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：2022-03-03 17:45:16

适用场景

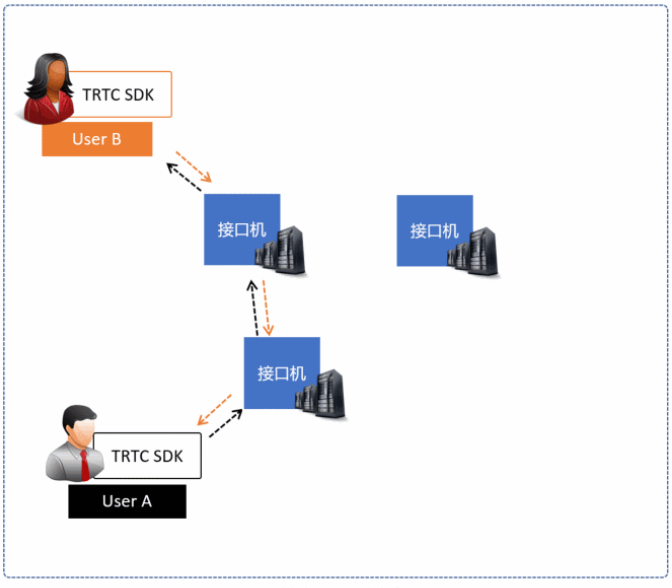
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（AudioCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

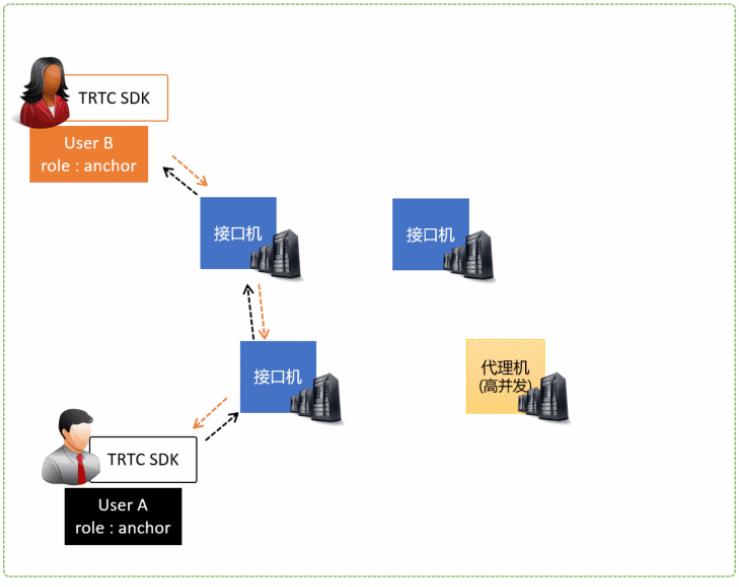
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **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。



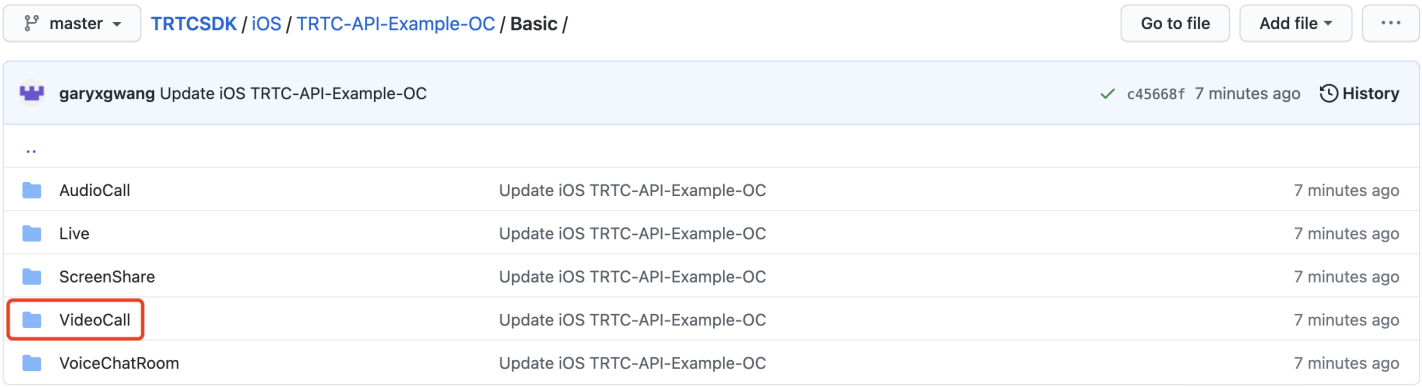
通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。



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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 TRTC SDK 集成到项目中。

方式一：使用 CocoaPods 集成

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

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

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

3. 执行以下命令安装 TRTC SDK 。

```
pod install
```

安装成功后当前项目根目录下会生成一个 xcworkspace 文件。

4. 打开新生成的 xcworkspace 文件即可。

方式二：下载 ZIP 包手动集成

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

步骤2：添加媒体设备权限

在 Info.plist 文件中添加摄像头和麦克风的申请权限：

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

步骤3：初始化 SDK 实例并监听事件回调

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

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

2. 设置delegate属性注册事件回调，并监听相关事件和错误通知。

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

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1Ssll...
roomId	数字	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：2022-03-03 17:43:46

适用场景

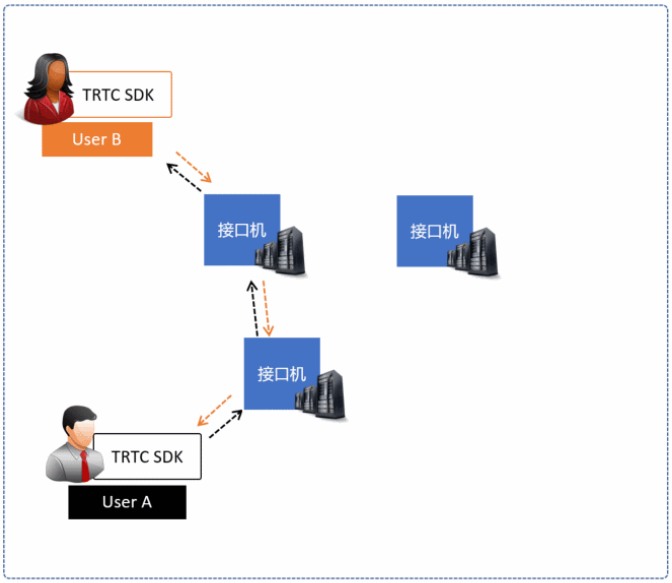
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（AudioCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

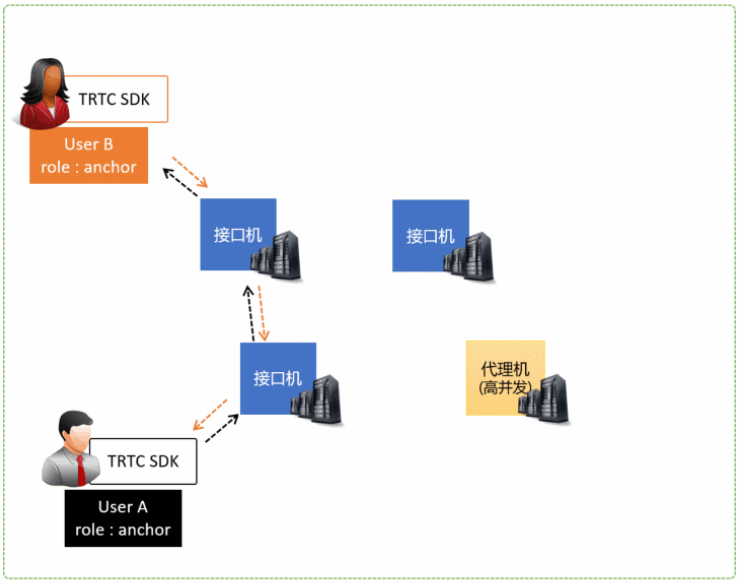
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **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。



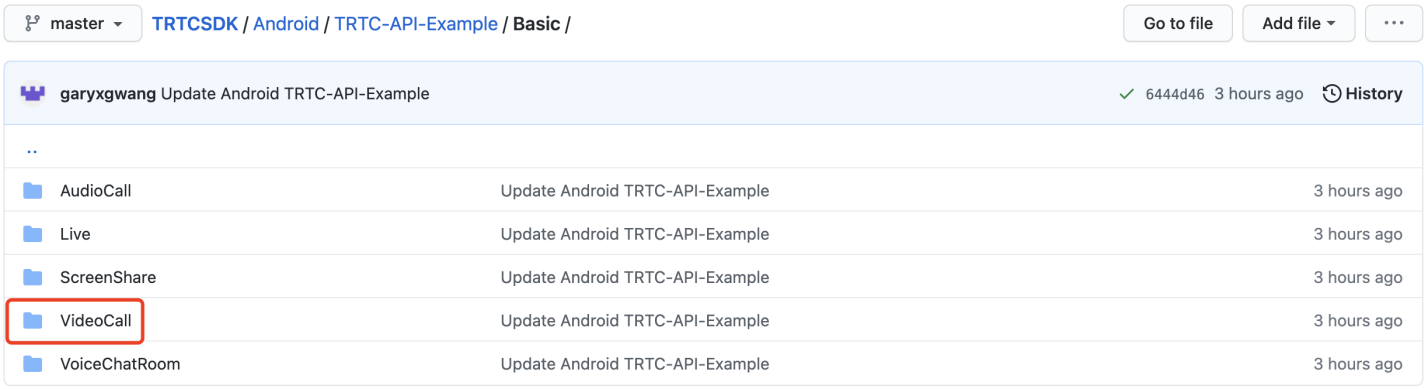
通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。



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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 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(){
// 回调处理理
...
});
```

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

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

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig。	eJyrVareCeYrSy1Ssll...
roomId	数字	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：2021-12-29 15:04:26

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++

```
// TRTCMainViewController.h

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

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

```
virtual void onRemoteUserLeaveRoom(const char* userId, int reason);
virtual void onUserVideoAvailable(const char* userId, bool available);
virtual void onUserAudioAvailable(const char* userId, bool available);
...
private:
ITRTCCloud * m_pTRTCSDK = NULL;
...
}

// TRTCMainViewController.cpp

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

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

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

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

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

C#

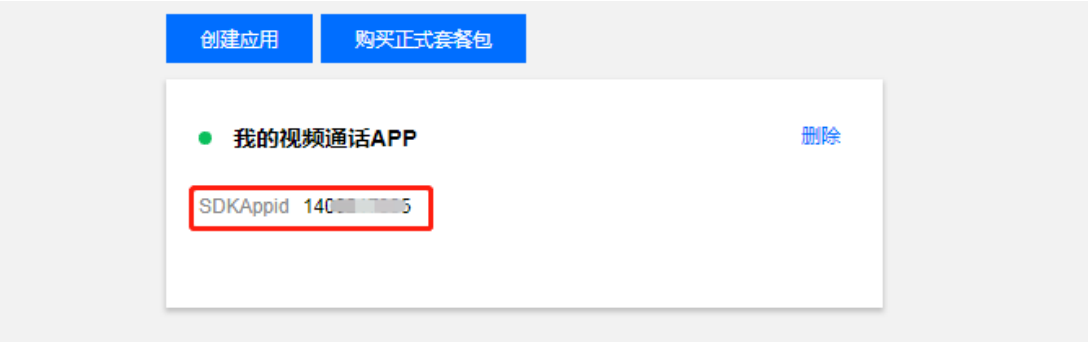
```
// TRTCMainForm.cs

// 继承 ITRTCCloudCallback 事件回调接口类
```

```
public partial class TRTCMainForm : Form, ITRTCCloudCallback, ITRTCLogCallback
{
...
private ITRTCCloud mTRTCCloud;
...

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

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

2. 组装 TRTCParams

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

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



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

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

- **roomId**
  房间号是数字类型，您可以随意指定，但请注意，同一个应用里的两个音视频房间不能分配同一个 roomId。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++

```
// TRTCMainViewController.cpp

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

```
}

...

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

...

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

C#

```
// TRTCMainForm.cs

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

...
```

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

...

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

⚠ 注意：

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

4. 收听远端音频流

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++

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

```
HWND hwnd = pRemoteVideoView->GetSafeHwnd();

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

C#

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

6. 开关本地声音采集

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++

```
// TRTCMainViewController.cpp

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

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

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

...
}
```

C#

```
// TRTCMainForm.cs

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

8. 屏蔽音视频数据流

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

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

9. 退出房间

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

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

C++

```
// TRTCMainViewController.cpp

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

...
}
```

C#

```
// TRTCMainForm.cs

public void OnExit()
{
if(mTRTCCloud != null)
{
mTRTCCloud.exitRoom();
}
}
```

```
...
public void onExitRoom(int reason)
{
// 退房成功
...
}
```

跑通通话模式(Electron)

最近更新时间：2022-03-03 17:44:39

适用场景

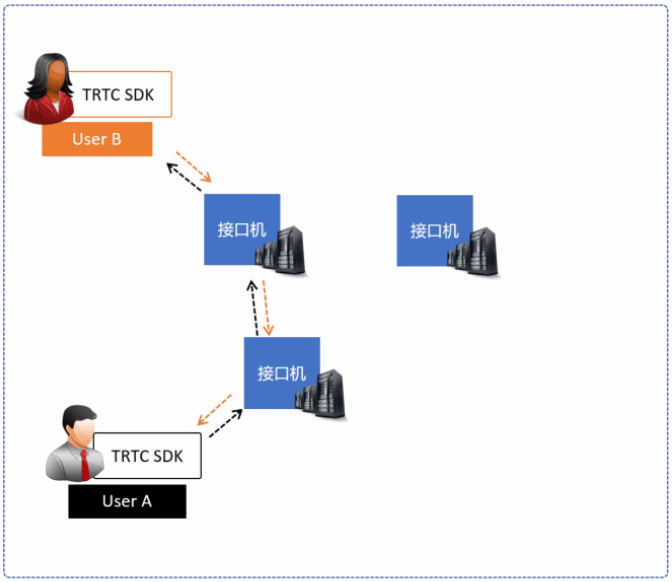
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为 直播模式。
通话模式下的 TRTC，支持单个房间最多300人同时在线，支持最多50人同时发言。适合1对1视频通话、300人视频会议、在线问诊、远程面试、视频客服、在线狼人杀等应用场景。

