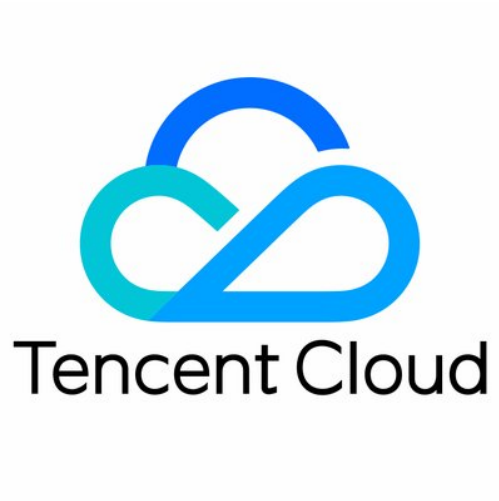
Tencent Real-Time Communication

ベストプラクティス

製品ドキュメント



カタログ：

ベストプラクティス
通話モード
通話モードクイックスタート（iOS&Mac）

最終更新日：：2022-03-09 16:44:11

ユースケース

TRTCは、4種類の異なる入室モードをサポートしています。このうち、ビデオ通話（VideoCall）および音声通話（AudioCall）を総称して通話モードといい、ビデオインタラクティブストリーミング（Live）およびボイスインタラクティブストリーミング（VoiceChatRoom）を総称して ライブストリーミングモードといいます。
通話モードでのTRTCは、1つのルームに最大で300人の同時オンラインをサポートし最大で50人の同時発言をサポートします。1対1のビデオ通話、300人のビデオミーティング、オンライン問診、リモート面接、ビデオカスタマーサービス、オンライン人狼ゲームなどのユースケースに適合しています。

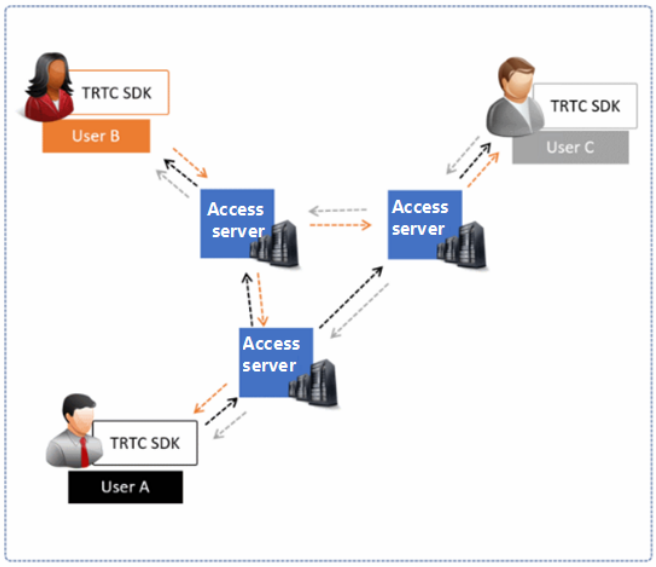
原理解析

TRTCクラウドサービスは、「インターフェースモジュール」および「プロキシモジュール」という2種類の異なるタイプのサーバーノードから構成されています。

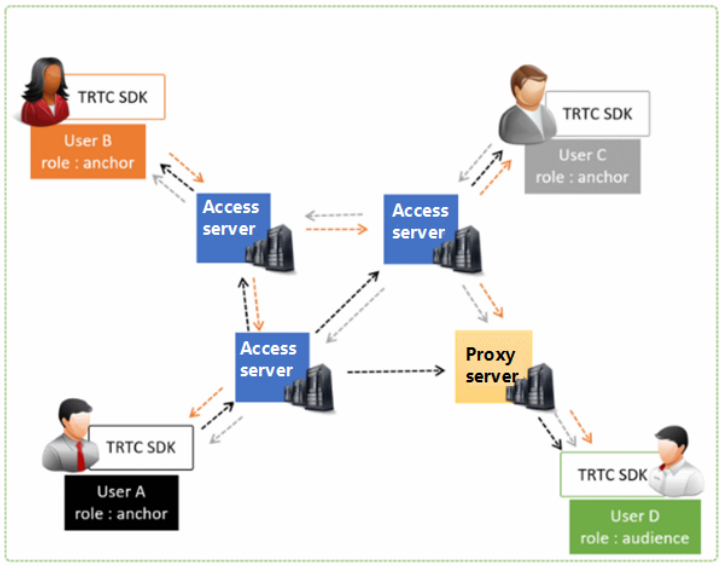
- **インターフェースモジュール**
  この種のノードは、最も良質の回線および高性能の機器を採用しており、エンドツーエンドの低遅延マイク接続通話の処理に優れますが、単位時間当たりの費用はやや高くなります
- **プロキシモジュール**
  この種のノードは、通常の回線および性能も一般的な機器を採用しており、同時進行性の高いプルストリーミング再生のニーズの処理にすぐれ、単位当たりの費用はやや低くなります。

通話モードでは、TRTCルームのすべてのユーザーはインターフェースモジュールに割り当てられます。各ユーザーは「キャスター」に相当し、各ユーザーは随時発言でき（同時アップストリームの最大制限は50）、このため

オンラインミーティングなどのユースケースに適していますが、1つのルームの人数制限は300人になります。



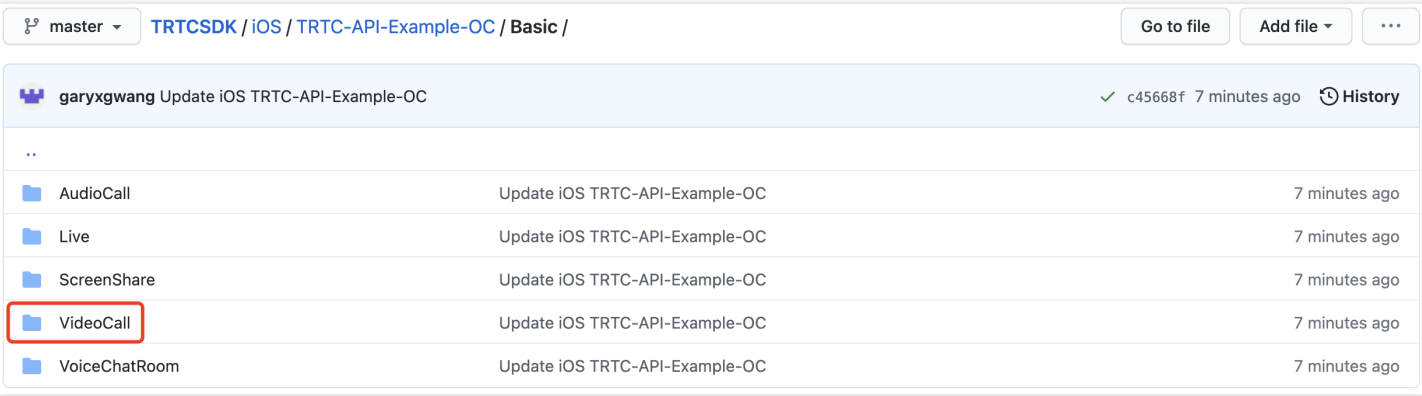
Call mode (no role differentiation)



Live streaming mode (two roles: anchor and viewer)

サンプルコード

Github にログインし、本ファイルに関連するサンプルコードを取得することができます。



Githubへのアクセスが遅い場合は、TXLiteAVSDK_TRTC_iOS_latest.zipを直接ダウンロードすることもできます。

操作手順

手順1：SDKの統合

以下の方式を選択して **TRTC SDK** をプロジェクトに統合することができます。

方法1： CocoaPodsを使用して統合

1. **CocoaPods**をインストールします。具体的な操作は CocoaPods公式サイトインストールの説明をご参照ください。

2. 現在のプロジェクトのルートディレクトリの `Podfile` ファイルを開き、以下のコンテンツを追加します。

> 説明：
> このディレクトリに `Podfile` ファイルがない場合は、まず `pod init` コマンドを実行しファイルを新規作成してから、以下の内容を追加してください。

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

3. 以下のコマンドを実行して **TRTC SDK** をインストールします。

```
pod install
```

インストールが成功したら、現在のプロジェクトのルートディレクトリに **xcworkspace** ファイルが生成されます。
4. 新規作成した **xcworkspace** ファイルを開けばOKです。

方法2：ZIPパッケージをダウンロードして手動で統合

一時的にCocoaPods環境をインストールしたくない場合、またはインストール済みだがCocoaPodsライブラリへのアクセスがやや遅い場合は、 ZIP圧縮パッケージを直接ダウンロードして、クイックインテグレーション(iOS)を参照してSDKをプロジェクトに統合することができます。

手順2：メディアデバイスの権限追加

`Info.plist` ファイルにカメラおよびマイクのアクセス許可のリクエストを追加します。

Key	Value
Privacy - Camera Usage Description	カメラ使用の許可をリクエストする理由を記述。例えば、ビデオチャットでビデオを表示するには、カメラへのアクセスが必要です

Key	Value
Privacy - Microphone Usage Description	マイク使用の許可をリクエストする理由を記述。例えば、チャットで音声を送信するには、マイクへのアクセスが必要です

手順3：SDKインスタンスを初期化し、イベントコールバックを監視する

1. sharedInstance()インターフェースを使用して、 `TRTCCloud` インスタンスを作成します。

```
// trtcCloudインスタンスを作成
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;
```

2. `delegate` 属性を設定しイベントのコールバックを登録し、関連イベントおよびエラー通知をモニタします。

```
// エラー通知は監視すべきもので、捕捉してユーザーに通知する必要があります
- (void)onError:(TXLiteAVError)errCode errMsg:(NSString *)errMsg extInfo:(NSDictionary *)extIn
fo {
if (ERR_ROOM_ENTER_FAIL == errCode) {
[self toastTip:@"入室失敗"];
[self.trtcCloud exitRoom];
}
}
```

手順4：入室パラメータTRTCParamsを組み立てる

enterRoom()インターフェースを呼び出す時に、キーパラメータTRTCParamsを入力する必要があります。このパラメータに含まれる入力必須のフィールドは下表に示すとおりです。

パラメータ名	フィールドタイプ	補足説明	記入例
sdkAppId	数字	アプリケーションID。TRTCコンソールでSDKAppIDを表示できます。	1400000123
userId	文字列	アルファベットの大文字、小文字（a-z、A-Z）、数字（0-9）、下線およびハイフンのみを許可。ビジネスの実際のアカウントシステムを組み合わせて設定することをお勧めします。	test_user_001

パラメータ名	フィールドタイプ	補足説明	記入例
userSig	文字列	userIdを基にuserSigを計算します。計算方法はUserSigの計算方法をご参照ください。	eJyrVareCeYrSy1SsII...
roomId	数字	数字タイプのルームナンバー。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。	29834

注意：
TRTCは、相互に干渉しないよう、2つの同じuserIdによる同時入室をサポートしていません。

手順5：ルームの新規作成および入室

1. enterRoom()を呼び出せば、TRTCParamsパラメータの `roomId` が示すオーディオ・ビデオルームに入室できます。該当するルームが存在しない場合は、SDKがフィールド `roomId` の値をルームナンバーとする新しいルームを自動的に作成します。
2. ユースケースに基づき適切な** `appScene` **パラメータを設定してください。誤った選択をすると、ラグ率または画面の解像度が想定レベルに到達しなくなります。

- ビデオ通話は、 `TRTCAppScene.videoCall` に設定してください。
- 音声通話は、 `TRTCAppScene.audioCall` に設定してください。

3. 入室に成功したら、SDKは `onEnterRoom(result)` イベントをコールバックします。そのうち、パラメータ `result` が0より大きいときは入室成功を示し、具体的な数値は入室のために消費した時間になります。単位はミリ秒（ms）です。 `result` が0より小さいときは入室失敗を示し、具体的な数値は入室失敗のエラーコードになります。

```
- (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:@"入室成功"];
}else{
[self toastTip:@"入室失敗"];
}
}
```

注意：

- 入室に失敗した場合は、SDKは同時に `onError` イベントもコールバックし、パラメータ `errCode` （エラーコード）、 `errMsg` （エラー原因）および `extraInfo` （保留パラメータ）を返します。
- すでに特定のルームにいる場合は、まず `exitRoom()` を呼び出して現在のルームを退出すると、もう一つのルームに入ることができるようになります。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

手順6：リモート側のオーディオビデオストリーミングの閲覧

SDKは自動閲覧および手動閲覧をサポートします。

自動閲覧モード（デフォルト）

自動閲覧モードでは、特定のルームに入った後、SDKはルームのその他のユーザーのオーディオストリームを自動で受信します。これによって最適な「インスタントブロードキャスティング効果が得られます。

1. ルーム内の他のユーザーがオーディオデータをアップストリームすると、onUserAudioAvailable()のイベント通知を受信して、SDKがそのリモートユーザーの音声を自動再生します。
2. muteRemoteAudio(userId, mute: true)によって、特定のuserIdのオーディオデータを遮断することができ、muteAllRemoteAudio(true)によって、すべてのリモートユーザーのオーディオデータを遮断することもできます。遮断後、SDKは、当該リモートユーザーのオーディオデータをそれ以降プルしなくなります。
3. ルームのその他のユーザーがビデオデータをアップストリームすると、onUserVideoAvailable()のイベント通知を受信しますが、このとき、SDKは、ビデオデータをどのように表示するかの指令を受け取っていないため、ビデオデータを自動処理しません。startRemoteView(userId, view: view)メソッドを呼び出し、リモートユーザーのビデオデータと `view` （表示）をバインドする必要があります。
4. setRemoteViewFillModeによって、ビデオ画面の表示モードを指定することができます。

- Fillモード：塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
- Fitモード：適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

5. stopRemoteView(userId)によって、特定のuserIdのビデオデータを遮断することができ、また stopAllRemoteView()によって、すべてのリモートユーザーのビデオデータを遮断することもできます。遮断後、SDKは、当該リモートユーザーのビデオデータをそれ以降プルしなくなります。

```
// インスタンスコード：通知に基づきリモート側ユーザーのビデオ画面を閲覧（または閲覧の取り消し）します。
- (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];
}
}
```

説明：
`onUserVideoAvailable()` イベントのコールバック受信した後、すぐに `startRemoteView()` を呼び出してビデオストリームを閲覧しない場合、SDKが5s以内にリモートからのビデオデータの受信を停止します。

手動閲覧モード

setDefaultStreamRecvMode()インターフェースによって、SDKを手動閲覧モードに指定できます。手動閲覧モードでは、SDKはルーム内の他のユーザーの音声・ビデオデータを自動受信しません。手動で、API関数を介してトリガーする必要があります。

1. **入室前**に、setDefaultStreamRecvMode(false, video: false)インターフェースを呼び出して、SDKを手動閲覧モードに設定します。
2. ルーム内の他のユーザーがオーディオデータをアップストリームすると、onUserAudioAvailable()のイベント通知を受信します。この時、muteRemoteAudio(userId, mute: false)メソッドを呼び出すことで、そのユーザーのオーディオデータを手動で閲覧する必要があります。SDKはそのユーザーのオーディオデータを受信した後、デコードして再生します。
3. ルーム内の他のユーザーがビデオデータをアップストリームすると、onUserVideoAvailable()イベントの通知を受信します。この時は、startRemoteView(userId, view: view)メソッドを呼び出すことで、そのユーザーのビデオデータを手動で閲覧する必要があります。SDKが、そのユーザーのビデオデータを受信後、デコードして再生します。

手順7：ローカルのオーディオビデオストリーミングの公開

1. startLocalAudio()を呼び出すと、ローカルのマイク集音を起動させ、採集した音声をエンコードして送信することができます。

2. startLocalPreview()を呼び出すと、ローカルのカメラを起動させ、キャプチャした画面をエンコードして送信することができます。
3. setLocalViewFillMode()を呼び出すと、ローカルのビデオ画面の表示モードを設定することができます。

- Fillモードは塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
- Fitモードは適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

4. setVideoEncoderParam()インターフェースを呼び出すと、ローカルビデオのエンコードパラメータを設定できます。このパラメータにより、ルーム内の他のユーザーが視聴する際の画面の画質が決定されます。

```
//サンプルコード：ローカルのオーディオ・ビデオストリーミングの公開
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

注意：
Mac版SDKは、デフォルトでは、現在のシステムのデフォルトのカメラおよびマイクを使用します。setCurrentCameraDevice()およびsetCurrentMicDevice()を呼び出すことで、その他のカメラおよびマイクを選択することができます。

手順8：現在のルームから退出する

exitRoom()メソッドを呼び出してルームを退出します。SDKは退室する時に、カメラ、マイクなどのハードウェアデバイスを停止してリリースする必要があるたため、退室の動作は瞬時に完了するものではなく、onExitRoom()のコールバックを受信してはじめて、実際の退室操作が完了します。

```
// 退室を呼び出した後は、onExitRoomイベントのコールバックをお待ちください
[self.trtcCloud exitRoom];

- (void)onExitRoom:(NSInteger)reason {
NSLog(@"ルームから退出: reason: %ld", reason)
}
```

注意：
Appの中で多くの音声ビデオのSDKを同時に統合した場合は、`onExitRoom`コールバックを受信してからその他の音声ビデオSDKを起動してください。そうしない場合は、ハード上の占有問題が生じることがあ

ります。

通話モードのクイックスタート(Android)

最終更新日：：2022-03-09 16:35:31

ユースケース

TRTCは、4種類の異なる入室モードをサポートしています。このうち、ビデオ通話（VideoCall）および音声通話（AudioCall）を総称して通話モードといい、ビデオインタラクティブストリーミング（Live）およびボイスインタラクティブストリーミング（VoiceChatRoom）を総称して ライブストリーミングモードといいます。
通話モードでのTRTCは、1つのルームに最大で300人の同時オンラインをサポートし最大で50人の同時発言をサポートします。1対1のビデオ通話、300人のビデオミーティング、オンライン問診、リモート面接、ビデオカスタマーサービス、オンライン人狼ゲームなどのユースケースに適合しています。

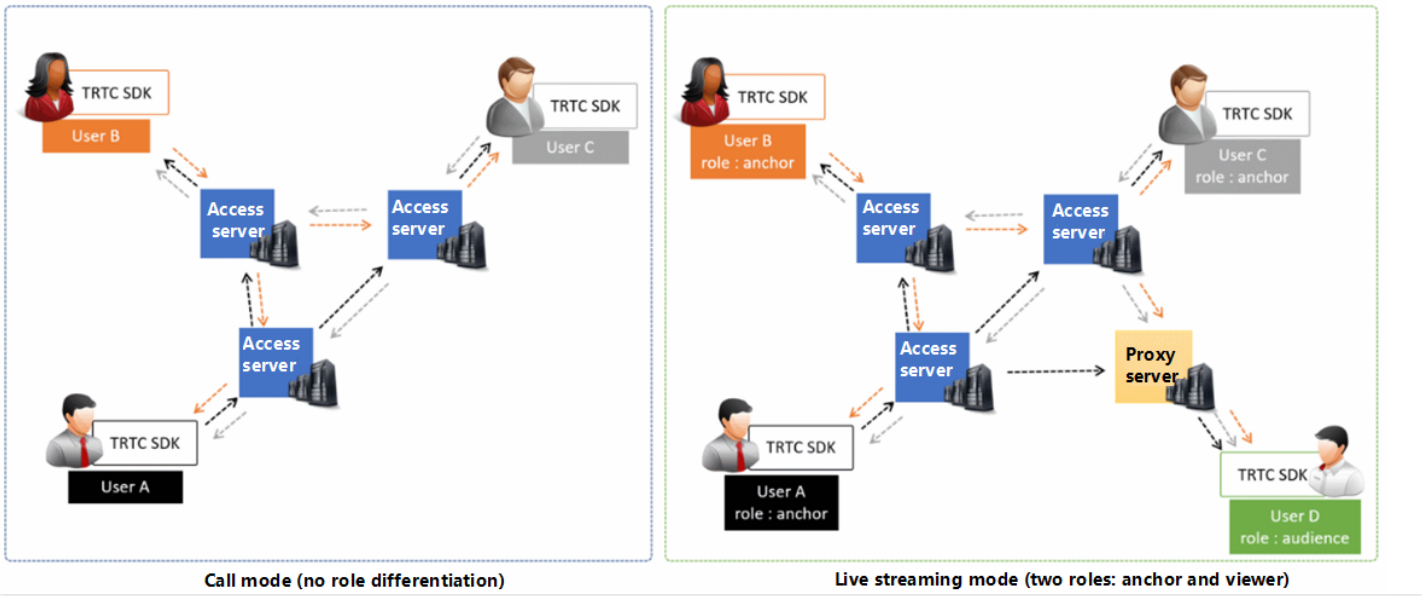
原理解析

TRTCクラウドサービスは、「インターフェースモジュール」および「プロキシモジュール」という2種類の異なるタイプのサーバーノードから構成されています。

- **インターフェースモジュール**
  この種のノードは、最も良質の回線および高性能の機器を採用しており、エンドツーエンドの低遅延マイク接続通話の処理に優れますが、単位時間当たりの費用はやや高くなります。
- **プロキシモジュール**
  この種のノードは、通常の回線および性能も一般的な機器を採用しており、同時進行性の高いプルストリーミング再生のニーズの処理にすぐれ、単位当たりの費用はやや低くなります。

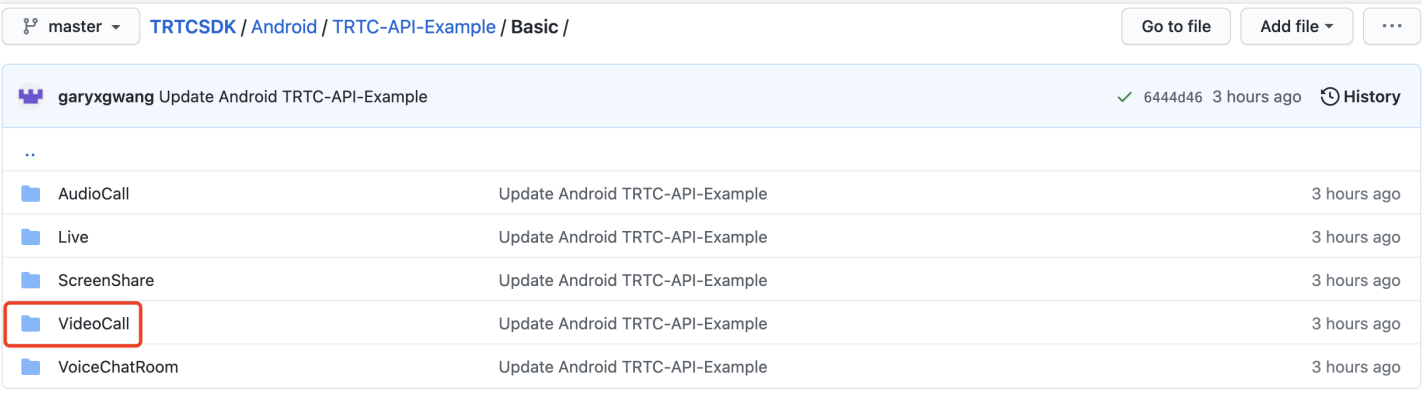
通話モードでは、TRTCルームのすべてのユーザーはインターフェースモジュールに割り当てられます。各ユーザーは「キャスター」に相当し、各ユーザーは随時発言でき（同時アップストリームの最大制限は50）、このため

オンラインミーティングなどのユースケースに適していますが、1つのルームの人数制限は300人になります。



サンプルコード

Github にログインし、本ファイルに関連するサンプルコードを取得することができます。



説明：
Githubへのアクセスが遅い場合は、 TXLiteAVSDK_TRTC_Android_latest.zipを直接ダウンロードすることもできます。

操作手順

手順1：SDKの統合

以下の方式を選択して **TRTC SDK** をプロジェクトに統合することができます。

方法1：自動ロード（aar）

TRTC SDKは、mavenCentral ライブラリにリリースされています。更新を自動的にダウンロードするようにgradleを構成することで自動でダウンロード、更新できます。
Android Studioを使用して、SDKを統合予定のプロジェクト（TRTC-API-Exampleは統合が完了済み、サンプルコードは参照用として提供）を開き、その後簡単な手順で `app/build.gradle` ファイルを修正するだけで、SDKの統合を完了できます。

1. dependenciesの中にTRTCSDKの依存を追加します。

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

2. defaultConfigでAppが使用するCPUアーキテクチャを指定します。

> 説明：
> 現在 TRTC SDKは、armeabi、armeabi-v7a、arm64-v8aをサポートしています。

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

3. 【Sync Now】をクリックし、SDKを同期します。
   mavenCentralへのネットワーク接続に問題がない場合、SDKは自動的にダウンロードされ、プロジェクトに統合されます。

方法2：ZIPパッケージをダウンロードして手動で統合

ZIP圧縮パッケージを直接ダウンロードして、 クイックインテグレーション(Android)を参照してSDKをプロジェクトに統合することができます。

手順2： App権限の設定

`AndroidManifest.xml` ファイルにカメラ、マイクおよびネットワークのアクセス許可のリクエストを追加します。

```
<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" />
```

手順3：SDKインスタンスを初期化し、イベントコールバックを監視する

1. sharedInstance() インターフェースを使用して `TRTCCloud` インスタンスを作成します。

```
// trtcCloudインスタンスを作成
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener(){
// コールバック処理
...
});
```

2. `setListener` 属性を設定しイベントのコールバックを登録し、関連イベントおよびエラー通知をモニタします。

```
// エラー通知のモニタ。エラー通知は、 SDK が動作を継続できないことを示します
@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();
}
}
}
```

手順4：入室パラメータTRTCParamsを組み立てる

enterRoom()インターフェースを呼び出すとき、キーパラメータTRTCParamsを入力する必要があります。このパラメータに含まれる入力必須のフィールドは下表に示すとおりです。

パラメータ名	フィールドタイプ	補足説明	記入例
sdkAppId	数字	アプリケーションID。TRTCコンソールでSDKAppIDを表示できます。	1400000123
userId	文字列	アルファベットの大文字、小文字（a-z、A-Z）、数字（0-9）、下線およびハイフンのみを許可。ビジネスの実際のアカウントシステムを組み合わせて設定することをお勧めします。	test_user_001
userSig	文字列	userIdを基にuserSigは計算されます。計算方法はUserSigの計算方法 をご参照ください。	eJyrVareCeYrSy1SsII...
roomId	数字	数字タイプのルームナンバー。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。	29834

注意：
TRTCは、2つの同じuserIdによる同時入室をサポートしていません。同時に入室した場合は相互に干渉します。

手順5：ルームの新規作成および入室

1. enterRoom() を呼び出せば、TRTCParamsパラメータの `roomId` が示すオーディオビデオルームに入室できます。該当するルームが存在しない場合は、SDKがフィールド `roomId` の値をルームナンバーとする新しいルームを自動的に作成します。
2. ユースケースに基づき適切な** `appScene` **パラメータを設定してください。誤った選択をすると、ラグ率または画面の解像度が想定レベルに到達しなくなります。

- ビデオ通話は、 `TRTC_APP_SCENE_VIDEOCALL` と設定してください。
- 音声通話は、 `TRTC_APP_SCENE_AUDIOCALL` に設定してください。

3. 入室に成功したら、SDKは `onEnterRoom(result)` イベントをコールバックします。そのうち、パラメータ `result` が0より大きいときは入室成功を示し、具体的な数値は入室のために消費した時間になります。単位

はミリ秒（ms）です。 `result` が0より小さいときは入室失敗を示し、具体的な数値は入室失敗のエラーコードになります。

```
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("入室成功，総消費時間[\(result)]ms")
}else{
toastTip("入室失敗，エラーコード[\(result)]")
}
}
```

注意：

- 入室に失敗した場合は、SDKは同時に `onError` イベントもコールバックし、パラメータ `errCode` （エラーコード）、 `errMsg` （エラー原因）および `extraInfo` （保留パラメータ）を返します。
- すでに特定のルームにいる場合は、まず `exitRoom()` を呼び出して現在のルームを退出すると、もう一つのルームに入ることができるようになります。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

手順6：リモート側のオーディオビデオストリーミングの閲覧

SDKは自動閲覧および手動閲覧をサポートします。

自動閲覧モード（デフォルト）

自動閲覧モードでは、特定のルームに入った後、SDKはルームのその他のユーザーのオーディオストリームを自動で受信します。これによって最適な「秒速開始」効果が得られます。

1. ルームのその他のユーザーがオーディオデータをアップストリームするとき、onUserAudioAvailable()イベントの通知を受信します。SDK はリモート側ユーザーの音声を自動再生します。
2. muteRemoteAudio(userId, true) によって、特定のuserIdのオーディオデータを遮断することができ、muteAllRemoteAudio(true)によって、すべてのリモートユーザーのオーディオデータを遮断することもでき

ます。遮断後、SDKは、当該リモートユーザーのオーディオデータをそれ以降プルしなくなります。

3. ルームのその他のユーザーがビデオデータを上りにするとき、 onUserVideoAvailable() インシデント通知を受信します。しかし、このとき SDKはビデオデータをどのように表示するか指令を受け取っていないため、ビデオデータを自動処理することはありません。 startRemoteView(userId, view) 方法をコールしてリモート側ユーザーのビデオデータおよび `view` 表示をバインドする必要があります。
4. setRemoteViewFillMode() を介してビデオ画面の表示モードを指定できます。

- Fillモード：塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
- Fitモード：適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

5. stopRemoteView(userId)を介して、特定のuserIdのビデオデータを遮断することができ、また stopAllRemoteView() を介して、すべてのリモートユーザーのビデオデータを遮断することもできます。遮断後、SDKは当該リモートユーザーのビデオデータをプルしなくなります。

```
@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);
}
}
```

> 説明：
> `onUserVideoAvailable()` イベントのコールバックを受信した後、すぐに `startRemoteView()` を呼び出してビデオストリームを閲覧しない場合、SDKは5s以内にリモートからのビデオデータの受信を停止します。

手動閲覧モード

setDefaultStreamRecvMode()インターフェースを介して、SDKを手動閲覧モードに指定できます。手動閲覧モードでは、SDKはルーム内の他のユーザーの音声・ビデオデータを自動受信しません。手動で、API関数を介してトリガーする必要があります。

1. **入室前**にsetDefaultStreamRecvMode(false, false)インターフェースを呼び出し、SDKを手動閲覧モードに設定します。
2. ルーム内の他のユーザーがオーディオデータをアップストリームすると、onUserAudioAvailable() のイベント通知を受信します。この時にmuteRemoteAudio(userId, false)を呼び出し、そのユーザーのオーディオデータを手動で閲覧する必要があります。SDKはそのユーザーのオーディオデータを受信した後、デコードして再生します。
3. ルーム内の他のユーザーがビデオデータをアップストリームすると、onUserVideoAvailable()のイベント通知を受信します。この時に、startRemoteView(userId, remoteView) メソッドを呼び出し、そのユーザーのビデオデータを手動で閲覧する必要があります。SDKが、そのユーザーのビデオデータを受信後、デコードして再生します。

手順7：ローカルのオーディオビデオストリーミングの公開

1. startLocalAudio()を呼び出すと、ローカルのマイク集音を起動させ、採集した音声をエンコードして送信することができます。
2. startLocalPreview()を呼び出すと、ローカルのカメラを起動させ、キャプチャした画面をエンコードして送信することができます。
3. setLocalViewFillMode()を呼び出し、ローカルのビデオ画面の表示モードを設定することができます。

- Fillモードは塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
- Fitモードは適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

4. setVideoEncoderParam()インターフェースを呼び出し、ローカルビデオのエンコードパラメータを設定できます。このパラメータにより、ルーム内の他のユーザーが視聴する際の画面の画質が決定されます。

```
//サンプルコード：ローカルのオーディオ・ビデオストリーミングの公開
mTRTCCloud.setLocalViewFillMode(TRTC_VIDEO_RENDER_MODE_FIT);
mTRTCCloud.startLocalPreview(mIsFrontCamera, mLocalView);
mTRTCCloud.startLocalAudio();
```

手順8：現在のルームから退出する

exitRoom()メソッドを呼び出してルームを退出します。SDKは退室する時に、カメラ、マイクなどのハードウェアデバイスを停止してリリースする必要があるため、退室の動作は瞬時に完了するものではなく、onExitRoom()のコールバックを受信してはじめて、実際に退室操作を完了したことになります。

```
// 退室を呼び出した後は、onExitRoomイベントのコールバックをお待ちください
mTRTCCloud.exitRoom()
@Override
```

```
public void onExitRoom(int reason) {
Log.i(TAG, "onExitRoom: reason = " + reason);
}
```

注意：
Appの中で多くの音声ビデオのSDKを同時に統合した場合は、`onExitRoom`コールバックを受信してからその他の音声ビデオSDKを起動してください。そうしない場合は、ハード上の占有問題が生じることがあります。

通話モードのクイックスタート(Windows)

最終更新日：：2021-11-30 12:05:48

ドキュメントガイド

このドキュメントでは、主にTRTC SDKをベースに簡単なビデオ通話機能を実現する方法を紹介します。ここでは最もよく使われるインターフェースを幾つかリストアップしただけですので、より多くのインターフェース関数を理解したい場合は、API ドキュメントをご参照ください。

サンプルコード

属するプラットフォーム	サンプルコード
Windows（MFC）	TRTCMainViewController.cpp
Windows(Duilib)	TRTCMainViewController.cpp
Windows(C#)	TRTCMainForm.cs

ビデオ通話

1. SDKの初期化

TRTC SDKを使用する第1ステップは、 `getTRTCShareInstance` エクスポートインターフェースによって、 `TRTCCloud` シングルインスタンスのオブジェクトポインタ `ITRTCCloud*` を取得し、SDKのイベントをモニタリングするコールバックを登録することです。

- `ITRTCCloudCallback` のイベントコールバックインターフェースクラスを継承し、キーとなるイベントのコールバックインターフェースをリライトします。これには、ローカルユーザーの入室/退室イベント、リモートユーザーの参加/退出イベント、エラーイベント、アラームイベントなどが含まれます。
- `addCallback` インターフェースを呼び出して、SDKのイベントの監視を登録します。

> 注意：
> `addCallback` でN回登録すると、同一イベントに対して、SDKがN回コールバックします、 `addCallback` を1回のみ呼び出すことをお勧めします。

- C++
- C#

```
// TRTCMainViewController.h

// ITRTCCloudCallbackのイベントコールバックインターフェースクラスを継承します
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()
{
// TRTCCloud インスタンスの作成
m_pTRTCSDK = getTRTCShareInstance();

// SDK イベントコールバックの登録
m_pTRTCSDK-&gt;addCallback(this);
}

TRTCMainViewController::~TRTCMainViewController()
{
// SDK イベントのモニタリング のキャンセル
if(m_pTRTCSDK) {
m_pTRTCSDK-&gt;removeCallback(this);
}

// TRTCCloud インスタンスのリリース
if(m_pTRTCSDK != NULL) {
```

```
destroyTRTCShareInstance();
m_pTRTCSDK = null;
}
}

// エラー通知はモニタリングする必要があります。エラー通知は、SDKの実行を継続できないことを意味します
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();
}
}
```

2. TRTCParamsの組み立て

TRTCParams は SDK で最も重要なパラメータであり、SDKAppID、userId、userSig 、roomIdの4つの記入必須のフィールドがあります。

- **SDKAppID**
  Tencent CloudのTencent Real-Time Communicationコンソールにアクセスします。まだアプリケーションをお持ちでない場合、作成してください。SDKAppIDが表示されます。

- **userId**
  自由に指定することができ、文字列タイプのため、お客様の既存のアカウント体系と同じのものにすることが可能です。但し、**同じ音声/ビデオルームには2つの同名の userIdが存在できません**ので、ご注意ください。

- **userSig**
  SDKAppIDとuserIdを基に、userSigを計算できます。計算方法については、UserSigの計算方法をご参照ください。

- **roomId**
  ルームナンバーは数字タイプとなり、自由に指定できます。但し、**同じアプリケーション内の2つの音声/ビデオルームに、同じroomIdをアサインすることはできません**ので、ご注意ください。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。

3. ルームへの入室（または作成）

`enterRoom` を呼び出し、 TRTCParams パラメータの中の roomIdが指定する音声/ビデオルームに参加できます。該当するルームが存在しない場合、SDK は roomId をルームナンバーとする新しいルームを自動作成しま

す。

appScene パラメータは、SDK のユースケースを指定します。ここでは、 `TRTCAppSceneVideoCall` （ビデオ通話）を使用しますが、このシナリオにおいては、SDK 内部のコーディックおよびネットワークコンポーネントは、映像のスムーズさをより重視し、通話のディレーとラグ率を低減させるものとなっています。

- 入室に成功すると、SDKが `onEnterRoom` インターフェースのコールバックを行います。パラメータ： `result` が0を上回る時は、入室に成功し、数値は入室に要した時間を表しています（単位はミリ秒（ms））。 `result` が0を下回る時は、入室に失敗し、数値は入室失敗のエラーコードを表しています。
- 入室に失敗した場合、SDK は同時に `onError` インターフェースをコールバックします。パラメータは、 `errCode` （エラーコード： `ERR_ROOM_ENTER_FAIL` 、エラーコードは `TXLiteAVCode.h` を参照のこと）、 `errMsg` （エラー原因）、 `extraInfo` （保留パラメータ）です。
- すでに特定のルームにいる場合は、まず `exitRoom` を呼び出して現在のルームを退出すると、もう一つのルームに入ることができるようになります。

- C++
- C#

