudio();
mTrtcCloud.startLocalPreview(mIsFrontCamera, localView);
//示例代码：观众下麦
mTrtcCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
mTrtcCloud.stopLocalAudio();
mTrtcCloud.stopLocalPreview();
```

步骤10：主播间进行跨房连麦 PK

TRTC 中两个不同音视频房间中的主播，可以在不退出原来的直播间的场景下，通过“跨房通话”功能拉通连麦通话功能进行“跨房连麦 PK”。

1. 主播 A 调用 connectOtherRoom() 接口，目前接口参数采用 JSON 格式，需要将主播 B 的 `roomId` 和 `userId` 拼装成格式为 `{”roomId”: ”978”,”userId”: ”userB”}` 的参数传递给接口函数。
2. 跨房成功后，主播 A 会收到 onConnectOtherRoom() 事件回调。同时，两个直播房间里的所有用户均会收到 onUserVideoAvailable() 和 onUserAudioAvailable() 事件通知。

例如，当房间“001”中的主播 A 通过 `connectOtherRoom()` 与房间“002”中的主播 B 拉通跨房通话后， 房间“001”中的用户会收到主播 B 的 `onUserVideoAvailable(B, true)` 回调和 `onUserAudioAvailable(B, true)` 回调。 房间“002”中的用户会收到主播 A 的 `onUserVideoAvailable(A, true)` 回调和 `onUserAudioAvailable(A, true)` 回调。

3. 两个房间里的用户通过调用 startRemoteView(userId, view) 即可显示另一房间里主播的画面，声音会自动播放。

```
//示例代码：跨房连麦 PK
mTRTCCloud.ConnectOtherRoom(String.format("{¥"roomId¥":%s,¥"userId¥":¥"%s¥"}", roomId, usernam
e));
```

步骤11：退出当前房间

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

```
// 调用退房后请等待 onExitRoom 事件回调
mTRTCCloud.exitRoom()
@Override
public void onExitRoom(int reason) {
Log.i(TAG, "onExitRoom: reason = " + reason);
}
```

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

跑通直播模式(Windows)

最近更新时间：2022-01-10 10:51:24

文档导读

本文主要介绍如何基于 TRTC SDK 实现一个既支持视频连麦，又支持上万人高并发观看的在线直播功能。本文仅罗列最常用的几个接口，如果您希望了解更多的接口函数，请参见 API 文档。

示例代码

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

在线直播

1. 初始化 SDK

使用 TRTC SDK 的第一步，是先获取 `TRTCCloud` 的单例对象，并注册监听 SDK 事件的回调。

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

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

- C++版
- C#版

```
// TRTCMainViewController.h

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

virtual void onError(TXLiteAVError errCode, const char* errMsg, void* arg);
virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void* arg);
virtual void onEnterRoom(uint64_t elapsed);
virtual void onExitRoom(int reason);
virtual void onRemoteUserEnterRoom(const char* userId);
virtual void onRemoteUserLeaveRoom(const char* userId, int reason);
virtual void onUserVideoAvailable(const char* userId, bool available);
virtual void onUserAudioAvailable(const char* userId, bool available);
...
private:
ITRTCCloud * m_pTRTCSDK = NULL ;
...
}

// TRTCMainViewController.cpp

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

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

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

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

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

2. 组装 TRTCParams

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

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

- **userId**
  您可以随意指定，由于是字符串类型，可以直接跟您现有的账号体系保持一致，但请注意，**同一个音视频房间里不应该有两个同名的 userId**。

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

- **roomId**
  房间号是数字类型，您可以随意指定，但请注意，**同一个应用里的两个音视频房间不能分配同一个 roomId**。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。

3. 主播预览摄像头画面

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

启动本地预览前，可调用 `setLocalViewFillMode` 指定视频显示模式为 `Fill` 或 `Fit` 模式。两种模式下视频尺寸都是等比缩放，区别在于：

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

- C++版

- C#版

```
void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
// 获取渲染窗口的句柄。
CWnd *pLocalVideoView = GetDlgItem(IDC_LOCAL_VIDEO_VIEW);
HWND hwnd = pLocalVideoView-&gt;GetSafeHwnd();

if(m_pTRTCSDK)
{
// 调用 SDK 接口设置渲染模式和渲染窗口。
m_pTRTCSDK-&gt;setLocalViewFillMode(TRTCVideoFillMode_Fit);
m_pTRTCSDK-&gt;startLocalPreview(hwnd);
}

}
```

4. 主播开启麦克风采集

TRTC SDK 并不会默认打开本地的麦克风采集，主播调用 `startLocalAudio` 可以开启本地的声音采集并将音视频数据广播出去， `stopLocalAudio` 则会关闭。您可以在 `startLocalPreview` 之后继续调用 `startLocalAudio` 。

> 说明：
> `startLocalAudio` 会检查麦克风使用权限，如果没有麦克风权限，SDK 会向用户申请开启。

5. 主播创建新房间开播

主播可以使用 `enterRoom` 创建一个音视频房间，参数 `TRTCParams` 中的 `roomId` 用于指定房间号，同时，我们还需要将 `role` 字段指定为 `TRTCRoleAnchor` （主播）。

`appScene` 参数指定 SDK 的应用场景，本文档中我们使用 `TRTCAppSceneLIVE` （在线直播）。

- 如果创建成功，SDK 会回调 `onEnterRoom` 接口，参数： `elapsed` 代表进入耗时，单位：ms。
- 如果创建失败，SDK 会回调 `onError` 接口，参数： `errCode` （错误码 `ERR_ROOM_ENTER_FAIL` ，错误码可参考 `TXLiteAVCode.h` ）、 `errMsg` （错误原因）、 `extraInfo` （保留参数）。

- C++版
- C#版

```
// TRTCMainViewController.cpp
```

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

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

...

void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
LOGI(L"onEnterRoom elapsed[%lld]", elapsed);

// 启动本地的视频预览，请参考下文设置视频编码参数和预览本地摄像头画面的内容
}
```

6. 主播开关隐私模式

直播过程中，主播可能出于隐私目的希望屏蔽本地的音视频数据，可以调用 `muteLocalVideo` 屏蔽本地的视频采集，调用 `muteLocalAudio` 屏蔽本地的音频采集。

7. 观众加入房间观看

观众调用 `enterRoom` 可以进入一个音视频房间，参数 TRTCParams 中的 `roomId` 用于指定房间号。`appScene` 同样填写 `TRTCAppSceneLIVE`（在线直播），但 `role` 字段需要指定为 `TRTCRoleAudience`（观众）。

- C++版
- C#版

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

如果主播在房间里，观众会通过 TRTCCloudDelegate 中的 `onUserVideoAvailable` 回调获知主播的 userid。然后观众可以调用 `startRemoteView` 方法来显示主播的视频画面。

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

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

- C++版
- C#版

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

// 设置远端用户视频的渲染模式。
m_pTRTCSDK-&gt;setRemoteViewFillMode(TRTCVideoFillMode_Fill);
// 调用 SDK 接口播放远端用户流。
m_pTRTCSDK-&gt;startRemoteView(userId,  hwnd);
} else {
m_pTRTCSDK-&gt;stopRemoteView(userId);
}
}
```

注意：

- 在 TRTCAppSceneLIVE 模式下，同一个房间中的观众（TRTCRoleAudience）人数没有限制。
- 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

8. 观众跟主播连麦

主播和观众都可以通过 TRTCCloud 提供的 `switchRole` 进行角色间的相互切换，最常见的场景是观众跟主播连麦：观众可以通过该接口切换成“小主播”，然后跟房间里原来的“大主播”进行连麦互动。

9. 退出房间

调用 `exitRoom` 方法退出房间。无论当前是否还在通话中，调用该方法会把视频通话相关的所有资源释放掉。在您调用 `exitRoom` 之后，SDK 会进入一个复杂的退房握手流程，当 SDK 回调 `onExitRoom` 方法时才算真正完成资源的释放。