原理解析

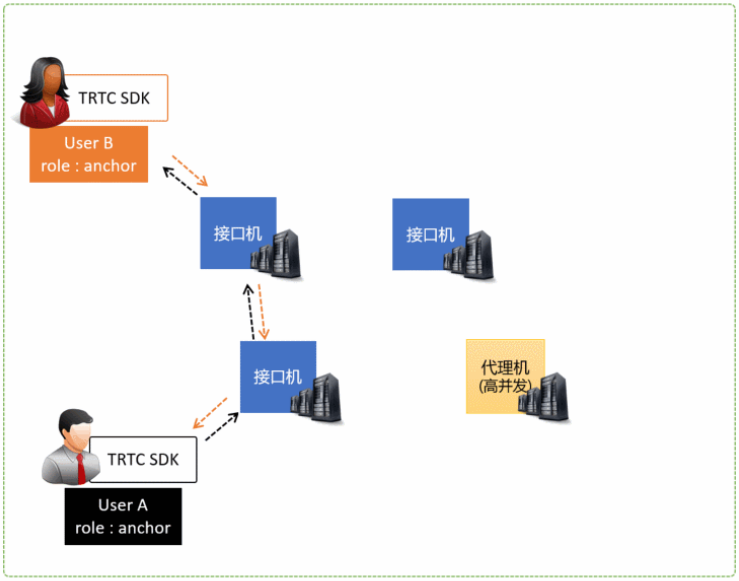
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **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。



通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。

操作步骤

步骤1：尝试跑通官网 SimpleDemo

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

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

步骤2：为您的项目集成 trtc-electron-sdk

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

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

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

步骤3：初始化 SDK 实例并监听事件回调

1. 创建 trtc-electron-sdk 实例：

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

2. 监听 onError 事件：

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

步骤4： 组装进房参数 TRTCParams

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

参数	类型	说明	示例
sdkAppId	数字	应用 ID，您可以在 控制台 >应用管理>应用信息中查找到。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1Ssll...
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 = 'eJyrVareCeYrSy1SsII...';
```

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

步骤5：创建并进入房间

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

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

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

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

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

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

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

步骤6: 订阅远端的音视频流

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

自动订阅（推荐）

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

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

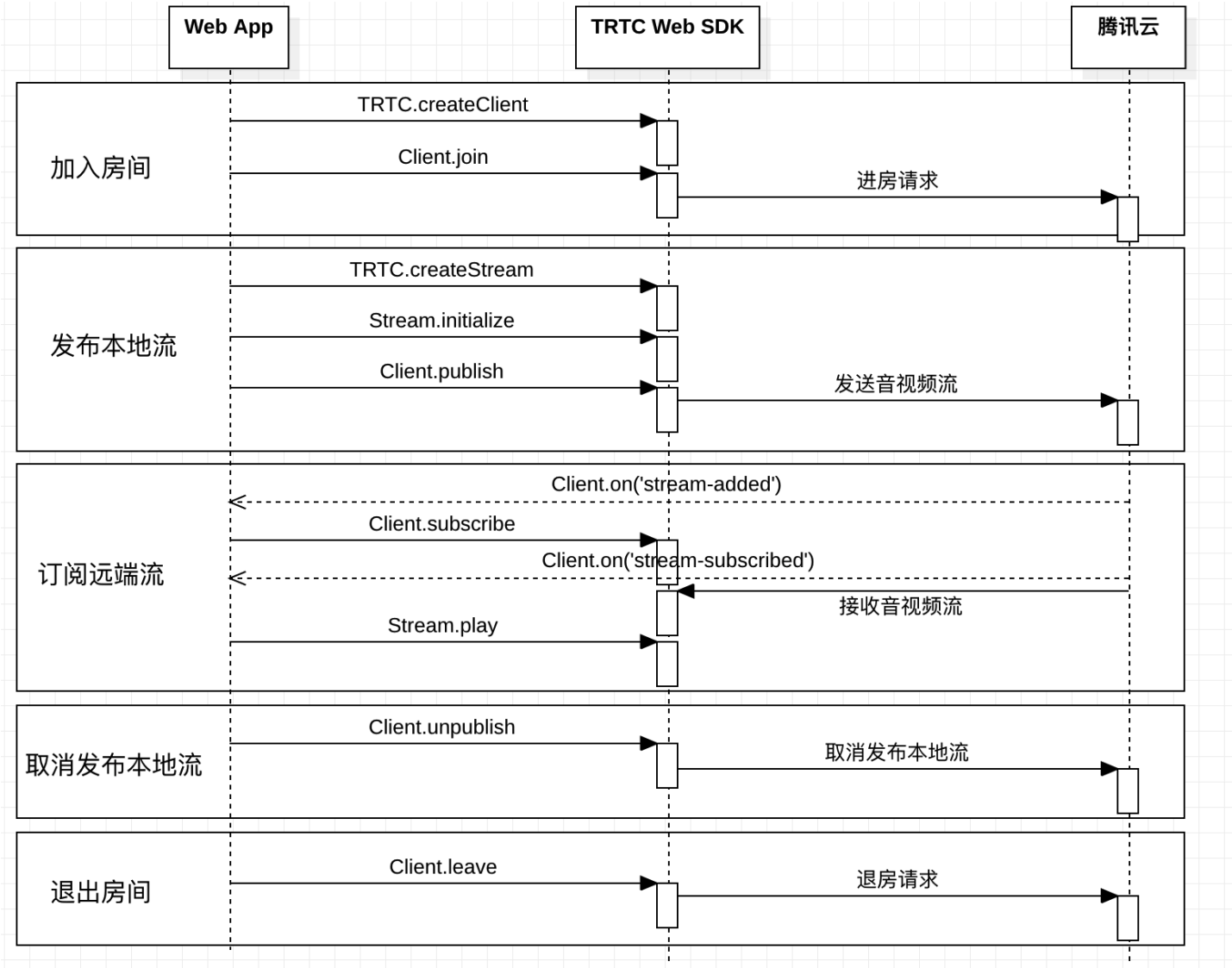
最近更新时间：2022-03-04 16:29:05

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。

跑通通话模式(小程序)

最近更新时间：2021-12-29 15:04:48

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

环境要求

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

前提条件

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

准备工作

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

- 创建了腾讯云实时音视频应用，购买了相应的套餐，并获取到 SDKAppID 和密钥信息。

- 开通小程序类目与推拉流标签权限。
- 小程序服务器域名配置。
- 在您的小程序项目中集成 <trtc-calling> 组件。

部署签名服务

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

组装参数

参数准备

要使用 <trtc-calling> 组件，必须准备好如下参数：

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

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

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

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

TRTCCalling 属性

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

config 参数

参数	类型	必填	说明
sdkAppID	Number	是	开通实时音视频服务创建应用后分配的 SDKAppID

参数	类型	必填	说明
userID	String	是	用户 ID，可以由您的帐号体系指定
userSig	String	是	身份签名（即相当于登录密码），由 userID 计算得出，具体计算方法请参见 如何计算及使用 UserSig
type	Number	是	指定通话类型。1：语音通话，2：视频通话

集成组件

1. 在需要引入组件的页面目录下，配置相应页面的 xxx.json 文件。

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

2. 在相应页面的 xxx.wxml 文件中使用标签。

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

登录

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

```
Page({
// ...
data: {
trtcConfig: {
sdkAppID: 0, // 开通实时音视频服务创建应用后分配的 SDKAppID
userID: 'test_user_001', // 用户 ID，可以由您的帐号系统指定
userSig: 'xxxxxxxxxxx', // 身份签名，相当于登录密码的作用
type: 2, // 通话模式
},
},
/**
* 生命周期函数--监听页面加载
*/
onLoad: function (options) {
// 将初始化后到TRTCCalling实例注册到this.TRTCCalling中，this.TRTCCalling 可使用TRTCCalling所以方法功能。
this.TRTCCalling = this.selectComponent('#TRTCCalling-component');
// 绑定需要监听到事件
```

```
this.bindTRTCCallingRoomEvent();
// 登录TRTCCalling
this.TRTCCalling.login();
},
/**
* 生命周期函数--监听页面卸载
*/
onUnload: function () {
// 取消监听事件
this.unbindTRTCCallingRoomEvent();
// 退出登录
this.TRTCCalling.logout();
},
invitedEvent() {
console.log('收到邀请')
},

hangupEvent() {
console.log('挂断')
},

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

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

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

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

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

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

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

```
},

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

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

进行通话

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

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

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

跑通直播模式
跑通直播模式(iOS&Mac)

最近更新时间：2022-03-03 17:46:55

适用场景

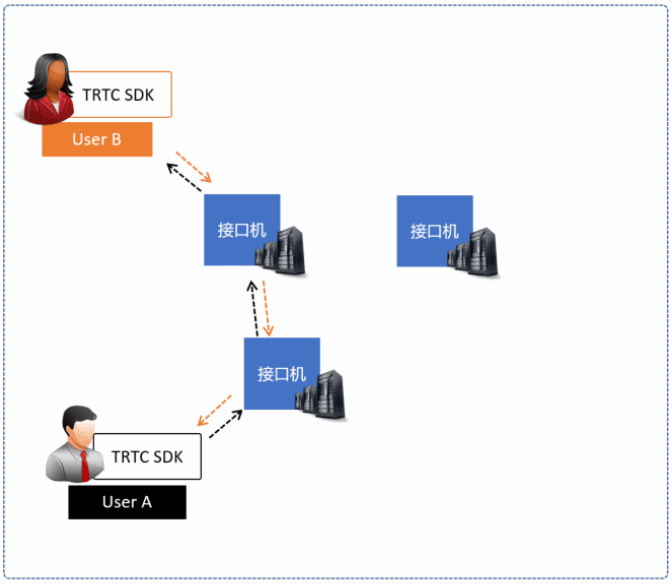
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为 通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为直播模式。
直播模式下的 TRTC，支持单个房间最多10万人同时在线，具备小于300ms的连麦延迟和小于1000ms的观看延迟，以及平滑上下麦切换技术。适用低延时互动直播、十万人互动课堂、视频相亲、在线教育、远程培训、超大型会议等应用场景。

原理解析

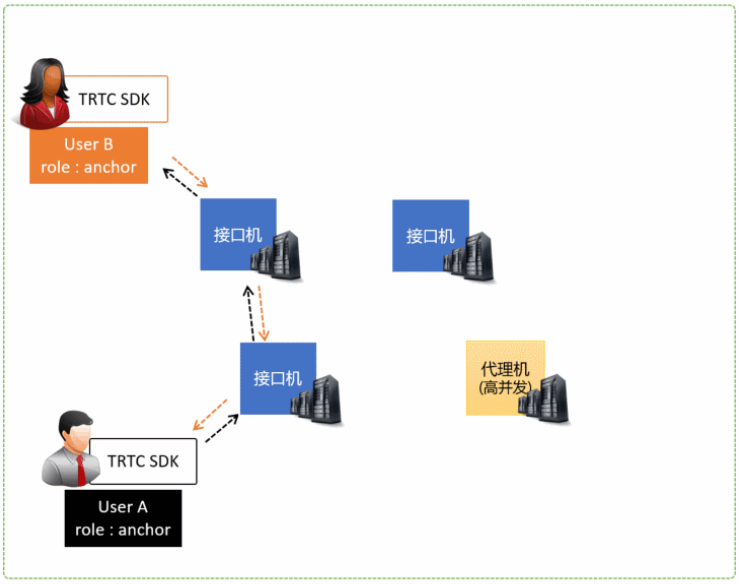
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **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。



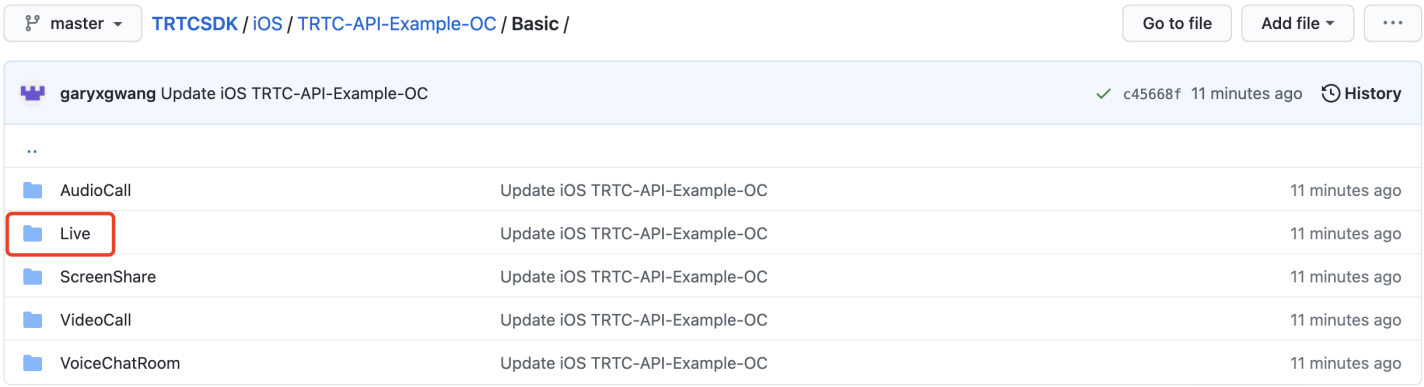
通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。



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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 TRTC SDK 集成到项目中。

方式一：使用 CocoaPods 集成

1. 安装 CocoaPods，具体操作请参见 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 *)extInfo {
if (ERR_ROOM_ENTER_FAIL == errCode) {
[self toastTip:@"进房失败"];
[self.trtcCloud exitRoom];
}
}
```

步骤4：组装进房参数 TRTCParams

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

参数名称	字段类型	补充说明	填写示例
sdkAppId	数字	应用 ID，您可以在 实时音视频控制台 中查看 SDKAppID。	1400000123
userId	字符串	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。建议结合业务实际账号体系自行设置。	test_user_001
userSig	字符串	基于 userId 可以计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	eJyrVareCeYrSy1Ssll...
roomId	数字	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：

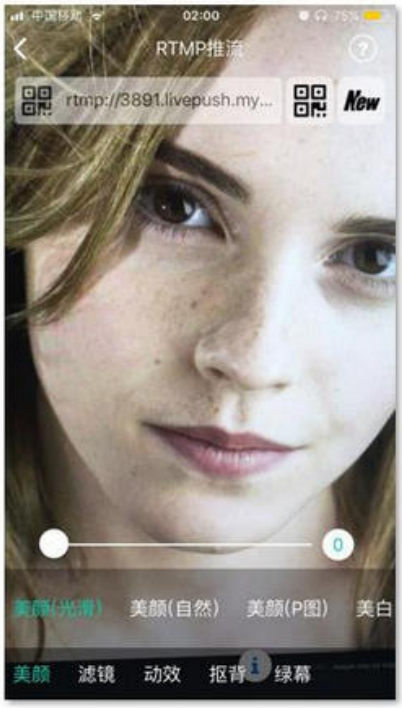
- Fill 模式表示填充，画面可能会被等比放大和裁剪，但不会有黑边。
- Fit 模式表示适应，画面可能会等比缩小以完全显示其内容，可能会有黑边。

