实时音视频

解决方案

产品文档

腾讯云

文档目录

解决方案
实时合唱
快速集成（TUIKaraoke）
iOS

最近更新时间：2022-11-10 11:57:41

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 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。

组件集成

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：

- 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/"
```

- 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>
```

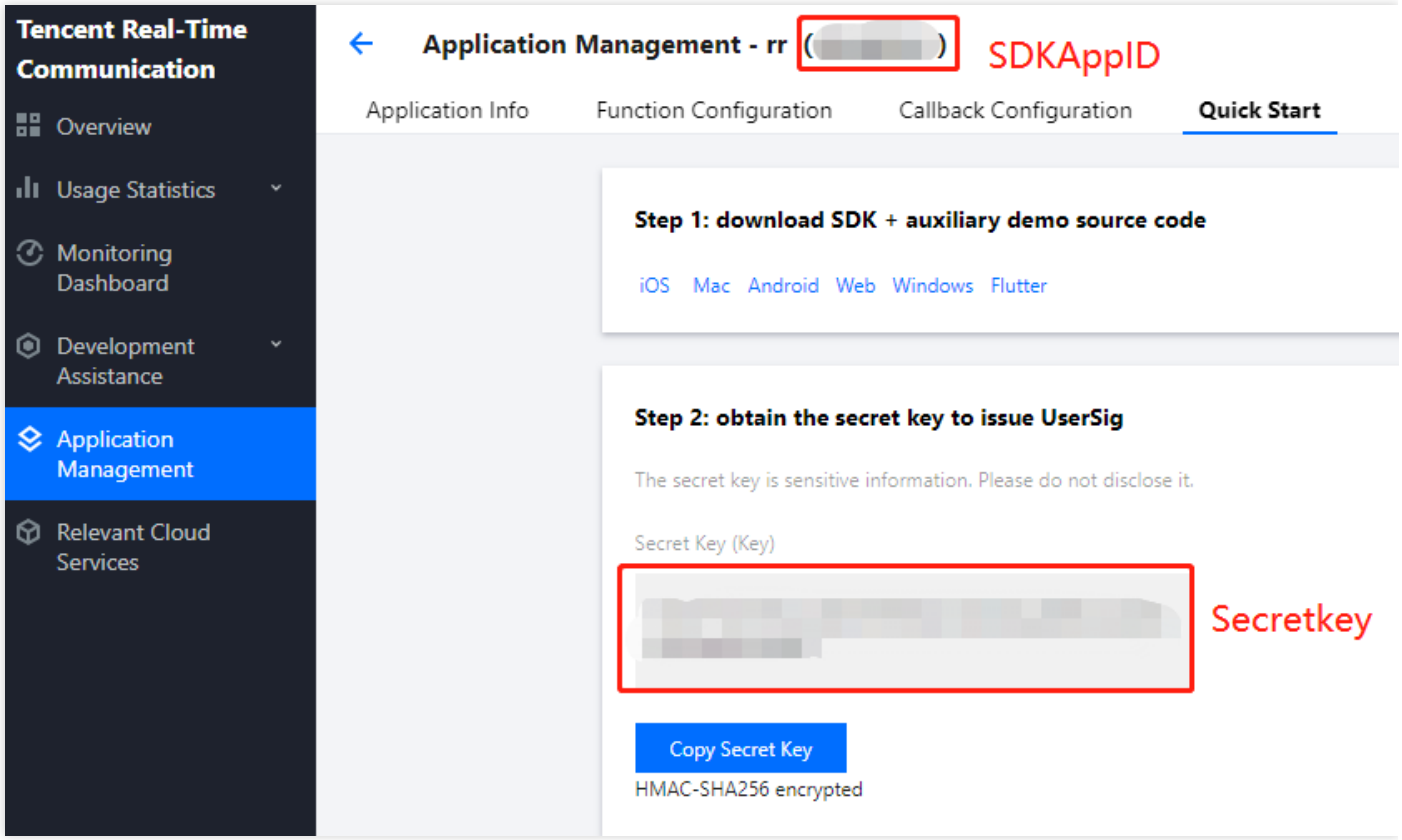
步骤三：初始化并登录

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: ProfileMana
ger.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，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 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。详情请参见 TUIKaraoke 音乐播放接口。

```
//播放音乐
karaokeRoom.startPlayMusic(musicID: musicID, originalUrl: muscicLocalPath, acco
mpanyUrl: 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)
}
}
```

常见问题

TUIKaraoke 组件支持变声、变调、混响等音效功能吗？

支持，具体请参见 TUIKaraoke 示例工程。

？如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

Android

最近更新时间：2022-11-10 11:55:52

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 iOS 平台的源代码，基本功能如下图所示：

组件集成

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：

- 在 `setting.gradle` 中完成导入，参考如下：

```
include ':Source'
include ':Debug'
```

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

```
api project(':Source')
```

- 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_TRTCl: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