```
// TRTCMainViewController.cpp

void TRTCMainViewController::enterRoom()
{
// TRTCParamsの定義はヘッダーファイル TRTCCloudDef.hを参照
TRTCParams params;
params.sdkAppId = sdkappid;
params.userId = userid;
params.userSig = usersig;
params.roomId = 908; // 入室したいルームを入力します
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;enterRoom(params, TRTCAppSceneVideoCall);
}
}

...

void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
if(errCode == ERR_ROOM_ENTER_FAIL)
{
LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
// userSigが合法か、ネットワークが正常かなどをチェックします
}
}
```

```
...

void TRTCMainViewController::onEnterRoom(int result)
{
LOGI(L"onEnterRoom result[%d]", result);
if(result &gt;= 0)
{
//入室に成功
}
else
{
//入室失敗、エラーコード = result ;
}
}
```

注意：

- ユースケースに基づき適切なsceneパラメータを選択してください。誤った選択をすると、ラグ率または画面の解像度が想定のレベルに到達しなくなります。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

4. リモート側オーディオストリームの視聴

TRTC SDKは、デフォルトの状態でリモートの音声ストリームを受信するようになっています。このため追加コードを作成する必要はありません。特定の useridの音声ストリームを受信したくない場合は、`muteRemoteAudio` を使用し、ミュートにすることができます。

5. リモートビデオストリームの視聴

TRTC SDKは、デフォルトの状態ではリモートのビデオストリームをプルしません。ルーム内のユーザーに上りビデオデータがあるときに、ルーム内の他のユーザーは、ITRTCCloudCallbackの中の `onUserVideoAvailable` のコールバックによって当該ユーザーのuseridを取得できます。その後、 `startRemoteView` のメソッドを呼び出せば当該ユーザーのビデオ画面を表示できます。

`setRemoteViewFillMode` によって、ビデオ表示モードを `Fill` または `Fit` モードに指定することができます。この2種類のモードはビデオサイズはいずれも同じ比率で拡大縮小します。違いは以下のとおりです。

- `Fill` モード：ビューウィンドウが全てコンテンツで埋まることを優先的に保証します。拡大縮小後のビデオサイズがビューウィンドウのサイズと一致しない場合、はみ出たビデオの部分はカットされます。
- `Fit` モードでは、すべてのビデオコンテンツを確実に表示することが優先されます。拡大・縮小されたビデオサイズと表示ウィンドウのサイズが一致しない場合、塗りつぶされていないウィンドウ領域は黒で塗りつぶさ

れます。

- C++
- C#

```
// TRTCMainViewController.cpp
void TRTCMainViewController::onUserVideoAvailable(const char* userId, bool available){
if (available) {
// レンダリングウィンドウのハンドルを取得します。
CWnd *pRemoteVideoView = GetDlgItem(IDC_REMOTE_VIDEO_VIEW);
HWND hwnd = pRemoteVideoView-&gt;GetSafeHwnd();

// リモートユーザーのビデオのレンダリングモードを設定します。
m_pTRTCSDK-&gt;setRemoteViewFillMode(TRTCVideoFillMode_Fill);
// SDK インターフェースを呼び出し、リモートユーザーのストリーミングを再生します。
m_pTRTCSDK-&gt;startRemoteView(userId、 hwnd);
} else {
m_pTRTCSDK-&gt;stopRemoteView(userId);
}
}
```

6. ローカル集音のオン/オフ

TRTC SDKは、デフォルトではローカルのマイクによる集音がオンになっていません。 `startLocalAudio` で、ローカルの集音をオンにして音声ビデオデータを発信することができ、 `stopLocalAudio` でこれをオフにします。

説明：
`startLocalPreview` の後に引き続き `startLocalAudio` を呼び出すことができます。

7. ローカルビデオ撮影のオン/オフ

TRTC SDK は、デフォルトではローカルのWebカメラの撮影が有効になっていません。 `startLocalPreview` でローカルのWebカメラをオンにしてプレビュー画面を表示でき、 `stopLocalPreview` でこれをオフにします。

- `startLocalPreview` を呼び出して、ローカルビデオのレンダリングウィンドウを指定します。**SDKがウィンドウのサイズをダイナミックに検出して、 `rendHwnd` が表示する全てのウィンドウでレンダリングを行います。**
- `setLocalViewFillMode` インターフェースを呼び出し、ローカルのビデオレンダリングモードを `Fill` または `Fit` に設定します。2種類のモードは、ビデオサイズはいずれも同じ比率で拡大縮小します。違いは次のとおりです。

- `Fill` モード：ウィンドウ全てにコンテンツを表示することを優先的に保証します。拡大縮小後のビデオサイズがビューウィンドウのサイズと一致しない場合、はみ出た部分はカットされます。
- `Fit` モードでは、すべてのビデオコンテンツを確実に表示することが優先されます。拡大・縮小されたビデオサイズとウィンドウのサイズが一致しない場合、塗りつぶされていないウィンドウ領域は黒で塗りつぶされます。

- C++
- C#

```
// TRTCMainViewController.cpp

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

// レンダリングウィンドウのハンドルを取得します。
CWnd *pLocalVideoView = GetDlgItem(IDC_LOCAL_VIDEO_VIEW);
HWND hwnd = pLocalVideoView-&gt;GetSafeHwnd();

if(m_pTRTCSDK)
{
// SDKインターフェースを呼び出し、レンダリングモードおよびレンダリングウィンドウを設定します。
m_pTRTCSDK-&gt;setLocalViewFillMode(TRTCVideoFillMode_Fit);
m_pTRTCSDK-&gt;startLocalPreview(hwnd);
}

...
}
```

8. オーディオ・ビデオデータストリームの遮断

- **ローカルビデオデータの遮断**
  ユーザーが通話の途中に、プライバシーを守る目的で、ローカルのビデオデータを隠したい場合は、`muteLocalVideo` を呼び出して、一時的に、ルーム内の他のユーザーが当該ユーザーの画面を視聴できなくすることができます。

- **ローカル音声データの遮断**
  ユーザーが通話の途中に、プライバシーを守る目的で、ローカルの音声データを遮断したい場合は、`muteLocalAudio` を呼び出して、一時的に、ルーム内の他のユーザーに当該ユーザーの音声を聞こえなくすることができます。

- **リモートビデオデータの遮断**
  `stopRemoteView` によって特定の userid のビデオデータを遮断することができます。
  `stopAllRemoteView` によって全てのリモートユーザーのビデオデータを遮断することができます。

- **リモート音声データの遮断**
  `muteRemoteAudio` によって特定のuseridの音声データを遮断することができます。
  `muteAllRemoteAudio` によって全てのリモートユーザーの音声データを遮断することができます。

9. ルームからの退出

`exitRoom` メソッドを呼び出してルームを退出します。通話中かどうかにかかわらず、このメソッドを呼び出せば、ビデオ通話に関するすべてのリソースがリリースされます。

> 説明：
> `exitRoom` を呼び出した後、SDKは複雑な退室のハンドシェイクのプロセスに進みます。SDKが `onExitRoom` メソッドをコールバックした時に、リソースのリリースが完了します。

- C++
- C#

```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;exitRoom();
}
}
....
void TRTCMainViewController::onExitRoom(int reason)
{
// 退室に成功しました。reasonパラメータは保留され、まだ使用されていません。

...
}
```

通話モードクイックスタート(Electron)

最終更新日：：2022-02-14 14:49:57

ユースケース

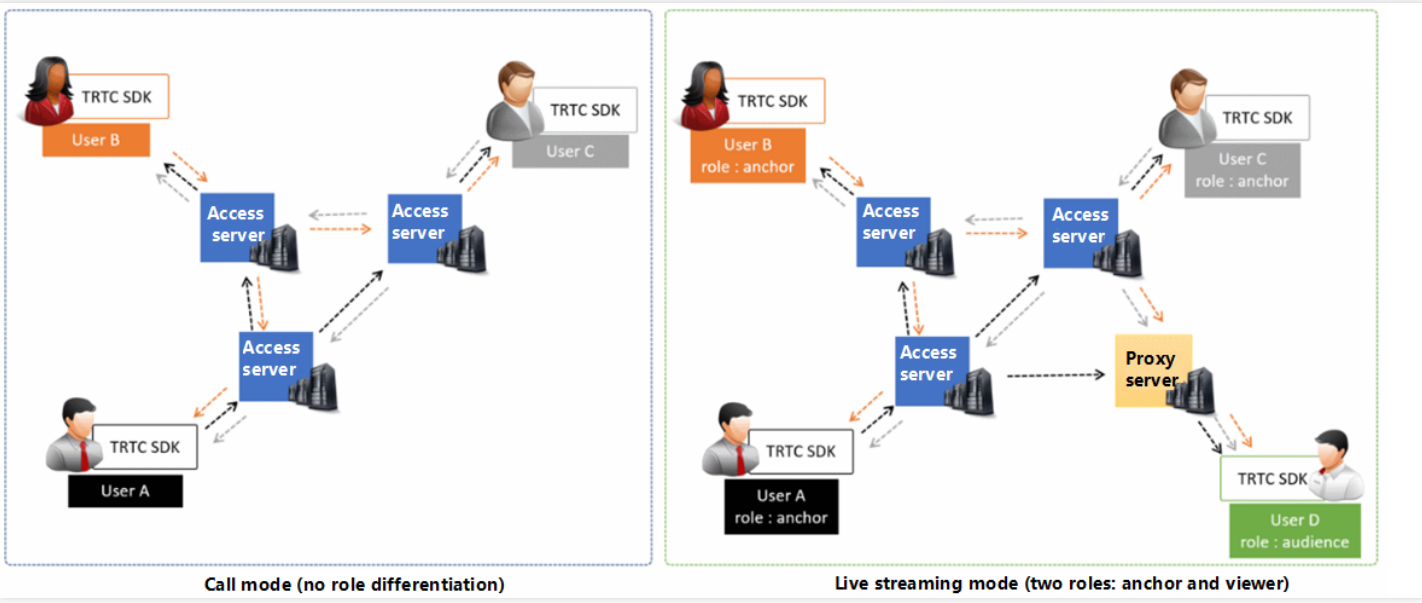
TRTCは、4種類の異なる入室モードをサポートしています。このうち、ビデオ通話（VideoCall）および音声通話（VoiceCall）を総称して通話モードといい、ビデオ・インタラクティブストリーミング（Live）およびボイス・インタラクティブストリーミング（VoiceChatRoom）を総称して ライブストリーミングモードといいます。
通話モードでのTRTCは、1つのルームに最大で300人の同時オンラインをサポートし最大で50人の同時発言をサポートします。1対1のビデオ通話、300人のビデオミーティング、オンライン問診、リモート面接、ビデオカスタマーサービス、オンライン人狼ゲームなどのユースケースに適合しています。

原理解析

TRTCクラウドサービスは、「インターフェースモジュール」および「プロキシモジュール」という2種類の異なるタイプのサーバーノードから構成されています。

- **インターフェースモジュール**
  この種のノードは、最も良質の回線および高性能の機器を採用しており、エンドツーエンドの低遅延マイク接続通話の処理に優れますが、単位時間当たりの費用はやや高くなります
- **プロキシモジュール**
  この種のノードは、通常の回線および性能も一般的な機器を採用しており、同時進行性の高いプルストリーミング再生のニーズの処理にすぐれ、単位当たりの費用はやや低くなります。

通話モードでは、TRTCルームのすべてのユーザーはインターフェースモジュールに割り当てられます。各ユーザーは「キャスター」に相当し、各ユーザーは随時発言でき（同時アップストリームの最大制限は50）、このためオンラインミーティングなどのユースケースに適していますが、1つのルームの人数制限は300人になります。



サンプルコード

Githubにログインして、このドキュメントに関連したサンプルコードを取得することができます。

操作手順

手順1：公式ウェブサイトのSimpleDemoクイックスタートを試行する

初めにドキュメントSimpleDemoクイックスタート(Electron)を読み、ドキュメントのガイドに従って、提供されている公式SimpleDemoクイックスタートを実行してください。

SimpleDemoが順調に動作する場合は、プロジェクトにおいてElectronのインストール方法をお客様が把握していることを意味します。

- 反対に、 SimpleDemo の動作に問題がある場合は、Electronのダウンロード、インストールに問題があったことが考えられます。この場合はElectron の公式サイトの インストールガイド をご参照ください。

手順2：お客様のプロジェクトにtrtc-electron-sdkを統合する

[手順1]](#step1) が正常に動作し、予想どおりの効果があった場合は、 Electron環境のインストール方法を把握していることを意味します。

- 弊社の公式Demoをベースとして二次開発を行うことができ、プロジェクトの初級段階がスムーズに進みます。
- 次のコマンドを実行して、既存のプロジェクトに `trtc-electron-sdk` をインストールすることもできます。

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

手順3：SDKインスタンスを初期化し、イベントコールバックを監視する

1. `trtc-electron-sdk` インスタンスの新規作成：

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

2. `onError` イベントの監視：

```
// エラー通知は監視すべきもので、捕捉してユーザーに通知する必要があります
let onError = function(err) {
console.error(err);
};
trtcCloud.on('onError',onError);
```

手順4：入室パラメータTRTCParamsを組み立てる

enterRoom()インターフェースを呼び出すときはキーパラメータTRTCParamsを入力する必要があります。このパラメータに含まれる入力必須フィールドは下表に示すとおりです。

パラメータ	タイプ	説明	サンプル
sdkAppId	数字	アプリケーションID。 コンソール >【アプリケーション管理】>【アプリケーション情報】にあります。	1400000123
userId	文字列	アルファベットの大文字、小文字（a-z、A-Z）、数字（0-9）、下線およびハイフンのみを許可。ビジネスの実際のアカウントシステムを組み合わせて設定することをお勧めします。	test_user_001
userSig	文字列	userIdを基にuserSigを計算できます。計算方法はUserSigの計算方法をご参照ください。	eJyrVareCeYrSy1SsII...
roomId	数字	数字タイプのルームナンバー。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。	29834

```
import {
TRTCParams,
```

```
TRTCRoleType
} from "trtc-electron-sdk/liteav/trtc_define";

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

注意：

- TRTCは、2つの同じuserIdによる同時入室をサポートしていません。同時に入室した場合、相互に干渉します。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

手順5：ルームの新規作成および入室

1. enterRoom()を呼び出せば、TRTCParamsパラメータの `roomId` が示すオーディオ・ビデオルームに参加できます。該当するルームが存在しない場合、SDKはフィールド `roomId` の値をルームナンバーとする新しいルームを自動的に作成します。
2. ユースケースに基づき適切な `appScene` パラメータを設定してください。誤った選択をすると、ラグ率または画面の解像度が想定のレベルに到達しなくなります。
   - ビデオ通話は、 `TRTCAppScene.TRTCAppSceneVideoCall` に設定してください。
   - 音声通話は、 `TRTCAppScene.TRTCAppSceneAudioCall` に設定してください。

   説明：
   `TRTCAppScene` の詳細な説明については、TRTCAppScene をご参照ください。

3. 入室に成功すると、SDKはonEnterRoom(result)イベントをコールバックします。このうち、パラメータ `result` が0より大きいときは、入室成功を意味し、具体的な数値は入室のために消費した時間となります。単位はミリ秒（ms）です。 `result` が0より小さいときは、入室失敗を意味し、具体的な数値は入室失敗のエラーコードとなります。

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

```
let onEnterRoom = function (result) {
if (result &gt; 0) {
console.log(`onEnterRoom、入室成功、${result}秒を使用`);
}else{
console.warn(`onEnterRoom: 入室失敗 ${result}`);
}
};

/ /入室成功イベントを閲覧
trtcCloud.on('onEnterRoom', onEnterRoom);

// 入室。ルームが存在しない場合は、TRTCバックエンドは新規ルームを自動作成
let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneVideoCall);
```

手順6：リモート側のオーディオビデオストリーミングの閲覧

SDKは、自動閲覧モードおよび手動閲覧モードの2種類のモードをサポートします。自動閲覧はインスタントブロードキャスティング速度を追求するので、少人数で通話するユースケースに適しています；手動閲覧はトラフィックの節約を追求するので、人数が多いミーティングでのユースケースに適しています。

自動閲覧（推奨）

特定のルームに入った後、SDKはルームのその他のユーザーのオーディオストリームを自動で受信します。これによって最高の「インスタントブロードキャスティング」効果に達します。

1. ルーム内の他のユーザーがオーディオデータをアップストリームすると、onUserAudioAvailable()のイベント通知を受信し、SDKがそのリモートユーザーの音声を自動再生します。
2. muteRemoteAudio(userId, true)によって、特定のuserId のオーディオデータを遮断でき、muteAllRemoteAudio(true)によって、すべてのリモートユーザーのオーディオデータを遮断することもできます。遮断した後、SDKは、当該リモートユーザーのオーディオデータをそれ以上プルしなくなります。
3. ルーム内の他のユーザーがビデオデータをアップストリームすると、onUserVideoAvailable()のイベント通知を受信しますが、このときSDKはビデオデータをどのように表示するか指令を受け取っていないため、ビデオデータを自動処理することはありません。startRemoteView(userId, view, streamType)メソッドを呼び出して、リモートユーザーのビデオデータと `view` （表示）をバインドする必要があります。
4. setLocalViewFillMode()によって、ビデオ画面の表示モードを指定することができます。
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` モード：塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。

- `TRTCVideoFillMode.TRTCVideoFillMode_Fit` モード：適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

5. stopRemoteView(userId)によって、特定のuserIdのビデオデータを遮断でき、stopAllRemoteView()によって、すべてのリモートユーザーのビデオデータを遮断することもできます。遮断した後、SDKは、該当するリモートユーザーのオーディオデータをそれ以上プルしなくなります。

```
<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;

/**
* ビデオのカメラ起動の有無
* @param {number} uid - ユーザーID
* @param {boolean} available - 画面の起動の有無
**/

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);
}
}
};

// インスタンスコード：通知に基づきリモート側ユーザーのビデオ画面を閲覧（または閲覧の取り消し）します。
trtcCloud.on('onUserVideoAvailable', onUserVideoAvailable);
```

```
</script>
```

説明：
`onUserVideoAvailable()` イベントのコールバックを受信した後、すぐに `startRemoteView()` を呼び出してビデオストリームを閲覧しない場合、SDKは5s以内にリモートからのビデオデータの受信を停止します。

手動閲覧

setDefaultStreamRecvMode(autoRecvAudio, autoRecvVideo)インターフェースによって、SDKを手動閲覧モードに指定できます。手動閲覧モードでは、SDKはルーム内のその他ユーザーの音声ビデオデータを自動受信しませんので、API関数を介して、手動でトリガーする必要があります。

1. **入室前**に、setDefaultStreamRecvMode(false, false)インターフェースを呼び出して、SDKを手動閲覧モードに設定します。
2. ルーム内の他のユーザーがオーディオデータをアップストリームすると、onUserAudioAvailable()のイベント通知を受信します。この時、muteRemoteAudio(userId, false)を呼び出して、当該ユーザーのオーディオデータを手動で閲覧する必要があります。SDKは、そのユーザーのオーディオデータを受信した後、デコードして再生します。
3. ルーム内の他のユーザーがビデオデータをアップストリームすると、onUserVideoAvailable(userId, available)のイベント通知を受信します。この時、startRemoteView(userId, view)メソッドを呼び出して、当該ユーザーのビデオデータを手動で閲覧する必要があります。SDKは、そのユーザーのビデオデータを受信した後、デコードして再生します。

手順7：ローカルのオーディオビデオストリーミングの公開

1. startLocalAudio()を呼び出すと、ローカルのマイク集音を起動し、採集した音声をエンコードして送信することができます。
2. startLocalPreview()を呼び出すと、ローカルのカメラを起動し、キャプチャした画面をエンコードして送信することができます。
3. setLocalViewFillMode()を呼び出すと、ローカルのビデオ画面の表示モードを設定することができます。
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` モードは塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fit` は適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。
4. setVideoEncoderParam()インターフェースを呼び出すと、ローカルビデオのエンコードパラメータを設定できます。このパラメータにより、ルーム内の他のユーザーが視聴する際の画面の画質が決定されます。

```
//サンプルコード：ローカルのオーディオ・ビデオストリーミングの公開
trtcCloud.startLocalPreview(view);
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);
trtcCloud.startLocalAudio();
//ローカルビデオコーデックパラメータの設定
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はデフォルトでは、現在のシステムのデフォルトのカメラおよびマイクを使用します。setCurrentCameraDevice()およびsetCurrentMicDevice()を呼び出して、その他のカメラおよびマイクを選択することができます。

手順8：現在のルームから退出する

exitRoom()メソッドを呼び出してルームを退出します。SDKは退出時にカメラやマイクなどのハードデバイスを停止またはリリースする必要があるため、退出動作は一瞬では完了しません。退出操作を完了するにはonExitRoom()コールバックを受信する必要があります。

```
// 退室を呼び出した後は、onExitRoomイベントのコールバックをお待ちください
let onExitRoom = function (reason) {
console.log(`onExitRoom, reason: ${reason}`);
};
trtcCloud.exitRoom();
trtcCloud.on('onExitRoom', onExitRoom);
```

注意：
Electronプログラムで複数の音声ビデオSDKを同時に統合した場合は、`onExitRoom`を受信してコールバックしてから、その他の音声ビデオSDKを起動してください。そうしない場合は、ハード上の占有問題が生じることがあります。

通話モードクイックスタート(Web)

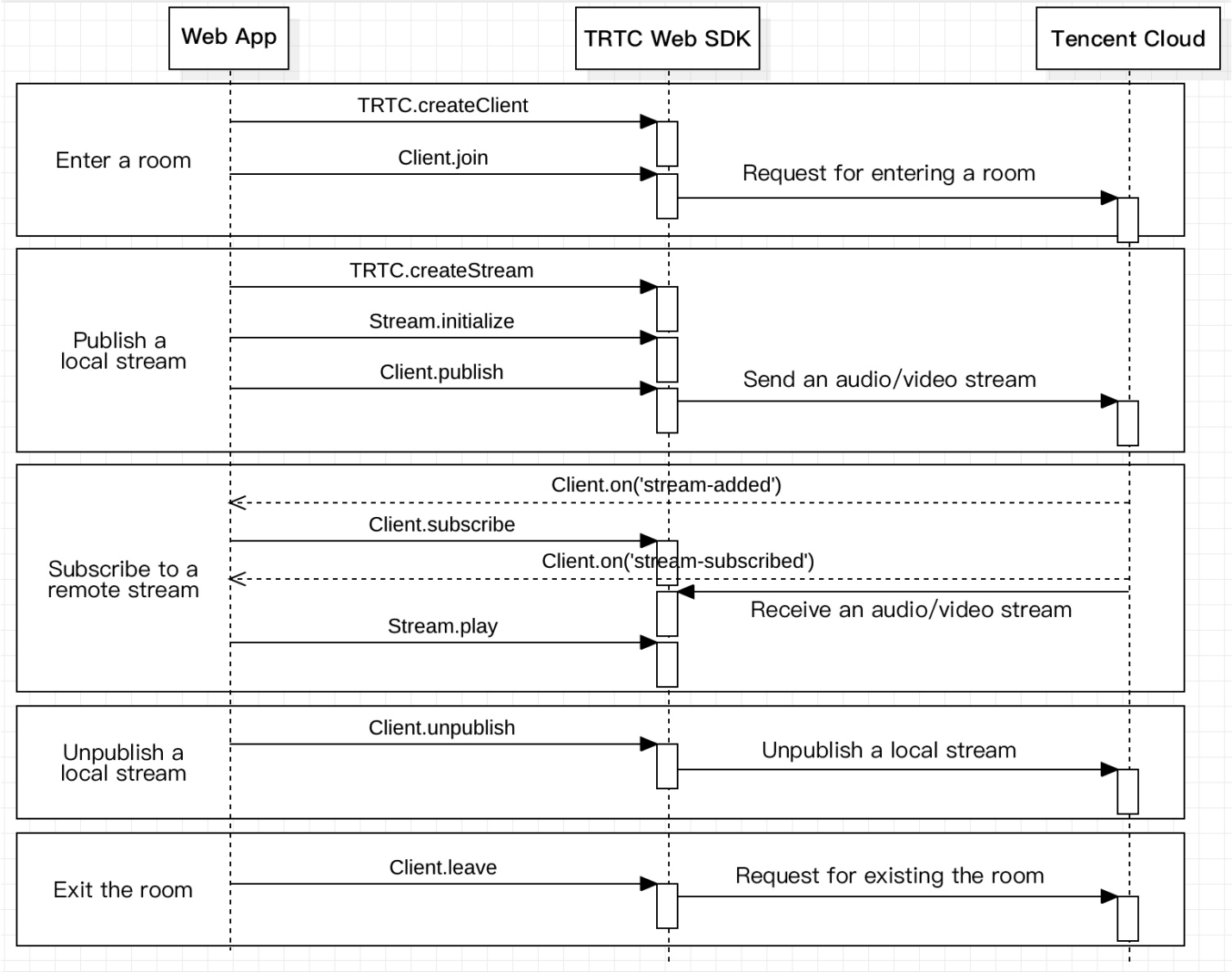
最終更新日：：2022-02-14 14:49:57

ここでは、主にTencent Cloud TRTC Web SDKの基本ワークフロー、およびリアルタイムなオーディオビデオ通話機能を実装する方法を紹介します。

TRTC Web SDKの使用中には、以下のオブジェクトが頻繁に登場します。

- Client オブジェクト。ローカルクライアントを表します。Clientクラスのメソッドにより、通話ルームへの参加、ローカルストリーミングの公開、リモートストリーミングの閲覧などの機能を提供します。
- Streamオブジェクト。オーディオビデオストリーミングオブジェクトを表し、ローカルのオーディオビデオストリーミングオブジェクトLocalStream、およびリモート側のオーディオビデオストリーミングオブジェクトRemoteStreamが含まれます。Streamクラスのメソッドでは主に、オーディオビデオストリーミングオブジェクトのアクションを提供し、これにはオーディオおよびビデオの再生コントロールが含まれます。

基本のオーディオビデオ通話のAPIコールフローは下図に示すとおりです。



手順1： Clientオブジェクトの新規作成

TRTC.createClient()メソッドによって、Clientオブジェクトを作成します。パラメータの設定は以下のとおりです。

- `mode` ：TRTC通話モードのこと。 `rtc` に設定
- `sdkAppId` ：Tencent Cloudから申請したsdkAppId
- `userId` ：ユーザーID
- `userSig` ：ユーザー署名。計算方式はUserSigの計算方法をご参照ください

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

手順2：オーディオビデオ通話ルームへの参加

Client.join()を呼び出して、オーディオビデオ通話ルームに参加します。
パラメータ `roomId` ：ルームID

```
client
.join({ roomId })
.then(() => {
console.log('入室成功');
})
.catch(error => {
console.error('入室失敗 '+ error);
});
```

手順3：ローカルストリーミングの公開およびリモートストリーミングの閲覧

1. TRTC.createStream()メソッドを使用して、ローカルのオーディオビデオストリーミングを作成します。
   以下のインスタンスは、カメラおよびマイクからオーディオビデオストリーミングを採集しています。パラメータは以下のとおり設定します。
   - `userId` ：ローカルストリーミング所属のユーザーID
   - `audio` ：オーディオの起動の有無
   - `video` ：ビデオの起動の有無

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

2. LocalStream.initialize()を呼び出して、ローカルのオーディオビデオストリーミングを初期化します。

   ```
   localStream
   .initialize()
   ```

```
.then(() => {
console.log('ローカルストリーミング初期化の成功');
})
.catch(error => {
console.error('ローカルストリーミング初期化の失敗 ' + error);
});
```

3. ローカルストリーミングの初期化に成功したら、Client.publish()メソッドを呼び出して、ローカルストリーミングを公開します。

```
client
.publish(localStream)
.then(() => {
console.log('ローカルストリーミング公開の成功');
})
.catch(error => {
console.error('ローカルストリーミング公開の失敗 ' + error);
});
```

4. リモートストリーミングは、Client.on('stream-added')のイベント監視によって取得され、このイベントを受信した後、Client.subscribe()を介して、リモート側のオーディオビデオストリーミングを閲覧します。

説明：

- Client.join()で入室する前に、Client.on('stream-added')イベントを登録し、リモートユーザーの入室通知を見落とすことがないようにしてください。
- リモートストリーミングから退出するなどその他のイベントは、API詳細ドキュメントで確認できます。

```
client.on('stream-added', event => {
const remoteStream = event.stream;
console.log('リモートストリーミングの増加: ' + remoteStream.getId());
//リモートストリーミングの閲覧
client.subscribe(remoteStream);
});
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('リモートストリーミングの閲覧成功: ' + remoteStream.getId());
// リモートストリーミングの再生
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

5. ローカルストリーミングの初期化成功のコールバック、またはリモートストリーミングの閲覧成功イベントのコールバックでは、Stream.play()メソッドを呼び出すことで、Webページ上で音声ビデオを再生します。 `play` メソッドがdivエレメントのIDをパラメータとして受け入れると、SDKの内部で、そのdivエレメント下に対応する音声ビデオタグを自動作成し、その上で音声ビデオを再生します。

   - ローカルストリーミングの初期化が成功したとき、ローカルストリーミングを再生します

```
localStream
.initialize()
.then(() => {
console.log('ローカルストリーミング初期化の成功');
localStream.play('local_stream');
})
.catch(error => {
console.error('ローカルストリーミング初期化の失敗 ' + error);
});
```

   - リモートストリーミングの閲覧が成功したとき、リモートストリーミングを再生

```
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('リモートストリーミングの閲覧成功: ' + remoteStream.getId());
// リモートストリーミングの再生
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

手順4：オーディオビデオ通話ルームからの退出

通話終了時は、Client.leave()メソッドを呼び出して、オーディオビデオ通話ルームを退出し、すべてのオーディオビデオ通話によるセッションが終了します。

```
client
.leave()
.then(() => {
// 退室成功。再度client.joinをコールして再入室し、新たに通話することもできます。
})
.catch(error => {
console.error('退室失敗 ' + error);
// このエラーは回復不可能です。ページを更新する必要があります。
});
```

注意：
各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

リアルタイム画面共有
リアルタイム画面共有(iOS)

最終更新日：：2022-03-09 16:24:44

Tencent CloudのTRTCは、iOSプラットフォームで2つの異なる画面共有ソリューションをサポートしています。

- **アプリ内共有**
  現在のアプリの画面のみを共有できます。この特性をサポートするには、iOSのバージョン13以降のOSが必要です。現在のアプリ以外の画面コンテンツを共有できないため、高度なプライバシー保護が必要なシーンに適しています。
- **アプリケーション間共有**
  AppleのReplaykitソリューションをベースとして、システム全体の画面コンテンツを共有できます。ただし、現在のアプリはExtension拡張コンポーネントを追加で提供する必要があるため、結合手順はアプリ内共有よりもやや多くなります。

> 注意：
> 注意すべき点としては、モバイル端末版のTRTC SDKは、デスクトップ版のように、「サブストリームの共有」をサポートしていません。これは、iOSとAndroidシステムは、どちらもバックグラウンドで実行されているアプリに対し、カメラの使用権を制限しているからです。従って、サブストリームの共有をサポートする意義はあまりありません。

サポートするプラットフォーム

iOS	Android	Mac OS	Windows	Electron	Chromeブラウザ
✓	✓	✓	✓	✓	✓

アプリ内共有

アプリ内共有のソリューションは非常にシンプルです。TRTC SDKが提供するインターフェース startScreenCaptureInAppを呼び出し、エンコードパラメータ `TRTCVideoEncParam` を渡すだけでOKです。このパラメータはnilに設定することができます。この場合、SDKは画面共有が開始される前のエンコードパラメータを引き続き使用します。

iOS画面共有に推奨されるエンコードパラメータは次のとおりです。

パラメータ項目	パラメータ名	通常の推奨値	テキスト教育シナリオ
解像度	videoResolution	1280 × 720	1920 × 1080
フレームレート	videoFps	10 FPS	8 FPS
最高ビットレート	videoBitrate	1600 kbps	2000 kbps
解像度アダプティブ	enableAdjustRes	NO	NO

- 画面共有されるコンテンツには通常、大幅な変更がなく、FPSを高く設定するのは経済的ではないため、10FPSを推奨します。
- 共有したい画面コンテンツに大量のテキストが含まれている場合、解像度とビットレートを適宜引き上げることができます。
- 最高ビットレート(videoBitrate)とは、画面が大きく変化したときの最高出力ビットレートのことです。画面コンテンツの変化が少ない場合、実際のエンコードビットレートは低くなります。

アプリケーション間共有

サンプルコード

GithubのScreenShareディレクトリに、アプリケーション間共有用のサンプルコードを設置しています。これには、次のようなテキストが含まれています。

```
├── TRTC-API-Example-OC // TRTC API Example
| ├── Basic // アプリケーション間画面共有機能のデモンストレーション
| | ├── ScreenShare // アプリケーション間画面共有機能のデモンストレーション
| | | ├──── ScreenAnchorViewController.h
| | | ├──── ScreenAnchorViewController.m // キャスターのスクリーンキャプチャステータス表示インターフェース
| | | ├──── ScreenAnchorViewController.xib
| | | ├──── ScreenAudienceViewController.h
| | | ├──── ScreenAudienceViewController.m // 視聴者が見るレコーディングインターフェース
| | | ├──── ScreenAudienceViewController.xib
| | | ├──── ScreenEntranceViewController.h
| | | ├──── ScreenEntranceViewController.m // 機能エントリーインターフェース
| | | ├──── ScreenEntranceViewController.xib
| | | ├──── TRTCBroadcastExtensionLauncher.h
| | | ├──── TRTCBroadcastExtensionLauncher.m // スクリーンキャプチャを呼び出すために用いられる補助コード
| | | ├──── TXReplayKit_Screen // スクリーンキャプチャのプロセスBroadcast Upload Extensionコードの詳細は手順2をご参照ください。
```

