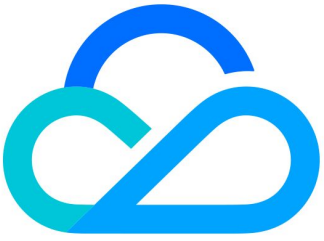
实时音视频

解决方案



腾讯云

文档目录

解决方案
第三方服务迁移
Agora
迁移指引
iOS
Android
API 参考
iOS
Android
第三方美颜
Android&iOS
实时合唱
组件介绍（TUIKaraoke）
快速集成（TUIKaraoke）
Android
iOS
方案介绍（TUIKaraoke）
实现步骤
歌曲同步
Android
iOS
歌词同步
Android
iOS
人声同步
Android
iOS
混流方案
Android
iOS
API 参考（TUIKaraoke）
Android
iOS
常见问题（TUIKaraoke）
Android
iOS

解决方案
第三方服务迁移
Agora
迁移指引
iOS

最近更新时间：2024-03-28 22:04:32

背景说明

AllInOneRTCEngine 适配了声网和腾讯两套 SDK，方便您同时集成声网和腾讯的服务，接口完全采用声网的接口调用及回调方式，仅需修改初始化逻辑，即可快速切换 SDK。
适配版本：

- 声网 AgoraRtcEngine_iOS： 4.3.0
- 腾讯 TRTC：11.7.15304

准备工作

- Xcode 13.0 或以上版本。
- 已安装 Cocoapods。如尚未安装 Cocoapods，参见 Getting Started with CocoaPods 安装说明。
- iOS 14.0 以上版本的设备。
- Apple开发者证书。

开通服务

使用 AllInOneRTCEngine 需要同时开通声网服务和 TRTC 服务。
开通声网服务参见 开通服务，创建项目，获取 appId 和 token。
开通TRTC服务步骤如下：

1. 进入 TRTC 控制台，如果没有创建应用，请先创建应用。
2. 进入应用概览，记录 `SDKAppID` 和 `SDKSecretKey` 。

集成SDK

cocoapod 集成

1、在终端里进入项目根目录，并运行 pod init 命令。项目文件夹下会生成一个 Podfile 文本文件。
2、打开 Podfile 文件，修改文件为如下内容。注意将 Target App 替换为你的 Target 名称。

```
platform :ios, '9.0'
target 'Target App' do
  # x.y.z 请填写具体的 SDK 版本号，如 4.0.1。
  pod 'AllInOneRTCEngine', 'x.y.z'
```

```
end
```

调用指引

1、初始化引擎

```
// 如果useTRTC传入YES，那么将使用TRTC进入房间，AppId输入TRTC控制台中获取的AppID
// 如果useTRTC传入NO，那么将使用Agora进入房间，AppId输入声网控制台中获取的AppID
_engine = [AIORTCEngineManager sharedEngineWithAppId:<#Your App ID#>
                                            delegate:self
                                            useTRTC: <#Use TRTC OR
Not#>];
```

参数	使用 TRTC	使用 agora
appID	TRTC 控制台中获取的 AppID	声网控制台中获取的 AppID
useTRTC	传入 YES，使用 TRTC	传入 NO，使用 agora

2、加入频道

```
AgoraRtcChannelMediaOptions *option = [[AgoraRtcChannelMediaOptions
alloc] init];
option.clientRoleType = AgoraClientRoleBroadcaster;
option.publishCameraTrack = YES;
option.publishMicrophoneTrack = YES;
option.autoSubscribeAudio = YES;
option.autoSubscribeVideo = YES;

[_engine joinChannelByToken:<#Your Token#>
                  channelId:<#Your Channel Name#>
                        uid:uid
                mediaOptions:option
                 joinSuccess:nil];
}
```

参数名	含义	使用 TRTC	使用 Agora

token	进房鉴权票据	您可以使用 AppId 和 uid 计算出 token，计算方法请参见 如何计算及使用 UserSig 。	参见 声网文档 获取临时token
channelId	房间号	字符串类型的房间号	字符串类型的房间号
uid	用户 ID	数字类型用户名	数字类型用户名

3、发布订阅音视频流

```
// 设置本地视图，并开启预览
AgoraRtcVideoCanvas *localCanvas = [[AgoraRtcVideoCanvas alloc] init];
localCanvas.uid = _param.userId.integerValue;
localCanvas.view = _localView;
localCanvas.renderMode = AgoraVideoRenderModeHidden;
[_engine setupLocalVideo:localCanvas];
[_engine startPreview];

//初始化远端用户视图，同时设置远端用户的视图在本地显示属性。
//你可以通过 didJoinedOfUid 回调获取远端用户的 uid。
AgoraRtcVideoCanvas *remoteCanvas = [[AgoraRtcVideoCanvas alloc] init];
remoteCanvas.uid = uid;
remoteCanvas.view = remoteView;
remoteCanvas.renderMode = AgoraVideoRenderModeHidden;
[_engine setupRemoteVideo:remoteCanvas];
```

4 退出房间

```
// 1、调用 leaveChannel 方法离开房间，相关的资源也会被释放。
[_engine leaveChannel:nil];
 // 2、调用 destroy 销毁引擎，并释放 SDK 中使用的所有资源。
 [AIORTCEngineManager destroy];
```

常见问题

RTC 线路选择：

当 APP 集成 Agora 和 TRTC 两套 SDK 时，两套 SDK 之间不互通，这里需要一个房间分配策略用于选择使用哪套 SDK ，这里提出两种方式供参空：

- 按房间号来分配，每次进房需要知道此房间是使用那种类型的 SDK。
- 按业务来区分，同一种业务全部使用一种类型 SDK。

关于房间号 TRTC 默认使用整型房间号，如果您需要字符串类型房间号，请联系**架构师**帮您开通。

Android

最近更新时间：2024-09-11 09:54:42

背景说明

AllInOneRTCEngine 适配了声网和腾讯两套 SDK，方便您同时集成声网和腾讯的服务，接口完全采用声网的接口调用及回调方式，仅需修改初始化逻辑，即可快速切换 SDK。当前适配版本：

- **声网 RtcEngine：4.3.0**
- **腾讯 TRTCCloud：11.7.0.13946**

准备工作

- 最低兼容 Android 4.1（SDK API Level 19），建议使用 Android 5.0（SDK API Level 21）及以上版本。
- Android Studio 3.5 及以上版本。
- App 要求 Android 4.1 及以上设备。

开通服务

使用 AllInOneRTCEngine 需要同时开通声网服务和 TRTC 服务，开通声网服务参见 开通服务，创建项目，获取 **appId** 和 **token**。
开通 TRTC 服务步骤如下：

1. 进入 TRTC 控制台，如果没有创建应用，请先创建应用。
2. 进入应用概览，记录 `SDKAppID` 和 `SDKSecretKey` 。

调试指引

导入并依赖 AllInOneRTCEngine 模块

导入 AllInOneRTCEngine 模块，以及 TRTC SDK（版本11.7.0.13946）和 Agora SDK（版本：4.3.0），在 app 模块下的 `build.gradle` 文件中输入以下代码：

```
dependencies {
....
// 依赖 AllInOneRTCEngine 模块（版本：1.0.0）。
implementation 'com.tencent.liteav:allinone-rtc-engine:latest.release'
// 依赖 TRTC SDK（版本：11.7.0.13946）。
implementation 'com.tencent.liteav:LiteAVSDK_TRTC:11.7.0.13946'
// 依赖 Agora SDK（版本：4.3.0）。
implementation 'io.agora.rtc:full-sdk:4.3.0'
}
```

成功依赖 "AllInOneRTCEngine" 模块、TRTC SDK、Agora SDK。

初始化引擎

1. 导入 AllInOneRTCEngine 包名。

```
import com.tencent.rtc.aio.AIORTCEngine;
import com.tencent.rtc.aio.RTCEngineManager;
```

2. 将 SDK 由类型 RtcEngine 改为类型 AIORTCEngine 。

```
// 修改前
private RtcEngine  mEngine;
// 修改后
private AIORTCEngine  mEngine;
```

3. 调用以下接口创建 AllInOneRTCEngine 实例，通过 参数enableTRTC 切换SDK。

```
RTCEngineManager.create(Context context, String appId,
IRtcEngineEventHandler listener, boolean enableTRTC)
```

参数：

参数名	含义
context	应用程序的环境信息，上下文。
appId	根据您想创建的 SDK 实例，输入对应 Agora 或 TRTC 的 appId。
listener	类型为 IRtcEngineEventHandler 用于 SDK 向 App 发送事件通知。
enableTRTC	true：创建的 SDK 实例为 TRTCCloud 类型。false：创建的 SDK 实例为 RtcEngine。

假设您创建的 SDK 实例变量名为 mEngine，切换到 TRTC SDK 的示例代码如下：

```
// 事件回调监听
static class TRTCCloudListenerImpl extends IRtcEngineEventHandler {
}
mTRTCListener = new TRTCCloudListenerImpl();
// 创建 TRTC SDK 实例：
mEngine = RTCEngineManager.create(this, appId, mTRTCListener, true);
```

加入频道

调用以下接口加入频道：

```
// 加入频道接口
joinChannel(String token, String channelId, String optionalInfo, int
uid);
```

参数：

参数名	含义	使用 TRTC	使用 Agora
token	进房鉴权票据	您可以使用 AppId 和 uid 计算出 token，计算方法请参见 如何计算及使用 UserSig 。	参见 声网文档 获取临时token
channelId	房间号	字符串类型的房间号	字符串类型的房间号
uid	用户 ID	数字类型用户名	数字类型用户名

结束互动

1. 调用以下接口离开频道：

```
leaveChannel();
```

2. 调用以下接口销毁 SDK 实例：

```
RTCEngineManager.destroy();
```

常见问题

RTC 线路如何选择?

当 APP 集成 Agora 和 TRTC 两套 SDK 时，两套 SDK 之间不互通，这里需要一个房间分配策略用于选择使用哪套 SDK ，这里提出两种方式供参考：

- 按房间号来分配，每次进房需要知道此房间是使用那种类型的 SDK。
- 按业务来区分，同一种业务全部使用一种类型 SDK。

关于房间号 TRTC 默认使用整型房间号，如果您需要字符串类型房间号，请联系**架构师**帮您开通。

API 参考
iOS

最近更新时间：2024-06-11 14:50:51

API 概览

频道相关接口

接口名	描述
joinChannel [1/2]	加入频道
joinChannel [2/2]	设置媒体选项并进入频道
leaveChannel	离开频道
setChannelProfile	设置频道场景
setClientRole [1/2]	设置用户角色
setClientRole [2/2]	设置用户角色
updateChannelWithMediaOptions	加入频道后更新频道媒体选项。

音频相关接口

接口名	描述
muteAllRemoteAudioStreams	取消或恢复订阅所有远端用户的音频流
muteLocalAudioStream	取消或恢复发布本地音频流
muteRemoteAudioStream	取消或恢复订阅指定远端用户的音频流
adjustUserPlaybackSignalVolume	调节本地播放的指定远端用户信号音量
adjustPlaybackSignalVolume	调节本地播放的所有远端用户信号音量
disableAudio	关闭音频模块

enableAudio	启动音频模块
enableAudioVolumeIndication	启用用户音量提示
setAudioProfile	设置音频编码属性
adjustRecordingSignalVolume	调节音频采集信号音量
enableInEarMonitoring	开启耳返功能
enableLocalAudio	开关本地音频采集和发布
setInEarMonitoringVolume	设置耳返音量
setEnableSpeakerphone	开启或关闭扬声器播放
isSpeakerphoneEnabled	检查扬声器启用状态
setDefaultAudioRoutetoSpeakerphone	设置默认的音频路由

视频相关接口

接口名	描述
enableVideo	启用视频模块
disableVideo	关闭视频模块
muteAllRemoteVideoStreams	取消或恢复订阅所有远端用户的视频流
muteLocalVideoStream	取消或恢复发布本地视频流
muteRemoteVideoStream	取消或恢复订阅指定远端用户的视频流
setVideoEncoderConfiguration	设置视频编码属性
startPreview	开启本地摄像头的预览画面
stopPreview	停止摄像头预览
setLocalRenderMode	设置本地画面的渲染参数
setRemoteRenderMode	设置远端画面的渲染模式

setupLocalVideo	初始化本地视图
setupRemoteVideo	初始化远端用户视图
switchCamera	选切换前置/后置摄像头

事件回调

接口名	描述
didClientRoleChanged	直播场景下用户角色已切换回调。
didClientRoleChangeFailed	直播场景下用户角色切换失败回调。
didJoinChannel	成功加入频道回调。
didLeaveChannelWithStats	离开频道回调。
didRejoinChannel	成功重新加入频道回调。
reportRtcStats	当前通话统计信息回调。
didJoinedOfUid	远端用户（通信场景）/主播（直播场景）加入当前频道回调。
didOfflineOfUid	远端用户（通信场景）/主播（直播场景）离开当前频道回调。
remoteAudioStats	通话中远端音频流的统计信息回调。
reportAudioVolumeIndicationOfSpeakers	用户音量提示回调。
firstLocalAudioFramePublished	已发布本地音频首帧回调。
localAudioStats	通话中本地音频流的统计信息回调。
remoteAudioStateChangedOfUid	远端音频流状态发生改变回调。
didAudioMuted	远端用户（通信场景）/主播（直播场景）停止或恢复发送音频流回调。
firstLocalVideoFrameWithSize	已显示本地视频首帧回调。
firstLocalVideoFramePublishedWithElapsed	已发布本地视频首帧回调。

firstRemoteVideoFrameOfUid	渲染器已接收首帧远端视频回调。
localVideoStateChangedOfState	本地视频状态发生改变回调。
localVideoStats	本地视频流统计信息回调。
remoteVideoStateChangedOfUid	远端视频状态发生改变回调。
remoteVideoStats	通话中远端视频流的统计信息回调。
videoSizeChangedOfSourceType	本地或远端视频大小和旋转信息发生改变回调。
didAudioRouteChanged	音频路由已发生变化回调。

三方美颜

setVideoFrameDelegate	注册原始视频观测器对象

API 详情

joinChannelByToken [1/2]（加入频道）

joinChannelByToken

```
- (int)joinChannelByToken:(NSString * _Nullable)token
                channelId:(NSString * _Nonnull)channelId
                     info:(NSString * _Nullable)info
                      uid:(NSUInteger)uid
              joinSuccess:(void(^ _Nullable)(NSString * _Nonnull
channel, NSUInteger uid, NSInteger elapsed))joinSuccessBlock;
```

详情:	该方法让用户加入通话频道，加入成功后本地会触发 didJoinChannel 回调。通信场景下的用户和直播场景下的主播加入频道后，远端会触发 didJoinedOfUid 回调。
参数:	token 用户签名（必填），当前 userId 对应的验证签名，相当于使用云服务的登录密码。具体使用方法参见： 如何使用 token。
	channelId 频道名，限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）：

	• 大小写英文字母（a-z A-Z） • 数字（0-9） • 空格、！ # $ % & ( ) + - : ; < = . > ? @ [ ] ^ _ { } | ~ ，。
	Info (非必选项) 预留参数。
	uid 用户 ID。该参数用于标识在实时音视频互动频道中的用户。
	joinSuccess 加入频道后的回调。joinSuccess 优先级高于 didJoinChannel，两个同时存在时，didJoinChannel 会被忽略。需要有 didJoinChannel 回调时，请将 joinSuccess 设置为 nil。

joinChannelByToken（设置媒体选项并进入频道）

joinChannelByToken

```
- (int)joinChannelByToken:(NSString * _Nullable)token
                channelId:(NSString * _Nonnull)channelId
                      uid:(NSUInteger)uid
             mediaOptions:(AgoraRtcChannelMediaOptions *
_Nonnull)mediaOptions
              joinSuccess:(void(^ _Nullable)(NSString * _Nonnull
channel, NSUInteger uid, NSInteger elapsed))joinSuccessBlock;
```

详情:	相比上一个 joinChannel[1/2] 方法，该方法该方法增加了 mediaOptions 参数，用于配置用户加入频道时是否自动订阅频道内所有远端音视频流。
参数:	**token** 用户签名（必填），当前 uid 对应的验证签名，相当于使用云服务的登录密码，具体使用方法参见： 如何使用token。
	channelId 频道名，限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： • 大小写英文字母（a-z A-Z） • 数字（0-9） • 空格、！ # $ % & ( ) + - : ; < = . > ? @ [ ] ^ _ { } | ~ ，。
	mediaOptions 频道媒体设置选项。详见 AgoraRtcChannelMediaOptions ，目前仅生效： publishCameraTrack、publishMicrophoneTrack、clientRoleType、autoSubscribeAudio、autoSubscribeVideo、channelProfile，其他设置不生效。

	uid 用户 ID。该参数用于标识在实时音视频互动频道中的用户。
	joinSuccess 优先级高于 didJoinChannel，两个同时存在时，didJoinChannel 会被忽略。需要有 didJoinChannel 回调时，请将 joinSuccess 设置为 nil。

leaveChannel（离开频道）

leaveChannel

```
- (int)leaveChannel:(void(^ _Nullable)(AgoraChannelStats * _Nonnull stat))leaveChannelBlock;
```

详情：	调用该接口使用户离开当前房间，并释放摄像头、麦克风、扬声器等资源，等资源释放完毕后触发以下回调： • 地：didLeaveChannelWithStats 回调。 • 远端：通信场景下的用户和直播场景下的主播离开频道后，远端会触发 didOfflineOfUid 回调。
参数：	leaveChannelBlock 成功离开频道的回调。leaveChannelBlock 优先级高于 didLeaveChannelWithStats，两个同时存在时，leaveChannelBlock 会被忽略。需要有 didLeaveChannelWithStats 回调时，请将 leaveChannelBlock 设置为 nil。

⚠ 注意：
leaveChannelBlock，回调中的通话相关的统计信息不生效，可以通过 reportRtcStats 回调获取统计信息。

setChannelProfile（设置频道场景）

setChannelProfile

```
- (int)setChannelProfile:(AgoraChannelProfile)profile;
```

详情：	SDK 会针对不同的使用场景采用不同的优化策略，以获取最佳的音视频传输体验。

参数:	profile 频道使用场景。详见 AgoraChannelProfile ，仅支持 AgoraChannelProfileCommunication 和 AgoraChannelProfileLiveBroadcasting。

> **注意:**
> 该方法必须在 joinChannel 前调用和进行设置，进入频道后无法再设置。

setClientRole（设置用户角色）

setClientRole

```
- (int)setClientRole:(AgoraClientRole)role;
```

详情:	只有加入频道后才可调用该方法设置用户角色。如果您在加入频道后调用该方法切换用户角色，调用成功后: • 本地触发 didClientRoleChanged。 • 远端触发 didJoinedOfUid 或 didOfflineOfUid 回调。
参数:	role 用户的角色: • CLIENT_ROLE_BROADCASTER (1):主播。 • CLIENT_ROLE_AUDIENCE (2):观众。

setClientRole（设置直播场景下的用户角色和级别）

setClientRole

```
- (int)setClientRole:(AgoraClientRole)role options:
(AgoraClientRoleOptions * _Nullable)options;
```

详情:	只有加入频道后才可调用该方法设置用户角色，该方法与 setClientRole [1/2] 效果一样，设置options参数无效。
参数:	role 用户的角色: CLIENT_ROLE_BROADCASTER (1):主播。

	CLIENT_ROLE_AUDIENCE (2):观众。
	options 用户具体设置，该参数无效。

> **注意：**
> 该方法只能修改角色，option 参数无效。

updateChannelWithMediaOptions（加入频道后更新频道媒体选项）

updateChannelWithMediaOptions

```
- (int)updateChannelWithMediaOptions:(AgoraRtcChannelMediaOptions*
_Nonnull)mediaOptions;
```

参数:	mediaOptions 媒体选项，详见 AgoraRtcChannelMediaOptions 。

> **注意：**
> mediaOptions 参数目前仅支持配置 publishCameraTrack、publishMicrophoneTrack、clientRoleType，其他设置不生效。

muteAllRemoteAudioStreams（取消或恢复订阅所有远端用户的音频流）

muteAllRemoteAudioStreams

```
- (int)muteAllRemoteAudioStreams:(BOOL)mute;
```

详情:	成功调用该方法后，本地用户会取消或恢复订阅所有远端用户的音频流。
参数:	mute 是否取消订阅所有远端用户的音频流： • true: 取消订阅所有远端用户的音频流。 • false:（默认）订阅所有远端用户的音频流。

muteLocalAudioStream（取消或恢复发布本地音频流）

muteLocalAudioStream

```
- (int)muteLocalAudioStream:(BOOL)mute;
```

详情：	成功调用该方法后，远端会触发 onUserMuteAudio 回调和 onRemoteAudioStateChanged 回调。
参数：	mute 是否取消订阅所有远端用户的音频流： • true: 取消发布本地音频流。 • false:发布本地音频流。

muteRemoteAudioStream（取消或恢复订阅指定远端用户的音频流）

muteRemoteAudioStream

```
- (int)muteRemoteAudioStream:(NSUInteger)uid mute:(BOOL)mute;
```

详情：	成功调用该方法后，远端会触发 onUserMuteAudio 回调和 onRemoteAudioStateChanged 回调。
参数：	uid 指定用户的用户 ID。
	mute 是否取消订阅所有远端用户的音频流： • true: 取消订阅指定用户的音频流。 • false:（默认）订阅指定用户的音频流。

⚠ 注意：

该方法需要在加入频道后调用。

adjustUserPlaybackSignalVolume（调节本地播放的指定远端用户信号音量）

adjustUserPlaySignalVolume

```
- (int)adjustUserPlaybackSignalVolume:(NSUInteger)uid volume:
(int)volume;
```

详情:	您可以在通话中调用该方法调节指定远端用户在本地播放的音量。如需调节多个用户在本地播放的音量，则需多次调用该方法。
参数:	uid 指定用户的用户 ID。
	volume 音量范围为 [0,100]。

adjustPlaybackSignalVolume（调节本地播放的所有远端用户信号音量）

adjustPlaybackSignalVolume

```
- (int)adjustPlaybackSignalVolume:(NSInteger)volume;
```

详情:	调节本地播放的所有远端用户信号音量。
参数:	volume 音量，取值范围为 [0,400]。

disableAudio（关闭音频模块）

disableAudio

```
- (int)disableAudio;
```

详情：	调用该接口后，音频相关接口将不生效。

enableAudio（启动音频模块）

enableAudio

```
- (int)enableAudio;
```

详情：	开启音频模块，调用该方法后，音频相关接口将生效。

enableAudioVolumeIndication（启用用户音量提示）

enableAudioVolumeIndicatio

```
- (int)enableAudioVolumeIndication:(NSInteger)interval
                           smooth:(NSInteger)smooth
                        reportVad:(BOOL)reportVad;
```

详情：	该方法允许 SDK 定期向 App 报告本地发流用户和远端的音量相关信息。启用该方法后，无论当前房间中是否有人说话，SDK 都会按照您设定的时间间隔定时抛出此 reportAudioVolumeIndicationOfSpeakers 回调。
参数：	**interval** 回调的触发间隔，单位为毫秒。 • ≤ 0: 关闭回调。 • > 0: 返回音量提示的间隔，单位为毫秒，最小间隔为 100ms，推荐值：300ms。
	smooth 平滑系数，该参数目前不生效。

	reportVad 是否开启本地人声检测功能，在 enableLocalAudio 之前调用才可以生效。

setAudioProfile（设置音频编码属性）

setAudioProfile

```
- (int)setAudioProfile:(AgoraAudioProfile)profile;
```

参数:	profile 音频编码属性，包含采样率、码率、编码模式和声道数。

注意:
该接口仅在进房前调用才会生效。

adjustRecordingSignalVolume（调节音频采集信号音量）

adjustRecordingSingalVolume

```
- (int)adjustRecordingSignalVolume:(NSInteger)volume;
```

参数:	volume 音量，取值范围为 [0,400]。

enableInEarMonitoring（开启耳返功能）

enableInEarMonitoring

```
- (int)enableInEarMonitoring:(BOOL)enabled;
```

详情：	成功调用该方法后，开启耳返功能
参数：	enabled 开启/关闭耳返功能： • true: 开启耳返功能。 • false: （默认）关闭耳返功能。

enableLocalAudio（开关本地音频采集和发布）

enableLocalAudio

```
- (int)enableLocalAudio:(BOOL)enabled;
```

详情：	SDK 默认不开启麦克风，当用户需要发布本地音频时，需要调用该接口开启麦克风采集，并将音频编码并发布到当前的房间中。开启本地音频的采集和发布后，房间中的其他用户会收到 didAudioMuted 的通知。
参数：	enabled 开启/关闭本地音频采集： • true: 开启。 • false: 关闭。

setInEarMonitoringVolume（设置耳返音量）

setInEarMonitoringVolume

```
- (int)setInEarMonitoringVolume:(NSInteger)volume;
```

详情：	该接口可设置耳返音量，取之范围[0,400]。
参数：	volume 音量大小，取值范围为[0,400]，默认值：400。

setEnableSpeakerphone（开启或关闭扬声器播放）

setEnableSpeakerphone

```
- (int)setEnableSpeakerphone:(BOOL)enableSpeaker;
```

详情:	设置“音频路由”，即设置声音是从手机的扬声器还是从听筒中播放出来，因此该接口仅适用于手机等移动端设备。成功改变音频路由后，SDK 会触发 didAudioRouteChanged 回调。
参数:	enabledSpeaker 设置是否开启扬声器播放: • true: 开启。音频路由为扬声器。 • false: 关闭。音频路由为听筒。

isSpeakerphoneEnabled（检查扬声器启用状态）

isSpeakerphoneEnabled

```
- (BOOL)isSpeakerphoneEnabled;
```

详情:	调用该接口可获得扬声器的状态（开启/关闭）
参数:	返回值: • true: 扬声器已开启，语音会输出到扬声器。 • false: 扬声器未开启，语音会输出到非扬声器（听筒，耳机等）。

setDefaultAudioRoutetoSpeakerphone（设置默认的音频路由）

setDefaultAudioRoutetoSpeakerphone

```
- (int)setDefaultAudioRouteToSpeakerphone:(BOOL)defaultToSpeaker;
```

详	调用该接口设置默认音频路由（听筒/扬声器）。

情:	
参数:	defaultToSpeaker 是否使用扬声器作为默认的音频路由: • true: 设置默认音频路由为扬声器。 • false: 设置默认音频路由为听筒。

enableVideo（启用视频模块）

enableVideo

```
- (int)enableVideo;
```

详情:	调用该方法后，可以正常使用视频相关接口。

disableVideo（关闭视频模块）

disableVideo

```
- (int)disableVideo;
```

详情:	调用该方法后，所有的视频相关接口均不生效。

muteAllRemoteVideoStreams（取消或恢复订阅所有远端用户的视频流）

muteAllRemoteVideoStreams

```
- (int)muteAllRemoteVideoStreams:(BOOL)mute;
```

详	该接口仅暂停/恢复接收所有用户的视频流，但并不释放显示资源，视频画面会被冻屏在接口调

情：	用时的最后一帧。
参数：	muted 是否取消订阅所有远端用户的视频流： • true: 取消订阅所有用户的视频流。 • false: （默认）订阅所有用户的视频流。

muteLocalVideoStream（取消或恢复发布本地视频流）

muteLocalVideoStream

```
- (int)muteLocalVideoStream:(BOOL)mute;
```

详情：	成功调用该方法后，远端用户会触发 remoteVideoStateChangedOfUid（AgoraVideoRemoteStateStarting or AgoraVideoRemoteStateStopped）回调。
参数：	muted 是否取消发送本地视频流： • true: 取消发送本地视频流。 • false: （默认）发送本地视频流。

muteRemoteVideoStream（取消或恢复订阅指定远端用户的视频流）

muteRemoteVideoStream

```
- (int)muteRemoteVideoStream:(NSUInteger)uid
                        mute:(BOOL)mute;
```

详情：	成功调用该方法后，会根据 userId 指定订阅远端用户的视频流
参数：	userId 指定用户的用户 ID。

	muted 是否取消订阅指定远端用户的视频流： • true: 取消订阅指定用户的视频流。 • false: （默认）订阅指定用户的视频流。

setVideoEncoderConfiguration（设置视频编码属性）

setVideoEncoderConfiguration

```
- (int)setVideoEncoderConfiguration:(AgoraVideoEncoderConfiguration
* _Nonnull)config;
```

详情:	设置本地视频的编码属性。每一种视频编码属性对应一系列视频相关参数设置，包含分辨率、帧率和码率。
参数:	**config** 用于设置视频编码器的相关参数。 **注意:** config仅支持dimensions、frameRate、bitrate、minbitrate、orientationMode、 mirrorMode几项配置，其他设置无效。 • bitrate推荐取值：请参见TRTCVideoResolution 在各档位注释的最佳码率，也可以在此基础上适当调高。例如：TRTCVideoResolution_1280_720 对应 1200kbps 的目标码率，您也可以设置为 1500kbps 用来获得更好的观感清晰度。minVideoBitrate推荐取值：您可以通过同时设置 videoBitrate 和 minVideoBitrate 两个参数，用于约束 SDK 对视频码率的调整范围。 • 如果您追求 弱网络下允许卡顿但要保持清晰 的效果，可以设置 minVideoBitrate 为 videoBitrate 的 60%。 • 如果您追求 弱网络下允许模糊但要保持流畅 的效果，可以设置 minVideoBitrate 为一个较低的数值（例如 100kbps）。 • 如果您将 videoBitrate 和 minVideoBitrate 设置为同一个值，等价于关闭 SDK 对视频码率的自适应调节能力。

startPreview（开启本地摄像头的预览画面）

startPreview

```
- (int)startPreview;
```

详情：	在 joinChannel 之前调用此函数，SDK 只会开启摄像头，并一直等到您调用 joinChannel 之后才开始推流。在 joinChannel 之后调用此函数，SDK 会开启摄像头并自动开始视频推流。调用该方法前，必须先调用 enableVideo开启视频功能、setupLocalVideo 初始化本地视图。

stopPreview（停止摄像头预览）

stopPreview

```
- (int)stopPreview;
```

详情：	调用 startPreview 开启预览后，如果您想关闭本地视频预览，请调用该方法。

setLocalRenderMode（设置本地画面的渲染参数）

setLocalRenderMode

```
- (int)setLocalRenderMode:(AgoraVideoRenderMode)mode
                   mirror:(AgoraVideoMirrorMode)mirror;
```

详情：	初始化本地用户视图后，您可以调用该方法更新本地用户视图的渲染和镜像模式。该方法只影响本地用户看到的视频画面，不影响本地发布视频。
参数：	**mode** 画面填充模式： • AgoraVideoRenderModeHidden：填充（画面可能会被拉伸裁剪） • AgoraVideoRenderModeFit：适应（画面可能会有黑边）
	mirror 画面镜像模式。

setRemoteRenderMode（设置远端画面的渲染模式）

setRemoteRenderMode

```
- (int)setRemoteRenderMode:(NSUInteger)uid
                      mode:(AgoraVideoRenderMode)mode
                    mirror:(AgoraVideoMirrorMode)mirror;
```

详情：	初始化远端用户视图后，您可以调用该方法更新远端用户视图在本地显示时的渲染和镜像模式。该方法只影响本地用户看到的视频画面。
参数：	uId 远端用户 ID。
	mode 画面填充模式： • AgoraVideoRenderModeHidden：填充（画面可能会被拉伸裁剪） • AgoraVideoRenderModeFit：适应（画面可能会有黑边）
	mirror 画面镜像模式。

setupLocalVideo（初始化本地视图）

setupLocalVideo

```
- (int)setupLocalVideo:(AgoraRtcVideoCanvas * _Nullable)local;
```

详情：	该方法初始化本地视图并设置本地用户视频显示属性，只影响本地用户看到的视频画面，不影响本地发布视频。调用该方法绑定本地视频流的显示视窗(view)，并设置本地用户视图的渲染模式和镜像模式。
参数：	local 本地视频显示属性。详见 AgoraRtcVideoCanvas 。

setupRemoteVideo（初始化远端用户视图）

setupRemoteVideo

```
- (int)setupRemoteVideo:(AgoraRtcVideoCanvas * _Nonnull)remote;
```

详情：	• 该方法绑定远端用户和显示视图，并设置远端用户视图在本地显示时的渲染模式和镜像模式，只影响本地用户看到的视频画面。 • 调用该方法时需要指定远端视频的用户 ID，一般可以在进频道前提前设置好。如果无法在加入频道前得到远端用户的 ID，可以在收到 didJoinedOfUid 回调时调用该方法。
参数：	remote 远端视频显示属性。详见 AgoraRtcVideoCanvas 。

switchCamera（选切换前置/后置摄像头）

switchCamera

```
- (int)switchCamera;
```

详情：	该方法必须在摄像头成功开启后调用，即 SDK 触发 onLocalVideoStateChanged 回调，返回本地视频状态为 LOCAL_VIDEO_STREAM_STATE_CAPTURING (1) 。

didClientRoleChanged（直播场景下用户角色已切换回调）

didClientRoleChanged

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
didClientRoleChanged:(AgoraClientRole)oldRole
newRole:(AgoraClientRole)newRole newRoleOptions:
(AgoraClientRoleOptions * _Nullable)newRoleOptions;
```

详情：	该回调是由本地用户在加入频道后调用 setClientRole 改变用户角色触发的。
参数：	oldRole 切换前的角色： • CLIENT_ROLE_BROADCASTER (1)：主播。

	• CLIENT_ROLE_AUDIENCE (2)：观众。
	newRole 切换后的角色： • CLIENT_ROLE_BROADCASTER (1)：主播。 • CLIENT_ROLE_AUDIENCE (2)：观众。
	newRoleOptions 切换后的角色属性，目前总是传空。

didClientRoleChangeFailed（直播场景下切换用户角色失败回调）

didClientRoleChangeFailed

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
didClientRoleChangeFailed:(AgoraClientRoleChangeFailedReason)reason
currentRole:(AgoraClientRole)currentRole;
```

详情：	直播场景下，本地用户加入频道后调用 setClientRole [2/2] 切换用户角色失败时，SDK 会触发该回调，报告切换失败的原因和当前的用户角色。
参数：	**reason** 切换用户角色失败的原因，注意，该参数目前不生效。
	currentRole 当前用户角色： • CLIENT_ROLE_BROADCASTER(1): 主播。主播可以发流也可以收流。 • CLIENT_ROLE_AUDIENCE(2): 观众。观众只能收流不能发流。

didJoinChannel（成功加入频道回调）

didJoinChannel

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
        didJoinChannel:(NSString * _Nonnull)channel withUid:
(NSUInteger)uid elapsed:(NSInteger) elapsed;
```

详情：	该回调方法表示该客户端成功加入了指定的频道。

参数：	engine 内部对象，注意，实际类型非AgoraRtcEngineKit。
	channel 频道名。
	uid 加入频道的用户 ID。
	elapsed 从本地调用 joinChannel [2/2] 开始到发生此事件过去的时间（毫秒）。

didLeaveChannelWithStats（离开频道回调）

didLeaveChannelWithStats

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    didLeaveChannelWithStats:(AgoraChannelStats * _Nonnull)stats;
```

详情：	App 调用 leaveChannel 方法时，SDK 提示 App 离开频道成功。
参数：	engine 内部对象，注意，实际类型非AgoraRtcEngineKit。
	stats 通话的统计数据: 该参数无效。

说明：
stats参数实际无效，可以通过reportRtcStats回调获取。

didRejoinChannel（成功重新加入频道回调）

didRejoinChannel

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    didRejoinChannel:(NSString * _Nonnull)channel withUid:
(NSUInteger)uid elapsed:(NSInteger) elapsed;
```

详情：	有时候由于网络原因，客户端可能会和服务器失去连接，SDK 会进行自动重连，自动重连成功后本地触发此回调方法。

参数：	engine 内部对象，注意，实际类型非AgoraRtcEngineKit。
	channel 频道名。
	uid 重新加入频道的用户 ID。
	elapsed 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。

reportRtcStats（当前通话统计信息回调）

reportRtcStats

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    reportRtcStats:(AgoraChannelStats * _Nonnull)stats;
```