- C++版
- C#版

```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;exitRoom();
}
}
....
void TRTCMainViewController::onExitRoom(int reason)
{
// 退房成功，reason 参数保留，暂未使用。

}
```

跑通直播模式(Electron)

最近更新时间：2022-01-10 10:52:48

适用场景

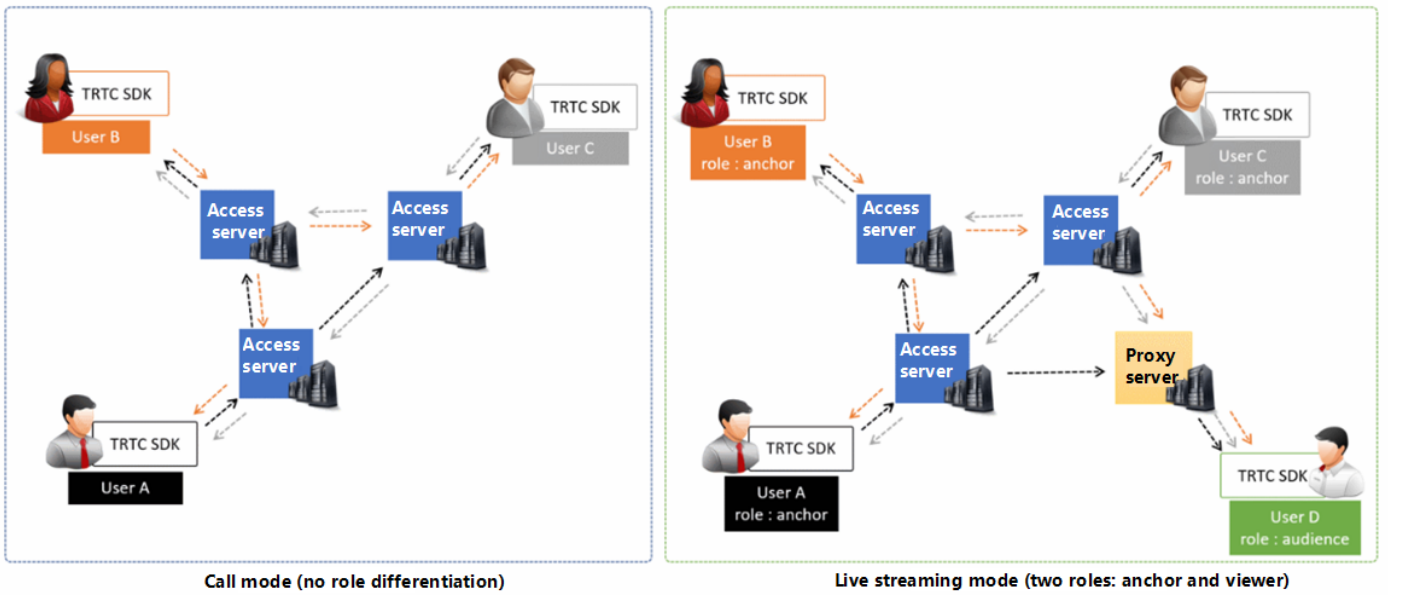
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为 通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为直播模式。
直播模式下的 TRTC，支持单个房间最多10万人同时在线，具备小于300ms的连麦延迟和小于1000ms的观看延迟，以及平滑上下麦切换技术。适用低延时互动直播、十万人互动课堂、视频相亲、在线教育、远程培训、超大型会议等应用场景。

原理解析

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

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

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



示例代码

您可以登录 Github 获取本文档相关的示例代码。

操作步骤

步骤1：尝试跑通官网 SimpleDemo

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

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

步骤2：为您的项目集成 trtc-electron-sdk

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

- 您可以在我们的官方 Demo 的基础上进行二次开发，项目的起步阶段会比较顺利。
- 您也可以执行以下指令，把 `trtc-electron-sdk` 安装到您现有的项目中：

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

步骤3：初始化 SDK 实例并监听事件回调

创建 `trtc-electron-sdk` 实例：

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

监听 `onError` 事件:

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

步骤4： 组装进房参数 TRTCParams

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

参数	类型	说明	示例

参数	类型	说明	示例
sdkAppId	数字	应用 ID，您可以在 控制台 >【应用管理】>【应用信息】中查找到。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1SsII...
roomId	数字	数字类型的房间号。如果您想使用字符串形式的房间号，请使用 TRTCParams 中的 strRoomId。	29834

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

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
param.role = TRTCRoleType.TRTCRoleAnchor; // 设置角色为"主播"
```

注意：

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

步骤5：主播端开启摄像头预览和麦克风采音

1. 主播端调用 startLocalPreview() 可以开启本地的摄像头预览，SDK 会向系统请求摄像头使用权限。
2. 主播端调用 `setLocalViewFillMode()` 可以设定本地视频画面的显示模式：
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` ：模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fit` ：模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

3. 主播端调用 setVideoEncoderParam() 接口可以设定本地视频的编码参数，该参数将决定房间里其他用户观看您的画面时所感受到的 画面质量。
4. 主播端调用 startLocalAudio() 开启麦克风，SDK 会向系统请求麦克风使用权限。

```
//示例代码：发布本地的音视频流
trtcCloud.startLocalPreview(view);
trtcCloud.startLocalAudio();
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);

//设置本地视频编码参数
let encParam = new TRTCVideoEncParam();
encParam.videoResolution = TRTCVideoResolution.TRTCVideoResolution_640_360;
encParam.resMode = TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape;
encParam.videoFps = 25;
encParam.videoBitrate = 600;
encParam.enableAdjustRes = true;
trtcCloud.setVideoEncoderParam(encParam);
```

步骤6：主播端设置美颜效果

1. 主播端可调用 setBeautyStyle(style, beauty, white, ruddiness) 来开启美颜效果
2. 参数说明：
   - style： 美颜风格，光滑或者自然，光滑风格磨皮更加明显，适合娱乐场景。
     - `TRTCBeautyStyle.TRTCBeautyStyleSmooth`：光滑，适用于美女秀场，效果比较明显。
     - `TRTCBeautyStyle.TRTCBeautyStyleNature`：自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然。
   - beauty：美颜级别，取值范围0 - 9，0表示关闭，1 - 9值越大，效果越明显。
   - white：美白级别，取值范围0 - 9，0表示关闭，1 - 9值越大，效果越明显。
   - ruddiness：红润级别，取值范围0 - 9，0表示关闭，1 - 9值越大，效果越明显，该参数 Windows 平台暂未生效。

```
// 开启美颜
trtcCloud.setBeautyStyle(TRTCBeautyStyle.TRTCBeautyStyleNature, 5, 5, 5);
```

步骤7：主播端创建房间并开始推流

1. 主播端设置 TRTCParams 中的字段 `role` 为 `TRTCRoleType.TRTCRoleAnchor`，表示当前用户的角色为主播。
2. 主播端调用 enterRoom( )即可创建 TRTCParams 参数字段 `roomId` 的值为房间号的音视频房间，并指定 `appScene` 参数：
   - `TRTCAppScene.TRTCAppSceneLIVE` ：视频互动直播，支持平滑上下麦，切换过程无需等待，主播延时小于300ms；支持十万级别观众同时播放，播放延时低至1000ms。本文以该模式为例。

- `TRTCAppScene.TRTCAppSceneVoiceChatRoom`：语音互动直播，支持平滑上下麦，切换过程无需等待，主播延时小于300ms；支持十万级别观众同时播放，播放延时低至1000ms。
- 关于 `TRTCAppScene` 的详细介绍，请参见 TRTCAppScene 。

3. 房间创建成功后，主播端开始音视频数据的编码和传输流程。同时，SDK 会回调 onEnterRoom(result) 事件，参数 `result` 大于0时表示进房成功，具体数值为加入房间所消耗的时间，单位为毫秒（ms）；当 `result` 小于0时表示进房失败，具体数值为进房失败的错误码。

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

trtcCloud.on('onEnterRoom', onEnterRoom);

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
param.role = TRTCRoleType.TRTCRoleAnchor;
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
```