```
| | | | ├── Info.plist
| | | | ├── SampleHandler.h
| | | | └── SampleHandler.m // システムからのスクリーンキャプチャデータを受信するために使用されます
```

READMEのガイドによって、このサンプルDemoを実行することができます。

結合手順

iOSシステムでアプリケーション間画面共有を行うには、Extensionスクリーンキャプチャプロセスを追加し、メインアプリプロセスと連携してプッシュを行う必要があります。Extensionスクリーンキャプチャプロセスは、システムによってスクリーンキャプチャが必要なときに作成され、システムが収集した画面イメージの受信を担当します。従って、次の事項を行う必要があります。

1. App Groupを作成し、XCodeにおいて設定を行います（オプション）。このステップは、Extensionスクリーンキャプチャプロセスが、同じメインアプリプロセスとプロセス間通信できるようにすることを目的としています。
2. お客様のプロジェクトにおいて、Broadcast Upload ExtensionのTargetを新規作成し、拡張コモジュール用としてカスタマイズされた `TXLiteAVSDK_ReplayKitExt.framework` をSDK圧縮パッケージに統合します。
3. メインアプリ側の受信ロジックと結合し、メインアプリにBroadcast UploadExtensionからのスクリーンキャプチャデータを待機させます。
4. Demoにあらかじめ実装されたhelper class (`RPSystemBroadcastPickerView`)を使用して、TencentMeetingのiOS版のように、ボタンをクリックすれば画面共有を呼び出せる効果を実現します（オプション）。

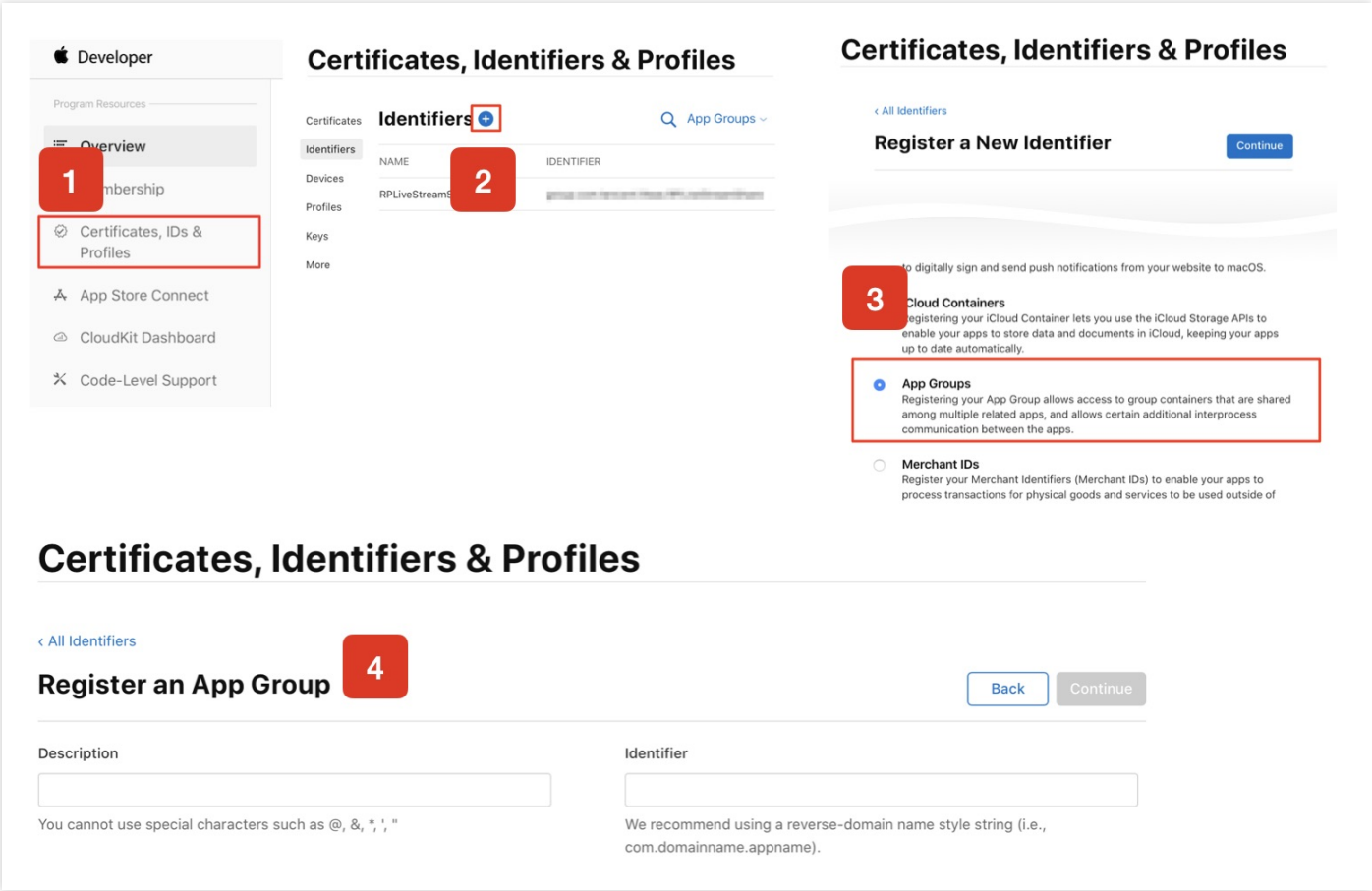
> 注意：
> 手順1をスキップした場合、すなわちApp Groupsを設定しない場合は（インターフェースはnilを渡します）、画面共有は実行できますが、安定性が若干損なわれます。手順はやや多いですが、できる限り正しいApp Groupsを設定して、画面共有機能の安定性を確保してください。

手順1：App Groupsの作成

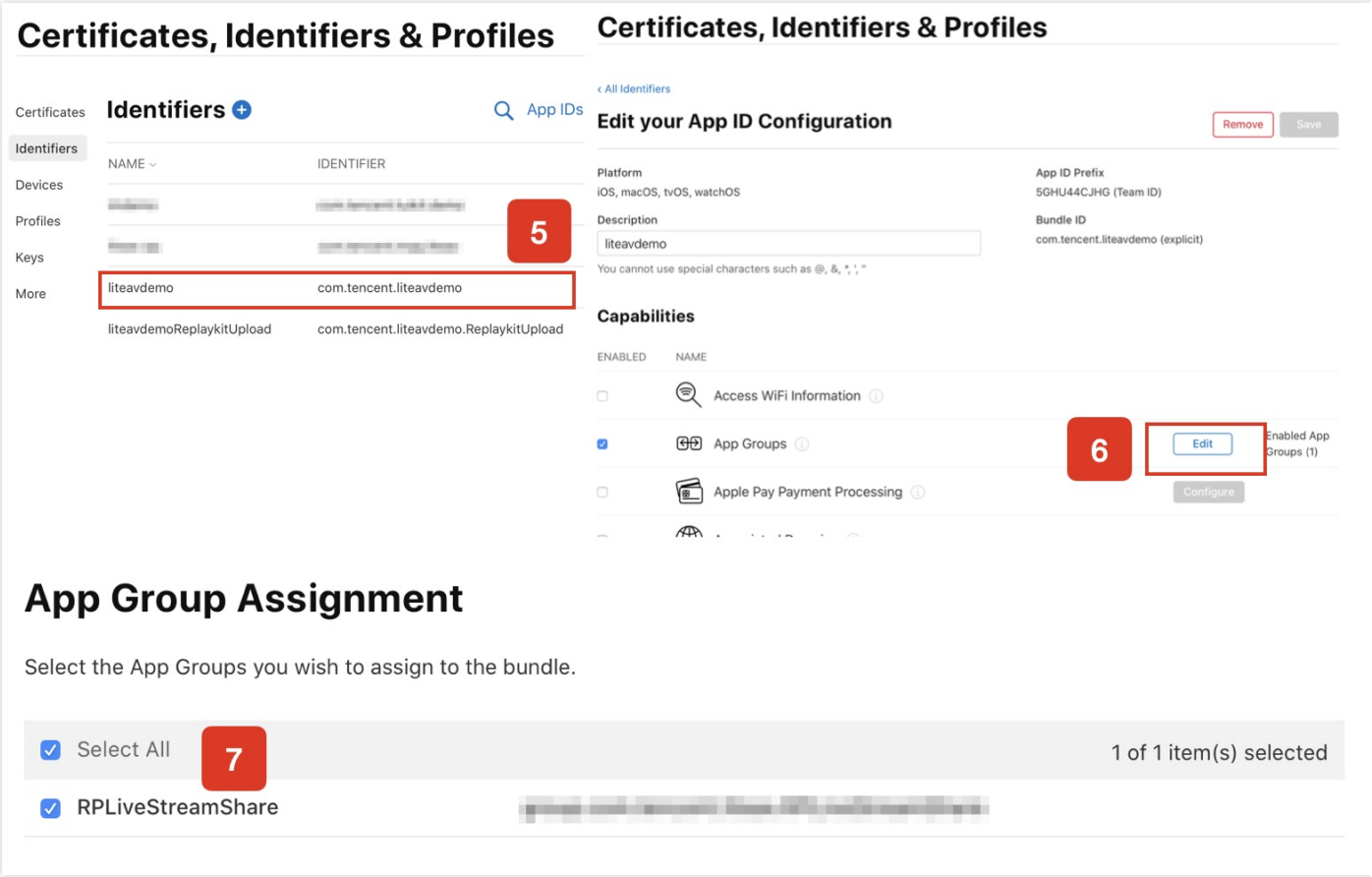
お客様のアカウントを使用して**https://developer.apple.com/**にログインし、次の操作を実行します。**完了後は、対応するProvisioning Profileを再ダウンロードする必要がありますので、ご注意ください。**

1. 【Certificates, IDs & Profiles】をクリックします。
2. 右側のインターフェースでプラス記号をクリックします。
3. 【App Groups】を選択して、【Continue】をクリックします。
4. ポップアップされたフォームにDescriptionとIdentifierを入力します。そのうちIdentifierは、インターフェースにおいて対応するAppGroupパラメータに渡す必要があります。完了したら、【Continue】をクリックしま

す。



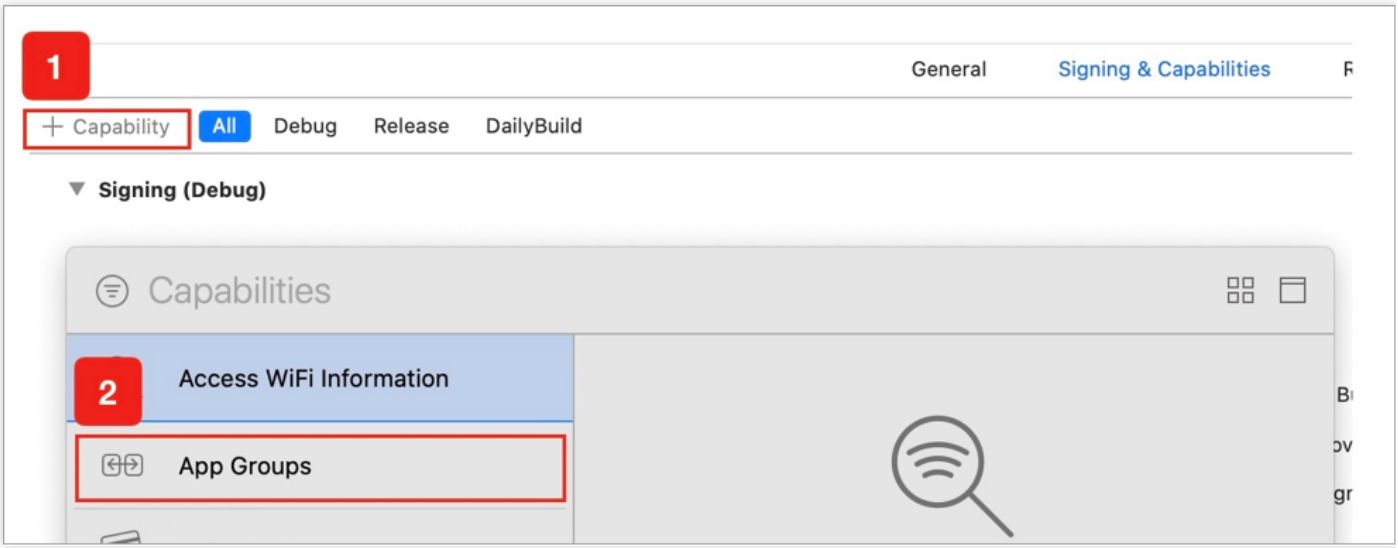
5. Identifierページに戻り、左上のメニューから【App IDs】を選択し、お客様のアプリIDをクリックします（メインアプリとExtensionのアプリIDにも同様の設定が必要です）。
6. 【App Groups】を選択し、【Edit】をクリックします。
7. ポップアップされたフォームからお客様が作成したApp Groupを選択し、【Continue】をクリックして編集ページに戻り、【Save】をクリックして保存します。



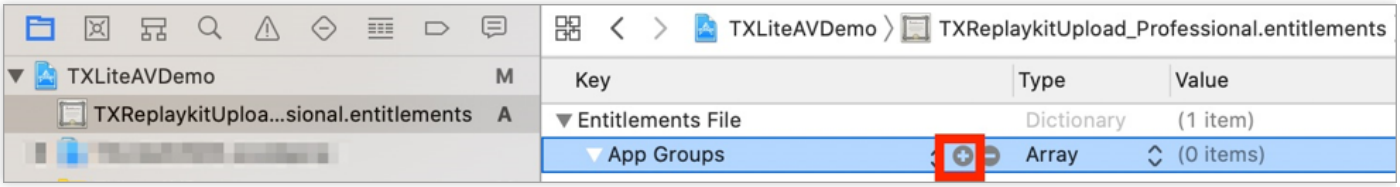
8. Provisioning Profileを再度ダウンロードし、XCodeに設定します。

手順2：Broadcast Upload Extensionの作成

1. Xcodeメニューの【File】、【New】、【Target...】を順にクリックし、【Broadcast Upload Extension】を選択します。
2. ポップアップされたダイアログボックスに関連情報を入力し、**【Include UI Extension】にはチェックを入れずに**【Finish】をクリックして作成を完了します。
3. ダウンロードしたSDK圧縮パッケージのプロジェクトにTXLiteAVSDK_ReplayKitExt.frameworkをドラッグし、先ほど作成したTargetにチェックを入れます。
4. 新しく追加したTargetを選択し、【+ Capability】を順番にクリックして、【App Groups】をダブルクリックします。下図のとおりです。



操作が完了すると、下図のようにファイルリストに `Target名.entitlements` という名前のファイルが生成されます。このファイルを選択し、+記号をクリックして上記の手順のApp Groupを入力すればOKです。



5. メインアプリのTargetを選択し、**上記の手順に従って、メインアプリのTargetに同様の処理を行います。**
6. 新規作成したTargetにおいて、Xcodeは「SampleHandler.h」という名前のファイルを自動的に作成し、それを次のコードに置き換えます。**コード内のAPPGROUPを先ほど作成したApp Group Identifierに変更する必要があります。**

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

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

@interface SampleHandler() <txreplaykitextdelegate>
@end

@implementation SampleHandler
// 注意：ここでのAPPGROUPは、先ほど作成したApp Group Identifierに変更する必要があります。
- (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 = @"画面共有は終了しました";
break;
case TXReplayKitExtReasonDisconnected:
tip = @"アプリケーションは切断されました";
break;
case TXReplayKitExtReasonVersionMismatch:
tip = @"統合エラー（SDKバージョン番号が一致しません）";
break;
}

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

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

default:
```

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

手順3：メインアプリ側の受信ロジックとの結合

以下の手順に従って、メインアプリ側の受信ロジックと結合します。すなわち、ユーザーが画面共有をトリガーする前に、メインアプリを「待機」状態にして、Broadcast Upload Extensionプロセスからスクリーンキャプチャデータをいつでも受信できるようにします。

1. TRTCCloudがカメラのキャプチャをオフにしていることを確認します。オフになっていない場合は、stopLocalPreviewを呼び出して、カメラのキャプチャをオフにしてください。
2. startScreenCaptureByReplaykit:appGroup:メソッドを呼び出し、手順1で設定したAppGroupを渡して、SDKを「待機」状態にします。
3. ユーザーが画面共有をトリガーするまで待機します。手順4における「トリガーボタン」が実装されていない場合、ユーザーがiOSシステムのコントロールセンターにおいて、スクリーンキャプチャボタンを長押しして画面共有をトリガーする必要があります。
4. stopScreenCaptureインターフェースを呼び出すことにより、いつでも画面共有を停止できます。

```
// 画面共有を開始するには、APPGROUPを上記の手順で作成したApp Group Identifierに置き換える必要があります。
- (void)startScreenCapture {
TRTCVideoEncParam *videoEncConfig = [[TRTCVideoEncParam alloc] init];
videoEncConfig.videoResolution = TRTCVideoResolution_1280_720;
videoEncConfig.videoFps = 10;
videoEncConfig.videoBitrate = 2000;
//APPGROUPを上記の手順で作成したApp Group Identifierに置き換える必要があります。
[[TRTCCloud sharedInstance] startScreenCaptureByReplaykit:videoEncConfig
appGroup:APPGROUP];
}

// 画面共有を停止します
- (void)stopScreenCapture {
[[TRTCCloud sharedInstance] stopScreenCapture];
}

// 画面共有の開始イベントの通知は、TRTCCloudDelegateを介して受信できます
- (void)onScreenCaptureStarted {
[self showTip:@"画面共有の開始"];
}
```

手順4：画面共有のトリガーボタンの追加（オプション）

手順3までに、ユーザーがコントロールセンターからスクリーンキャプチャボタンを長押しして、画面共有を手動で開始する必要があります。以下の方法で、VooVMeetingのように、ボタンをクリックすることでトリガーする効果を実現できます。

1. Demoにおいて `TRTCBroadcastExtensionLauncher` というクラスを見つけて、お客様のプロジェクトに追加します。
2. インターフェースにボタンを設置し、ボタンの応答関数において `TRTCBroadcastExtensionLauncher` の `launch` 関数を呼び出すと、画面共有機能を呼び出すことができます。

```
// カスタムボタンによる応答メソッド
- (IBAction)onScreenButtonTapped:(id)sender {
[TRTCBroadcastExtensionLauncher launch];
}
```

注意：

- AppleはiOS12.0に `RPSystemBroadcastPickerView` を追加しました。これによって、ユーザーがアプリケーションからランチャーをポップアップして、画面共有の開始を確認することができます。現時点では、 `RPSystemBroadcastPickerView` はインターフェースのカスタマイズをサポートしておらず、公式の呼び出しメソッドもありません。
- TRTCBroadcastExtensionLauncherの原理とは、 `RPSystemBroadcastPickerView` のサブViewをスキャンしてUIButtonを見つけ、そのクリックイベントをトリガーするというものです。
- **ただし、このソリューションはAppleで公式に推奨されているものではなく、システムが新たにアップデートされた場合、無効になる可能性があります。従って、手順4はオプションのソリューションにすぎず、お客様の自己責任でこのソリューションをご利用ください。**

画面共有の確認

- **Mac/Windows画面共有の確認**
  ルームにいるMac/Windowsユーザーが画面共有を起動し、サブストリームを介して共有を実行します。ルームにいるその他のユーザーは、TRTCCloudDelegate内のonUserSubStreamAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteSubStreamViewインターフェースを介して、リモートユーザーのサブストリーム画面のレンダリングを起動することができます。

- **Android / iOS画面共有の確認**
  ユーザーがAndroid / iOSを介して画面共有を実行する場合は、メインストリームを介して共有を実行します。ルームにいるその他ユーザーはTRTCCloudDelegateの中のonUserVideoAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteViewインターフェースを介して、リモートユーザーのメインストリーム画面のレンダリングを起動することができます。

リアルタイム画面共有(Android)

最終更新日：：2022-03-09 15:49:06

Tencent CloudのTRTCは、Androidシステムでの画面共有をサポートしており、現在のシステムの画面コンテンツは、TRTC SDKを介してルーム内の他のユーザーと共有されます。この機能については、次のような2つの注意点があります。

- TRTC Android版の画面共有は、デスクトップ版のように「サブチャネルの共有」をサポートしていません。従って、画面共有を開始するときは、あらかじめカメラのキャプチャを停止する必要があります。停止しない場合、衝突が生じます。
- Androidシステムのバックグラウンドアプリが引き続きCPUを使用すると、システムによって強制終了されやすくなり、画面共有自体が必然的にCPUを消費してしまいます。この矛盾するような衝突を解消するには、アプリで画面共有を開始するとともに、Androidシステムにフローティングウィンドウをポップアップする必要があります。Androidはフォアグラウンドを含むアプリプロセスを強制終了しないため、このソリューションによって、お客様のアプリはシステムに自動回収されることなく画面を共有し続けることができます。

サポートするプラットフォーム

iOS	Android	Mac OS	Windows	Electron	Chromeブラウザ
✓	✓	✓	✓	✓	✓

画面共有を開始

Androidデバイスでの画面共有は、 `TRTCCloud` のstartScreenCapture()インターフェースを呼び出すだけで開始することができます。ただし、明瞭で安定した共有効果を実現するには、次の3つの点に注意する必要があります。

Activityの追加

次のactivityをmanifestファイルに貼り付けます（項目コードに存在する場合は追加する必要はありません）。

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

ビデオコーデックパラメータの設定

startScreenCapture()で最初のパラメータ `encParams` を設定することにより、画面共有のエンコード品質を指定することができます。 `encParams` をnullに指定した場合、SDKは以前に設定されたエンコードパラメータを自動的に使用します。推奨されるパラメータの設定は、次のとおりです。

パラメータ項目	パラメータ名	通常の推奨値	テキスト教育シナリオ
解像度	videoResolution	1280 × 720	1920 × 1080
フレームレート	videoFps	10 FPS	8 FPS
最高ビットレート	videoBitrate	1600 kbps	2000 kbps
解像度アダプティブ	enableAdjustRes	NO	NO

- 画面共有されるコンテンツには通常、大幅な変更がなく、FPSを高く設定するのは経済的ではないため、10FPSを推奨します。
- 共有したい画面コンテンツに大量のテキストが含まれている場合、解像度とビットレートを適宜引き上げることができます。
- 最高ビットレート(videoBitrate)とは、画面が大きく変化したときの最高出力ビットレートのことです。画面コンテンツの変化が少ない場合、実際のエンコードビットレートは低くなります。

フローティングウィンドウのポップアップで強制終了を回避

Android 7.0以降のシステムから、バックグラウンドで実行されている通常のアプリプロセスに切り替わりますが、CPUのアクティビティがある場合、常にシステムによって強制終了されやすくなります。従って、アプリをバックグラウンドに切り替えてサイレントで画面共有すると、フローティングウィンドウのポップアップというソリューションによって強制終了を回避することができます。また、携帯電話の画面にフローティングウィンドウを表示することは、ユーザーが今、画面共有を行っていることをユーザーに知らせ、プライバシー情報の漏えい防止につながります。

- **方法1：通常のフローティングウィンドウをポップアップする**
  「VooVMeeting」のようなミニフローティングウィンドウをポップアップするには、サンプルコードFloatingView.java の実装を参照すればOKです。

```
public void showView(View view, int width, int height) {
mWindowManager = (WindowManager) mContext.getSystemService(Context.WINDOW_SERVICE);
//TYPE_TOASTは、4.4以降のシステムにのみ適用できます。下位バージョンをサポートするには、TYPE_SYSTEM_ALERTを使用します（manifestで権限を宣言する必要があります）
//7.1（7.1を含む）以降のシステムは、TYPE_TOASTに対して制限を設けています
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);
}
```

- **方法2：カメラプレビューウィンドウをポップアップする**
  TRTC Android版の画面共有は、デスクトップ版のように「サブチャネルの共有」をサポートしていません。従って、画面共有を開始するときは、カメラというチャネルのビデオデータをアップストリームすることはできません。アップストリームすると、衝突が生じます。
  では、画面とカメラの画像を同時に共有するにはどうすればよいでしょうか。
  答えは簡単です。画面上でカメラの画面をフローティングすればOKです。これをすると、TRTCはスクリーンの画面を収集すると同時に、カメラ画面も共有します。

画面共有の確認

- **Mac / Windows画面共有の確認**
  ルームにいるMac / Windowsユーザーが画面共有を起動し、サブストリームを介して共有を実行します。ルームにいるその他ユーザーはTRTCCloudDelegate中のonUserSubStreamAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteSubStreamViewインターフェースを介してリモートユーザーのサブストリーム画面のレンダリングを起動することができます。

- **Android/iOS画面共有の確認**
  ユーザーがAndroid/iOSを介して画面共有を実行する場合は、メインストリームを介して共有を実行することができます。ルームにいるその他のユーザーは、TRTCCloudListener内のonUserVideoAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteViewインターフェースを介して、リモートユーザーのメインストリーム画面のレンダリングを起動することができます。

```
//サンプルコード：画面共有の画面を見る
@Override
public void onUserSubStreamAvailable(String userId, boolean available) {
startRemoteSubStreamView(userId);
}
```

よくあるご質問

1つのルームで同時にいくつの画面を共有できますか。

現在、1つの TRTC オーディオ・ビデオルームで共有できる画面は1つだけです。

リアルタイム画面共有(Windows)

最終更新日：：2021-11-08 11:41:30

Tencent Cloud TRTCは画面共有機能をサポートし、Windowsプラットフォームにおける画面共有はビッグストリーム共有とサブストリーム共有の2つのスキームをサポートします。

- **サブストリーム共有**
  TRTCでは、画面共有のための「サブストリーム（**substream**）」という1チャネルのアップストリームのビデオストリームを個別にスタートできます。サブストリーム共有は、キャスターがカメラ画面とスクリーン画面の両方を同時にアップロードします。これはTencentMeetingの使用スキームであり、`startScreenCapture` インターフェースを呼び出す場合に、`TRTCVideoStreamType` パラメータを `TRTCVideoStreamTypeSub` に指定して、このモードをイネーブルできます。このストリーム画面を閲覧するには専用の `startRemoteSubStreamView` インターフェースを使用する必要があります。

- **ビッグストリーム共有**
  TRTCでは、通常、カメラが動くチャネルを「ビッグストリーム（**bigstream**）」といい、ビッグストリーム共有は、カメラチャネルを使用して画面を共有します。このモードでは、キャスターは、アップストリームのカメラ画面、もしくはアップストリームのスクリーン画面のいずれかのアップストリームのビデオストリームを1チャンネルのみ有し、両者は相互に排他的です。`startScreenCapture` インターフェースを呼び出す場合に、`TRTCVideoStreamType` パラメータを `TRTCVideoStreamTypeBig` に指定し、このモードをイネーブルできます。

サポートするプラットフォーム

iOS	Android	Mac OS	Windows	Electron	Chromeブラウザ
✓	✓	✓	✓	✓	✓

依存するAPI

API機能	C++バージョン	C#バージョン	Electronバージョン
共有ターゲットの選択	selectScreenCaptureTarget	selectScreenCaptureTarget	selectScreenCaptureTarget

API機能	C++バージョン	C#バージョン	Electronバージョン
画面共有を開始	startScreenCapture	startScreenCapture	startScreenCapture
画面共有の一時停止	pauseScreenCapture	pauseScreenCapture	pauseScreenCapture
画面共有のリカバー	resumeScreenCapture	resumeScreenCapture	resumeScreenCapture
画面共有の終了	stopScreenCapture	stopScreenCapture	stopScreenCapture

共有ターゲットの取得

`getScreenCaptureSources` を介して 共有可能なウィンドウのリストをリストアップでき、リストは出力パラメータ sourceInfoList を介して戻されます。

> 説明：
> Windowsのデスクトップ画面もウィンドウの1つであり、デスクトップウィンドウ（Desktop）と呼ばれ、モニターが2台ある場合は、各モニターに対応するデスクトップウィンドウがあります。したがって、getScreenCaptureSourcesを介して返されるウィンドウリストにもDesktopウィンドウがあります。

sourceInfoListの各sourceInfoが共有可能なターゲットは、次のフィールドのように説明されます。

フィールド	タイプ	意味
type	TRTCScreenCaptureSourceType	キャプチャソースタイプ。指定タイプはウィンドウまたはスクリーン
sourceId	HWND	ソースIDの収集 • ウィンドウについては、当該フィールドにウィンドウのハンドル • を表示します。画面については、当該フィールドに画面ID を表示します

フィールド	タイプ	意味
sourceName	string	ウィンドウ名。画面である場合は Screen0 Screen1...を返します
thumbWidth	int32	ウィンドウサムネイル幅
thumbHeight	int32	ウィンドウサムネイル高さ
thumbBGRA	buffer	ウィンドウサムネイルのバイナリーbuffer
iconWidth	int32	ウィンドウアイコンの幅
iconHeight	int32	ウィンドウアイコンの高さ
iconBGRA	buffer	ウィンドウアイコンのバイナリーbuffer

上述の情報に基づき、ユーザーの選択のために共有できるターゲットを一覧表示するシンプルなリストページを実現できます。

共有ターゲットの選択

TRTC SDK は3種の共有モードをサポートしており、 `selectScreenCaptureTarget` を介して指定できます：

- **全画面の共有**：
  全スクリーンウィンドウの共有であり、マルチモニター分割画面をサポートします。1つのsourceInfoList中のtypeに `TRTCScreenCaptureSourceTypeScreen` のsourceパラメータを指定し、captureRectを{ 0, 0, 0, 0 }に設定する必要があります。

- **指定領域の共有**：
  スクリーンの特定領域の共有であり、ユーザーが領域の位置座標を決定する必要があります。1つのsourceInfoList中のtypeに `TRTCScreenCaptureSourceTypeScreen` のsourceパラメータを指定し、captureRectを{ 100, 100, 300, 300 }などの非NULLに設定する必要があります。

- **指定ウィンドウの共有**：
  ターゲットウィンドウのコンテンツの共有であり、ユーザーが共有したいウィンドウを指定する必要があります。1つのsourceInfoList中のtypeに `TRTCScreenCaptureSourceTypeWindow` のsourceパラメータを指定し、captureRectを{ 0, 0, 0, 0 }に設定する必要があります。

> 説明：

2つの追加パラメータ：

- パラメータ captureMouse はマウスポインタをキャプチャするかどうかを指定するために使用されます。
- パラメータhighlightWindow は共有するウィンドウを強調表示するかどうかを指定し、キャプチャされた画像に隠れた箇所がある場合、それを取り除くようユーザーに促します。この部分のUI特殊効果は、SDK内で実現されます。

画面共有の開始

- 共有ターゲットを選択した後、 `startScreenCapture` インターフェースを使用して画面共有を起動することができます。
- 共有プロセスにおいても、 `selectScreenCaptureTarget` を呼び出し、共有ターゲットを変更することができます。
- `pauseScreenCapture` と `stopScreenCapture` の違いは、pauseはスクリーンコンテンツのキャプチャを停止し、その瞬間の画面を一時停止するため、resumeするまで最後の画面がリモート側に表示され続けます。

```
/**
* ¥brief 7.5 【画面共有】画面共有の起動
* ¥param：rendHwnd - プレビュー画面をロードするHWND
*/
void startScreenCapture(HWND rendHwnd);
/**
* ¥brief 7.6 【画面共有】画面共有の一時停止
*/
void pauseScreenCapture();
/**
* ¥brief 7.7 【画面共有】画面共有のリカバー
*/
void resumeScreenCapture();
/**
* ¥brief 7.8 【画面共有】画面共有の終了
*/
void stopScreenCapture();
```

画質の設定

`setSubStreamEncoderParam` インターフェースを介して解像度、ビットレートとフレームレートを含む画面共有の画面品質を設定できます。推奨する参考値を以下に提示します。

解像度レベル	解像度	フレームレート	ビットレート
超高精細（HD+）	1920 × 1080	10	800kbps
高精細（HD）	1280 × 720	10	600kbps
標準（SD）	960 × 720	10	400kbps

画面共有の確認

- **Mac / Windows画面共有の確認**
  ルームにいるMac / Windowsユーザーが画面共有を起動し、サブストリームを介して共有を実行します。ルームにいるその他ユーザーはTRTCCloudDelegate中のonUserSubStreamAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteSubStreamViewインターフェースを介してリモートユーザーのサブストリーム画面のレンダリングを起動することができます。

- **Android / iOS画面共有の確認**
  ユーザーがAndroid / iOSを介して画面共有を実行する場合は、メインストリームを介して共有を実行します。ルームにいるその他ユーザーはTRTCCloudDelegateの中のonUserVideoAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteViewインターフェースを介してリモートユーザーのメインストリーム画面のレンダリングを起動することができます。

```
//サンプルコード：画面共有の画面を見る
void CTRTCCloudSDK::onUserSubStreamAvailable(const char * userId, bool available)
{
LINFO(L"onUserSubStreamAvailable userId[%s] available[%d]¥n", UTF82Wide(userId).c_str(), availabl
e);
if (available) {
startRemoteSubStreamView(userId, hWnd);
} else {
stopRemoteSubStreamView(userId);
}
}
```

よくあるご質問

1つのルームで同時にいくつの画面を共有できますか。
現在、1つのTRTCオーディオ・ビデオルームで共有できる画面は1つだけです。

ウィンドウの共有（SourceTypeWindow）を指定し、ウィンドウのサイズが変化した場合は、ビデオストリームの解像度も変化しますか。
デフォルトでは、SDK内で共有ウィンドウのサイズに従ってエンコーディングパラメータを自動的に調整します。
解像度を固定する必要がある場合は、setSubStreamEncoderParamインターフェースを呼び出し画面共有のエンコーディングパラメータを設定するか、またはstartScreenCaptureを呼び出すときに、対応するエンコーディングパラメータを指定する必要があります。

リアルタイム画面共有(Mac)

最終更新日：：2021-11-08 11:39:29

Tencent Cloud TRTC は画面共有機能をサポートし、Macプラットフォームにおける画面共有はビッグストリーム共有とサブストリーム共有の2つのスキームをサポートします。

- **サブストリーム共有**
  TRTCでは、画面共有のための「サブストリーム（**substream**）」という1チャネルのアップストリームのビデオストリームを個別にスタートできます。サブストリーム共有は、キャスターがカメラ画面とスクリーン画面の両方を同時にアップロードします。これはTencentMeetingの使用スキームであり、 `startScreenCapture` インターフェースを呼び出す場合に、 `TRTCVideoStreamType` パラメータを `TRTCVideoStreamTypeSub` に指定して、このモードをイネーブルできます。このストリーム画面を閲覧するには専用の `startRemoteSubStreamView` インターフェースを使用する必要があります。

- **ビッグストリーム共有**
  TRTCでは、通常、カメラが動くチャネルを「ビッグストリーム（**bigstream**）」といい、ビッグストリーム共有は、カメラチャネルを使用して画面を共有します。このモードでは、キャスターは、アップストリームのカメラ画面、もしくはアップストリームのスクリーン画面のいずれかのアップストリームのビデオストリームを1チャンネルのみ有し、両者は相互に排他的です。 `startScreenCapture` インターフェースを呼び出す場合に、 `TRTCVideoStreamType` パラメータを `TRTCVideoStreamTypeBig` に指定し、このモードをイネーブルできます。

サポートするプラットフォーム

iOS	Android	Mac OS	Windows	Electron	Chromeブラウザ
✓	✓	✓	✓	✓	✓

共有ターゲットの取得

getScreenCaptureSourcesWithThumbnailSizeを介して共有可能なウィンドウをリストアップでき、共有可能な各ターゲットはいずれも `TRTCScreenCaptureSourceInfo` オブジェクトです。

Mac OS内のデスクトップスクリーンも共有可能なターゲットであり、通常のMacウィンドウのtypeは `TRTCScreenCaptureSourceTypeWindow` で、デスクトップスクリーンのtypeは `TRTCScreenCaptureSourceTypeScreen` です。

typeに加え、各 `TRTCScreenCaptureSourceInfo` には次のフィールド情報があります。

フィールド	タイプ	意味
type	TRTCScreenCaptureSourceType	キャプチャソースタイプ:指定タイプはウィンドウまたはスクリーン
sourceId	NSString	キャプチャソースID：ウィンドウの場合、このフィールドはウィンドウハンドルを示します； スクリーンの場合、このフィールドはスクリーンIDを示します
sourceName	NSString	ウィンドウ名、画面である場合は、Screen0 Screen1...を返します
extInfo	NSDictionary	共有ウィンドウの追加情報
thumbnail	NSImage	ウィンドウサムネイル
icon	NSImage	ウィンドウアイコン

上述のこれらの情報があれば、ユーザーの選択のために共有可能なターゲットを一覧表示するシンプルなリストページを実現できます。

共有ターゲットの選択

TRTC SDKは3種類の共有モードをサポートしており、selectScreenCaptureTargetを介して指定できます。

- **全画面の共有**：
  全スクリーンウィンドウの共有であり、マルチモニター分割画面をサポートします。1つのtypeが `TRTCScreenCaptureSourceTypeScreen` のscreenSourceパラメータを指定し、rectを{ 0, 0, 0, 0 }に設定する必要があります。

- **指定領域の共有**：
  スクリーンの特定領域の共有であり、ユーザーが領域の位置座標を決定する必要があります。1つのtypeが `TRTCScreenCaptureSourceTypeScreen` のscreenSourceパラメータを指定し、captureRectを{100、100、300、300}などの非NULLに設定する必要があります。

- **指定ウィンドウの共有**：
  ターゲットウィンドウのコンテンツを共有するには、ユーザーが共有したいウィンドウを指定する必要があり

ます。1つのtypeが `TRTCScreenCaptureSourceTypeWindow` のscreenSourceパラメータを指定し、captureRectを{ 0, 0, 0, 0 }に設定する必要があります。

説明：
2つの追加パラメータ：

- パラメータcapturesCursorはマウスポインタをキャプチャするかどうかを指定するために使用されます。
- パラメータhighlightは共有するウィンドウを強調表示するかどうかを指定し、キャプチャされた画像がブロックされた場合に、オクルージョンを移動するようユーザーに促すために使用されます。（この部分のUI特殊効果は、SDK内で実現されます）

画面共有の開始

- 共有ターゲットを選択した後、startScreenCaptureインターフェースを使用して画面共有を起動することができます。
- 2つの関数pauseScreenCaptureとstopScreenCaptureの違いは、pauseはスクリーンコンテンツのキャプチャを停止し、その瞬間のスクリーンパッドを一時停止するため、resumeScreenCaptureまで最後の画面がリモート側に表示され続けることです。

```
/**
* 7.6 【画面共有】画面共有の起動
* @param viewレンダリングウィジェットが配置される親ウィジェット
*/
- (void)startScreenCapture:(NSView *)view;
/**
* 7.7 【画面共有】画面キャプチャの停止
* @return 0：成功 <0:失敗
*/
- (int)stopScreenCapture;
/**
* 7.8 【画面共有】画面共有の一時停止
* @return 0：成功 <0:失敗
*/
- (int)pauseScreenCapture;
/**
* 7.9 【画面共有】画面共有のリカバー
*
* @return 0：成功 <0:失敗
```

```
*/
- (int)resumeScreenCapture;
```