详情：	SDK 定期向 App 报告当前通话的统计信息，每两秒触发一次。
参数：	engine 内部对象，注意，实际类型非AgoraRtcEngineKit。
	stats 通话的统计数据。

didJoinedOfUid（远端用户（通信场景）/主播（直播场景）加入当前频道回调）

didJoinedOfUid

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    didJoinedOfUid:(NSUInteger)uid elapsed:(NSInteger)elapsed;
```

详情：	该回调用于提示有远端用户加入频道,如果加入之前,已有其他用户在频道中,新加入的用户也会收到这些已有用户加入频道的回调。
参数：	engine 内部对象，注意，实际类型非AgoraRtcEngineKit。
	uid 新加入频道的远端用户/主播 ID。
	elapsed 从本地用户调用 joinChannel 到该回调触发的延迟（毫秒）。

didOfflineOfUid（远端用户（通信场景）/主播（直播场景）离开当前频道回调）

didOfflineOfUid

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    didOfflineOfUid:(NSUInteger)uid
    reason:(AgoraUserOfflineReason)reason;
```

详情：	该回调用于提示有远端用户离开频道。
参数：	engine 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	uid 离线用户或主播的用户 ID。
	reason 远端用户（通信场景）或主播（直播场景）下线的原因： • USER_OFFLINE_QUIT(0)：用户主动离开。此时离开频道的用户会发送一个类似“再见”的消息。收到该消息是，SDK 判定该用户离开频道。 • USER_OFFLINE_DROPPED(1)：因过长时间收不到对方数据包，SDK 判定该远端用户超时掉线。注意：在网络连接不稳定时，该判定 可能会有误。建议使用实时消息 SDK 来做可靠的掉线检测。 • USER_OFFLINE_BECOME_AUDIENCE(2)：用户的角色从主播切换为观众。

remoteAudioStats（通话中远端音频流的统计信息回调）

remoteAudioStats

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    remoteAudioStats:(AgoraRtcRemoteAudioStats * _Nonnull)stats;
```

详情：	该回调针对每个发送音频流的远端用户/主播每 2 秒触发一次。如果远端有多个用户/主播发送音频流，该回调每 2 秒会被触发多次。

参数：	engine 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	stats 接收到的远端音频统计数据，详见 AgoraRtcRemoteAudioStats 。

reportAudioVolumeIndicationOfSpeakers（用户音量提示回调）

reportAudioVolumeIndicationOfSpeakers

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
        reportAudioVolumeIndicationOfSpeakers:
(NSArray<AgoraRtcAudioVolumeInfo *> * _Nonnull)speakers
        totalVolume:(NSInteger)totalVolume;
```

详情：	该回调默认禁用，您可以通过 enableAudioVolumeIndication 开启。 开启后，无论当前房间中是否有人说话，SDK 都会按照您设定的时间间隔定时抛出此事件回调。每次会触发两个 reportAudioVolumeIndicationOfSpeakers 回调，一个报告本地发流用户的音量相关信息，另一个报告远端用户的音量相关信息。
参数：	engine 内部对象，注意，实际类型非AgoraRtcEngineKit。
	speakers 用户音量信息，详见 AgoraRtcAudioVolumeInfo 数组。如果 speakers 为空，则表示远端用户不发流或没有远端用户。
	totalVolume 所有远端用户的总音量大,取值范围 0 - 255。

firstLocalAudioFramePublished（已发布本地音频首帧回调）

firstLocalAudioFramePublished

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
firstLocalAudioFramePublished:(NSInteger)elapsed;
```

详情：	当 SDK 成功地向云端送出自己的第一帧音频数据帧以后，就会抛出 firstLocalAudioFramePublished 事件回调。在以下三个时机会触发该回调： • 当您成功进入房间并通过 enableLocalAudio 开启本地音频采集之后。 • 调用 muteLocalVideoStream(YES)，再调用 muteLocalVideoStream(NO) 后。

	• 在房间内，调用 disableVideo，再调用 enableVideo 后。
参数：	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	elapsed 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。

localAudioStats（通话中本地音频流的统计信息回调）

localAudioStats

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    localAudioStats:(AgoraRtcLocalAudioStats * _Nonnull)stats;
```

详情：	SDK 每 2 秒触发该回调一次。
参数：	**engine** 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	stats 本地音频统计数据。详见 AgoraRtcLocalAudioStats 。

remoteAudioStateChangedOfUid（远端音频流状态发生改变回调）

remoteAudioStateChangedOfUid

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    remoteAudioStateChangedOfUid:(NSUInteger)uid state:
(AgoraAudioRemoteState)state reason:(AgoraAudioRemoteReason)reason
elapsed:(NSInteger)elapsed;
```

详情：	远端用户（通信场景）或主播（直播场景）的音频状态发生改变时，SDK 会触发该回调向本地用户报告当前的远端音频流状态。
参数：	**engine** 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	uid 发生音频状态改变的远端用户 ID。
	state 远端音频流状态，远端音频流状态，详见 AgoraAudioRemoteState 。

	注意： state仅存在AgoraAudioRemoteStateStopped、AgoraAudioRemoteStateStarting、AgoraAudioRemoteStateDecoding三个状态。
	reason 远端音频流状态改变的具体原因，详见 AgoraAudioRemoteReason 。
	elapsed 从本地用户调用 joinChannel [2/2] 方法到发生本事件经历的时间，单位为毫秒。

didAudioMuted（远端用户（通信场景）/主播（直播场景）停止或恢复发送音频流回调）

didAudioMuted

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
didAudioMuted:(BOOL)muted byUid:(NSUInteger)uid;
```

详情：	该回调是由远端用户调用 muteLocalAudioStream 方法关闭或开启音频发送触发的。
参数：	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	uid 用户 ID。
	muted 该用户是否静音： • true: 该用户已将音频静音。 • false: 该用户取消了音频静音。

firstLocalVideoFrameWithSize（已显示本地视频首帧回调）

firstLocalVideoFrameWithSize

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
                  firstLocalVideoFrameWithSize:(CGSize)size
                  elapsed:(NSInteger)elapsed
```

```
sourceType:(AgoraVideoSourceType)sourceType;
```

详情：	本地视频首帧显示在本地视图上时，SDK 会触发此回调。
参数：	**engine** 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	size 本地视频首帧的宽高。
	elapsed 从调用 joinChannel 方法到触发该回调的时间间隔（毫秒）。
	sourceType 视频源类型，详见 AgoraVideoSourceType ，目前仅实现了 AgoraVideoSourceTypeCamera 类型的流。

firstLocalVideoFramePublishedWithElapsed（已发布本地视频首帧回调）

firstLocalVideoFramePublishedWithElapsed

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
                firstLocalVideoFramePublishedWithElapsed:
(NSInteger)elapsed
                sourceType:(AgoraVideoSourceType)sourceType;
```

详情：	当 SDK 成功地向云端送出自己的第一帧视频数据帧以后，就会抛出该事件回调。以下三个时机会触发该回调： • 当您成功进入房间并通过 startPreview 开启本地视频采集之后。 • 调用 muteLocalVideoStream(YES)，再调用 muteLocalVideoStream(NO) 后。 • 在房间内调用 disableVideo，再调用 enableVideo 后。
参数：	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	elapsed 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。
	sourceType 视频源类型，详见 AgoraVideoSourceType ，目前仅实现了 AgoraVideoSourceTypeCamera类型的流。

firstRemoteVideoFrameOfUid（渲染器已接收首帧远端视频回调）

firstRemoteVideoFrameOfUid

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
        firstRemoteVideoFrameOfUid:(NSUInteger)uid size:
(CGSize)size elapsed:(NSInteger)elapsed;
```

详情:	渲染器已接收首帧远端视频回调。
参数:	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	uid 远端用户 ID。
	size 视频流尺寸。
	elapsed 从调用 joinChannel 方法到触发该回调的时间间隔（毫秒）。

localVideoStateChangedOfState（本地视频状态发生改变回调）

localVideoStateChangedOfState

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
localVideoStateChangedOfState:(AgoraVideoLocalState)state reason:
(AgoraLocalVideoStreamReason)reason sourceType:
(AgoraVideoSourceType)sourceType;
```

详情:	目前只有摄像头准备就绪时会触发该回调。
参数:	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	state 本地视频状态，详见 AgoraVideoLocalState 。 **注意:** state只会有 AgoraVideoLocalStateCapturing 这一个状态
	reason 本地视频状态改变的具体原因，详见 AgoraLocalVideoStreamReason 。

	⚠ **注意：** reason只会有 AgoraLocalVideoStreamReasonOK 这一个reason。
	sourceType 视频源类型，详见 AgoraVideoSourceType ，目前仅实现了 AgoraVideoSourceTypeCamera 类型的流。

localVideoStats（本地视频流统计信息回调）

localVideoStats

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
 localVideoStats:(AgoraRtcLocalVideoStats * _Nonnull)stats
 sourceType:(AgoraVideoSourceType)sourceType;
```

详情：	该回调描述本地设备发送视频流的统计信息，每 2 秒触发一次。
参数：	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	stats 本地视频流统计信息。详见 AgoraRtcLocalVideoStats 。
	source 视频源的类型。详见 AgoraVideoSourceType 。目前仅实现了 AgoraVideoSourceTypeCamera类型的流。

remoteVideoStateChangedOfUid（远端视频状态发生改变回调）

remoteVideoStateChangedOfUid

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
     remoteVideoStateChangedOfUid:(NSUInteger)uid state:
(AgoraVideoRemoteState)state reason:(AgoraVideoRemoteReason)reason
elapsed:(NSInteger)elapsed;
```

详	远端视频状态发生改变回调。

参数：	**engine** 内部对象，注意，实际类型非AgoraRtcEngineKit。
	uid 发生视频状态改变的远端用户 ID。
	state 远端视频流状态。 ⚠ **注意：** state只有AgoraVideoRemoteStateStopped和AgoraVideoRemoteStateStarting两个状态。
	reason 端视频流状态改变的具体原因，该参数不生效。
	elapsed 从本地用户调用 joinChannel 方法到发生本事件经历的时间，单位为毫秒。

remoteVideoStats（通话中远端视频流的统计信息回调）

remoteVideoStats

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
    remoteVideoStats:(AgoraRtcRemoteVideoStats * _Nonnull)stats;
```

详情：	该回调描述远端用户在通话中端到端的视频流统计信息， 针对每个远端用户/主播每 2 秒触发一次。如果远端同时存在多个用户/主播， 该回调每 2 秒会被触发多次。
参数：	**engine** 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	stats 远端视频统计数据，详见 AgoraRtcRemoteVideoStats 。

videoSizeChangedOfSourceType（本地或远端视频大小和旋转信息发生改变回调）

videoSizeChangedOfSourceType

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
videoSizeChangedOfSourceType:(AgoraVideoSourceType)sourceType uid:
(NSString * _Nonnull)uid size:(CGSize)size rotation:
```

```
(NSInteger)rotation
NS_SWIFT_NAME(rtcEngine(_:videoSizeChangedOf:uid:size:rotation:));
```

详情:	该回调表示该路画面大小发生了调整，调整的原因可能是该用户调用了 setVideoEncoderParam 重新设置了画面尺寸。
参数:	**engine** 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	source 视频源的类型，详见 AgoraVideoSourceType 。目前仅实现了 AgoraVideoSourceTypeCamera 类型的流。
	uid 图像尺寸和旋转信息发生变化的用户 ID
	size 视频流的尺寸。
	rotation 旋转信息，旋转信息，取值范围 [0,360)。 **注意:** rotation参数值始终为 0。

didAudioRouteChanged（音频路由已发生变化回调）

didAudioRouteChanged

```
- (void)rtcEngine:(AgoraRtcEngineKit * _Nonnull)engine
didAudioRouteChanged:(AgoraAudioOutputRouting)routing;
```

详情:	当音频路由发生变化时，触发该回调。
参数:	**engine** 内部对象，注意，实际类型非 AgoraRtcEngineKit。
	routing 当前的音频路由。 **注意:**

setVideoFrameDelegate（注册原始视频观测器对象）

setVideoFrameDelegate

```
- (BOOL)setVideoFrameDelegate:(id<AgoraVideoFrameDelegate>
_Nullable)delegate;
```

详情：	设置该回调之后，SDK 会把采集到的视频帧通过您设置的 listener 回调出来，用于第三方美颜组件进行二次处理，之后 SDK 会将处理后的视频帧进行编码和发送。
参数：	observer 自定义预处理回调，当您使用 AllInOneRTCEngine 创建的实例为 TRTCCloud 类型时，需要注册 TRTCVideoFrameDelegate 类，您可以根据需要注册 TRTCVideoFrameDelegate 类中的回调。在成功注册视频观测器后，SDK 会在捕捉到每个视频帧时，触发您所注册的上述回调。而当您创建的实例为 RtcEngine 时，则需要注册 AgoraVideoFrameDelegate 类。

Android

最近更新时间：2024-06-07 15:12:02

API 概览

频道相关接口：

接口名	描述
joinChannel [1/2]	加入频道
joinChannel [2/2]	设置媒体选项并进入频道
leaveChannel	离开频道
setChannelProfile	设置频道场景
setClientRole [1/2]	设置用户角色
setClientRole [2/2]	设置直播场景下的用户角色和级别
updateChannelMediaOptions	加入频道后更新频道媒体选项

音频相关接口

接口名	描述
muteAllRemoteAudioStreams	取消或恢复订阅所有远端用户的音频流
muteLocalAudioStream	取消或恢复发布本地音频流
muteRemoteAudioStream	取消或恢复订阅指定远端用户的音频流
adjustUserPlaybackSignalVolume	调节本地播放的指定远端用户信号音量
adjustPlaybackSignalVolume	调节本地播放的所有远端用户信号音量
disableAudio	关闭音频模块
enableAudio	启用音频模块

enableAudioVolumeIndication	启用用户音量提示
setAudioProfile	设置音频编码属性
adjustRecordingSignalVolume	调节音频采集信号音量
enableInEarMonitoring	开启耳返功能
enableLocalAudio	开关本地音频采集
setInEarMonitoringVolume	设置耳返音量

路由设置

接口名	描述
setEnableSpeakerphone	开启或关闭扬声器播放
isSpeakerphoneEnabled	检查扬声器状态启用状态
setDefaultAudioRoutetoSpeakerphone	设置默认的音频路由
setRouteInCommunicationMode	选择通话音量模式下的音频路由

视频相关接口

接口名	描述
enableVideo	启用视频模块
disableVideo	关闭视频模块
muteAllRemoteVideoStreams	取消或恢复订阅所有远端用户的视频流
muteLocalVideoStream	取消或恢复发布本地视频流
muteRemoteVideoStream	取消或恢复订阅指定远端用户的视频流

setVideoEncoderConfiguration	设置视频编码属性
startPreview	开启视频预览
stopPreview	停止摄像头预览
setLocalRenderMode	设置本地画面的渲染参数
setRemoteRenderMode	设置远端画面的渲染模式
setupLocalVideo	初始化本地视图
setupRemoteVideo	初始化远端用户视图
switchCamera	选切换前置/后置摄像头

事件回调

接口名	描述
onError	发生错误回调
onClientRoleChanged	直播场景下用户角色已切换回调
onClientRoleChangeFailed	直播场景下切换用户角色失败回调
onJoinChannelSuccess	成功加入频道回调
onLeaveChannel	离开频道回调
onRejoinChannelSuccess	成功重新加入频道回调
onUserJoined	远端用户（通信场景）/主播（直播场景）加入当前频道回调
onUserOffline	远端用户（通信场景）/主播（直播场景）离开当前频道回调
onRtcStats	当前通话统计信息回调
onRemoteAudioStats	通话中远端音频流的统计信息回调
onLocalAudioStats	通话中本地音频流的统计信息回调
onRemoteVideoStats	通话中远端视频流的统计信息回调
onLocalVideoStats	本地视频流统计信息回调

onAudioVolumeIndication	用户音量提示回调
onFirstLocalAudioFramePublished	已发布本地音频首帧回调
onRemoteAudioStateChanged	远端音频流状态发生改变回调
onUserMuteAudio	远端用户（通信场景）/主播（直播场景）停止或恢复发送音频流回调
onLocalVideoStateChanged	本地视频状态发生改变回调
onRemoteVideoStateChanged	远端视频状态发生改变回调
onVideoSizeChanged	本地或远端视频大小和旋转信息发生改变回调
onAudioRouteChanged	音频路由已发生变化回调
onFirstLocalVideoFramePublished	已发布本地视频首帧回调
onFirstLocalVideoFrame	已显示本地视频首帧回调
onFirstRemoteVideoFrame	渲染器已接收首帧远端视频回

第三方美颜

registerVideoFrameObserver	注册原始视频观测器对象

API 详情

joinChannel [1/2]（加入频道）

joinChannel

```
abstract public int joinChannel(String token, String channelId,
String optionalInfo, int uid);
```

详情:	该方法让用户加入通话频道，加入成功后本地会触发 onJoinChannelSuccess 回调。通信场景下的用户和直播场景下的主播加入频道后，远端会触发 onUserJoined 回调。
参数:	**token** 用户签名（必填），当前 userId 对应的验证签名，相当于使用云服务的登录密码，具体使用方法参见： 如何使用token。
	channelId 频道名，限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： • 大小写英文字母（a-z A-Z） • 数字（0-9） • 空格、！ # $ % & ( ) + - : ; < = . > ? @ [ ] ^ _ { } | ~ ，。
	optionalInfo (非必选项) 预留参数。
	uid 用户 ID。该参数用于标识在实时音视频互动频道中的用户。

joinChannel [2/2]（设置媒体选项并进入频道）

joinChannel

```
abstract public int joinChannel(String token, String channelId, int
uid, ChannelMediaOptions options);
```

详情:	相比上一个 joinChannel[1/2] 方法，该方法该方法增加了 options 参数，用于配置用户加入频道时是否自动订阅频道内所有远端音视频流。
参数:	**token** 用户签名（必填），当前 userId 对应的验证签名，相当于使用云服务的登录密码，具体使用方法参见： 如何使用token。
	channelId 频道名，限制长度为 64 字节。以下为支持的字符集范围（共 89 个字符）： • 大小写英文字母（a-z A-Z） • 数字（0-9） • 空格、！ # $ % & ( ) + - : ; < = . > ? @ [ ] ^ _ { } | ~ ，。
	options 频道媒体设置选项，详见 ChannelMediaOptions ，目前仅生效： • publishCameraTrack 设置是否发布摄像头采集的视频：true：发布摄像头采集的视频。false：不发布摄像头采集的视频。 • publishMicrophoneTrack 设置是否发布麦克风采集到的音频：true：发布麦克风采集到的音频。false：不发布麦克风采集到的音频。

	• clientRoleType CLIENT_ROLE_BROADCASTER (1)：主播。CLIENT_ROLE_AUDIENCE (2)：观众。 • autoSubscribeAudio 设置是否自动订阅所有音频流：true：自动订阅所有音频流。false：不自动订阅任何音频流。 • autoSubscribeVideo 设置是否自动订阅所有视频流：true：自动订阅所有视频流。false：不自动订阅任何视频流。 • channelProfile 频道使用场景：CHANNEL_PROFILE_COMMUNICATION (0)：通信。声网推荐使用直播场景以获取更好的音视频体验。CHANNEL_PROFILE_LIVE_BROADCASTING (1)：（默认）直播。
	uid 用户 ID。该参数用于标识在实时音视频互动频道中的用户。

leaveChannel（离开频道）

leaveChannel

```
abstract public int leaveChannel();
```

详情：	调用该接口使用户离开当前房间，并释放摄像头、麦克风、扬声器等资源，等资源释放完毕后： • 本地会触发 onLeaveChannel 回调。 • 远端会触发 onUserOffline 回调。

setChannelProfile（设置频道场景）

setChannelProfile

```
abstract public int setChannelProfile(int profile);
```

详情：	SDK 会针对不同的使用场景采用不同的优化策略，以获取最佳的音视频传输体验。

参数:	profile 频道使用场景: • CHANNEL_PROFILE_COMMUNICATION (0)：通信。声网推荐使用直播场景以获取更好的音视频体验。 • HANNEL_PROFILE_LIVE_BROADCASTING (1)：（默认）直播。

注意:
该方法必须在 joinChannel 前调用和进行设置，进入频道后无法再设置。

setClientRole [1/2]（设置用户角色）

setClientRole

```
abstract public int setClientRole(int role);
```

详情:	SDK 会针对不同的使用场景采用不同的优化策略，以获取最佳的音视频传输体验。
参数:	role 用户的角色: • CLIENT_ROLE_BROADCASTER (1):主播。 • CLIENT_ROLE_AUDIENCE (2):观众。

注意:
当您创建的engine为TRTCCloud时，该方法只能在加入频道后使用。

setClientRole [2/2]（设置直播场景下的用户角色和级别）

setClientRole

```
public abstract int setClientRole(int role, ClientRoleOptions
options);
```

详情:	只有加入频道后才可调用该方法设置用户角色，该方法与 setClientRole [1/2] 效果一样，设置options参数无效。

参数:	role 用户的角色： • CLIENT_ROLE_BROADCASTER (1):主播。 • CLIENT_ROLE_AUDIENCE (2):观众。
	options 用户具体设置，包含用户级别，详见 ClientRoleOptions（该参数不生效）。

⚠ 注意：

- 当您创建的 engine 为 TRTCCloud 时，该参数不生效。
- 当您创建的 engine 为 TRTCCloud 时，该方法只能在加入频道后使用。

updateChannelMediaOptions（加入频道后更新频道媒体选项）

updateChannelMediaOptions

```
public abstract int updateChannelMediaOptions(ChannelMediaOptions
options);
```

参数:	Options 频道媒体选项，详见 ChannelMediaOptions 。目前仅生效： • publishCameraTrack 设置是否发布摄像头采集的视频： ○ true：发布摄像头采集的视频。 ○ false：不发布摄像头采集的视频。 • publishMicrophoneTrack 设置是否发布麦克风采集到的音频： ○ true：发布麦克风采集到的音频。 ○ false：不发布麦克风采集到的音频。 • clientRoleType 用户的角色： ○ CLIENT_ROLE_BROADCASTER (1)：主播。 ○ CLIENT_ROLE_AUDIENCE (2)：观众。其他设置不生效。

⚠ 注意：

- 当您创建的 engine 为 TRTCCloud 时，该参数不生效。
- 当您创建的 engine 为 TRTCCloud 时，该方法只能在加入频道后使用。

muteAllRemoteAudioStreams（取消或恢复订阅所有远端用户的音频流）

muteAllRemoteAudioStreams

```
abstract public int muteAllRemoteAudioStreams(boolean muted);
```

详情:	成功调用该方法后，本地用户会取消或恢复订阅所有远端用户的音频流。
参数:	**muted** 是否取消订阅所有远端用户的音频流： • true: 取消订阅所有远端用户的音频流。 • false: （默认）订阅所有远端用户的音频流。

muteLocalAudioStream（取消或恢复发布本地音频流）

muteLocalAudioStream

```
abstract public int muteLocalAudioStream(boolean muted);
```

详情:	成功调用该方法后，远端会触发 onUserMuteAudio 回调和 onRemoteAudioStateChanged 回调。
参数:	**muted** 是否取消订阅所有远端用户的音频流： • true: 取消发布本地音频流。 • false:发布本地音频流。

muteRemoteAudioStream（取消或恢复订阅指定远端用户的音频流）

muteRemoteAudioStream

```
abstract public int muteRemoteAudioStream(int uid, boolean muted);
```

详情:	成功调用该方法后，远端会触发 onUserMuteAudio 回调和 onRemoteAudioStateChanged 回调。

参数:	uid 指定用户的用户 ID。
	muted 是否取消订阅所有远端用户的音频流： • true: 取消订阅指定用户的音频流。 • false: (默认) 订阅指定用户的音频流。

注意：
该方法需要在加入频道后调用。

adjustUserPlaybackSignalVolume（调节本地播放的指定远端用户信号音量）

adjustUserPlaySignalVolume

```
abstract public int adjustUserPlaybackSignalVolume(int uid, int
volume);
```

详情:	您可以在通话中调用该方法调节指定远端用户在本地播放的音量。如需调节多个用户在本地播放的音量，则需多次调用该方法。
参数:	uid 指定用户的用户 ID。
	volume 音量范围为 [0,100]。

adjustPlaybackSignalVolume（调节本地播放的所有远端用户信号音量）

adjustPlaybackSignalVolume

```
abstract public int adjustPlaybackSignalVolume(int volume);
```

详情:	调节本地播放的所有远端用户信号音量。
参数:	volume 音量，取值范围为 [0,400]。

disableAudio（关闭音频模块）

disableAudio

```
abstract public int disableAudio();
```

详情：	调用该接口后，音频相关接口将不生效。

注意：
当您创建的实例为 RtcEngine 时，会停止本地麦克风采集、停止发布本地音频流、停止接收所有远端音频流。而当您创建的实例为 TRTCCloud 时，音频相关接口调用将不再生效。

enableAudio（启用音频模块）

enableAudio

```
abstract public int enableAudio();
```

详情：	调用该接口后，音频相关接口将生效。

注意：
当您创建的实例为 RtcEngine 时，会开启本地麦克风采集、创建并发布本地音频流、接收并播放所有远端音频流。而当您创建的实例为TRTCCloud 时，音频相关接口将生效。

enableAudioVolumeIndication（启用用户音量提示）

enableAudioVolumeIndicatio

```
abstract public int enableAudioVolumeIndication(int interval, int
smooth, boolean reportVad);
```

详情：	调用该接口后，engine 会按设置的时间间隔触发 onAudioVolumeIndication 回调报告音量信息。
参数：	interval 回调的触发间隔，单位为毫秒。 • ≤ 0: 关闭回调。 • > 0: 返回音量提示的间隔，单位为毫秒，最小间隔为 100ms，推荐值：300ms。
	smooth 平滑系数，该参数目前不生效。
	reportVad 是否开启本地人声检测功能，在 enableLocalAudio 之前调用才可以生效。

setAudioProfile（设置音频编码属性）

setAudioProfile

```
abstract public int setAudioProfile(int profile);
```

参数：	profile 音频编码属性，详见参数 profile，目前仅有以下参数生效： • profile = 1：流畅：采样率：16k；单声道；音频裸码率：16kbps；适合语音通话为主的场景，例如在线会议，语音通话。 • profile = 2：默认：采样率：48k；单声道；音频裸码率：50kbps；SDK 默认的音频质量，如无特殊需求推荐选择之。 • profile = 4、profile = 5：高音质：采样率：48k；双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线K歌、音乐直播等。 • profile等于其他值均为：默认：采样率：48k；单声道；音频裸码率：50kbps；SDK 默认的音频质量。

adjustRecordingSignalVolume（调节音频采集信号音量）

adjustRecordingSingalVolume

```
abstract public int adjustRecordingSignalVolume(int volume);
```

参数:	volume 音量，取值范围为 [0,400]。

enableInEarMonitoring（开启耳返功能）

enableInEarMonitoring

```
abstract public int enableInEarMonitoring(boolean enabled);
```

详情:	成功调用该方法后，开启耳返功能
参数:	enabled 开启/关闭耳返功能： • true: 开启耳返功能。 • false: （默认）关闭耳返功能。

enableLocalAudio（开关本地音频采集）

enableLocalAudio

```
abstract public int enableLocalAudio(boolean enabled);
```

详情:	SDK 默认不开启麦克风，当用户需要发布本地音频时，需要调用该接口开启麦克风采集，并将音频编码并发布到当前的房间中。 开启本地音频的采集和发布后，房间中的其他用户会收到 onUserMuteAudio 的通知。
参数:	enabled 开启/关闭本地音频采集： • true: 开启。 • false: 关闭。

setInEarMonitoringVolume（设置耳返音量）

setInEarMonitoringVolume

```
abstract public int setInEarMonitoringVolume(int volume);
```

详情:	该接口可设置耳返音量，取之范围[0,400]。
参数:	volume 音量大小，取值范围为[0,400]，默认值：400。

setEnableSpeakerphone（开启或关闭扬声器播放）

setEnableSpeakerphone

```
abstract public int setEnableSpeakerphone(boolean enabled);
```

详情:	手机有两个扬声器：一个是位于手机顶部的听筒，一个是位于手机底部的立体声扬声器。 • 设置音频路由为听筒时，声音比较小，只有将耳朵凑近才能听清楚，隐私性较好，适合用于接听电话。 • 设置音频路由为扬声器时，声音比较大，不用将手机贴脸也能听清，因此可以实现“免提”的功能。
参数:	enabled 设置是否开启扬声器播放： • true: 开启。音频路由为扬声器。 • false: 关闭。音频路由为听筒。

isSpeakerphoneEnabled（检查扬声器状态启用状态）

isSpeakerphoneEnabled

```
abstract public boolean isSpeakerphoneEnabled();
```

详情:	调用该接口可获得扬声器的状态（开启/关闭）
参数:	**返回值:** • true: 扬声器已开启，语音会输出到扬声器。 • false: 扬声器未开启，语音会输出到非扬声器（听筒，耳机等）。

setDefaultAudioRoutetoSpeakerphone（设置默认的音频路由）

setDefaultAudioRoutetoSpeakerphone

```
abstract public int setDefaultAudioRoutetoSpeakerphone(boolean
defaultToSpeaker);
```

详情:	调用该接口设置默认音频路由（听筒/扬声器）。
参数:	**defaultToSpeaker** 是否使用扬声器作为默认的音频路由: • true: 设置默认音频路由为扬声器。 • false: 设置默认音频路由为听筒。

注意:
该方法需要在加入频道前调用。如需在加入频道后切换音频路由，请调用 setEnableSpeakerphone。

setRouteInCommunicationMode（选择通话音量模式下的音频路由）

setRouteInCommunicationMode

```
abstract public int setRouteInCommunicationMode(int route);
```

详情:	调用该接口设置通话模式下的音频路由。

参数：	route 期望使用的音频路由，详见参数 route，当您创建的实例为TRTCCloud时，目前仅有以下参数生效： • 1：听筒。 • 其他值均为：扬声器。

注意：
当您创建的实例为 TRTCCloud 类型时，目前只设置听筒和扬声器生效。

enableVideo（启用视频模块）

enableVideo

```
abstract public int enableVideo();
```

详情：	当您调用该接口后，视频相关接口将生效。

注意：
当您创建的 engine 为 RtcEngine 类型时，会创建并发布本地视频流，接收并播放远端所有视频流。而当当您创建的实例 为TRTCCloud 类型时，调用该接口后视频相关接口将生效。

disableVideo（关闭视频模块）

disableVideo

```
abstract public int disableVideo();
```

详情：	当您调用该接口后，视频相关接口将不生效。

注意：

> 当您创建的实例为 RtcEngine 类型时，会停止发布本地视频流，停止接收远端所有视频流。而当当您创建的实例 为 TRTCCloud 类型时，调用该接口后视频相关接口将生效。

muteAllRemoteVideoStreams（取消或恢复订阅所有远端用户的视频流）

muteAllRemoteVideoStreams

```
abstract public int muteAllRemoteVideoStreams(boolean muted);
```

详情：	该接口仅暂停/恢复接收所有用户的视频流，但并不释放显示资源，视频画面会被冻屏在接口调用时的最后一帧。
参数：	muted 是否取消订阅所有远端用户的视频流： • true: 取消订阅所有用户的视频流。 • false: （默认）订阅所有用户的视频流。

muteLocalVideoStream（取消或恢复发布本地视频流）

muteLocalVideoStream

```
abstract public int muteLocalVideoStream(boolean muted);
```

详情：	成功调用该方法后，远端会触发 onUserMuteVideo 回调。
参数：	muted 是否取消发送本地视频流： • true: 取消发送本地视频流。 • false: （默认）发送本地视频流。

muteRemoteVideoStream（取消或恢复订阅指定远端用户的视频流）

muteRemoteVideoStream

```
abstract public int muteRemoteVideoStream(int userId, boolean
muted);
```

详情：	成功调用该方法后，会根据 userId 指定订阅远端用户的视频流
参数：	**userId** 指定用户的用户 ID。
	muted 是否取消订阅指定远端用户的视频流： • true: 取消订阅指定用户的视频流。 • false: （默认）订阅指定用户的视频流。

setVideoEncoderConfiguration（设置视频编码属性）

setVideoEncoderConfiguration

```
abstract public int
setVideoEncoderConfiguration(VideoEncoderConfiguration config);
```

详情：	成功调用该方法后，可设置视频编码属性，当您创建的实例为 TRTCCloud 类型时，目前仅支持 dimensions（分辨率）、frameRate（视频编码的帧率）、bitrate（视频编码码率）、orientationMode（视频编码的方向模式）几项配置。
参数：	config 指用于设置视频编码器的相关参数，详见 VideoEncoderConfiguration 。
	• 0: 调用成功。 • <0: 调用失败。

注意：
当您创建的实例 为 TRTCCloud 类型时，参数 config 仅支持 dimensions（分辨率）、frameRate（视频编码的帧率）、bitrate（视频编码码率）、orientationMode（视频编码的方向模式）几项配置。

startPreview（开启视频预览）

startPreview

```
public abstract int startPreview();
```

详情:	在 joinChannel 之前调用此函数，SDK 只会开启摄像头，并一直等到您调用 joinChannel 之后才开始推流。在 joinChannel 之后调用此函数，SDK 会开启摄像头并自动开始视频推流。调用该方法前，必须先调用 enableVideo开启视频功能、setupLocalVideo 初始化本地视图。

stopPreview（停止摄像头预览）

stopPreview

```
abstract public int stopPreview();
```

详情:	调用 startPreview 开启预览后，如果您想关闭本地视频预览，请调用该方法。

setLocalRenderMode（设置本地画面的渲染参数）

setLocalRenderMode

```
abstract public int setLocalRenderMode(int renderMode, int
mirrorMode);
```

详情:	初始化本地用户视图后，您可以调用该方法更新本地用户视图的渲染和镜像模式。该方法只影响本地用户看到的视频画面，不影响本地发布视频。
参数:	renderMode 画面填充模式: • RENDER_MODE_HIDDEN (1)：填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。 • RENDER_MODE_FIT (2)：适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。

	mirrorMode 画面镜像模式： • VIDEO_MIRROR_MODE_AUTO (0)：自动模式：如果正使用前置摄像头则开启镜像，如果是后置摄像头则不开启镜像（仅适用于移动设备）。 • VIDEO_MIRROR_MODE_ENABLED (1)：强制开启镜像，不论当前使用的是前置摄像头还是后置摄像头。 • VIDEO_MIRROR_MODE_DISABLED (2)：强制关闭镜像，不论当前使用的是前置摄像头还是后置摄像头。

setRemoteRenderMode（设置远端画面的渲染模式）

setRemoteRenderMode

```
abstract public int setRemoteRenderMode(int userId, int renderMode,
int mirrorMode);
```

详情：	初始化远端用户视图后，您可以调用该方法更新远端用户视图在本地显示时的渲染和镜像模式。该方法只影响本地用户看到的视频画面。
参数：	**userId** 远端用户 ID。
	renderMode 画面填充模式： • RENDER_MODE_HIDDEN (1)：填充模式：即将画面内容居中等比缩放以充满整个显示区域，超出显示区域的部分将会被裁剪掉，此模式下画面可能不完整。 • RENDER_MODE_FIT (2)：适应模式：即按画面长边进行缩放以适应显示区域，短边部分会被填充为黑色，此模式下图像完整但可能留有黑边。
	mirrorMode 画面镜像模式： • VIDEO_MIRROR_MODE_AUTO (0)：自动模式：如果正使用前置摄像头则开启镜像，如果是后置摄像头则不开启镜像（仅适用于移动设备）。 • VIDEO_MIRROR_MODE_ENABLED (1)：强制开启镜像，不论当前使用的是前置摄像头还是后置摄像头。 • VIDEO_MIRROR_MODE_DISABLED (2)：强制关闭镜像，不论当前使用的是前置摄像头还是后置摄像头。

setupLocalVideo（初始化本地视图）

setupLocalVideo

```
abstract public int setupLocalVideo(VideoCanvas local);
```

详情：	该方法初始化本地视图并设置本地用户视频显示属性，只影响本地用户看到的视频画面，不影响本地发布视频。调用该方法绑定本地视频流的显示视窗(view)，并设置本地用户视图的渲染模式和镜像模式。
参数：	local 本地视频显示属性。详见 VideoCanvas 。目前仅成效 local.view、local.renderMode、local.mirrorMode。

注意：
当您创建的实例为 TRTCCloud 类型时，传入参数仅生效 local.view、local.renderMode、local.mirrorMode。

setupRemoteVideo（初始化远端用户视图）

setupRemoteVideo

```
abstract public int setupRemoteVideo(VideoCanvas remote);
```

详情：	调用该方法绑定远端视频流的显示视窗(view)。
参数：	remote 本地视频显示属性。详见 VideoCanvas 。目前仅成效 local.view、local.renderMode、local.mirrorMode。