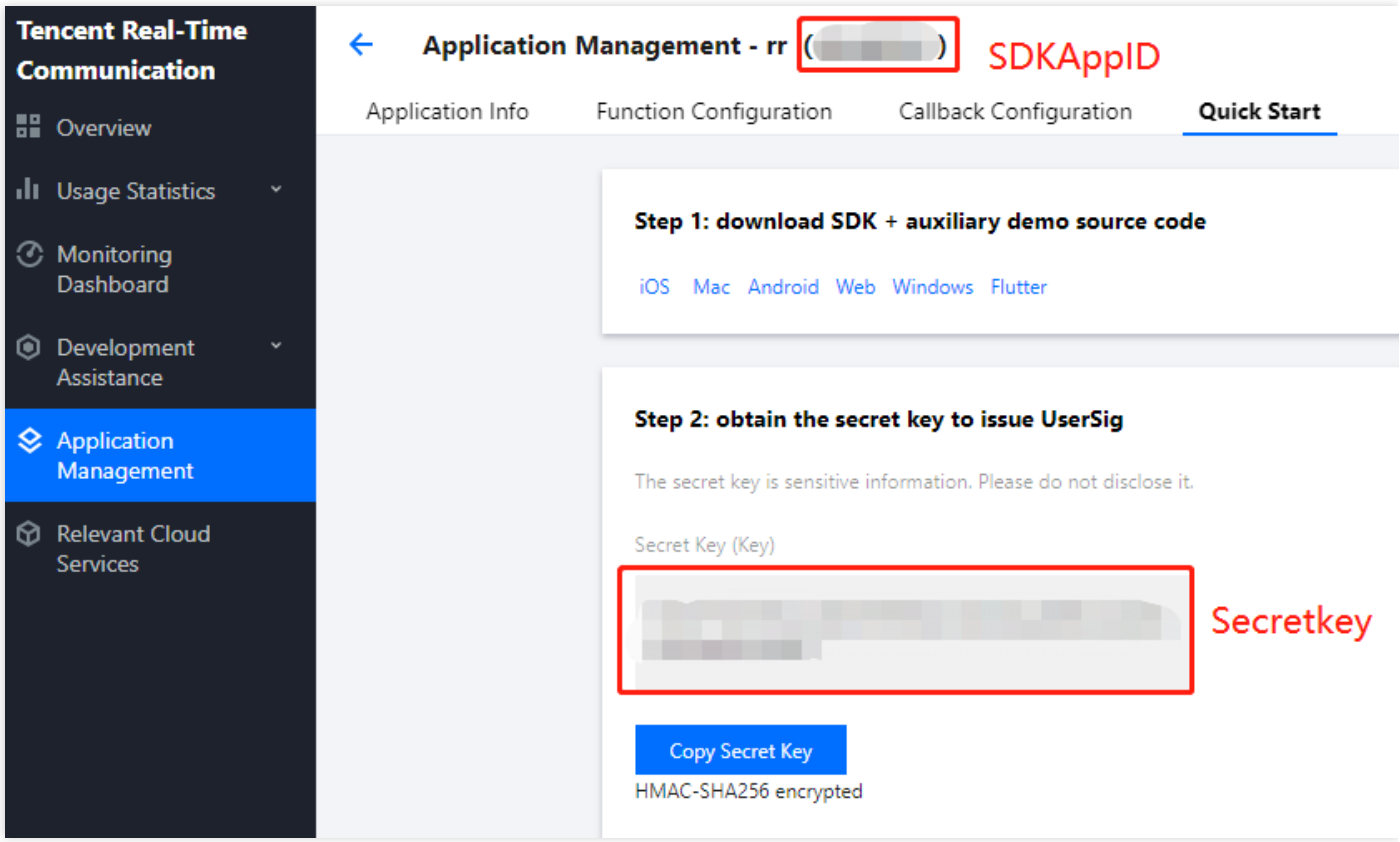
相关接口详情请参见 TUIKaraoke。

```
// 1.初始化
TRTCKaraokeRoom mTRTCKaraokeRoom = TRTCKaraokeRoom.sharedInstance(this);
mTRTCKaraokeRoom.setDelegate(this);
```

```
// 2.登录
mTRTCKaraokeRoom.login(SDKAppID, UserID, UserSig, new TRTCKaraokeRoomCallback.Act
ionCallback() {
@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，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.Acti
onCallback() {
@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.ActionCallbac
k() {
@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> seatInfoLi
st) {
}
```