步骤8：观众端进入房间观看直播

1. 观众端设置TRTCParams中的字段 `role` 为 `TRTCRoleType.TRTCRoleAudience`，表示当前用户的角色为观众。
2. 观众端调用 `enterRoom()` 即可进入 `TRTCParams` 参数中 `roomId` 代指的音视频房间，并指定 `appScene` 参数：
   - `TRTCAppScene.TRTCAppSceneLIVE`：视频互动直播。
   - `TRTCAppScene.TRTCAppSceneVoiceChatRoom`：语音互动直播。
3. 观看主播的画面：
   - 如果观众端事先知道主播的 `userId`，直接在进房成功后使用主播 `userId` 调用 startRemoteView(userId, view) 即可显示主播的画面。
   - 如果观众端不知道主播的 `userId`，观众端在进房成功后会收到 onUserVideoAvailable() 事件通知，使用回调中获取的主播 `userId` 调用 startRemoteView(userId, view) 便可显示主播的画面。

```
<div id="video-container"></div>
<script>
const videoContainer = document.querySelector('#video-container');
```

```
const roomId = 29834;
// 进房回调，当进房成功时，会触发此回调
let onEnterRoom = function(result) {
if (result > 0) {
console.log(`onEnterRoom, 进房成功，使用了 ${result} 秒`);
} else {
console.warn(`onEnterRoom: 进房失败 ${result}`);
}
};
// 当主播开启/关闭摄像头推流时，会触发此回调
let onUserVideoAvailable = function(userId, available) {
if (available === 1) {
let id = `${userId}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (!view) {
view = document.createElement('div');
view.id = id;
videoContainer.appendChild(view);
}
trtcCloud.startRemoteView(userId, view);
trtcCloud.setRemoteViewFillMode(userId, TRTCVideoFillMode.TRTCVideoFillMode_Fill);
} else {
let id = `${userId}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (view) {
videoContainer.removeChild(view);
}
}
};

trtcCloud.on('onEnterRoom', onEnterRoom);
trtcCloud.on('onUserVideoAvailable', onUserVideoAvailable);

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = roomId;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
param.role = TRTCRoleType.TRTCRoleAudience; // 设置角色为“观众”
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
</script>
```

步骤9：观众跟主播连麦

1. 观众端调用 switchRole(TRTCRoleType.TRTCRoleAnchor) 将角色切换为主播（`TRTCRoleType.TRTCRoleAnchor`）。
2. 观众端调用 startLocalPreview() 可以开启本地的画面。

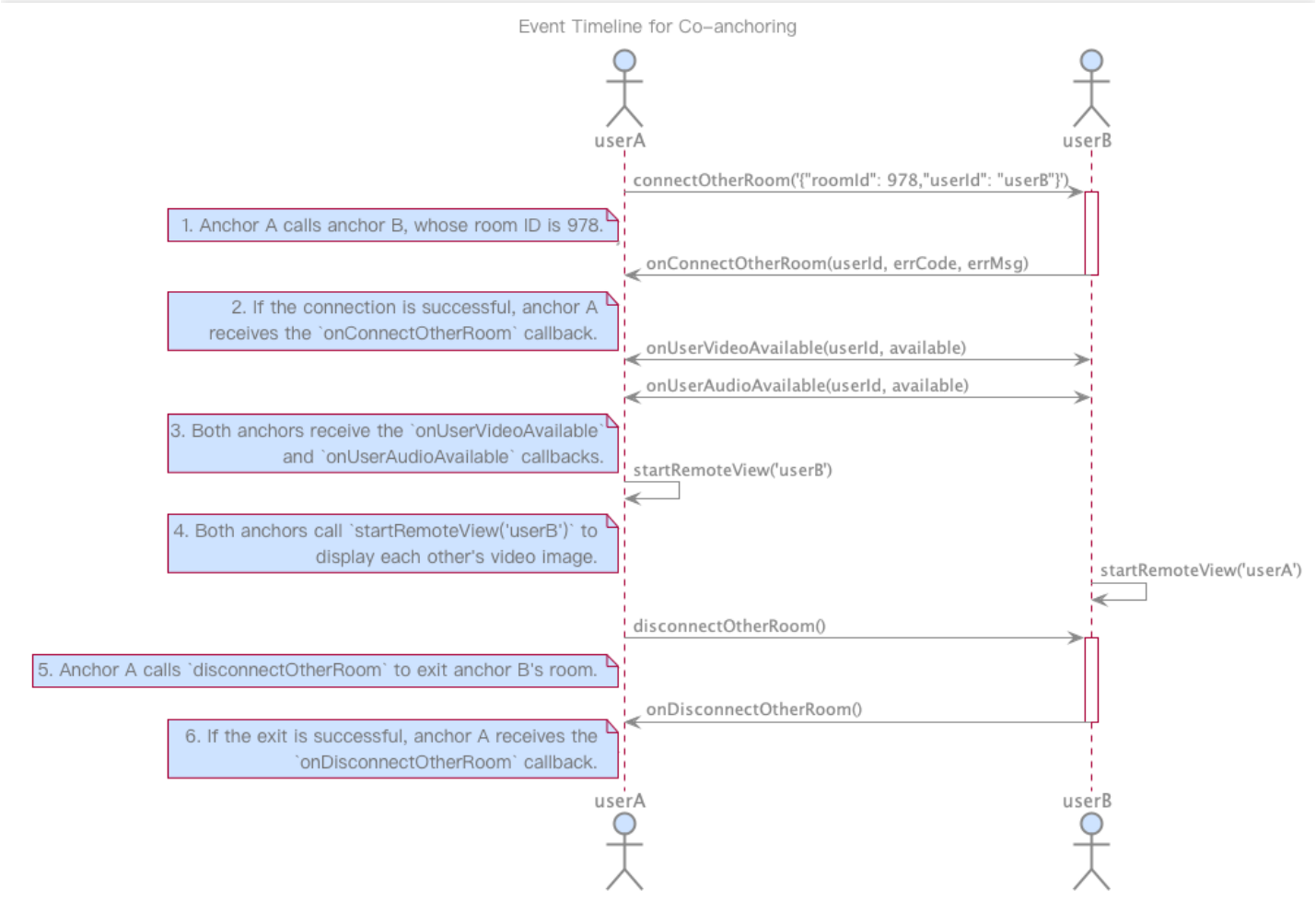
3. 观众端调用 startLocalAudio() 开启麦克风采音。

```
//示例代码：观众上麦
trtcCloud.switchRole(TRTCRoleType.TRTCRoleAnchor);
trtcCloud.startLocalAudio();
trtcCloud.startLocalPreview(frontCamera, view);

//示例代码：观众下麦
trtcCloud.switchRole(TRTCRoleType.TRTCRoleAudience);
trtcCloud.stopLocalAudio();
trtcCloud.stopLocalPreview();
```