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。

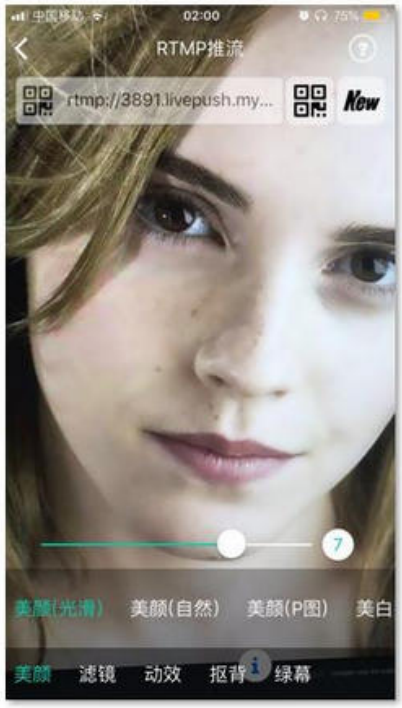
```
//示例代码：发布本地的音视频流
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];

//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

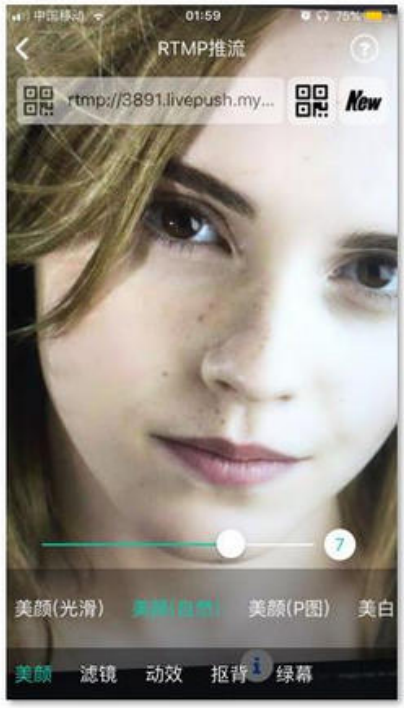
1. 主播端调用 getBeautyManager() 可以获取美颜设置接口 TXBeautyManager。
2. 主播端调用 setBeautyStyle() 可以设置美颜风格：
   - Smooth：光滑，效果比较明显，类似网红风格。
   - Nature：自然，磨皮算法更多地保留了面部细节，主观感受上会更加自然。
   - Pitu：仅 企业版 才支持。
3. 主播端调用 setBeautyLevel() 可以设置磨皮的级别，一般设置为5即可。
4. 主播端调用 setWhitenessLevel() 可以设置美白级别，一般设置为5即可。
5. 由于 iPhone 的摄像头调色默认偏黄，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。



美颜关闭



美颜风格：光滑



美颜风格：自然

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:NSJSONWritingPrettyPrinted error:ni
l];
```

```
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：2022-03-03 17:47:02

适用场景

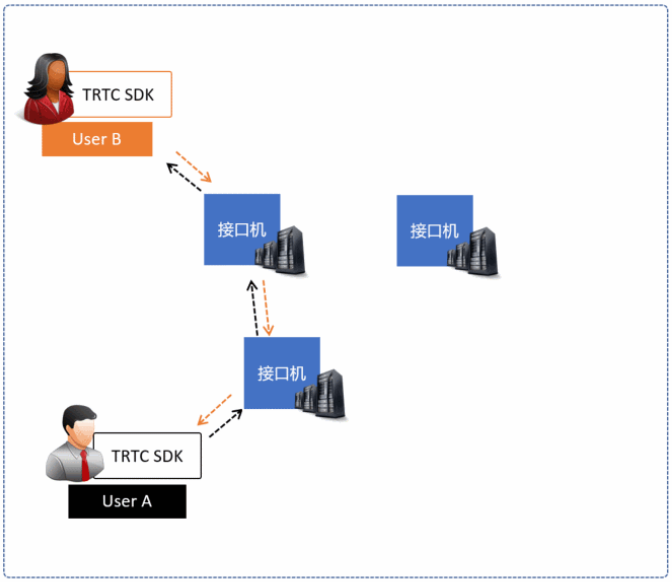
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为 通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为直播模式。
直播模式下的 TRTC，支持单个房间最多10万人同时在线，具备小于300ms的连麦延迟和小于1000ms的观看延迟，以及平滑上下麦切换技术。适用低延时互动直播、十万人互动课堂、视频相亲、在线教育、远程培训、超大型会议等应用场景。

原理解析

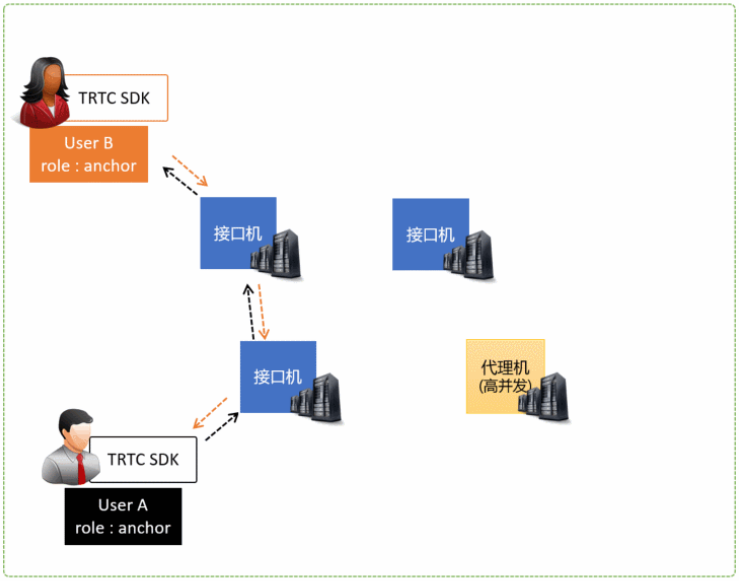
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **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。



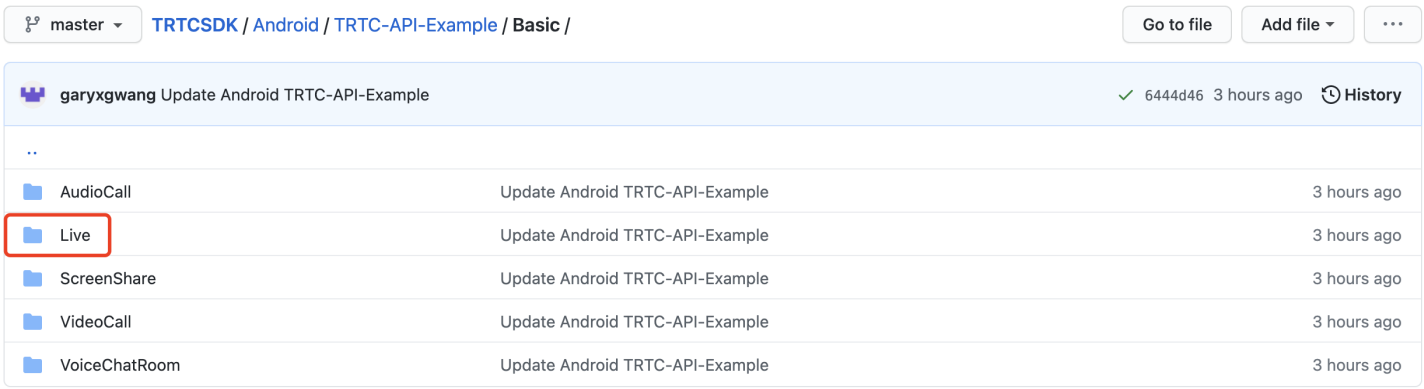
通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。



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

操作步骤

步骤1：集成 SDK

您可以选择以下方式将 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
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, username));
```

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：2021-12-29 15:05:48

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

```
// 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->addCallback(this);
}

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

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

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

C#版

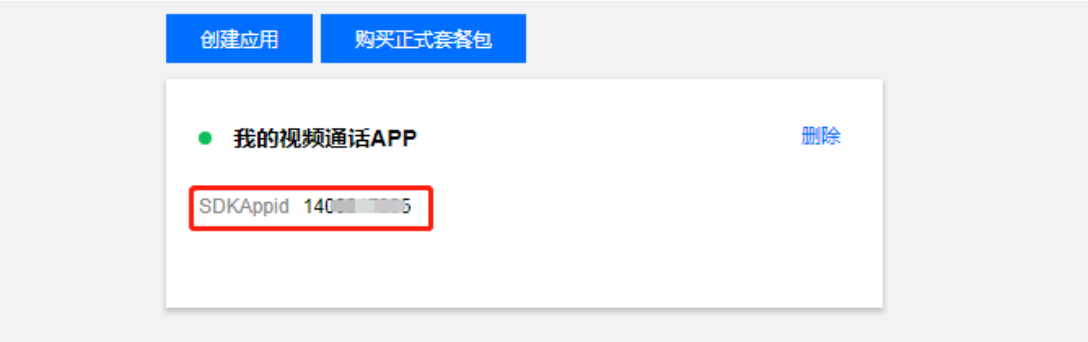
```
// TRTCMainForm.cs

// 继承 ITRTCCloudCallback 事件回调接口类
public partial class TRTCMainForm : Form, ITRTCCloudCallback, ITRTCLogCallback
```

```
{
...
private ITRTCCloud mTRTCCloud;
...

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

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

2. 组装 TRTCParams

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

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



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

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

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

}
```

C#版

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

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

```
// 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->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
}
```

C#版

```
// TRTCMainForm.cs

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

...

public void onError(TXLiteAVError errCode, string errMsg, IntPtr arg)
{
if(errCode == TXLiteAVError.ERR_ROOM_ENTER_FAIL)
{
Log.E(String.Format("errCode : {0}, errMsg : {1}, arg = {2}", errCode, errMsg, arg));
```

```
// 检查 userSig 是否合法、网络是否正常等
}
}

...

public void onEnterRoom(int result)
{
// 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

```
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->enterRoom(params, TRTCAppSceneLIVE);
}
}
```

C#版

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

```
{
mTRTCCloud.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

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

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

C#版

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

```
}
}
```

> ⚠ 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

```
// TRTCMainViewController.cpp

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

}
```

C#版

```
// TRTCMainForm.cs

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

```
// 退房成功
...
}
```

跑通直播模式(Electron)

最近更新时间：2022-03-03 17:47:16

适用场景

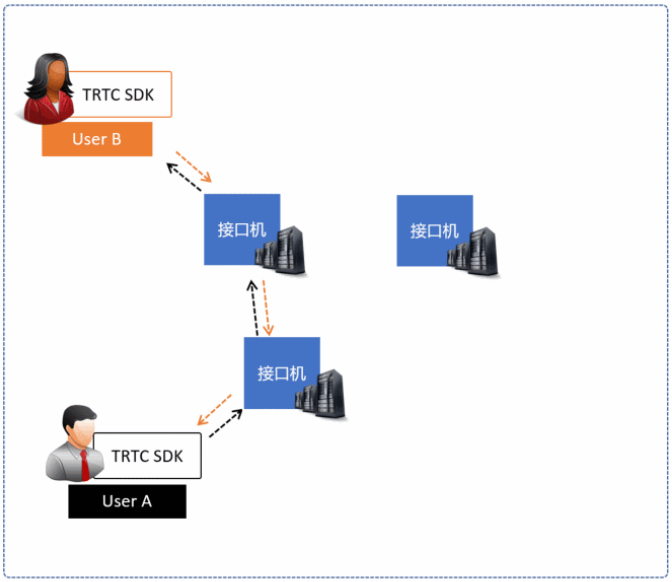
TRTC 支持四种不同的进房模式，其中视频通话（VideoCall）和语音通话（VoiceCall）统称为 通话模式，视频互动直播（Live）和语音互动直播（VoiceChatRoom）统称为直播模式。
直播模式下的 TRTC，支持单个房间最多10万人同时在线，具备小于300ms的连麦延迟和小于1000ms的观看延迟，以及平滑上下麦切换技术。适用低延时互动直播、十万人互动课堂、视频相亲、在线教育、远程培训、超大型会议等应用场景。

原理解析

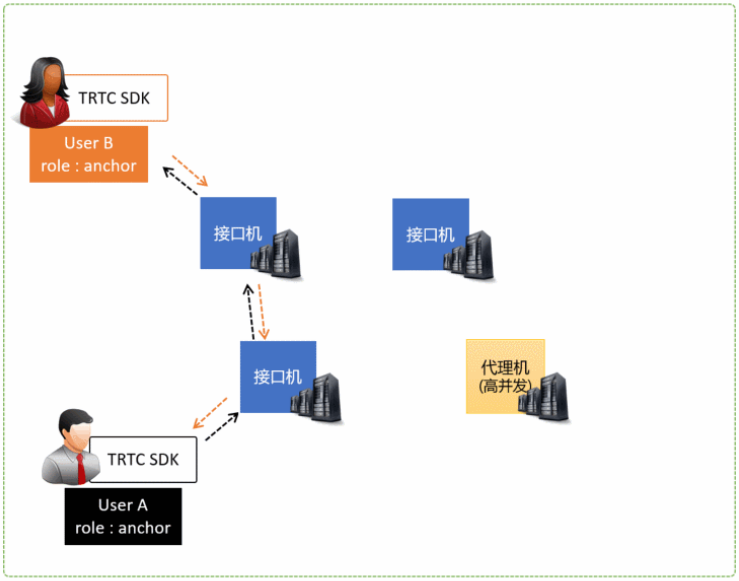
TRTC 云服务由两种不同类型的服务器节点组成，分别是“接口机”和“代理机”：

- **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。



通话模式（不区分角色）



直播模式（有主播和观众两种角色）

示例代码

您可以登录 Github 获取本文档相关的示例代码。

操作步骤

步骤1：尝试跑通官网 SimpleDemo

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

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

步骤2：为您的项目集成 trtc-electron-sdk

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

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

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

步骤3：初始化 SDK 实例并监听事件回调

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
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 。	eJyrVareCeYrSy1Ssll...
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 = 'eJyrVareCeYrSy1SsII...';
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 > 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 = 'eJyrVareCeYrSy1SsII...';
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 = 'eJyrVareCeYrSy1SsII...';
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）。