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.Actio
nCallback() {
@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 userInf
o) {
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.A
ctionCallback() {
@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.Us
erInfo 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);
}
}
});
```

常见问题

TUIKaraoke 组件支持变声、变调、混响等音效功能吗？

支持。

> 说明：
> 如果有任何需要或者反馈，您可以联系：colleenyu@tencent.com。

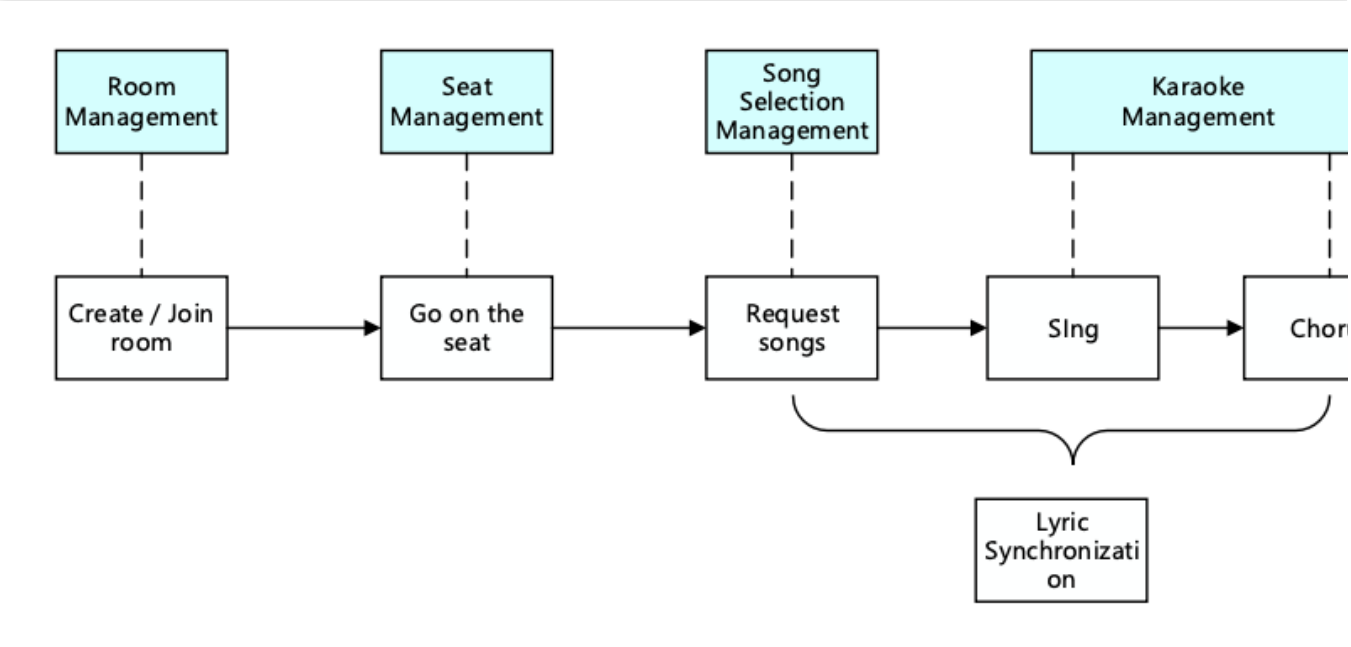
方案介绍（TUIKaraoke）
实现步骤

最近更新时间：2023-09-26 16:42:08

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。

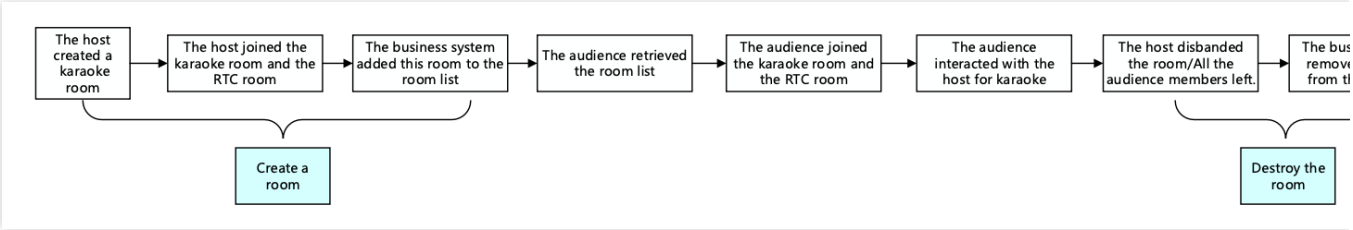
房间管理	麦位管理	点歌管理	K歌管理
房间列表	上/下麦	歌单展示	唱歌玩法
创建房间	麦位控制	搜索歌曲	切歌
加入房间	锁麦位	点歌	人声音量调节
退出房间	邀请上麦	歌曲置顶	混响/声效
销毁房间	麦位禁言	已点列表	歌词同步

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

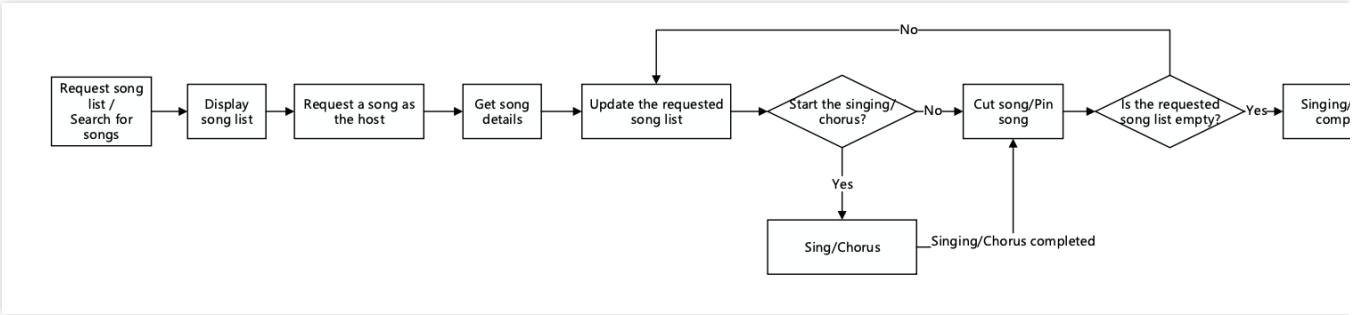
歌房内的麦位一般都是有序且有限的。麦位管理主要负责根据业务场景定义房间内的麦位数量，以及当前房间所有麦位的状态管理。麦位管理主要包含的功能：上/下麦、锁麦位、邀请上麦、麦位禁言等。