注意：
当您创建的实例为 TRTCCloud 类型时，传入参数仅生效 local.view、local.renderMode、local.mirrorMode。

switchCamera（选切换前置/后置摄像头）

switchCamera

```
abstract public int switchCamera();
```

详情：	该方法开启摄像头前后均可使用。

onError（发生错误回调）

onError

```
public void onError(int err) {}
```

详情：	该回调方法表示 SDK 运行时出现了网络或媒体相关的错误。通常情况下，SDK 上报的错误意味着 SDK 无法自动恢复，需要 App 干预或提示用户。
参数：	err 错误码，详情请参考 onError，目前仅有以下参数生效： • ERR_OK(0)：没有错误。 • ERR_JOIN_CHANNEL_REJECTED(17)。加入频道被拒绝。 • ERR_INVALID_APP_ID(101)：不是有效的 App ID。请更换有效的 App ID 重新加入频道。 • ERR_INVALID_CHANNEL_NAME(102)：不是有效的频道名。可能的原因是设置的参数数据类型不正确。请更换有效的频道名重新加入频道。 • ERR_CONNECTION_INTERRUPTED(111)：网络连接中断。SDK 在和服务器建立连接后，失去网络连接超过 4 秒。 • ERR_ADM_STOP_PLAYOUT(1010)：停止播放设备出错。

说明：
当您创建的实例为TRTCCloud时，除以上有效错误码外，其他错误码均为ERR_FAILED(1)：一般性的错误（没有明确归类的错误原因）

onClientRoleChanged（直播场景下用户角色已切换回调）

onClientRoleChanged

```
public void onClientRoleChanged(int oldRole, int newRole,
ClientRoleOptions newRoleOptions) {}
```

详情：	该回调是由本地用户在加入频道后调用 setClientRole 改变用户角色触发的。
参数：	**oldRole** 切换前的角色： • CLIENT_ROLE_BROADCASTER (1)：主播。 • CLIENT_ROLE_AUDIENCE (2)：观众。
	newRole 切换后的角色： • CLIENT_ROLE_BROADCASTER (1)：主播。 • CLIENT_ROLE_AUDIENCE (2)：观众。
	newRoleOptions 切换后的角色：详见 ClientRoleOptions（该参数无效）。

注意：
当您创建的实例为 TRTCCloud 类型时，newRoleOptions参数不生效。

onClientRoleChangeFailed（直播场景下切换用户角色失败回调）

onClientRoleChangeFailed

```
public void onClientRoleChangeFailed(int reason, int currentRole) {}
```

详情：	直播场景下，本地用户加入频道后调用 setClientRole [2/2] 切换用户角色失败时，SDK 会触发该回调，报告切换失败的原因和当前的用户角色。
参数：	**reason** 切换用户角色失败的原因，详见 reason ，由于错误码不统一，reason 统一为：CLIENT_ROLE_CHANGE_FAILED_REQUEST_TIME_OUT(3)请求超时。建议提示用户检查网络连接状况后重新尝试切换用户角色。
	currentRole 当前用户角色： • CLIENT_ROLE_BROADCASTER(1): 主播。主播可以发流也可以收流。 • CLIENT_ROLE_AUDIENCE(2): 观众。观众只能收流不能发流。

> **注意：**
> 由于错误码不统一，当您创建的实例为 TRTCCloud 类型时，reason 统一为：CLIENT_ROLE_CHANGE_FAILED_REQUEST_TIME_OUT(3)。

onJoinChannelSuccess（成功加入频道回调）

onJoinChannelSuccess

```
public void onJoinChannelSuccess(String channel, int uid, int
elapsed) {}
```

详情：	该回调方法表示该客户端成功加入了指定的频道。
参数：	**channel** 频道名。
	uid 加入频道的用户 ID。
	elapsed 从本地调用 joinChannel [2/2] 开始到发生此事件过去的时间（毫秒）。

> **注意：**
> 由于错误码不统一，当您创建的实例为 TRTCCloud 类型时，reason 统一为：CLIENT_ROLE_CHANGE_FAILED_REQUEST_TIME_OUT(3)。

onLeaveChannel（离开频道回调）

onLeaveChannel

```
public void onLeaveChannel(RtcStats stats) {}
```

详情：	当您调用 leaveChannel [2/2] 方法成功时，会触发该回调。
参	**stats** 通话的统计数据: 该参数无效。

> **注意：**
> 当您创建的实例为 TRTCCloud 类型时，**stats** 无效，可以通过 onRtcStats 回调获取。

onRejoinChannelSuccess（成功重新加入频道回调）

onRejoinChannelSuccess

```
public void onRejoinChannelSuccess(String channel, int uid, int
elapsed) {}
```

详情：	有时候由于网络原因，客户端可能会和服务器失去连接，SDK 会进行自动重连，自动重连成功后本地触发此回调方法。
参数：	**channel** 频道名。
	uid 重新加入频道的用户 ID。
	elapsed 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。

onUserJoined（远端用户（通信场景）/主播（直播场景）加入当前频道回调）

onUserJoined

```
public void onUserJoined(int uid, int elapsed) {}
```

详情：	该回调用于提示有远端用户加入频道,如果加入之前,已有其他用户在频道中,新加入的用户也会收到这些已有用户加入频道的回调。
参数：	**uid** 新加入频道的远端用户/主播 ID。
	elapsed 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。

onUserOffline（远端用户（通信场景）/主播（直播场景）离开当前频道回调）

onUserOffline

```
public void onUserOffline(int uid, int reason) {}
```

详情：	该回调用于提示有远端用户离开频道。
参数：	uid 离线用户或主播的用户 ID。
	reason 远端用户（通信场景）或主播（直播场景）下线的原因： • USER_OFFLINE_QUIT(0)：用户主动离开。此时离开频道的用户会发送一个类似“再见”的消息。收到该消息是，SDK 判定该用户离开频道。 • USER_OFFLINE_DROPPED(1)：因过长时间收不到对方数据包，SDK 判定该远端用户超时掉线。注意：在网络连接不稳定时，该判定 可能会有误。建议使用实时消息 SDK 来做可靠的掉线检测。 • USER_OFFLINE_BECOME_AUDIENCE(2)：用户的角色从主播切换为观众。

onRtcStats（当前通话统计信息回调）

onRtcStats

```
public void onRtcStats(RtcStats stats) {}
```

详情：	SDK 定期向 App 报告当前通话的统计信息，每两秒触发一次。
参数：	stats 通话的统计数据，详见 RtcStats 。目前仅有以下参数生效： • totalDuration： 本地用户通话时长（秒），累计值。 • cpuAppUsage： 当前应用的 CPU 使用率，单位 (%)，Android 8.0 以上不支持。 • cpuTotalUsage： 当前系统的 CPU 使用率，单位 (%)，Android 8.0 以上不支持。 • lastmileDelay： 从 SDK 到云端的往返延时，单位 ms该数值代表从 SDK 发送一个网络包到云端，再从云端回送一个网络包到 SDK 的总计耗时，也就是一个网络包经历“SDK=>云端=>SDK” 的总耗时。 • rxAudioKBitRate： 音频接收码率 (Kbps)。 • txAudioKBitRate： 音频包的发送码率 (Kbps)。 • rxVideoKBitRate： 视频接收码率 (Kbps)。

- txVideoKBitRate： 视频发送码率 (Kbps)。
- txKBitRate： 发送码率 (Kbps)。
- rxKBitRate： 接收码率 (Kbps)。
- rxBytes： 接收字节数 (bytes)。
- txBytes： 发送字节数 (bytes)。
- users： 当前频道内的用户人数。

onRemoteAudioStats（通话中远端音频流的统计信息回调）

onRemoteAudioStats

```
public void onRemoteAudioStats(RemoteAudioStats stats) {}
```

详情：	该回调针对每个发送音频流的远端用户每 2 秒触发一次，用于通知 SDK 内部音频关的专业技术指标。
参数：	**stats** 接收到的远端音频统计数据，详见 RemoteAudioStats，目前生效的指标为： • uid： 远端用户的用户 ID。 • jitterBufferDelay：音频接收端到网络抖动缓冲的网络延迟（毫秒）。 • audioLossRate：音频流的总丢包率（%）。 • receivedSampleRate：音频的采样率，单位 Hz。 • receivedBitrate：音频的码率，单位 Kbps。 • totalFrozenTime：音频播放的累计卡顿时长，单位 ms。 • frozenRate：音频播放卡顿率，单位 (%)。

onLocalAudioStats（通话中本地音频流的统计信息回调）

onLocalAudioStats

```
public void onLocalAudioStats(LocalAudioStats stats) {}
```

详情：	SDK 每 2 秒触发该回调一次，用于通知 SDK 内部音频关的专业技术指标。

参数：	stats 本地音频统计数据，详见 LocalAudioStats ，目前生效的指标为： • sentBitrate：发送本地音频的码率。 • sentSampleRate：发送本地音频的采样率，单位为 Hz。

onRemoteVideoStats（通话中远端视频流的统计信息回调）

onRemoteVideoStats

```
public void onRemoteVideoStats(RemoteVideoStats stats) {}
```

详情：	SDK 每 2 秒触发该回调一次，用于通知 SDK 内部视频相关的专业技术指标。
参数：	stats 远端视频统计数据, 详见 RemoteVideoStats 目前生效的参数为： • uid：远端用户的用户 ID。 • width ： 远端视频的宽度，单位 px。 • height：远端视频的高度，单位 px。 • e2eDelay ： 端到端视频延时（毫秒） • receivedBitrate：接收到的码率(Kbps)。 • frameLossRate ： 远端视频丢包率(%)。 • rxStreamType ： 视频流类型，大流或小流。 • totalFrozenTime：视频播放的累计卡顿时长，单位 ms。 • frozenRate：远端用户在加入频道后发生视频卡顿的累计时长占视频总有效时长的百分比(%)。

onLocalVideoStats（本地视频流统计信息回调）

onLocalVideoStats

```
public void onLocalVideoStats(Constants.VideoSourceType source,
LocalVideoStats stats) {}
```

详情：	SDK 每 2 秒触发该回调一次，用于通知 SDK 内部视频相关的专业技术指标。

参数：	source 视频源的类型：目前仅支持 VIDEO_SOURCE_CAMERA_PRIMARY（0）（默认）视频源为第一个摄像头。
	stats 本地视频流统计信息：详见 LocalVideoStats，目前生效的参数为： • uid：本地用户的 ID。 • encodedFrameWidth：视频编码宽度（px）。 • encodedFrameHeight：视频编码高度（px）。 • sentBitrate：实际发送码率 (Kbps)。 • sentFrameRate：实际发送帧率 (fps)。

onAudioVolumeIndication（用户音量提示回调）

onAudioVolumeIndication

```
public void onAudioVolumeIndication(AudioVolumeInfo[] speakers, int
totalVolume) {}
```

详情：	该回调默认禁用，您可以通过 enableAudioVolumeIndication 开启。开启后，只要频道内有发流用户，SDK 会在加入频道后按 enableAudioVolumeIndication 中设置的时间间隔触发 onAudioVolumeIndication 回调。每次会触发两个 onAudioVolumeIndication 回调，一个报告本地发流用户的音量相关信息，另一个报告瞬时音量最高的远端用户的音量相关信息。
参数：	speakers 用户音量信息，详见 AudioVolumeInfo 数组。无论当前房间中是否有人说话，SDK 都会按照您设定的时间间隔定时抛出此事件回调。
	totalVolume 混音后的总音量，取值范围为 [0,255]。

注意：
当您创建的实例为 TRTCCloud 类型时，无论当前房间中是否有人说话，SDK 都会按照您设定的时间间隔定时抛出此事件回调，数组 speakers 不会为空。而当您创建的实例为 RtcEngine 类型时，如果 speakers 为空，则表示远端用户不发流或没有远端用户。

onFirstLocalAudioFramePublished（已发布本地音频首帧回调）

onFirstLocalAudioFramePublished

```
public void onFirstLocalAudioFramePublished(int elapsed) {}
```

详情：	SDK 会在以下时机触发该回调： 开启本地音频的情况下，调用 joinChannel [2/2] 成功加入频道后。调用 muteLocalAudioStream(true)，再调用 muteLocalAudioStream(false) 后。
参数：	**elapsed** 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。

onRemoteAudioStateChanged（远端音频流状态发生改变回调）

onRemoteAudioStateChanged

```
public void onRemoteAudioStateChanged(int uid, int state, int
reason, int elapsed) {}
```

详情：	远端用户的音频状态发生改变时，SDK 会触发该回调向本地用户报告当前的远端音频流状态。
参数：	**uid** 发生音频状态改变的远端用户 ID。
	state 远端音频流状态，详见 参数 state，目前仅有以下参数生效： • REMOTE_AUDIO_STATE_STOPPED (0): 远端音频默认初始状态。 • REMOTE_AUDIO_STATE_STARTING (1): 本地用户已接收远端音频首包。 • REMOTE_AUDIO_STATE_DECODING (2): 远端音频流正在解码。
	reason 远端音频流状态改变的具体原因： • REMOTE_AUDIO_REASON_INTERNAL (0): 音频状态发生改变时，会报告该原因。 • REMOTE_AUDIO_REASON_NETWORK_CONGESTION (1): 网络阻塞。 • REMOTE_AUDIO_REASON_NETWORK_RECOVERY (2): 网络恢复正常。 • REMOTE_AUDIO_REASON_LOCAL_MUTED (3): 本地用户停止接收远端音频流或本地用户禁用音频模块。 • REMOTE_AUDIO_REASON_LOCAL_UNMUTED (4): 本地用户恢复接收远端音频流或本地用户启动音频模块。 • REMOTE_AUDIO_REASON_REMOTE_MUTED (5): 远端用户停止发送音频流或远端用户禁用音频模块。

	• REMOTE_AUDIO_REASON_REMOTE_UNMUTED (6): 远端用户恢复发送音频流或远端用户启用音频模块。 • REMOTE_AUDIO_REASON_REMOTE_OFFLINE (7): 远端用户离开频道。（该参数无效）
	elapsed 从本地用户调用 joinChannel [2/2] 方法到发生本事件经历的时间，单位为毫秒。

注意：
当您创建的 engine 为 TRTCCloud 时，参数 REMOTE_AUDIO_REASON_REMOTE_OFFLINE (7) 不生效。

onUserMuteAudio（远端用户（通信场景）/主播（直播场景）停止或恢复发送音频流回调）

onUserMuteAudio

```
public void onUserMuteAudio(int uid, boolean muted) {}
```

详情：	远端用户的音频状态发生改变时，SDK 会触发该回调向本地用户报告当前的远端音频流状态。
参数：	**uid** 用户 ID。
	muted 该用户是否静音： • true: 该用户已将音频静音。 • false: 该用户取消了音频静音。

onLocalVideoStateChanged（本地视频状态发生改变回调）

onLocalVideoStateChanged

```
public void onLocalVideoStateChanged(Constants.VideoSourceType
source, int state, int reason) {}
```

详情：	目前只有摄像头准备就绪时会触发该回调。
参数：	**sourceType** 视频源类型，详见 VideoSourceType 。当您创建的实例为 TRTCCloud 类型时，目前仅生效 VIDEO_SOURCE_CAMERA_PRIMARY（0）类型的流。
	state 本地视频状态，详见 state。当您创建的实例为 TRTCCloud 类型时，目前仅生效 LOCAL_VIDEO_STREAM_STATE_CAPTURING (1): 本地视频采集设备启动成功。
	reason 本地视频状态改变的具体原因，详见参数 reason 。当您创建的实例为 TRTCCloud 类型时，目前仅生效 LOCAL_VIDEO_STREAM_REASON_OK (0): 本地视频状态正常。

注意：

当您创建的实例 为 TRTCCloud 类型时：

- sourceType 仅生效 VIDEO_SOURCE_CAMERA_PRIMARY（0）类型的流。
- state 仅生效 LOCAL_VIDEO_STREAM_STATE_CAPTURING (1): 本地视频采集设备启动成功。
- reason 仅生效 LOCAL_VIDEO_STREAM_REASON_OK (0): 本地视频状态正常。

onRemoteVideoStateChanged（远端视频状态发生改变回调）

onRemoteVideoStateChanged

```
public void onRemoteVideoStateChanged(int uid, int state, int
reason, int elapsed) {}
```

详情：	远端视频状态发生改变回调。
参数：	**uid** 发生视频状态改变的远端用户 ID。
	state 远端视频流状态，详见 state，目前仅有以下参数生效： • 该回调的参数 state 为 REMOTE_VIDEO_STATE_STARTING (1)：本地用户已接收远端视频首包。 • 该回调的参数 state 为 REMOTE_VIDEO_STATE_STOPPED (0)：远端视频默认初始状态。

	reason 本地视频状态改变的具体原因，详见参数 reason 。当您创建的实例为 TRTCCloud 类型时，目前仅生效 REMOTE_VIDEO_STATE_REASON_INTERNAL (0)
	elapsed 从本地用户调用 joinChannel [2/2] 方法到发生本事件经历的时间，单位为毫秒。

onVideoSizeChanged（本地或远端视频大小和旋转信息发生改变回调）

onVideoSizeChanged

```
public void onVideoSizeChanged(Constants.VideoSourceType source, int
uid, int width, int height, int rotation) {}
```

详情:	本地或远端视频大小和旋转信息发生改变回调。
参数:	**source** 视频源的类型，详见 VideoSourceType ，当您创建的实例为 TRTCCloud 类型时，目前仅有以下参数生效： • 当画面为辅路（Sub）一般用于承载屏幕分享画面，参数source为：VIDEO_SOURCE_CAMERA_SECONDARY（1）。 • 主路（Main）一般用于承载摄像头画面，参数source为：VIDEO_SOURCE_CAMERA_PRIMARY（0）。
	uid 图像尺寸和旋转信息发生变化的用户 ID
	width 视频流的宽度（像素）。
	height 视频流的高度（像素）。
	rotation 旋转信息，当您创建的实例为 TRTCCloud 类型时，该参数不生效。

onAudioRouteChanged（音频路由已发生变化回调）

onAudioRouteChanged

```
public void onAudioRouteChanged(int routing) {}
```

详情：	当音频路由发生变化时，触发该回调。
参数：	routing 当前的音频路由，详见参数 routing ,当您创建的实例为 TRTCCloud 类型时，目前仅生效以下参数： • AUDIO_ROUTE_SPEAKERPHONE (3)：音频路由为设备自带的扬声器。 • AUDIO_ROUTE_EARPIECE (1)：音频路由为听筒。

onFirstLocalVideoFramePublished（已发布本地视频首帧回调）

onFirstLocalVideoFramePublished

```
public void
onFirstLocalVideoFramePublished(Constants.VideoSourceType source,
int elapsed) {}
```

详情：	开启本地视频的情况下，调用 joinChannel [2/2] 成功加入频道后。调用 muteLocalVideoStream(true)，再调用 muteLocalVideoStream(false) 后。
参数：	source 视频源的类型，详见 VideoSourceType ，当您创建的实例为 TRTCCloud 类型时，目前仅有以下参数生效： • 当画面为辅路（Sub）一般用于承载屏幕分享画面，参数source为：VIDEO_SOURCE_CAMERA_SECONDARY（1）。 • 主路（Main）一般用于承载摄像头画面，参数source为：VIDEO_SOURCE_CAMERA_PRIMARY（0）。
	elapsed 从调用 joinChannel [2/2] 方法到触发该回调的时间间隔（毫秒）。

onFirstLocalVideoFrame（已显示本地视频首帧回调）

onFirstLocalVideoFrame

```
public void onFirstLocalVideoFrame(Constants.VideoSourceType source,
int width, int height, int elapsed) {}
```

详情：	本地视频首帧显示在本地视图上时，SDK 会触发此回调。
参数：	source 视频源的类型，详见 VideoSourceType，当您创建的实例为 TRTCCloud 类型时，目前仅有以下参数生效： • VIDEO_SOURCE_CAMERA_PRIMARY 0：（默认）视频源为第一个摄像头。 • VIDEO_SOURCE_SCREEN_PRIMARY 2：视频源为第一个屏幕。
	width 本地渲染视频的宽 (px) 。
	height 本地渲染视频的高 (px)。
	elapsed 从调用 joinChannel [2/2] 到发生此事件过去的时间（毫秒）。

onFirstRemoteVideoFrame（渲染器已接收首帧远端视频回调）

onFirstRemoteVideoFrame

```
public void onFirstRemoteVideoFrame(int uid, int width, int height,
int elapsed) {}
```

详情：	渲染器已接收首帧远端视频回调。
参数：	uid 远端用户 ID。
	width 视频流宽（px）。
	height 视频流高（px）。
	elapsed 从调用 joinChannel [2/2] 到发生此事件过去的时间（毫秒）。

registerVideoFrameObserver（注册原始视频观测器对象）

registerVideoFrameObserver

```
public abstract int registerVideoFrameObserver(Object observer);
```

详情:	设置该回调之后，SDK 会把采集到的视频帧通过您设置的 listener 回调出来，用于第三方美颜组件进行二次处理，之后 SDK 会将处理后的视频帧进行编码和发送。
参数:	**observer** 自定义预处理回调，当您使用 AllInOneRTCEngine 创建的实例为 TRTCCloud 类型时，该参数需要传入 TRTCVideoFrameListener 类型的对象，而当您创建的实例为 RtcEngine 时，该参数需要传入 IVideoFrameObserver 类型的对象。

第三方美颜
Android&iOS

最近更新时间：2024-12-03 16:33:13

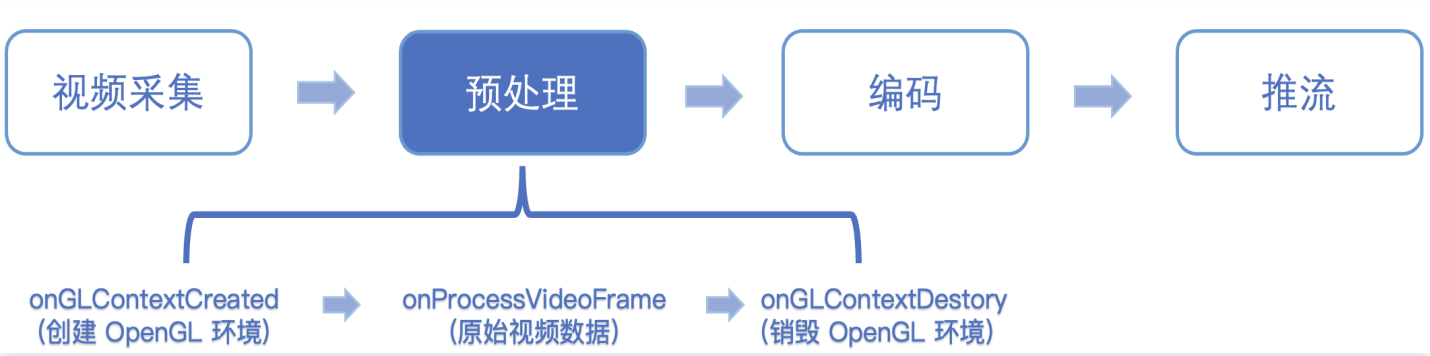
背景说明

本章节将介绍使用 AllInOneRTCEngine 服务时怎样接入第三方美颜。在此之前，请确保您已完成：

- **已成功接入 AllInOneRTCEngine。**
- **已成功导入第三方美颜 SDK。**

接入第三方美颜的方法

关于 TRTC 的视频处理流程如下图所示，您只需在预处理阶段，通过第三方美颜 SDK 处理原始视频数据即可完成美颜接入。



原始视频数据的获取方法：

- **Android**：通过调用 registerVideoFrameObserver（注册原始视频观测器对象），并注册 TRTCVideoFrameListener 类中的 onProcessVideoFrame 回调获得。

registerVideoFrameObserver

注册原始视频观测器对象

```
public abstract int registerVideoFrameObserver(Object observer);
```

详情：	设置该回调之后，SDK 会把采集到的视频帧通过您设置的 listener 回调出来，用于第三方美颜组件进行二次处理，之后 SDK 会将处理后的视频帧进行编码和发送。

参数：	observer 自定义预处理回调，当您使用 AllInOneRTCEngine 创建的实例为 TRTCCloud 类型时，该参数需要创建并传入 TRTCVideoFrameListener 类型的对象，而当您创建的实例为 RtcEngine 时，该参数需要创建并传入 IVideoFrameObserver 类型的对象。

- **iOS：**通过调用 setVideoFrameDelegate（注册原始视频观测器对象），并注册类 TRTCVideoFrameDelegate 类中的 onProcessVideoFrame 回调获得。

setVideoFrameDelegate

注册原始视频观测器对象

```
- (BOOL)setVideoFrameDelegate:(id _Nullable)delegate;
```

详情：	设置该回调之后，SDK 会把采集到的视频帧通过您设置的 listener 回调出来，用于第三方美颜组件进行二次处理，之后 SDK 会将处理后的视频帧进行编码和发送。
参数：	delegate 接口对象实例，当您使用 AllInOneRTCEngine 创建的实例为 TRTCCloud 类型时，需要注册 TRTCVideoFrameDelegate 类，您可以根据需要注册 TRTCVideoFrameDelegate 类中的回调。在成功注册视频观测器后，SDK 会在捕捉到每个视频帧时，触发您所注册的上述回调。而当您创建的实例为 RtcEngine 时，则需要注册 AgoraVideoFrameDelegate 类的回调。

接入第三方美颜示例

相芯美颜（Android）

当您使用 AllInOneRTCEngine 服务创建的实例为 TRTCCloud 类型时：

- 导入包名 import com.tencent.trtc.TRTCCloudListener，创建一个 TRTCVideoFrameListener 类型的对象，**并实现在 onProcessVideoFrame 回调中使用第三方美颜 SDK 处理原始视频数据（重要）。**

```
// 导入
import com.tencent.trtc.TRTCCloudListener;

// 创建一个 TRTCVideoFrameListener 类型的对象
TRTCCloudListener.TRTCVideoFrameListener mListener = new
TRTCCloudListener.TRTCVideoFrameListener() {
```

```
    @Override
    public void onGLContextCreated() {
        // SDK 内部 OpenGL 环境已经创建的通知
        mFURenderer.onSurfaceCreated();
    }

    @Override
    public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame
srcFrame,
                                   TRTCCloudDef.TRTCVideoFrame
dstFrame) {
        // mFURenderer 为相芯美颜SDK的实例
        dstFrame.texture.textureId = mFURenderer
        .onDrawFrameSingleInput(srcFrame.texture.textureId,
srcFrame.width, srcFrame.height);
        return 0;
    }

    @Override
    public void onGLContextDestory() {
        // SDK 内部 OpenGL 环境被销毁的通知
        mFURenderer.onSurfaceDestroyed();
    }
}
```

- 在进房前调用 registerVideoFrameObserver 注册原始视频观测器对象，成功接入第三方美颜。

```
private AIORTCEngine  mEngine;

// 创建 AIORTCEngine 实例，并切换 SDK 为 TRTCCloud。
mEngine = RTCEngineManager.create(this, appId, mTRTCListener, true);

// 调用 registerVideoFrameObserver 注册原始视频观测器对象，并传入上一步已创建
的 TRTCVideoFrameListener 类型对象
mEngine.registerVideoFrameObserver(mListener);
```

相芯美颜（iOS）

当您使用 AllInOneRTCEngine 服务创建的实例为 TRTCCloud 类型时：

- 在进房前调用 setVideoFrameDelegate：

```
#import <TXLiteAVSDK_TRTC/TRTCCloudDelegate.h>
```

```
@interface TRTCMainViewController : TRTCVideoFrameDelegate
@property (nonatomic, strong) id<AIORTCEngine> engine;
......
@end

- (void)enableThirdBeauty {
    // 初始化相芯美颜SDK
    [self initThirdBeauty];
    // 调用 setVideoFrameDelegate 成功接入第三方美颜
    [_engine setVideoFrameDelegate:self];
}
```

- **注册 TRTCVideoFrameDelegate 类的 onProcessVideoFrame 回调，在该回调中，使用第三方美颜SDK处理原始视频数据（重要），成功接入第三方美颜。**

```
@implementation TRTCMainViewController
- (uint32_t)onProcessVideoFrame:(TRTCVideoFrame *)srcFrame dstFrame:
(TRTCVideoFrame *)dstFrame{
    // FUManager 为相芯美颜SDK实例
    dstFrame.textureId = [[FUManager shareManager]
renderItemWithTexture:srcFrame.textureId Width:srcFrame.width
Height:srcFrame.height];
    return 0;
}
```

实时合唱
组件介绍（TUIKaraoke）

最近更新时间：2022-10-31 16:21:59

组件背景

艾媒咨询数据显示，2021年中国在线 K 歌用户规模约为5.1亿人，渗透率约为49.7%。在线 K 歌具有更强的沉浸感体验，多元的玩法也迎合了不同用户群体的个性化需求，目前已经成为了在线泛娱乐领域的主要项目之一。在网络技术革新的基础上，在线 K 歌 App 不断推出多元化的歌唱模式和玩法，不断丰富的功能提升了在线 K 歌 App 的实用性和可玩性。下来将介绍腾讯实时音视频（Tencent RTC）技术在线 K 歌场景下的组件实现方案。

TUIKaraoke组件介绍

TUIKaraoke 是一个开源的音视频 UI 组件，通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加在线 K 歌场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。TUIKaraoke 同时支持 Android、iOS 平台的源代码，基本功能如下图所示：

说明：

TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信 IM 服务。即时通信 IM 服务详细计费规则请参见 即时通信 – 价格说明 ，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

快速集成（TUIKaraoke）Android

最近更新时间：2025-01-08 11:48:02

步骤一：下载并导入TUIKaraoke 组件

单击进入 Github ，选择克隆/下载代码，然后拷贝 Android 目录下的 Source 和 Debug 目录到您的工程中，并完成如下导入动作：

1. 在 `setting.gradle` 中完成导入，参考如下：

```
include ':tuikaraoke'
include ':debug'
```

2. 在 app 的 `build.gradle` 文件中添加对 TUIKaraoke 的依赖：

```
api project(':tuikaraoke')
```

3. 在根目录的 `build.gradle` 文件中添加 TRTC SDK 和 IM SDK 的依赖：

```
ext {
  liteavSdk = "com.tencent.liteav:LiteAVSDK_TRTC:latest.release"
  imSdk = "com.tencent.imsdk:imsdk-plus:latest.release"
}
```

步骤二：配置权限及混淆规则

在 AndroidManifest.xml 中配置 App 的权限，SDK 需要以下权限（6.0以上的 Android 系统需要动态申请麦克风、读取存储权限等）：

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"
/>         // 使用场景：悬浮窗功能需要此权限；
<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.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"
/>
```

```
<uses-permission android:name="android.permission.BLUETOOTH" />
// 使用场景：使用蓝牙耳机时需要此权限；
```

在 proguard-rules.pro 文件，将 SDK 相关类加入不混淆名单：

```
-keep class com.tencent.** { *; }
```

步骤三：初始化并登录

相关接口详情请参见 TUIKaraoke 。

```
// 1.初始化
TRTCKaraokeRoom mTRTCKaraokeRoom = TRTCKaraokeRoom.sharedInstance(this);
mTRTCKaraokeRoom.setDelegate(this);
// 2.登录
mTRTCKaraokeRoom.login(SDKAppID, UserID, UserSig, new
TRTCKaraokeRoomCallback.ActionCallback() {
    @Override
    public void onCallback(int code, String msg) {
        if (code == 0) {
        //登录成功
        }
    }
});
```

参数说明：

- **SDKAppID**：TRTC 应用ID，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**。SDKAppID 信息如下图所示：

- **Secretkey**：TRTC 应用密钥，和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，长度不超过32字节，不支持使用特殊字符，建议使用英文或数字，可结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey 等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的 UserSig，也可以参照我们的 TUIKaraoke 示例工程 自行计算，更多信息见 如何计算及使用 UserSig。

步骤四：实现在线KTV场景

1. 主播创建房间 TUIKaraoke.createRoom

```
int roomId = "房间ID";
TRTCKaraokeRoomDef.RoomParam roomParam = new
TRTCKaraokeRoomDef.RoomParam();
roomParam.roomName = "房间名称";
roomParam.needRequest = false; // 上麦是否需要房主确认
roomParam.seatCount = 8;       // 房间座位数，一共8个座位
roomParam.coverUrl = "房间封面图URL";
```

```
mTRTCKaraokeRoom.createRoom(roomId, roomParam, new
TRTCKaraokeRoomCallback.ActionCallback() {
 @Override
 public void onCallback(int code, String msg) {
   if (code == 0) {
     //创建成功
   }
 }
});
```

2. 听众进入房间 TUIKaraoke.enterRoom

```
mTRTCKaraokeRoom.enterRoom(roomId, new
TRTCKaraokeRoomCallback.ActionCallback() {
 @Override
 public void onCallback(int code, String msg) {
     if (code == 0) {
     //进房成功
     }
 }
});
```

3. 听众主动上麦 TUIKaraoke.enterSeat

```
// 1.听众调用上麦
int seatIndex = 1;
mTRTCKaraokeRoom.enterSeat(seatIndex, new
TRTCKaraokeRoomCallback.ActionCallback() {
 @Override
 public void onCallback(int code, String msg) {
     if (code == 0) {
     //上麦成功
     }
 }
});
// 2.收到 onSeatListChange 回调，刷新麦位列表
@Override
public void onSeatListChange(final List<TRTCKaraokeRoomDef.SeatInfo>
seatInfoList) {
}
```

> **说明：**
> 其他关于麦位管理的相关操作，您可参考 TUIKaraoke 接口文档 按需实现，或者可以参考我们的 TUIKaraoke示例工程。

4. 实现音乐播放并体验 KTV 场景

您可以根据自己的业务获取音乐 ID 和 URL链接播放歌曲，接口详情请参见 TUIKaraoke 音乐播放接口 。

```
//播放音乐
mTRTCKaraokeRoom.startPlayMusic(musicID,url);
//停止音乐
mTRTCKaraokeRoom.stopPlayMusic();
```

完成以上步骤，您可以实现 KTV 基本功能。如果您的业务还需要文字聊天、发送礼物等功能，可以接入以下能力。

步骤五：文字聊天功能（可选）

如果您需要实现各主播或听众之间文字聊天的功能，可以通过以下方法发送或接收聊天信息。
相关接口详情请参见 TRTCKaraokeRoom.sendRoomTextMsg 。

```
// 发送端：发送文本消息
mTRTCKaraokeRoom.sendRoomTextMsg("Hello Word!", new
TRTCKaraokeRoomCallback.ActionCallback() {
    @Override
    public void onCallback(int code, String msg) {
        if (code == 0) {
            //发送成功
        }
    }
});
// 接收端：监听文本消息
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {
  @Override
  public void onRecvRoomTextMsg(String message,
TRTCKaraokeRoomDef.UserInfo userInfo) {
      Log.d(TAG, "收到来自" + userInfo.userName + "的消息:" + message);
  }
});
```

步骤六：礼物发送功能（可选）

如果您需要实现礼物发送和接收功能，可以通过以下方法发送或接收礼物并展示。

```
// 发送端：通过自定义 "CMD_GIFT" 来区分礼物消息
mTRTCKaraokeRoom.sendRoomCustomMsg("CMD_GIFT",date, new
TRTCKaraokeRoomCallback.ActionCallback() {
    @Override
    public void onCallback(int code, String msg) {
        if (code == 0) {
            //发送成功
        }
    }
});

// 接收端：监听礼物消息
mTRTCKaraokeRoom.setDelegate(new TRTCKaraokeRoomDelegate() {
    @Override
    public void onRecvRoomCustomMsg(String cmd, String message,
TRTCKaraokeRoomDef.UserInfo userInfo) {
        if ("CMD_GIFT".equals(cmd)) {
            // 收到礼物消息
            Log.d(TAG, "收到来自" + userInfo.userName + "的礼物：" +
message);
        }
    }
});
```

iOS

最近更新时间：2025-01-08 11:48:02

步骤一：下载并导入 TUIKaraoke 组件

单击进入 Github ，选择克隆/下载代码，然后拷贝 iOS目录下的 `Source` 、 `Resources` 、 `TXAppBasic` 文件夹和 `TUIKaraoke.podspec` 文件到您的工程中，并完成如下导入动作：

1. 在您的 `Podfile` 文件内添加导入命令，参考如下：