画質の設定

setSubStreamEncoderParamインターフェースを介して解像度、ビットレートとフレームレートを含む画面共有の画面品質を設定できます。推奨する参考値を以下に提示します。

解像度レベル	解像度	フレームレート	ビットレート
超高精細（HD+）	1920 × 1080	10	800kbps
高精細（HD）	1280 × 720	10	600kbps
標準（SD）	960 × 720	10	400kbps

画面共有の確認

- **Mac/Windows画面共有の確認**
  ルームにいるMac/Windowsユーザーが画面共有を起動し、サブストリームを介して共有を実行します。ルームにいるその他のユーザーは、TRTCCloudDelegate内のonUserSubStreamAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteSubStreamViewインターフェースを介して、リモートユーザーのサブストリーム画面のレンダリングを起動することができます。

- **Android/iOS画面共有の確認**
  ユーザーがAndroid / iOSを介して画面共有を実行する場合は、メインストリームを介して共有を実行します。ルームにいるその他ユーザーはTRTCCloudDelegateの中のonUserVideoAvailableイベントを介してこの通知を受け取ります。
  画面共有を確認したいユーザーはstartRemoteViewインターフェースを介して、リモートユーザーのメインストリーム画面のレンダリングを起動することができます。

```
//サンプルコード：画面共有の画面を見る
- (void)onUserSubStreamAvailable:(NSString *)userId available:(BOOL)available {
if (available) {
[self.trtcCloud startRemoteSubStreamView:userId view:self.capturePreviewWindow.contentView];
} else {
[self.trtcCloud stopRemoteSubStreamView:userId];
```

```
}
}
```

よくあるご質問

1つのルームで同時にいくつの画面を共有できますか。
現在、1つのTRTCオーディオ・ビデオルームで共有できる画面は1つだけです。

ウィンドウの共有（SourceTypeWindow）を指定し、ウィンドウのサイズが変化した場合は、ビデオストリームの解像度も変化しますか。
デフォルトでは、SDK内で共有ウィンドウのサイズに従ってエンコーディングパラメータを自動的に調整します。
解像度を固定する必要がある場合は、setSubStreamEncoderParamインターフェースを呼び出し画面共有のエンコーディングパラメータを設定するか、またはstartScreenCaptureを呼び出すときに、対応するエンコーディングパラメータを指定する必要があります。

リアルタイム画面共有（Web）

最終更新日：：2022-04-02 16:27:04

TRTC Web SDK画面共有のサポートレベルについて、ブラウザのサポート状況をご参照ください。また、SDKはTRTC.isScreenShareSupportedインターフェースを提供し、現在のブラウザが画面共有をサポートしているかどうかを判断します。
このドキュメントでは、以下のような異なるシーンで実装プロセスを説明します。

注意：

- Web端末ではセカンダリストリームの公開が現時点でサポートされていません。画面共有の公開はプライマリストリームの公開によって実現されます。リモート画面共有ストリームがWebユーザーからのものである場合は、RemoteStream.getType()は'main'を返し、通常、Web端末からの画面共有ストリームであることはuserIdによって識別されます。
- Native（iOS、Android、Mac、Windowsなど）端末ではセカンダリストリームの公開をサポートし、画面共有の公開はセカンダリストリームの公開によって実現されます。リモート画面共有ストリームがNativeユーザーからのものである場合は、RemoteStream.getType()は'auxiliary'を返します。

画面共有ストリームの作成と公開

注意：
画面共有ストリームを作成および公開するコードは、以下の順序で厳密に実行してください。

ステップ1：画面共有を行うクライアントオブジェクトの作成

通常、画面共有のクライアントオブジェクトであることを示すために、userIdには `share_` というプレフィックスを付けることをお勧めします。

```
const shareClient = TRTC.createClient({
mode: 'rtc',
sdkAppId,
userId, // 例：‘share_teacher’
userSig
});
// クライアントオブジェクトがルームに参加します
try {
```

```
await shareClient.join({ roomId });
// ShareClient join room success
} catch (error) {
// ShareClient join room failed
}
```

ステップ2：画面共有ストリームの作成

画面共有ストリームには、ビデオストリームとオーディオストリームが含まれます。このうち、オーディオストリームは、マイクオーディオまたはシステムオーディオに分けられます。

```
// Goodの正しい使い方
// 画面ビデオストリームのみの収集
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// or マイクオーディオおよび画面ビデオストリームの収集
const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// or システムオーディオおよび画面ビデオストリームの収集
const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });

// Badの間違った使い方
const shareStream = TRTC.createStream({ camera: true, screen: true });
// or
const shareStream = TRTC.createStream({ camera: true, screenAudio: true });
```

注意：

- audio属性とscreenAudio属性を同時にtrueに設定することはできません。camera属性とscreenAudio属性を同時にtrueに設定することはできません。screenAudioの詳細については、このドキュメントのパート5で説明します。
- camera属性とscreen属性を同時にtrueに設定することはできません。

ステップ3：画面共有ストリームの初期化

初期化時にブラウザはユーザーに画面共有の内容と権限を要求します。ユーザーが許可を拒否した場合、またはシステムがブラウザに画面共有の権限を付与しなかった場合は、コードは `NotReadableError` または `NotAllowedError` のエラーをキャッチします。この場合、ユーザーがブラウザ設定またはシステム設定で画面共有の権限をオンにするように指示する必要があります。

```
try {
await shareStream.initialize();
} catch (error) {
// 画面共有ストリームの初期化に失敗した場合、ユーザーに警告し、その後の入室公開プロセスを停止しま
```

```
す
switch (error.name) {
case 'NotReadableError':
// 現在のブラウザで画面の内容を取得できるようなシステムの確保にユーザーに通知します
return;
case 'NotAllowedError':
if (error.message.includes('Permission denied by system')) {
// 現在のブラウザで画面の内容を取得できるようなシステムの確保にユーザーに通知します
}else{
// ユーザーが画面共有を拒否/キャンセルします
}
return;
default:
// 画面共有ストリームの初期化中に不明なエラーが発生し、ユーザーにリトライを通知します
return;
}
}
```

ステップ4：画面共有ストリームの公開

ステップ1で作成されたクライアントオブジェクトにより公開されます。公開が成功すると、リモート端末で画面共有ストリームを受信できます。

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

完全コード

```
const shareClient = TRTC.createClient({
mode: 'rtc',
sdkAppId,
userId, // 例：‘share_teacher’
userSig
});
// クライアントオブジェクトがルームに参加します
try {
await shareClient.join({ roomId });
// ShareClient join room success
} catch (error) {
// ShareClient join room failed
}
// 画面ビデオストリームのみの収集
```

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// or マイクオーディオおよび画面ビデオストリームの収集
// const shareStream = TRTC.createStream({ audio: true, screen: true, userId });
// or システムオーディオおよび画面ビデオストリームの収集
// const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
try {
await shareStream.initialize();
} catch (error) {
// 画面共有ストリームの初期化に失敗した場合、ユーザーに警告し、その後の入室公開プロセスを停止します
switch (error.name) {
case 'NotReadableError':
// 現在のブラウザで画面の内容を取得できるようなシステムの確保にユーザーに通知します
return;
case 'NotAllowedError':
if (error.message.includes('Permission denied by system')) {
// 現在のブラウザで画面の内容を取得できるようなシステムの確保にユーザーに通知します
}else{
// ユーザーが画面共有を拒否/キャンセルします
}
return;
default:
// 画面共有ストリームの初期化中に不明なエラーが発生し、ユーザーにリトライを通知します
return;
}
}
try {
await shareClient.publish(shareStream);
} catch (error) {
// ShareClient failed to publish local stream
}
```

画面共有パラメータの設定

設定可能なパラメータには、解像度、フレームレートおよびコードレートがあります。必要に応じてsetScreenProfile()インターフェースを使用してprofileを指定できます。各profileは解像度、フレームレートおよびコードレートのセットに対応しています。SDKはデフォルトで'1080p'の構成を使用します。

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// setScreenProfile()はinitialize()の前に呼び出す必要があります。
shareStream.setScreenProfile('1080p');
await shareStream.initialize();
```

またはカスタマイズ解像度、フレームレートとビットレートを使用します。

```
const shareStream = TRTC.createStream({ audio: false, screen: true, userId });
// setScreenProfile()はinitialize()の前に呼び出す必要があります。
shareStream.setScreenProfile({ width: 1920, height: 1080, frameRate: 5, bitrate: 1600 /* kbps */
});
await shareStream.initialize();
```

画面共有属性推奨リスト：

profile	解像度（幅 × 高さ）	フレームレート（fps）	ビットレート (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：
想定外の問題の原因となる高すぎるパラメータ設定を避けるために、推奨されるパラメータで設定することをお勧めします。

画面共有の停止

```
// 画面共有クライアントによるストリームの公開解除
await shareClient.unpublish(shareStream);
// 画面共有ストリームをオフにする
shareStream.close();
// 画面共有クライアントの退室
await shareClient.leave();

// 上記の3つのステップは必須ではなく、シーンのニーズに応じて必要なコードを実行すればよい。通常は、入室したかどうか、ストリームを開示したかどうかの判断を追加する必要があります。より詳細なコード例については、[demoソースコード](https://github.com/LiteAVSDK/TRTC_Web/blob/main/base-js/js/share-client.js)をご参照ください。
```

またユーザーはブラウザに付属のボタンで画面共有を停止することもあるため、画面共有ストリームは画面共有停止イベントを傍受し、それに応じた処理を行う必要があります。

```
// 画面共有ストリームが画面共有停止イベントを傍受します
shareStream.on('screen-sharing-stopped', event => {
// 画面共有クライアントがプッシュを停止します
await shareClient.unpublish(shareStream);
// 画面共有ストリームをオフにする
shareStream.close();
// 画面共有クライアントの退室
await shareClient.leave();
});
```

カメラ動画と画面共有の同時公開

1つのClientでは、最大1チャネルのオーディオと1チャネルのビデオを同時に公開することができます。カメラビデオと画面共有を同時に公開するには、2つのClientを作成して、それらを「それぞれの役割」に任せる必要があります。
例えば、次のように2つのClientを作成します。

- **client**：ローカルのオーディオビデオストリームを公開し、shareClientを除くすべてのリモートストリームをサブスクライブします。
- **shareClient**：画面共有ストリームを公開し、リモートストリームをサブスクライブしません。

注意：

- 画面共有を行うshareClientは、自動サブスクリプションをオフにする必要があります。そうしないと、リモートストリームの重複サブスクリプションが発生します。APIの説明をご参照ください。
- ローカルのオーディオビデオストリームを公開するclientは、shareClientが公開するストリームのサブスクリプションを取り消す必要があります。そうしないと自分で自分をサブスクリプションするケースが発生する可能性があります。

サンプルコード：

```
const client = TRTC.createClient({ mode: 'rtc', sdkAppId, userId, userSig });
// shareClientのリモートストリーム自動サブスクリプションをオフにすると設定する必要があります。すなわちautoSubscribe：false
const shareClient = TRTC.createClient({ mode: 'rtc', sdkAppId, `share_${userId}`, userSig, autoSubscribe: false,});
```

```
// ローカルのオーディオビデオストリームの公開を行うclientは、shareClientが公開するストリームのサブスクリプションを取り消す必要があります。
client.on('stream-added', event => {
const remoteStream = event.stream;
const remoteUserId = remoteStream.getUserId();
if (remoteUserId === `share_${userId}`) {
// 自身の画面共有ストリームのサブスクリプションをキャンセル
client.unsubscribe(remoteStream);
}else{
//その他一般的なリモートストリームのサブスクリプション
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 });

// ... 関連コードの初期化や公開を省略し、必要に応じてストリームを公開すればよい。
```

画面共有のシステムサウンド収集

現時点でシステム音声のキャプチャはChrome M74+のみサポートしており、WindowsおよびChrome OS上ではシステム全体のオーディオをキャプチャすることができ、LinuxおよびMacではタブのオーディオのみキャプチャできます。その他のChromeのバージョン、その他のシステム、その他のブラウザはいずれもサポートしていません。

```
// 画面共有ストリームの作成screenAudioはtrueに設定してください。システムとマイクの音量の同時収集がサポートされていませんので、同時にaudio属性をtrueに設定しないでください
const shareStream = TRTC.createStream({ screenAudio: true, screen: true, userId });
await shareStream.initialize();
...
```

表示されるダイアログで「オーディオ共有」にチェックを入れると、公開されたストリームにシステムサウンドが付きます。

リアルタイム画面共有(Flutter)

最終更新日：：2021-10-13 15:43:06

Androidプラットフォームの場合

Tencent CloudのTRTCは、Androidシステムでの画面共有をサポートしており、現在のシステムの画面コンテンツは、TRTC SDKを介してルーム内の他のユーザーと共有されます。この機能については、次のような2つの注意点があります。

- TRTC Android版の画面共有は、デスクトップ版のように「サブチャネルの共有」をサポートしていません。従って、画面共有を開始するときは、あらかじめカメラのキャプチャを停止する必要があります。停止しない場合、衝突が生じます。
- Androidシステムのバックグラウンドアプリが引き続きCPUを使用すると、システムによって強制終了されやすくなり、画面共有自体が必然的にCPUを消費してしまいます。この矛盾するような衝突を解消するには、アプリで画面共有を開始するとともに、Androidシステムにフローティングウィンドウをポップアップする必要があります。Androidはフォアグラウンドを含むアプリプロセスを強制終了しないため、このソリューションによって、お客様のアプリはシステムに自動回収されることなく画面を共有し続けることができます。

画面共有を開始

Androidデバイスでの画面共有は、 `TRTCCloud` のstartScreenCapture()インターフェースを呼び出すだけで開始することができます。ただし、明瞭で安定した共有効果を実現するには、次の3つの点に注意する必要があります。

Activityの追加

次のactivityをmanifestファイルに貼り付けます（項目コードに存在する場合は追加する必要はありません）。

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

ビデオコーデックパラメータの設定

startScreenCapture()で最初のパラメータ `encParams` を設定することにより、画面共有のエンコード品質を指定することができます。 `encParams` をnullに指定した場合、SDKは以前に設定されたエンコードパラメータを自動的に使用します。推奨されるパラメータの設定は、次のとおりです。

パラメータ項目	パラメータ名	通常の推奨値	テキスト教育シナリオ

パラメータ項目	パラメータ名	通常の推奨値	テキスト教育シナリオ
解像度	videoResolution	1280 × 720	1920 × 1080
フレームレート	videoFps	10 FPS	8 FPS
最高ビットレート	videoBitrate	1600 kbps	2000 kbps
解像度アダプティブ	enableAdjustRes	NO	NO

説明：

- 画面共有されるコンテンツには通常、大幅な変更がなく、FPSを高く設定するのは経済的ではないため、10FPSを推奨します。
- 共有したい画面コンテンツに大量のテキストが含まれている場合、解像度とビットレートを適宜引き上げることができます。
- 最高ビットレート(videoBitrate)とは、画面が大きく変化したときの最高出力ビットレートのことです。画面コンテンツの変化が少ない場合、実際のエンコードビットレートは低くなります。

フローティングウィンドウのポップアップで強制終了を回避

Android 7.0以降のシステムから、バックグラウンドで実行されている通常のアプリプロセスに切り替わりますが、CPUのアクティビティがある場合、常にシステムによって強制終了されやすくなります。従って、アプリをバックグラウンドに切り替えてサイレントで画面共有すると、フローティングウィンドウのポップアップというソリューションによって強制終了を回避することができます。また、携帯電話の画面にフローティングウィンドウを表示することは、ユーザーが今、画面共有を行っていることをユーザーに知らせ、プライバシー情報の漏えい防止につながります。

方法：通常のフローティングウィンドウをポップアップする

「TencentMeeting」に似たミニフローティングウィンドウのポップアップは、サンプルコード tool.dart 中の実装を参考とすることができます。

```
//画面共有でミニフローティングウィンドウをポップアップした時、バックエンドに切り替えられたアプリケーションが強制終了されることを防止します。
static void showOverlayWindow() {
SystemWindowHeader header = SystemWindowHeader(
title: SystemWindowText(
text: 「画面共有中」, 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,
);
}
```

iOSプラットフォームの場合

- **アプリケーション内の共有**
  現在のアプリの画面のみを共有できます。この特性をサポートするには、iOSのバージョン13以降のOSが必要です。現在のアプリ以外の画面コンテンツを共有できないため、高度なプライバシー保護が必要なシーンに適しています。
- **アプリケーション間共有**
  AppleのReplaykitソリューションをベースとして、システム全体の画面コンテンツを共有できます。ただし、現在のアプリはExtension拡張コンポーネントを追加で提供する必要があるため、結合手順はアプリ内共有よりもやや多くなります。

> 注意：
> 注意すべき点としては、モバイル版のTRTC SDKは、デスクトップ版のように、「サブストリームの共有」をサポートしていません。これは、iOSとAndroidシステムは、どちらもバックグラウンドで実行されているアプリに対し、カメラの使用権を制限しているからです。従って、サブストリームの共有をサポートする意義はあまりありません。

方法1：iOSプラットフォームアプリケーション内の共有

アプリ内共有のソリューションは非常にシンプルです。TRTC SDKが提供するインターフェース startScreenCaptureを呼び出し、エンコードパラメータ `TRTCVideoEncParam` と `appGroup` を渡して `''` にセットするだけです。 `TRTCVideoEncParam` パラメータはnullに設定することができます。この場合、SDKは画面共有が開始される前のエンコードパラメータを引き続き使用します。

iOS画面共有に推奨されるエンコードパラメータは次のとおりです。

パラメータ項目	パラメータ名	通常の推奨値	テキスト教育シナリオ
解像度	videoResolution	1280 × 720	1920 × 1080
フレームレート	videoFps	10 FPS	8 FPS

パラメータ項目	パラメータ名	通常の推奨値	テキスト教育シナリオ
最高ビットレート	videoBitrate	1600 kbps	2000 kbps
解像度アダプティブ	enableAdjustRes	NO	NO

説明：

- 画面共有されるコンテンツには通常、大幅な変更がなく、FPSを高く設定するのは経済的ではないため、10FPSを推奨します。
- 共有したい画面コンテンツに大量のテキストが含まれている場合、解像度とビットレートを適宜引き上げることができます。
- 最高ビットレート(videoBitrate)とは、画面が大きく変化したときの最高出力ビットレートのことです。画面コンテンツの変化が少ない場合、実際のエンコードビットレートは低くなります。

方法2：iOSプラットフォームアプリケーション間の共有

サンプルコード

Githubの**trtc_demo/ios**ディレクトリに、アプリケーション間共有用のサンプルコードを設置しています。これには、次のようなテキストが含まれています。

```
├─── Broadcast.Upload //スクリーンキャプチャのプロセスBroadcast Upload Extensionコードの詳細は手順2をご参照ください。
|   ├─── Broadcast.Upload.entitlements //プロセス間通信の設定に使用されるAppGroup情報
|   ├─── Broadcast.UploadDebug.entitlements //プロセス間通信の設定に使用されるAppGroup情報（debug環境）
|   ├─── Info.plist
|   └─── SampleHandler.swift // システムからのスクリーンキャプチャデータを受信するために使用されます
├─── Resource // リソースファイル
├─── Runner // TRTC簡易化Demo
├─── TXLiteAVSDK_ReplayKitExt.framework //TXLiteAVSDK_ReplayKitExt SDK
```

READMEのガイドによって、このサンプルDemoを実行することができます。

結合手順

iOSシステムでアプリケーション間画面共有を行うには、Extensionスクリーンキャプチャプロセスを追加し、メインアプリプロセスと連携してプッシュを行う必要があります。Extensionスクリーンキャプチャプロセスは、システムによってスクリーンキャプチャが必要なときに作成され、システムが収集した画面イメージの受信を担当します。従って、次の事項を行う必要があります。

1. App Groupを作成し、XCodeにおいて設定を行います（オプション）。このステップは、Extensionスクリーンキャプチャプロセスが、同じメインアプリプロセスとプロセス間通信できるようにすることを目的としています。
2. お客様のプロジェクトにおいて、Broadcast Upload ExtensionのTargetを新規作成し、拡張コモジュール用としてカスタマイズされた `TXLiteAVSDK_ReplayKitExt.framework` をSDK圧縮パッケージに統合します。
3. メインアプリ側の受信ロジックと結合し、メインアプリにBroadcast Upload Extensionからのスクリーンキャプチャデータを待機させます。
4. `pubspec.yaml` ファイルを編集して `replay_kit_launcher` プラグインを導入し、TRTC Demo Screenのようにボタンをクリックすれば画面共有の呼び出しを実現します（オプション）。

```
# trtc sdkとreplay_kit_launcherの導入
dependencies:
tencent_trtc_cloud: ^0.2.1
replay_kit_launcher: ^0.2.0+1
```

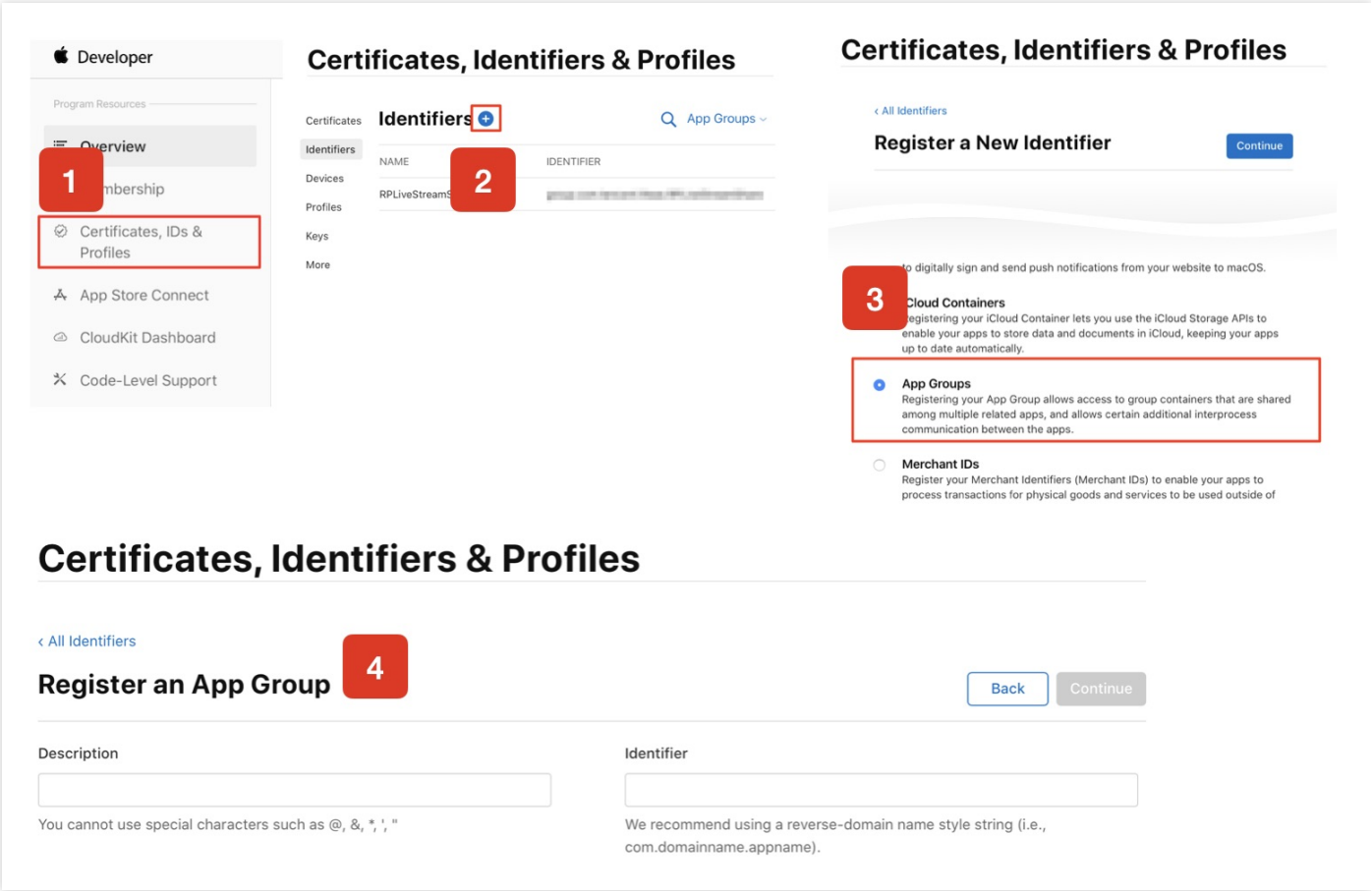
注意：
手順1をスキップした場合、すなわちApp Groupを設定しない場合は（インターフェースはnullを渡します）、画面共有は実行できますが、安定性が若干損なわれます。手順はやや多いですが、できる限り正しいApp Groupを設定して、画面共有機能の安定性を確保してください。

手順1：App Groupsの作成

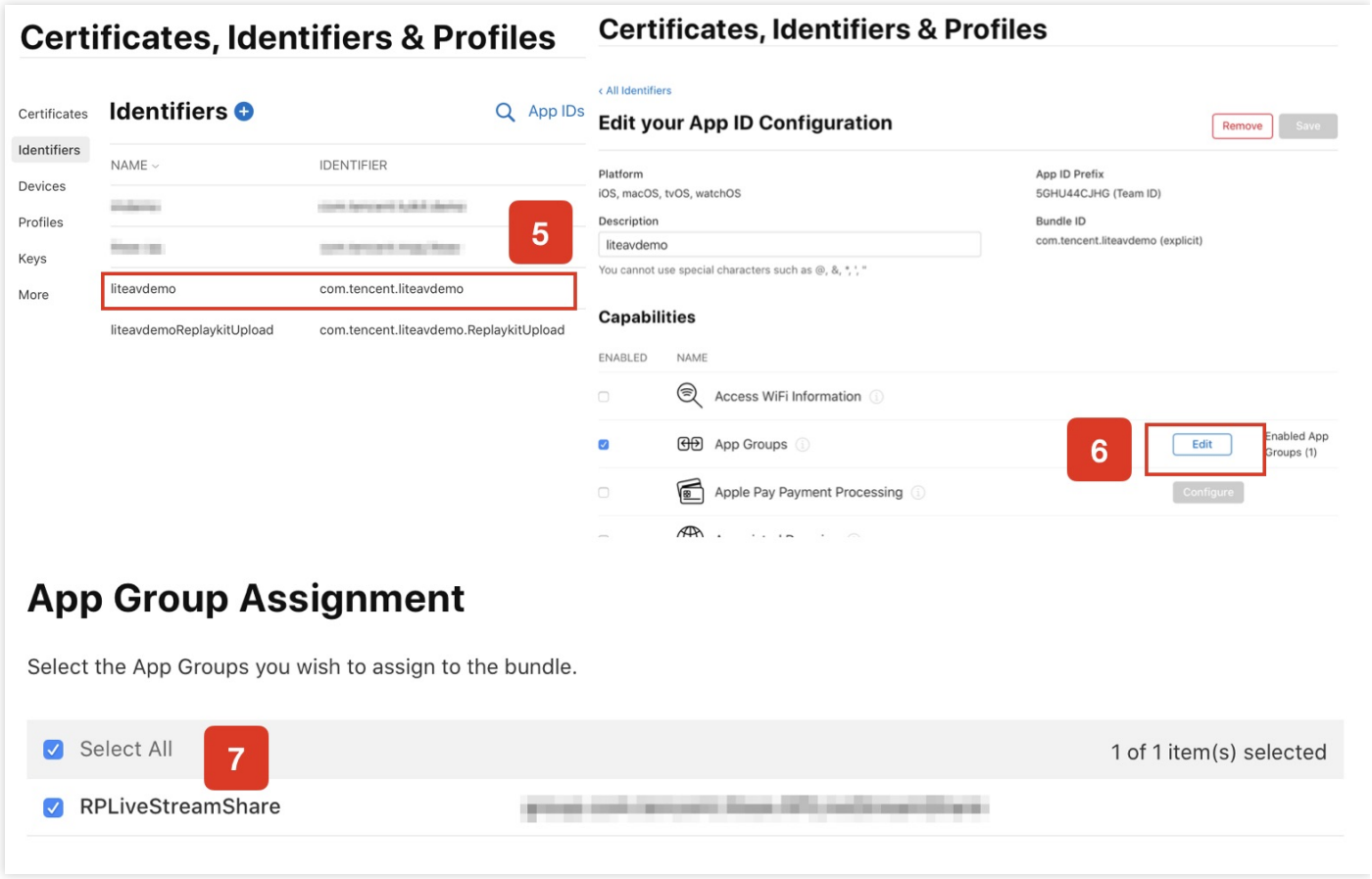
お客様のアカウントを使用して**https://developer.apple.com/**にログインし、次の操作を実行します。**完了後は、対応するProvisioning Profileを再ダウンロードする必要がありますので、ご注意ください。**

1. 【Certificates, IDs & Profiles】をクリックします。
2. 右側のインターフェースでプラス記号をクリックします。
3. 【App Groups】を選択して、【Continue】をクリックします。
4. ポップアップされたフォームにDescriptionとIdentifierを入力します。そのうちIdentifierは、インターフェースにおいて対応するAppGroupパラメータに渡す必要があります。完了したら、【Continue】をクリックしま

す。



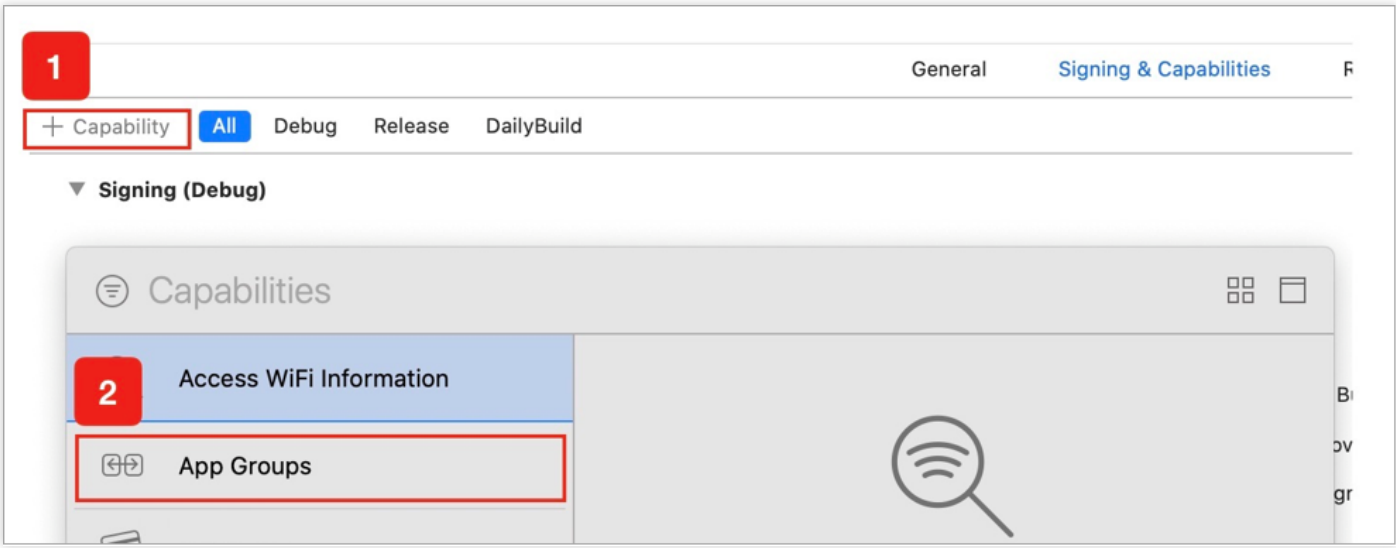
5. Identifierページに戻り、左上のメニューから【App IDs】を選択し、お客様のアプリIDをクリックします（メインアプリとExtensionのアプリIDにも同様の設定が必要です）。
6. 【App Groups】を選択し、【Edit】をクリックします。
7. ポップアップされたフォームからお客様が作成したApp Groupを選択し、【Continue】をクリックして編集ページに戻り、【Save】をクリックして保存します。



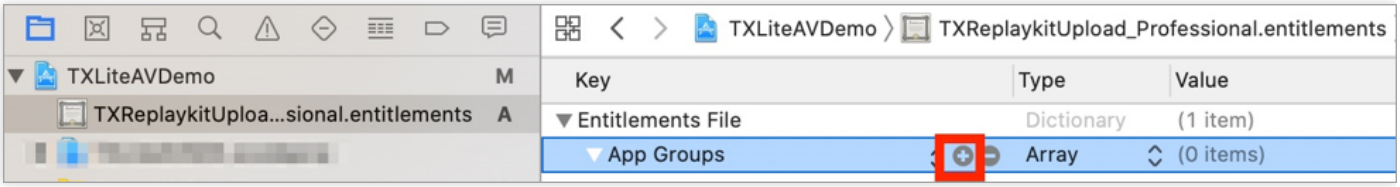
8. Provisioning Profileを再度ダウンロードし、XCodeに設定します。

手順2：Broadcast Upload Extensionの作成

1. Xcodeメニューの【File】 > 【New】 > 【Target...】を順にクリックし、【Broadcast Upload Extension】を選択します。
2. ポップアップされたダイアログボックスに関連情報を入力し、**【Include UI Extension】にはチェック"を入れずに**【Finish】をクリックして作成を完了します。
3. ダウンロードしたSDK圧縮パッケージのプロジェクトにTXLiteAVSDK_ReplayKitExt.frameworkをドラッグし、先ほど作成したTargetにチェックを入れます。
4. 新しく追加したTargetを選択し、【+ Capability】を順番にクリックして、【App Groups】をダブルクリックします。下図のとおりです。



操作が完了すると、下図のようにファイルリストに `Target名.entitlements` という名前のファイルが生成されます。このファイルを選択し、+記号をクリックして上記の手順のApp Groupを入力すればOKです。



5. メインアプリのTargetを選択し、**上記の手順に従って、メインアプリのTargetに同様の処理を行います。**
6. 新規作成したTargetにおいて、Xcodeは「SampleHandler.swift」という名前のファイルを自動的に作成し、それを次のコードに置き換えます。**コード内のAPPGROUPを先ほど作成したApp Group Identifierに変更する必要があります。**

```
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 = 「画面共有は終了しました」
break
case TXReplayKitExtReason.disconnected:
tip = 「アプリケーションは切断されました」
break
case TXReplayKitExtReason.versionMismatch:
tip = 「統合エラー（SDKバージョン番号が一致しません）」
break
default:
break
}

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

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

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

手順3：メインアプリ側の受信ロジックとの結合

以下の手順に従って、メインアプリ側の受信ロジックと結合します。すなわち、ユーザーが画面共有をトリガーする前に、メインアプリを「待機」状態にして、Broadcast Upload Extensionプロセスからスクリーンキャプチャデータをいつでも受信できるようにします。

1. TRTCCloudがカメラのキャプチャをオフにしていることを確認します。オフになっていない場合は、stopLocalPreview を呼び出して、カメラのキャプチャをオフにしてください。

2. startScreenCaptureメソッドを呼び出し、手順1で設定したAppGroupを渡して、SDKを「待機」状態にします。

3. ユーザーが画面共有をトリガーするまで待機します。手順4における「トリガーボタン」が実装されていない場合、ユーザーがiOSシステムのコントロールセンターにおいて、スクリーンキャプチャボタンを長押しして画面共有をトリガーする必要があります。

4. stopScreenCaptureインターフェースを呼び出すことにより、いつでも画面共有を停止できます。

```
// 画面共有を開始するには、APPGROUPを上記の手順で作成したApp Groupに置き換える必要があります
trtcCloud.startScreenCapture(
TRTCVideoEncParam(
videoFps: 10,
videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720,
videoBitrate: 1600,
videoResolutionMode: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT,
),
iosAppGroup,
);

// 画面共有を停止します
await trtcCloud.stopScreenCapture();


// 画面共有の開始イベントの通知は、TRTCCloudListenerを介して受信できます
onRtcListener(type, param){
if (type == TRTCCloudListener.onScreenCaptureStarted) {
//画面共有の開始
```

```
}
}
```

手順4：画面共有のトリガーボタンの追加（オプション）

手順3までに、ユーザーがコントロールセンターからスクリーンキャプチャボタンを長押しして、画面共有を手動で開始する必要があります。以下の方法で、TRTC Demo Screenのように、ボタンをクリックすることでトリガーする効果を実現できます。

1. `replay_kit_launcher` プラグインをご使用のプロジェクトに導入します。
2. インターフェースにボタンを設置し、ボタンの応答関数において `ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);` 関数を呼び出すと、画面共有機能を呼び出すことができます。