用户进入房间后，只有空闲状态的麦位才可以申请上麦。
房主同意用户上麦后，需要修改麦位状态为非空闲状态。
用户停止推流下麦后，需要重置麦位状态。
房主有权锁定麦位、邀请上麦、强制下麦、麦上禁言等。
说明：
麦位管理是实现在线K歌的必要模块，但并非主要功能模块，具体实现可以结合业务系统和 IM SDK，详见语聊房场景接入方案。

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

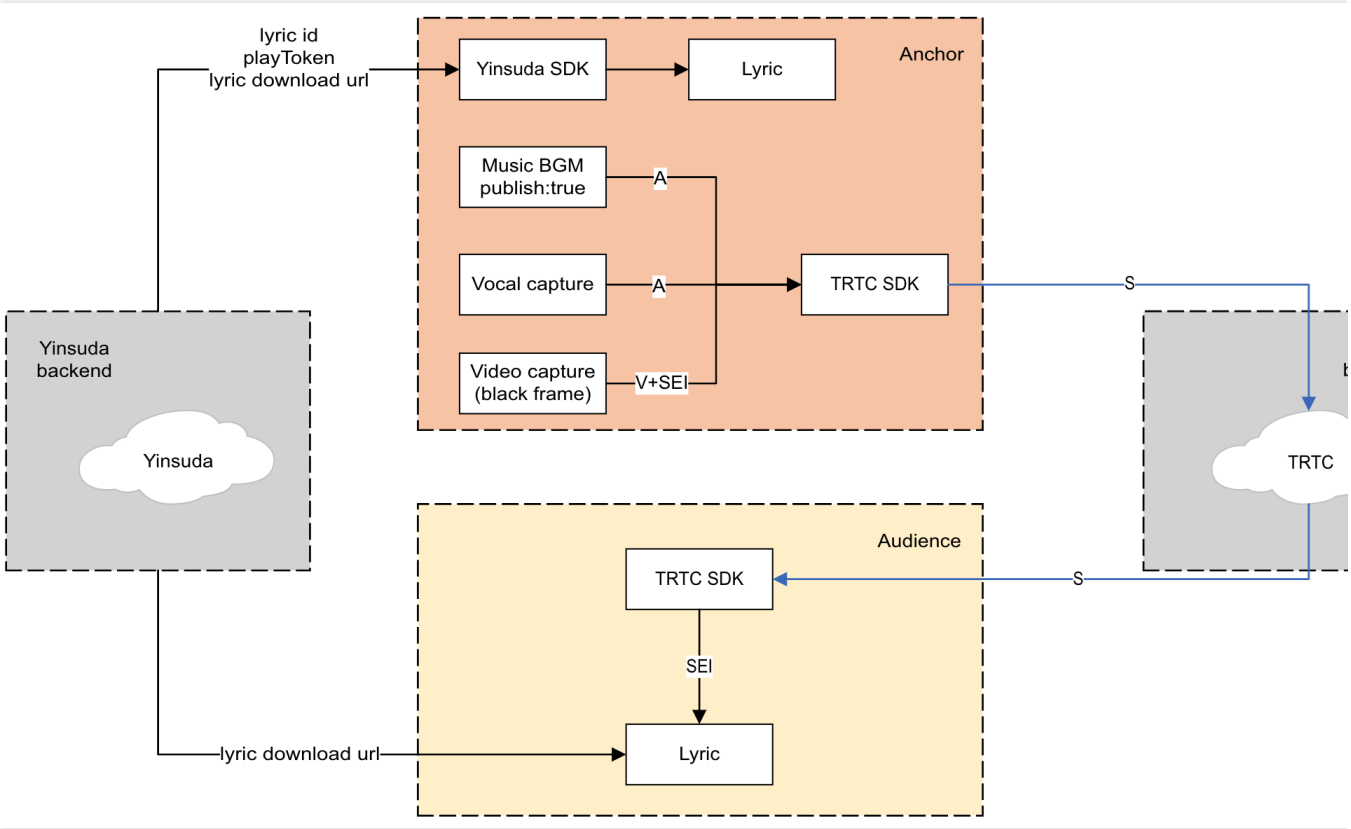
K 歌系统主要包含功能：唱歌玩法、开始/停止/切歌、人声音量调节、混响/声效、歌词同步。下面我们将通过演唱和实时合唱两种典型的 K 歌玩法来详细介绍 K 歌管理模块的实现流程。

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