```
pod 'TUIKaraoke', :path => "./", :subspecs => ["TRTC"]
pod 'TXLiteAVSDK_TRTC'
pod 'TXAppBasic', :path => "TXAppBasic/"
```

2. 打开终端，进入 Podfile 文件所在目录下执行安装命令，参考如下：

```
pod install
```

步骤二：配置权限

在您的工程的 info.plist文件中配置 App 的权限，SDK 需要以下权限（iOS 系统需要动态申请麦克风）：

```
<key>NSMicrophoneUsageDescription</key>
<string>Karaoke 需要访问您的麦克风权限</string>
```

步骤三：初始化并登录

相关接口说明请参见 TUIKaraoke 。

```
  // 1.初始化
  let karaokeRoom = TRTCKaraokeRoom.shared()
  karaokeRoom.setDelegate(delegate: self)
  // 2.登录
  karaokeRoom.login(SDKAppID: Int32(SDKAppID), UserId: UserId, UserSig:
ProfileManager.shared.curUserSig()) { code, message in
        if code == 0 {
            //登录成功
        }
  }
```

参数说明：

- **SDKAppID**：TRTC 应用ID，如果您未开通腾讯云 TRTC 服务，可进入 腾讯云实时音视频控制台，创建一个新的 TRTC 应用后，单击**应用信息**。SDKAppID 信息如下图所示：

- **Secretkey**：TRTC 应用密钥，和 SDKAppId 对应，进入 TRTC 应用管理 后，SecretKey 信息如上图所示。
- **userId**：当前用户的 ID，字符串类型，长度不超过32字节，不支持使用特殊字符，建议使用英文或数字，可结合业务实际账号体系自行设置。
- **userSig**：根据 SDKAppId、userId，Secretkey 等信息计算得到的安全保护签名，您可以单击 这里 直接在线生成一个调试的 UserSig，也可以参照我们的 TUICalling示例工程自行计算，更多信息见 如何计算及使用 UserSig。

步骤四：实现在线KTV场景

1. 主播创建房间 TUIKaraoke.createRoom

```
int roomId = "房间ID";
let param = RoomParam.init()
param.roomName = "房间名称";
param.needRequest = false; // 上麦是否需要房主确认
param.seatCount = 8;       // 房间座位数，一共8个座位
```

```
param.coverUrl = "房间封面图URL";

karaokeRoom.createRoom(roomID: Int32(roomInfo.roomID), roomParam:
param) { [weak self] (code, message) in
 guard let `self` = self else { return }
 if code == 0 {
     //创建成功
 }
}
```

2. 听众进入房间 TUIKaraoke.enterRoom

```
karaokeRoom.enterRoom(roomID: roomInfo.roomID) { [weak self] (code,
message) in
 guard let `self` = self else { return }
 if code == 0 {
     //进房成功
 }
}
```

3. 听众主动上麦 TUIKaraoke.enterSeat

```
// 1.听众调用上麦
int seatIndex = 1;
karaokeRoom.enterSeat(seatIndex: seatIndex) { [weak self] (code,
message) in
 guard let `self` = self else { return }
 if code == 0 {
     //上麦成功
 }
}
// 2.收到 onSeatListChange 回调，刷新麦位列表
func onSeatListChange(seatInfoList: [SeatInfo]) {
}
```

> **说明：**
> 其他关于麦位管理的相关操作，您可参考 TUIKaraoke 接口文档 按需实现，或者可以参考我们的 TUIKaraoke示例工程。

4. 实现音乐播放并体验 KTV 场景
5. 您可以根据自己的业务获取音乐 ID 和 URL 链接，播放歌曲。详情请参见 TUIKaraoke 音乐播放接口 。

```
//播放音乐
karaokeRoom.startPlayMusic(musicID: musicID, originalUrl:
muscicLocalPath, accompanyUrl: accompanyLocalPath);
//停止音乐
karaokeRoom.stopPlayMusic();
```

完成以上步骤，就可以实现 KTV 基本功能。如果您的业务还需要聊天、发送礼物等功能，可以接入以下能力。

步骤五：文字聊天功能（可选）

如果您需要实现各主播或听众之间文字聊天的功能，可以通过以下方法发送或接收聊天信息。
相关接口说明请参见 TRTCKaraokeRoom.sendRoomTextMsg 。

```
// 发送端：发送文本消息
karaokeRoom.sendRoomTextMsg(message: message) { [weak self] (code,
message) in
    if code == 0 {
        //发送成功
    }
}
// 接收端：监听文本消息
karaokeRoom.setDelegate(delegate: self)
func onRecvRoomTextMsg(message: String, userInfo: UserInfo) {
    debugPrint("收到来自：" + userInfo.userName + "的消息：" + message)
}
```

步骤六：礼物发送功能（可选）

如果您需要实现礼物发送和接收功能，可以通过以下方法发送礼物或接收礼物并展示。

```
// 发送端：通过自定义 "IMCMD_GIFT" 来区分礼物消息
karaokeRoom.sendRoomCustomMsg(cmd: kSendGiftCmd, message: message) {
code, msg in
    if (code == 0) {
        //发送成功
    }
}


// 接收端：监听礼物消息
karaokeRoom.setDelegate(delegate: self)
```

```
func onRecvRoomCustomMsg(cmd: String, message: String, userInfo:
UserInfo) {
    if cmd == kSendGiftCmd {
        debugPrint("收到来自：" + userInfo.userName + "的礼物：" + message)
    }
}
```

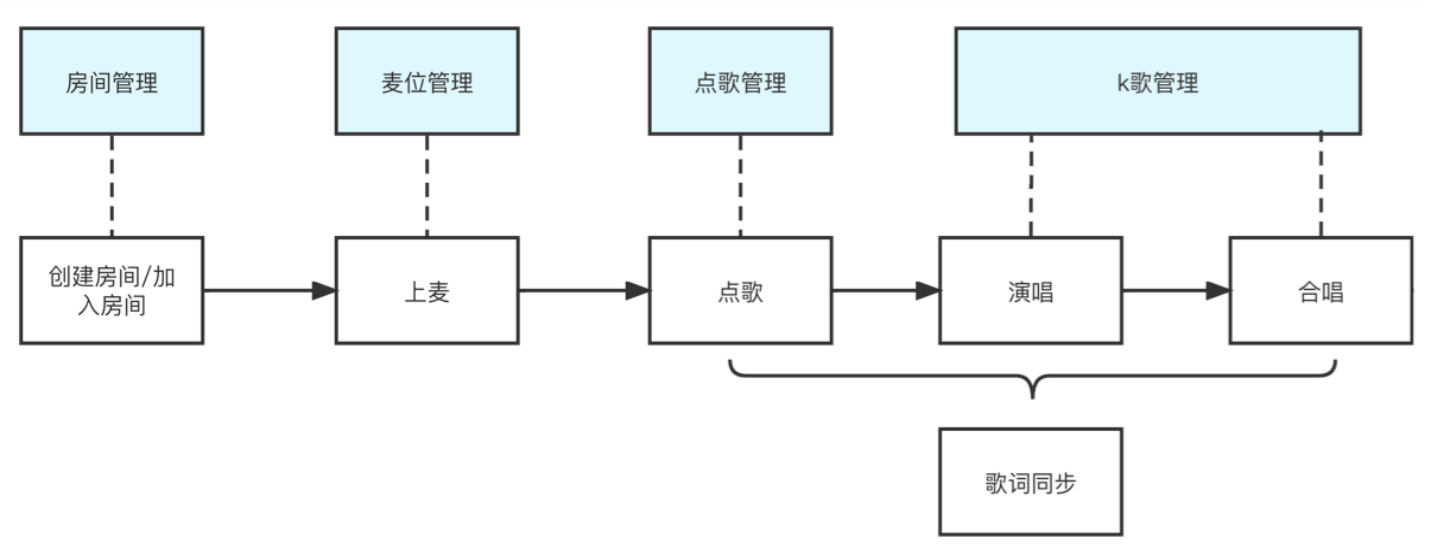
方案介绍（TUIKaraoke）
实现步骤

最近更新时间：2024-11-21 17:31:22

通常实现一个完整的在线 K 歌场景，需要涉及多个功能模块：房间管理、麦位管理、点歌管理、K 歌管理等。每个功能模块下的关键动作及功能点如下表所示。接下来会逐个介绍各个功能模块，通过介绍可以对搭建K歌房所需的功能有个完整的认知。

房间管理	麦位管理	点歌管理	K歌管理
房间列表	上/下麦	歌单展示	唱歌玩法
创建房间	麦位控制	搜索歌曲	切歌
加入房间	锁麦位	点歌	人声音量调节
退出房间	邀请上麦	歌曲置顶	混响/声效
销毁房间	麦位禁言	已点列表	歌词同步

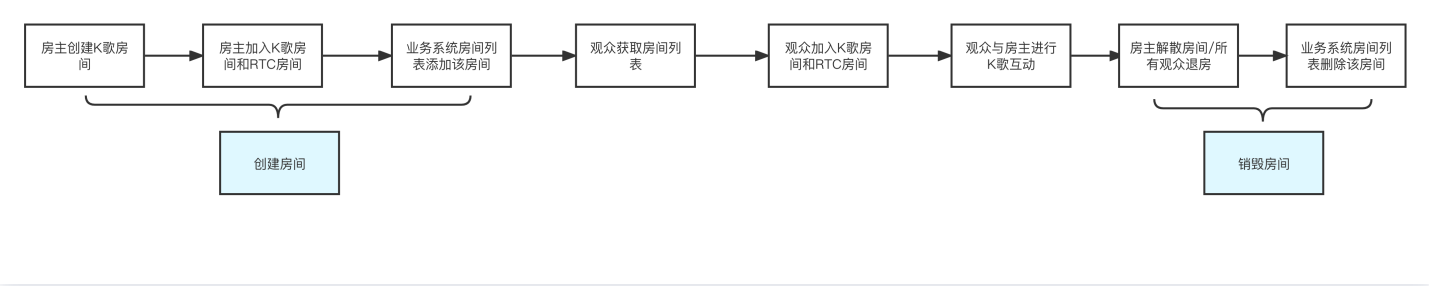
房主创建 K 歌房，用户可以选择感兴趣的歌房加入。进入房间后用户可以上麦参与互动，上麦后可以和房主语音互动。当然，用户也可以选择直接上麦参与合唱，这是两种不同的 K 歌玩法。在线 K 歌场景整体的业务流程如下图所示。



房间管理

房间管理是主要负责对房间列表的维护。主要包含的功能：创建房间、加入房间、销毁房间、退出房间。而且 K 歌房间有区别于普通房间，需要有单独的 K 歌房间标识符，以启动相关的组件管理：点歌管理、K 歌管理等功能。

- **创建房间**：用户登录业务系统后，可以创建房间，创建房间后房间列表要做新增操作。
- **销毁房间**：所有用户退出房间后，需要销毁房间，销毁房间后房间列表要做删除操作。



> **说明：**
> 房间管理是实现在线K歌的必要模块，但并非主要功能模块，具体实现可以结合业务系统和 TRTC SDK，详见语聊房场景接入方案。

麦位管理

歌房内的麦位一般都是有序且有限的。麦位管理主要负责根据业务场景定义房间内的麦位数量，以及当前房间所有麦位的状态管理。麦位管理主要包含的功能：上/下麦、锁麦位、邀请上麦、麦位禁言等。

- 用户进入房间后，只有空闲状态的麦位才可以申请上麦。
- 房主同意用户上麦后，需要修改麦位状态为非空闲状态。
- 用户停止推流下麦后，需要重置麦位状态。
- 房主有权锁定麦位、邀请上麦、强制下麦、麦上禁言等。

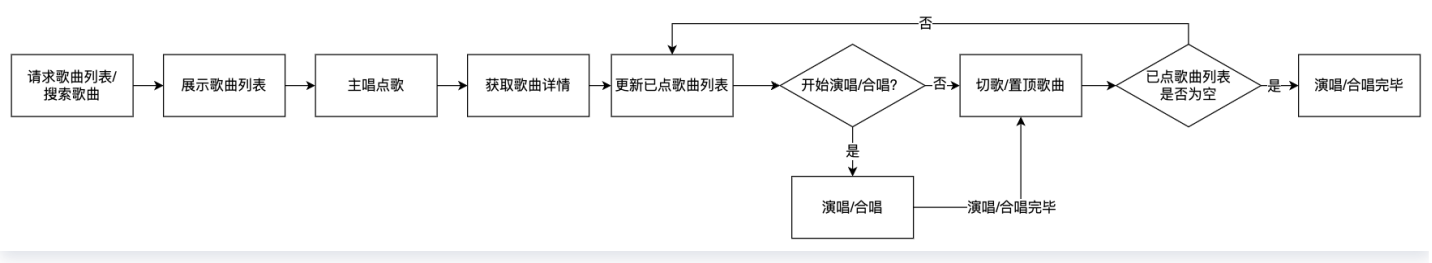
> **说明：**
> 麦位管理是实现在线K歌的必要模块，但并非主要功能模块，具体实现可以结合业务系统和 IM SDK，详见语聊房场景接入方案。

点歌管理

基本介绍

点歌管理属于在线 K 歌场景比较重要的一环，主要包含的功能：歌单展示、搜索歌曲、点歌和排麦、已点列表。而且每个 K 歌房间要有一个维护已点歌曲列表和自动排麦功能，都是需要业务后台来实现，而歌单展示、搜索歌曲需要结合 音速达直播音乐版权引擎（Yinsuda Authorized Music for Live Streaming）实现。

实现流程



整个点歌管理中，主要涉及业务端 App、业务后台、音速达后台，其中各自的职能分别为：

- **业务端 App：**
  - 调用点歌接口上报歌曲信息.
  - 调用切歌接口通知业务后台更新已点歌单列表.
  - 调用演唱确认接口通知业务后台。
- **业务后台：**
  - 维护已点歌单列表。
  - 下发通知告诉业务端 App 更新当前已点歌曲列表。
- **音速达后台：**
  - 提供获取直播互动歌曲推荐 歌单列表/歌单详情 的接口。
  - 提供 获取直播互动曲目详情 的接口（播放playToken、歌词下载URL）。
  - 提供 搜索直播互动歌曲 的接口。

K 歌管理

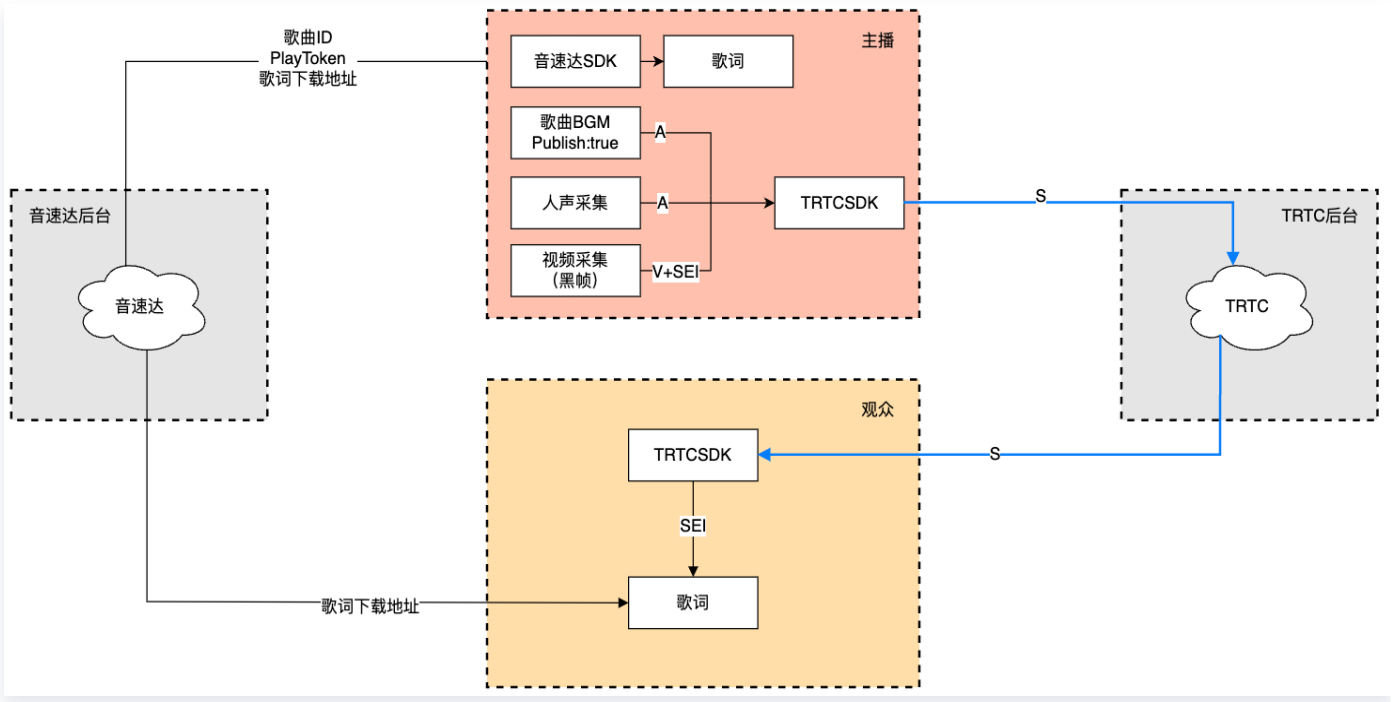
K 歌系统主要包含功能：唱歌玩法、开始/停止/切歌、人声音量调节、混响/声效、歌词同步。下面我们将通过演唱和实时合唱两种典型的 K 歌玩法来详细介绍 K 歌管理模块的实现流程。

演唱

主要是多人互动的KTV场景下，主播上麦后，才可以进行点歌，主播点歌成功后，会统一在点歌台展示，主播选择开始演唱。

（1）方案架构

在整体方案上，主要是用到的音速达SDK，实现歌曲下载，音速达后台，获取歌曲 playToken 和歌词下载地址；用到的 TRTC SDK，来实现演唱者人声的推流、歌曲的播放以及推流。整体的方案架构如下：



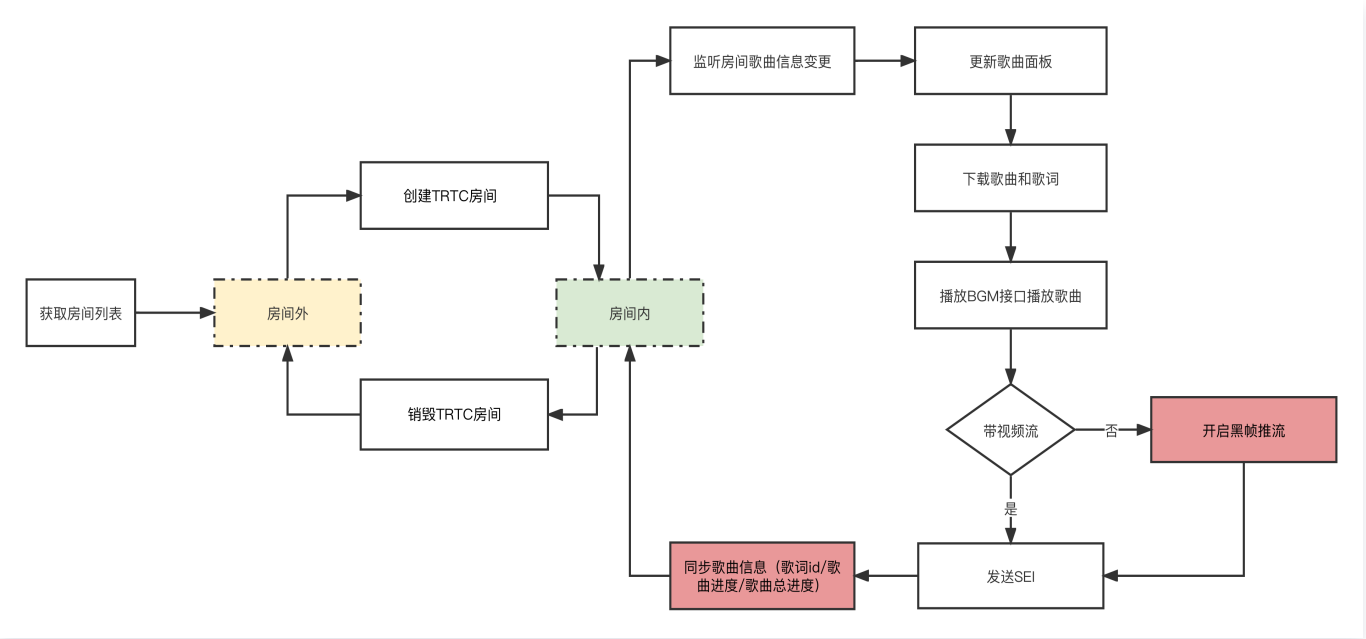
（2）具体实现

在演唱的场景，会区分不同角色有不同的实现流程，分为演唱者、观众 2种角色。

角色	描述	区别
演唱者	KTV 房间的演唱者，是房间的房主点歌以后演变而来的，房主创建房间后自动上麦点歌并演唱，退房后房间自动解散，已点歌曲自动清空	• 角色必须为主播 • 上行音视频（无视频上行黑帧） • 播放 BGM • 发送 SEI 信息（发送歌词信息） • 点歌
观众	KTV 房间的观众，播放演唱者的流	• 角色为观众，亦可上麦成为主播 • 下行音视频流 • 接收 SEI 信息（接收歌词信息）

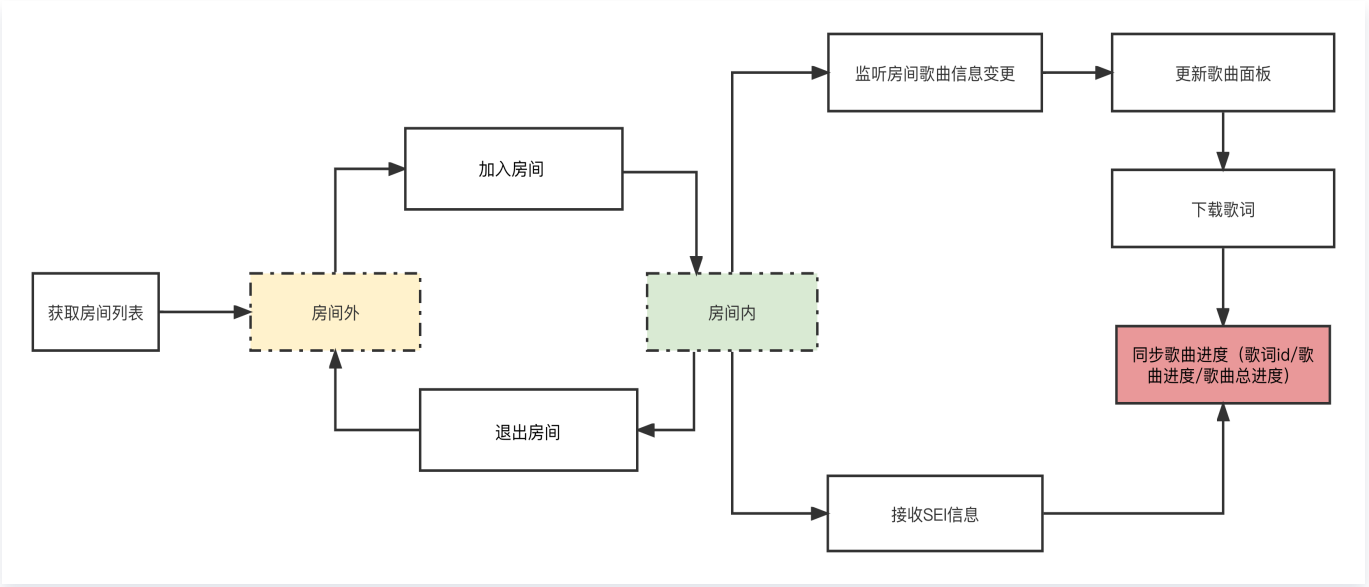
不同角色的基本实现流程如下：

房主



- 房主创建并加入 TRTC 的房间，自动上麦后点歌成为演唱者。
- 点歌后进行歌曲/歌词的下载，然后通过播放 BGM 接口播放歌曲。
- 演唱者没有带视频上行的话，需要开启视频上行。
- 通过 SEI 信息同步所有人的歌词进度。
- 演唱者在唱歌过程中可以随时切歌，并重新开始对歌曲/歌词进行下载，下载完成后进行演唱。
- 房主退房后将解散 TRTC 房间。

观众



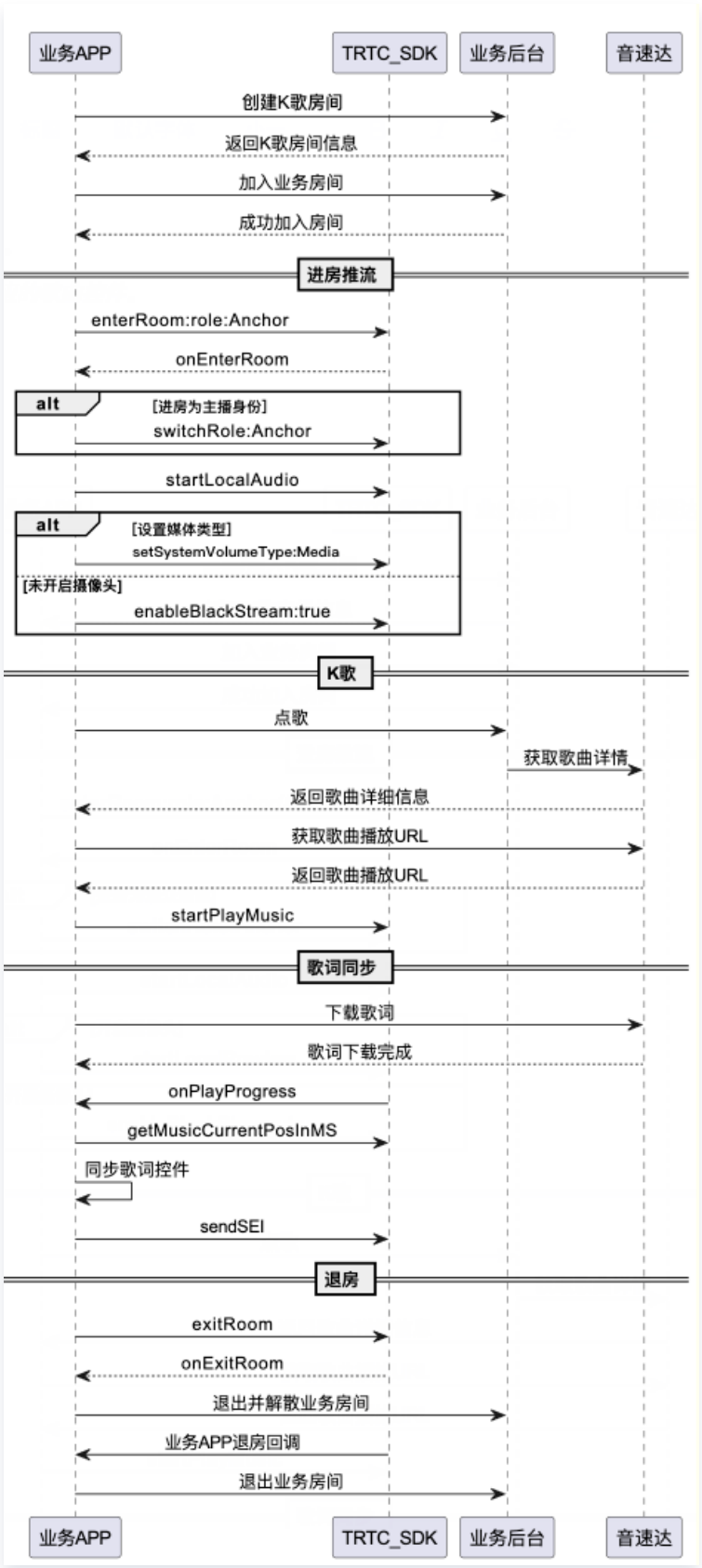
- 观众加入 TRTC 的房间。
- 监听房间歌曲变化，并加载歌词。
- 拉取演唱者的流。
- 解析演唱者发送的 SEI 信息，并同步歌词。

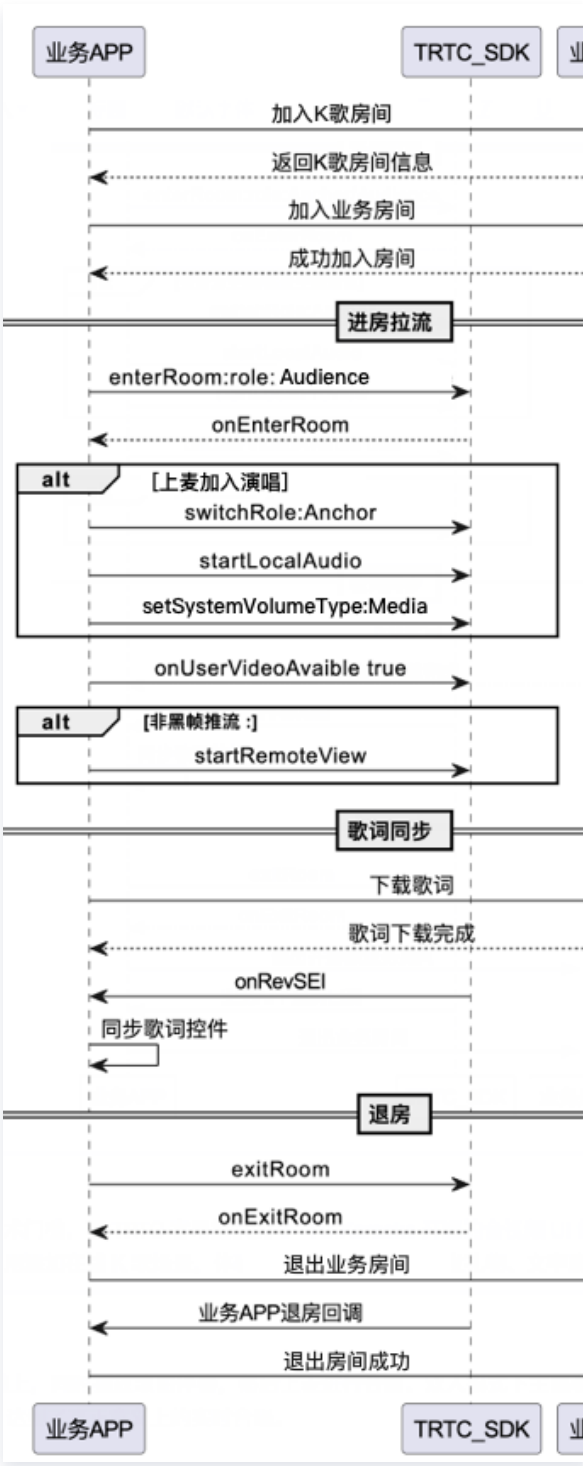
主要是要监听歌曲的 SEI 信息，更新对应的歌曲控件。

（3）API 调用时序

不同角色的 API 时序调用如下：

房主	观众


业务APP
TRTC_SDK
业务后台
音速达
创建K歌房间
返回K歌房间信息
加入业务房间
成功加入房间
进房推流
enterRoom:role:Anchor
onEnterRoom
alt
[进房为主播身份]
switchRole:Anchor
startLocalAudio
alt
[设置媒体类型]
setSystemVolumeType:Media
[未开启摄像头]
enableBlackStream:true
K歌
点歌
获取歌曲详情
返回歌曲详细信息
获取歌曲播放URL
返回歌曲播放URL
startPlayMusic
歌词同步
下载歌词
歌词下载完成
onPlayProgress
getMusicCurrentPosInMS
同步歌词控件
sendSEI
退房
exitRoom
onExitRoom
退出并解散业务房间
业务APP退房回调
退出业务房间
业务APP
TRTC_SDK
业务后台
音速达


业务APP
TRTC_SDK
加入K歌房间
返回K歌房间信息
加入业务房间
成功加入房间
进房拉流
enterRoom:role: Audience
onEnterRoom
alt
[上麦加入演唱]
switchRole:Anchor
startLocalAudio
setSystemVolumeType:Media
onUserVideoAvaible true
alt
[非黑帧推流 :]
startRemoteView
歌词同步
下载歌词
歌词下载完成
onRevSEI
同步歌词控件
退房
exitRoom
onExitRoom
退出业务房间
业务APP退房回调
退出房间成功
业务APP
TRTC_SDK

说明：

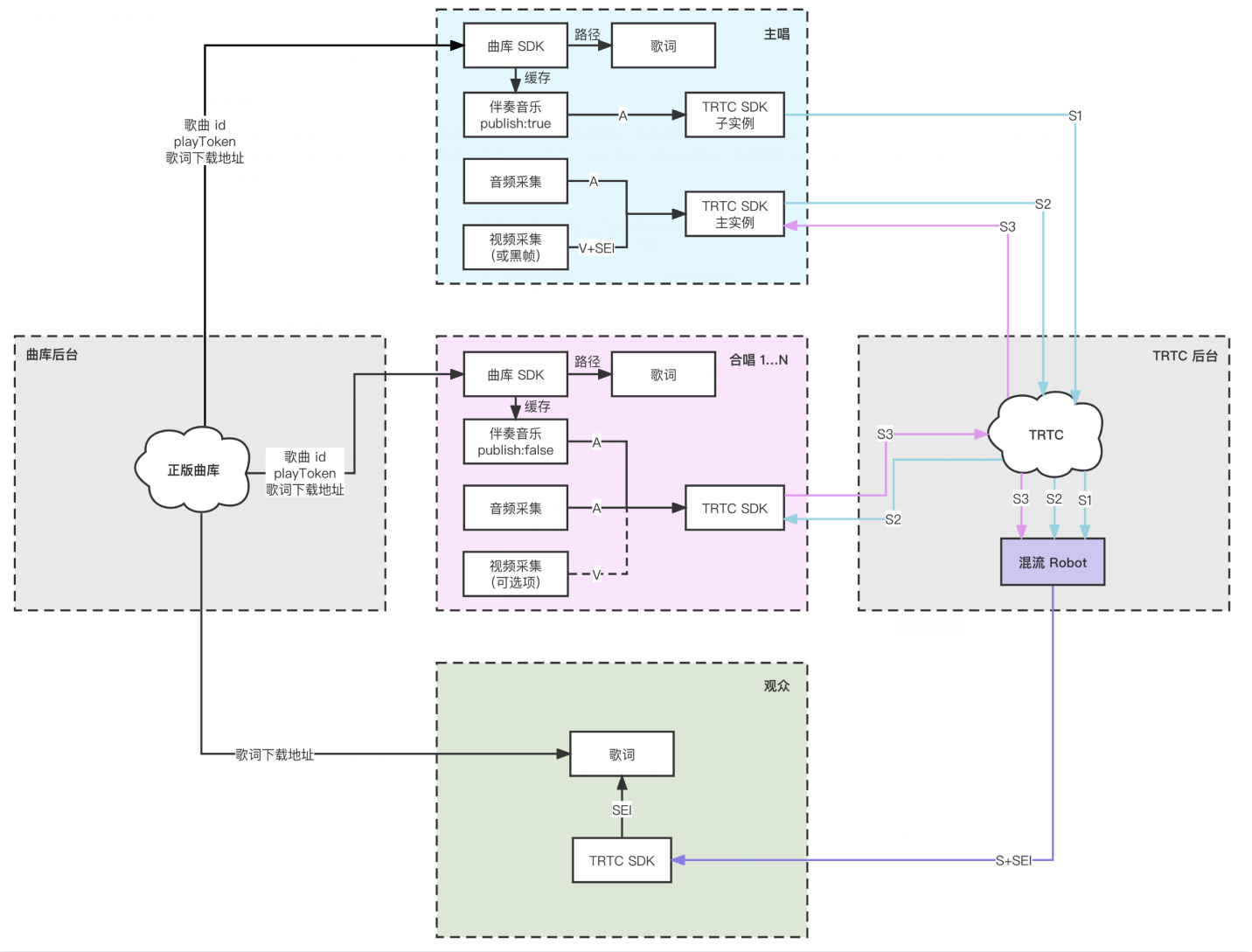
> 鉴于以上实现方案需要一定的技术门槛，而 TRTC 官网提供的 TUIKaraoke 开源的音视频 UI 组件，可以通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加在线 K 歌场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。

实时合唱

实时合唱是指各端在连麦的基础上，同时播放歌曲，然后上麦进行合唱。多人模式下合唱者之间都能听到彼此声音，几乎感受不到延迟，达到了真正意义上的实时合唱。

（1）方案架构

在媒体流方面，合唱者互相进行推拉流，同时会由一名***主唱者推出音乐***，其他***合唱者在本地播放音乐***，经过 NTP 进行时间同步。另外，歌曲和所有合唱者的声音都通过混流机器人进行混流处理形成一条流，并回推到 TRTC 房间，观众只需拉一条流即可听到各端同步的声音，实现多人合唱的效果。实时合唱方案架构如下图所示。



该方案的优点在于：

- 降低了端到端的时延。
- 提供了用户中途加入合唱的解决方案。
- 精准同步不同端之间的音乐、歌词、人声。
- 改善各端设备性能和本地时间不精准的情况，降低网络环境造成的时延影响。

说明：
- 根据业务需要，可以选择纯音频场景或音视频场景的实时合唱方案；若为纯音频场景则需要补黑帧以发送 SEI 消息用于歌词同步；
- 主唱需要使用子实例同时上行音乐及人声；其他合唱者仅互相拉取人声流，同时本地播放音乐；观众只需拉取一路混流；
- 图中展示的是 RTC 观看方案，混流机器人将合流回推至 RTC 房间；CDN 观看方案下，混流机器人将合流转推至直播 CDN，观众拉取 CDN 流观看。

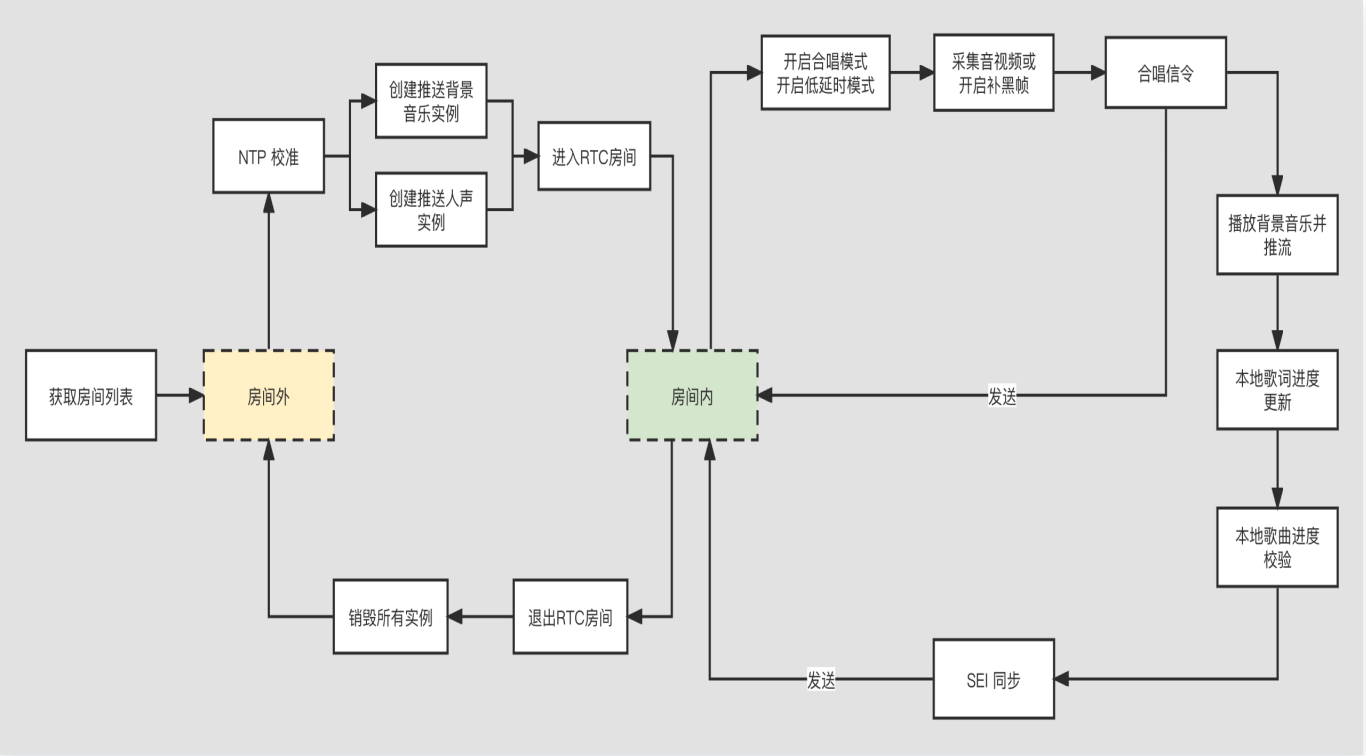
（2）具体实现

我们可以将在线 K 歌房里的用户划分为三种角色：主唱、合唱和观众，如下表所示。

角色	描述	区别
主唱	主唱负责点歌和发送合唱信令，以及 SEI 的发送	• 角色为 Anchor • 上行音乐和人声 • 点歌及发起合唱 • 混流回推 • 发送 SEI 消息
合唱	合唱者可以接收并处理合唱信令，麦上参与合唱	• 角色为 Anchor • 上行人声 • 本地播放音乐 • 接收合唱
观众	进入歌房后，在麦下拉流的观众，也可上麦合唱	• 角色为 Audience • 下行混流 • 接收 SEI 消息 • 申请上麦成为 Anchor

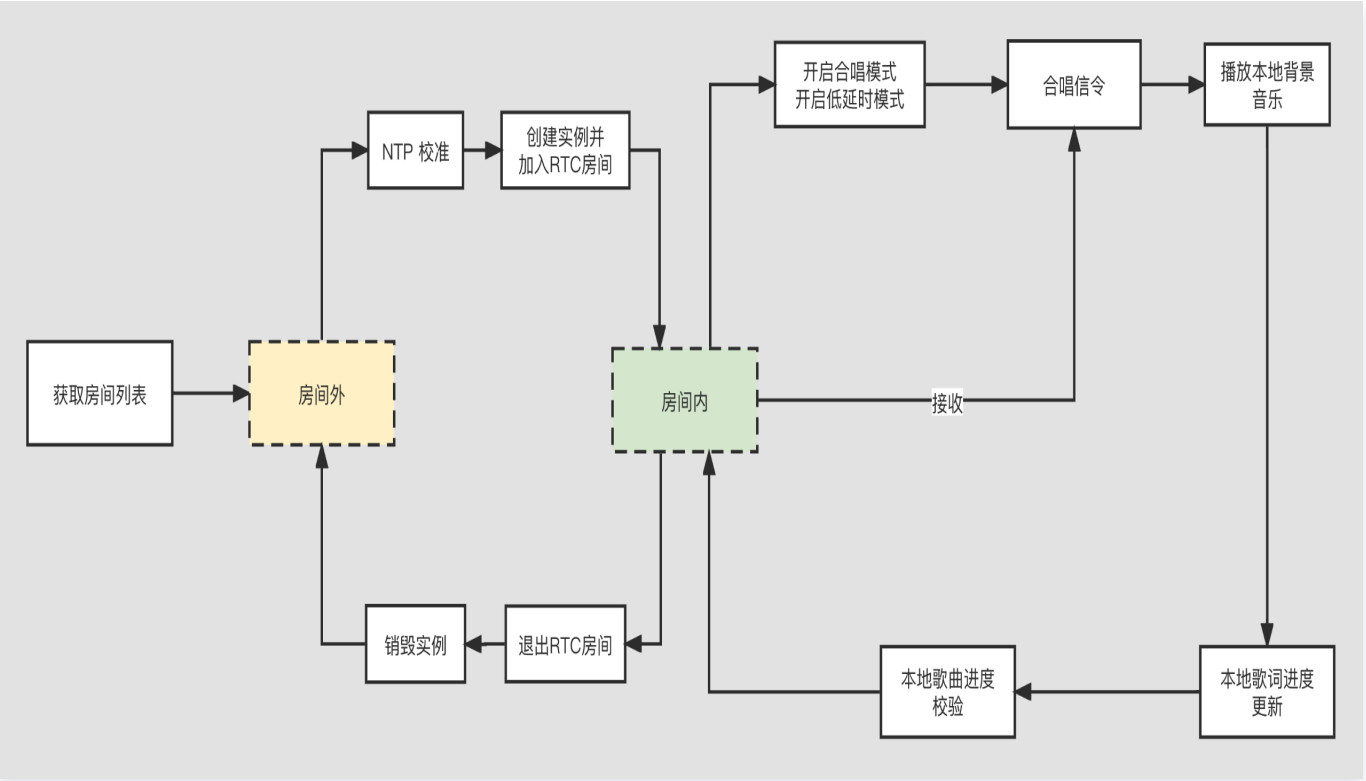
不同角色的基本实现流程如下图所示：

主唱



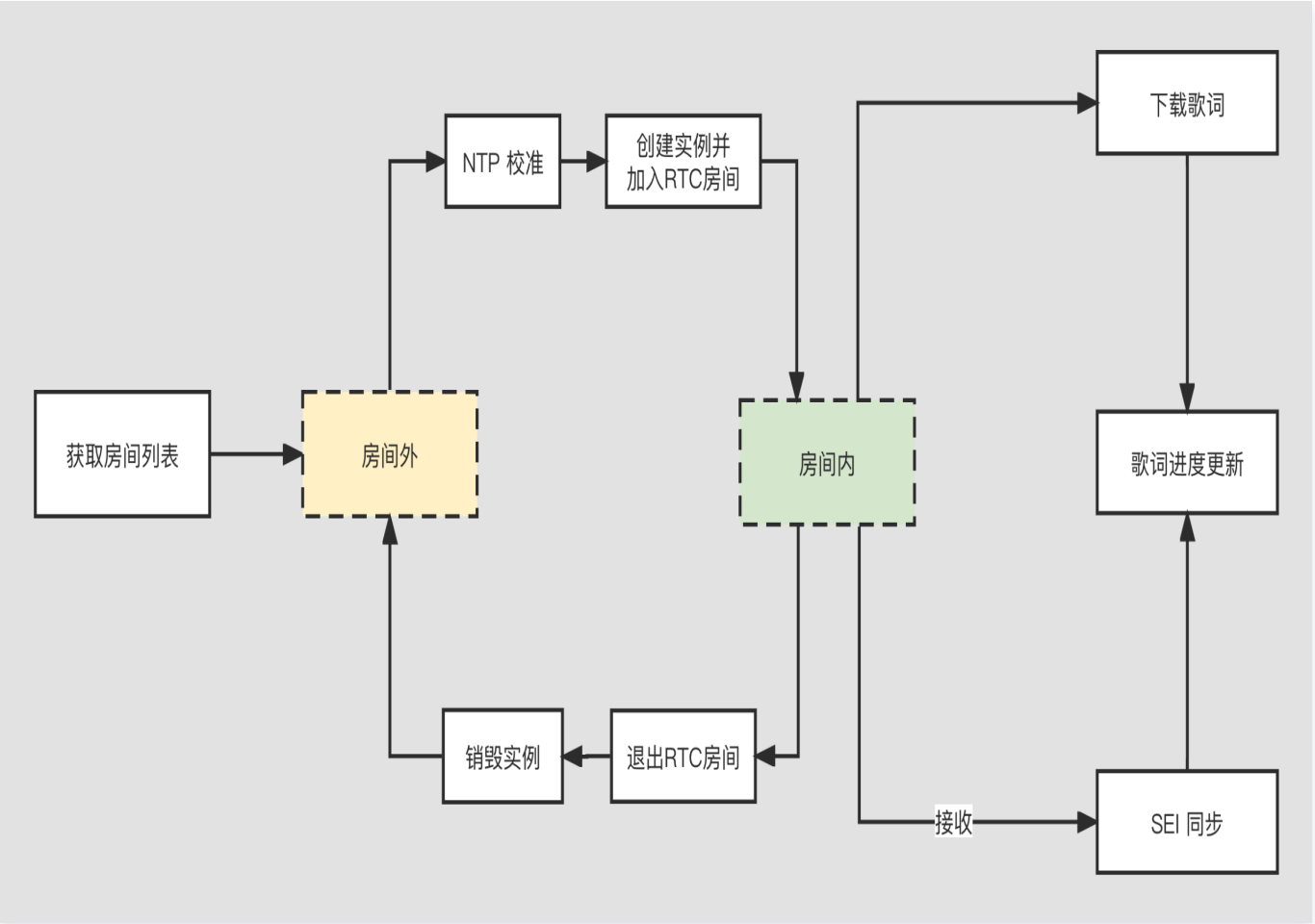
- 主唱需要点播歌曲，发送合唱信令。
- 主唱创建子实例推送人声和音乐，并拉取其他合唱者的人声。
- 在推流后，主唱负责发起混流转推任务。
- 开唱后，播放音乐，通过播放进度回调同步歌词。
- 需要通过 SEI 发送歌曲进度，以便观众端同步歌词。
- 所有演唱者需要根据 NTP 校准本地歌曲播放进度。

合唱



- 合唱者推送一路人声流，拉麦上合唱用户的人声流。
- 合唱者需要监听并接收合唱信令，预加载音乐资源。
- 开唱后，本地播放音乐，合唱者通过播放进度回调同步歌词。
- 所有演唱者需要根据 NTP 校准本地歌曲播放进度。

观众

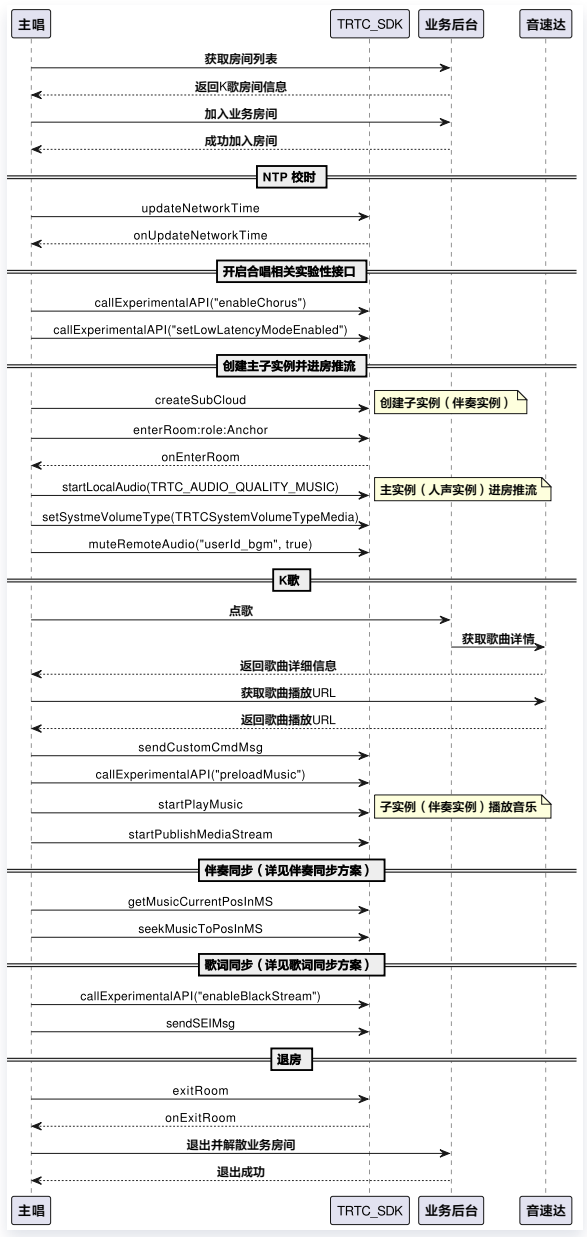


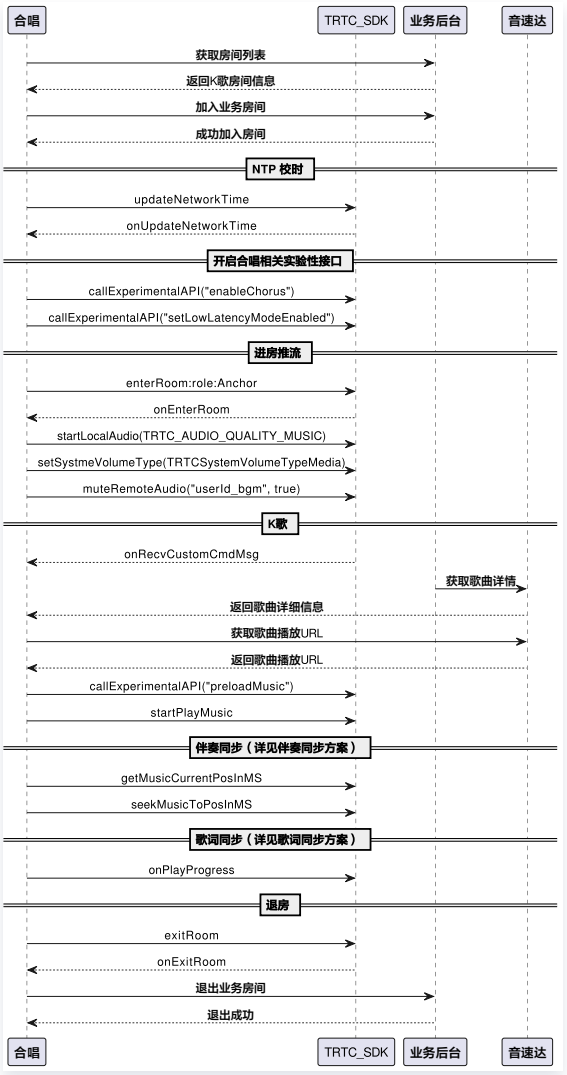
- 拉混流收听合唱。
- 解析混流中 SEI 的歌曲进度信息，用于同步歌词；
- 上麦后停止拉混流，转为拉麦上人声流，同时开启合唱模式。

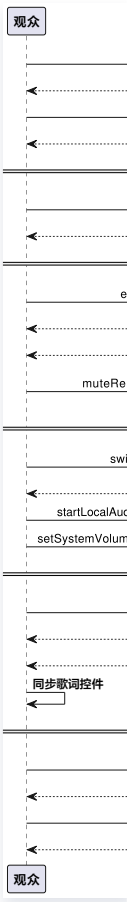
（3）API 调用时序

不同角色的 API 调用时序如下：

主唱 API 时序	合唱 API 时序	







说明：

鉴于以上实现方案需要一定的技术门槛，而 TRTC 官网提供的 TUIKaraoke 开源的音视频 UI 组件，可以通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加实时合唱场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。

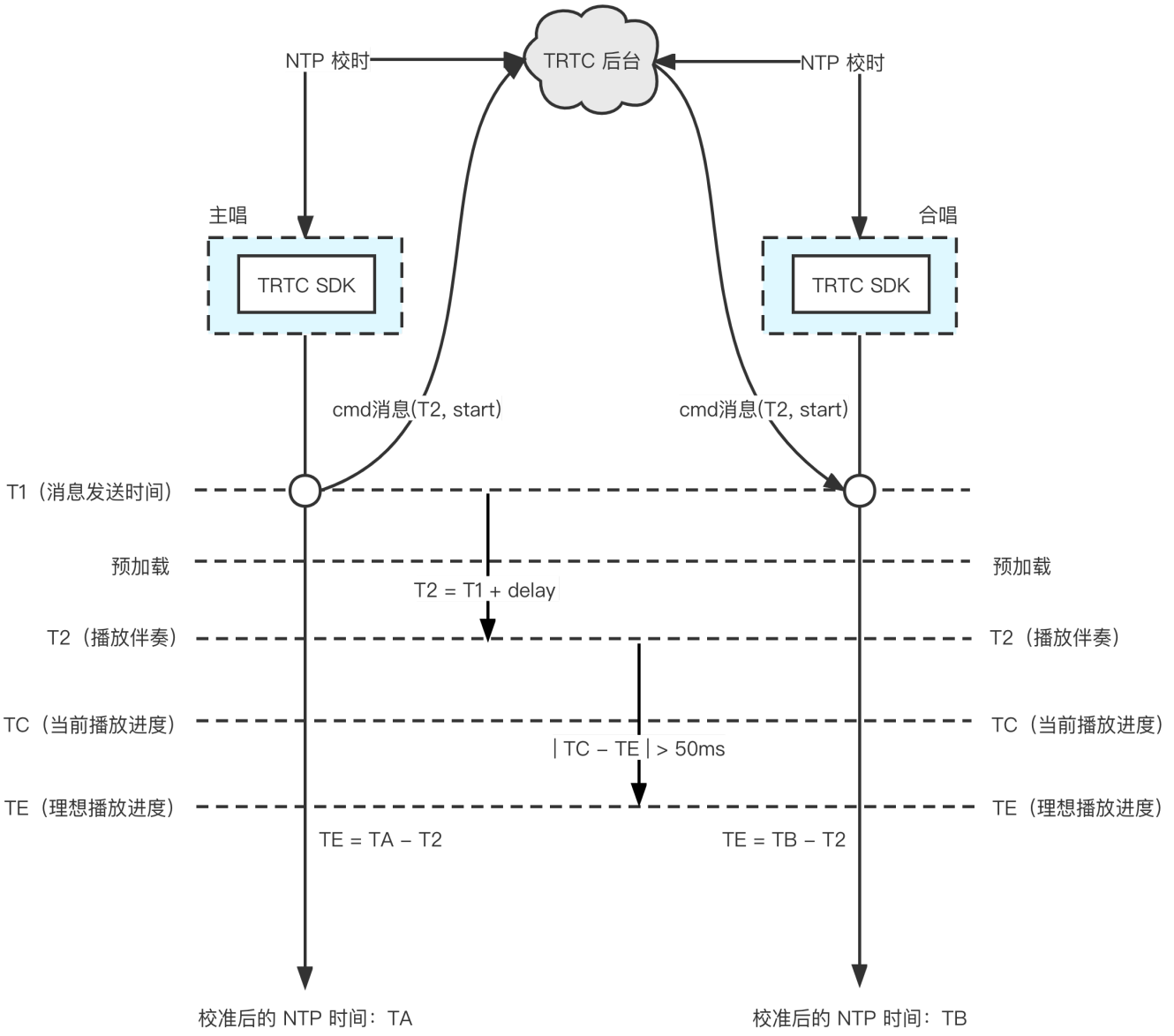
歌曲同步
Android

最近更新时间：2022-10-31 16:18:12

实时方案中需要在开唱后实时同步歌曲进度，避免因歌曲误差而增加端到端延迟。同步歌曲需要基于 NTP 时间，不同设备的本地时钟并不一致，存在一定误差，因此需要引入腾讯云自研 NTP 服务。同时，中途加入合唱的用户也需要同步歌曲进度，只有同步进度后，才能参与合唱。

实现流程

歌曲同步的做法为：主唱用户约定在未来某个时间点（如延迟 N 秒后）开始播放歌曲，其他用户参与合唱。各端的时间都以 NTP 时间为准，NTP 时间会在 TRTC SDK 初始化后开始同步。

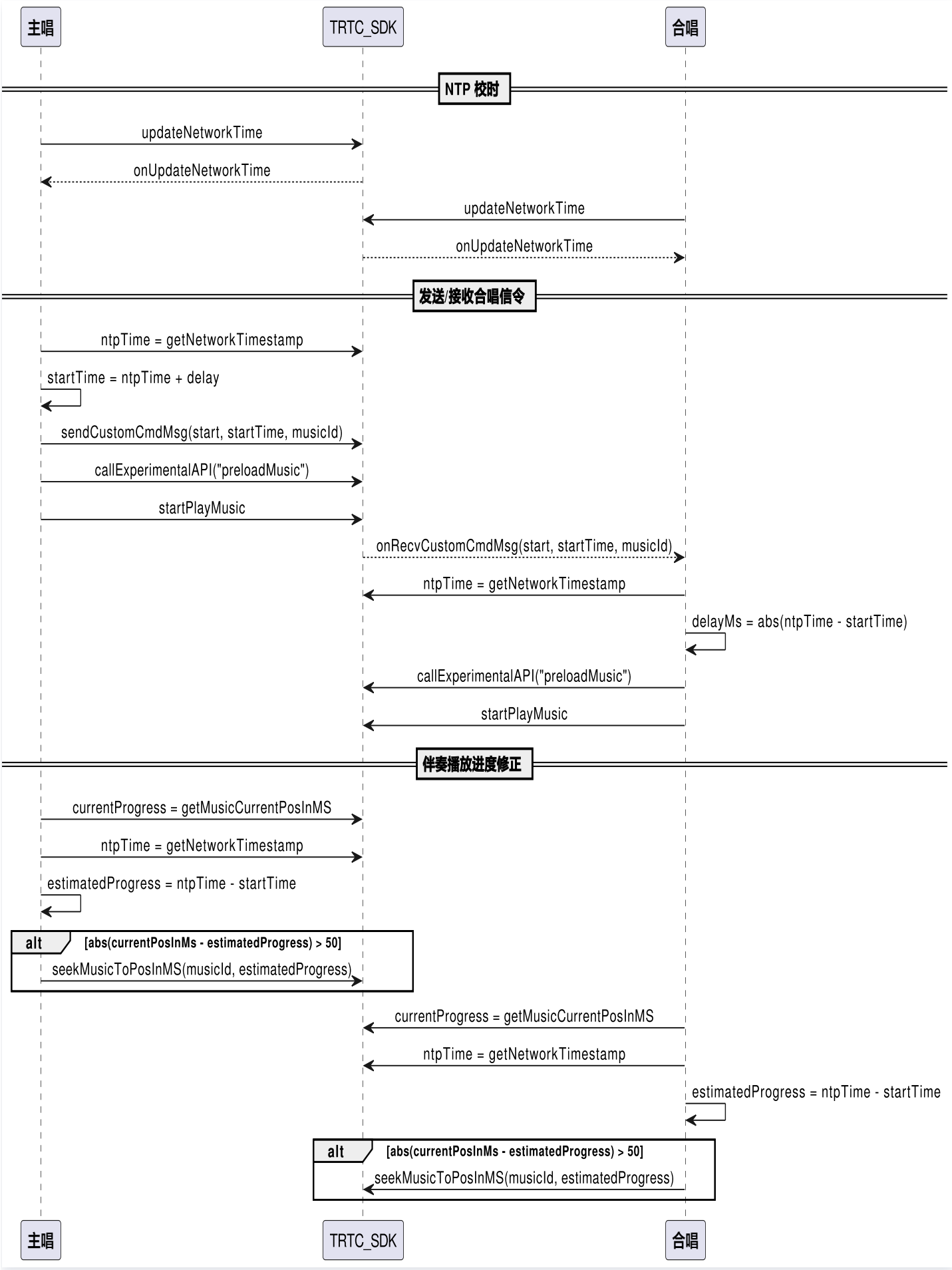


具体流程如下：

1. 各端进行 NTP 校时，向 TRTC 云端更新并获取最新的 NTP 时间 T。
2. 主唱端发送合唱信令（自定义消息），约定合唱开始时间 T2。
3. 本地根据 T2 预加载歌曲，定时播放。
4. 其他合唱用户接收到合唱信令后执行步骤 3。
5. 过程中对本地歌曲播放进度进行校验，当 TE 与 TC 差值超过 50ms 即 seek 校准。

> **说明：**
> 这里的 50ms 误差是个典型值，可以根据业务容忍度适当调整，建议在 50ms 上下浮动。

时序图


主唱
TRTC_SDK
合唱
NTP 校时
updateNetworkTime
onUpdateNetworkTime
updateNetworkTime
onUpdateNetworkTime
发送/接收合唱信令
ntpTime = getNetworkTimestamp
startTime = ntpTime + delay
sendCustomCmdMsg(start, startTime, musicId)
callExperimentalAPI("preloadMusic")
startPlayMusic
onRecvCustomCmdMsg(start, startTime, musicId)
ntpTime = getNetworkTimestamp
delayMs = abs(ntpTime - startTime)
callExperimentalAPI("preloadMusic")
startPlayMusic
伴奏播放进度修正
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
主唱
TRTC_SDK
合唱

歌曲同步时序主要可以分为三个部分：NTP 校时、发送及接收合唱信令、歌曲播放进度修正。下面将针对这四部分给出具体的代码实现。

关键代码实现

1. NTP 校时服务