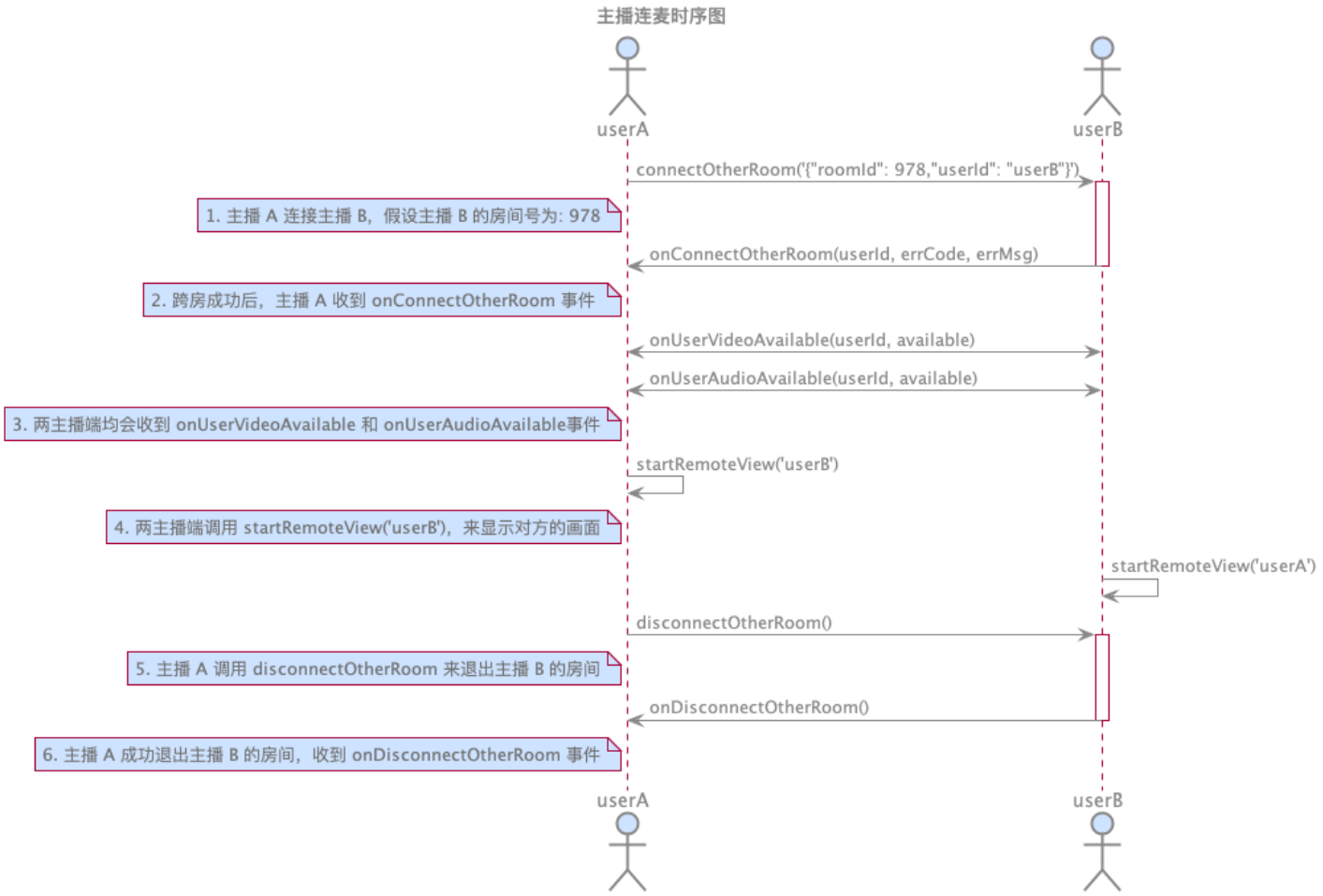
2. 观众端调用 startLocalPreview() 可以开启本地的画面。
3. 观众端调用 startLocalAudio() 开启麦克风采音。

```
//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：2022-03-03 14:16:27

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

示例

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

步骤1：创建 Client 对象

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

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

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

步骤2：以观众身份进入直播房间

调用 Client.join() 进入音视频通话房间。参数设置：

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

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

步骤3：收看直播

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

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

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

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

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

步骤4：跟主播连麦互动

步骤4.1：切换角色

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

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

步骤4.2：连麦互动

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

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

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

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

3. 初始化本地流成功时，播放本地流。

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

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

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

步骤5：退出直播房间

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

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

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

实时屏幕分享
实时屏幕分享(iOS)

最近更新时间：2022-03-03 17:42:37

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	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

应用内分享

应用内分享的方案非常简单，只需要调用 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：

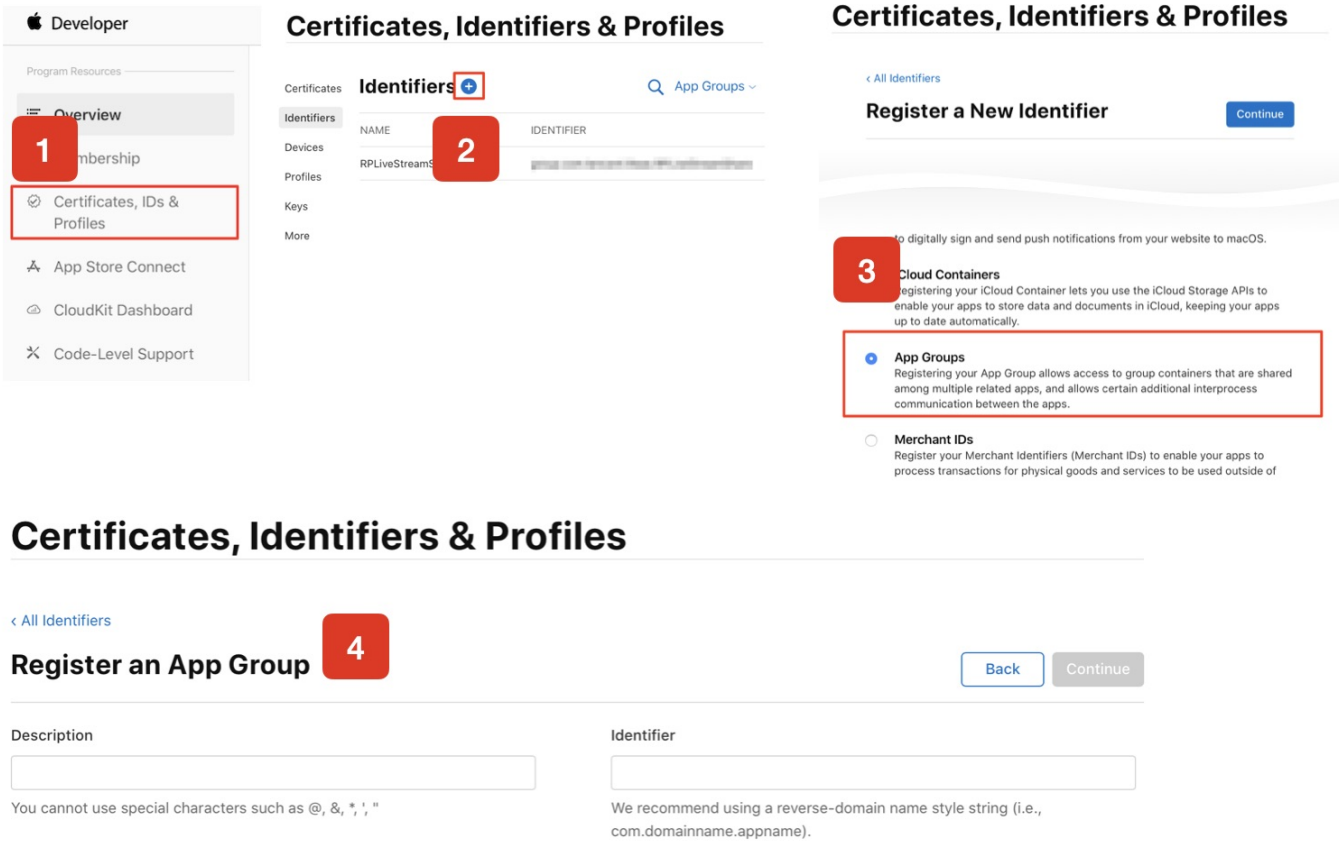
1. 创建 App Group，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 Broadcast Upload Extension 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 TXLiteAVSDK_ReplayKitExt.framework。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 Broadcast Upload Extension 的录屏数据。
4. 使用 Demo 中预先实现的一个 helper class（RPSystemBroadcastPickerView），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

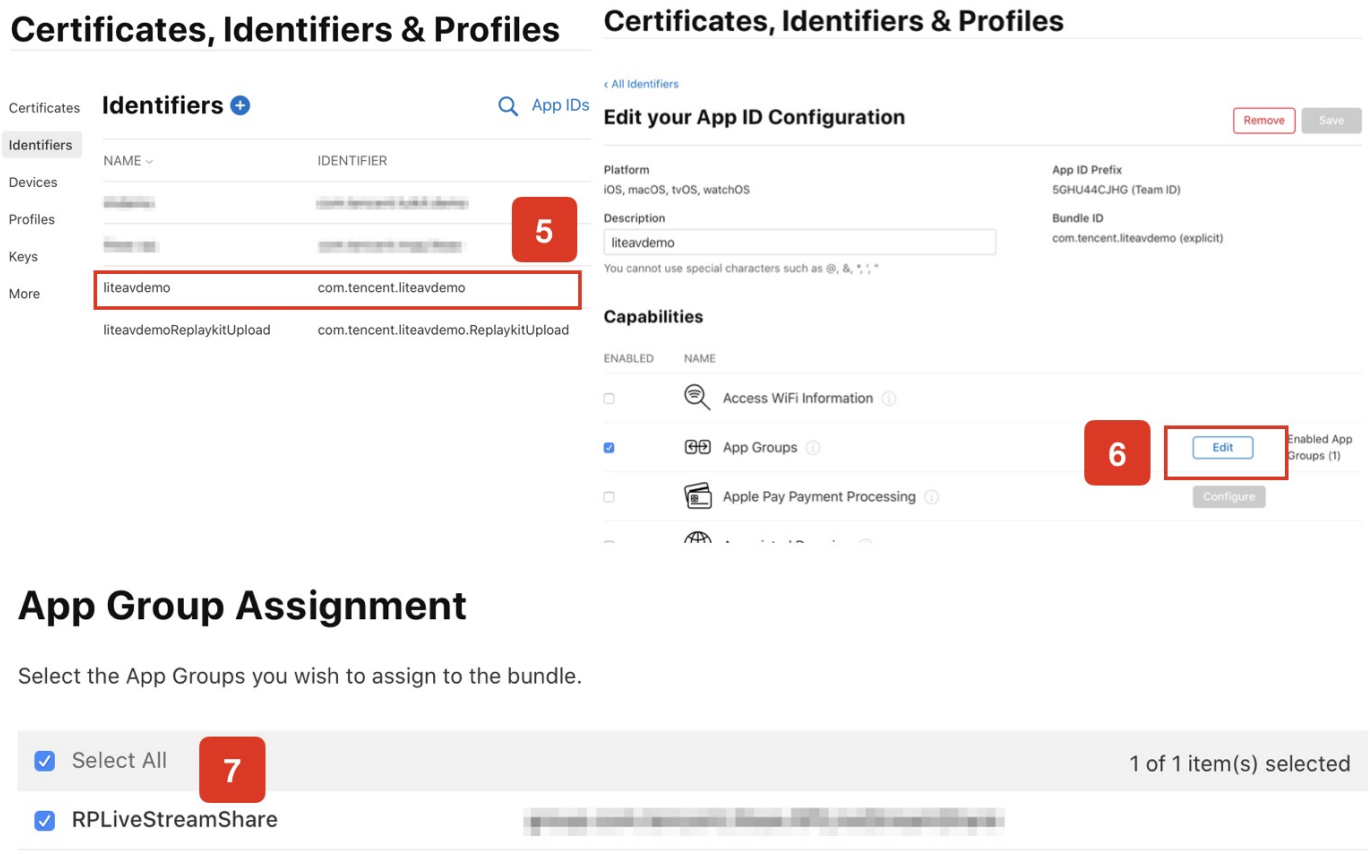
使用您的帐号登录 https://developer.apple.com/，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

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】。

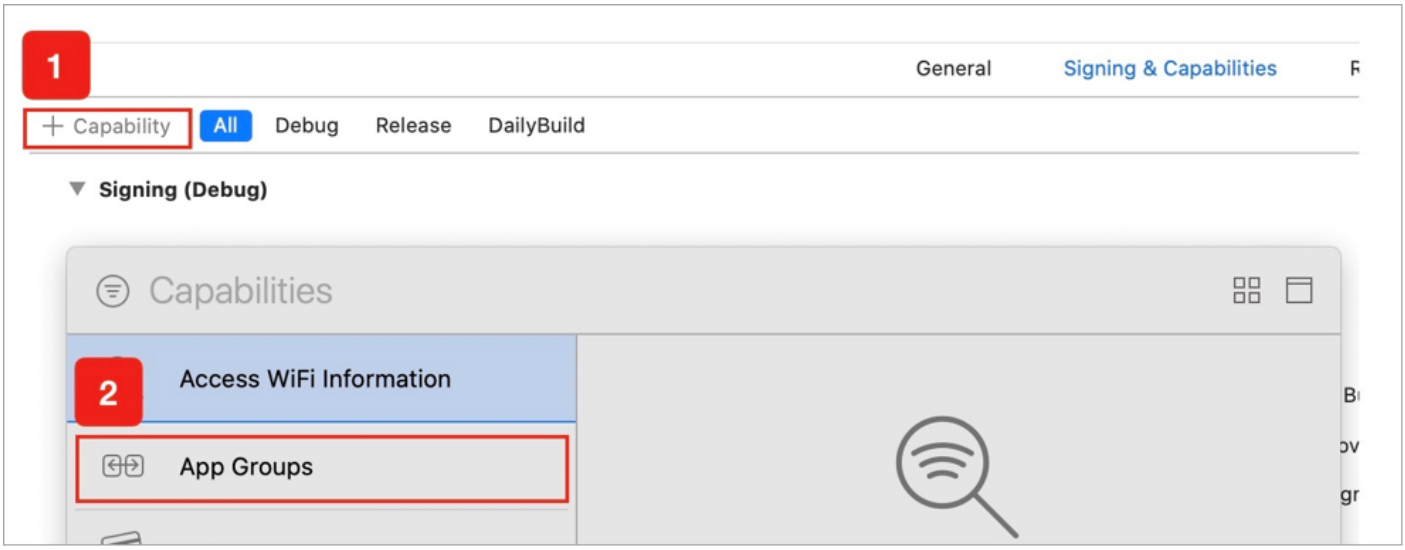
7. 在弹出的表单中选择您之前创建的 App Group，单击【Continue】返回编辑页，单击【Save】保存。



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

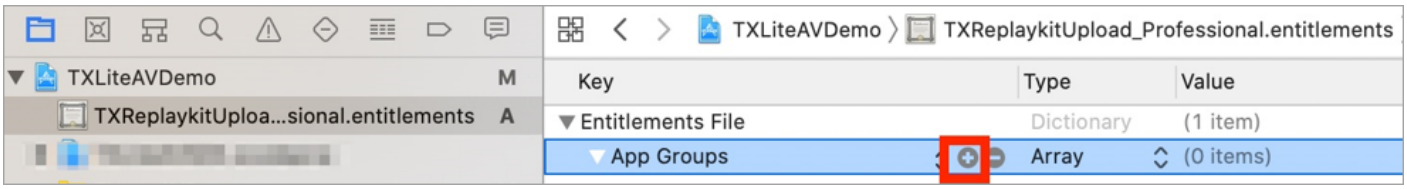
步骤2：创建 Broadcast Upload Extension

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。**

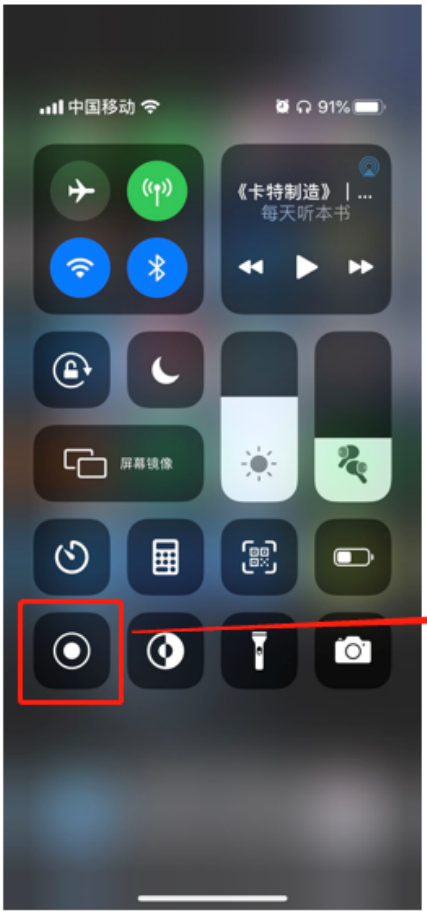
```
#import "SampleHandler.h"
@import TXLiteAVSDK_ReplayKitExt;