步骤10：主播间进行跨房连麦 PK

TRTC 中两个不同音视频房间中的主播，可以在不退出原来的直播间的场景下，通过“跨房通话”功能拉通连麦通话功能进行“跨房连麦 PK”。

1. 主播 A 调用 connectOtherRoom() 接口，接口参数目前采用 JSON 格式，需要将主播 B 的 `roomId` 和 `userId` 拼装成格式为 `{”roomId”: 978,”userId”: ”userB”}` 的参数传递给接口函数。
2. 跨房成功后，主播 A 会收到 onConnectOtherRoom(userId, errCode, errMsg) 事件回调。同时，两个直播房间里的所有用户均会收到 onUserVideoAvailable() 和 onUserAudioAvailable() 事件通知。
   例如，当房间“001”中的主播 A 通过 `connectOtherRoom()` 与房间“002”中的主播 B 拉通跨房通话后， 房间“001”中的用户会收到主播 B 的 `onUserVideoAvailable(B, true)` 回调和 `onUserAudioAvailable(B, true)` 回调。 房间“002”中的用户会收到主播 A 的 `onUserVideoAvailable(A, true)` 回调和 `onUserAudioAvailable(A, true)` 回调。
3. 两个房间里的用户通过调用 startRemoteView(userId, view) 即可显示另一房间里主播的画面，声音会自动播放。




```
//示例代码：跨房连麦 PK
let onConnectOtherRoom = function(userId, errCode, errMsg) {
if(errCode === 0) {
console.log(`成功连上主播 ${userId} 的房间`);
} else {
console.warn(`连接其他主播房间失败：${errMsg}`);
}
};

const paramJson = '{"roomId": "978","userId": "userB"}';
trtcCloud.connectOtherRoom(paramJson);
trtcCloud.on('onConnectOtherRoom', onConnectOtherRoom);
```

步骤11：退出当前房间

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

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

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

跑通直播模式(Web)

最近更新时间：2022-03-10 09:48:31

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

示例

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

步骤1：创建 Client 对象

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

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

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

步骤2：以观众身份进入直播房间

调用 Client.join() 进入音视频通话房间。参数设置：

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

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

步骤3：收看直播

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

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

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

2. 在远端流订阅成功事件回调中，通过调用 [Stream.play()](https://web.sdk.qcloud.com/trtc/webrtc/doc/zh-cn/Stream.html#play) 方法在网页中播放音视频。`play` 方法接受一个 div 元素 ID 作为参数，SDK 内部会在该 div 元素下自动创建相应的音视频标签并在其上播放音视频。

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

```
// 播放远端流
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

步骤4：跟主播连麦互动

步骤4.1：切换角色

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

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

步骤4.2：连麦互动

1. 使用 TRTC.createStream() 方法创建本地音视频流。以下实例从摄像头及麦克风中采集音视频流，参数设置如下：

- `userId` ：本地流所属用户 ID
- `audio` ：是否开启音频
- `video` ：是否开启视频

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

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

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