```
// カスタムボタンによる応答メソッド
onShareClick() async {
if (Platform.isAndroid) {
if (await SystemAlertWindow.requestPermissions) {
MeetingTool.showOverlayWindow();
}
} else {
//画面共有機能は実機でのみテストすることができます
ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);
}
}
```

画面共有の確認

- **Android/iOS画面共有の確認**
  ユーザーがAndroid / iOSを介して画面共有を実行する場合は、メインストリームを介して共有を実行することができます。ルームにいるその他ユーザーはTRTCCloudListener中のonUserVideoAvailable イベントを介してこの通知を受け取ります。
  画面共有を確認する場合は、startRemoteViewインターフェースを介してリモートユーザーのメインストリーム画面のレンダリングを起動することができます。

よくあるご質問

1つのルームで同時にいくつの画面を共有できますか。
現在、1つの TRTC オーディオ・ビデオルームで共有できる画面は1つだけです。

ライブストリーミングモード
ライブストリーミングクイックスタート (iOS&Mac)

最終更新日：：2022-03-09 18:04:28

ユースケース

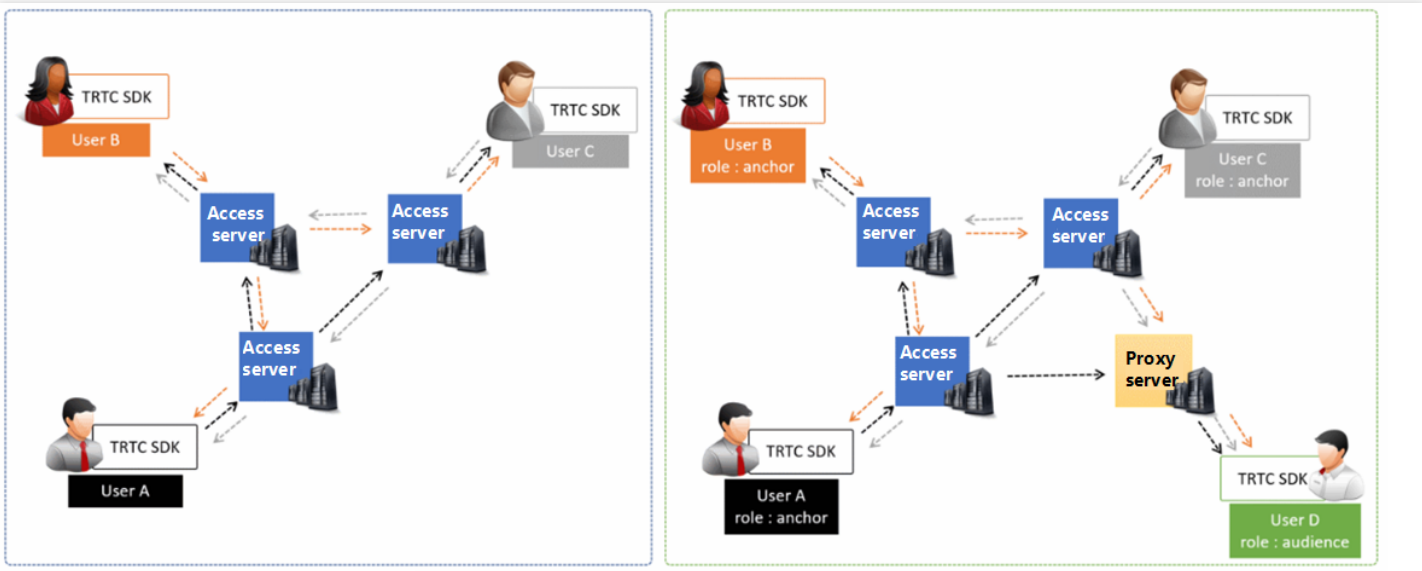
TRTCは、4種類の異なる入室モードをサポートしています。このうち、ビデオ通話（VideoCall）および音声通話（VoiceCall）を総称して通話モードといい、ビデオ・インタラクティブストリーミング（Live）およびボイス・インタラクティブストリーミング（VoiceChatRoom）を総称してライブストリーミングモードといいます。
ライブストリーミングモードでのTRTCは、1つのルームで最大10万人の同時接続をサポートし、300ms未満のマイク接続遅延、1000ms未満の視聴遅延およびマイクのオン・オフのスムーズな切り替え技術を備えています。低レイテンシーインタラクティブストリーム、10万人のインタラクティブ教室、ビデオ婚活、eラーニング、リモート研修、超大規模ミーティングなどのユースケースに適しています。

原理解析

TRTCクラウドサービスは、「インターフェースモジュール」および「プロキシモジュール」という2種類の異なるタイプのサーバーノードから構成されています。

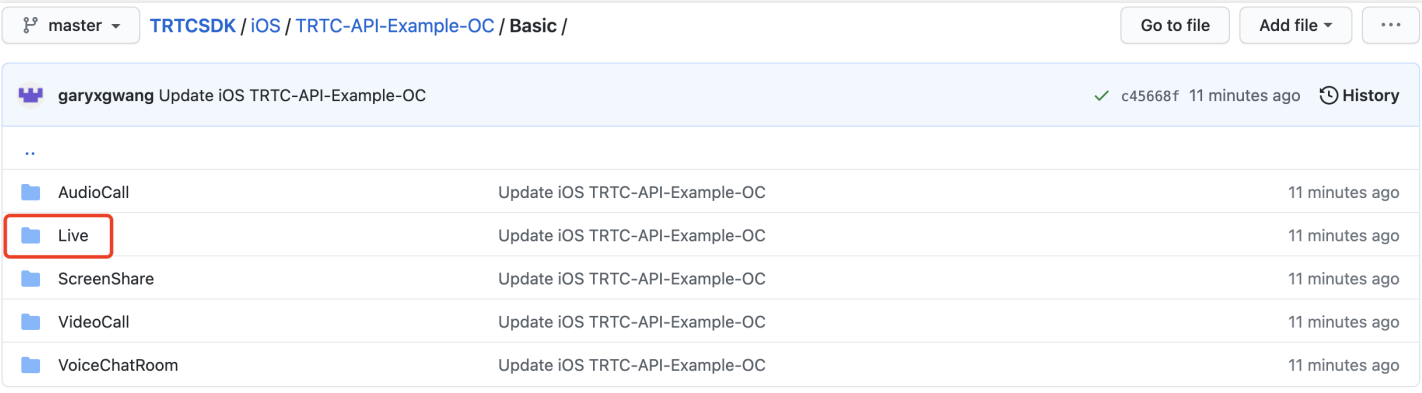
- **インターフェースモジュール**
  この種のノードは、最も良質の回線および高性能の機器を採用しており、エンドツーエンドの低遅延マイク接続通話の処理に優れています。
- **プロキシモジュール**
  この種のノードは、通常の回線および性能も一般的な機器を採用しており、同時進行性の高いプルストリーミング再生ニーズの処理にすぐれています。

ライブストリーミングモードでは、TRTCはロールのコンセプトを導入し、ユーザーは「キャスター」および「視聴者」の2種類のロールに分けられ、「キャスター」はインターフェースモジュールに配分され、「視聴者」はプロキシモジュールに分配されます。同一ルームの視聴者の上限は10万人です。
「視聴者」をマイク・オンにしたい場合、まずロール（switchRole）を「キャスター」に切り替えると発言できます。ロールを切り替えることで、ユーザーをプロキシモジュールからインターフェースモジュールに移動させ、TRTC特有の低遅延視聴技術およびスムーズなマイクのオン/オフ切替技術によって、すべての切り替え時間を非常に短くすることができます。



サンプルコード

Github にログインし、本ファイルに関連するサンプルコードを取得することができます。



Githubへのアクセスが遅い場合は、TXLiteAVSDK_TRTC_iOS_latest.zipを直接ダウンロードすることもできます。

操作手順

手順1：SDKの統合

以下の方式を選択して **TRTC SDK** をプロジェクトに統合することができます。

方法1： CocoaPodsを使用して統合

1. **CocoaPods**をインストールします。具体的な操作は CocoaPods公式サイトインストールの説明をご参照ください。

2. 現在のプロジェクトのルートディレクトリの `Podfile` ファイルを開き、以下のコンテンツを追加します。

> 説明：
> このディレクトリに `Podfile` ファイルがない場合は、まず `pod init` コマンドを実行しファイルを新規作成してから、以下の内容を追加してください。

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

3. 以下のコマンドを実行して **TRTC SDK** をインストールします。

```
pod install
```

インストールが成功したら、現在のプロジェクトのルートディレクトリに **xcworkspace** ファイルが生成されます。
4. 新規作成した **xcworkspace** ファイルを開けばOKです。

方法2：ZIPパッケージをダウンロードして手動で統合

一時的にCocoaPods環境をインストールしたくない場合、またはインストール済みだがCocoaPodsライブラリへのアクセスがやや遅い場合は、 ZIP圧縮パッケージを直接ダウンロードして、クイックインテグレーション(iOS)を参照してSDKをプロジェクトに統合することができます。

手順2：メディアデバイスの権限追加

`Info.plist` ファイルにカメラおよびマイクのアクセス許可のリクエストを追加します。

Key	Value
Privacy - Camera Usage Description	カメラ使用の許可をリクエストする理由を記述。例えば、ビデオチャットでビデオを表示するには、カメラへのアクセスが必要です

Key	Value
Privacy - Microphone Usage Description	マイク使用の許可をリクエストする理由を記述。例えば、チャットで音声を送信するには、マイクへのアクセスが必要です

手順3：SDKインスタンスを初期化し、イベントコールバックを監視する

1. sharedInstance() インターフェースを使用して `TRTCCloud` インスタンスを作成します。

```
// trtcCloudインスタンスを作成
_trtcCloud = [TRTCCloud sharedInstance];
_trtcCloud.delegate = self;
```

2. `delegate` 属性を設定しイベントのコールバックを登録し、関連イベントおよびエラー通知をモニタします。

```
// エラー通知は監視すべきもので、捕捉してユーザーに通知する必要があります
- (void)onError:(TXLiteAVError)errCode errMsg:(NSString *)errMsg extInfo:(NSDictionary *)extIn
fo {
if (ERR_ROOM_ENTER_FAIL == errCode) {
[self toastTip:@"入室失敗"];
[self.trtcCloud exitRoom];
}
}
```

手順4：入室パラメータTRTCParamsを組み立てる

enterRoom()インターフェースを呼び出す時に、キーパラメータTRTCParamsを入力する必要があります。このパラメータに含まれる入力必須のフィールドは下表に示すとおりです。

パラメータ名	フィールドタイプ	補足説明	記入例
sdkAppId	数字	アプリケーションID。TRTCコンソールでSDKAppIDを表示できます。	1400000123
userId	文字列	アルファベットの大文字、小文字（a-z、A-Z）、数字（0-9）、下線およびハイフンのみを許可。ビジネスの実際のアカウントシステムを組み合わせて設定することをお勧めします。	test_user_001
userSig	文字列	userIdを基にuserSigを計算します。計算方法はUserSigの計算方法をご参照ください。	eJyrVareCeYrSy1SsII...

パラメータ名	フィールドタイプ	補足説明	記入例
roomId	数字	数字タイプのルームナンバー。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。	29834

注意：

- TRTCは、2つの同じuserIdによる同時入室をサポートしていません。同時に入室した場合、相互に干渉します。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

手順5：キャスター端末でのカメラのプレビューとマイク集音を起動する

1. キャスター側は、startLocalPreview()を呼び出すと、ローカルのカメラのプレビューを起動することができ、SDKがシステムにカメラの使用許可をリクエストします。
2. キャスター側は、setLocalViewFillMode()を呼び出すと、ローカルのビデオ画面の表示モードを設定することができます。

- Fillモードは塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
- Fitモードは適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

3. キャスター側は、setVideoEncoderParam()インターフェースを呼び出すと、ローカルビデオのエンコードパラメータを設定できます。このパラメータにより、ルーム内の他のユーザーが視聴する際の画面の画質が決定されます。
4. キャスター側は、startLocalAudio()を呼び出すと、マイクを起動でき、SDKがシステムにマイクの使用許可をリクエストします。

```
//サンプルコード：ローカルのオーディオ・ビデオストリーミングの公開
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];

//ローカルビデオコーデックパラメータの設定
TRTCVideoEncParam *encParams = [TRTCVideoEncParam new];
encParams.videoResolution = TRTCVideoResolution_640_360;
encParams.videoBitrate = 550;
```

```
encParams.videoFps = 15;

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

手順6：キャスター端末により美顔エフェクトを設定する

1.キャスター側は、getBeautyManager()を呼び出すと、美顔設定インターフェース[TXBeautyManagerを取得できます。
2. キャスター側は、setBeautyStyle()を呼び出すと、美顔スタイルを設定できます。

- Smooth：スムース。明らかな効果が感じられます。インフルエンサーのスタイルに近づけます。
- Nature：ナチュラル。美肌補正のアルゴリズムは顔の詳細な質感を維持し、より自然な感じになります。
- Pitu ：エンタープライズ版 のみサポートしています。

3. キャスター側は、setBeautyLevel()を呼び出すと、美肌補正レベルを設定できます。通常、5の設定でOKです。
4. キャスター側は、setWhitenessLevel()を呼び出すと、美白レベルを設定できます。通常、5の設定でOKです。
5. iPhoneのカメラの色調はデフォルトだと黄色味がかっているため、setFilter()を呼び出して、キャスターに美白特殊効果を追加することをお勧めします。美白特殊効果に対応するフィルターのファイルのダウンロードアドレスは、次となります。フィルターファイル。

手順7：キャスター端末からルームを新規作成し、プッシュを開始する

1. キャスター側は、TRTCParamsのフィールド `role` を ** `TRTCRoleType.anchor` **に設定します。これは現在のユーザーのロールがキャスターであることを表します。
2. キャスター側は、enterRoom()を呼び出すと、TRTCParamsパラメータのフィールド `roomId` の値をルームナンバーとするオーディオ・ビデオルームを作成し、** `appScene` **パラメータを指定することができます。

- TRTCAppScene.LIVE：ビデオ・インタラクティブストリーミングモード。ここではこのモードを例として取り上げます。
- TRTCAppScene.voiceChatRoom：ボイス・インタラクティブストリーミングモード。

3. ルームの新規作成が成功すると、キャスター側は、音声ビデオデータのエンコードおよび伝送フローを開始します。同時にSDKがonEnterRoom(result) イベントをコールバックします。パラメータ `result` が0より大きいときは入室成功を意味し、具体的な数値は入室してからの消費時間であり、単位はミリ秒（ms）です； `result` が0より小さい時は入室失敗を意味し、具体的な数値は入室失敗のエラーコードになります。

```
- (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:@"入室成功"];
}else{
[self toastTip:@"入室失敗"];
}
}
```

手順8：視聴者が入室しライブストリーミングを視聴する

1. 視聴者側は、TRTCParamsのフィールド `role` を** `TRTCRoleType.audience` **に設定します。これは現在のユーザーのロールが視聴者であることを表します。
2. 視聴者側は、enterRoom()を呼び出すと、TRTCParamsパラメータのフィールド `roomId` が示すオーディオ・ビデオルームに入室し、** `appScene` **パラメータを指定することができます。

- TRTCAppScene.LIVE：ビデオ・インタラクティブストリーミングモード。ここではこのモードを例として取り上げます。
- TRTCAppScene.voiceChatRoom：ボイス・インタラクティブストリーミングモード。

3. キャスター画面の視聴：

- 視聴者側が事前にキャスターのuserIdを知っている場合は、入室に成功した後、直接キャスターの `userId` を使用してstartRemoteView(userId, view: view)を呼び出せば、キャスターの画面を表示することができます。
- 視聴者側がキャスターのuserIdを知らない場合は、視聴者側が、入室に成功するとonUserVideoAvailable()のイベント通知を受信しますので、コールバックにより受け取ったキャスター `userId` を使用して、startRemoteView(userId, view: view)を呼び出せば、キャスターの画面を表示することができます。

手順9：視聴者とキャスターとのマイク接続

1. 視聴者側がswitch(TRTCRoleType.TRTCRoleAnchor)を呼び出し、ロールをキャスター（TRTCRoleType.TRTCRoleAnchor）に切り替えます。
2. 視聴者側がstartLocalPreview()を呼び出すと、ローカルの画面をアクティブにすることができます。
3. 視聴者側がstartLocalAudio()を呼び出すと、マイクの集音が開始されます。

```
//サンプルコード：視聴者マイク・オン
[self.trtcCloud switchRole:TRTCRoleAnchor];
[self.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
[self.trtcCloud startLocalPreview:_isFrontCamera view:self.view];

//サンプルコード：視聴者マイク・オフ
[self.trtcCloud switchRole:TRTCRoleAudience];
[self.trtcCloud stopLocalAudio];
[self.trtcCloud stopLocalPreview];
```

手順10：キャスター間でルーム間マイク接続PKを行う

TRTCでは、異なるオーディオ・ビデオルームにいる2人のキャスターが当初のライブストリーミングルームを退出しない場合にも、「ルーム間通話」機能によってマイク接続通話機能をプルし、「ルーム間マイク接続PK」を行うことができます。

1.キャスターAが、connectOtherRoom()インターフェースを呼び出します。インターフェースパラメータは現在、JSON形式を採用しており、キャスターBの `roomId` と `userId` を接合して、形式が `{"roomId": "978","userId": "userB"}` となるパラメータをインターフェース関数に渡す必要があります。
2. クロスルームに成功すると、キャスターAは onConnectOtherRoom()のイベントコールバックを受け取ります。同時に、2つのライブストリーミングルームのすべてのユーザーがonUserVideoAvailable()とonUserAudioAvailable()のイベント通知を受け取ります。
例えば、ルーム「001」のキャスターAがルーム「002」のキャスターBと `connectOtherRoom()` を介してルーム間通話をする場合、ルーム「001」のユーザーはキャスターBの `onUserVideoAvailable(B, available: true)` コールバックと `onUserAudioAvailable(B, available: true)` コールバックを受信します。ルーム「002」のユーザーはキャスターAの `onUserVideoAvailable(A, available: true)` コールバックと `onUserAudioAvailable(A, available: true)` コールバックを受信します。
3. 2つのルームにいるユーザーは、startRemoteView(userId, view: view)を呼び出すことで、もう一方のルームのキャスターの画面を表示することができ、音声が自動再生されます。

```
//サンプルコード：ルーム間マイク接続PK
NSMutableDictionary * jsonDict = [[NSMutableDictionary alloc] init];
[jsonDict setObject:@([_otherRoomIdTextField.text intValue]) forKey:@"roomId"];
[jsonDict setObject:_otherUserIdTextField.text forKey:@"userId"];
NSData* jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict options:NSJSONWritingPrettyPr
inted error:nil];
NSString* jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
[self.trtcCloud connectOtherRoom:jsonString];
```

手順11：現在のルームから退出する

exitRoom()メソッドを呼び出してルームを退出します。SDKは退室する時に、カメラ、マイクなどのハードウェアデバイスを停止してリリースする必要があるたため、退室の動作は瞬時に完了するものではなく、onExitRoom()のコールバックを受信してはじめて、実際の退室操作が完了します。

```
// 退室を呼び出した後は、onExitRoomイベントのコールバックをお待ちください
[self.trtcCloud exitRoom];

- (void)onExitRoom:(NSInteger)reason {
NSLog(@"ルームから退出: reason: %ld", reason)
}
```

注意：
Appの中で多くの音声ビデオのSDKを同時に統合した場合は、`onExitRoom`コールバックを受信してからその他の音声ビデオSDKを起動してください。そうしない場合は、ハード上の占有問題が生じることがあります。

ライブストリーミングクイックスタート (Android)

最終更新日：：2022-03-14 16:05:25

ユースケース

TRTCは、4種類の異なる入室モードをサポートしています。このうち、ビデオ通話（VideoCall）および音声通話（VoiceCall）を総称して通話モードといい、ビデオ・インタラクティブストリーミング（Live）およびボイス・インタラクティブストリーミング（VoiceChatRoom）を総称してライブストリーミングモードといいます。
ライブストリーミングモードでのTRTCは、1つのルームで最大10万人の同時接続をサポートし、300ms未満のマイク接続遅延、1000ms未満の視聴遅延およびマイクのオン・オフのスムーズな切り替え技術を備えています。
低レイテンシーインタラクティブストリーム、10万人のインタラクティブ教室、ビデオ婚活、eラーニング、リモート研修、超大規模ミーティングなどのユースケースに適しています。

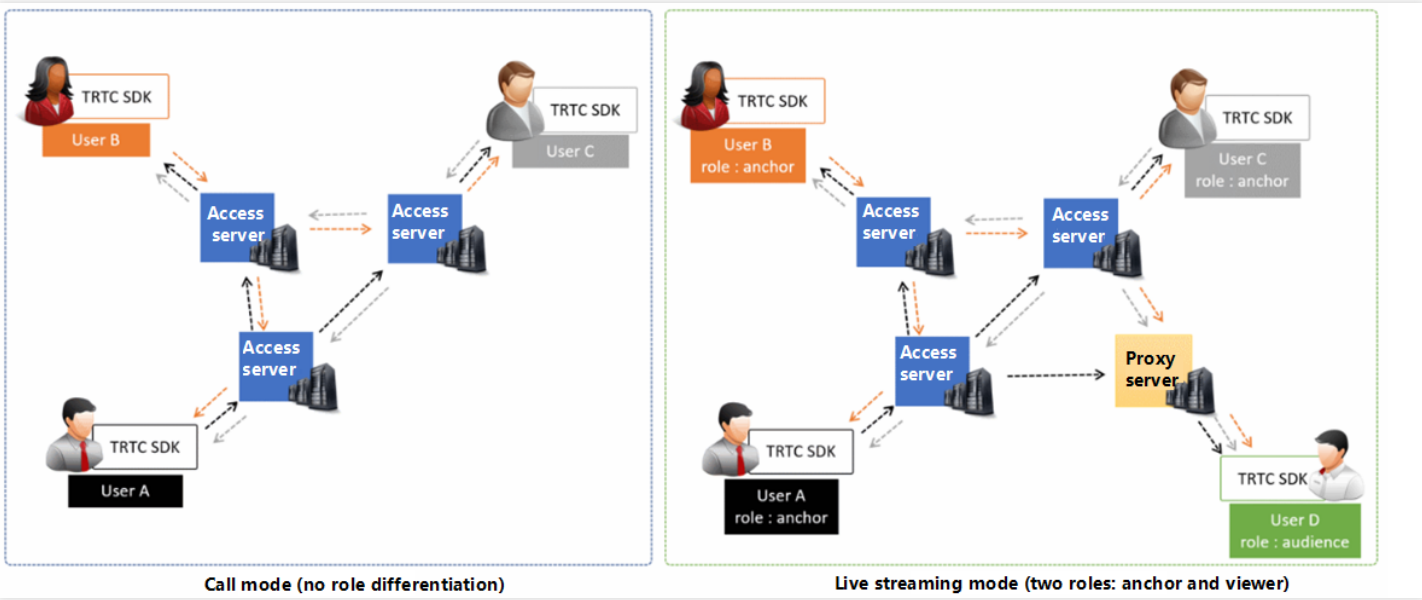
原理解析

TRTCクラウドサービスは、「インターフェースモジュール」および「プロキシモジュール」という2種類の異なるタイプのサーバーノードから構成されています。

- **インターフェースモジュール**
  この種のノードは、最も良質の回線および高性能の機器を採用しており、エンドツーエンドの低遅延マイク接続通話の処理に優れています。
- **プロキシモジュール**
  この種のノードは、通常の回線および性能も一般的な機器を採用しており、同時進行性の高いプルストリーミング再生ニーズの処理にすぐれています。

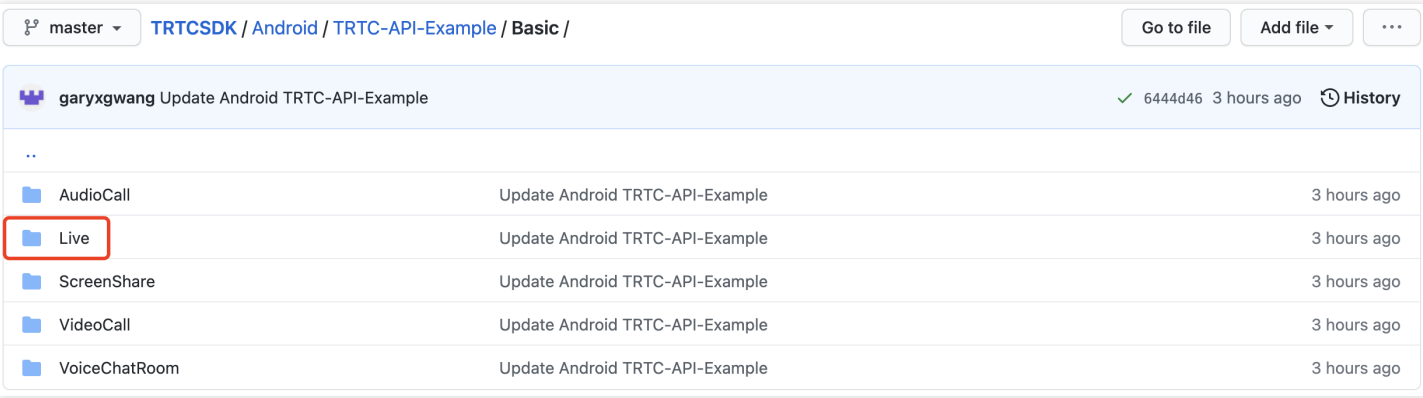
ライブストリーミングモードでは、TRTCはロールのコンセプトを導入し、ユーザーは「キャスター」および「視聴者」の2種類のロールに分けられ、「キャスター」はインターフェースモジュールに配分され、「視聴者」はプロキシモジュールに分配されます。同一ルームの視聴者の上限は10万人です。
「視聴者」をマイク・オンにしたい場合、まずロール（switchRole）を「キャスター」に切り替えると発言できます。ロールを切り替えることで、ユーザーをプロキシモジュールからインターフェースモジュールに移動させ、TRTC特有の低遅延視聴技術およびスムーズなマイクのオン/オフ切替技術によって、すべての切り替え時間を

非常に短くすることができます。



サンプルコード

Github にログインし、本ファイルに関連するサンプルコードを取得することができます。



説明：
Githubへのアクセスが遅い場合は、 TXLiteAVSDK_TRTC_Android_latest.zipを直接ダウンロードすることもできます。

操作手順

手順1：SDKの統合

以下の方式を選択して **TRTC SDK** をプロジェクトに統合することができます。

方法1：自動ロード（aar）

TRTC SDKは、mavenCentralライブラリにリリースされています。更新を自動的にダウンロードするようにgradleを構成することで自動でダウンロード、更新できます。
Android Studioを使用して、SDKを統合予定のプロジェクト（TRTC-API-Exampleは統合が完了済み、サンプルコードは参照用として提供）を開き、その後簡単な手順で `app/build.gradle` ファイルを修正するだけで、SDKの統合を完了できます。

1. dependenciesの中にTRTCSDKの依存を追加します。

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

2. defaultConfigでAppが使用するCPUアーキテクチャを指定します。

> 説明：
> 現在 TRTC SDKは、armeabi、armeabi-v7a、arm64-v8aをサポートしています。

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

3. **Sync Now**をクリックし、SDKを同期します。
   mavenCentralへのネットワーク接続に問題がない場合、SDKは自動的にダウンロードされ、プロジェクトに統合されます。

方法2：ZIPパッケージをダウンロードして手動で統合

ZIP圧縮パッケージを直接ダウンロードして、 クイックインテグレーション(Android)を参照してSDKをプロジェクトに統合することができます。

手順2： App権限の設定

`AndroidManifest.xml` ファイルにカメラ、マイクおよびネットワークのアクセス許可のリクエストを追加します。

```
<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" />
```

手順3：SDKインスタンスを初期化し、イベントコールバックを監視する

1. sharedInstance() インターフェースを使用して `TRTCCloud` インスタンスを作成します。

```
// trtcCloudインスタンスを作成
mTRTCCloud = TRTCCloud.sharedInstance(getApplicationContext());
mTRTCCloud.setListener(new TRTCCloudListener());
```

2. `setListener` 属性を設定しイベントのコールバックを登録し、関連イベントおよびエラー通知をモニタします。

```
// エラー通知のモニタ。エラー通知は、 SDK が動作を継続できないことを示します
@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();
}
}
}
```

手順4：入室パラメータTRTCParamsを組み立てる

enterRoom()インターフェースを呼び出すとき、キーパラメータTRTCParamsを入力する必要があります。このパラメータに含まれる入力必須のフィールドは下表に示すとおりです。

パラメータ名	フィールドタイプ	補足説明	記入例
sdkAppId	数字	アプリケーションID。TRTCコンソールでSDKAppIDを表示できます。	1400000123
userId	文字列	アルファベットの大文字、小文字（a-z、A-Z）、数字（0-9）、下線およびハイフンのみを許可。ビジネスの実際のアカウントシステムを組み合わせて設定することをお勧めします。	test_user_001
userSig	文字列	userIdを基にuserSigは計算されます。計算方法はUserSigの計算方法 をご参照ください。	eJyrVareCeYrSy1SsII...
roomId	数字	数字タイプのルームナンバー。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。	29834

注意：

- TRTCは、2つの同じuserIdによる同時入室をサポートしていません。同時に入室した場合、相互に干渉します。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

手順5：キャスター端末でのカメラのプレビューとマイク集音を起動する

1. キャスター側は、startLocalPreview()を呼び出すと、ローカルのカメラのプレビューを起動することができ、SDKがシステムにカメラの使用許可をリクエストします。
2. キャスター側は、setLocalViewFillMode()を呼び出すと、ローカルのビデオ画面の表示モードを設定することができます。

- Fillモードは塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
- Fitモードは適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。

3. キャスター側は、setVideoEncoderParam()インターフェースを呼び出すと、ローカルビデオのエンコードパラメータを設定できます。このパラメータにより、ルーム内の他のユーザーが視聴する際の画面の画質が決定されます。
4. キャスター側は、startLocalAudio()を呼び出すと、マイクを起動でき、SDKがシステムにマイクの使用許可をリクエストします。

```
//サンプルコード：ローカルのオーディオ・ビデオストリーミングの公開
mTRTCCloud.setLocalViewFillMode(TRTC_VIDEO_RENDER_MODE_FIT);
mTRTCCloud.startLocalPreview(mIsFrontCamera, localView);
//ローカルビデオコーデックパラメータの設定
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();
```

手順6：キャスター端末により美顔エフェクトを設定する

1. キャスター側は、getBeautyManager()を呼び出すと、美顔設定インターフェースTXBeautyManagerを取得できます。
2. キャスター側は、setBeautyStyle()を呼び出すと、美顔スタイルを設定できます。

- Smooth：スムース。明らかな効果が感じられます。インフルエンサーのスタイルに近づけます。
- Nature：ナチュラル。美肌補正のアルゴリズムは顔の詳細な質感を維持し、より自然な感じになります。
- Pitu ：エンタープライズ版 のみサポートしています。

3. キャスター側は、setBeautyLevel()を呼び出すと、美肌補正レベルを設定できます。通常、5の設定でOKです。
4. キャスター側は、setWhitenessLevel()を呼び出すと、美白レベルを設定できます。通常、5の設定でOKです。

手順7：キャスター端末からルームを新規作成し、プッシュを開始する

1. キャスター側は、TRTCParamsのフィールド `role` を** `TRTCCloudDef.TRTCRoleAnchor` **に設定します。これは現在のユーザーのロールがキャスターであることを表します。
2. キャスター側は、enterRoom()を呼び出すと、TRTCParamsパラメータのフィールド `roomId` の値をルームナンバーとするオーディオ・ビデオルームを作成し、** `appScene` **パラメータを指定することができます。

- TRTCCloudDef.TRTC_APP_SCENE_LIVE：ビデオ・インタラクティブストリーミングモード。ここではこのモードを例にします。
- TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM：ボイス・インタラクティブストリーミングモード。

3. ルームの新規作成が成功すると、キャスター側は、音声ビデオデータのエンコードおよび伝送フローを開始します。同時にSDKがonEnterRoom(result)イベントをコールバックします。パラメータ `result` が0より大きいときは入室成功を表し、具体的な数値は入室のために消費した時間になります。単位はミリ秒（ms）です。 `result` が0より小さい時は入室失敗を表し、具体的な数値は入室失敗のエラーコードになります。

```
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("入室成功，総消費時間[\(result)]ms")
}else{
toastTip("入室失敗，エラーコード[\(result)]")
}
}
```

手順8：視聴者が入室しライブストリーミングを視聴する

1. 視聴者側は、TRTCParamsのフィールド `role` を** `TRTCCloudDef.TRTCRoleAudience` **に設定します。これは現在のユーザーのロールが視聴者であることを表します。
2. 視聴者側は、enterRoom()を呼び出すと、TRTCParamsパラメータの `roomId` が示すオーディオ・ビデオルームに入室し、** `appScene` **パラメータを指定することができます。

- TRTCCloudDef.TRTC_APP_SCENE_LIVE：ビデオ・インタラクティブストリーミングモード。ここではこのモードを例にします。
- TRTCCloudDef.TRTC_APP_SCENE_VOICE_CHATROOM：ボイス・インタラクティブストリーミングモード。

3. キャスター画面の視聴：

- 視聴者側が事前にキャスターのuserIdを知っている場合は、入室に成功した後、直接キャスターの `userId` を使用してstartRemoteView(userId, view)を呼び出せば、キャスターの画面を表示することができます。
- 視聴者側がキャスターのuserIdを知らない場合、視聴者側が、入室に成功すると、onUserVideoAvailable()イベント通知を受信しますので、コールバックにより受け取ったキャスター `userId` を使用してstartRemoteView(userId, view)を呼び出せば、キャスターの画面を表示することができます。

手順9：視聴者とキャスターとのマイク接続

1. 視聴者側がswitchRole(TRTCCloudDef.TRTCRoleAnchor)を呼び出し、ロールをキャスター（TRTCCloudDef.TRTCRoleAnchor）に切り替えます。
2. 視聴者側がstartLocalPreview()を呼び出すと、ローカルの画面をアクティブにすることができます。
3. 視聴者側がstartLocalAudio()を呼び出すと、マイクの集音が開始されます。

```
//サンプルコード：視聴者マイク・オン
mTrtcCloud.switchRole(TRTCCloudDef.TRTCRoleAnchor);
mTrtcCloud.startLocalAudio();
mTrtcCloud.startLocalPreview(mIsFrontCamera, localView);
//サンプルコード：視聴者マイク・オフ
mTrtcCloud.switchRole(TRTCCloudDef.TRTCRoleAudience);
mTrtcCloud.stopLocalAudio();
mTrtcCloud.stopLocalPreview();
```

手順10：キャスター間でルーム間マイク接続PKを行う

TRTCでは、異なるオーディオ・ビデオルームにいる2人のキャスターが当初のライブストリーミングルームを退出しない場合にも、「ルーム間通話」機能によってマイク接続通話機能をプルし、「ルーム間マイク接続PK」を行うことができます。

1. キャスターAが、connectOtherRoom()インターフェースを呼び出します。インターフェースパラメータは現在JSON形式を採用しており、キャスターBの `roomId` と `userId` を接合して `{"roomId": "978","userId": "userB"}` の形式にしたパラメータをインターフェース関数に渡す必要があります。
2. クロスルームに成功すると、キャスターAはonConnectOtherRoom()のイベントコールバックを受け取ります。同時に、2つのライブストリーミングルームのすべてのユーザーがonUserVideoAvailable()とonUserAudioAvailable()のイベント通知を受け取ります。
   例えば、ルーム「001」のキャスターAがルーム「002」のキャスターBと `connectOtherRoom()` を介してルーム間通話をする場合、ルーム「001」のユーザーはキャスターBの `onUserVideoAvailable(B, true)` コールバックと `onUserAudioAvailable(B, true)` コールバックを受信します。ルーム「002」のユーザーはキャスターAの `onUserVideoAvailable(A, true)` コールバックと `onUserAudioAvailable(A, true)` コールバックを受信します。
3. 2つのルームにいるユーザーは、startRemoteView(userId, view)を呼び出すことで、もう一方のルームのキャスターの画面を表示することができ、音声が自動的に再生されます。

```
//サンプルコード：ルーム間マイク接続PK
mTRTCCloud.ConnectOtherRoom(String.format("{¥"roomId¥":%s,¥"userId¥":¥"%s¥"}", roomId, usernam
e));
```

手順11：現在のルームから退出する

exitRoom()メソッドを呼び出してルームを退出します。SDKは退室する時に、カメラ、マイクなどのハードウェアデバイスを停止してリリースする必要があるため、退室の動作は瞬時に完了するものではなく、onExitRoom()のコールバックを受信してはじめて、実際に退室操作を完了したことになります。

```
// 退室を呼び出した後は、onExitRoomイベントのコールバックをお待ちください
mTRTCCloud.exitRoom()
@Override
public void onExitRoom(int reason) {
Log.i(TAG, "onExitRoom: reason = " + reason);
}
```