#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)sampleBufferType
{
switch (sampleBufferType) {
case RPSampleBufferTypeVideo:
[[TXReplayKitExt sharedInstance] sendVideoSampleBuffer:sampleBuffer];
break;
case RPSampleBufferTypeAudioApp:
// Handle audio sample buffer for app audio
break;
case RPSampleBufferTypeAudioMic:
// Handle audio sample buffer for mic audio
break;

default:
break;
}
}
@end
```

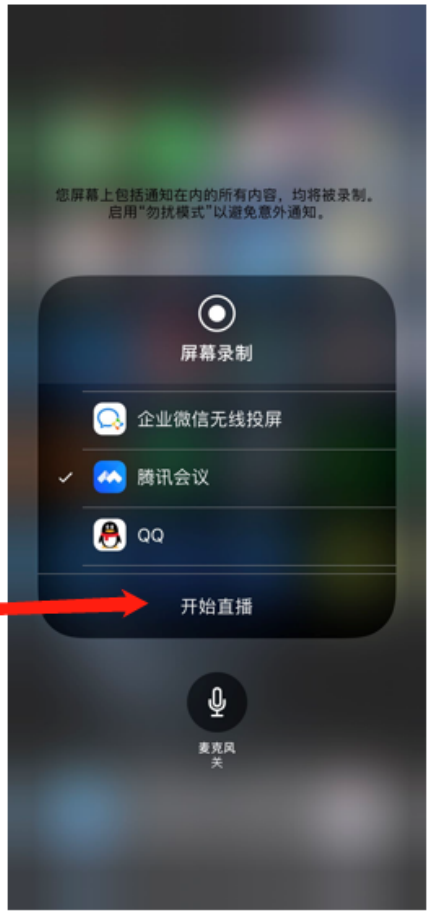
步骤3：对接主 App 端的接收逻辑

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

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



长按“录屏” 按钮



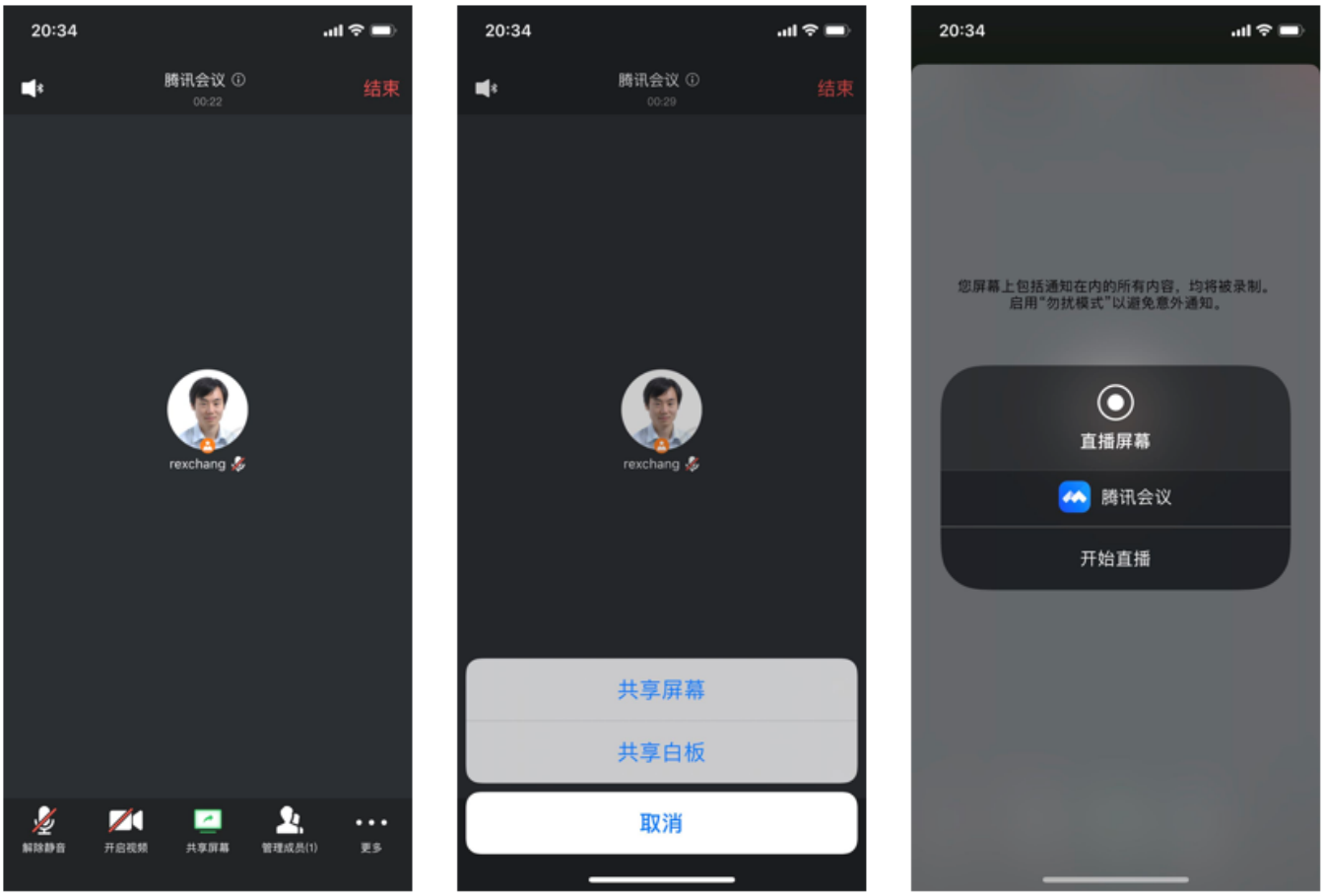
点击“开始直播”

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

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

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

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

步骤4：增加屏幕分享的触发按钮（可选）

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



点击分享按钮　　触发屏幕分享　　开始直播

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

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

⚠ 注意：

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

观看屏幕分享

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

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

实时屏幕分享(Android)

最近更新时间：2022-03-03 17:41:54

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。如下图所示：

支持的平台

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

启动屏幕分享

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

添加 Activity

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

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

设定视频编码参数

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

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

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

弹出悬浮窗以避免被强杀

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

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

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

- **方案2：弹出摄像头预览窗**
  由于 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：2021-08-09 14:55:36

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	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

依赖的 API

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

获取分享目标

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

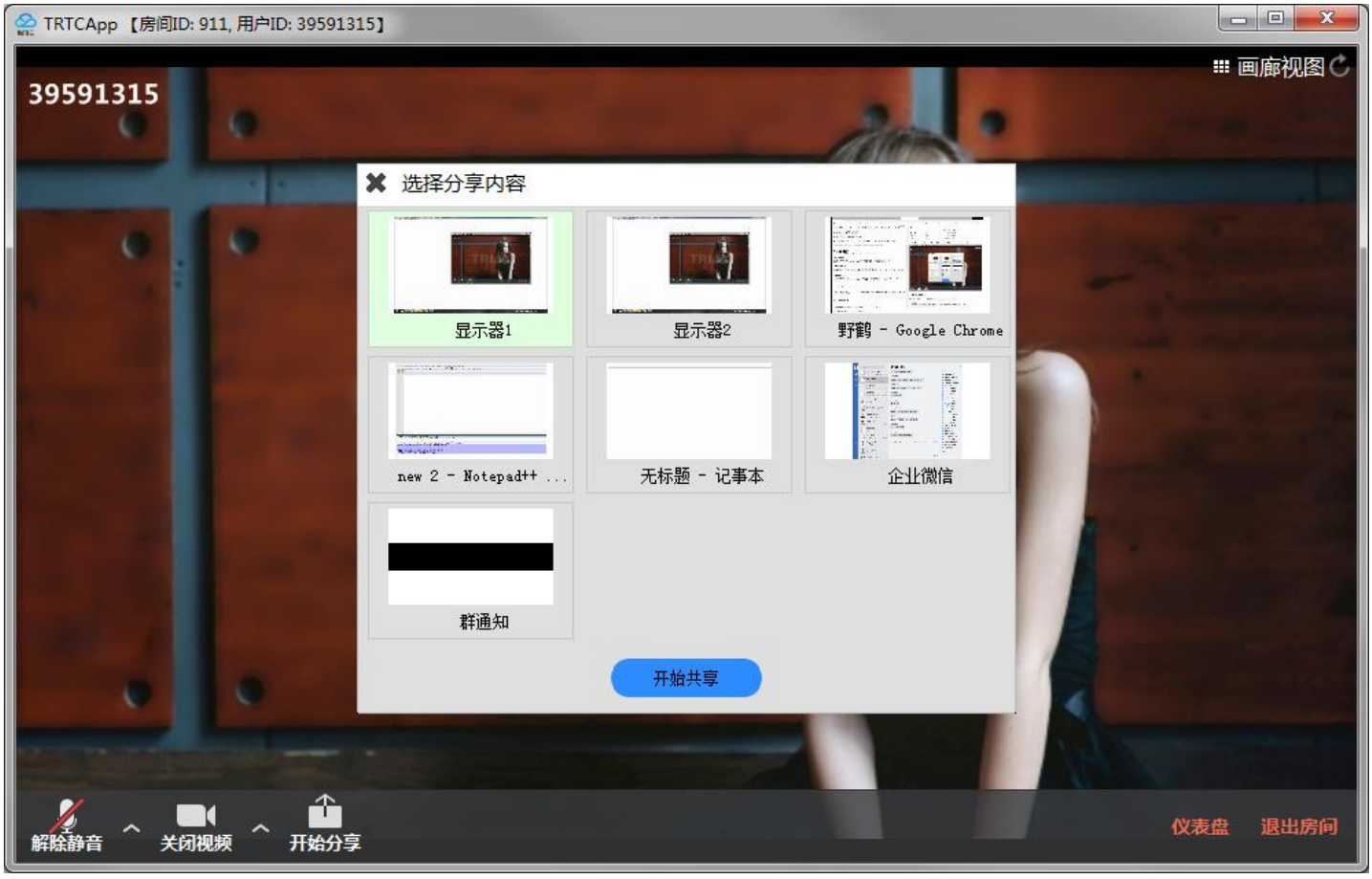
> 说明：
> Windows 里的桌面屏幕也是一个窗口，叫桌面窗口（Desktop），有两台显示器时，每一台显示器都有一个对应的桌面窗口。所以，getScreenCaptureSources 返回的窗口列表里也会有 Desktop 窗口。

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

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

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

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



选择分享目标

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

- **整个屏幕分享：**
  即分享整个屏幕窗口，支持多显示器分屏的情况。需要指定一个 sourceInfoList 中 type 为 TRTCScreenCaptureSourceTypeScreen

的 source 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

- **指定区域分享：**
  即分享屏幕的某个区域，需要用户圈定区域的位置坐标。需要指定一个 sourceInfoList 中 type 为 TRTCScreenCaptureSourceTypeScreen 的 source 参数，并将 captureRect 设为非 NULL，例如 { 100, 100, 300, 300 }。
- **指定窗口分享：**
  即分享目标窗口的内容，需要用户选择要分享的窗口。需要指定一个 sourceInfoList 中 type 为 TRTCScreenCaptureSourceTypeWindow 的 source 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

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

开始屏幕分享

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

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

设定画面质量

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

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

观看屏幕分享

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

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

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

常见问题

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

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

实时屏幕分享(Mac)

最近更新时间：2021-08-09 14:54:45

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	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

获取分享目标

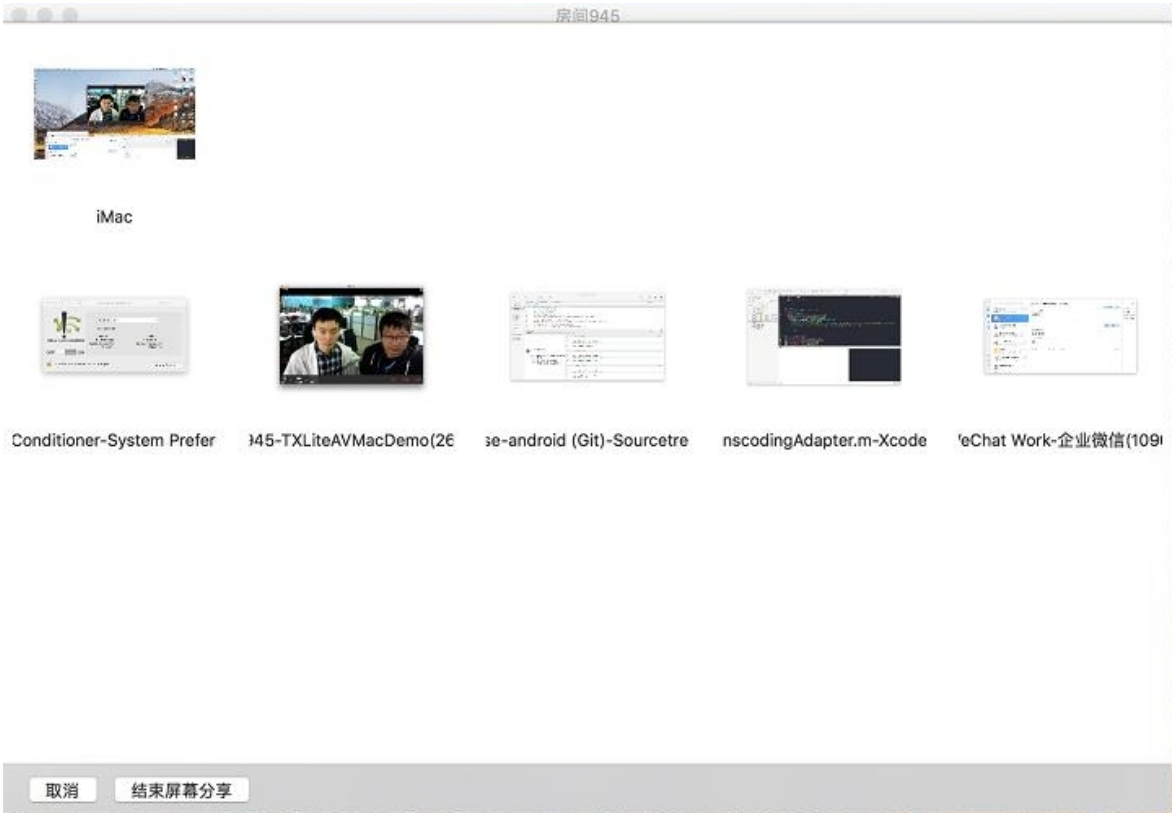
通过 getScreenCaptureSourcesWithThumbnailSize 可以枚举可共享的窗口列表，每一个可共享的目标都是一个 TRTCScreenCaptureSourceInfo 对象。

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

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

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

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



选择分享目标

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：2022-03-15 19:39:53

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, autoSubscribe: false,});

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