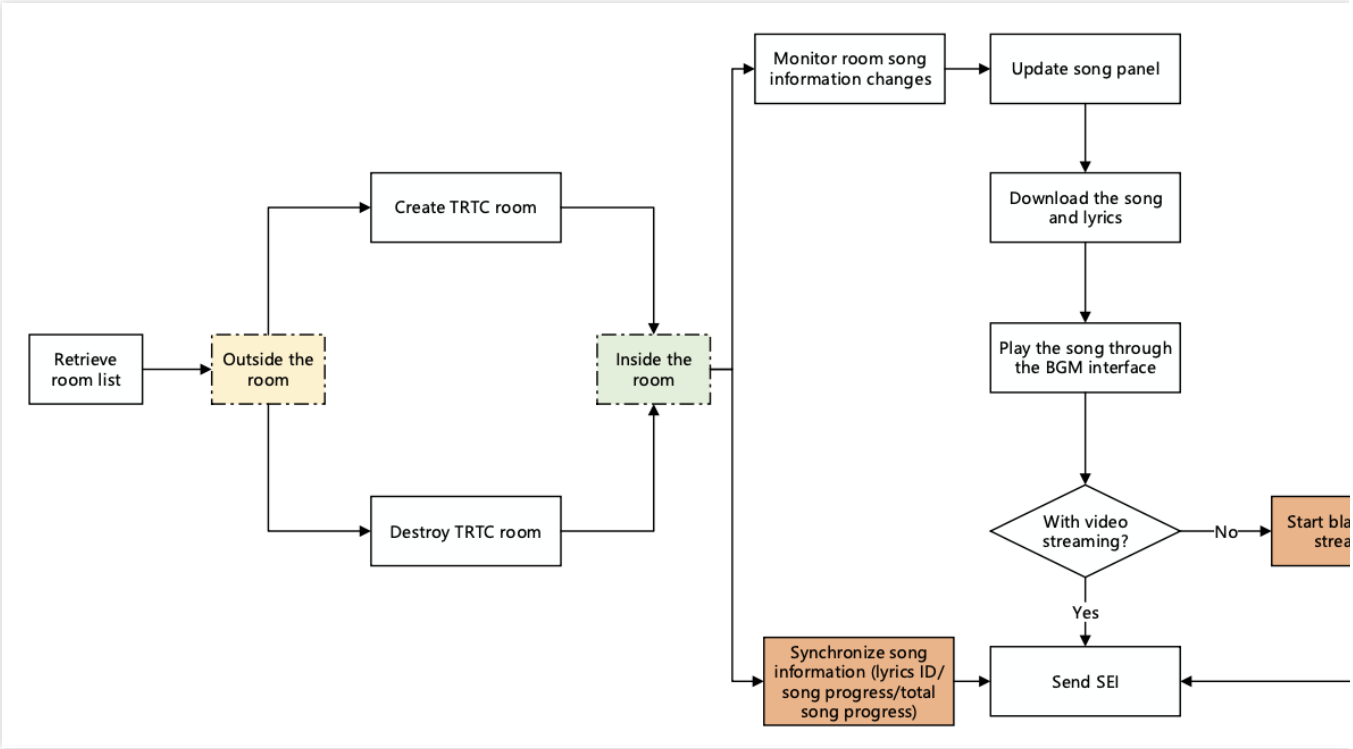
在演唱的场景，会区分不同角色有不同的实现流程，分为演唱者、观众 2种角色。

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

观众



房主创建并加入 TRTC 的房间，自动上麦后点歌成为演唱者。

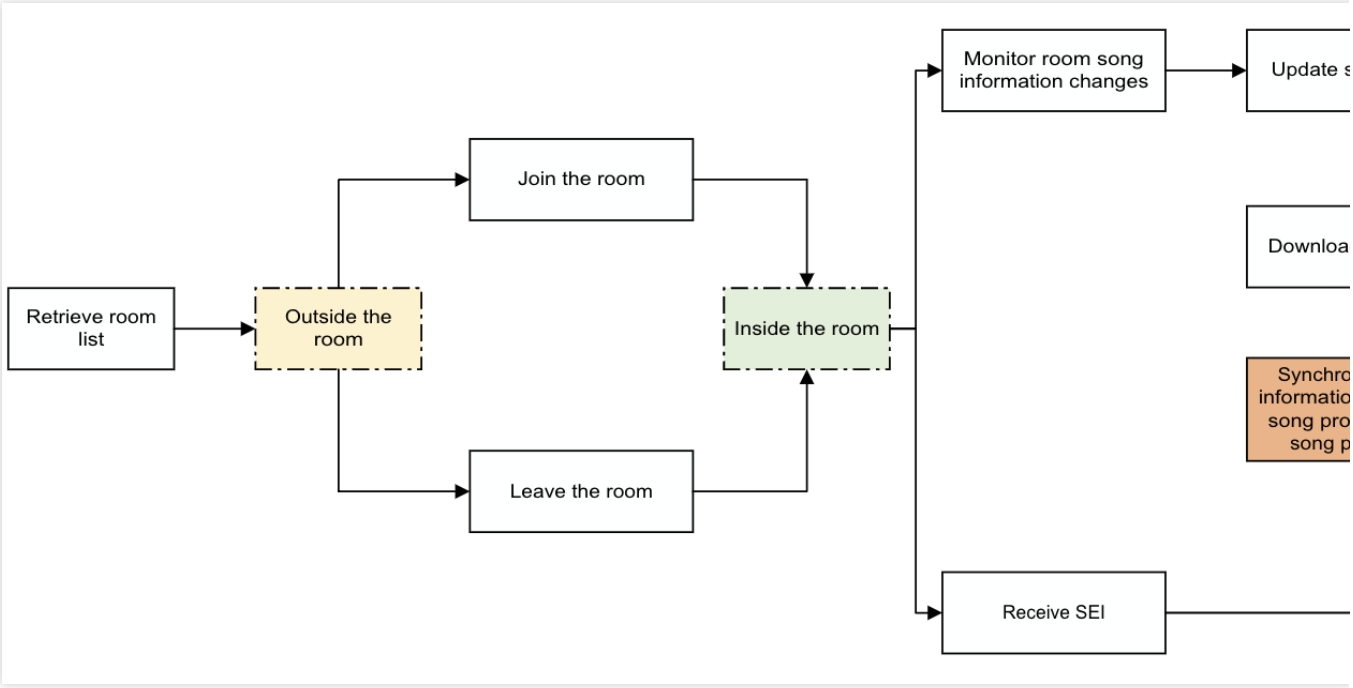
点歌后进行歌曲/歌词的下载，然后通过播放 BGM 接口播放歌曲。

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。



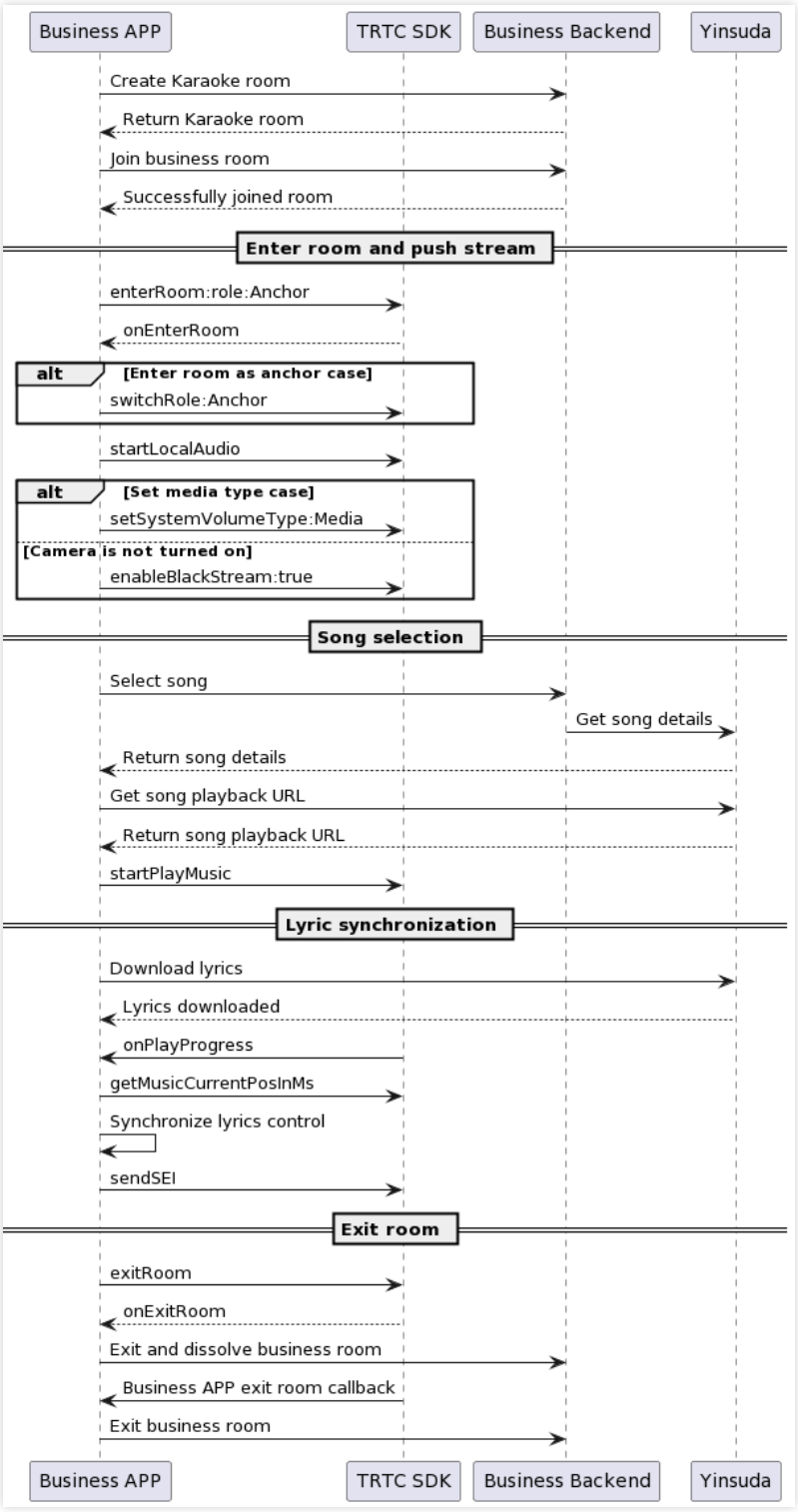
观众加入 TRTC 的房间。
监听房间歌曲变化，并加载歌词。
拉取演唱者的流。
解析演唱者发送的 SEI 信息，并同步歌词。

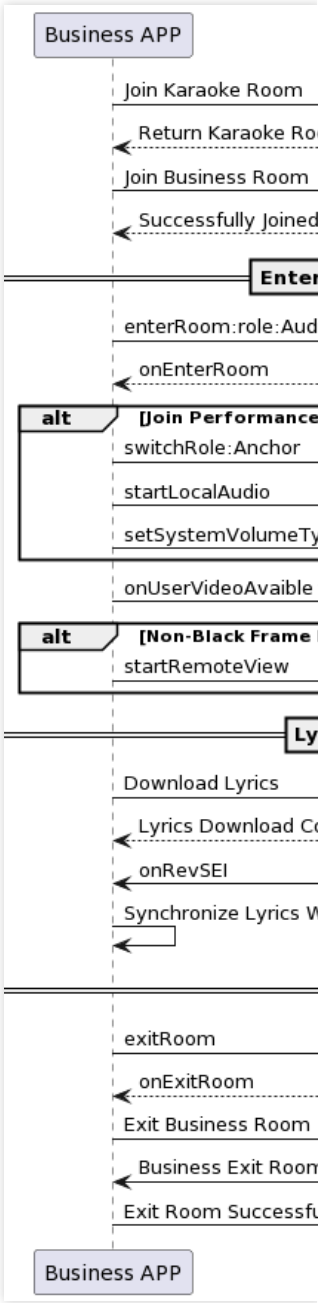