注意：
Appの中で多くの音声ビデオのSDKを同時に統合した場合は、`onExitRoom`コールバックを受信してからその他の音声ビデオSDKを起動してください。そうしない場合は、ハード上の占有問題が生じることがあります。

ライブストリーミングクイックスタート (Windows)

最終更新日：：2022-02-14 14:53:23

ドキュメントガイド

ここでは主に、TRTC SDKをベースとしてビデオ・マイク接続をサポートするだけでなく、万人単位に及ぶ同時視聴をサポートするオンラインライブストリーミング機能についてご紹介します。本稿では最もよく用いられるインターフェースのみをリストアップしています。インターフェース関数に関する詳しい情報をご希望の場合、APIドキュメントをご参照ください。

サンプルコード

属するプラットフォーム	サンプルコード
Windows（MFC）	TRTCMainViewController.cpp
Windows（Duilib）	TRTCMainViewController.cpp
Windows(C#)	TRTCMainForm.cs

オンラインライブストリーミング

1. SDKの初期化

TRTC SDKを使用するときの最初のステップでは、まず `TRTCCloud` のシングルトンオブジェクトを取得し、SDKイベントを監視するためのコールバックを登録します。

- `ITRTCCloudCallback` イベントのコールバックインターフェースクラスを継承して、ローカルユーザーの入室/退室イベント、リモートユーザーの入室/退室イベント、エラーイベント、警告イベントなど、重要なイベントのコールバックインターフェースを書き換えます。
- `addCallback` インターフェースをコールして、SDKイベントを登録、監視します。

注意：

`addCallback` でN回登録すると、同一イベントに対して、SDKがN回コールバックします。 `addCallback` を1回のみ呼び出すことをお勧めします。

- C++版
- C#版

```
// TRTCMainViewController.h

// ITRTCCloudCallbackのイベントコールバックインターフェースクラスを継承します
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()
{
// TRTCCloud インスタンスの作成
m_pTRTCSDK = getTRTCShareInstance();

// SDK イベントコールバックの登録
m_pTRTCSDK->addCallback(this);
}

TRTCMainViewController::~TRTCMainViewController()
{
// SDK イベントのモニタリング のキャンセル
if(m_pTRTCSDK) {
m_pTRTCSDK->removeCallback(this);
```

```
}

// TRTCCloud インスタンスのリリース
if(m_pTRTCSDK != NULL) {
destroyTRTCShareInstance();
m_pTRTCSDK = null;
}
}

// エラー通知はモニタリングする必要があります。エラー通知は、SDKの実行を継続できないことを意味します
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();
}
}
```

2. TRTCParamsの組み立て

TRTCParamsは、SDKの最も重要なパラメータであり、sdkAppId、userId、userSig、roomIdという4つの入力必須フィールドが含まれます。

- **SDKAppID**
  Tencent Cloud TRTC コンソールに入ります。アプリケーションがない場合は、作成してください。作成するとSDKAppIDが確認できます。

- **userId**
  自由に指定することができ、文字列タイプのため、お客様の既存のアカウント体系と同じのものにすることが可能です。但し、**同じ音声/ビデオルームには2つの同名の userIdが存在できません**ので、ご注意ください。

- **userSig**
  SDKAppIDとuserIdを基に、userSigを計算できます。計算方法については、UserSigの計算方法をご参照ください。

- **roomId**
  ルーム番号は数字タイプであり、自由に指定できますが、**同一のアプリケーション内の2つのオーディオビデオルームに同じroomIdを割り当てることはできません**ので、ご注意ください。文字列形式のルーム番号を使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。

3.キャスターのプレビューカメラ画面

TRTC SDK は、デフォルトではローカルのWebカメラの撮影が有効になっていません。 `startLocalPreview` でローカルのWebカメラをオンにしてプレビュー画面を表示でき、 `stopLocalPreview` でこれをオフにします。

ローカルプレビューを起動する前に、 `setLocalViewFillMode` を呼び出して、ビデオ表示モードを `Fill` または `Fit` モードに指定することができます。この2種類のモードでは、ビデオサイズは同じ比率のまま拡大・縮小され、次のような違いがあります。

- `Fill` モードでは、ウィンドウを確実に塗りつぶすことが優先されます。拡大・縮小されたビデオサイズと表示ウィンドウのサイズが一致しない場合、余分なビデオ部分は削除されます。
- `Fit` モードでは、すべてのビデオコンテンツを確実に表示することが優先されます。拡大・縮小されたビデオサイズと表示ウィンドウのサイズが一致しない場合、塗りつぶされていないウィンドウ領域は黒で塗りつぶされます。

- C++版
- C#版

```
void TRTCMainViewController::onEnterRoom(uint64_t elapsed)
{
// レンダリングウィンドウのハンドルを取得します。
CWnd *pLocalVideoView = GetDlgItem(IDC_LOCAL_VIDEO_VIEW);
HWND hwnd = pLocalVideoView-&gt;GetSafeHwnd();

if(m_pTRTCSDK)
{
// SDKインターフェースを呼び出し、レンダリングモードおよびレンダリングウィンドウを設定します。
m_pTRTCSDK-&gt;setLocalViewFillMode(TRTCVideoFillMode_Fit);
m_pTRTCSDK-&gt;startLocalPreview(hwnd);
}

}
```

4.キャスターによるマイク集音の起動

TRTC SDKは、デフォルトではローカルのマイク集音がオンになりません。キャスターは `startLocalAudio` を呼び出してローカルの集音を起動し、オーディオビデオデータを配信することができます。 `stopLocalAudio` はその機能をオフにします。 `startLocalPreview` の後、続いて `startLocalAudio` を呼び出すことができます。

> 説明：
> `startLocalAudio` はマイクの使用権限をチェックします。マイクの権限がない場合、SDKはユーザーに起動の申請をします。

5.キャスターによるルームの新規作成、配信の開始

キャスターは `enterRoom` を使用してオーディオビデオルームを作成することができます。パラメータ `TRTCParams` の `roomId` は、ルーム番号を指定するために使用します。同時に、 `role` フィールドを `TRTCRoleAnchor` （キャスター）に指定する必要もあります。

`appScene` パラメータは、SDKのユースケースを指定します。ここでは、 `TRTCAppSceneLIVE` （オンラインライブストリーミング）を使用します。

- 作成に成功すると、SDKは `onEnterRoom` インターフェースをコールバックします。パラメータ `elapsed` は入室の所要時間を表し、単位はmsです。
- 作成に失敗すると、SDKは `onError` インターフェースをコールバックします。パラメータは、 `errCode` （エラーコード `ERR_ROOM_ENTER_FAIL` 、エラーコードは `TXLiteAVCode.h` を参照できます）、 `errMsg` （エラー原因）、 `extraInfo` （保留用パラメータ）です。

- C++版
- C#版

```
// TRTCMainViewController.cpp

void TRTCMainViewController::startBroadCasting()
{
// TRTCParamsの定義はヘッダーファイル TRTCCloudDef.hを参照
TRTCParams params;
params.sdkAppId = sdkappid;
params.userId = userid;
params.userSig = usersig;
params.roomId = 908; // 入室したいルームを入力します
params.role = TRTCRoleAnchor; //キャスター
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;enterRoom(params, TRTCAppSceneLIVE);
}
}

void TRTCMainViewController::onError(TXLiteAVError errCode, const char* errMsg, void* arg)
{
if(errCode == ERR_ROOM_ENTER_FAIL)
{
LOGE(L"onError errorCode[%d], errorInfo[%s]", errCode, UTF82Wide(errMsg).c_str());
// userSigが合法か、ネットワークが正常かなどをチェックします
}
}
```

```
...

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

// ローカルのビデオプレビューを起動します。以下のドキュメントを参照して、ビデオコーデックパラメータを設定し、ローカルのカメラ画面の内容をプレビューしてください
}
```

6.キャスターによるプライバシーモードのオン・オフ

キャスターはライブストリーミング中、プライバシーを保護するために、ローカルのオーディオビデオデータをブロックしたい場合が出てきます。このような場合、 `muteLocalVideo` を呼び出せばローカルのビデオ収集をブロックでき、 `muteLocalAudio` を呼び出せばローカルのオーディオ収集をブロックできます。

7.視聴者が入室して視聴

視聴者は `enterRoom` を呼び出すことでオーディオビデオルームに入室することができます。パラメータTRTCParamsの `roomId` は、ルームナンバーを指定するために使用されます。
`appScene` は、同様に `TRTCAppSceneLIVE` （オンラインライブストリーミング）にも入力しますが、 `role` フィールドは `TRTCRoleAudience` （視聴者）と指定する必要があります。

- C++版
- C#版

```
void TRTCMainViewController::startPlaying()
{
// TRTCParamsの定義はヘッダーファイル TRTCCloudDef.hを参照
TRTCParams params;
params.sdkAppId = sdkappid;
params.userId = userid;
params.userSig = usersig;
params.roomId = 908; // 入室したいルームを入力します
params.role = TRTCRoleAudience; //視聴者
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;enterRoom(params, TRTCAppSceneLIVE);
}
}
```

キャスターがルーム内にいる場合、視聴者はTRTCCloudDelegateの `onUserVideoAvailable` コールバックを介してキャスターのuseridを取得します。次に、視聴者が `startRemoteView` メソッドを呼び出すと、キャスターのビ

デオ画面が表示されます。

`setRemoteViewFillMode` によって、ビデオ表示モードを `Fill` または `Fit` モードに指定することができます。この2種類のモードはビデオサイズはいずれも同じ比率で拡大縮小します。違いは以下のとおりです。

- `Fill` モード：ビューウィンドウが全てコンテンツで埋まることを優先的に保証します。拡大縮小後のビデオサイズがビューウィンドウのサイズと一致しない場合、はみ出たビデオの部分はカットされます。
- `Fit` モード：ビデオのコンテンツが全て表示されることを優先的に保証します。拡大縮小後のビデオサイズがビューウィンドウのサイズと一致しない場合、欠けているウィンドウエリアは黒色で補填されます。

- C++版
- C#版

```
void TRTCMainViewController::onUserVideoAvailable(const char* userId, bool available){
if (available) {
// レンダリングウィンドウのハンドルを取得します。
CWnd *pRemoteVideoView = GetDlgItem(IDC_REMOTE_VIDEO_VIEW);
HWND hwnd = pRemoteVideoView-&gt;GetSafeHwnd();

// リモートユーザーのビデオのレンダリングモードを設定します。
m_pTRTCSDK-&gt;setRemoteViewFillMode(TRTCVideoFillMode_Fill);
// SDK インターフェースを呼び出し、リモートユーザーのストリーミングを再生します。
m_pTRTCSDK-&gt;startRemoteView(userId、 hwnd);
}else{
m_pTRTCSDK-&gt;stopRemoteView(userId);
}
}
```

注意：

- TRTCAppSceneLIVEモードでは、同じルームにいる視聴者(TRTCRoleAudience)人数に制限はありません。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

8.視聴者とキャスターとのマイク接続

キャスターと視聴者のいずれも、TRTCCloudが提供する `switchRole` を介して、ロールを相互に切り替えることができます。最もよくあるのは、視聴者とキャスターとのマイク接続シナリオです。視聴者はこのインターフェースを通じて「アシスタントキャスター」に切り替わった上で、ルーム内のもとの「メインキャスター」と双方向のマイク接続を行うことができます。

9. ルームからの退出

`exitRoom` メソッドを呼び出してルームを退出します。通話中か否かにかかわらず、このメソッドを呼び出すと、ビデオ通話に関するすべてのリソースがリリースされます。 `exitRoom` をコールした後、SDKは複雑な退室ハンドシェイクフローに進みます。SDKが `onExitRoom` メソッドをコールバックすると、リソースのリリースが完全に終了します。

- C++版
- C#版

```
// TRTCMainViewController.cpp

void TRTCMainViewController::exitRoom()
{
if(m_pTRTCSDK)
{
m_pTRTCSDK-&gt;exitRoom();
}
}
....
void TRTCMainViewController::onExitRoom(int reason)
{
// 退出に成功、reason パラメータは保留され、現在使用されていません。

}
```

ライブストリーミングクイックスタート (Electron)

最終更新日：：2022-02-14 14:49:57

ユースケース

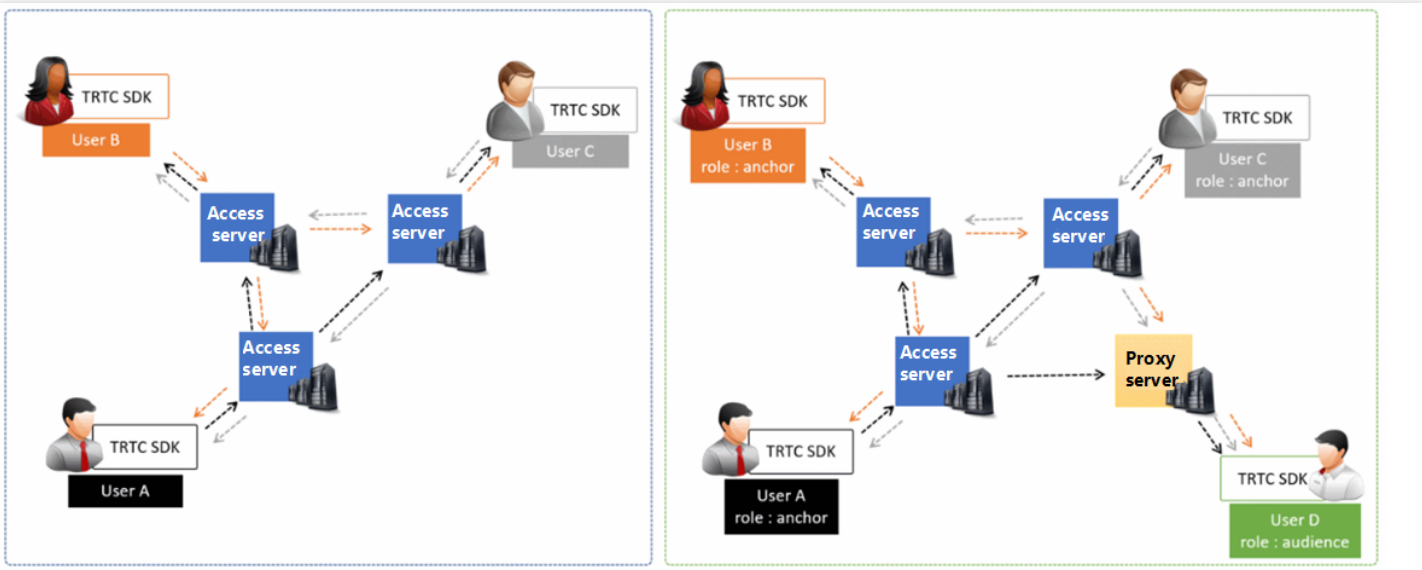
TRTCは、4種類の異なる入室モードをサポートしています。このうち、ビデオ通話（VideoCall）および音声通話（VoiceCall）を総称して通話モードといい、ビデオ・インタラクティブストリーミング（Live）およびボイス・インタラクティブストリーミング（VoiceChatRoom）を総称してライブストリーミングモードといいます。
ライブストリーミングモードでのTRTCは、1つのルームで最大10万人の同時接続をサポートし、300ms未満のマイク接続遅延、1000ms未満の視聴遅延およびマイクのオン・オフのスムーズな切り替え技術を備えています。
低レイテンシーインタラクティブストリーム、10万人のインタラクティブ教室、ビデオ婚活、eラーニング、リモート研修、超大規模ミーティングなどのユースケースに適しています。

原理解析

TRTCクラウドサービスは、「インターフェースモジュール」および「プロキシモジュール」という2種類の異なるタイプのサーバーノードから構成されています。

- **インターフェースモジュール**
  この種のノードは、最も良質の回線および高性能の機器を採用しており、エンドツーエンドの低遅延マイク接続通話の処理に優れています。
- **プロキシモジュール**
  この種のノードは、通常の回線および性能も一般的な機器を採用しており、同時進行性の高いプルストリーミング再生ニーズの処理にすぐれています。

通話モードでは、TRTCルームのすべてのユーザーはインターフェースモジュールに割り当てられます。各ユーザーは「キャスター」に相当し、各ユーザーは随時発言でき（同時アップストリームの最大制限は50）、このためオンラインミーティングなどのユースケースに適していますが、1つのルームの人数制限は300人になります。



サンプルコード

Githubにログインして、このドキュメントに関連したサンプルコードを取得することができます。

操作手順

手順1：公式ウェブサイトのSimpleDemoクイックスタートを試行する

初めにドキュメントSimpleDemoクイックスタート(Electron)を読み、ドキュメントのガイドに従って、提供されている公式SimpleDemoクイックスタートを実行してください。
SimpleDemoが順調に動作する場合は、プロジェクトにおいてElectronのインストール方法をお客様が把握していることを意味します。

- 反対に、 SimpleDemo の動作に問題がある場合は、Electronのダウンロード、インストールに問題があったことが考えられます。この場合はElectron の公式サイトの インストールガイド をご参照ください。

手順2：お客様のプロジェクトにtrtc-electron-sdkを統合する

[手順1]](#step1) が正常に動作し、予想どおりの効果があった場合は、 Electron環境のインストール方法を把握していることを意味します。

- 弊社の公式Demoをベースとして二次開発を行うことができ、プロジェクトの初級段階がスムーズに進みます。
- 次のコマンドを実行して、既存のプロジェクトに `trtc-electron-sdk` をインストールすることもできます。

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

手順3：SDKインスタンスを初期化し、イベントコールバックを監視する

`trtc-electron-sdk` インスタンスの新規作成：

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

`onError` イベントの監視：

```
// エラー通知は監視すべきもので、捕捉してユーザーに通知する必要があります
let onError = function(err) {
console.error(err);
}
trtcCloud.on('onError',onError);
```

手順4：入室パラメータTRTCParamsを組み立てる

enterRoom()インターフェースを呼び出すときはキーパラメータTRTCParamsを入力する必要があります。このパラメータに含まれる入力必須フィールドは下表に示すとおりです。

パラメータ	タイプ	説明	サンプル
sdkAppId	数字	アプリケーションID。 コンソール >【アプリケーション管理】>【アプリケーション情報】にあります。	1400000123
userId	文字列	アルファベットの大文字、小文字（a-z、A-Z）、数字（0-9）、下線およびハイフンのみを許可。ビジネスの実際のアカウントシステムを組み合わせて設定することをお勧めします。	test_user_001
userSig	文字列	userIdを基にuserSigを計算できます。計算方法はUserSigの計算方法をご参照ください。	eJyrVareCeYrSy1SsII...
roomId	数字	数字タイプのルームナンバー。文字列形式のルームナンバーを使用したい場合は、TRTCParamsのstrRoomIdをご使用ください。	29834

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

```
let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = 29834;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
param.role = TRTCRoleType.TRTCRoleAnchor; // ロールを「キャスター」に設定します
```

注意：

- TRTCは、2つの同じuserIdによる同時入室をサポートしていません。同時に入室した場合、相互に干渉します。
- 各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

手順5：キャスター端末でのカメラのプレビューとマイク集音を起動する

1. キャスターはstartLocalPreview()を呼び出してローカルのカメラプレビューを起動することができます。SDKはカメラのアクセス許可をシステムにリクエストします。
2. キャスター側で `setLocalViewFillMode()` を呼び出して、ローカルビデオ画面の表示モードを設定することができます。
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fill` ：塗りつぶしを意味し、画面は同じ比率で拡大およびトリミングできますが、黒い縁取りは付きません。
   - `TRTCVideoFillMode.TRTCVideoFillMode_Fit` ：適応を意味し、画面は同じ比率で縮小してスクリーンにフィットしてそのコンテンツを完全に表示しますが、黒い縁取りが付くことがあります。
3. キャスターはsetVideoEncoderParam()インターフェースを呼び出してローカルビデオのコーデックパラメータを設定することができます。このパラメータによりルームの他のユーザーが視聴する画面の画質が決定されます。
4. キャスターはstartLocalAudio()を呼び出してマイクをオンにします。SDKはマイクの使用をシステムにリクエストします。

```
//サンプルコード：ローカルのオーディオ・ビデオストリーミングの公開
trtcCloud.startLocalPreview(view);
trtcCloud.startLocalAudio();
trtcCloud.setLocalViewFillMode(TRTCVideoFillMode.TRTCVideoFillMode_Fill);

//ローカルビデオコーデックパラメータの設定
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);
```

手順6：キャスター端末により美顔エフェクトを設定する

1. キャスターはsetBeautyStyle(style, beauty, white, ruddiness)を呼び出して、美顔エフェクトをオンにすることができます
2. パラメータの説明：
   - style：美顔スタイルで、「スムース」または「ナチュラル」があります。スムースタイプは美肌補正がより強く、華やかなシーンに適しています。
     - `TRTCBeautyStyle.TRTCBeautyStyleSmooth`：スムース。美女が登場するシーンに適しており、明らかな効果が感じられます。
     - `TRTCBeautyStyle.TRTCBeautyStyleNature`：ナチュラル。美肌補正のアルゴリズムにより顔の細部の質感が保たれ、より自然な感じに仕上がります。
   - beauty：美顔レベル。数値の範囲は0～9です。0はオフを表し、1～9まで数値が大きくなるほど効果が高くなります
   - white：美白レベル。数値の範囲は0～9です。0はオフを表し、1～9まで数値が大きくなるほど効果が高くなります
   - ruddiness：肌色補正レベル。数値の範囲は0～9です。0はオフを表し、1～9まで数値が大きくなるほど効果が高くなります。このパラメータは、Windowsプラットフォームではまだ有効ではありません

```
// 美顔エフェクトをオン
trtcCloud.setBeautyStyle(TRTCBeautyStyle.TRTCBeautyStyleNature, 5, 5, 5);
```

手順7：キャスター端末からルームを新規作成し、プッシュを開始する

1. キャスターはTRTCParamsのフィールド `role` を **`TRTCRoleType.TRTCRoleAnchor`** に設定します。これは現在のユーザーのロールがキャスターであることを示します。
2. キャスター側でenterRoom（）を呼び出すと、TRTCParamsパラメータフィールド `roomId` の値がルーム番号となるオーディオ・ビデオルームを作成し、 `appScene` パラメータを指定することができます。
   - `TRTCAppScene.TRTCAppSceneLIVE` ：ビデオ・インタラクティブストリーミングは、スムーズなマイクのオン・オフをサポートしています。切り替え中に待機の必要がなく、キャスターの遅延は300ミリ秒未満です。10万人規模の視聴者の同時再生をサポートしつつ、再生遅延は1000ミリ秒に抑えます。ここでは、このモードを例として取り上げます。
   - `TRTCAppScene.TRTCAppSceneVoiceChatRoom` ：ボイス・インタラクティブストリーミングは、スムーズなマイクのオン・オフをサポートします。切り替え中に待機の必要がなく、キャスターの遅延は300ミリ秒未満です。10万人規模の視聴者の同時再生をサポートしつつ、再生遅延は1000ミリ秒に抑えます。

- `TRTCAppScene` の詳細については、TRTCAppScene をご参照ください。

3. ルームの作成が成功したら、キャスター側はオーディオ・ビデオデータのエンコードと伝送フローを開始します。それと同時に、SDKはonEnterRoom(result)イベントをコールバックします。パラメータ `result` が0より大きいときは入室成功を示し、具体的な数値は入室してからの消費時間になります。単位はミリ秒（ms）です。 `result` が0より小さいときは入室失敗を示し、具体的な数値は入室失敗のエラーコードになります。

```
let onEnterRoom = function (result) {
if (result &gt; 0) {
console.log(`onEnterRoom、入室成功、${result}秒を使用`);
}else{
console.warn(`onEnterRoom: 入室失敗 ${result}`);
}
};

trtcCloud.on('onEnterRoom', onEnterRoom);

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

手順8：視聴者が入室しライブストリーミングを視聴する

1. 視聴者側はTRTCParamsのフィールド `role` を `TRTCRoleType.TRTCRoleAudience` に設定します。これは現在のユーザーロールが視聴者であることを示します。
2. 視聴者側で `enterRoom()` を呼び出すと、 `TRTCParams` パラメータの `roomId` が示すオーディオ・ビデオルームに入室し、 `appScene` パラメータを指定することができます。
   - `TRTCAppScene.TRTCAppSceneLIVE` ：ビデオ・インタラクティブストリーミング。
   - `TRTCAppScene.TRTCAppSceneVoiceChatRoom` ：ボイス・インタラクティブストリーミング。
3. キャスター画面の視聴：
   - 視聴者側が事前にキャスターの `userId` を知っている場合、ルームに入室成功後に直ちにキャスターの `userId` を使用してstartRemoteView(userId, view)を呼び出し、キャスターの画面を表示することができます。
   - 視聴者がキャスターの `userId` を知らない場合、視聴者は入室成功後にonUserVideoAvailable()イベント通知を受け取り、コールバック中に取得するキャスターの `userId` を使用してstartRemoteView(userId, view)を呼び出すと、キャスターの画面を表示することができます。

```
<div id="video-container"></div>
<script>
const videoContainer = document.querySelector('#video-container');
const roomId = 29834;
// 入室コールバック。このコールバックは、入室に成功したときにトリガーされます。
let onEnterRoom = function(result) {
if (result > 0) {
console.log(`onEnterRoom、入室成功、${result}秒を使用`);
}else{
console.warn(`onEnterRoom: 入室失敗 ${result}`);
}
};
// このコールバックは、キャスターがカメラプッシュのオン/オフを切り替えたときにトリガーされます
let onUserVideoAvailable = function(userId, available) {
if (available === 1) {
let id = `${userId}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (!view) {
view = document.createElement('div');
view.id = id;
videoContainer.appendChild(view);
}
trtcCloud.startRemoteView(userId, view);
trtcCloud.setRemoteViewFillMode(userId, TRTCVideoFillMode.TRTCVideoFillMode_Fill);
}else{
let id = `${userId}-${roomId}-${TRTCVideoStreamType.TRTCVideoStreamTypeBig}`;
let view = document.getElementById(id);
if (view) {
videoContainer.removeChild(view);
}
}
};

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

let param = new TRTCParams();
param.sdkAppId = 1400000123;
param.roomId = roomId;
param.userId = 'test_user_001';
param.userSig = 'eJyrVareCeYrSy1SslI...';
param.role = TRTCRoleType.TRTCRoleAudience; // ロールを「視聴者」に設定します
trtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
</script>
```

手順9：視聴者とキャスターとのマイク接続

1. 視聴者はswitchRole(TRTCRoleType.TRTCRoleAnchor)を呼び出して、ロールをキャスター（ `TRTCRoleType.TRTCRoleAnchor` ）に切り替えます。
2. 視聴者はstartLocalPreview()を呼び出してローカル画面を起動することができます。
3. 視聴者はstartLocalAudio()を呼び出してマイクの集音をオンにします。

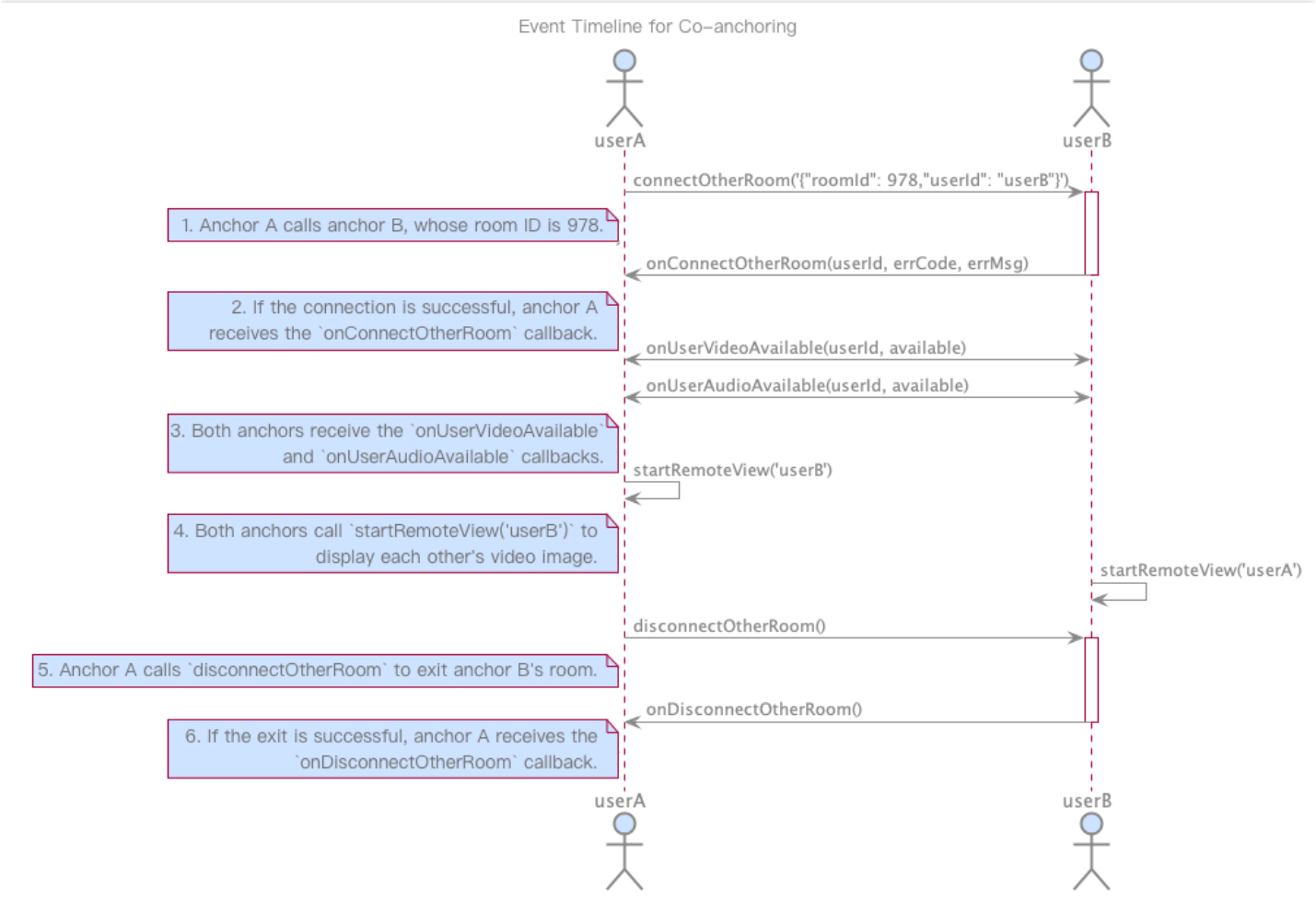
```
//サンプルコード：視聴者マイク・オン
trtcCloud.switchRole(TRTCRoleType.TRTCRoleAnchor);
trtcCloud.startLocalAudio();
trtcCloud.startLocalPreview(frontCamera, view);

//サンプルコード：視聴者マイク・オフ
trtcCloud.switchRole(TRTCRoleType.TRTCRoleAudience);
trtcCloud.stopLocalAudio();
trtcCloud.stopLocalPreview();
```

手順10：キャスター間でルーム間マイク接続PKを行う

TRTCでは、異なるオーディオ・ビデオルームにいる2人のキャスターが当初のライブストリーミングルームを退出しない場合にも、「ルーム間通話」機能によってマイク接続通話機能をプルし、「ルーム間マイク接続PK」を行うことができます。

1. キャスターAはconnectOtherRoom()インターフェースを呼び出します。インターフェースのパラメータは現在JSONフォーマットを採用し、キャスターBの `roomId` と `userId` の形式に組み立てた `{"roomId": 978,"userId": "userB"}` のパラメータをインターフェース関数に渡す必要があります。
2. ルームを跨ぐことに成功した後、キャスターAはonConnectOtherRoom(userId, errCode, errMsg)イベントコールバックを受け取ります。同時に、2つのライブストリーミングルームのすべてのユーザーはいずれもonUserVideoAvailable()およびonUserAudioAvailable()イベント通知を受け取ります。
   例えば、ルーム「001」のキャスターAがルーム「002」のキャスターBと `connectOtherRoom()` を介してルーム間通話をする場合、ルーム「001」のユーザーはキャスターBの `onUserVideoAvailable(B, true)` コールバックと `onUserAudioAvailable(B, true)` コールバックを受信します。ルーム「002」のユーザーはキャスターAの `onUserVideoAvailable(A, true)` コールバックと `onUserAudioAvailable(A, true)` コールバックを受信します。
3. 両方のルームのユーザーはstartRemoteView(userId, view)を呼び出すと、もう一方のルームのキャスター画面を表示することができ、音声は自動再生されます。




```
//サンプルコード：ルーム間マイク接続PK
let onConnectOtherRoom = function(userId, errCode, errMsg) {
if(errCode === 0) {
console.log(`キャスター${userId}のルームとの接続に成功しました`);
}else{
console.warn(`他のキャスターのルームとの接続に失敗しました：${errMsg}`);
}
};

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

手順11：現在のルームから退出する

exitRoom()メソッドを呼び出してルームを退出します。SDKは退出時にカメラやマイクなどのハードデバイスを停止またはリリースする必要があるため、退出動作は一瞬では完了しません。退出操作を完了するにはonExitRoom()コールバックを受信する必要があります。

```
// 退室を呼び出した後は、onExitRoomイベントのコールバックをお待ちください
let onExitRoom = function (reason) {
console.log(`onExitRoom, reason: ${reason}`);
};
trtcCloud.exitRoom();
trtcCloud.on('onExitRoom', onExitRoom);
```

注意：
Electronプログラムで複数の音声ビデオSDKを同時に統合した場合は、`onExitRoom`コールバックを受信してから、その他の音声ビデオSDKを起動してください。そうしない場合、ハード上の占有問題が生じることがあります。

ライブストリーミングクイックスタート (Web)

最終更新日：：2022-03-10 09:48:54

ここでは主に、視聴者としてライブストリーミングルームに入室し、マイク接続して双方向通信を行うシナリオをご紹介します。キャスターとして入室してライブストリーミングを行うシナリオは、TRTCの通話シナリオと同様です。TRTC通話をご参照ください。

事例

Demoをクリックしてオーディオビデオ機能の体験に移動し、またはGitHub にログインして、このドキュメントに関連するサンプルコードを取得できます。

手順1： Clientオブジェクトの新規作成

TRTC.createClient()メソッドによって、Clientオブジェクトを作成します。パラメータの設定は次のとおりです。

- `mode` ：インタラクティブライブストリーミングモードの場合、 `live` に設定します
- `sdkAppId` ：Tencent Cloudから申請したsdkAppId
- `userId` ：ユーザーID
- `userSig` ：ユーザー署名

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

手順2：視聴者としてライブストリーミングルームに入室

Client.join()を呼び出して、オーディオビデオ通話ルームに参加します。パラメータの設定は次のとおりです。

- `roomId` ：ルームID

- `role` ：ユーザーロール
  - `anchor` ：キャスター。キャスターのロールには、ローカルストリームを公開し、リモートストリームを受信する権限があります。デフォルトはキャスターロールです。
  - `audience` ：視聴者。視聴者のロールには、リモートストリームを受信する権限のみがあり、ローカルストリームを公開する権限はありません。視聴者がキャスターとマイク接続して双方向通信を行いたい場合、Client.switchRole()を介してロールを `anchor` キャスターロールに切り替えた上で、ローカルストリームを公開する必要があります。

```
// 視聴者ロールで入室して視聴します
client
.join({ roomId, role: 'audience' })
.then(() => {
console.log('入室成功');
})
.catch(error => {
console.error('入室失敗 '+ error);
});
```

手順3：ライブストリーミングの視聴

1. リモートストリーミングは、 `client.on('stream-added')` イベントのモニタによって取得され、そのイベントを受信した後は、 Client.subscribe()によってリモートオーディオビデオストリーミングを閲覧します。

   説明：
   Client.join()で入室する前に `client.on('stream-added')` イベントを登録し、リモートユーザーの入室通知を見落とすことがないようにしてください。

```
client.on('stream-added', event => {
const remoteStream = event.stream;
console.log('リモートストリーミングの増加: ' + remoteStream.getId());
//リモートストリーミングの閲覧
client.subscribe(remoteStream);
});
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('リモートストリーミングの閲覧成功: ' + remoteStream.getId());
// リモートストリーミングの再生
```

```
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

2. リモートストリーミングの閲覧成功イベントのコールバックでは、[Stream.play()](https://web.sdk.qcloud.com/trtc/webrtc/doc/zh-cn/Stream.html#play)メソッドを呼び出すことで、Webページ上で音声ビデオを再生します。`play`メソッドがdivエレメントのIDをパラメータとして受け入れると、SDKの内部で、そのdivエレメント下に対応する音声ビデオタグを自動作成し、その上で音声ビデオを再生します。