```
// 取消订阅自己的屏幕分享流
client.unsubscribe(remoteStream);
} else {
// 订阅其他一般远端流
client.subscribe(remoteStream);
}
});

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

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

// ... 省略初始化和发布相关代码，按需发布流即可。
```

屏幕分享采集系统声音

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

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

在弹出的对话框中勾选 分享音频，发布的 stream 将会带上系统声音。

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：2021-09-14 15:24:27

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。如下图所示：

启动屏幕分享

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

添加 Activity

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

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

设定视频编码参数

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

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

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

弹出悬浮窗以避免被强杀

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

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）。

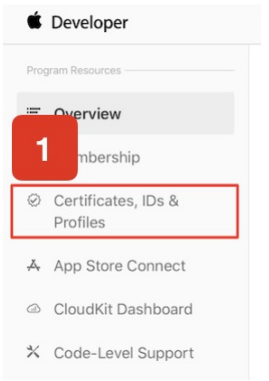
```
# 引入 trtc sdk和replay_kit_launcher
dependencies:
tencent_trtc_cloud: ^0.2.1
replay_kit_launcher: ^0.2.0+1
```

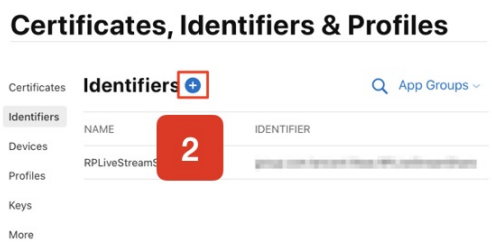
> ⚠ 注意：
> 如果跳过 步骤1，也就是不配置 App Group（接口传 null），屏幕分享依然可以运行，但稳定性要打折扣，故虽然步骤较多，但请尽量配置正确的 App Group 以保障屏幕分享功能的稳定性。

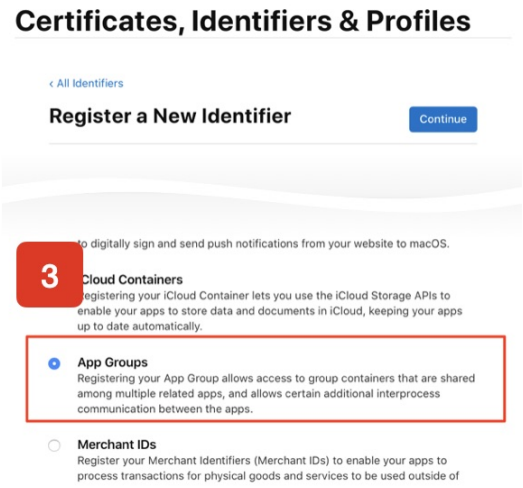
步骤1：创建 App Group

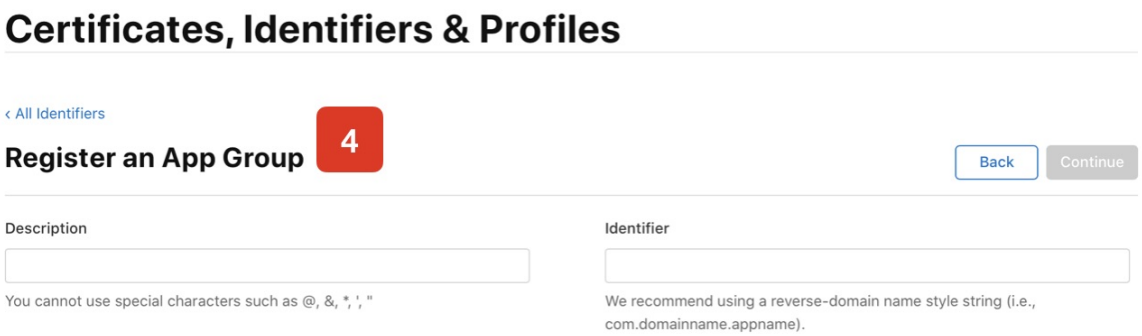
使用您的帐号登录 https://developer.apple.com/，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile。**

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】。

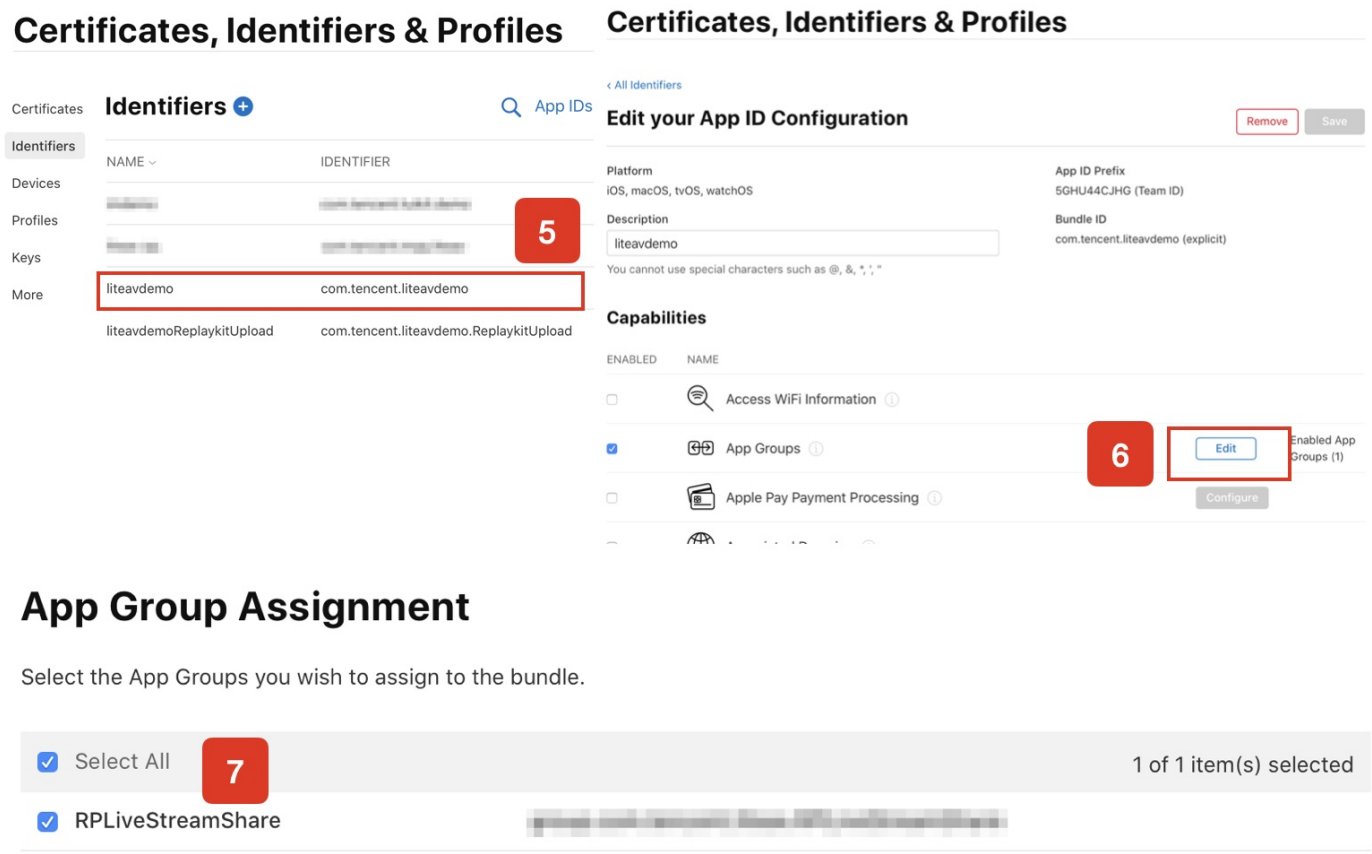








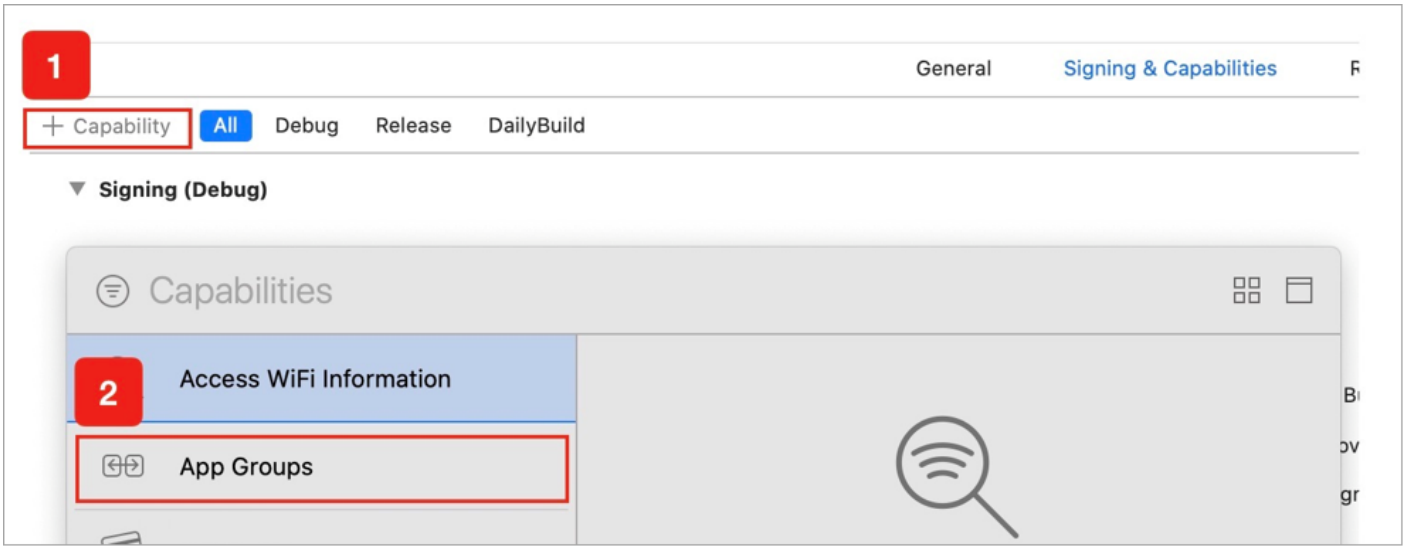
5. 回到 Identifier 页面，左上边的菜单中选择【App IDs】，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中【App Groups】并单击【Edit】。
7. 在弹出的表单中选择您之前创建的 App Group，单击【Continue】返回编辑页，单击【Save】保存。



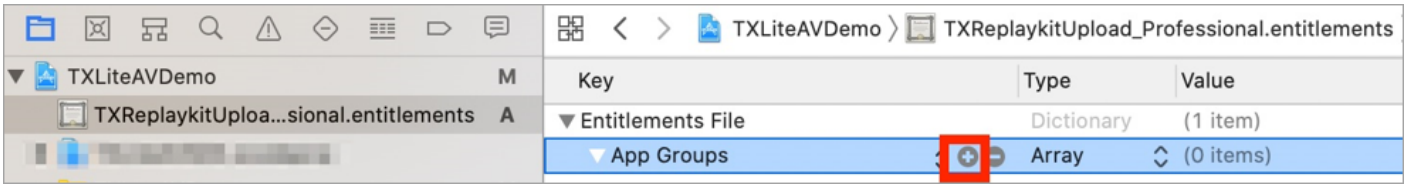
8. 重新下载 Provisioning Profile 并配置到 XCode 中。

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。**

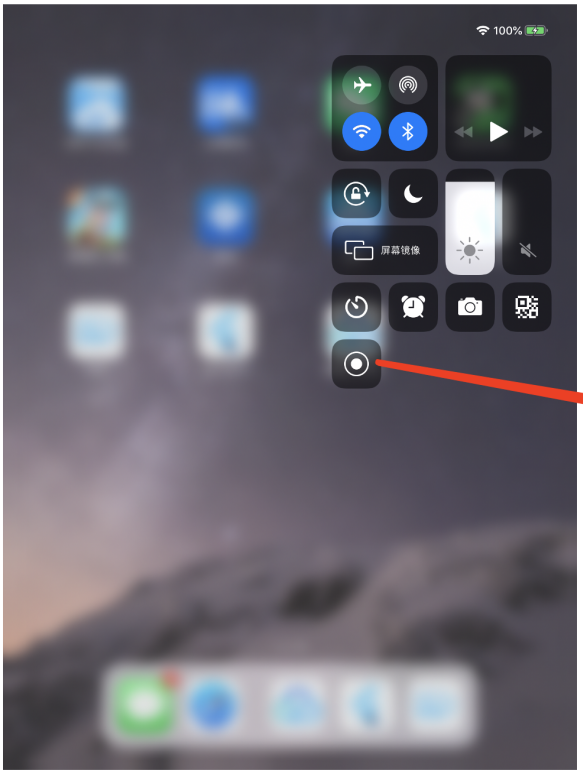
```
import ReplayKit
import TXLiteAVSDK_ReplayKitExt