```
console.error('初始化本地流失败 ' + error);
});
```

3. 初始化本地流成功时，播放本地流。

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

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

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

步骤5：退出直播房间

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

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

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

TRTC 云端录制说明

最近更新时间：2022-03-11 15:32:06

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。

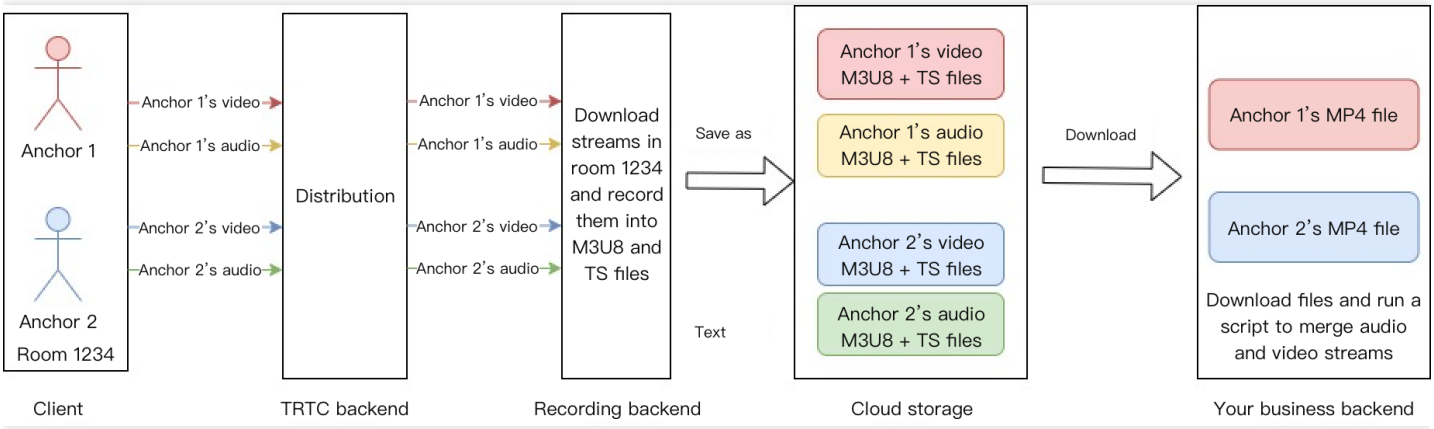
功能概述

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

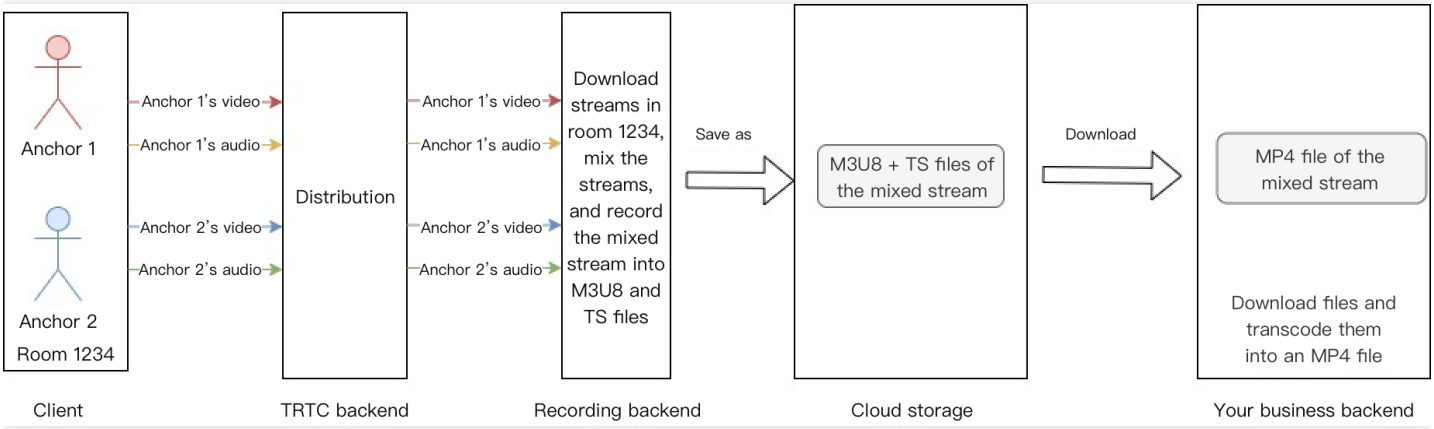


如图所示为单流录制的场景，房间1234里面主播1和主播2都上行了音视频流，假设您订阅了主播1和主播2的音视频流，并设置录制模式为单流录制，录制后台会分别拉取主播1和主播2的音视频流，并把他们录制成独立的媒体文件，包含：