主要是要监听歌曲的 SEI 信息，更新对应的歌曲控件。

（3）API 调用时序

不同角色的 API 时序调用如下：

房主	观众





说明：

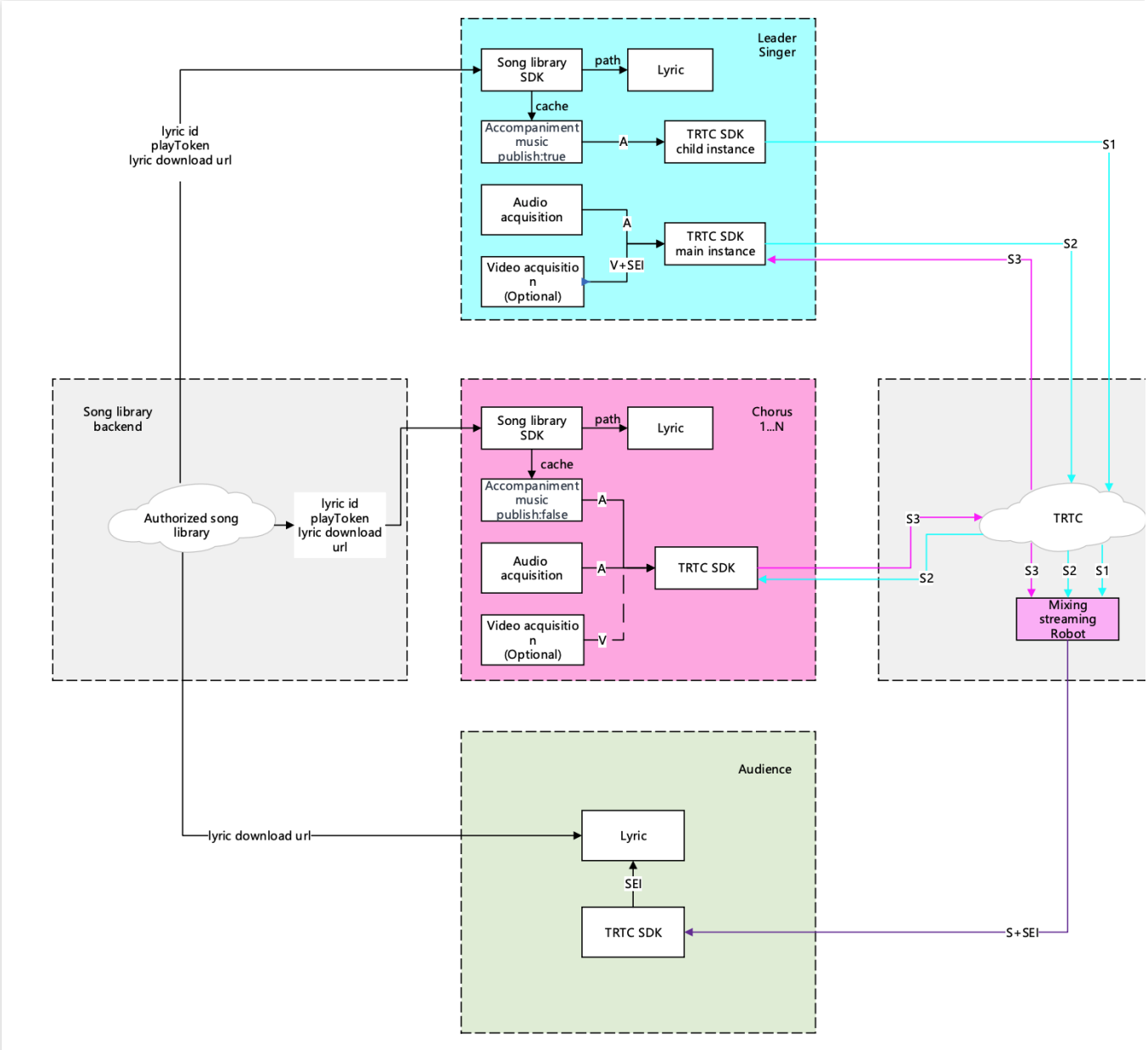
鉴于以上实现方案需要一定的技术门槛，而 TRTC 官网提供的TUIKaraoke 开源的音视频 UI 组件，可以通过在项目中集成 TUIKaraoke 组件，您只需要编写几行代码就可以为您的应用添加在线 K 歌场景，体验 K 歌、麦位管理、收发礼物、文字聊天等 TRTC 在 KTV 场景下的相关能力。

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，完美实现多人合唱的效果。实时合唱方案架构如下图所示。



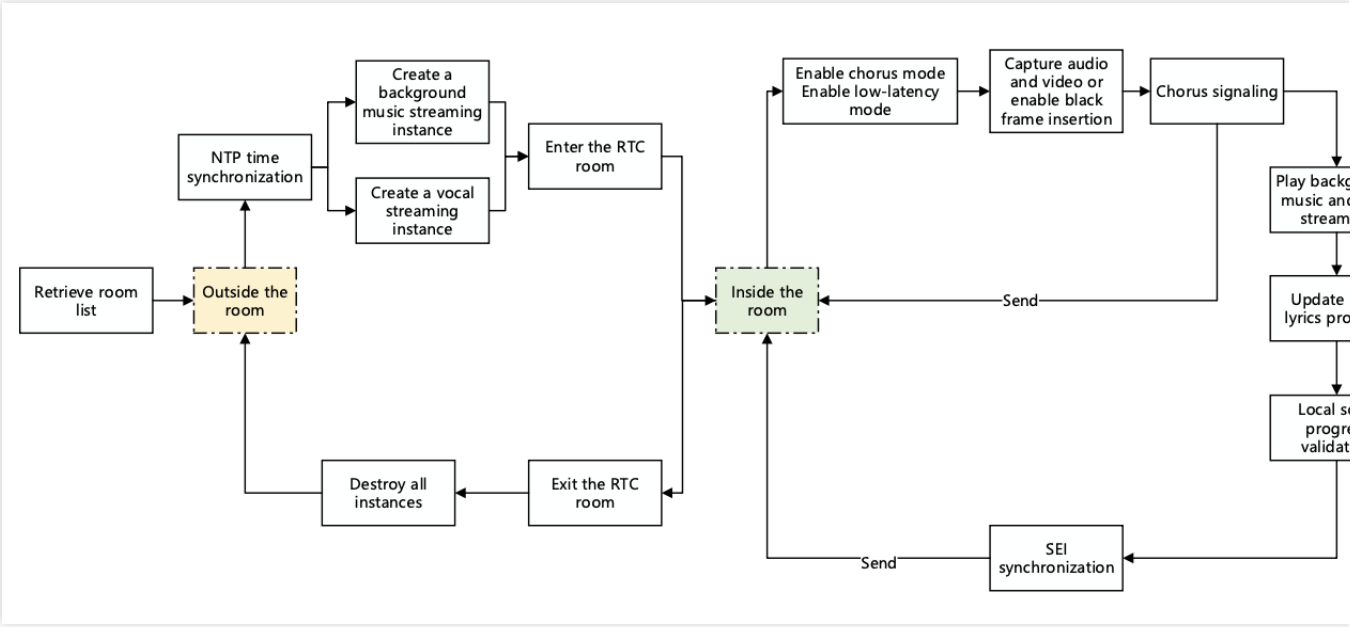
该方案的优点在于：
降低了端到端的时延。
提供了用户中途加入合唱的解决方案。
精准同步不同端之间的音乐、歌词、人声。