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: [NSLocalizedFailureReason
ErrorKey:tip])
finishBroadcastWithError(error)
}

override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType: RPSampleBufferTyp
e) {
switch sampleBufferType {
case RPSampleBufferType.video:
// Handle video sample buffer
TXReplayKitExt.sharedInstance() .sendVideoSampleBuffer(sampleBuffer)
break
case RPSampleBufferType.audioApp:
// Handle audio sample buffer for app audio
break
case RPSampleBufferType.audioMic:
// Handle audio sample buffer for mic audio
break
@unknown default:
// Handle other sample buffer types
fatalError("Unknown type of sample buffer")
}
}
}
```

步骤3：对接主 App 端的接收逻辑

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

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



长按“录制”按钮

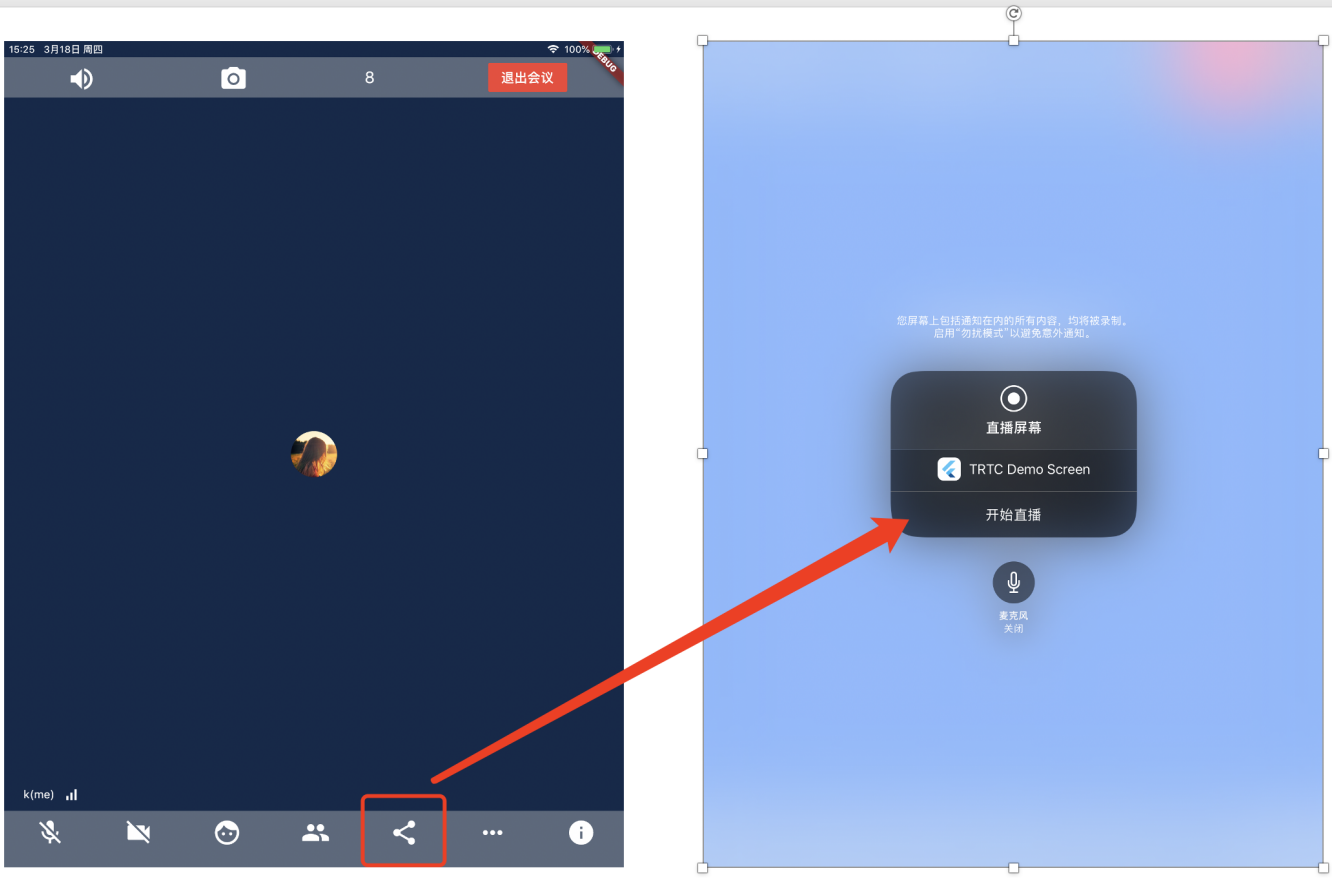
点击开始直播

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

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

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


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

步骤4：增加屏幕分享的触发按钮（可选）

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



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。

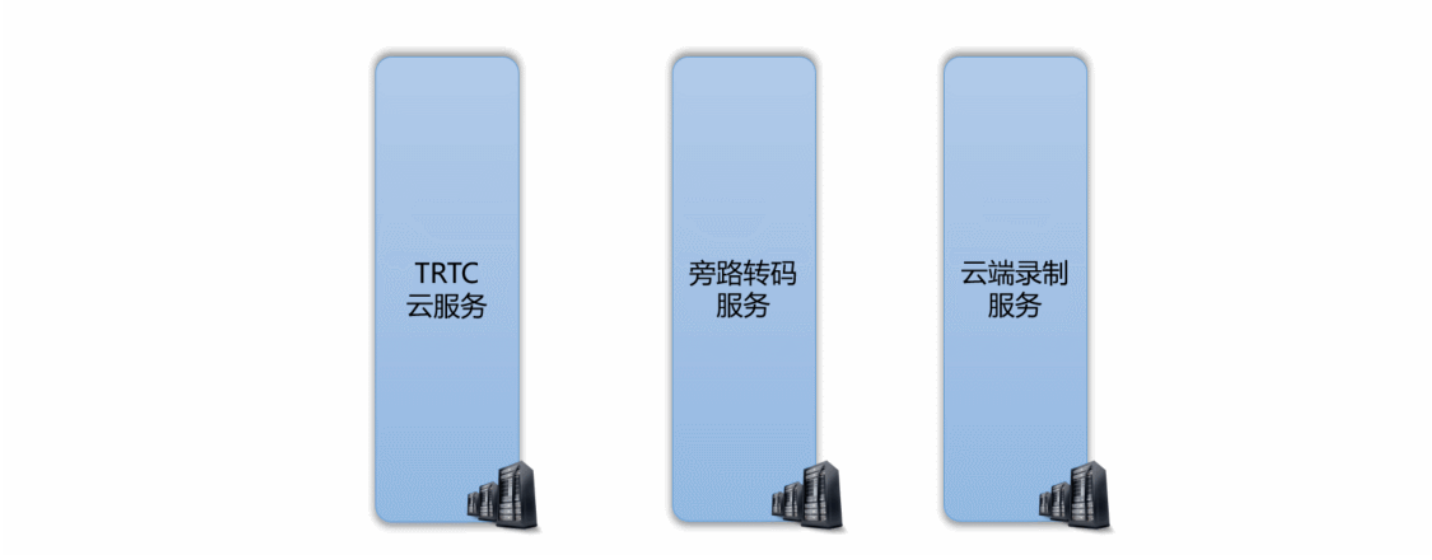
实现云端录制与回放

最近更新时间：2021-09-15 15:11:16

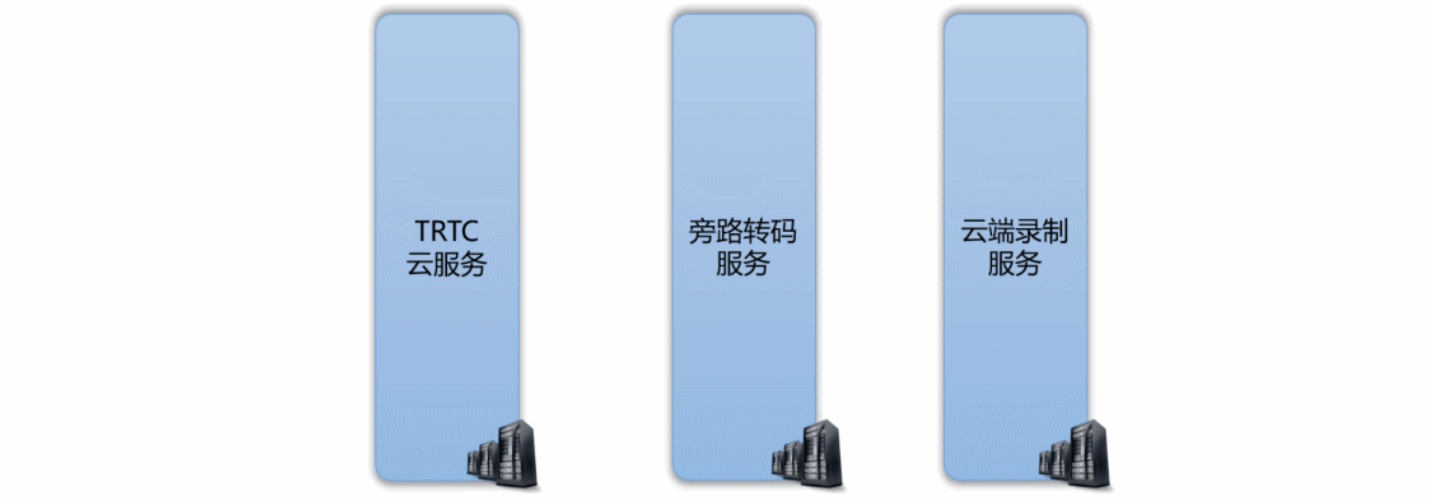
适用场景

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

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



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



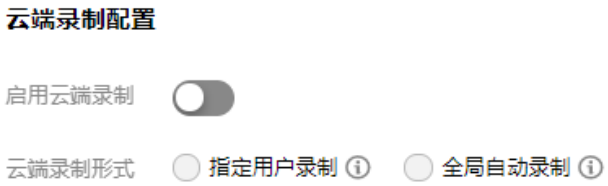
控制台指引

开通录制服务

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

选择录制形式

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



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

选择文件格式

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

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

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

选择存储位置

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

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

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

设置录制回调

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

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

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

录制控制方案

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

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

方案一：全局自动录制

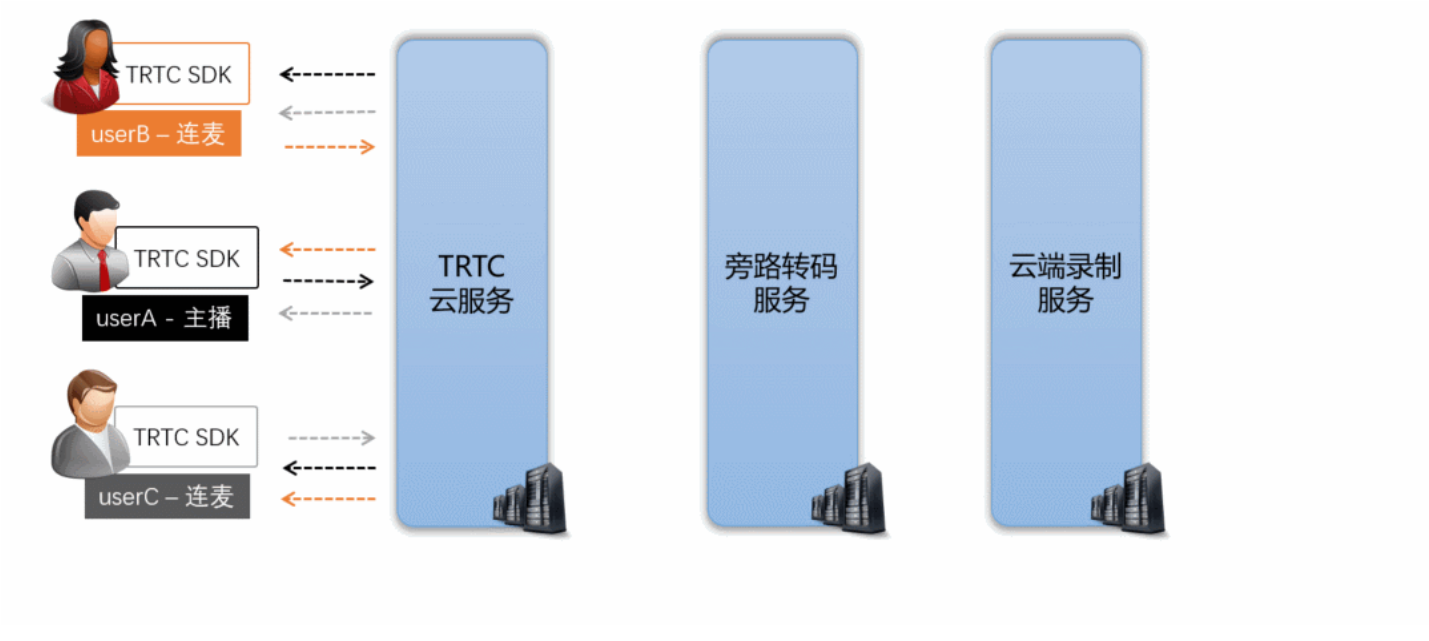
- **控制台中的设定**
  要使用该种录制方案，请在控制台中 选择录制形式 时，设定为“全局自动录制”。

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

方案二：指定用户录制（SDK API）

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

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



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

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

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

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

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

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

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

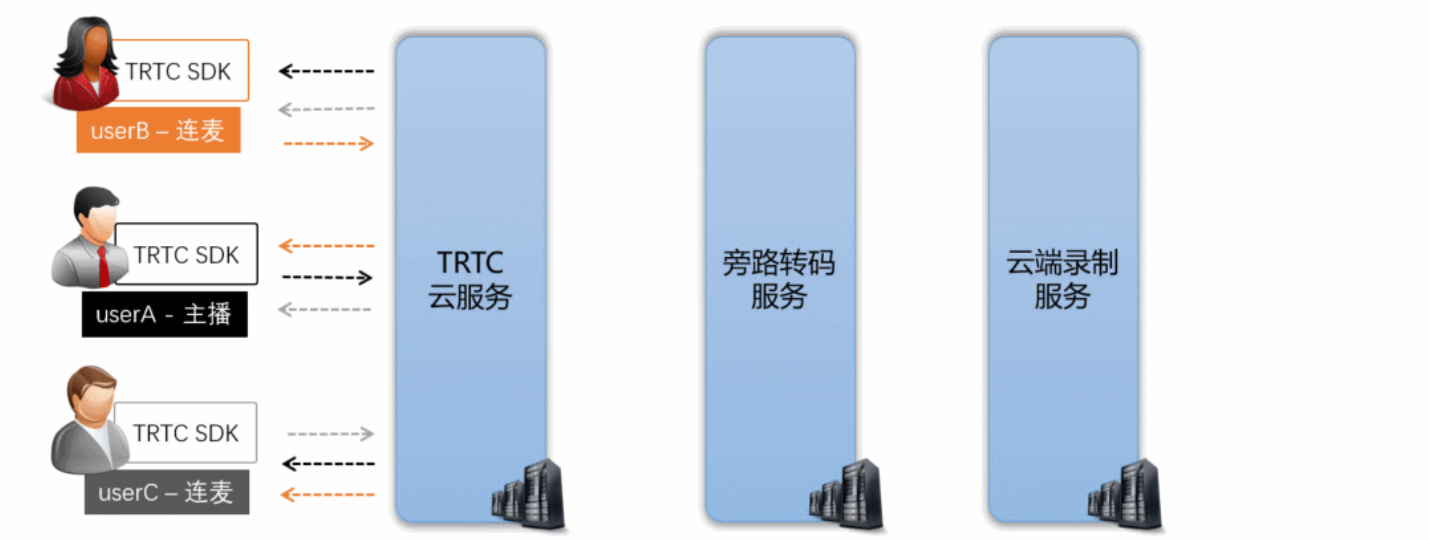
- **已经支持的平台**
  支持 iOS、Android、Windows、Mac、Electron 等终端发起录制控制，暂不支持由 Web 浏览器和微信小程序端发起控制。

方案三：指定用户录制（REST API）

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

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

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



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

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

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

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

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

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

> **说明：**
> 关于云端混流的详细介绍，具体请参见 云端混流转码。

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

- **已经支持的平台**
  由您的服务端控制，不受客户端平台的限制。

查找录制文件

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

方式一：在点播控制台手动查找

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

方式二：通过点播 REST API 查找

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

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

接收录制文件

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

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