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。

调用流程

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

字段名	解释
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

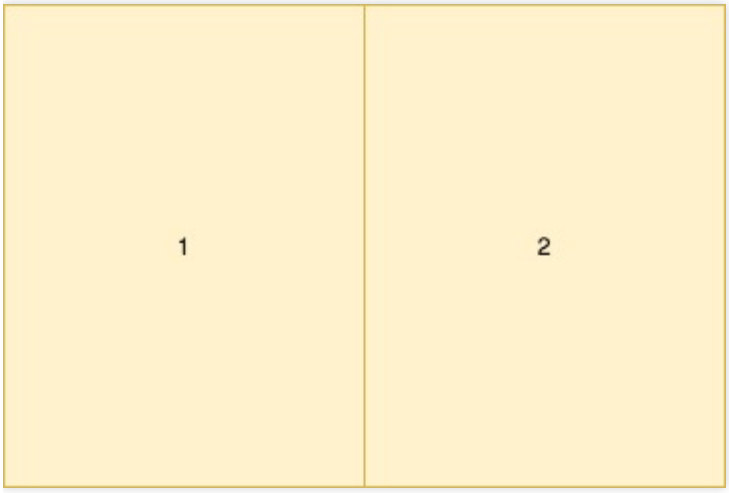
九宫格布局：

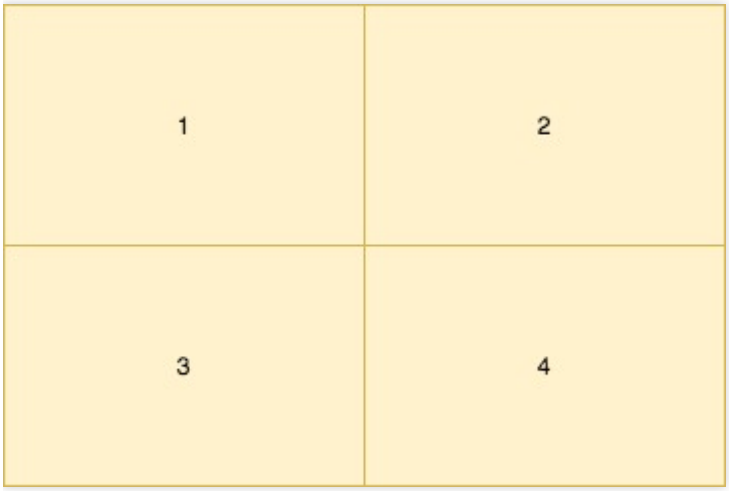
根据主播的数量自动调整每个画面的大小，每个主播的画面大小一致，最多支持25个画面。

- 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：


1


1
2


1
2
3
4

1	2	3
4	5	6
7	8	9

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

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。

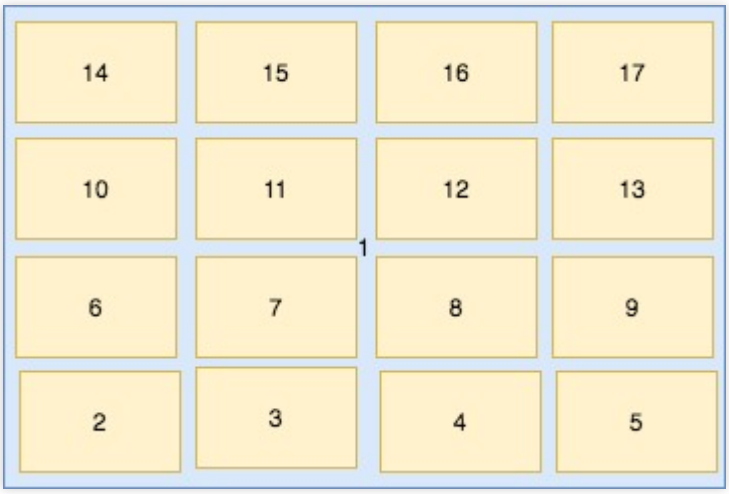
- 子画面小于等于17个时：

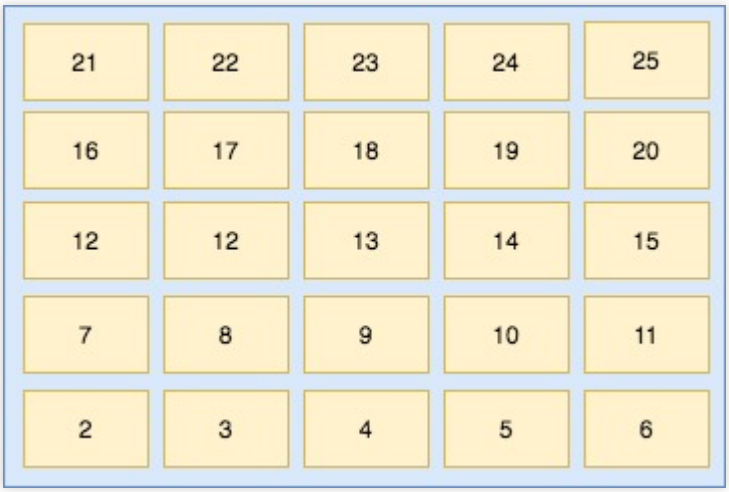
```
每个小画面的宽和高分别为整个画布宽和高的 0.235；
相邻小画面的左右和上下间距分别为整个画布宽和高的 0.012；
小画面距离画布的水平和垂直边距也分别为整个画布宽和高的 0.012；
```

- 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;