```
client.on('stream-subscribed', event => {
const remoteStream = event.stream;
console.log('リモートストリーミングの閲覧成功: ' + remoteStream.getId());
// リモートストリーミングの再生
remoteStream.play('remote_stream-' + remoteStream.getId());
});
```

手順4：キャスターとのマイク接続・双方向通信

手順4.1：ロールの切り替え

Client.switchRole()を使用して、ロールを `anchor` キャスターに切り替えます。

```
client
.switchRole('anchor')
.then(() => {
// ロールの切り替えに成功しました。現在のロールはキャスターです
})
.catch(error => {
console.error('ロールの切り替えに失敗しました' + error);
});
```

手順4.2：マイク接続・双方向通信

1. TRTC.createStream()メソッドを使用して、ローカルのオーディオビデオストリーミングを作成します。以下のインスタンスではカメラとマイクからオーディオビデオストリーミングをキャプチャします。パラメータの設定は次のとおりです。

- `userId` ：ローカルストリーミング所属のユーザーID
- `audio` ：オーディオの起動の有無
- `video` ：ビデオの起動の有無

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

2. LocalStream.initialize()を呼び出して、ローカルのオーディオビデオストリーミングを初期化します。

```
localStream
.initialize()
.then(() => {
console.log('ローカルストリーミング初期化の成功');
})
.catch(error => {
console.error('ローカルストリーミング初期化の失敗 ' + error);
});
```

3. ローカルストリーミングの初期化に成功すると、ローカルストリーミングが再生されます。

```
localStream
.initialize()
.then(() => {
console.log('ローカルストリーミング初期化の成功');
localStream.play('local_stream');
})
.catch(error => {
console.error('ローカルストリーミング初期化の失敗 ' + error);
});
```

4. ローカルストリーミングの初期化に成功したら、Client.publish()メソッドを呼び出して、ローカルストリーミングを公開し、視聴者とのマイク接続・双方向通信をオンにします。

```
client
.publish(localStream)
.then(() => {
console.log('ローカルストリーミング公開の成功');
})
.catch(error => {
console.error('ローカルストリーミング公開の失敗 ' + error);
});
```

手順5：ライブストリーミングルームから退出する

ライブストリーミング終了時は、Client.leave()メソッドを呼び出してライブストリーミングルームを退出し、すべてのライブストリーミングセッションを終了します。

```
client
.leave()
.then(() => {
// 退出に成功
})
.catch(error => {
console.error('退室失敗 ' + error);
});
```

注意：
各端末のユースケースappSceneについては、統一する必要があります。統一していない場合、想定外のトラブルが生じる恐れがあります。

TRTCクラウドレコーディングの説明

最終更新日：：2022-04-06 16:26:53

eラーニング、ステージライブストリーミング、ビデオミーティング、オンライン医療相談、リモートバンキングなどのユースケースでは、コンテンツ審査、録画アーカイブ、ビデオ再生などのニーズを考慮して、ビデオ通話やインタラクティブライブストリーミングのプロセス全体をレコーディングし保存する必要が度々出てきます。これらはクラウドレコーディング機能によって実現することができます。

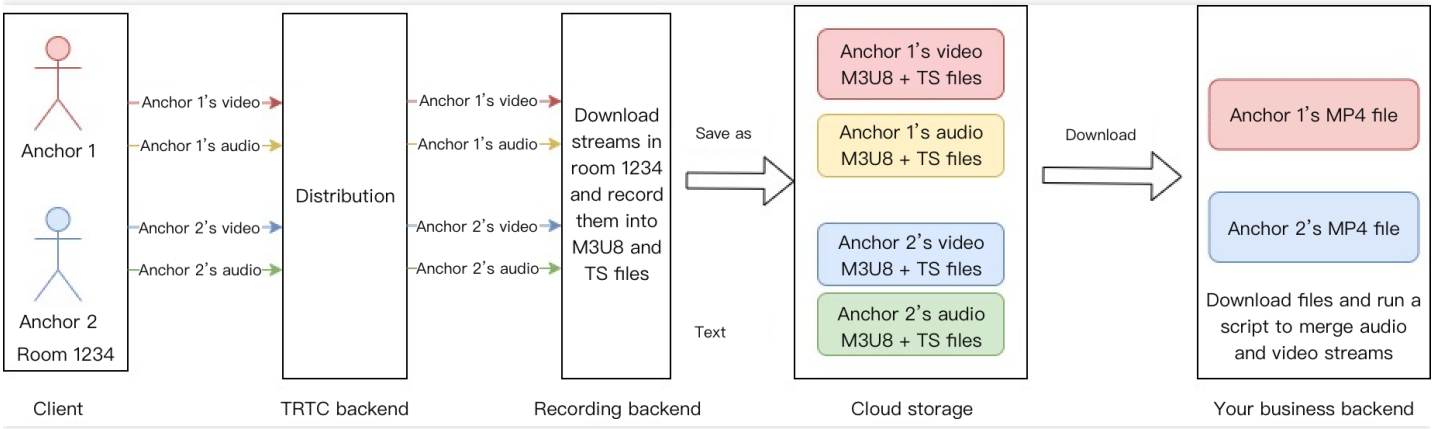
機能の説明

TRTCのクラウドレコーディング機能では、REST APIインターフェースを呼び出してクラウドレコーディングタスクを開始し、レコーディングしたいオーディオビデオストリーミングをサブスクライブし、リアルタイムかつフレキシブルに管理することができます。また、開発者が自らサーバーやレコーディング関連モジュールをデプロイする必要がなく、より軽く使いやすくなっています。

- レコーディングモード：シングルストリームレコーディングでは、ルームの各ユーザーのオーディオビデオストリームをそれぞれ独立したファイルとしてレコーディングできます。ミクスストリームレコーディングでは、同一のルームのオーディオビデオメディアストリームを1つのファイルとしてレコーディングすることができます

- ストリームサブスクリプション：サブスクリプションユーザーのブラックリスト/ホワイトリストを制定する方法で、サブスクライブしたいユーザーのメディアストリームを指定することができます
- トランスコードパラメータ：ミクスストリーミングのシーンで、コーデックのパラメータを設定することで、レコーディングするビデオファイルの品質を指定することができます
- ミクスストリームパラメータ：ミクスストリームのシーンで、複数のフレキシブルな自動マルチ画面レイアウトテンプレートやカスタムレイアウトテンプレートをサポートします
- ファイルストレージ：レコーディングしたファイルの保存先にクラウドストレージ/VODを指定することができます。現時点で、クラウドストレージベンダーはTencent CloudのCOSストレージをサポートし、VODベンダーはTencent Cloud VODをサポートしています
  （今後、その他のクラウドベンダーのストレージやVODサービスをサポートする場合があり、その際はお客様のクラウドサービスアカウントのご提供が必要です。クラウドストレージサービスにはストレージパラメータのご提供が必要です）
- コールバック通知：コールバック通知機能をサポートし、コールバックドメイン名を設定することで、クラウドレコーディングのイベントステータスがお客様のコールバックサーバーに通知されます

シングルストリームレコーディング

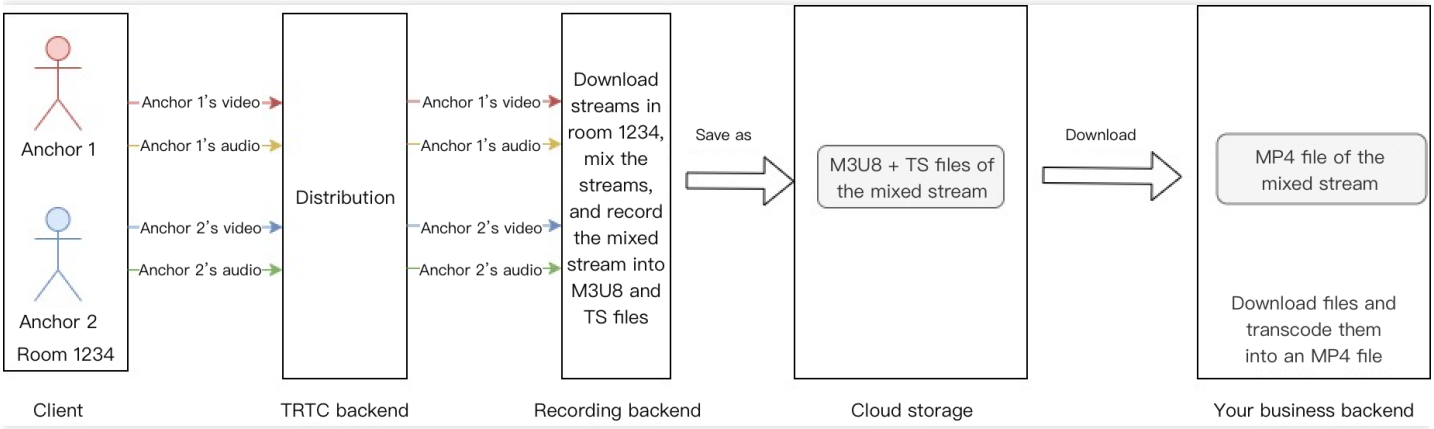


シングルストリームレコーディングのシーンを図にしたものです。ルーム1234でキャスター1とキャスター2がどちらもオーディオビデオストリームを行っています。キャスター1とキャスター2のオーディオビデオストリームをサブスクライブし、レコーディングモードをシングルストリームレコーディングに設定し、レコーディングバックエンドでキャスター1とキャスター2のオーディオビデオストリームをそれぞれプルし、それらを独立したメディアファイルとしてレコーディングすると仮定します。ファイルには次のものが含まれます。

1. キャスター1のビデオM3U8インデックスファイル1個。
2. キャスター1のいくつかのビデオTS分割ファイル。
3. キャスター1のオーディオM3U8インデックスファイル1個。
4. キャスター1のいくつかのオーディオTS分割ファイル。
5. キャスター2のビデオM3U8インデックスファイル1個。
6. キャスター2のいくつかのビデオTS分割ファイル。
7. キャスター2のオーディオM3U8インデックスファイル1個。
8. キャスター2のいくつかのオーディオTS分割ファイル。

レコーディングバックエンドはこれらのファイルを、お客様の指定するクラウドストレージサーバーにアップロードします。お客様の業務バックエンドはこれらのレコーディングファイルをプルし、業務ニーズに応じてファイルの結合やトランスコード操作を行う必要があります。オーディオビデオの結合とトランスコードのスクリプトをご提供できます。

ミクスストリームレコーディング



ミクスストリームレコーディングのシーンを図にしたものです。ルーム1234でキャスター1とキャスター2がどちらもオーディオビデオストリームを行っています。キャスター1とキャスター2のオーディオビデオストリームをサブスクライブし、レコーディングモードをミクスストリームレコーディングに設定し、レコーディングバックエンドでキャスター1とキャスター2のオーディオビデオストリームをそれぞれプルし、それらのビデオストリームを設定したマルチ画面テンプレートに従ってミクスストリームし、オーディオストリームをミキシングし、最後にメディアストリームを1つのメディアファイルに統合すると仮定します。ファイルには次のものが含まれます。

1. ミクスストリーム後のビデオM3U8インデックスファイル1個。
2. ミクスストリーム後のいくつかのビデオTS分割ファイル。

レコーディングバックエンドはこれらのファイルを、お客様の指定するクラウドストレージサーバーにアップロードします。お客様の業務バックエンドはこれらのレコーディングファイルをプルし、業務ニーズに応じてファイルの結合やトランスコード操作を行う必要があります。オーディオビデオの結合とトランスコードのスクリプトをご提供できます。

注意：

- レコーディングインターフェースの呼び出し頻度は20qpsに制限されています
- 1つのインターフェースのタイムアウト時間は6秒です
- デフォルトでサポートされている同時レコーディングチャネル数は100チャネルです。チャネル数を増やしたい場合は、チケットを提出してご連絡ください。
- 1つのレコーディングタスクにつき、同時サブスクリプションをサポートする1ルーム内のオーディオビデオストリームは最大で25チャネルです。

呼び出しフロー

1. レコーディング開始

お客様のバックエンドサービスからREST API （CreateCloudRecording）を呼び出してクラウドレコーディングを開始します。特に重視すべきパラメータは次のとおりです。

タスクID（TaskId）

このパラメータはそのレコーディングタスクの固有識別子であり、後にこのレコーディングタスクインターフェースの操作を行うための入力パラメータとして、このタスクIDを保存しておく必要があります。

レコーディングモード（RecordMode）

- シングルストリームレコーディングでは、ルーム内でお客様がサブスクライブしたキャスターのオーディオおよびビデオファイルをそれぞれレコーディングし、レコーディングファイル（M3U8/TS）をクラウドストレージにアップロードします。
- ミクスストリームレコーディングでは、ルーム内でお客様がサブスクライブしたすべてのキャスターのオーディオビデオストリームを1つのオーディオビデオファイルにレコーディングし、レコーディングファイル[M3U8/TS]をクラウドストレージにアップロードします。

レコーディングユーザーのブラックリスト/ホワイトリスト（SubscribeStreamUserIds）

デフォルトでは、クラウドレコーディングはルーム内のすべてのメディアストリーム（最大25チャネル）をサブスクライブできます。このパラメータによって、サブスクライブするキャスターユーザーのブラックリスト/ホワイトリスト情報を指定することも可能です。もちろん、レコーディング中の更新操作もサポートしています。

クラウドストレージパラメータ（StorageParams）

クラウドストレージパラメータを指定すると、レコーディング後のファイルはお客様がアクティブ化した指定のクラウドストレージ/VODサービスにアップロードされます。クラウドストレージ/VODスペースのパラメータの有効性と、未払いの料金がない状態を維持するようご注意ください。ここでは、レコーディングファイルの名前に決まったルールがあることに注意が必要です

レコーディングファイル名の命名ルール

- シングルストリームレコーディングのM3U8ファイル名ルール：<prefix>/<taskid>/<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>.m3u8

- シングルストリームレコーディングのTSファイル名ルール：<prefix>/<taskid>/<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>_<utc>.ts

- シングルストリームレコーディングのmp4ファイル名ルール：<prefix>/<taskid>/<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<index>.

mp4

- ミクスストリームレコーディングのM3U8ファイル名ルール：
<prefix>/<taskid>/<sdkappid>_<roomid>.m3u8

- ミクスストリームレコーディングのTSファイル名ルール：
<prefix>/<taskid>/<sdkappid>_<roomid>_<utc>.ts

- ミクスストリームレコーディングのmp4ファイル名ルール：
<prefix>/<taskid>/<sdkappid>_<roomid>_<index>.mp4

- 高可用性プル後のファイル名ルール
クラウドレコーディングサービス中にデータセンターに障害が発生した場合、高可用性ソリューションによってレコーディングタスクを復旧させることがあります。このような場合、元のレコーディングファイルを上書きしないよう、プル後にプレフィックスha<1/2/3>を付けることで、高可用性の発生回数を表します。1つのレコーディングタスクにつき、プルの回数が最大で3回まで許容されます。

- シングルストリームレコーディングのM3U8ファイル名ルール：
<prefix>/<taskid>/ha<1/2/3>_<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>.m3u8

- シングルストリームレコーディングのTSファイル名ルール：
<prefix>/<taskid>/ha<1/2/3>_<sdkappid>_<roomid>__UserId_s_<userid>__UserId_e_<mediaid>_<type>_<utc>.ts

- ミクスストリームレコーディングのM3U8ファイル名ルール：
<prefix>/<taskid>/ha<1/2/3>_<sdkappid>_<roomid>.m3u8

- ミクスストリームレコーディングのTSファイル名ルール：
<prefix>/<taskid>/ha<1/2/3>_<sdkappid>_<roomid>_<utc>.ts

フィールドの意味の説明：

<prefix>: レコーディングパラメータに設定するファイル名プレフィックスです。設定していなければ存在しません。
<taskid>: レコーディングのタスクIDです。世界唯一のもので、レコーディング開始後に返されるパラメータ内に付帯します。
<sdkappid>: レコーディングタスクのSdkAppIdです。
<roomid>: レコーディングルーム番号です。

<userid>: 特殊なbase64処理後のレコーディングストリームのユーザーIDです。下記の注意事項をご参照ください。
<mediaid>: メイン/サブストリームを識別します。mainまたはauxです。
<type>: レコーディングストリームのタイプです。audioまたはvideoです。
<utc>: このファイルのUTCレコーディングの開始時間です。タイムゾーンはUTC+0で、年、月、日、時間、分、秒およびミリ秒から成ります。
<index>: mp4分割ロジック（サイズが2GBを超えるか、時間が24時間を超える）がトリガーされていなければ、このフィールドはありません。トリガーされている場合は分割ファイルのインデックス番号であり、1から順に付与されます。
ha<1/2/3>：高可用性プルのプレフィックスです。例えば、最初のプルにより、
<prefix>/<taskid>/ha1_<sdkappid>_<roomid>.m3u8に変わります。

> 注意：
> ここで<roomid>が文字列のルームIDの場合、ルームIDに対してまずbase64操作を行ってから、base64後の文字列中の記号'/'を'-' (ダッシュ)に、記号'='を'.'にそれぞれ置き換えます。
> レコーディングストリームの<userid>についてはまずbase64操作を行ってから、base64後の文字列中の記号'/'を'-' (ダッシュ)に、記号'='を'.'にそれぞれ置き換えます。

レコーディング開始時間の取得

レコーディング開始時間の定義は、最初にキャスターのオーディオビデオデータを受信し、最初のファイルのレコーディングを開始したサーバーunix時間です。
次の3つの方法で、レコーディング開始のタイムスタンプを取得することができます。

- レコーディングファイルインターフェース（DescribeCloudRecording）のBeginTimeStampフィールドを照会する

例えば、次の照会インターフェースから返された情報からは、BeginTimeStampは1622186279144msであることがわかります。

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

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

- M3U8ファイル中の、対応するタグの項目（#EXT-X-TRTC-START-REC-TIME）を読みとる

例えば、次のM3U8ファイルからは、レコーディング開始時間のunixタイムスタンプが1622425551884msであることがわかります。

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:70
#EXT-X-TRTC-START-REC-TIME:1622425551884
#EXT-X-TRTC-VIDEO-METADATA:WIDTH:1920 HEIGHT:1080
#EXTINF:12.074
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094551825.ts
#EXTINF:11.901
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094603825.ts
#EXTINF:12.076
1400123456_12345__UserId_s_MTY4NjExOQ..__UserId_e_main_video_20330531094615764.ts
#EXT-X-ENDLIST
```

- レコーディングコールバックイベントを監視する

サブスクリプションのコールバックにより、イベントタイプ307のBeginTimeStampフィールドで、レコーディングファイルに対応するレコーディング開始時間のタイムスタンプを取得することができます。
例えば、次のコールバックイベントからは、このファイルのレコーディング開始時間のunixタイムスタンプが1622186279144msであることが読み取れます。

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

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

ミクスストリームレコーディングのウォーターマークパラメータ（MixWatermark）

ミクスストリームレコーディング中の、画像ウォーターマーク追加をサポートしています。最大サポート数は25個で、キャンバスの任意の位置にウォーターマークを追加することができます。

フィールド名	説明
Top	ウォーターマークの左上隅に対しての垂直移動
Left	ウォーターマークの左上隅に対しての水平移動
Width	ウォーターマークの表示幅
Height	ウォーターマーク表示の高さ
url	ウォーターマークファイルの保存先url

ミクスストリームレコーディングのレイアウトモードパラメータ（MixLayoutMode）

9グリッドレイアウト（デフォルト）、フロートレイアウト、画面共有レイアウト、カスタムレイアウトの、4種類のレイアウトをサポートしています

9グリッドレイアウト：

キャスターの数に応じて各画面のサイズは自動調整されます。各キャスターの画面サイズは同一で、最大25画面をサポートします。

- サブ画面が1枚の場合：
  各小画面の幅と高さはそれぞれキャンバス全体の幅と高さとなります。

- サブ画面が2枚の場合：
  各小画面の幅はキャンバス全体の幅の1/2となります。
  各小画面の高さはキャンバス全体の高さとなります。

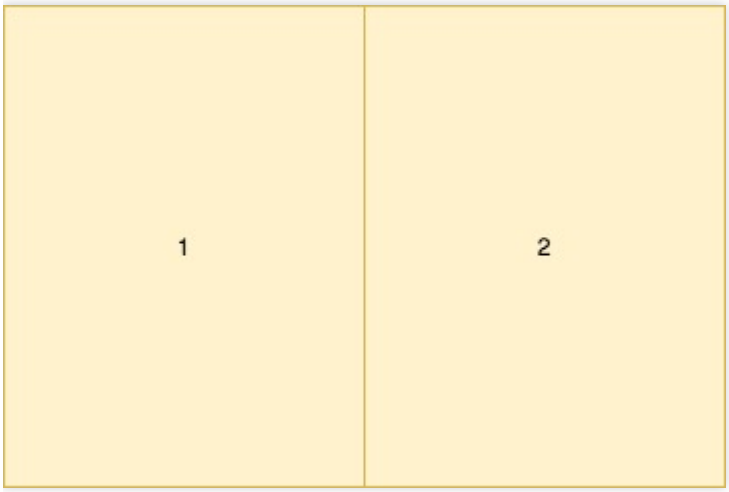
- サブ画面が4枚以下の場合：
  各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/2となります。

- サブ画面が9枚以下の場合：
  各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/3となります。

- サブ画面が16枚以下の場合：
  各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/4となります。

- サブ画面が16枚を超える場合：
  各小画面の幅と高さはそれぞれキャンバス全体の幅と高さの1/5となります。

  9グリッドレイアウトはサブスクリプションのサブ画面が増えるに従って、下図のように変化します。



1	2
3	4

1	2	3
4	5	6
7	8	9

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

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

フロートレイアウト：

デフォルトでは最初に入室したキャスター（キャスター1名を指定することもできます）のビデオ画面はスクリーン全体となります。その他のキャスターのビデオ画面は左下隅から順に、入室の順序に従って水平に並び、小画面として表示され、小画面が大画面の上にフロート表示されます。画面の数が17枚以下の場合は、1列に4画面（4 x 4列）となります。画面の数が17枚より多い場合は、小画面が1列に5画面（5 x 5列）となるよう再レイアウトされます。最大25画面をサポートし、ユーザーがオーディオのみを送信している場合でも、画面の位置を占有することができます。

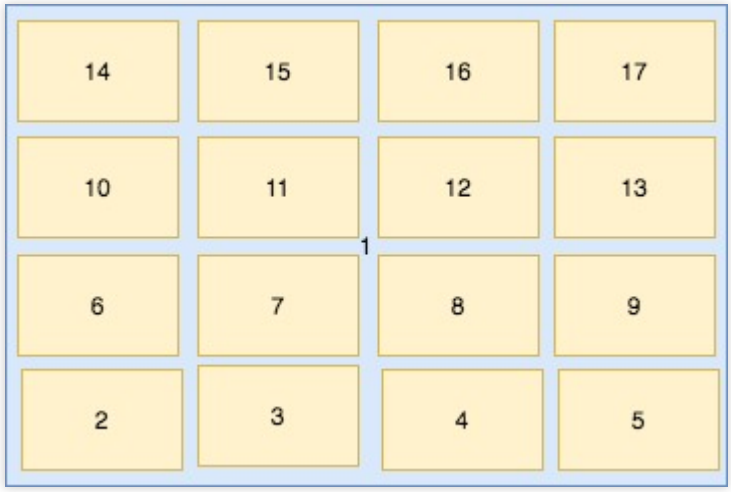
- サブ画面が17枚以下の場合：

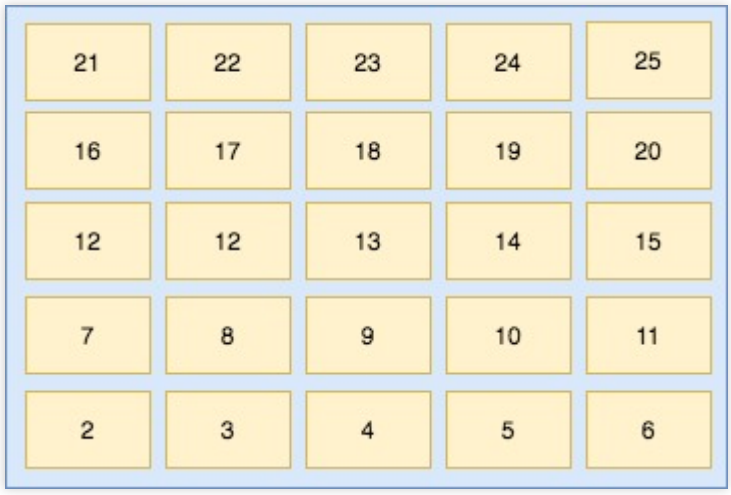
> 各小画面の幅と高さはそれぞれキャンバス全体の幅と高さ×0.235となります。
> 隣接する小画面の左右と上下の間隔はそれぞれキャンバス全体の幅と高さ×0.012となります。
> 小画面からキャンバスの縦枠および横枠までの余白も、それぞれキャンバス全体の幅と高さ×0.012となります。

- サブ画面が17枚を超える場合：

  各小画面の幅と高さはそれぞれキャンバス全体の幅と高さ×0.188となります。
  隣接する小画面の左右と上下の間隔はそれぞれキャンバス全体の幅と高さ×0.01となります。
  小画面からキャンバスの縦枠および横枠までの余白も、それぞれキャンバス全体の幅と高さ×0.01となります。

  フロートレイアウトはサブスクリプションのサブ画面が増えるに従って、下図のように変化します。





画面共有レイアウト：

キャスター1名を画面左側の大画面位置に指定し（指定しない場合、大画面位置は背景色になります）、その他のキャスターは右側に上から下へ順に垂直に配置します。画面の数が17枚より少ない場合は、右側の各列に最大8名、最大2列まで配置します。画面の数が17枚より多い場合は、17枚目以降のキャスターの画面を左下隅から順に水平に配置します。最大24画面をサポートし、キャスターがオーディオのみを送信している場合でも、画面の位置を占有することができます。

- サブ画面が5枚以下の場合:
  右側の小画面の幅はキャンバス全体の幅の1/5、右側の小画面の高さはキャンバス全体の高さの1/4となります。
  左側の大画面の幅はキャンバス全体の幅の4/5、左側の大画面の高さはキャンバス全体の高さとなります。

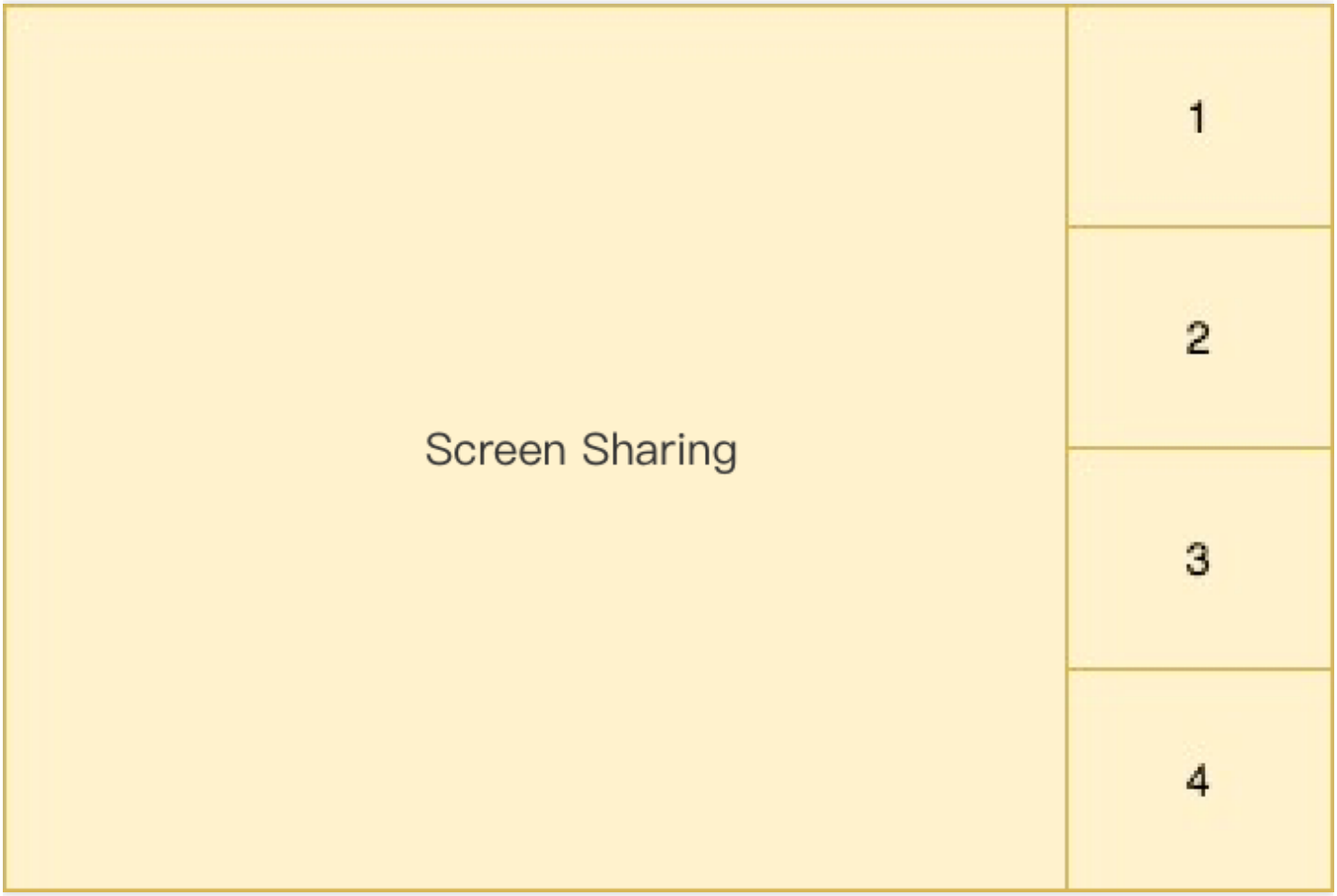
- サブ画面が5枚を超え、7枚以下の場合:
  右側の小画面の幅はキャンバス全体の幅の1/7、右側の小画面の高さはキャンバス全体の高さの1/6となります。
  左側の大画面の幅はキャンバス全体の幅の6/7、左側の大画面の高さはキャンバス全体の高さとなります。

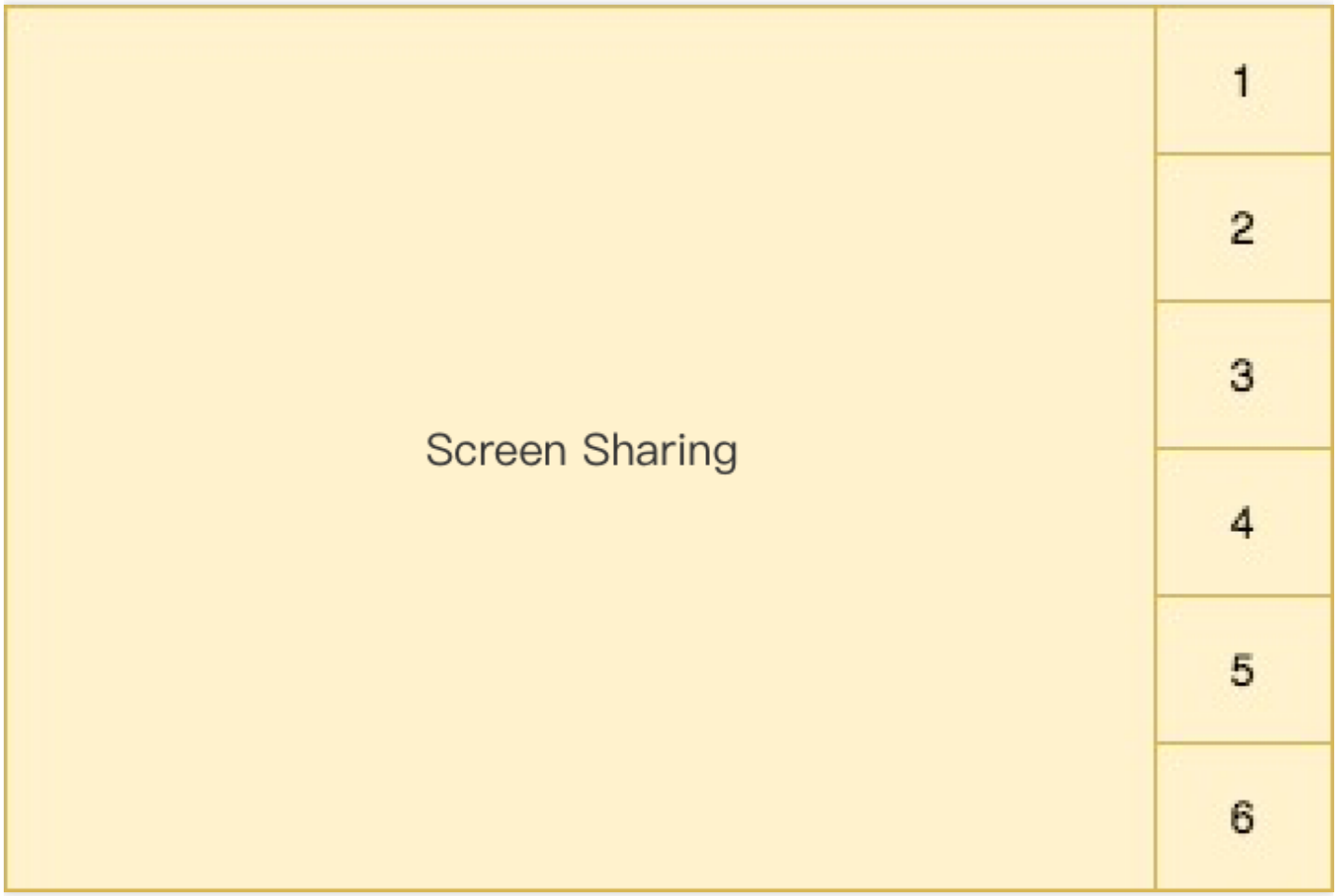
- サブ画面が7枚を超え、9枚以下の場合:
  右側の小画面の幅はキャンバス全体の幅の1/9、右側の小画面の高さはキャンバス全体の高さの1/8となります。
  左側の大画面の幅はキャンバス全体の幅の8/9、左側の大画面の高さはキャンバス全体の高さとなります。

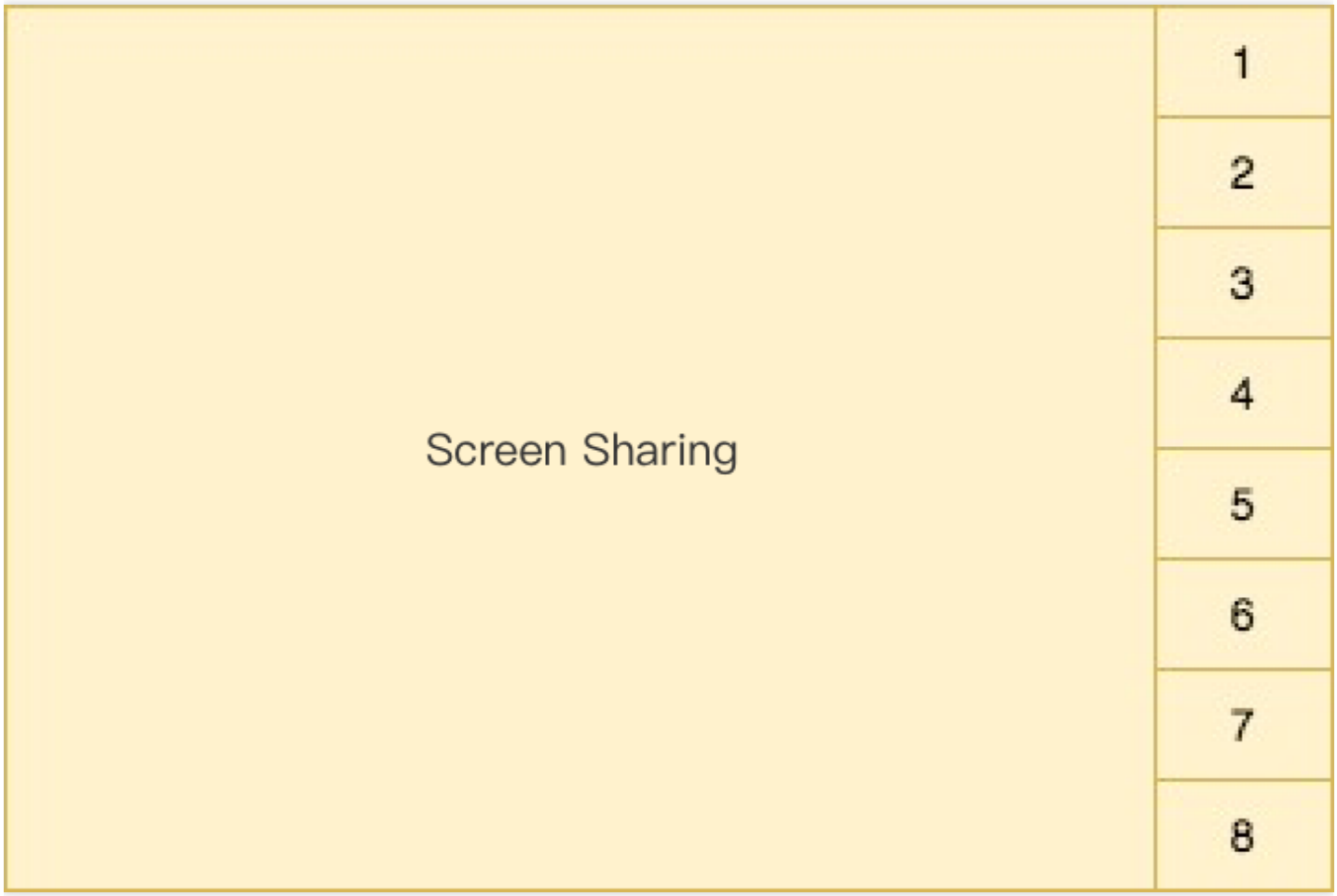
- サブ画面が9枚を超え、17枚以下の場合:
  右側の小画面の幅はキャンバス全体の幅の1/10、右側の小画面の高さはキャンバス全体の高さの1/8となります。
  左側の大画面の幅はキャンバス全体の幅の4/5、左側の大画面の高さはキャンバス全体の高さとなります。