```
{
    "app": "8888.livepush.myqcloud.com",
    "appid": 1234567890,
    "appname": "trtc_1400000123",
    "channel_id": "1400000123_1001_rexchang_main",
    "duration": 39,
    "end_time": 1581926501,
    "end_time_usec": 817545,
    "event_type": 100,
    "file_format": "mp4",
    "file_id": "5285890798833426017",
    "file_size": 3337482,
    "media_start_time": 0,
    "record_bps": 0,
    "record_file_id": "5285890798833426017",
    "start_time": 1581926464,
    "start_time_usec": 588890,
    "stream_id": "1400000123_1001_rexchang_main",
    "stream_param": "txSecret=ecd19683f6cfla7615f13670e0834de1&txTime=70d4653d&from=interactive&client_business_type=0&
                     sdkappid=1400000123&sdkapptype=1&groupid=1001&userid=cmV4Y2hhbmc=&ts=5e4a483c&userdefinerecordid=my
                     testfile&tinyid=1441152145405491 89&roomid=2294546&cliRecoId=0&trtcclientip=222.248.237.246&useMixPl
                     ayer=1",
    "task_id": "1802569503626226707",
    "video_id": "1234567890_46ac1271218249118752d57423120300",
    "video_url": "http://1234567890.vod2.myqcloud.com/5022b150vodcq1234567890/39e578f1228079589883342601 7/f0.mp4"
}
```


您可以通过下表中的字段来确定当前回调是对应的哪一次通话（或直播）：

序号	字段名	说明

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

> 说明：
> 更多回调字段说明，请参见 云直播-录制事件通知。

删除录制文件

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

```
https://vod.tencentcloudapi.com/?Action=DeleteMedia
&FileId=5285890798866415058 7
&<公共请求参数>
```

回放录制文件

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

选择文件格式（HLS）

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

获取点播地址（video_url）

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

对接点播播放器

根据使用平台对接点播播放器，具体操作参考如下：

- iOS 平台
- Android 平台
- Web 浏览器

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

相关费用

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

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

录制费用：转码或转封装产生的计算费用

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

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

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

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

存储费用：如将文件存储于腾讯云则会产生该费用

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

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

观看费用：如将文件用于点播观看则会产生该费用

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

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

转码费用：如开启混流录制则会产生该费用

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

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

相关问题

实时音视频如何实现服务端录制?

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

实现 CDN 直播观看

最近更新时间：2022-01-17 15:37:21

适用场景

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

给 TRTC 对接 CDN 观看，一般被用于解决如下两类问题：

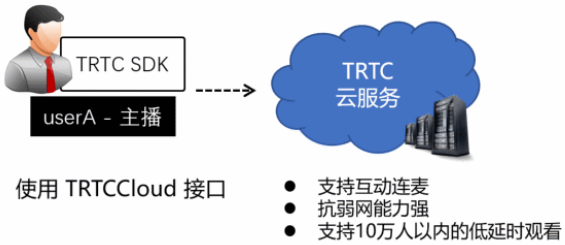
- **问题一：超高并发观看**
  TRTC 的低延时观看能力，单房间支持的最大人数上限为10万人。CDN 观看虽然延迟要高一些，但支持10万人以上的并发观看，且 CDN 的计费价格更加便宜。
- **问题二：移动端网页播放**
  TRTC 虽然支持 WebRTC 协议接入，但主要用于 Chrome 桌面版浏览器，移动端浏览器的兼容性非常不理想，尤其是 Android 手机浏览器对 WebRTC 的支持普遍都很差。所以如果希望通过 Web 页面在移动端分享直播内容，还是推荐使用 HLS(m3u8) 播放协议，这也就需要借助直播 CDN 的能力来支持 HLS 协议。

原理解析

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

单路画面的旁路直播

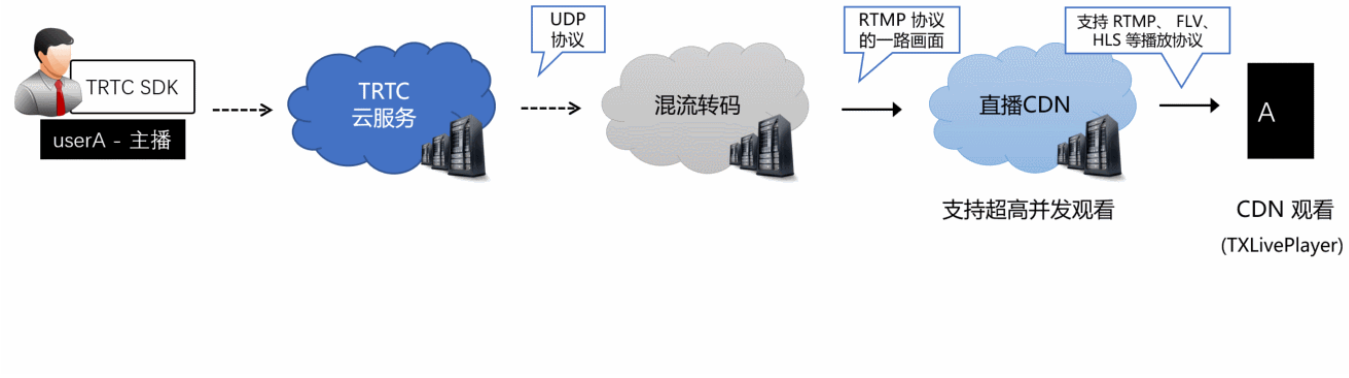
当 TRTC 房间中只有一个主播时，TRTC 的旁路推流跟标准的 RTMP 协议直推功能相同，不过 TRTC 的 UDP 相比于 RTMP 有更强大的弱网络抗性。



混合画面的旁路直播

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

混流服务 将多路画面合并成一路，其原理如下图所示：



说明：

为什么不直接播放多路 CDN 画面?

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

前提条件

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

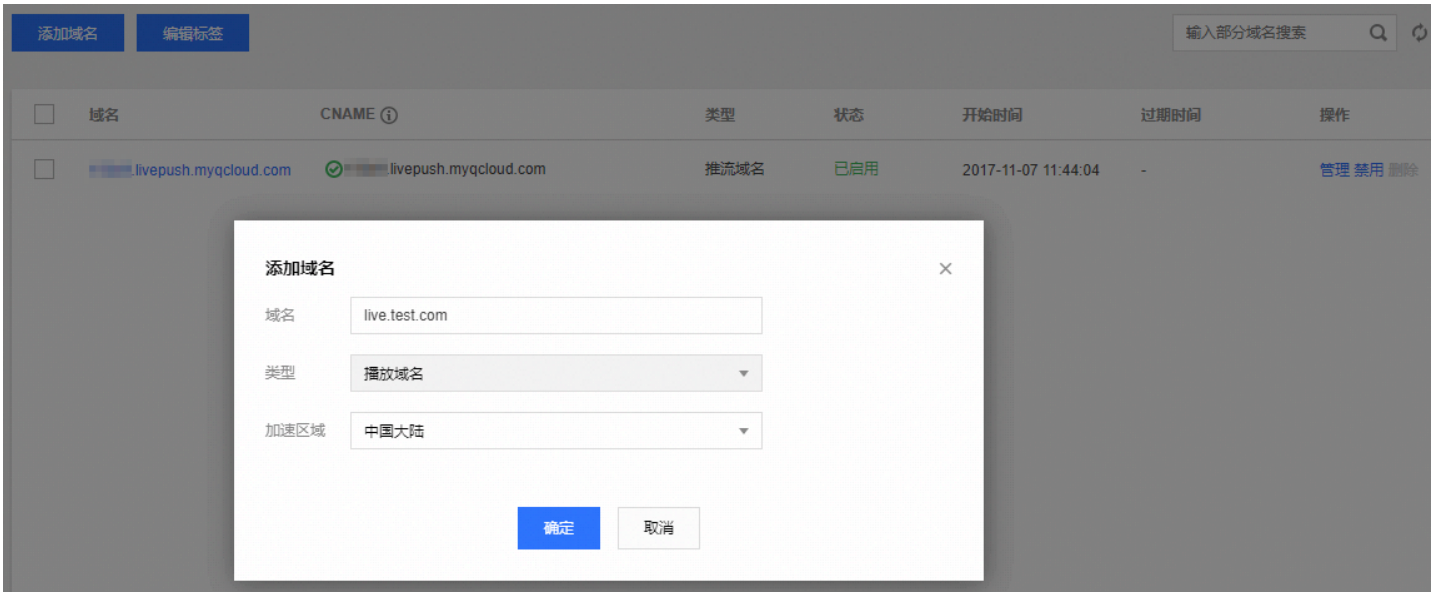
使用步骤

步骤1：开启旁路推流功能

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

步骤2：配置播放域名并完成 CNAME

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



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

步骤3：关联 TRTC 的音视频流到直播 streamId

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

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

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

方式一：自定义指定 streamId

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

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

userSig 的计算方法请参见 如何计算及使用 UserSig。

方式二：系统指定 streamId

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

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

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

步骤4：控制多路画面的混合方案

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

- 各个子画面的摆放位置和大小。
- 混合画面的画面质量和编码参数。

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

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

步骤5：获取播放地址并对接播放

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

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

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

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

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

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

所属平台	对接文档	API 概览	支持的格式
iOS App	接入指引	V2TXLivePlayer(iOS)	推荐 FLV
Android App	接入指引	V2TXLivePlayer(Android)	推荐 FLV
Web 浏览器	接入指引	–	• 桌面端 Chrome 浏览器支持 FLV • Mac 端 Safari 和移动端手机浏览器仅支持 HLS
微信小程序	接入指引	<live-player> 标签	推荐 FLV

步骤6：优化播放延时

开启旁路直播后的 http – flv 地址，由于经过了直播 CDN 的扩散和分发，观看时延肯定要比直接在 TRTC 直播间里的通话时延要高。
按照目前腾讯云的直播 CDN 技术，如果配合 V2TXLivePlayer 播放器，可以达到下表中的延时标准：

旁路流类型	V2TXLivePlayer 的播放模式	平均延时	实测效果
独立画面	极速模式（推荐）	2s – 3s	下图中左侧对比图（橙色）
混合画面	极速模式（推荐）	4s – 5s	下图中右侧对比图（蓝色）

下图中的实测效果，采用了同样的一组手机，左侧 iPhone 6s 使用了 TRTC SDK 进行直播，右侧的小米6 使用 V2TXLivePlayer 播放器播放 FLV 协议的直播流。

如果您在实测中延时比上表中的更大，可以按照如下指引优化延时：

- **使用 TRTC SDK 自带的 V2TXLivePlayer**
  普通的 ijkplayer 或者 ffmpeg 基于 ffmpeg 的内核包装出的播放器，缺乏延时调控的能力，如果使用该类播放器播放上述直播流地址，时延一般不可控。V2TXLivePlayer 有一个自研的播放引擎，具备延时调控的能力。

- **设置 V2TXLivePlayer 的播放模式为极速模式**
  可以通过设置 V2TXLivePlayer 的参数来实现极速模式，以 iOS 为例：

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

//设置完成之后再启动播放
```

相关费用

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

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

观看费用：通过直播 CDN 观看时产生的费用

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

转码费用：启用多路画面混合时收取

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

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

常见问题

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