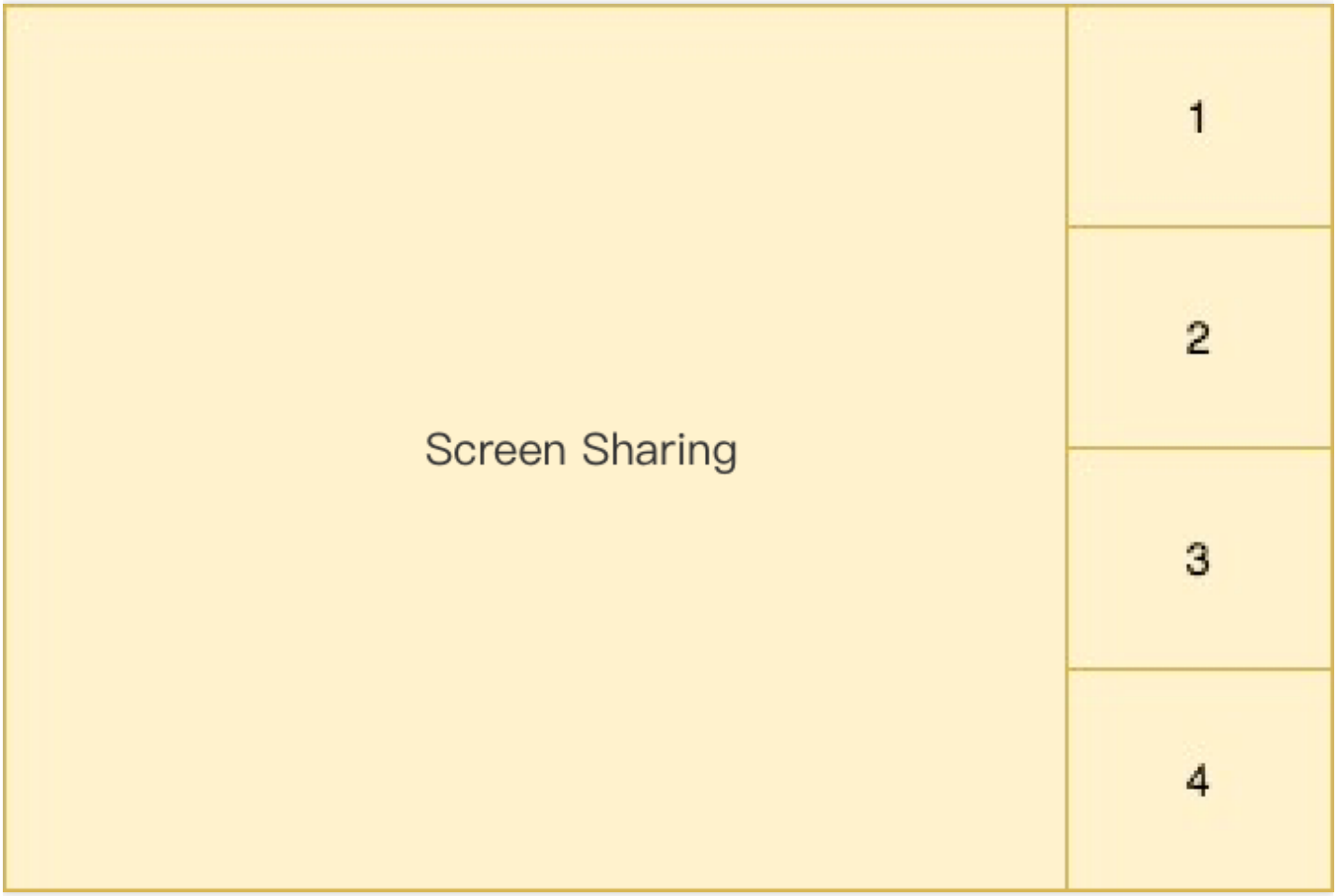
- 子画面大于5且小于等于7个时:
  右侧小画面的宽为整个画布宽的1/7, 右侧小画面的高为整个画布高的1/6；
  左侧大画面的宽为整个画布宽的6/7, 左侧大画面的高为整个画布高;

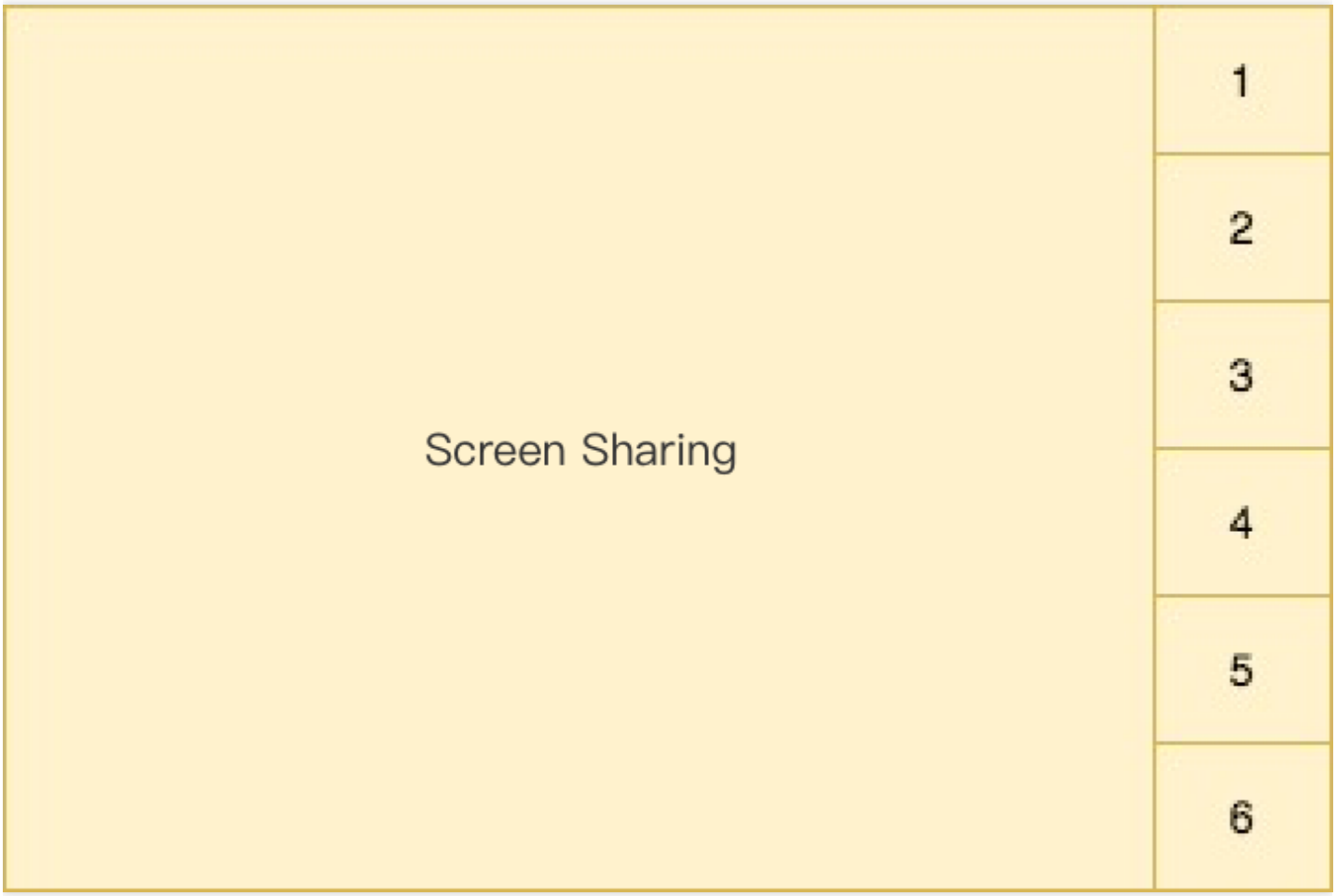
- 子画面大于7且小于等于9个时:
  右侧小画面的宽为整个画布宽的1/9, 右侧小画面的高为整个画布高的1/8；
  左侧大画面的宽为整个画布宽的8/9, 左侧大画面的高为整个画布高;

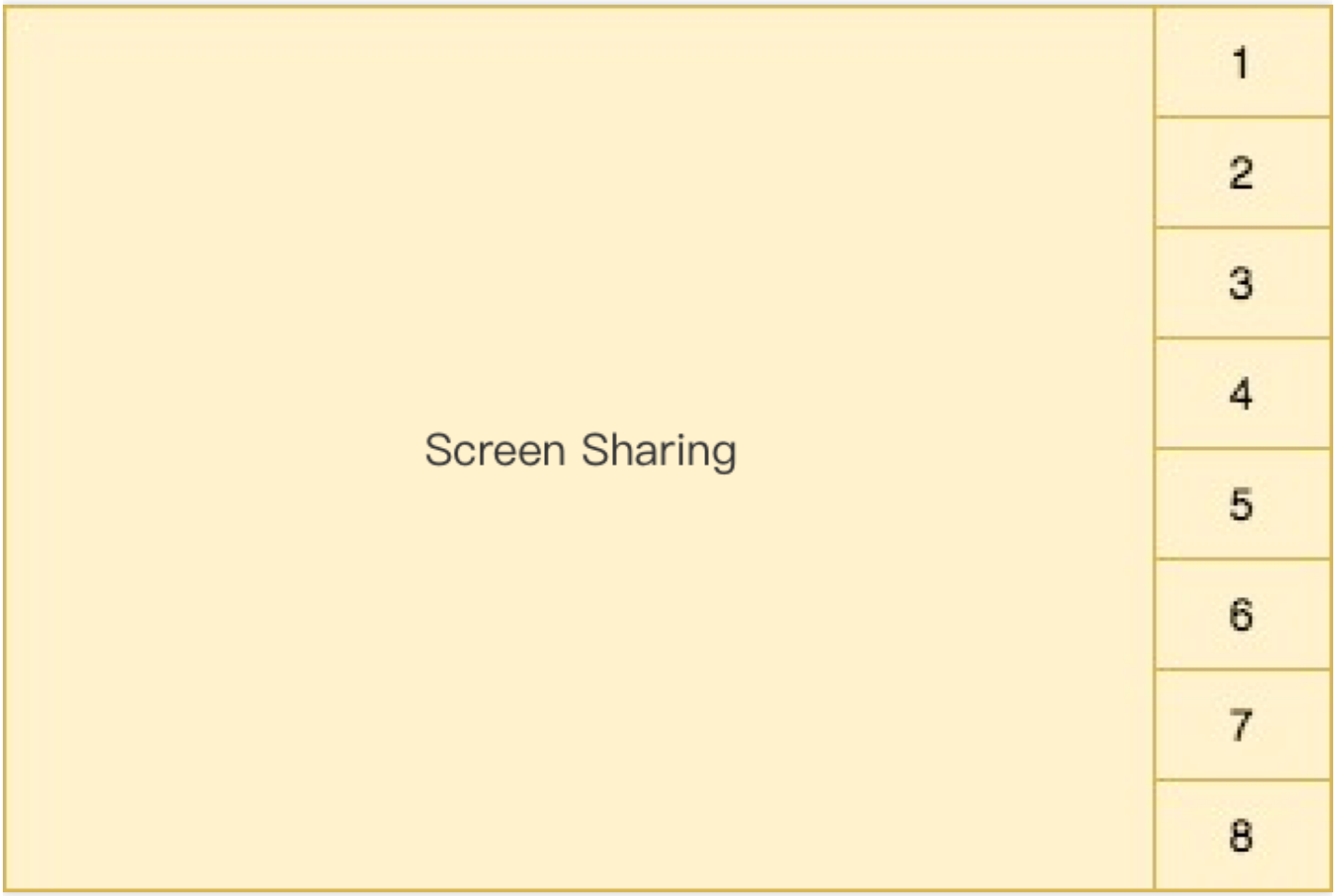
- 子画面大于9小于等于17个时:
  右侧小画面的宽为整个画布宽的1/10,右侧小画面的高为整个画布高的1/8；
  左侧大画面的宽为整个画布宽的4/5,左侧大画面的高为整个画布高;

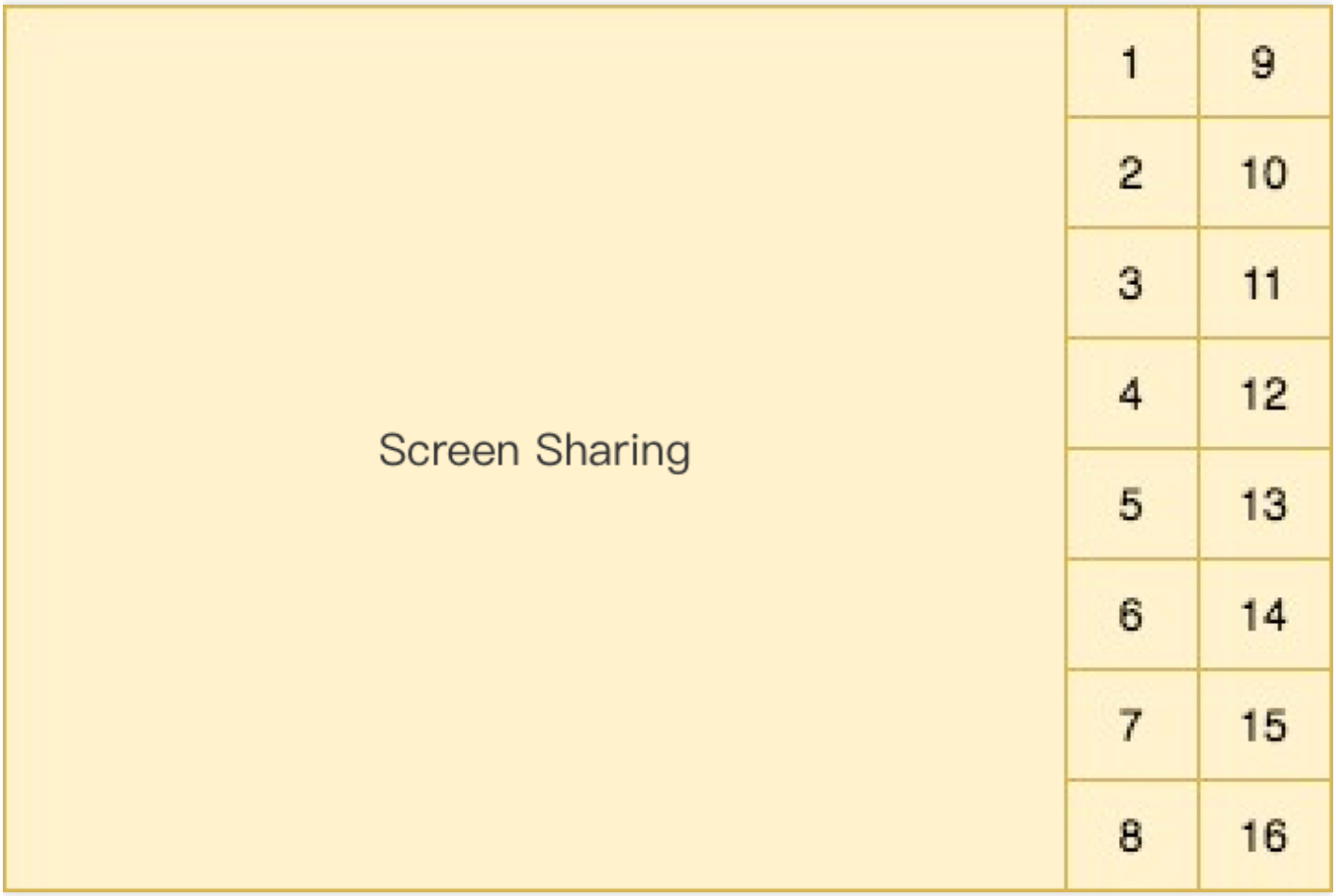
- 子画面大于17个时:
  右（下）侧小画面的宽为整个画布宽的1/10,右（下）侧小画面的高为整个画布高的1/8；
  左侧大画面的宽为整个画布宽的4/5,左侧大画面的高为整个画布高的7/8;

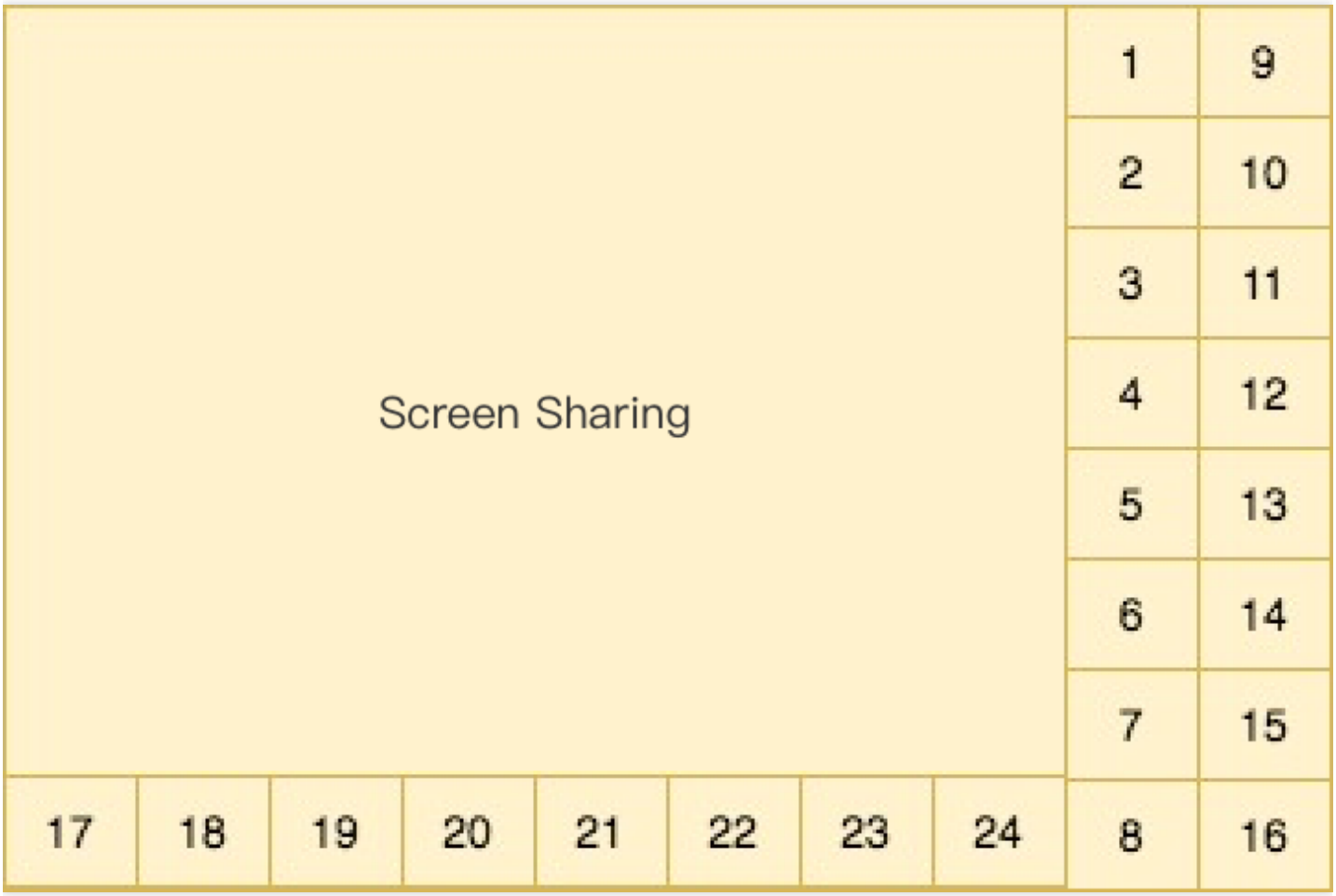
屏幕分享布局随着订阅的子画面增加按照下图进行变化：




Screen Sharing
1
2
3
4
5
6




Screen Sharing
1
9
2
10
3
11
4
12
5
13
6
14
7
15
8
16



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。