```
TXLiveBase.setListener(new TXLiveBaseListener() {
    @Override
    public void onUpdateNetworkTime(int errCode, String errMsg) {
        super.onUpdateNetworkTime(errCode, errMsg);
        // errCode 0: 校时成功且偏差在30ms以内；1: 校时成功但偏差可能在30ms以
上；-1: 校时失败
        if (errCode == 0) {
            // 调用TXLivebase的getNetworkTimestamp即可获取NTP时间戳
            long ntpTime = TXLiveBase.getNetworkTimestamp();
        } else {
            // 重新调用updateNetworkTime启动一次校时
            TXLiveBase.updateNetworkTime();
        }
    }
});
TXLiveBase.updateNetworkTime();
```

2. 主唱端发送合唱信令

```
JSONObject jsonObject = new JSONObject();
jsonObject.put("cmd", "startChorus");
// 约定时间合唱
jsonObject.put("startPlayMusicTS", startTs);
jsonObject.put("musicId", "musicId");
String body = jsonObject.toString();
mTRTCCloud.sendCustomCmdMsg(0, body.getBytes(), false, false);
```

说明：

- 建议主唱以固定时间频率循环向房间内发送合唱信令消息，以便合唱者可以中途加入合唱。
- **不使用 SEI 消息发送合唱信令**的原因：SEI 信息会插入到视频帧中，导致观众侧拉取的视频流中携带很多无效信息。

3. 合唱端接收合唱信令

```
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[]
message) {
    JSONObject json = new JSONObject(new String(message, "UTF-8"));
    String cmd = json.getString("cmd");
    // 合唱命令
    if (cmd.equals("startChorus")) {
    // 合唱开始时间
    long startPlayMusicTs = json.getLong("startPlayMusicTS");
    int musicId = json.getInt("musicId");
    // 合唱约定时间和当前时间差值
    long delayMs = Math.abs(startPlayMusicTs - getNtpTime());
    // 开启预加载，根据合唱约定时间和当前ntp差值，跳跃歌曲进度
    mTRTCCloud.callExperimentalAPI("
{\"api\":\"preloadMusic\",\"params\":
{\"musicId\":musicId,\"path\":\"path\",\"startTimeMS\":delayMs}}");
    // 播放歌曲
    TXAudioEffectManager.AudioMusicParam param = new
TXAudioEffectManager.AudioMusicParam(musicId, musicPath);
    param.publish = false;
    mTRTCCloud.getAudioEffectManager().startPlayMusic(param);
}
```

说明：
合唱端接收到第一个 startChorus 信令后，状态应从“未合唱”转为“合唱中”，此后不再响应 startChorus 信令，避免重复启动 BGM 播放。

4. 歌曲播放进度修正

```
long mStartPlayMusicTs = "最初约定的合唱时间";
long currentProgress =
subCloud.getAudioEffectManager().getMusicCurrentPosInMS(musicID);
// 当前歌曲的理想播放时间进度
long estimatedProgress = getNtpTime() - mStartPlayMusicTs;
// 当播放进度超过50ms，进行修正
if (estimatedProgress >= 0 &&; Math.abs(currentProgress -
estimatedProgress) > 50) {
    subCloud.getAudioEffectManager().seekMusicToPosInMS(mMusicID, (int)
estimatedProgress);
```

```
}
```

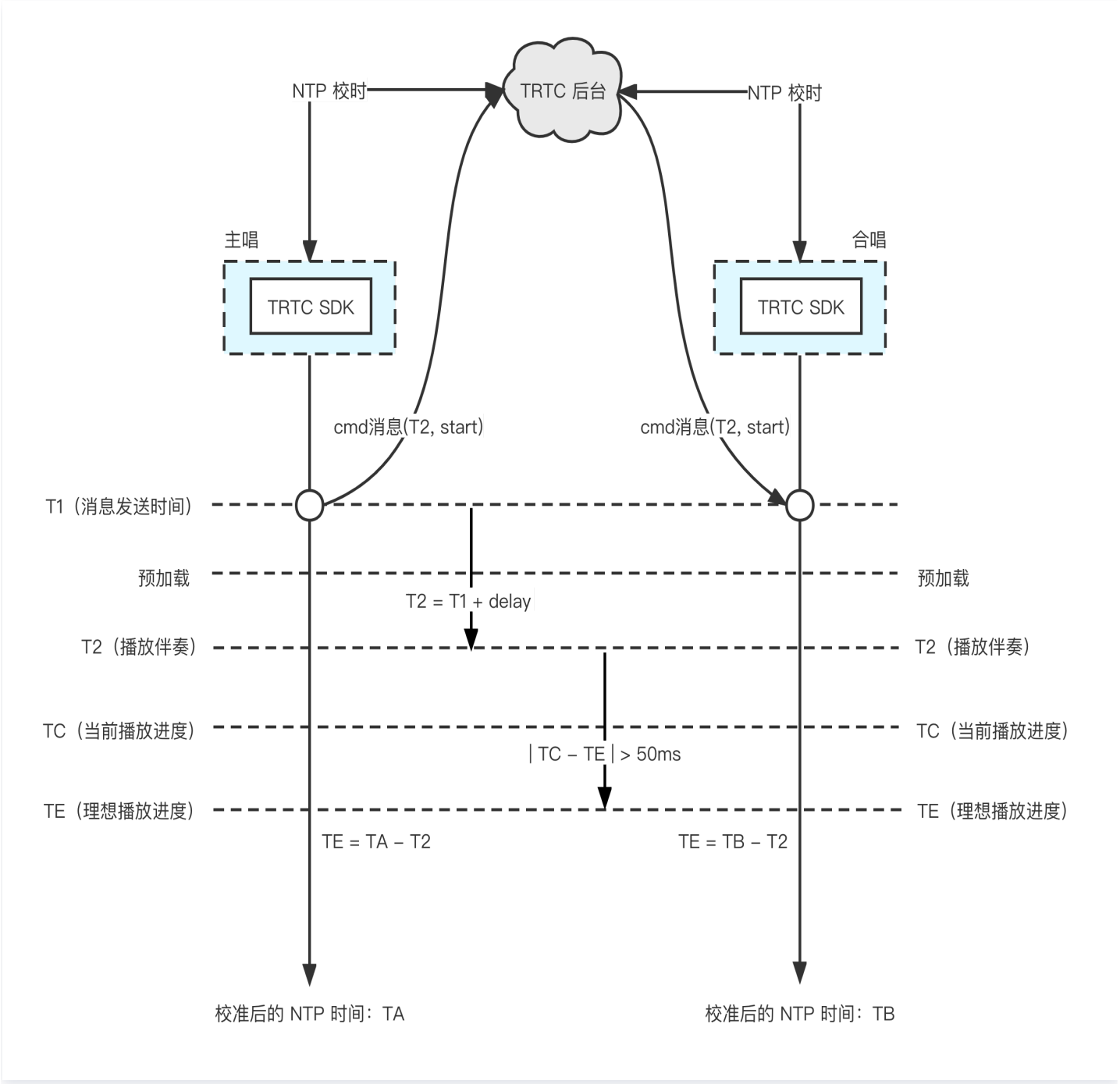
iOS

最近更新时间：2022-10-31 16:18:12

实时方案中需要在开唱后实时同步歌曲进度，避免因歌曲误差而增加端到端延迟。同步歌曲需要基于 NTP 时间，不同设备的本地时钟并不一致，存在一定误差，因此需要引入腾讯云自研 NTP 服务。同时，中途加入合唱的用户也需要同步歌曲进度，只有同步进度后，才能参与合唱。

实现流程

歌曲同步的做法为：主唱用户约定在未来某个时间点（如延迟 N 秒后）开始播放歌曲，其他用户参与合唱。各端的时间都以 NTP 时间为准，NTP 时间会在 TRTC SDK 初始化后开始同步。

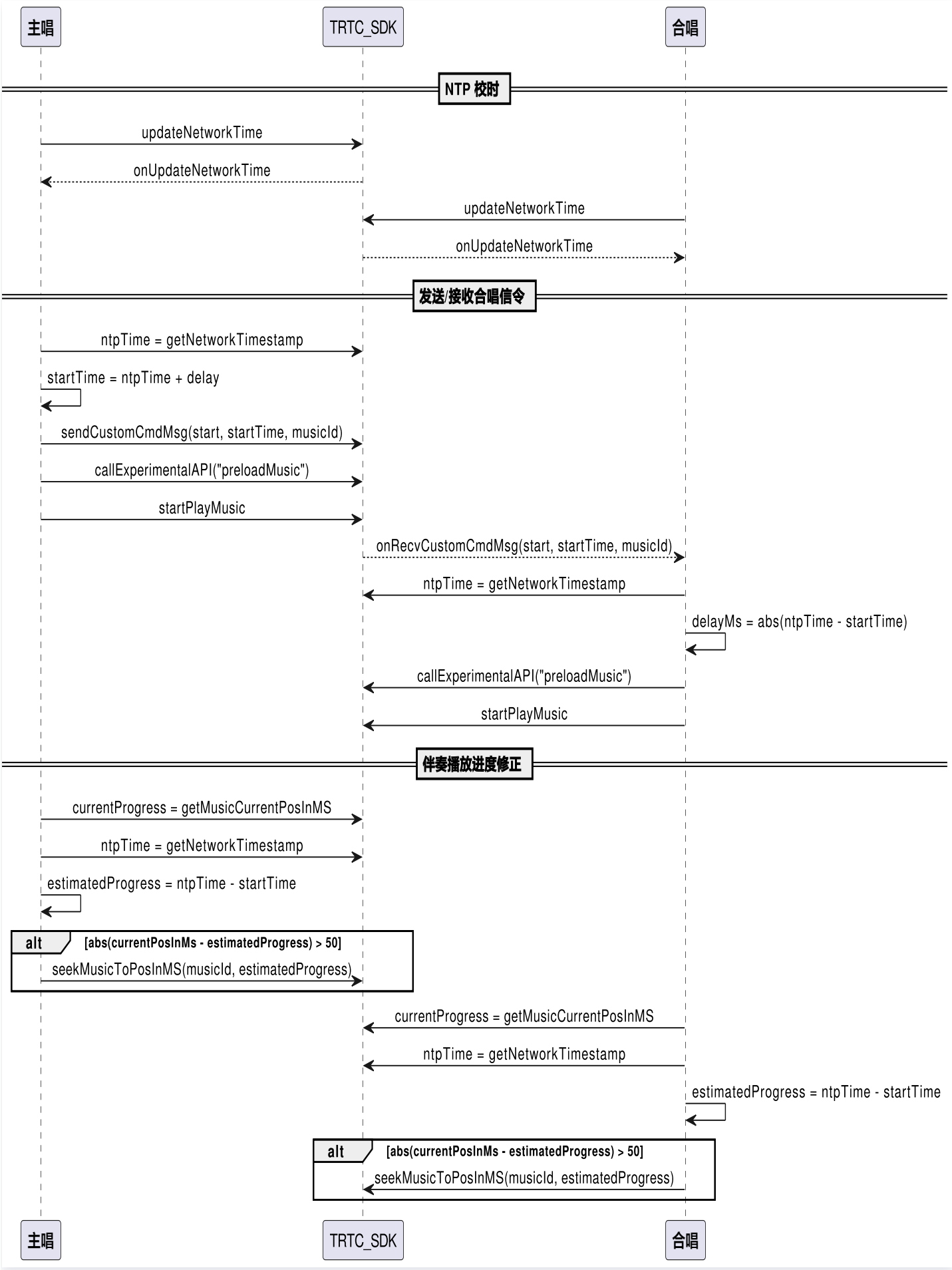


具体流程如下：

1. 各端进行 NTP 校时，向 TRTC 云端更新并获取最新的 NTP 时间 T。
2. 主唱端发送合唱信令（自定义消息），约定合唱开始时间 T2。
3. 本地根据 T2 预加载歌曲，定时播放。
4. 其他合唱用户接收到合唱信令后执行步骤 3。
5. 过程中对本地歌曲播放进度进行校验，当 TE 与 TC 差值超过 50ms 即 seek 校准。

> **说明：**
> 这里的 50ms 误差是个典型值，可以根据业务容忍度适当调整，建议在 50ms 上下浮动。

时序图


主唱
TRTC_SDK
合唱
NTP 校时
updateNetworkTime
onUpdateNetworkTime
updateNetworkTime
onUpdateNetworkTime
发送/接收合唱信令
ntpTime = getNetworkTimestamp
startTime = ntpTime + delay
sendCustomCmdMsg(start, startTime, musicId)
callExperimentalAPI("preloadMusic")
startPlayMusic
onRecvCustomCmdMsg(start, startTime, musicId)
ntpTime = getNetworkTimestamp
delayMs = abs(ntpTime - startTime)
callExperimentalAPI("preloadMusic")
startPlayMusic
伴奏播放进度修正
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
主唱
TRTC_SDK
合唱

歌曲同步时序主要可以分为三个部分：NTP 校时、发送及接收合唱信令、歌曲播放进度修正。下面将针对这四部分给出具体的代码实现。

关键代码实现

1. NTP 校时服务