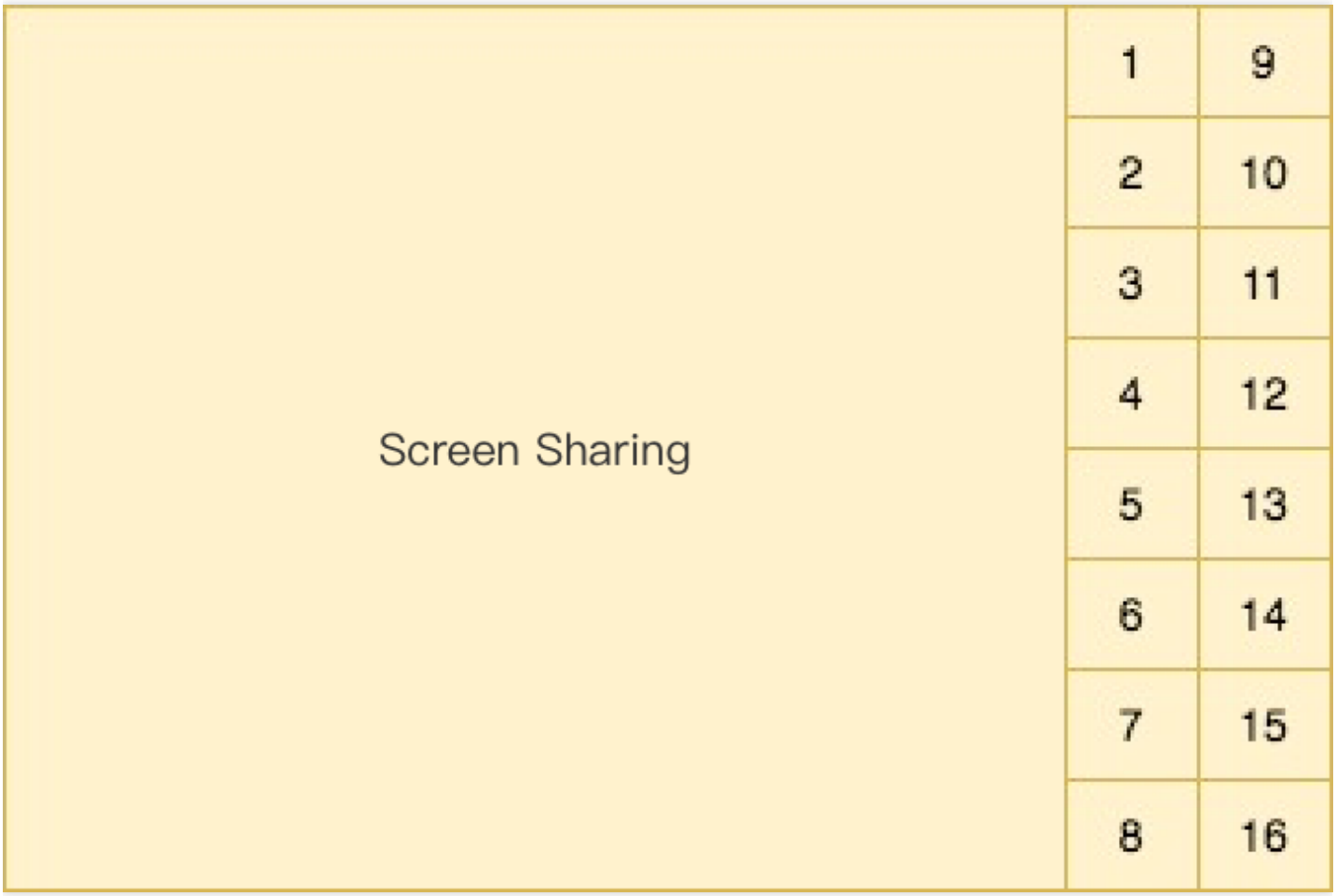
- サブ画面が17枚を超える場合:
  右（下）側の小画面の幅はキャンバス全体の幅の1/10、右（下）側の小画面の高さはキャンバス全体の高さの1/8となります。
  左側の大画面の幅はキャンバス全体の幅の4/5、左側の大画面の高さはキャンバス全体の高さの7/8となります。

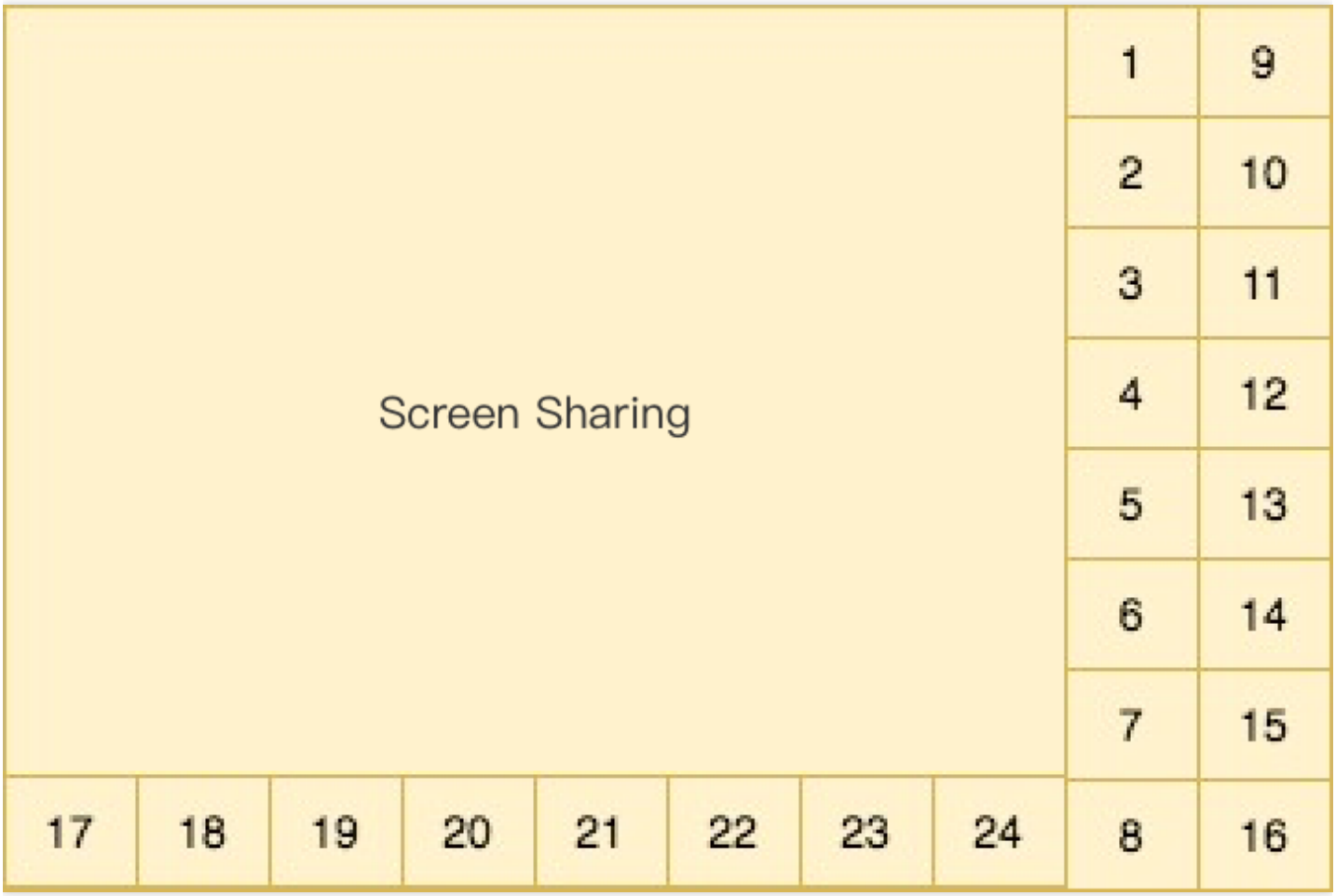
画面共有レイアウトはサブスクリプションのサブ画面が増えるに従って、下図のように変化します。











カスタムレイアウト：

業務の必要性に応じ、MixLayoutList内で各キャスター画面のレイアウト情報をカスタマイズできます。

2. レコーディング照会（DescribeCloudRecording）

必要に応じ、このインターフェースを呼び出してレコーディングサービスのステータスを照会することができます。レコーディングタスクが存在する場合にのみ、情報の照会が可能なことにご注意ください。レコーディングタスクがすでに終了している場合はエラーが返されます。
レコーディングしたファイルのリスト（StorageFile）には、今回レコーディングしたすべてのM3U8インデックスファイルおよびレコーディング開始時のUnixタイムスタンプ情報が含まれます。VODにアップロードしたタスクについては、このインターフェースから返されるStorageFileは空となることにご注意ください。

3. レコーディング更新（ModifyCloudRecording）

必要に応じ、このインターフェースを呼び出してレコーディングサービスのパラメータを変更することができます。例えば、サブスクリプションのブラックリスト/ホワイトリストSubscribeStreamUserIds（シングルストリーミングとミクスストリーミングレコーディングで有効）、レコーディングテンプレートのパラメータMixLayoutParams（ミクスストリーミングレコーディングで有効）などがあります。更新操作はすべて上書きの

操作であり、増分更新の操作ではないことにご注意ください。毎回の更新時に、テンプレートパラメータMixLayoutParamsおよびブラックリスト/ホワイトリストSubscribeStreamUserIdsを含む、全情報を引き継ぐ必要があるため、以前のレコーディング開始パラメータまたは再計算した完全なレコーディング関連パラメータを保存する必要があります。

4. レコーディング停止（DeleteCloudRecording）

レコーディング終了後に、レコーディング停止（DeleteCloudRecording）インターフェースを呼び出してレコーディングタスクを終了する必要があります。これを行わなかった場合、レコーディングタスクは設定したタイムアウト時間MaxIdleTimeになると自動的に終了します。注意が必要な点は、MaxIdleTimeの定義が、ルーム内に継続的にキャスターがいない状態がMaxIdleTimeの時間を超えることを指す点です。ルーム内にキャスターが存在しているが、キャスターがデータをアップストリームしていないという場合は、タイムアウト時間のカウント状態に入りません。レコーディング終了時にこのインターフェースを呼び出してレコーディングタスクを終了することをお勧めします。

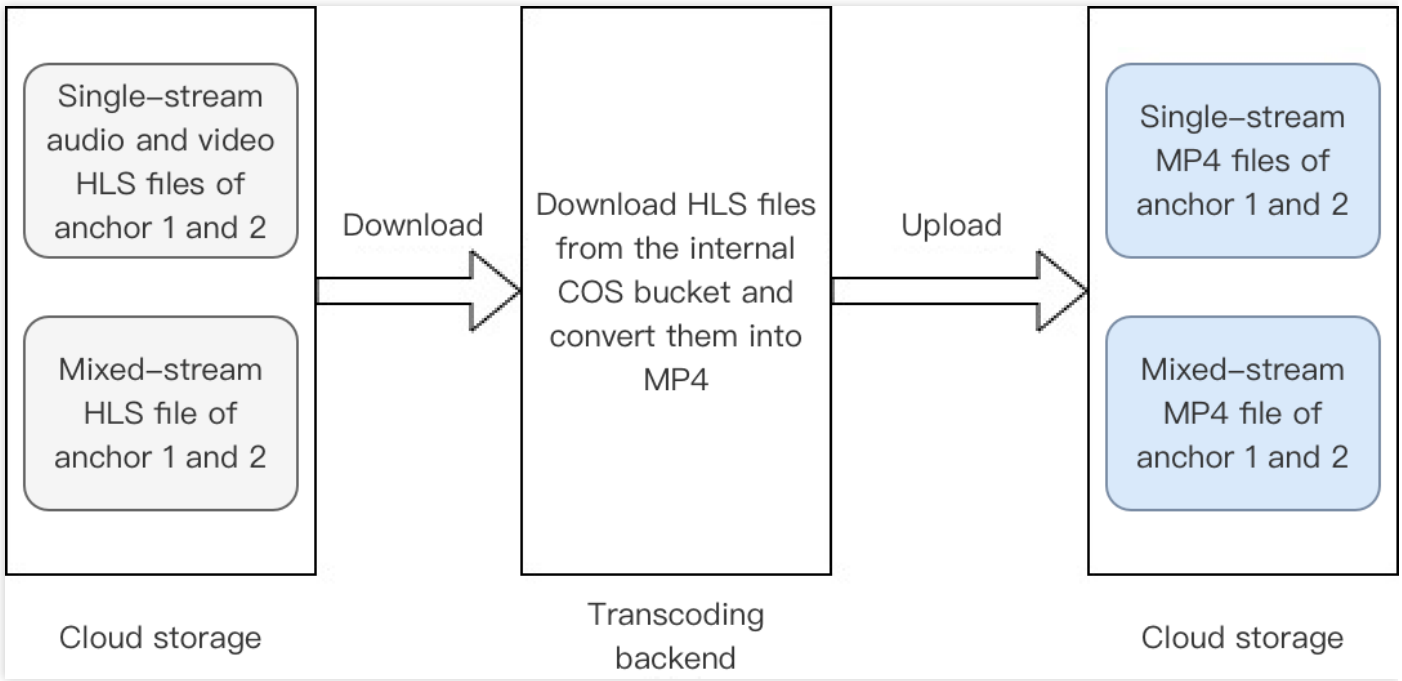
高度な機能

mp4ファイルのレコーディング

レコーディングの出力ファイル形式をmp4形式にしたい場合は、レコーディング開始パラメータ(OutputFormat)でレコーディングファイルの出力形式をhls+mp4に指定することができます。デフォルトではhlsファイルへのレコーディングのままですが、hlsレコーディングの終了後、ただちにmp4への変換タスクがトリガーされ、hlsの存在するcos内からプルされてmp4パッケージ化が行われ、お客様のcosにアップロードされます。
ここではCOSのファイルプル権限を提供する必要があることにご注意ください。これを行わなければ、mp4変換タスクはhlsファイルのプル失敗によって中止されます。
mp4ファイルは次のような場合、強制的に分割されます。

1. レコーディングファイルの長さが24時間を超えた場合。
2. 1個のmp4ファイルのサイズが2GBに達した場合。

具体的なフローは図のとおりです

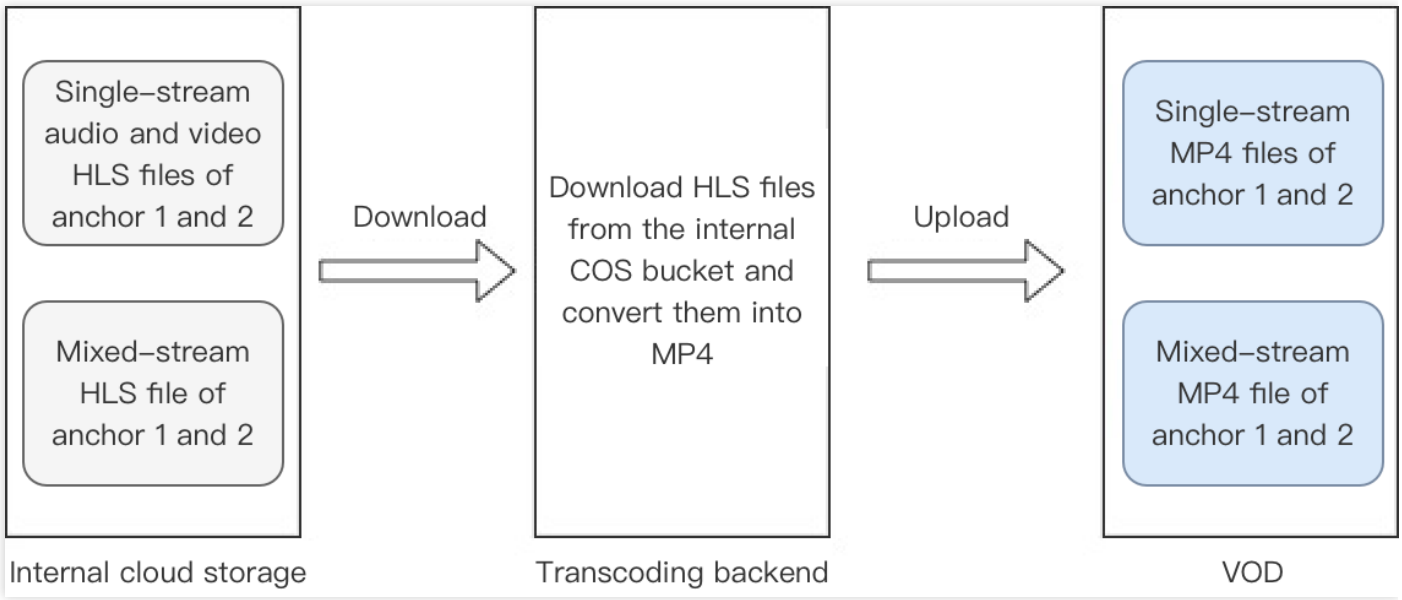


VODへのレコーディング

レコーディングした出力ファイルをVODプラットフォームにアップロードしたい場合は、レコーディング開始のストレージパラメータ(StorageParams)の中でCloudVodメンバーパラメータを指定することができます。レコーディングバックエンドはレコーディング終了後、レコーディングしたmp4ファイルを指定された方式でVODプラットフォームにアップロードし、コールバックの形式で再生アドレスをお客様に送信します。レコーディングモードがシングルストリーミングレコーディングモードの場合は、各サブスクリプションのキャスターが1つずつ対応する再生アドレスを持っています。レコーディングモードがミクスストリーミングレコーディングモードの場合は、ミクスストリーミング後のメディアの再生アドレスは1つのみとなります。VODアップロードタスクには次のいくつかの注意事項があります。

1. ストレージパラメータ中のCloudVodとCloudStorageは同時に1つしか指定できません。それ以上を指定した場合はレコーディングの開始に失敗します。
2. VODアップロードタスクの過程でDescribeCloudRecordingを使用して照会したタスクステータスには、レコーディングファイルの情報が含まれません。

具体的なフローは図のとおりです。ここに記載したクラウドストレージはレコーディング内部のクラウドストレージであり、お客様が関連のパラメータを入力する必要はありません。



シングルストリームファイル結合スクリプト

シングルストリームオーディオビデオファイル結合スクリプトをご提供しています。シングルストリームレコーディングのオーディオビデオファイルを結合してMP4ファイルとするために用います。

> 説明：
> 2つの分割ファイル間の時間間隔が15秒を超え、その間に何のオーディオ/ビデオ情報もない場合（サブストリームを無効にしている場合、サブストリーム情報は無視できます）、2つの分割ファイルはそれぞれ異なるセグメントであると判断されます。それらのうちレコーディング時間が早い方の分割ファイルが前のセグメントの終了分割ファイル、レコーディング時間が遅い方の分割ファイルが後のセグメントの開始分割ファイルとそれぞれ判断されます。

- セグメントモード（-m 0）
  このモードでは、スクリプトは各UserId下のレコーディングファイルをセグメントごとに結合し、1つのUserId下のセグメントがそれぞれのファイルに独立して結合されます。
- 結合モード（-m 1）
  同一のUserId下の全セグメントを1つのオーディオビデオファイルとして結合します。-sオプションを利用して、各セグメント間の間隔を埋めるかどうかを選択できます。

依存環境

Python3

- Centos: sudo yum install python3
- Ubuntu: sudo apt-get install python3

Python3依存パッケージ

- sortedcontainers：pip3 install sortedcontainers

使用方法

1.結合スクリプトを実行します：python3 TRTC_Merge.py [option]
2.レコーディングファイルディレクトリ下で結合後のmp4ファイルを生成します
例：python3 TRTC_Merge.py -f /xxx/file -m 0

オプションパラメータおよび対応する機能は下表のとおりです。

パラメータ	機能
-f	結合対象のファイルのストレージパスを指定します。複数のUserIdを持つレコーディングファイルの場合、スクリプトはそれぞれ結合操作を行います。
-m	0：セグメントモード(デフォルト設定)。このモードでは、スクリプトは各UserId下のレコーディングファイルをセグメントごとに結合します。各UserIdにつき、複数のファイルが生成される可能性があります。 1：結合モード。1つのUserId下の全オーディオビデオファイルを1つのファイルとして結合します。
-s	保存モード。このパラメータを設定すると、結合モードにおけるセグメント間の空白部分が削除されます。ファイルの実際の時間は物理時間より短くなります。
-a	0: メインストリーム結合(デフォルト設定)。同一のUserIdのメインストリームをオーディオと結合させます。サブストリームはオーディオと結合しません。 1: 自動結合。メインストリームが存在する場合は、メインストリームをオーディオと結合し、メインストリームが存在せずサブストリームが存在する場合は、サブストリームをオーディオと結合します。 2: サブストリーム結合。同一のUserIdのサブストリームをオーディオと結合させます。メインストリームはオーディオと結合しません
-p	出力するビデオのfpsを指定します。デフォルトでは15fps、有効範囲は5～120fpsです。5fps未満は5fpsとして、120fpsを超える場合は120fpsとしてカウントします。
-r	出力するビデオの解像度を指定します。-r 640 360であれば、出力するビデオの幅が640、高さが360であることを表します。

ファイル名のルール

- オーディオビデオファイル名のルール：UserId_timestamp_av.mp4

- オーディオのみのファイル名のルール：UserId_timestamp.m4a

説明：
ここでのUserIdはレコーディングストリームのユーザーIDの特殊なbase64値です。詳細についてはレコーディング開始の項のファイル名命名ルールに関する説明をご参照ください。timestampは各セグメントの最初のts分割ファイルのレコーディング開始時間となっています。

コールバックインターフェース

コールバックを受信するHttp/Httpsサービスゲートウェイを設けてコールバックメッセージをサブスクライブすることができます。関連するイベントが発生した際、クラウドレコーディングシステムによってイベント通知がお客様のメッセージ受信サーバーにコールバックされます。

イベントコールバックメッセージ形式

イベントコールバックメッセージは、HTTP/HTTPS POSTリクエストとしてサーバーに送信されます。以下がその詳細となります。
文字エンコード形式：UTF-8。
リクエスト：bodyの形式はJSONです。
応答：HTTP STATUS CODE = 200、サーバーは応答パケットの具体的な内容を無視します。プロトコルとの親和性のため、クライアントの応答内容に、JSON：`{"code":0}` を付けることをお勧めします。

パラメータの説明

イベントコールバックメッセージのheaderには、次のフィールドが含まれています。

フィールド名	値
Content-Type	application/json
Sign	署名値
SdkAppId	sdk application id

イベントコールバックメッセージのbodyには、次のフィールドが含まれています。

フィールド名	タイプ	意味
EventGroupId	Number	イベントグループID。クラウドレコーディングでは3に固定されています

フィールド名	タイプ	意味
EventType	Number	コールバック通知のイベントタイプ
CallbackTs	Number	イベントコールバックサーバーからお客様のサーバーに送信されるコールバックリクエストのUnixタイムスタンプ。単位はミリ秒
EventInfo	JSON Object	イベント情報

イベントタイプの説明

フィールド名	タイプ	意味
EVENT_TYPE_CLOUD_RECORDING_RECORDER_START	301	クラウドレコーディング レコーディングモジュール起動
EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP	302	クラウドレコーディング レコーディングモジュール退出
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START	303	クラウドレコーディング アップロードモジュール起動
EVENT_TYPE_CLOUD_RECORDING_FILE_INFO	304	クラウドレコーディング m3u8インデックスファイルを生成し、初回生成とアップロードの成功後にコールバック
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP	305	クラウドレコーディング アップロード終了
EVENT_TYPE_CLOUD_RECORDING_FAILOVER	306	クラウドレコーディング マイグレーションが発生し、元のレコーディングタスクが新たなロードに移行した際にトリガー

フィールド名	タイプ	意味
EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE	307	クラウドレコーディング M3U8ファイル(最初のts分割ファイルの切り出し) 生成後にコールバック
EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR	308	クラウドレコーディング アップロードモジュールにエラー発生
EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR	309	クラウドレコーディング デコード画像ファイルのダウンロード時にエラー発生
EVENT_TYPE_CLOUD_RECORDING_MP4_STOP	310	クラウドレコーディング mp4レコーディングタスク終了、コールバックにはレコーディングしたmp4ファイル名を含める
EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT	311	クラウドレコーディング vodレコーディングタスクメディアリソースアップロード完了
EVENT_TYPE_CLOUD_RECORDING_VOD_STOP	312	クラウドレコーディング vodレコーディングタスク終了

イベント情報の説明

フィールド名	タイプ	意味
RoomId	String/Number	ルーム名（タイプはクライアントのルーム番号タイプと同様）
EventTs	Number	時間で発生するUnixタイムスタンプ。単位は秒
UserId	String	レコーディングボットのユーザーID
TaskId	String	レコーディングID。1回のクラウドレコーディングタスクの固有ID
Payload	JsonObject	イベントタイプごとに異なる定義

イベントタイプ301 EVENT_TYPE_CLOUD_RECORDING_RECORDER_START時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0：レコーディングモジュール起動成功。1：レコーディングモジュール起動失敗。

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

イベントタイプ302 EVENT_TYPE_CLOUD_RECORDING_RECORDER_STOP時のPayloadの定義

フィールド名	タイプ	意味
LeaveCode	Number	0：レコーディングモジュールが正常に呼び出し、レコーディングを停止して退出した。 1: レコーディングボットがクライアントから強制退室させられた。 2：クライアントがルームを解散した。 3：サーバーがレコーディングボットを強制退室させた。 4：サーバーがルームを解散した。 99：ルーム内にレコーディングボット以外のユーザーストリームがなく、指定の時間を超過したため退出した。 100：ルームのタイムアウトにより退出した。 101：同一のユーザーが同じルームに重複して入室したため、ボットを退出させた

```
{
"EventGroupId": 3,
"EventType": 302,
"CallbackTs": 1622186354806,
"EventInfo": {
"RoomId": "xx",
```

```
"EventTs": "1622186354",
"UserId": "xx",
"TaskId": "xx",
"Payload": {
"LeaveCode": 0
}
}
}
```

イベントタイプ303 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_START時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0：アップロードモジュールが正常に起動 1：アップロードモジュール初期化失敗。

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

イベントタイプ304 EVENT_TYPE_CLOUD_RECORDING_FILE_INFO時のPayloadの定義

フィールド名	タイプ	意味
FileList	String	生成したM3U8ファイル名

```
{
"EventGroupId": 3,
"EventType": 304,
"CallbackTs": 1622191965350,
"EventInfo": {
"RoomId": "20015",
"EventTs": 1622191965,
"UserId": "xx",
```

```
"TaskId": "xx",
"Payload": {
"FileList": "xx.m3u8"
}
}
}
```

イベントタイプ305 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_STOP時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0: 今回のレコーディングアップロードタスクは完了し、指定されたサードパーティクラウドストレージに全ファイルがアップロードされた 1：今回のレコーディングアップロードタスクは完了しているが、少なくとも1つ以上のファイルがサーバーまたはバックアップストレージ上で滞留している 2: サーバーまたはバックアップストレージ上で滞留していたファイルが復元され、指定されたサードパーティクラウドストレージにアップロードされた

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

イベントタイプ306 EVENT_TYPE_CLOUD_RECORDING_FAILOVER時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0：今回のマイグレーションはすでに完了

```
{
"EventGroupId": 3,
"EventType": 306,
"CallbackTs": 1622191989674,
"EventInfo": {
```

```
"RoomId": "20015",
"EventTs": 1622191989,
"UserId": "xx",
"TaskId": "xx",
"Payload": {
"Status": 0
}
}
}
```

イベントタイプ307 EVENT_TYPE_CLOUD_RECORDING_FILE_SLICE時のPayloadの定義

フィールド名	タイプ	意味
FileName	String	m3u8ファイル名
UserId	String	このレコーディングファイルに対応するユーザーID
TrackType	String	audio/video/audio_video
BeginTimeStamp	Number	レコーディング開始時のサーバーUnixタイムスタンプ（ミリ秒）

```
{
"EventGroupId": 3,
"EventType": 307,
"CallbackTs": 1622186289148,
"EventInfo": {
"RoomId": "xx",
"EventTs": "1622186289",
"UserId": "xx",
"TaskId": "xx",
"Payload": {
"FileName": "xx.m3u8",
"UserId": "xx",
"TrackType": "audio",
"BeginTimeStamp": 1622186279144
}
}
}
```

イベントタイプ308 EVENT_TYPE_CLOUD_RECORDING_UPLOAD_ERROR時のPayloadの定義

フィールド名	タイプ	意味
Code	String	サードパーティクラウドストレージから返されるcode

フィールド名	タイプ	意味
Message	String	サードパーティクラウドストレージから返されるメッセージの内容


```
{
"Code": "InvalidParameter",
"Message": "AccessKey invalid"
}
{
"EventGroupId": 3,
"EventType": 308,
"CallbackTs": 1622191989674,
"EventInfo": {
"RoomId": "20015",
"EventTs": 1622191989,
"UserId": "xx",
"TaskId": "xx",
"Payload": {
"Code": "xx",
"Message": "xx"
}
}
}
```


イベントタイプ309 EVENT_TYPE_CLOUD_RECORDING_DOWNLOAD_IMAGE_ERROR時のPayloadの定義

フィールド名	タイプ	意味
Url	String	ダウンロードに失敗したurl


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

イベントタイプ310 EVENT_TYPE_CLOUD_RECORDING_MP4_STOP時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0: 今回のmp4レコーディングタスクは正常に終了し、指定されたサードパーティクラウドストレージに全ファイルがアップロードされた 1：今回のmp4レコーディングタスクは正常に終了しているが、少なくとも1つ以上のファイルがサーバーまたはバックアップストレージ上で滞留している 2: 今回のmp4レコーディングタスクは正常に終了しなかった(cosのhlsファイルのプルに失敗したことが原因の可能性あり)
FileList	String	生成したM3U8ファイル名

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

イベントタイプ311 EVENT_TYPE_CLOUD_RECORDING_VOD_COMMIT時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0: このレコーディングファイルはVODプラットフォームに正常にアップロードされた 1：このレコーディングファイルがサーバーまたはバックアップストレージ上で滞留している 2: このレコーディングファイルのVODアップロードタスクに異常が発生した
UserId	String	このレコーディングファイルに対応するユーザーID（レコーディングモードがミクスストリーミングモードの場合、このフィールドは空）
TrackType	String	audio/video/audio_video
MediaId	String	main/aux

フィールド名	タイプ	意味
FileId	String	このレコーディングファイルのVODプラットフォームでの固有id
VideoUrl	String	このレコーディングファイルのVODプラットフォームでの再生アドレス
CacheFile	String	このレコーディングファイルに対応するmp4ファイル名（VODアップロード前）
Errmsg	String	statueが0ではない場合、対応するエラー情報

アップロード成功のコールバック：

```
{
"EventGroupId": 3,
"EventType": 311,
"CallbackTs": 1622191965320,
"EventInfo": {
"RoomId": "20015",
"EventTs": 1622191965,
"UserId": "xx",
"TaskId": "xx",
"Payload": {
"Status": 0,
"TencentVod": {
"UserId": "xx",
"TrackType": "audio_video",
"MediaId": "main",
"FileId": "xxxx",
"VideoUrl": "http://xxxx"
}
}
}
}
```

アップロード失敗のコールバック：

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

```
"Payload": {
"Status": 1,
"Errmsg": "xxx",
"TencentVod": {
"UserId": "123",
"TrackType": "audio_video",
"CacheFile": "xxx.mp4"
}
}
}
}
```


イベントタイプ312 EVENT_TYPE_CLOUD_RECORDING_VOD_STOP時のPayloadの定義

フィールド名	タイプ	意味
Status	Number	0: 今回のvodアップロードタスクは正常に終了した 1：今回のvodアップロードタスクは正常に終了しなかった


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


ベストプラクティス

レコーディングの高可用性を保証するため、Restful APIを統合する際は、次のいくつかの点に注意するようお勧めします。

1. CreateCloudRecordingリクエストを呼び出した後は、http responseに注目してください。リクエストが失敗した場合は、具体的なステータスコードに応じて必要なリトライポリシーをとる必要があります。

エラーコードは「1次エラーコード」と「2次エラーコード」の組み合わせから成り、「InvalidParameter.SdkAppId」のようになっています。

- **返されたCodeがInternalError.xxxxxであれば、サーバーエラーに遭遇したことを示しています。リターンが正常となり、taskidが得られるまで、同じパラメータを使用して何度かリトライすることができます。例えば、1回目は3sのリトライ、2回目は6sのリトライ、3回目は12sのリトライというように、バックオフを使用してポリシーをリトライすることをお勧めします。**
- 返されたCodeがInValidParameter.xxxxxであれば、入力したパラメータに誤りがあることを示しています。表示に従ってパラメータをチェックしてください。
- 返されたCodeがFailedOperation.RestrictedConcurrencyであれば、お客様の同時レコーディングタスク数がバックエンドの予備リソースを超過したことを示しています(デフォルトでは100チャネル)。Tencent Cloudテクニカルサポートにご連絡いただき、最大同時チャネル数制限の調整を依頼してください。

2. CreateCloudRecordingを呼び出す際、指定するUserId/UserSigはレコーディングを行う単独のボットユーザーがルームに入室する際のidですので、TRTCルーム内のその他のユーザーと重複しないようにしてください。また、TRTCクライアントが入室するルームのタイプはレコーディングインターフェースが指定するルームのタイプと常に同じものにする必要があります。例えば、SDKがルームを新規作成する際に文字列のルームナンバーを使用した場合は、クラウドレコーディングのルームタイプもそれに合わせて文字列のルームナンバーに設定する必要があります。

3. レコーディングステータスの照会を行う場合、次のいくつかの方法でレコーディングに対応するファイル情報を取得することができます。

   - CreateCloudRecordingタスクの開始に成功してから約15秒後に、DescribeCloudRecordingインターフェースを呼び出してレコーディングファイルに対応する情報を照会します。ステータスがidleであることがわかった場合、レコーディングがアップストリームされたオーディオビデオストリームをプルできていなかったことを示しています。ルーム内にアップストリーム中のキャスターがいるかどうかを確認してください。
   - CreateCloudRecordingの開始に成功した後、ルーム内にオーディオビデオのアップストリームが存在することが確実な状況では、レコーディングファイル名の生成ルールに従ってレコーディングファイル名をスプライシングすることができます。具体的なファイル名ルールについては上記をご参照ください(ここにファイル名ルールへのリンクを貼る)。
   - レコーディングファイルのステータスはコールバックによってお客様のサーバーに送信されます。関連するコールバックをサブスクライブすると、レコーディングファイルのステータス情報を受け取ることができます。（ここにコールバックページへのリンクを貼る）
   - Cosを通じてレコーディングファイルを照会します。クラウドレコーディングの開始時点でcosに保存されているディレクトリを指定し、レコーディングタスクの終了後に、対応するディレクトリからレコーディングファイルを見つけることができます。

4. レコーディングユーザー(userid)のusersigの有効期限は、レコーディングタスクの有効期限よりも長く設定する必要があります。レコーディングタスクマシンのネットワーク接続が切れた後、内部高可用性が有効になり、レコーディングを再開しようするときに、usersigの期限切れによって失敗することを防止できます。