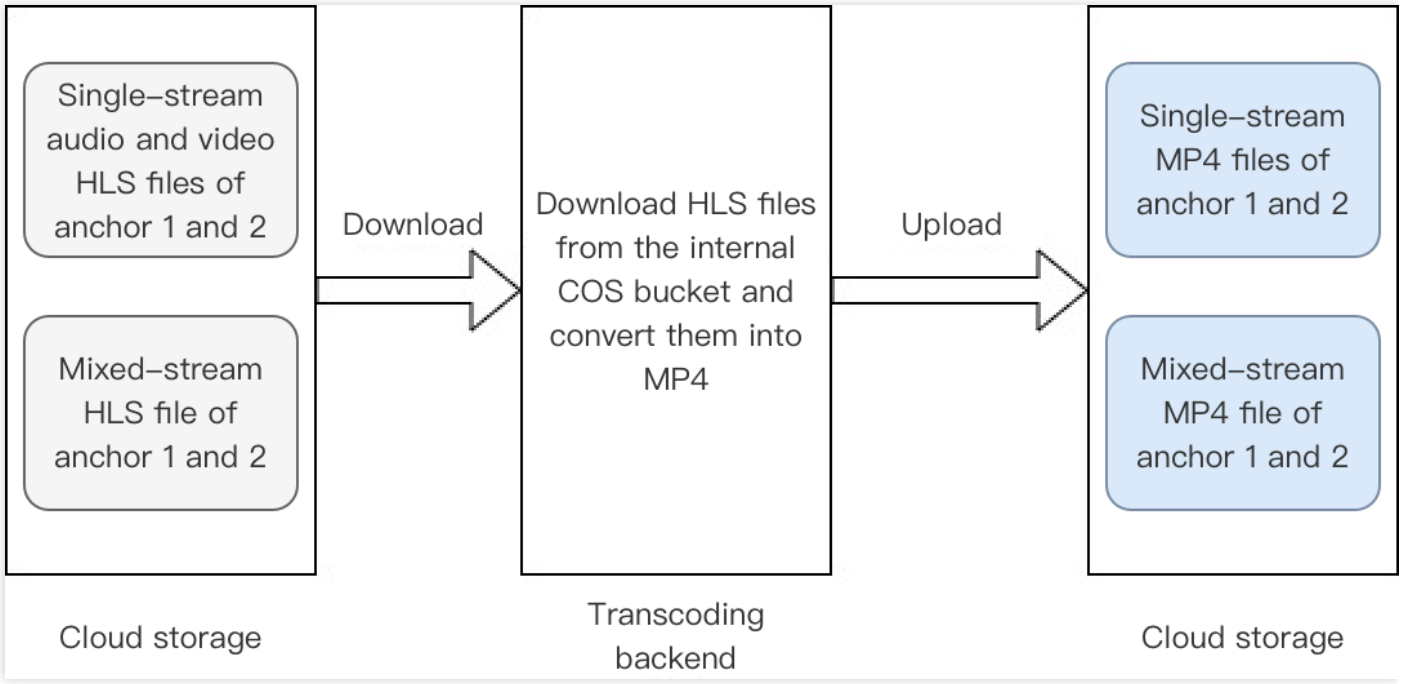
高级功能

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

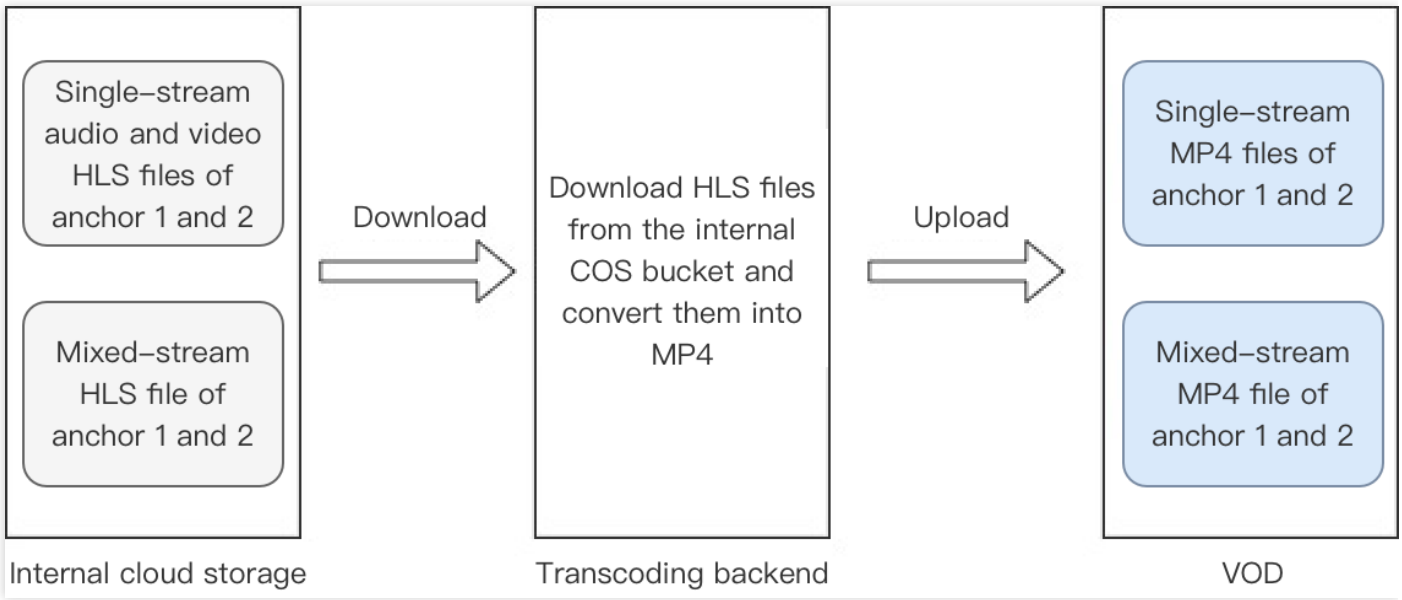


录制上传点播

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。

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
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

字段名	类型	含义
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

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	String	生成的M3U8文件名

```
{
"EventGroupId": 3,
"EventType": 310,
"CallbackTs": 1622191989674,
"EventInfo": {
"RoomId": "20015",
"EventTs": 1622191989,
"UserId": "xx",
"TaskId": "xx",
"Payload": {
"Status": 0,
"FileList": ["xxxx1.mp4", "xxxx2.mp4"]
}
}
}
```

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