```
//进房时调用NTP校时接口
[TXLiveBase updateNetworkTime];

// TXLiveBaseDelegate 回调内判断是否校时成功
- (void)onUpdateNetworkTime:(int)errCode message:(NSString *)errMsg {
    // errCode 0: 校时成功且偏差在30ms以内；1: 校时成功但偏差可能在30ms以上；-1:
校时失败
    if (errCode == 0) {
        // 调用TXLivebase的getNetworkTimestamp即可获取NTP时间戳
        NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    } else {
        // 重新调用updateNetworkTime启动一次校时
        [TXLiveBase updateNetworkTime];
    }
}
```

2. 主唱端发送合唱信令

```
NSDictionary *json = @{
                        @"cmd": @"startChorus",
                        // 约定时间合唱
                        @"startPlayMusicTS": @(startTs),
                        @"musicId": @"musicId",
                        @"musicDuration": @(musicDuration),
                      };
NSString *jsonString = [self jsonStringFrom:json];
[trtcCloud sendCustomMessage:jsonString reliable:NO];
```

说明：

- 建议主唱以固定时间频率循环向房间内发送合唱信令消息，以便合唱者可以中途加入合唱；
- **不使用 SEI 消息发送合唱信令**的原因： SEI 信息会插入到视频帧中，导致观众侧拉取的视频流中携带很多无效信息。

3. 合唱端接收合唱信令

```
- (void)onRecvCustomCmdMsgUserId:(NSString *)userId cmdID:
(NSInteger)cmdId seq:(UInt32)seq message:(NSData *)message {

    NSString *msg = [[NSString alloc] initWithData:message
encoding:NSUTF8StringEncoding];
    NSError *error;
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[msg
dataUsingEncoding:NSUTF8StringEncoding]

options:NSJSONReadingMutableContainers

error:&error];

    NSObject *cmdObj = [json objectForKey:@"cmd"];

    NSInteger musicDuration = [[json objectForKey:@"musicDuration"]
integerValue];

    NSString *cmd = (NSString *)cmdObj;

    // 合唱命令
    if ([cmd isEqualToString:@"startChorus"]) {
        // 合唱开始时间
        NSObject *startPlayMusicTsObj = [json
objectForKey:@"startPlayMusicTS"];
        NSString *musicId = [json objectForKey:@"musicId"];

        NSInteger startPlayMusicTs = ((NSNumber
*)startPlayMusicTsObj).longLongValue;
        // 合唱约定时间和当前时间差值
        NSInteger startDelayMS = labs(startPlayMusicTs - [TXLiveBase
getNetworkTimestamp]);
        // 开启预加载，根据合唱约定时间和当前ntp差值，跳跃歌曲进度
        NSDictionary *jsonDict = @{
            @"api": @"preloadMusic",
            @"params": @{
                    @"musicId": @(musicId),
                    @"path": path,
                    @"startTimeMS": @(startDelayMS),
            }
        };
```

```
        NSData *jsonData = [NSJSONSerialization
dataWithJSONObject:jsonDict options:0 error:NULL];
        NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
        [subCloud callExperimentalAPI:jsonString];

        // 播放歌曲
        TXAudioMusicParam *param = [[TXAudioMusicParam alloc] init];
        param.ID = musicId;
        param.path = url;
        param.loopCount = 0;
        param.publish = NO;

        [[subCloud getAudioEffectManager] startPlayMusic:param
onStart:^(NSInteger errCode) {
            // 开始播放回调
        } onProgress:^(NSInteger progressMs, NSInteger durationMs) {
            // 歌曲进度回调
        } onComplete:^(NSInteger errCode) {
            // 播放完成回调
        }];

    }
}
```

说明：
合唱端接收到第一个 startChorus 信令后，状态应从“未合唱”转为“合唱中”，此后不再响应 startChorus 信令，避免重复启动 BGM 播放。

4. 歌曲播放进度修正

```
self.startPlayChorusMusicTs; //最初约定的合唱时间
// 当前播放进度
NSInteger currentProgress = [[self audioEffecManager]
getMusicCurrentPosInMS:self.currentPlayMusicID];
// 当前歌曲的理想播放时间进度
NSInteger estimatedProgress = [TXLiveBase getNetworkTimestamp] -
self.startPlayChorusMusicTs;
if (estimatedProgress >= 0 && labs(currentProgress - estimatedProgress)
> 50) {
    // 当播放进度超过50ms，进行修正
```

```
    [[subCloud getAudioEffectManager]
seekMusicToPosInMS:self.currentPlayMusicID pts:estimatedProgress];
}
```

歌词同步
Android

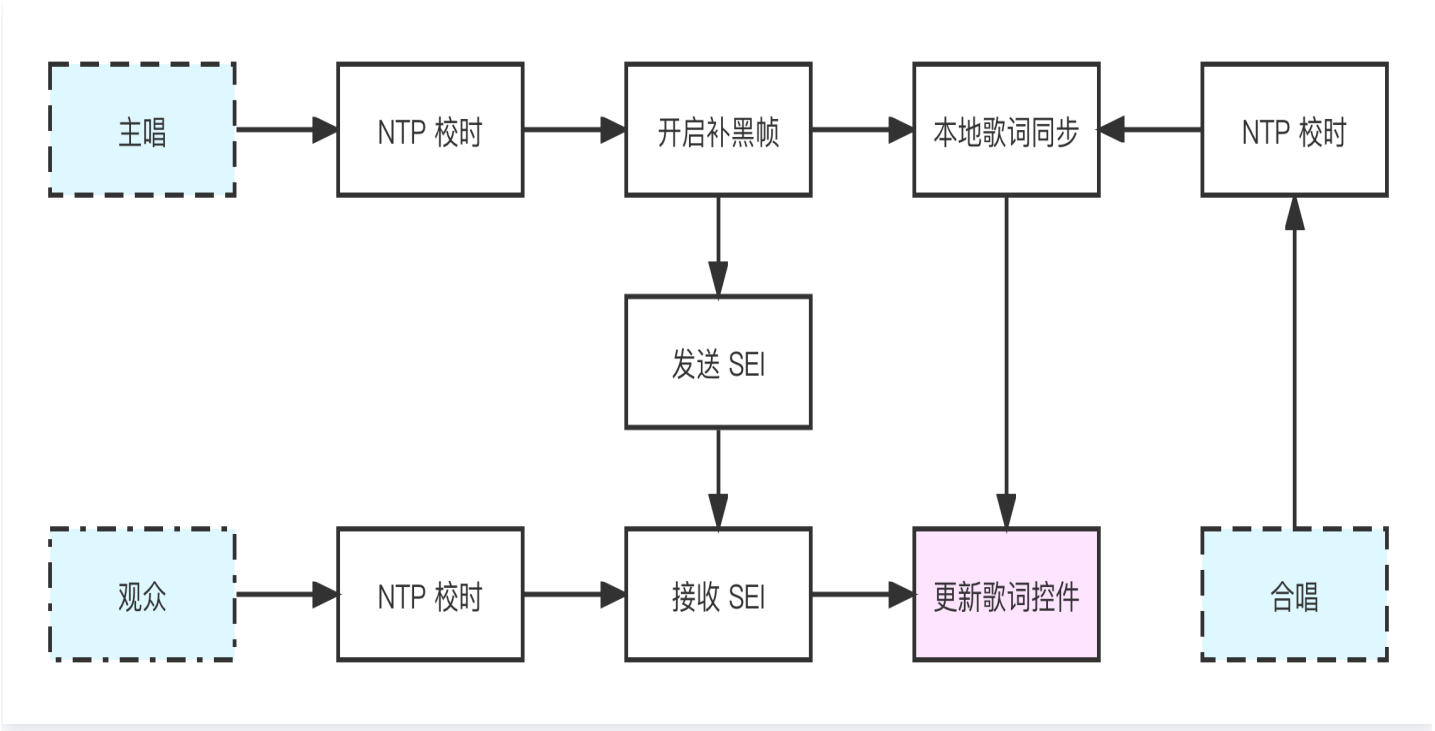
最近更新时间：2023-09-22 14:39:12

实现流程

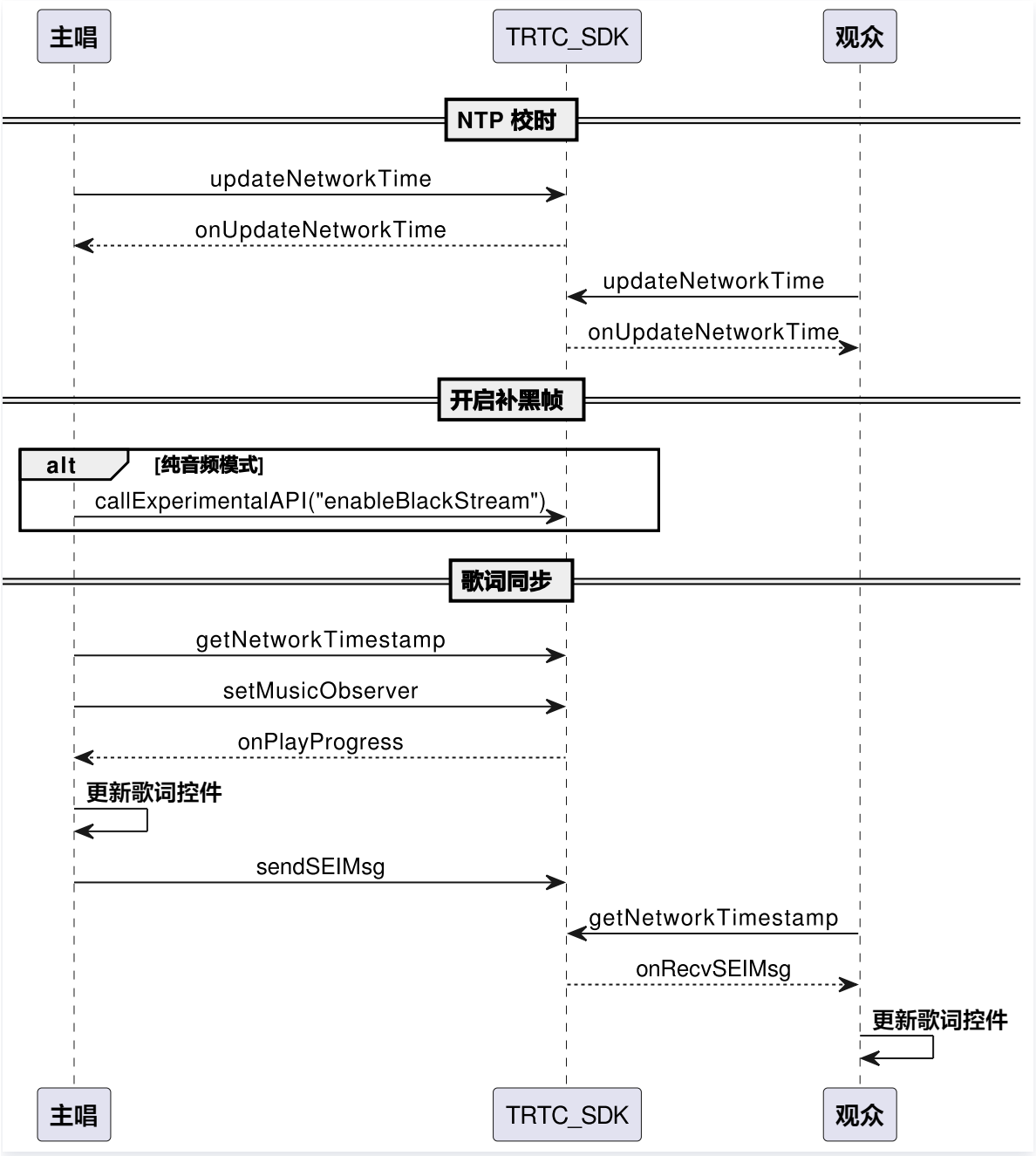
歌词同步方案中，三种不同角色的动作如下：

主唱	合唱	观众
• NTP 校时 • 开启补黑帧 • 发送 SEI 消息 • 本地歌词同步 • 更新歌词控件	• NTP 校时 • 本地歌词同步 • 更新歌词控件	• NTP 校时 • 接收 SEI 消息 • 更新歌词控件

其中，主唱及合唱根据同步后的歌曲播放进度，在本地更新歌词进度；观众端则需要接收由主唱端发送的，包含最新歌词进度的 SEI 消息来更新本地的歌词进度。



时序图



歌词同步时序主要可以分为三个部分：NTP 校时、开启补黑帧、本地及远端歌词同步。NTP 校时的代码实现在歌曲同步中已经给出，下面将针对后两个部分给出具体的代码实现。

关键代码实现

1. 开启补黑帧

```
// 纯音频模式下，主实例（人声实例）需要开启补黑帧以携带 SEI 消息
mTRTCCloud.callExperimentalAPI("
{\"api\":\"enableBlackStream\",\"params\": {\"enable\":true}}");
```

说明：

- 该实验性接口 enableBlackStream 需要在进房之后调用。
- 在 Android 端，enable 参数的值类型为布尔型，在 iOS 端为整型。
- 接收端需要在收到 onUserVideoAvailable(userId, true) 时调用 startRemoteView(userId, null)。

2. 通过 SEI 消息发送歌曲进度

```
mAudioEffectManager.setMusicObserver(mCurPlayMusicId, new
TXAudioEffectManager.TXMusicPlayObserver() {
    @Override
    public void onPlayProgress(int id, long curPtsMS, long durationMS) {
        JSONObject jsonObject = new JSONObject();
        // 当前 ntp 时间
        long ntpTime = TXLiveBase.getNetworkTimestamp();
        jsonObject.put("bgmProgressTime", curTime);
        jsonObject.put("ntpTime", ntpTime);
        jsonObject.put("musicId", musicId);
        jsonObject.put("duration", duration);
        jsonObject.toString().getBytes();
        mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
    }
}
```

说明：

- 主唱发送 SEI 消息的频率由背景音乐播放事件回调的频率决定，一般为 200ms；
- **不直接使用 CMD 消息发送歌曲进度**的原因：SEI 通道传输的信令可以伴随视频帧一直传输到直播 CDN 上，对于拉取 CDN 流的观众具有更好的兼容性。

3. 本地及远端歌词同步

```
// 本地歌词同步
mAudioEffectManager.setMusicObserver(mCurPlayMusicId, new
TXAudioEffectManager.TXMusicPlayObserver() {
    @Override
    public void onPlayProgress(int id, long curPtsMS, long durationMS) {
        ...
        // TODO 更新歌词控件逻辑：
```

```
            // 根据最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
            ...
        }
    }

    // 远端歌词同步
    @Override
    public void onRecvSEIMsg(String userId, byte[] data) {
        String result = new String(data);
        JSONObject jsonObject = new JSONObject(result);
        long bgmProgressTime = jsonObject.getLong("bgmProgressTime");
        long ntpTime = jsonObject.getLong("ntpTime");
        String musicId = jsonObject.getString("musicId");
        long duration = jsonObject.getLong("duration");
        ...
        // TODO 更新歌词控件逻辑：
        // 根据接收到的最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
        ...
    }
```

说明：
如果复用 TUIKaraoke 组件的歌词控件，请参照 TUIKaraoke LyricsView 部分的代码逻辑同步歌词控件进度。

iOS

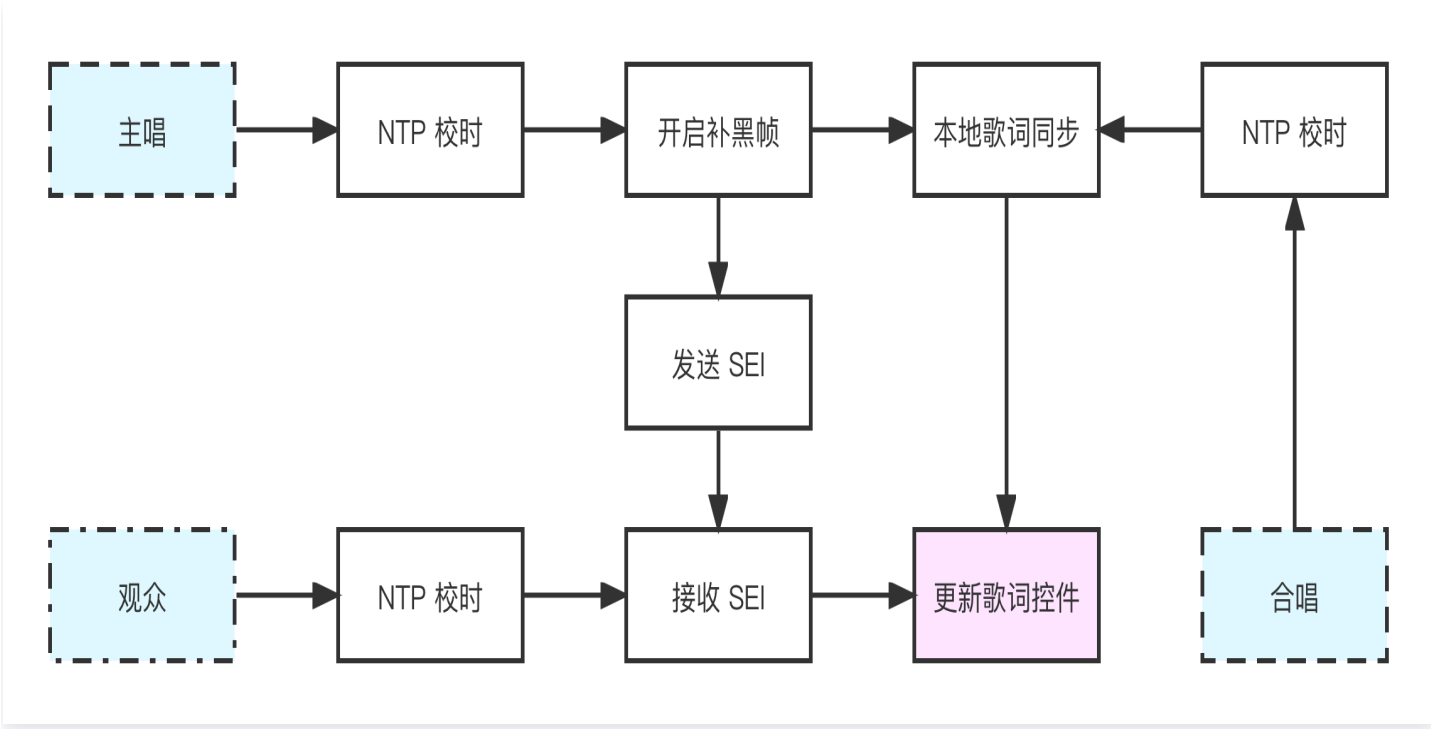
最近更新时间：2022-10-31 16:18:12

1.1 实现流程

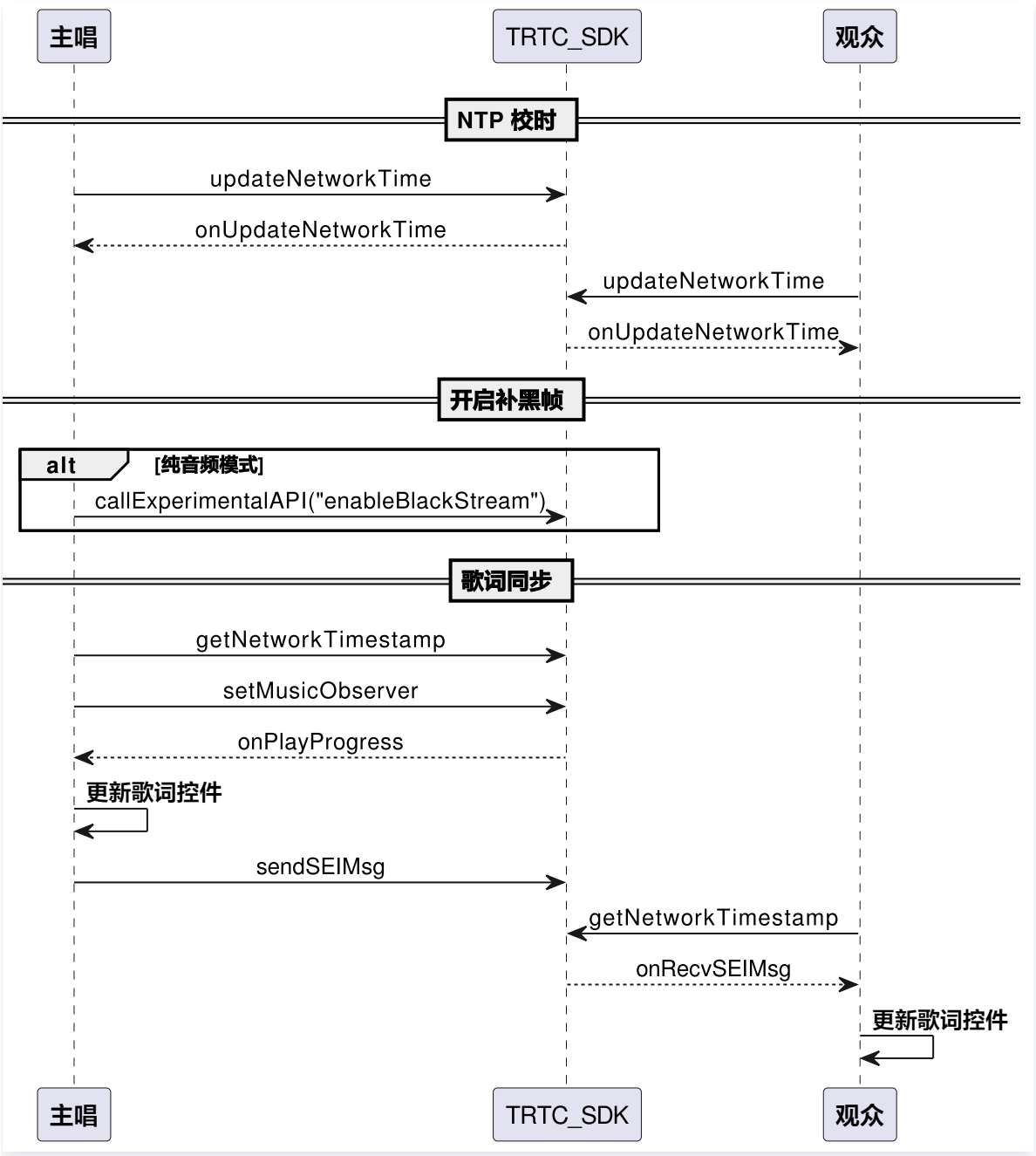
歌词同步方案中，三种不同角色的动作如下：

主唱	合唱	观众
• NTP 校时 • 开启补黑帧 • 发送 SEI 消息 • 本地歌词同步 • 更新歌词控件	• NTP 校时 • 本地歌词同步 • 更新歌词控件	• NTP 校时 • 接收 SEI 消息 • 更新歌词控件

其中，主唱及合唱根据同步后的歌曲播放进度，在本地更新歌词进度；观众端则需要接收由主唱端发送的，包含最新歌词进度的 SEI 消息来更新本地的歌词进度。



时序图



歌词同步时序主要可以分为三个部分：NTP 校时、开启补黑帧、本地及远端歌词同步。NTP 校时的代码实现在歌曲同步中已经给出，下面将针对后两个部分给出具体的代码实现。

关键代码实现

1. 开启补黑帧

```
// 纯音频模式下，主实例（人声实例）需要开启补黑帧以携带 SEI 消息
NSDictionary *jsonDic = @{
                          @"api": @"enableBlackStream",
                          @"params":
                              @{
```

```
                                    @"enable": @(1)
                                }
                            };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[trtcCloud callExperimentalAPI:jsonString];
```

说明：

- 该实验性接口 enableBlackStream 需要在进房之后调用；
- 在 Android 端，enable 参数的值类型为布尔型，在 iOS 端为整型；
- 接收端需要在收到 onUserVideoAvailable(userId, true) 时调用 startRemoteView(userId, null)。

2. 通过 SEI 消息发送歌曲进度

```
TXAudioMusicProgressBlock progressBlock = ^(NSInteger progressMs,
NSInteger durationMs) {
    //当前 ntp 时间
    NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    //通知歌曲进度，用户会在这里进行歌词的滚动
    NSDictionary *progressMsg = @{
            @"bgmProgressTime":@(progressMs),
            @"ntpTime":@(ntpTime),
            @"musicId": @(musicId),
            @"duration": @(durationMs),
    };
    NSData *jsonData = [NSJSONSerialization
dataWithJSONObject:progressMsg options:NSJSONWritingPrettyPrinted
error:nil];
    NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
    [trtcCloud sendSEIMsg:[jsonString
dataUsingEncoding:NSUTF8StringEncoding] repeatCount:1];
};
```

说明：

- 主唱发送 SEI 消息的频率由背景音乐播放事件回调的频率决定，一般为 200ms；

- **不直接使用 CMD 消息发送歌曲进度的原因：SEI 通道传输的信令可以伴随视频帧一直传输到直播 CDN 上，对于拉取 CDN 流的观众具有更好的兼容性。**

3. 本地及远端歌词同步

```
// 本地歌词同步
TXAudioMusicProgressBlock progressBlock = ^(NSInteger progressMs,
NSInteger durationMs) {
    ...
    // TODO 更新歌词控件逻辑：
    // 根据最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
    ...
};

// 远端歌词同步
- (void)onRecvSEIMsg:(NSString *)userId message:(NSData *)message {
    NSError *err = nil;
    NSDictionary *dic = [NSJSONSerialization JSONObjectWithData:message
options:NSJSONReadingMutableContainers error:&err];
    if (err || ![dic isKindOfClass:[NSDictionary class]]) {
        // 解析出错
        return;
    }
    NSInteger bgmProgressTime = [[dic objectForKey:@"bgmProgressTime"]
integerValue];
    NSInteger ntpTime = [[dic objectForKey:@"ntpTime"] integerValue];
    int32_t musicId = [[dic objectForKey:@"musicId"] intValue];
    NSInteger duration = [[dic objectForKey:@"duration"] integerValue];
    ...
    // TODO 更新歌词控件逻辑：
    // 根据接收到的最新进度和本地歌词进度误差，判断是否需要 seek 歌词控件
    ...
}
```

说明：
如果复用 TUIKaraoke 组件的歌词控件，请参照 TUIKaraoke TRTCLyricView 部分的代码逻辑同步歌词控件进度。

人声同步
Android

最近更新时间：2023-09-19 14:58:12

人声与歌曲同步介绍

因为本地人声采集的 jitter buffer、歌曲播放混音的 jitter buffer 以及声音播放到人耳到歌唱存在有一定的 GAP 的，所以演唱者完全对着歌词和 BGM 播放时候，在远端观众感觉 BGM 播放、人声、歌词是有一定的延迟的。合唱方案在 TRTC SDK 内部使用了使用低延迟的 AAudio 采集，具体只需要在进房后开启合唱模式与低延时模式。

具体代码实现

开启合唱模式

```
// 主实例（人声实例）开启合唱模式（调低缓冲区间、音频冗余保护）
mTRTCCloud.callExperimentalAPI("{\"api\":\"enableChorus\",\"params\":
{\"enable\":1,\"audioSource\":0}}");
// 子实例（伴奏实例）开启合唱模式（调低缓冲区间、音频冗余保护）
subCloud.callExperimentalAPI("{\"api\":\"enableChorus\",\"params\":
{\"enable\":1,\"audioSource\":1}}");
```

说明：

开启合唱模式的实验性接口 enableChorus 的参数设置：

- audioSource：0（人声）。
- audioSource：1（伴奏）。

开启低延时模式（高性能音频 AAudio）

```
// 主实例（人声实例）开启高性能音频 AAudio
mTRTCCloud.callExperimentalAPI("
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":1}}");
// 子实例（伴奏实例）开启高性能音频 AAudio
subCloud.callExperimentalAPI("
{\"api\":\"setLowLatencyModeEnabled\",\"params\":{\"enable\":1}}");
```

iOS

最近更新时间：2022-10-31 16:18:12

人声与歌曲同步介绍

因为本地人声采集的 jitter buffer、歌曲播放混音的 jitter buffer 以及声音播放到人耳到歌唱存在有一定的 GAP 的，所以演唱者完全对着歌词和 BGM 播放时候，在远端观众感觉 BGM 播放、人声、歌词是有一定的延迟的，合唱方案在 TRTC SDK 内部使用了使用低延迟的 AAudio 采集，具体只需要在进房后开启合唱模式与低延时模式。

具体代码实现

开启合唱模式

```
// 主实例（人声实例）开启合唱模式（调低缓冲区间、音频冗余保护）
NSDictionary *jsonDic = @{
                            @"api": @"enableChorus",
                            @"params": @{
                                        @"enable": @(YES),
                                        @"audioSource": @(0)
                                       }
                         };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[trtcCloud callExperimentalAPI:jsonString];
// 子实例（伴奏实例）开启合唱模式（调低缓冲区间、音频冗余保护）
NSDictionary *jsonDic = @{
                            @"api": @"enableChorus",
                            @"params": @{
                                        @"enable": @(YES),
                                        @"audioSource": @(1)
                                       }
                         };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[subCloud callExperimentalAPI:jsonString];
```

说明：

开启合唱模式的实验性接口 enableChorus 的参数设置：

- audioSource：0（人声）
- audioSource：1（伴奏）

开启低延时模式（高性能音频 AAudio）

```
// 主实例（人声实例）开启高性能音频 AAudio
NSDictionary *jsonDic = @{
                            @"api": @"setLowLatencyModeEnabled",
                            @"params": @{
                                        @"enable": @(1)
                                       }
                         };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[trtcCloud callExperimentalAPI:jsonString];
// 子实例（伴奏实例）开启高性能音频 AAudio
NSDictionary *jsonDic = @{
                            @"api": @"setLowLatencyModeEnabled",
                            @"params": @{
                                        @"enable": @(1)
                                       }
                         };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[subCloud callExperimentalAPI:jsonString];
```

混流方案
Android

最近更新时间：2022-10-31 16:18:13

具体代码实现

1. 创建主子实例

```
// 创建 TRTCCloud 主实例（人声实例）
TRTCCloud mTRTCCloud =
TRTCCloud.sharedInstance(getApplicationContext());
// 创建 TRTCCloud 子实例（伴奏实例）
TRTCCloud subCloud = mTRTCCloud.createSubCloud();
```

说明：
实时合唱方案中，主唱端需要分别创建主实例-人声实例和子实例-伴奏实例，分别用于上行人声及伴奏音乐。

2. 人声实例进房推流

```
TRTCCloudDef.TRTCParams params = new TRTCCloudDef.TRTCParams();
params.sdkAppId = sdkAppId;
params.userId = mUserId;
params.userSig = userSig;
params.role = TRTCCloudDef.TRTCRoleAnchor;
params.roomId = mRoomId;
mTRTCCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
// 打开音频上行，设置音质
mTRTCCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
// 设置媒体类型
mTRTCCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);
// 静音远端伴奏音乐
mTRTCCloud.muteRemoteAudio(mUserId + "_bgm", true);
```

注意：
- 纯 RTC 音频场景下，进房场景推荐选用 VOICE_CHATROOM。

- 若有视频或转推 CDN 需求，进房场景则须选用 LIVE，VOICE_CHATROOM 会在转推时添加纯音频参数，从而导致 SEI 消息无法透传。
- 主唱/合唱端需要 muteRemoteAudio(true) 取消订阅伴奏实例上行的音频流，否则会重复播放本地及远端的伴奏音乐。

3. 伴奏实例进房推流

```
TRTCCloudDef.TRTCParams bgmParams = new TRTCCloudDef.TRTCParams();
bgmParams.sdkAppId = sdkAppId;
bgmParams.userId = mUserId + "_bgm";
bgmParams.userSig = userSig;
bgmParams.role = TRTCCloudDef.TRTCRoleAnchor;
bgmParams.roomId = mRoomId;
subCloud.enterRoom(bgmParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
//设置媒体类型
subCloud.setSystemVolumeType(TRTCCloudDef.TRTCSystemVolumeTypeMedia);

// 开启预加载
subCloud.callExperimentalAPI("{\"api\":\"preloadMusic\",\"params\":
{\"musicId\":musicId,\"path\":\"path\",\"startTimeMS\":startTimeMS}}");
// 播放伴奏音乐并推流（在约定时间播放）
TXAudioEffectManager.AudioMusicParam param = new
TXAudioEffectManager.AudioMusicParam(musicID, musicPath);
// 将伴奏音乐传到远端
param.publish = true;
subCloud.getAudioEffectManager().startPlayMusic(param);
```

⚠ 注意：

- 注意区分主实例和子实例的 userId，确保不重复且容易辨别；
- 伴奏实例播放背景音乐参数 AudioMusicParam 设置：
  - publish：true（音乐在本地播放的同时，远端用户也能听到该音乐）
  - publish：false（默认值，只能在本地听到该音乐，远端用户听不到）

4. 发起混流转码回推

```
// 创建 TRTCPublishTarget 对象
TRTCCloudDef.TRTCPublishTarget target = new
TRTCCloudDef.TRTCPublishTarget();
// 混流后回推到房间，若发布到 CDN 应填 TRTC_PublishMixStream_ToCdn
```

```
target.mode = TRTCCloudDef.TRTC_PublishMixStream_ToRoom;
target.mixStreamIdentity.intRoomId = Integer.parseInt(mRoomId);
// 混流机器人的 userid，不能和房间内其他用户的 userid 重复
target.mixStreamIdentity.userId = mUserId + "_mix";

// 设置转码后的音频流的编码参数
TRTCCloudDef.TRTCStreamEncoderParam trtcStreamEncoderParam = new
TRTCCloudDef.TRTCStreamEncoderParam();
trtcStreamEncoderParam.audioEncodedChannelNum = 2;
trtcStreamEncoderParam.audioEncodedKbps = 64;
trtcStreamEncoderParam.audioEncodedCodecType = 2;
trtcStreamEncoderParam.audioEncodedSampleRate = 48000;

// 设置音频混流参数
TRTCCloudDef.TRTCStreamMixingConfig trtcStreamMixingConfig = new
TRTCCloudDef.TRTCStreamMixingConfig();
// 支持填写空值，会自动将所有主播的音频混合输出
trtcStreamMixingConfig.audioMixUserList = null;

// 发起混流转推请求
mTRTCCloud.startPublishMediaStream(target, trtcStreamEncoderParam,
trtcStreamMixingConfig);
```

⚠ 注意：

- 优先选择主唱通过混流机器人向后台发起混流转推，将伴奏音乐和各路人声混合后回推至 TRTC 房间，或转推至直播 CDN。
- 自动订阅模式下，参与混流转码的主播默认互相拉取单流，不接收回推房间的混流；观众自动拉取回推房间的混流，不再接收单流。
- 这里的混流转推方法 startPublishMediaStream 采用全新的后台架构，旧版应用需提供 SdkAppId 申请升级后方可使用。

iOS

最近更新时间：2022-10-31 16:18:13

具体代码实现

1. 创建主子实例

```
// 创建 TRTCCloud 主实例（人声实例）
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
// 创建 TRTCCloud 子实例（伴奏实例）
TRTCCloud *subCloud = [trtcCloud createSubCloud];
```

注意：

实时合唱方案中，主唱端需要分别创建主实例-人声实例和子实例-伴奏实例，分别用于上行人声及伴奏音乐。

2. 人声实例进房推流

```
TRTCParams *params = [[TRTCParams alloc] init];
params.sdkAppId = sdkAppId;
params.userId = userId;
params.userSig = userSign;
params.role = TRTCRoleAnchor;
params.roomId = roomIdIntValue;
[trtcCloud enterRoom:params appScene:TRTCAppSceneLIVE];
// 打开音频上行，设置音质
[trtcCloud startLocalAudio:TRTCAudioQualityMusic];
// 设置媒体类型
[trtcCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
// 静音远端伴奏音乐
[trtcCloud muteRemoteAudio:remoteAudioId mute:YES];
```

注意：

- 纯 RTC 音频场景下，进房场景推荐选用 VOICE_CHATROOM。
- 若有视频或转推 CDN 需求，进房场景则须选用 LIVE，VOICE_CHATROOM 会在转推时添加纯音频参数，从而导致 SEI 消息无法透传。

- 主唱/合唱端需要 muteRemoteAudio(true) 取消订阅伴奏实例上行的音频流，否则会重复播放本地及远端的伴奏音乐。

3. 伴奏实例进房推流

```
TRTCParams *bgmParams = [[TRTCParams alloc] init];
bgmParams.sdkAppId = sdkAppId;
bgmParams.userId = [NSString stringWithFormat:@"%@%@",userId,@"_bgm"];
bgmParams.userSig = bgmUserSign;
bgmParams.role = TRTCRoleAnchor;
bgmParams.roomId = roomIdIntValue;
[subCloud enterRoom:bgmParams appScene:TRTCAppSceneLIVE];
//设置媒体类型
[subCloud setSystemVolumeType:TRTCSystemVolumeTypeMedia];
// 开启预加载
NSDictionary *jsonDict = @{
                           @"api": @"preloadMusic",
                           @"params": @{
                                   @"musicId":
@(self.currentPlayMusicID),
                                   @"path": path,
                                   @"startTimeMS": @(startMs),
                                  }
                          };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDict
options:0 error:NULL];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[subCloud callExperimentalAPI:jsonString];
// 播放伴奏音乐并推流（在约定时间播放）
TXAudioMusicParam *musicParam = [[TXAudioMusicParam alloc] init];
musicParam.ID = musicID;
musicParam.path = url;
musicParam.loopCount = 0;
musicParam.publish = YES;
// 将伴奏音乐传到远端
param.publish = YES;
[[subCloud getAudioEffectManager] startPlayMusic:musicParam
onStart:startBlock onProgress:progressBlock onComplete:completedBlock]
```

⚠ 注意：

- 注意区分主实例和子实例的 userId，确保不重复且容易辨别。
- 伴奏实例播放背景音乐参数 musicParam 设置：
  - publish：YES（音乐在本地播放的同时，远端用户也能听到该音乐）
  - publish：NO（默认值，只能在本地听到该音乐，远端用户听不到）

4. 发起混流转码回推

```
// 创建 TRTCPublishTarget 对象
TRTCPublishTarget *publishTarget = [[TRTCPublishTarget alloc] init];
// 混流后回推到房间，若发布到 CDN 应填 TRTCPublishMixStreamToCdn
publishTarget.mode = TRTCPublishMixStreamToRoom;
// 混流机器人的 userid，不能和房间内其他用户的 userid 重复
publishTarget.mixStreamIdentity = [NSString
stringWithFormat:@"%@%@",userId,@"_mix"];

// 设置转码后的音频流的编码参数
TRTCStreamEncoderParam *streamEncoderParam = [[TRTCStreamEncoderParam
alloc] init];
streamEncoderParam.videoEncodedFPS = 15;
streamEncoderParam.videoEncodedGOP = 3;
streamEncoderParam.videoEncodedKbps = 30;
streamEncoderParam.audioEncodedSampleRate = 48000;
streamEncoderParam.audioEncodedChannelNum = 2;
streamEncoderParam.audioEncodedKbps = 64;
streamEncoderParam.audioEncodedCodecType = 2;

// 设置音频混流参数
TRTCStreamMixingConfig *streamMixingConfig = [[TRTCStreamMixingConfig
alloc] init];
// 支持填写空值，会自动将所有主播的音频混合输出
streamMixingConfig.audioMixUserList = @[];

// 发起混流转推请求
[trtcCloud startPublishMediaStream:publishTarget
encoderParam:streamEncoderParam mixingConfig:streamMixingConfig];
```

⚠ 注意：

- 优先选择主唱通过混流机器人向后台发起混流转推，将伴奏音乐和各路人声混合后回推至 TRTC 房间，或转推至直播 CDN。

- 自动订阅模式下，参与混流转码的主播默认互相拉取单流，不接收回推房间的混流；观众自动拉取回推房间的混流，不再接收单流。
- 这里的混流转推方法 startPublishMediaStream 采用全新的后台架构，旧版应用需提供 SdkAppId 申请升级后方可使用。

API 参考（TUIKaraoke）Android

最近更新时间：2022-10-31 16:18:13

TRTCKaraokeRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的组件，支持以下功能：

- 房主创建新的 Karaoke 房间开播，听众进入 Karaoke 房间收听/互动。
- 房主可以管理点歌、将座位上的麦上主播踢下麦。
- 房主还能对座位进行封禁，其他听众就不能再进行申请上麦了。
- 听众可以申请上麦，变成麦上主播，上麦后可以点歌和唱歌，也可以随时下麦成为普通的听众。
- 支持发送礼物和各种文本、自定义消息，自定义消息可用于实现弹幕、点赞等。

> **说明**
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 – 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCKaraokeRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 方案介绍-实现步骤 。

- TRTC SDK：使用 TRTC SDK 作为低延时语音聊天组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现聊天室的功能，同时，通过 IM 的属性接口来存储麦位表等房间信息，邀请信令可以用于上麦申请/抱麦申请。

TRTCKaraokeRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateHandler	设置事件回调所在的线程。
login	登录。

logout	登出。
setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（房主调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（房主调用）。
enterRoom	进入房间（听众调用）。
exitRoom	退出房间（听众调用）。
getRoomInfoList	获取房间列表的详细信息。
getUserInfoList	获取指定 userId 的用户信息，如果为 null，则获取房间内所有人的信息。

音乐播放接口

API	描述
startPlayMusic	开始播放音乐。
stopPlayMusic	停止播放音乐。
pausePlayMusic	暂停播放音乐。
resumePlayMusic	恢复播放音乐。

麦位管理接口

API	描述
enterSeat	主动上麦（听众端和房主均可调用）。
leaveSeat	主动下麦（主播调用）。
pickSeat	抱人上麦（房主调用）。
kickSeat	踢人下麦（房主调用）。
muteSeat	静音/解除静音某个麦位（房主调用）。
closeSeat	封禁/解禁某个麦位（房主调用）。

本地音频操作接口

API	描述
startMicrophone	开启麦克风采集。
stopMicrophone	停止麦克风采集。
setAudioQuality	设置音质。
muteLocalAudio	开启/关闭本地静音。
setSpeaker	设置开启扬声器。
setAudioCaptureVolume	设置麦克风采集音量。
setAudioPlayoutVolume	设置播放音量。
setVoiceEarMonitorEnable	开启/关闭 耳返。

远端用户音频操作接口

API	描述
muteRemoteAudio	静音/解除静音指定成员。
muteAllRemoteAudio	静音/解除静音所有成员。

背景音乐音效相关接口

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager 。

消息发送相关接口

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。

sendRoomCustomMsg	发送自定义文本消息。

邀请信令相关接口

API	描述
sendInvitation	向用户发送邀请。
acceptInvitation	接受邀请。
rejectInvitation	拒绝邀请。
cancelInvitation	取消邀请。

TRTCKaraokeRoomDelegate API 概览

通用事件回调

API	描述
onError	错误回调。
onWarning	警告回调。
onDebugLog	Log 回调。

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	Karaoke 房间信息变更回调。
onUserVolumeUpdate	用户通话音量回调。

麦位变更回调

API	描述
onSeatListChange	全量的麦位列表变化。

onAnchorEnterSeat	有成员上麦（主动上麦/房主抱人上麦）。
onAnchorLeaveSeat	有成员下麦（主动下麦/房主踢人下麦）。
onSeatMute	房主禁麦。
onUserMicrophoneMute	用户麦克风是否静音。
onSeatClose	房主封麦。

听众进出事件回调

API	描述
onAudienceEnter	收到听众进房通知。
onAudienceExit	收到听众退房通知。

消息事件回调

API	描述
onRecvRoomTextMsg	收到文本消息。
onRecvRoomCustomMsg	收到自定义消息。

信令事件回调

API	描述
onReceiveNewInvitation	收到新的邀请请求。
onInviteeAccepted	被邀请人接受邀请。
onInviteeRejected	被邀请人拒绝邀请。
onInvitationCancelled	邀请人取消邀请。

歌曲事件回调

API	描述
onMusicProgressUpdate	歌曲播放进度的回调。
onMusicPrepareToPlay	准备播放音乐的回调。
onMusicCompletePlaying	播放完成音乐的回调。

SDK 基础函数

sharedInstance

获取 TRTCKaraokeRoom 单例对象。

```
 public static synchronized TRTCKaraokeRoom sharedInstance(Context
context);
```

参数如下表所示：

参数	类型	含义
context	Context	Android 上下文，内部会转为 ApplicationContext 用于系统 API 调用

destroySharedInstance

销毁 TRTCKaraokeRoom 单例对象。

> **说明**
> 销毁实例后，外部缓存的 TRTCKaraokeRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。

```
public static void destroySharedInstance();
```

setDelegate

TRTCKaraokeRoom 事件回调，您可以通过 TRTCKaraokeRoomDelegate 获得 TRTCKaraokeRoom 的各种状态通知。

```
public abstract void setDelegate(TRTCKaraokeRoomDelegate delegate);
```

> **说明**
> setDelegate 是 TRTCKaraokeRoom 的代理回调。

setDelegateHandler

设置事件回调所在的线程。

```
public abstract void setDelegateHandler(Handler handler);
```

参数如下表所示：

参数	类型	含义
handler	Handler	TRTCKaraokeRoom 中的各种状态通知，会派发到您指定的 handler 线程。

login

登录。

```
public abstract void login(int sdkAppId,
 String userId, String userSig,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
sdkAppId	int	您可以在实时音视频控制台 > 应用管理 > 应用信息中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。

```
public abstract void logout(TRTCKaraokeRoomCallback.ActionCallback
callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。

```
public abstract void setSelfProfile(String userName, String avatarURL,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
userName	String	昵称。
avatarURL	String	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（房主调用）。

```
public abstract void createRoom(int roomId, TRTCKaraokeRoomDef.RoomParam
roomParam, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个 Karaoke 房间列表，腾讯云暂不提供 Karaoke 房间列表的管理服务，请自行管理您的 Karaoke 房间列表。

roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息。例如房间名称、麦位信息、封面信息等。如果需要麦位管理，必须要填入房间的麦位数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

房主开播的正常调用流程如下：

1. 房主调用 `createRoom` 创建新的 Karaoke 房间，此时传入房间 ID、上麦是否需要房主确认、麦位数等房间属性信息。
2. 房主创建房间成功后，调用 `enterSeat` 进入座位。
3. 房主收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
4. 房主还会收到麦位表有成员进入的 `onAnchorEnterSeat` 的事件通知，此时会自动打开麦克风采集。

destroyRoom

销毁房间（房主调用）。房主在创建房间后，可以调用这个函数来销毁房间。

```
public abstract void destroyRoom(TRTCKaraokeRoomCallback.ActionCallback
callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（听众调用）。

```
public abstract void enterRoom(int roomId,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识。

callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

听众进房收听的正常调用流程如下：

1. 听众向您的服务端获取最新的 Karaoke 房间列表，可能包含多个 Karaoke 房间的 roomId 和房间信息。
2. 听众选择一个 Karaoke 房间，调用 `enterRoom` 并传入房间号即可进入该房间。
3. 进房后会收到组件的 `onRoomInfoChange` 房间属性变化事件通知，此时可以记录房间属性并做相应改变，例如 UI 展示房间名、记录上麦是否需要请求房主同意等。
4. 进房后会收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
5. 进房后还会收到麦位表有主播进入的 `onAnchorEnterSeat` 的事件通知。

exitRoom

退出房间。

```
public abstract void exitRoom(TRTCKaraokeRoomCallback.ActionCallback
callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfoList

获取房间列表的详细信息，其中房间名称、房间封面是房主在创建 `createRoom()` 时通过 roomInfo 设置的。

> **说明**
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
public abstract void getRoomInfoList(List<Integer> roomIdList,
TRTCKaraokeRoomCallback.RoomInfoCallback callback);
```

参数如下表所示：

参数	类型	含义
roomIdList	List<Integer>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getUserInfoList

获取指定userId的用户信息。

```
public abstract void getUserInfoList(List<String> userIdList,
TRTCKaraokeRoomCallback.UserListCallback userlistcallback);
```

参数如下表所示：

参数	类型	含义
userIdList	List<String>	需要获取的用户 ID 列表，如果为 null，则获取房间内所有人的信息。
userlistcallback	UserListCallback	用户详细信息回调。

音乐播放接口

startPlayMusic

播放音乐（上麦后调用）。

> **说明**
> - 播放音乐后，自身会收到 `onMusicPrepareToPlay` 的事件通知。
> - 音乐播放中，房间内所有成员会不断收到 `onMusicProgressUpdate` 的事件通知。
> - 音乐播放完成，自身会收到 `onMusicCompletePlaying` 的事件通知。

```
public abstract void startPlayMusic(int musicID, String originalUrl,
String accompanyUrl);
```

参数如下表所示：

参数	类型	含义
musicID	int	音乐的 ID。
originalUrl	String	原唱音乐的绝对路径。
accompanyUrl	String	伴奏音乐的绝对路径。

调用该接口后会停止上一个正在播放的歌曲。

stopPlayMusic

停止播放音乐（播放音乐时调用）。

> **说明**
> 停止播放后，会收到 `onMusicCompletePlaying` 的事件通知。

```
public abstract void stopPlayMusic();
```

pausePlayMusic

暂停正在播放的音乐（播放音乐时调用）。

> **说明**
> - `onMusicProgressUpdate` 的事件通知会暂停
> - 不会收到 `onMusicCompletePlaying` 的事件通知。

```
public abstract void pausePlayMusic();
```

resumePlayMusic

恢复暂停过的音乐（暂停后调用）。

> **说明**
> 不会收到 `onMusicPrepareToPlay` 的事件通知。

```
public abstract void resumePlayMusic();
```

麦位管理接口

enterSeat

主动上麦（听众端和房主均可调用）。

> **说明**
> 上麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
public abstract void enterSeat(int seatIndex,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要上麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

leaveSeat

主动下麦（主播调用）。

> **说明**
> **下麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。**

```
public abstract void leaveSeat(TRTCKaraokeRoomCallback.ActionCallback
callback);
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

pickSeat

抱人上麦（房主调用）。

> **说明**
> **房主抱人上麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。**

```
public abstract void pickSeat(int seatIndex, String userId,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要抱上麦的麦位序号。
userId	String	用户 ID。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是房主需要听众同意，听众才会上麦的场景，可以先调用 `sendInvitation` 向听众申请，收到 `onInvitationAccept` 后再调用该函数。

kickSeat

踢人下麦（房主调用）。

> **说明**
> 房主踢人下麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
public abstract void kickSeat(int seatIndex,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要踢下麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。

muteSeat

静音/解除静音某个麦位（房主调用）。

> **说明**
> 静音/解除静音某个麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatMute` 的事件通知。

```
public abstract void muteSeat(int seatIndex, boolean isMute,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isMute	boolean	true：静音对应麦位；false：解除静音对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。对应 seatIndex 座位上的主播，会自动调用 muteAudio 进行静音/解禁。

closeSeat

封禁/解禁某个麦位（房主调用）。

> **说明**
> 房主封禁/解禁对应麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatClose` 的事件通知。

```
public abstract void closeSeat(int seatIndex, boolean isClose,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isClose	boolean	true：封禁对应麦位； false：解封对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。封禁对应 seatIndex 座位上的主播，会自动下麦。

本地音频操作接口

startMicrophone

开启麦克风采集。

```
public abstract void startMicrophone();
```

stopMicrophone

停止麦克风采集。

```
public abstract void stopMicrophone();
```

setAudioQuality

设置音质。

```
public abstract void setAudioQuality(int quality);
```

参数如下表所示：

参数	类型	含义
quality	int	音频质量，详情请参见 TRTC SDK 。

muteLocalAudio

静音/取消静音本地的音频。

```
public abstract void muteLocalAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	静音/取消静音，详情请参见 TRTC SDK 。

setSpeaker

设置开启扬声器。

```
public abstract void setSpeaker(boolean useSpeaker);
```

参数如下表所示：

参数	类型	含义
useSpeaker	boolean	true：扬声器；false：听筒。

setAudioCaptureVolume

设置麦克风采集音量。

```
public abstract void setAudioCaptureVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	int	采集音量，0 - 100， 默认100。

setAudioPlayoutVolume

设置播放音量。

```
public abstract void setAudioPlayoutVolume(int volume);
```

参数如下表所示：

参数	类型	含义
volume	int	播放音量，0 - 100， 默认100。

muteRemoteAudio

静音/解除静音指定成员。

```
public abstract void muteRemoteAudio(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	指定的用户 ID。
mute	boolean	true：开启静音；false：关闭静音。

muteAllRemoteAudio

静音/解除静音所有成员。

```
public abstract void muteAllRemoteAudio(boolean mute);
```

参数如下表所示：

参数	类型	含义
mute	boolean	true：开启静音；false：关闭静音。

setVoiceEarMonitorEnable

开启/关闭 耳返。

```
public abstract void setVoiceEarMonitorEnable(boolean enable);
```

参数如下表所示：

参数	类型	含义
enable	boolean	true：开启耳返；false：关闭耳返。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager 。

```
public abstract TXAudioEffectManager getAudioEffectManager();
```

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
public abstract void sendRoomTextMsg(String message,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
public abstract void sendRoomCustomMsg(String cmd, String message,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	ActionCallback	发送结果回调。

邀请信令相关接口

sendInvitation

向用户发送邀请。

```
public abstract String sendInvitation(String cmd, String userId, String
content, TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
cmd	String	业务自定义指令。
userId	String	邀请的用户 ID。
content	String	邀请的内容。
callback	ActionCallback	发送结果回调。

返回值：

返回值	类型	含义
inviteId	String	用于标识此次邀请 ID。

acceptInvitation

接受邀请。

```
public abstract void acceptInvitation(String id,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

rejectInvitation

拒绝邀请。

```
public abstract void rejectInvitation(String id,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

cancelInvitation

取消邀请。

```
public abstract void cancelInvitation(String id,
TRTCKaraokeRoomCallback.ActionCallback callback);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

TRTCKaraokeRoomDelegate 事件回调

通用事件回调

onError

错误回调。

> **说明**
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
void onError(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onWarning

警告回调。

```
void onWarning(int code, String message);
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	警告信息。

onDebugLog

Log 回调。

```
void onDebugLog(String message);
```

参数如下表所示：

参数	类型	含义

message	String	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。房主解散房间时，房间内的所有用户都会收到此通知。

```
void onRoomDestroy(String roomId);
```

参数如下表所示：

参数	类型	含义
roomId	String	房间 ID。

onRoomInfoChange

进房成功后会回调该接口，roomInfo 中的信息在房主创建房间的时候传入。

```
void onRoomInfoChange(TRTCKaraokeRoomDef.RoomInfo roomInfo);
```

参数如下表所示：

参数	类型	含义
roomInfo	RoomInfo	房间信息。

onUserMicrophoneMute

用户麦克风是否静音回调，当用户调用 muteLocalAudio，房间内的其他用户都会收到此通知。

```
void onUserMicrophoneMute(String userId, boolean mute);
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID。
mute	boolean	音量大小，取值：0 - 100。

onUserVolumeUpdate

启用音量大小提示，会通知每个成员的音量大小。

```
void onUserVolumeUpdate(List<TRTCCloudDef.TRTCVolumeInfo> userVolumes,
int totalVolume);
```

参数如下表所示：

参数	类型	含义
userVolumes	List	用户列表。
totalVolume	int	音量大小，取值：0 - 100。

麦位回调

onSeatListChange

全量的麦位列表变化，包含了整个麦位表。

```
void onSeatListChange(List<SeatInfo> seatInfoList);
```

参数如下表所示：

参数	类型	含义
seatInfoList	List<SeatInfo>	全量的麦位列表。

onAnchorEnterSeat

有成员上麦(主动上麦/房主抱人上麦)。

```
void onAnchorEnterSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

参数如下表所示：

参数	类型	含义
index	int	成员上麦的麦位。
user	UserInfo	上麦用户的详细信息。

onAnchorLeaveSeat

有成员下麦(主动下麦/房主踢人下麦)。

```
void onAnchorLeaveSeat(int index, TRTCKaraokeRoomDef.UserInfo user);
```

参数如下表所示：

参数	类型	含义
index	int	下麦的麦位。
user	UserInfo	下麦用户的详细信息。

onSeatMute

房主禁麦。

```
void onSeatMute(int index, boolean isMute);
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isMute	boolean	true：静音麦位； false：解除静音。

onSeatClose

房主封麦。

```
void onSeatClose(int index, boolean isClose);
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isClose	boolean	true：封禁麦位； false： 解禁麦位。

听众进出事件回调

onAudienceEnter

收到听众进房通知。

```
void onAudienceEnter(TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	进房听众信息。

onAudienceExit

收到听众退房通知。

```
void onAudienceExit(TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	退房听众信息。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
void onRecvRoomTextMsg(String message, TRTCKaraokeRoomDef.UserInfo
userInfo);
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
void onRecvRoomCustomMsg(String cmd, String message,
TRTCKaraokeRoomDef.UserInfo userInfo);
```

参数如下表所示：

参数	类型	含义
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

邀请信令事件回调

onReceiveNewInvitation

收到新的邀请请求。

```
void onReceiveNewInvitation(String id, String inviter, String cmd,
String content);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。
cmd	String	业务指定的命令字，由开发者自定义。
content	String	业务指定的内容。

onInviteeAccepted

被邀请者接受邀请。

```
void onInviteeAccepted(String id, String invitee);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInviteeRejected

被邀请者拒绝邀请。

```
void onInviteeRejected(String id, String invitee);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInvitationCancelled

邀请人取消邀请。

```
void onInvitationCancelled(String id, String inviter);
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。

音乐播放状态回调

onMusicPrepareToPlay

准备播放音乐的回调

```
void onMusicPrepareToPlay(int musicID);
```

参数如下表所示：

参数	类型	含义
musicID	int	播放时传入的 musicID。

onMusicProgressUpdate

歌曲播放进度的回调

```
void onMusicProgressUpdate(int musicID, long progress, long total);
```

参数如下表所示：

参数	类型	含义
musicID	int	播放时传入的 musicID。
progress	long	当前播放时间，单位： ms。
total	long	总时间，单位： ms。

onMusicCompletePlaying

播放完成音乐的回调

```
void onMusicCompletePlaying(int musicID);
```

参数如下表所示：

参数	类型	含义
musicID	int	播放时传入的 musicID。

iOS

最近更新时间：2023-08-25 15:47:44

TRTCKaraokeRoom 是基于腾讯云实时音视频（TRTC）和即时通信 IM 服务组合而成的组件，支持以下功能：

- 房主创建新的 Karaoke 房间开播，听众进入 Karaoke 房间收听/互动。
- 房主可以管理点歌、将座位上的麦上主播踢下麦。
- 房主还能对座位进行封禁，其他听众就不能再进行申请上麦了。
- 听众可以申请上麦，变成麦上主播，上麦后可以点歌和唱歌，也可以随时下麦成为普通的听众。
- 支持发送礼物和各种文本、自定义消息，自定义消息可用于实现弹幕、点赞等。

> **说明**
> TUIKit 系列组件同时使用了腾讯云 实时音视频 TRTC 和 即时通信 IM 两个基础 PaaS 服务，开通实时音视频后会同步开通即时通信IM服务。即时通信 IM 服务详细计费规则请参见 即时通信 – 价格说明，TRTC 开通会默认关联开通 IM SDK 的体验版，仅支持100个 DAU。

TRTCKaraokeRoom 是一个开源的 Class，依赖腾讯云的两个闭源 SDK，具体的实现过程请参见 方案介绍-实现步骤 。

- TRTC SDK：使用 TRTC SDK 作为低延时语音聊天组件。
- IM SDK：使用 IM SDK 的 AVChatroom 实现聊天室的功能，同时，通过 IM 的属性接口来存储麦位表等房间信息，邀请信令可以用于上麦申请/抱麦申请。

TRTCKaraokeRoom API 概览

SDK 基础函数

API	描述
sharedInstance	获取单例对象。
destroySharedInstance	销毁单例对象。
setDelegate	设置事件回调。
setDelegateQueue	设置事件回调所在的线程。
login	登录。
logout	登出。

setSelfProfile	修改个人信息。

房间相关接口函数

API	描述
createRoom	创建房间（房主调用），若房间不存在，系统将自动创建一个新房间。
destroyRoom	销毁房间（房主调用）。
enterRoom	进入房间（听众调用）。
exitRoom	退出房间（听众调用）。
getRoomInfoList	获取房间列表的详细信息。
getUserInfoList	获取指定 userId 的用户信息，如果为 nil，则获取房间内所有人的信息。

音乐播放接口

API	描述
startPlayMusic	开始播放音乐。
stopPlayMusic	停止播放音乐。
pausePlayMusic	暂停播放音乐。
resumePlayMusic	恢复播放音乐。

麦位管理接口

API	描述
enterSeat	主动上麦（听众端和房主均可调用）。
leaveSeat	主动下麦（主播调用）。
pickSeat	抱人上麦（房主调用）。
kickSeat	踢人下麦（房主调用）。
muteSeat	静音/解除静音某个麦位（房主调用）。
closeSeat	封禁/解禁某个麦位（房主调用）。

本地音频操作接口

API	描述
startMicrophone	开启麦克风采集。
stopMicrophone	停止麦克风采集。
setAudioQuality	设置音质。
muteLocalAudio	开启/关闭本地静音。
setSpeaker	设置开启扬声器。
setAudioCaptureVolume	设置麦克风采集音量。
setAudioPlayoutVolume	设置播放音量。
setVoiceEarMonitorEnable	开启/关闭 耳返。

远端用户音频操作接口

API	描述
muteRemoteAudio	静音/解除静音指定成员。
muteAllRemoteAudio	静音/解除静音所有成员。

背景音乐音效相关接口

API	描述
getAudioEffectManager	获取背景音乐音效管理对象 TXAudioEffectManager 。

消息发送相关接口

API	描述
sendRoomTextMsg	在房间中广播文本消息，一般用于弹幕聊天。
sendRoomCustomMsg	发送自定义文本消息。

邀请信令相关接口

API	描述

sendInvitation	向用户发送邀请。
acceptInvitation	接受邀请。
rejectInvitation	拒绝邀请。
cancelInvitation	取消邀请。

TRTCKaraokeRoomDelegate API 概览

通用事件回调

API	描述
onError	错误回调。
onWarning	警告回调。
onDebugLog	Log 回调。

房间事件回调

API	描述
onRoomDestroy	房间被销毁的回调。
onRoomInfoChange	语聊房间信息变更回调。
onUserVolumeUpdate	用户通话音量回调。

麦位变更回调

API	描述
onSeatListChange	全量的麦位列表变化。
onAnchorEnterSeat	有成员上麦（主动上麦/房主抱人上麦）。
onAnchorLeaveSeat	有成员下麦（主动下麦/房主踢人下麦）。
onSeatMute	房主禁麦。
onUserMicrophoneMute	用户麦克风是否静音。
onSeatClose	房主封麦。

听众进出事件回调

API	描述
onAudienceEnter	收到听众进房通知。
onAudienceExit	收到听众退房通知。

消息事件回调

API	描述
onRecvRoomTextMsg	收到文本消息。
onRecvRoomCustomMsg	收到自定义消息。

信令事件回调

API	描述
onReceiveNewInvitation	收到新的邀请请求。
onInviteeAccepted	被邀请人接受邀请。
onInviteeRejected	被邀请人拒绝邀请。
onInvitationCancelled	邀请人取消邀请。

歌曲事件回调

API	描述
onMusicProgressUpdate	歌曲播放进度的回调。
onMusicPrepareToPlay	准备播放音乐的回调。
onMusicCompletePlaying	播放完成音乐的回调。

SDK 基础函数

sharedInstance

获取 TRTCKaraokeRoom 单例对象。

```
/**
* 获取 TRTCKaraokeRoom 单例对象
*
* - returns: TRTCKaraokeRoom 实例
* - note: 可以调用 {@link TRTCKaraokeRoom#destroySharedInstance()} 销毁单例
对象
*/
+ (instancetype)sharedInstance NS_SWIFT_NAME(shared());
```

destroySharedInstance

销毁 TRTCKaraokeRoom 单例对象。

> **说明**
> 销毁实例后，外部缓存的 TRTCKaraokeRoom 实例无法再使用，需要重新调用 sharedInstance 获取新实例。

```
/**
* 销毁 TRTCKaraokeRoom 单例对象
*
* - note: 销毁实例后，外部缓存的 TRTCKaraokeRoom 实例不能再使用，需要重新调用
{@link TRTCKaraokeRoom#sharedInstance()} 获取新实例
*/
+ (void)destroySharedInstance NS_SWIFT_NAME(destroyShared());
```

setDelegate

TRTCKaraokeRoom 事件回调，您可以通过 TRTCKaraokeRoomDelegate 获得 TRTCKaraokeRoom 的各种状态通知。

```
/**
* 设置组件回调接口
*
* 您可以通过 TRTCKaraokeRoomDelegate 获得 TRTCKaraokeRoom 的各种状态通知
*
* - parameter delegate 回调接口
* - note: TRTCKaraokeRoom 中的回调事件，默认是在 Main Queue 中回调给您；如果您
需要指定事件回调所在的队列，可使用 {@link
TRTCKaraokeRoom#setDelegateQueue(queue)}
*/
```

```
- (void)setDelegate:(id<TRTCKaraokeRoomDelegate>)delegate
NS_SWIFT_NAME(setDelegate(delegate:));
```

> **说明**
> setDelegate 是 TRTCKaraokeRoom 的代理回调。

setDelegateQueue

设置事件回调所在的线程队列，默认发送到主线程 MainQueue 中。

```
/**
* 设置事件回调所在的队列
*
* - parameter queue 队列，TRTCKaraokeRoom 中的各种状态通知回调，会派发到您指定的
queue。
*/
- (void)setDelegateQueue:(dispatch_queue_t)queue
NS_SWIFT_NAME(setDelegateQueue(queue:));
```

参数如下表所示：

参数	类型	含义
queue	dispatch_queue_t	TRTCKaraokeRoom 中的各种状态通知，会派发到您指定的线程队列里去。

login

登录。

```
- (void)login:(int)sdkAppID
       userId:(NSString *)userId
      userSig:(NSString *)userSig
     callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(login(sdkAppID:userId:userSig:callback:));
```

参数如下表所示：

参数	类型	含义

sdkAppId	int	您可以在实时音视频控制台 >应用管理> 应用信息中查看 SDKAppID。
userId	String	当前用户的 ID，字符串类型，只允许包含英文字母（a-z 和 A-Z）、数字（0-9）、连词符（-）和下划线（_）。
userSig	String	腾讯云设计的一种安全保护签名，获取方式请参见 如何计算及使用 UserSig。
callback	ActionCallback	登录回调，成功时 code 为0。

logout

登出。

```
- (void)logout:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(logout(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	登出回调，成功时 code 为0。

setSelfProfile

修改个人信息。

```
- (void)setSelfProfile:(NSString *)userName avatarURL:(NSString
*)avatarURL callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(setSelfProfile(userName:avatarURL:callback:));
```

参数如下表所示：

参数	类型	含义
userName	String	昵称。
avatarURL	String	头像地址。
callback	ActionCallback	个人信息设置回调，成功时 code 为0。

房间相关接口函数

createRoom

创建房间（房主调用）。

```
- (void)createRoom:(int)roomID roomParam:(RoomParam *)roomParam
callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(createRoom(roomID:roomParam:callback:));
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识，需要由您分配并进行统一管理。多个 roomID 可以汇总成一个语聊房间列表，腾讯云暂不提供语聊房间列表的管理服务，请自行管理您的语聊房间列表。
roomParam	TRTCCreateRoomParam	房间信息，用于房间描述的信息。例如房间名称、麦位信息、封面信息等。如果需要麦位管理，必须要填入房间的麦位数。
callback	ActionCallback	创建房间的结果回调，成功时 code 为0。

房主开播的正常调用流程如下：

1. 房主调用 `createRoom` 创建新的 Karaoke 房间，此时传入房间 ID、上麦是否需要房主确认、麦位数等房间属性信息。
2. 房主创建房间成功后，调用 `enterSeat` 进入座位。
3. 房主收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
4. 房主还会收到麦位表有成员进入的 `onAnchorEnterSeat` 的事件通知，此时会自动打开麦克风采集。

destroyRoom

销毁房间（房主调用）。房主在创建房间后，可以调用这个函数来销毁房间。

```
- (void)destroyRoom:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(destroyRoom(callback:));
```

参数如下表所示：

参数	类型	含义

callback	ActionCallback	销毁房间的结果回调，成功时 code 为0。

enterRoom

进入房间（听众调用）。

```
- (void)enterRoom:(NSInteger)roomID callback:(ActionCallback
_Nullable)callback NS_SWIFT_NAME(enterRoom(roomID:callback:));
```

参数如下表所示：

参数	类型	含义
roomId	int	房间标识。
callback	ActionCallback	进入房间的结果回调，成功时 code 为0。

听众进房收听的正常调用流程如下：

1. 听众向您的服务端获取最新的 Karaoke 房间列表，可能包含多个语聊房间的 roomId 和房间信息。
2. 听众选择一个 Karaoke 房间，调用 `enterRoom` 并传入房间号即可进入该房间。
3. 进房后会收到组件的 `onRoomInfoChange` 房间属性变化事件通知，此时可以记录房间属性并做相应改变，例如 UI 展示房间名、记录上麦是否需要请求房主同意等。
4. 进房后会收到组件的 `onSeatListChange` 麦位表变化事件通知，此时可以将麦位表变化刷新到 UI 界面上。
5. 进房后还会收到麦位表有主播进入的 `onAnchorEnterSeat` 的事件通知。

exitRoom

退出房间。

```
- (void)exitRoom:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(exitRoom(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	退出房间的结果回调，成功时 code 为0。

getRoomInfoList

获取房间列表的详细信息，其中房间名称、房间封面是房主在创建 `createRoom()` 时通过 roomInfo 设置的。

> **说明**
> 如果房间列表和房间信息都由您自行管理，可忽略该函数。

```
- (void)getRoomInfoList:(NSArray<NSNumber *> *)roomIdList callback:
(KaraokeInfoCallback _Nullable)callback
NS_SWIFT_NAME(getRoomInfoList(roomIdList:callback:));
```

参数如下表所示：

参数	类型	含义
roomIdList	List<Integer>	房间号列表。
callback	RoomInfoCallback	房间详细信息回调。

getUserInfoList

获取指定 userId 的用户信息。

```
- (void)getUserInfoList:(NSArray<NSString *> * _Nullable)userIDList
callback:(KaraokeUserListCallback _Nullable)callback
NS_SWIFT_NAME(getUserInfoList(userIDList:callback:));
```

参数如下表所示：

参数	类型	含义
userIdList	List<String>	需要获取的用户 ID 列表，如果为 null，则获取房间内所有人的信息。
userlistcallback	UserListCallback	用户详细信息回调。

音乐播放接口

startPlayMusic

播放音乐（上麦后调用）。

> **说明**
> - 播放音乐后，自身会收到 `onMusicPrepareToPlay` 的事件通知。
> - 音乐播放中，房间内所有成员会不断收到 `onMusicProgressUpdate` 的事件通知。

- 音乐播放完成，自身会收到 `onMusicCompletePlaying` 的事件通知。

```
- (void)startPlayMusic:(int32_t)musicID originalUrl:(NSString
*)originalUrl accompanyUrl:(NSString *)backingUrl
NS_SWIFT_NAME(startPlayMusic(musicID:originalUrl:accompanyUrl:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	音乐的 ID。
originalUrl	String	原唱音乐的绝对路径。
accompanyUrl	String	伴奏音乐的绝对路径。

调用该接口后会停止上一个正在播放的歌曲。

stopPlayMusic

停止播放音乐（播放音乐时调用）。

说明

停止播放后，会收到 `onMusicCompletePlaying` 的事件通知。

```
- (void)stopPlayMusic NS_SWIFT_NAME(stopPlayMusic());
```

pausePlayMusic

暂停正在播放的音乐（播放音乐时调用）。

说明

- `onMusicProgressUpdate` 的事件通知会暂停
- 不会收到 `onMusicCompletePlaying` 的事件通知。

```
- (void)pausePlayMusic NS_SWIFT_NAME(pausePlayMusic());
```

resumePlayMusic

恢复暂停过的音乐（暂停后调用）。

> **说明**
> 不会收到 `onMusicPrepareToPlay` 的事件通知。

```
- (void)resumePlayMusic NS_SWIFT_NAME(resumePlayMusic());
```

麦位管理接口

enterSeat

主动上麦（听众端和房主均可调用）。

> **说明**
> 上麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
- (void)enterSeat:(NSInteger)seatIndex callback:(ActionCallback
_Nullable)callback NS_SWIFT_NAME(enterSeat(seatIndex:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要上麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是听众申请上麦需要房主同意的场景，可以先调用 `sendInvitation` 向房主申请，收到 `onInvitationAccept` 后再调用该函数。

leaveSeat

主动下麦（主播调用）。

> **说明**
> 下麦成功后，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
- (void)leaveSeat:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(leaveSeat(callback:));
```

参数如下表所示：

参数	类型	含义
callback	ActionCallback	操作回调。

pickSeat

抱人上麦（房主调用）。

> **说明**
> 房主抱人上麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorEnterSeat` 的事件通知。

```
- (void)pickSeat:(NSInteger)seatIndex userId:(NSString *)userId
callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(pickSeat(seatIndex:userId:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要抱上麦的麦位序号。
userId	String	用户 ID。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。如果是房主需要听众同意，听众才会上麦的场景，可以先调用 `sendInvitation` 向听众申请，收到 `onInvitationAccept` 后再调用该函数。

kickSeat

踢人下麦（房主调用）。

> **说明**
> 房主踢人下麦，房间内所有成员会收到 `onSeatListChange` 和 `onAnchorLeaveSeat` 的事件通知。

```
- (void)kickSeat:(NSInteger)seatIndex callback:(ActionCallback
_Nullable)callback NS_SWIFT_NAME(kickSeat(seatIndex:callback:));
```

参数如下表所示：

参数	类型	含义

seatIndex	int	需要踢下麦的麦位序号。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。

muteSeat

静音/解除静音某个麦位（房主调用）。

> **说明**
> 静音/解除静音某个麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatMute` 的事件通知。

```
- (void)muteSeat:(NSInteger)seatIndex isMute:(BOOL)isMute callback:
(ActionCallback _Nullable)callback
NS_SWIFT_NAME(muteSeat(seatIndex:isMute:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isMute	boolean	true：静音对应麦位；false：解除静音对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。对应 seatIndex 座位上的主播，会自动调用 muteAudio 进行静音/解禁。

closeSeat

封禁/解禁某个麦位（房主调用）。

> **说明**
> 房主封禁/解禁对应麦位，房间内所有成员会收到 `onSeatListChange` 和 `onSeatClose` 的事件通知。

```
- (void)closeSeat:(NSInteger)seatIndex isClose:(BOOL)isClose callback:
(ActionCallback _Nullable)callback
NS_SWIFT_NAME(closeSeat(seatIndex:isClose:callback:));
```

参数如下表所示：

参数	类型	含义
seatIndex	int	需要操作的麦位序号。
isClose	boolean	true：封禁对应麦位； false：解封对应麦位。
callback	ActionCallback	操作回调。

调用该接口会立即修改麦位表。封禁对应 seatIndex 座位上的主播，会自动下麦。

本地音频操作接口

startMicrophone

开启麦克风采集。

```
- (void)startMicrophone;
```

stopMicrophone

停止麦克风采集。

```
- (void)stopMicrophone;
```

setAudioQuality

设置音质。

```
- (void)setAuidoQuality:(NSInteger)quality
NS_SWIFT_NAME(setAuidoQuality(quality:));
```

参数如下表所示：

参数	类型	含义
quality	int	音频质量，详情请参见 TRTC SDK 。

muteLocalAudio

静音/取消静音本地的音频。

```
- (void)muteLocalAudio:(BOOL)mute NS_SWIFT_NAME(muteLocalAudio(mute:));
```

参数如下表所示：

参数	类型	含义
mute	boolean	静音/取消静音，详情请参见 TRTC SDK 。

setSpeaker

设置开启扬声器。

```
- (void)setSpeaker:(BOOL)userSpeaker
NS_SWIFT_NAME(setSpeaker(userSpeaker:));
```

参数如下表所示：

参数	类型	含义
useSpeaker	boolean	true：扬声器；false：听筒。

setAudioCaptureVolume

设置麦克风采集音量。

```
- (void)setAudioCaptureVolume:(NSInteger)volume
NS_SWIFT_NAME(setAudioCaptureVolume(volume:));
```

参数如下表所示：

参数	类型	含义
volume	int	采集音量，0 – 100， 默认100。

setAudioPlayoutVolume

设置播放音量。

```
- (void)setAudioPlayoutVolume:(NSInteger)volume
NS_SWIFT_NAME(setAudioPlayoutVolume(volume:));
```

参数如下表所示：

参数	类型	含义

volume	int	播放音量，0 – 100， 默认100。

muteRemoteAudio

静音/解除静音指定成员。

```
- (void)muteRemoteAudio:(NSString *)userId mute:(BOOL)mute
NS_SWIFT_NAME(muteRemoteAudio(userId:mute:));
```

参数如下表所示：

参数	类型	含义
userId	String	指定的用户 ID。
mute	boolean	true：开启静音；false：关闭静音。

muteAllRemoteAudio

静音/解除静音所有成员。

```
- (void)muteAllRemoteAudio:(BOOL)isMute
NS_SWIFT_NAME(muteAllRemoteAudio(isMute:));
```

参数如下表所示：

参数	类型	含义
isMute	boolean	true：开启静音；false：关闭静音。

setVoiceEarMonitorEnable

开启/关闭 耳返。

```
- (void)setVoiceEarMonitorEnable:(BOOL)enable
NS_SWIFT_NAME(setVoiceEarMonitor(enable:));
```

参数如下表所示：

参数	类型	含义
enable	boolean	true：开启耳返；false：关闭耳返。

背景音乐音效相关接口函数

getAudioEffectManager

获取背景音乐音效管理对象 TXAudioEffectManager 。

```
- (TXAudioEffectManager * _Nullable)getAudioEffectManager;
```

消息发送相关接口函数

sendRoomTextMsg

在房间中广播文本消息，一般用于弹幕聊天。

```
- (void)sendRoomTextMsg:(NSString *)message callback:(ActionCallback
_Nullable)callback NS_SWIFT_NAME(sendRoomTextMsg(message:callback:));
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
callback	ActionCallback	发送结果回调。

sendRoomCustomMsg

发送自定义文本消息。

```
- (void)sendRoomCustomMsg:(NSString *)cmd message:(NSString *)message
callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(sendRoomCustomMsg(cmd:message:callback:));
```

参数如下表所示：

参数	类型	含义
cmd	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
callback	ActionCallback	发送结果回调。

邀请信令相关接口

sendInvitation

向用户发送邀请。

```
- (NSString *)sendInvitation:(NSString *)cmd
                      userId:(NSString *)userId
                     content:(NSString *)content
                    callback:(ActionCallback _Nullable)callback
NS_SWIFT_NAME(sendInvitation(cmd:userId:content:callback:));
```

参数如下表所示：

参数	类型	含义
cmd	String	业务自定义指令。
userId	String	邀请的用户 ID。
content	String	邀请的内容。
callback	ActionCallback	发送结果回调。

返回值：

返回值	类型	含义
inviteId	String	用于标识此次邀请 ID。

acceptInvitation

接受邀请。

```
- (void)acceptInvitation:(NSString *)identifier callback:(ActionCallback
_Nullable)callback
NS_SWIFT_NAME(acceptInvitation(identifier:callback:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。

callback	ActionCallback	发送结果回调。

rejectInvitation

拒绝邀请。

```
- (void)rejectInvitation:(NSString *)identifier callback:(ActionCallback
_Nullable)callback
NS_SWIFT_NAME(rejectInvitation(identifier:callback:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

cancelInvitation

取消邀请。

```
- (void)cancelInvitation:(NSString *)identifier callback:(ActionCallback
_Nullable)callback
NS_SWIFT_NAME(cancelInvitation(identifier:callback:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
callback	ActionCallback	发送结果回调。

TRTCKaraokeRoomDelegate 事件回调

通用事件回调

onError

错误回调。

> **说明**
> SDK 不可恢复的错误，一定要监听，并分情况给用户适当的界面提示。

```
- (void)onError:(int)code
                message:(NSString*)message
NS_SWIFT_NAME(onError(code:message:));
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	错误信息。

onWarning

警告回调。

```
- (void)onWarning:(int)code
                  message:(NSString *)message
NS_SWIFT_NAME(onWarning(code:message:));
```

参数如下表所示：

参数	类型	含义
code	int	错误码。
message	String	警告信息。

onDebugLog

Log 回调。

```
- (void)onDebugLog:(NSString *)message
NS_SWIFT_NAME(onDebugLog(message:));
```

参数如下表所示：

参数	类型	含义

message	String	日志信息。

房间事件回调

onRoomDestroy

房间被销毁的回调。房主解散房间时，房间内的所有用户都会收到此通知。

```
- (void)onRoomDestroy:(NSString *)message
NS_SWIFT_NAME(onRoomDestroy(message:));
```

参数如下表所示：

参数	类型	含义
message	String	回调信息。

onRoomInfoChange

进房成功后会回调该接口，roomInfo 中的信息在房主创建房间的时候传入。

```
- (void)onRoomInfoChange:(KaraokeInfo *)roomInfo
NS_SWIFT_NAME(onRoomInfoChange(roomInfo:));
```

参数如下表所示：

参数	类型	含义
roomInfo	RoomInfo	房间信息。

onUserMicrophoneMute

用户麦克风是否静音回调，当用户调用muteLocalAudio，房间内的其他用户都会收到此通知。

```
- (void)onUserMicrophoneMute:(NSString *)userId mute:(BOOL)mute
NS_SWIFT_NAME(onUserMicrophoneMute(userId:mute:));
```

参数如下表所示：

参数	类型	含义
userId	String	用户 ID。

mute	boolean	音量大小，取值：0 – 100。

onUserVolumeUpdate

启用音量大小提示，会通知每个成员的音量大小。

```
- (void)onUserVolumeUpdate:(NSArray<TRTCVolumeInfo *> *)userVolumes
totalVolume:(NSInteger)totalVolume
NS_SWIFT_NAME(onUserVolumeUpdate(userVolumes:totalVolume:));
```

参数如下表所示：

参数	类型	含义
userVolumes	List	用户列表。
totalVolume	int	音量大小，取值：0 – 100。

麦位回调

onSeatListChange

全量的麦位列表变化，包含了整个麦位表。

```
- (void)onSeatInfoChange:(NSArray<KaraokeSeatInfo *> *)seatInfolist
NS_SWIFT_NAME(onSeatListChange(seatInfoList:));
```

参数如下表所示：

参数	类型	含义
seatInfoList	List<SeatInfo>	全量的麦位列表。

onAnchorEnterSeat

有成员上麦(主动上麦/房主抱人上麦)。

```
- (void)onAnchorEnterSeat:(NSInteger)index
                     user:(KaraokeUserInfo *)user
NS_SWIFT_NAME(onAnchorEnterSeat(index:user:));
```

参数如下表所示：

参数	类型	含义
index	int	成员上麦的麦位。
user	UserInfo	上麦用户的详细信息。

onAnchorLeaveSeat

有成员下麦(主动下麦/房主踢人下麦)。

```
- (void)onAnchorLeaveSeat:(NSInteger)index
                     user:(KaraokeUserInfo *)user
NS_SWIFT_NAME(onAnchorLeaveSeat(index:user:));
```

参数如下表所示：

参数	类型	含义
index	int	下麦的麦位。
user	UserInfo	上麦用户的详细信息。

onSeatMute

房主禁麦。

```
- (void)onSeatMute:(NSInteger)index
            isMute:(BOOL)isMute
NS_SWIFT_NAME(onSeatMute(index:isMute:));
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isMute	boolean	true：静音麦位； false：解除静音。

onSeatClose

房主封麦。

```
- (void)onSeatClose:(NSInteger)index
            isClose:(BOOL)isClose
```

```
NS_SWIFT_NAME(onSeatClose(index:isClose:));
```

参数如下表所示：

参数	类型	含义
index	int	操作的麦位。
isClose	boolean	true：封禁麦位； false： 解禁麦位。

听众进出事件回调

onAudienceEnter

收到听众进房通知。

```
- (void)onAudienceEnter:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceEnter(userInfo:));
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	进房听众信息。

onAudienceExit

收到听众退房通知。

```
- (void)onAudienceExit:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onAudienceExit(userInfo:));
```

参数如下表所示：

参数	类型	含义
userInfo	UserInfo	退房听众信息。

消息事件回调

onRecvRoomTextMsg

收到文本消息。

```
- (void)onRecvRoomTextMsg:(NSString *)message
                 userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomTextMsg(message:userInfo:));
```

参数如下表所示：

参数	类型	含义
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

onRecvRoomCustomMsg

收到自定义消息。

```
- (void)onRecvRoomCustomMsg:(NSString *)cmd
                    message:(NSString *)message
                   userInfo:(KaraokeUserInfo *)userInfo
NS_SWIFT_NAME(onRecvRoomCustomMsg(cmd:message:userInfo:));
```

参数如下表所示：

参数	类型	含义
command	String	命令字，由开发者自定义，主要用于区分不同消息类型。
message	String	文本消息。
userInfo	UserInfo	发送者用户信息。

邀请信令事件回调

onReceiveNewInvitation

收到新的邀请请求。

```
- (void)onReceiveNewInvitation:(NSString *)identifier
                       inviter:(NSString *)inviter
                           cmd:(NSString *)cmd
                       content:(NSString *)content
NS_SWIFT_NAME(onReceiveNewInvitation(identifier:inviter:cmd:content:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
inviter	String	邀请人的用户 ID。
cmd	String	业务指定的命令字，由开发者自定义。
content	UserInfo	业务指定的内容。

onInviteeAccepted

被邀请者接受邀请。

```
- (void)onInviteeAccepted:(NSString *)identifier
                  invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeAccepted(identifier:invitee:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInviteeRejected

被邀请者拒绝邀请。

```
- (void)onInviteeRejected:(NSString *)identifier
                  invitee:(NSString *)invitee
NS_SWIFT_NAME(onInviteeRejected(identifier:invitee:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	被邀请人的用户 ID。

onInvitationCancelled

邀请人取消邀请。

```
- (void)onInvitationCancelled:(NSString *)identifier
                      invitee:(NSString *)invitee
NS_SWIFT_NAME(onInvitationCancelled(identifier:invitee:));
```

参数如下表所示：

参数	类型	含义
id	String	邀请 ID。
invitee	String	邀请人的用户 ID。

音乐播放状态回调

onMusicPrepareToPlay

准备播放音乐的回调

```
- (void)onMusicPrepareToPlay:(int32_t)musicID
NS_SWIFT_NAME(onMusicPrepareToPlay(musicID:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	播放时传入的 musicID。

onMusicProgressUpdate

歌曲播放进度的回调

```
- (void)onMusicProgressUpdate:(int32_t)musicID
                     progress:(NSInteger)progress total:(NSInteger)total
NS_SWIFT_NAME(onMusicProgressUpdate(musicID:progress:total:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	播放时传入的 musicID。

progress	NSInteger	当前播放时间，单位： ms。
total	NSInteger	总时间，单位： ms。

onMusicCompletePlaying

播放完成音乐的回调

```
- (void)onMusicCompletePlaying:(int32_t)musicID
NS_SWIFT_NAME(onMusicCompletePlaying(musicID:));
```

参数如下表所示：

参数	类型	含义
musicID	int32_t	播放时传入的 musicID。

常见问题（TUIKaraoke）Android

最近更新时间：2024-11-18 15:52:52

耳返相关问题

K 歌场景大概率会用到耳返，如何开启耳返功能?

```
mTRTCCloud.getAudioEffectManager().enableVoiceEarMonitor(true)
```

开启耳返功能后没有效果?

由于蓝牙耳机的硬件延迟非常高，请尽量在用户界面上提示主播佩戴有线耳机。 同时也需要注意，并非所有的手机开启此特效后都能达到优秀的耳返效果，TRTC SDK 已经对部分耳返效果不佳的手机屏蔽了该特效。

耳返延迟过高?

请检查是否使用的是蓝牙耳机，由于蓝牙耳机的硬件延迟非常高，请尽量使用有线耳机。
另外，可以尝试通过实验性接口 setSystemAudioKitEnabled 开启硬件耳返来改善耳返延迟过高的问题。目前，对于华为和 Vivo设备，SDK 默认使用硬件耳返，其他设备默认使用软件耳返。

```
// 开启硬件耳返
mTRTCCloud.callExperimentalAPI("{\"api\":\"setSystemAudioKitEnabled\",
\"params\": {\"enable\":1}}");
```

NTP 校时问题

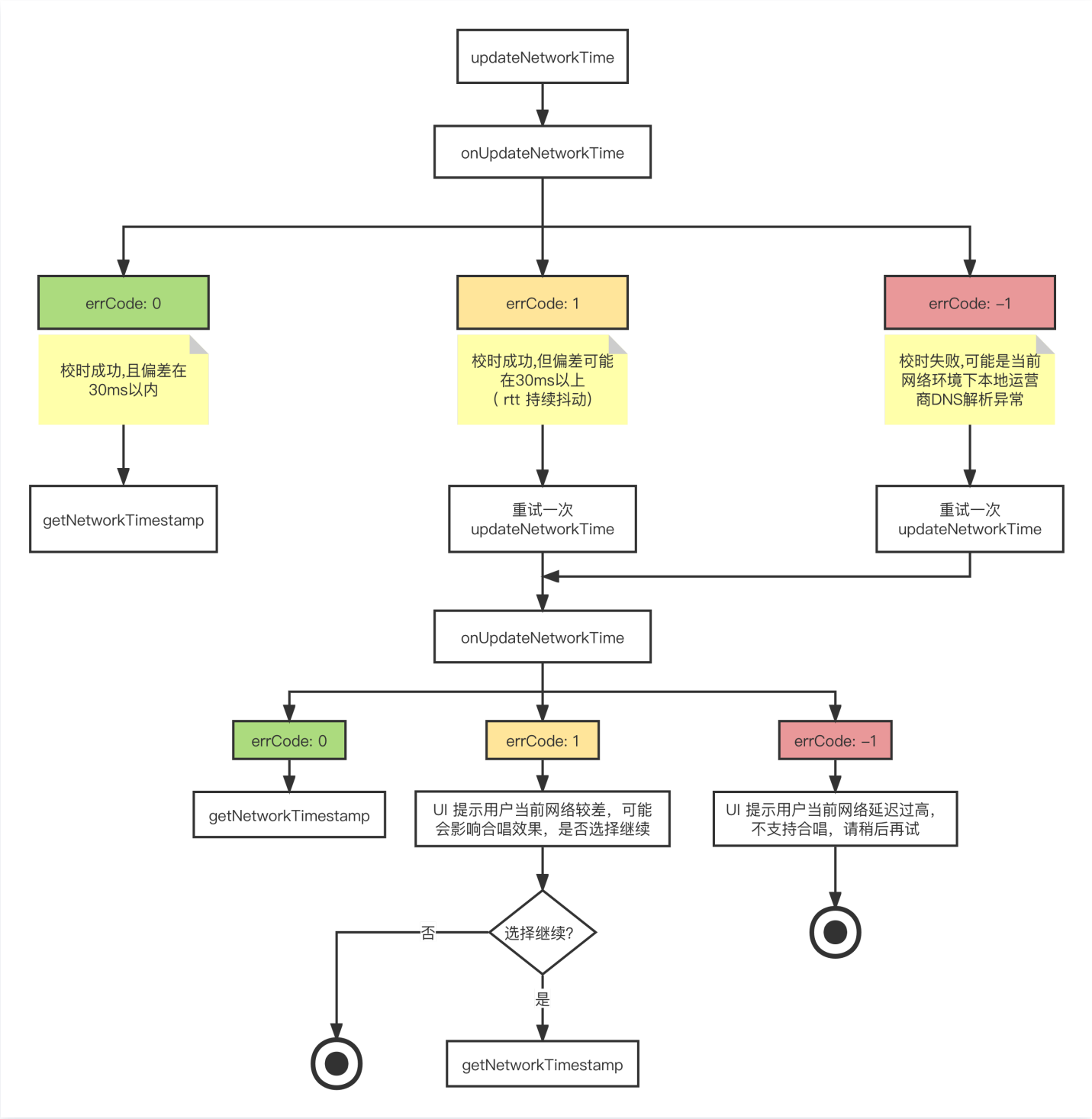
提醒：“NTP time sync finished, but result maybe inaccurate”？

NTP 校时成功，但偏差可能在30ms以上，反映客户端网络环境差，rtt 持续抖动。

提醒：“Error in AddressResolver: No address associated with hostname”？

NTP 校时失败，可能是当前网络环境下本地运营商 DNS 解析暂时异常，请稍后再试。

NTP 服务重试处理逻辑?



网络测速建议

在线 K 歌场景对用户的网络条件要求较高，特别是实时合唱，优质稳定的网络环境才能保障良好的 K 歌体验。因此，建议在用户进入房间前对该用户进行一次网络测速，对网络条件不符合要求的用户给予 UI 层的提醒，禁止其加入 K 歌房或参与合唱。

TRTC SDK 网络测速的发起：

```
TRTCCloudDef.TRTCSpeedTestParams speedTestParams = new
TRTCCloudDef.TRTCSpeedTestParams();
speedTestParams.sdkAppId = SDK_APP_ID;
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// 若实际带宽高于预期值，则测试结果即为预期值；若实际带宽低于预期值，则测试结果为实际
带宽值
speedTestParams.expectedDownBandwidth = 3000; // 预期的下行带宽，取值范围
10~5000 kbps
speedTestParams.expectedUpBandwidth = 3000; // 预期的上行带宽，取值范围
10~5000 kbps
mTRTCCloud.startSpeedTest(speedTestParams);
```

⚠ 注意：

- 同一时间只允许一项网速测试任务运行；
- 请在进入房间前进行网速测试，在房间中网速测试会影响正常的音视频传输效果，而且由于干扰过多，网速测试结果也不准确。

TRTC SDK 网络测速结果的回调：

```
@Override
public void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result) {
    String tquality = "未定义";
    switch (result.quality) {
        case 0: tquality = "未定义";
            break;
        case 1: tquality = "当前网络非常好";
            break;
        case 2: tquality = "当前网络比较好";
            break;
        case 3: tquality = "当前网络一般";
            break;
        case 4: tquality = "当前网络较差";
            break;
        case 5: tquality = "当前网络很差";
            break;
        case 6: tquality = "当前网络不满足 TRTC 的最低要求";
            break;
    }
    if (result.success) {
        mTextTestResult.append("测速成功！ " + "\n");
```

```
            mTextTestResult.append("IP 地址: " + result.ip + "\n");
            mTextTestResult.append("上行丢包率: " + result.upLostRate + "\n");
            mTextTestResult.append("下行丢包率: " + result.downLostRate +
"\n");
            mTextTestResult.append("网络延迟: " + result.rtt + "ms\n");
            mTextTestResult.append("下行带宽: " +
result.availableDownBandwidth + "kbps\n");
            mTextTestResult.append("上行带宽: " + result.availableUpBandwidth
+ "kbps\n");
            mTextTestResult.append("网络质量: " + tquality + "\n");
        } else {
            mTextTestResult.append("测速失败! " + "\n");
            mTextTestResult.append("errMsg: " + result.errMsg + "\n");
        }
    }
```

中途加入合唱

实时合唱方案理论上对合唱者数量没有限制，支持多人同时参与合唱，也支持中途加入合唱。
下面列出中途加入合唱的关键动作：

- NTP 校时。
- 开启合唱实验性接口。
- 进入房间并上麦推流。
- 接收合唱信令，获取伴奏资源及合唱约定时间。
- 计算约定时间与当前时间差，预加载 seek 伴奏。
- 开始参与合唱，并实时同步伴奏进度及歌词进度。

以上关键动作涉及的具体实现流程及代码实现详见 歌曲同步 、歌词同步 、人声同步 文档。

iOS

最近更新时间：2024-11-18 15:52:52

耳返相关问题

K 歌场景大概率会用到耳返，如何开启耳返功能?

```
[[trtcCloud getAudioEffectManager] enableVoiceEarMonitor:YES];
```

开启耳返功能后没有效果?

由于蓝牙耳机的硬件延迟非常高，请尽量在用户界面上提示主播佩戴有线耳机。 同时也需要注意，并非所有的手机开启此特效后都能达到优秀的耳返效果，TRTC SDK 已经对部分耳返效果不佳的手机屏蔽了该特效。

耳返延迟过高?

请检查是否使用的是蓝牙耳机，由于蓝牙耳机的硬件延迟非常高，请尽量使用有线耳机。
另外，可以尝试通过实验性接口 setSystemAudioKitEnabled 开启硬件耳返来改善耳返延迟过高的问题。目前，对于华为和 Vivo设备，SDK 默认使用硬件耳返，其他设备默认使用软件耳返。

```
// 开启硬件耳返
NSDictionary *jsonDic = @{
                            @"api": @"setSystemAudioKitEnabled",
                            @"params": @{@"enable": @(1)}
                        };
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:jsonDic
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
[trtcCloud callExperimentalAPI:jsonString];
```

NTP 校时问题

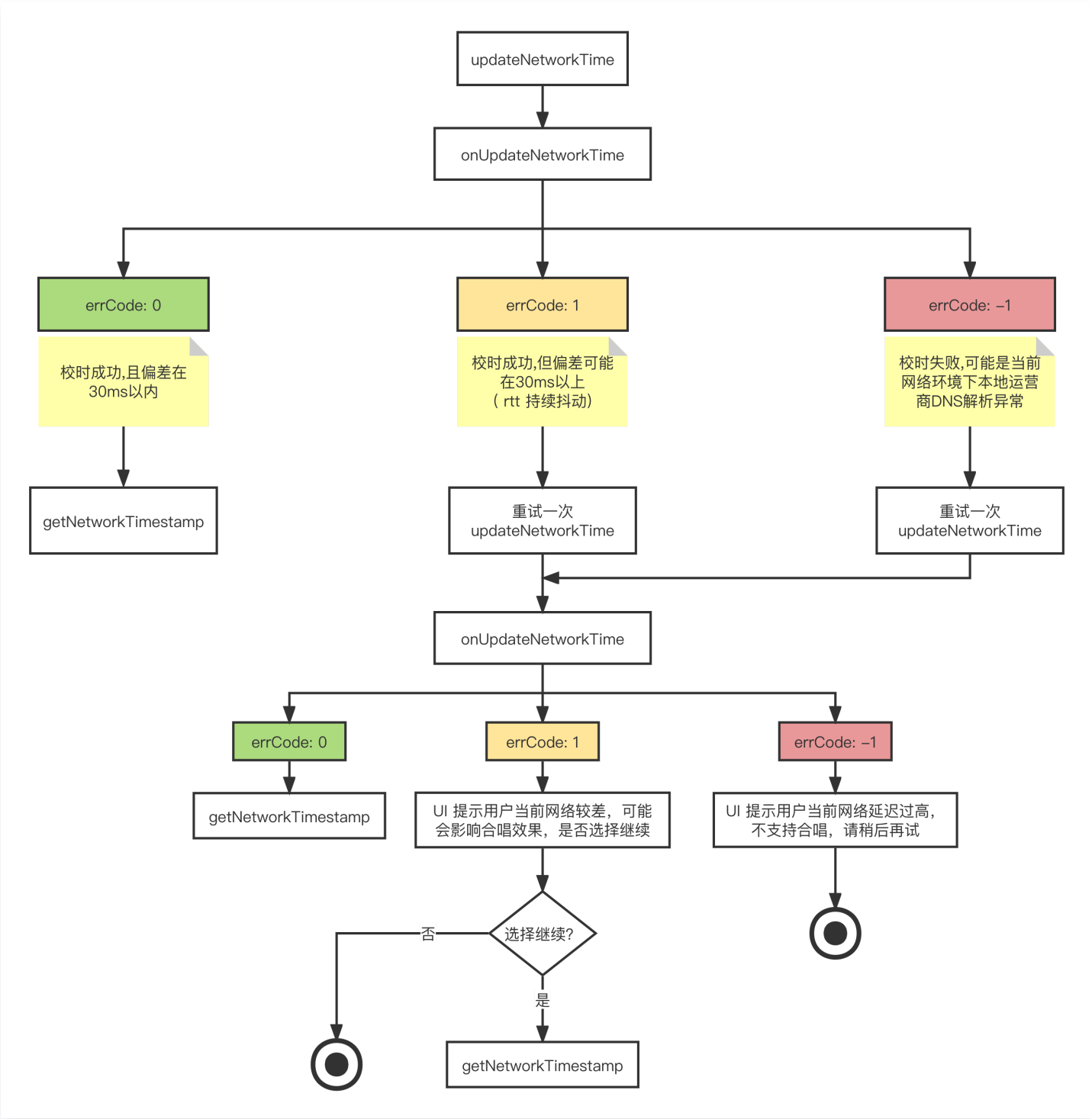
提醒：“NTP time sync finished, but result maybe inaccurate”？

NTP 校时成功，但偏差可能在30ms以上，反映客户端网络环境差，rtt 持续抖动。

提醒：“Error in AddressResolver: No address associated with hostname”？

NTP 校时失败，可能是当前网络环境下本地运营商 DNS 解析暂时异常，请稍后再试。

NTP 服务重试处理逻辑?



网络测速建议

在线 K 歌场景对用户的网络条件要求较高，特别是实时合唱，优质稳定的网络环境才能保障良好的 K 歌体验。因此，建议在用户进入房间前对该用户进行一次网络测速，对网络条件不符合要求的用户给予 UI 层的提醒，禁止其加入 K 歌房或参与合唱。

TRTC SDK 网络测速的发起：

```
TRTCSpeedTestParams *speedTestParams = [[TRTCSpeedTestParams alloc]
init];
speedTestParams.sdkAppId = SDK_APP_ID;
speedTestParams.userId = userId;
speedTestParams.userSig = userSig;
// 若实际带宽高于预期值，则测试结果即为预期值；若实际带宽低于预期值，则测试结果为实际
带宽值
speedTestParams.expectedDownBandwidth = 3000; // 预期的下行带宽，取值范围
10~5000 kbps
speedTestParams.expectedUpBandwidth = 3000; // 预期的上行带宽，取值范围
10~5000 kbps
[trtcCloud startSpeedTest:speedTestParams];
```

⚠ 注意：

- 同一时间只允许一项网速测试任务运行；
- 请在进入房间前进行网速测试，在房间中网速测试会影响正常的音视频传输效果，而且由于干扰过多，网速测试结果也不准确。

TRTC SDK 网络测速结果的回调：

```
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
    NSString *tquality = @"未定义";
    switch (result.quality) {
        case TRTCQuality_Unknown:
            tquality = @"未定义";
            break;
        case TRTCQuality_Excellent:
            tquality = @"当前网络非常好";
            break;
        case TRTCQuality_Good:
            tquality = @"当前网络比较好";
            break;
        case TRTCQuality_Poor:
            tquality = @"当前网络一般";
            break;
        case TRTCQuality_Bad:
```

```
            tquality = @"当前网络较差";
            break;
        case TRTCQuality_Vbad:
            tquality = @"当前网络很差";
            break;
        case TRTCQuality_Down:
            tquality = @"当前网络不满足 TRTC 的最低要求";
            break;
        default:
            break;
    }

    if (result.success) {
        [mTextTestResult addObject:@"测速成功！\n"];
        [mTextTestResult addObject:[NSString stringWithFormat:@"IP 地址:
%@ \n", result.ip]];
        [mTextTestResult addObject:[NSString stringWithFormat:@"上行丢包
率: %f \n", result.upLostRate]];
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行丢包
率: %f \n", result.downLostRate]];
        [mTextTestResult addObject:[NSString stringWithFormat:@"网络延迟:
%u ms \n", result.rtt]];
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行带宽:
%ld kbps \n", (long)result.availableDownBandwidth]];
        [mTextTestResult addObject:[NSString stringWithFormat:@"上行带宽:
%ld kbps \n", result.availableUpBandwidth]];
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行带宽:
%@ \n", tquality]];
    } else {
        [mTextTestResult addObject:@"测速成功！\n"];
        [mTextTestResult addObject:[NSString stringWithFormat:@"errMsg:
%@ \n", result.errMsg]];
    }
}
```

中途加入合唱

实时合唱方案理论上对合唱者数量没有限制，支持多人同时参与合唱，也支持中途加入合唱。

下面列出中途加入合唱的关键动作：

- NTP 校时。
- 开启合唱实验性接口。
- 进入房间并上麦推流。

- 接收合唱信令，获取伴奏资源及合唱约定时间。
- 计算约定时间与当前时间差，预加载 seek 伴奏。
- 开始参与合唱，并实时同步伴奏进度及歌词进度。

以上关键动作涉及的具体实现流程及代码实现详见 歌曲同步 、歌词同步 、人声同步 文档。