改善各端设备性能和本地时间不精准的情况，降低网络环境造成的时延影响。
说明：
根据业务需要，可以选择纯音频场景或音视频场景的实时合唱方案；若为纯音频场景则需要补黑帧以发送 SEI 消息用于歌词同步；
主唱需要使用子实例同时上行音乐及人声；其他合唱者仅互相拉取人声流，同时本地播放音乐；观众只需拉取一路混流；
图中展示的是 RTC 观看方案，混流机器人将合流回推至 RTC 房间；CDN 观看方案下，混流机器人将合流转推至直播 CDN，观众拉取 CDN 流观看。

（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
合唱
观众



主唱需要点播歌曲，发送合唱信令。

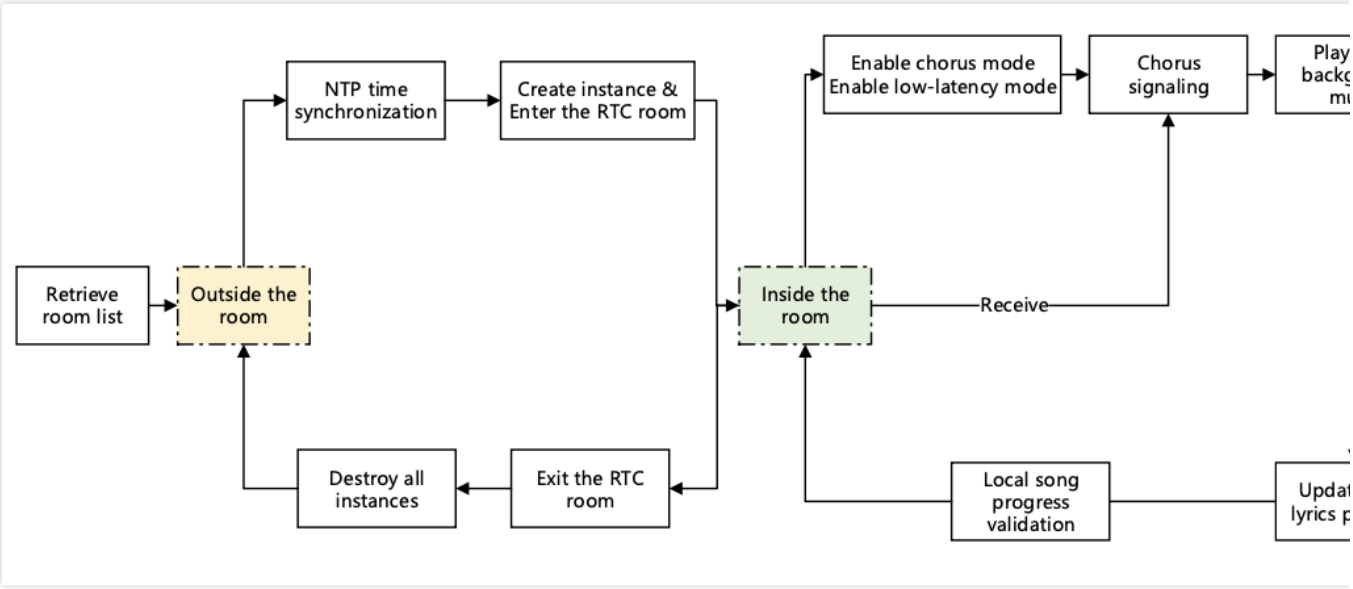
主唱创建子实例推送人声和音乐，并拉取其他合唱者的人声。

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。

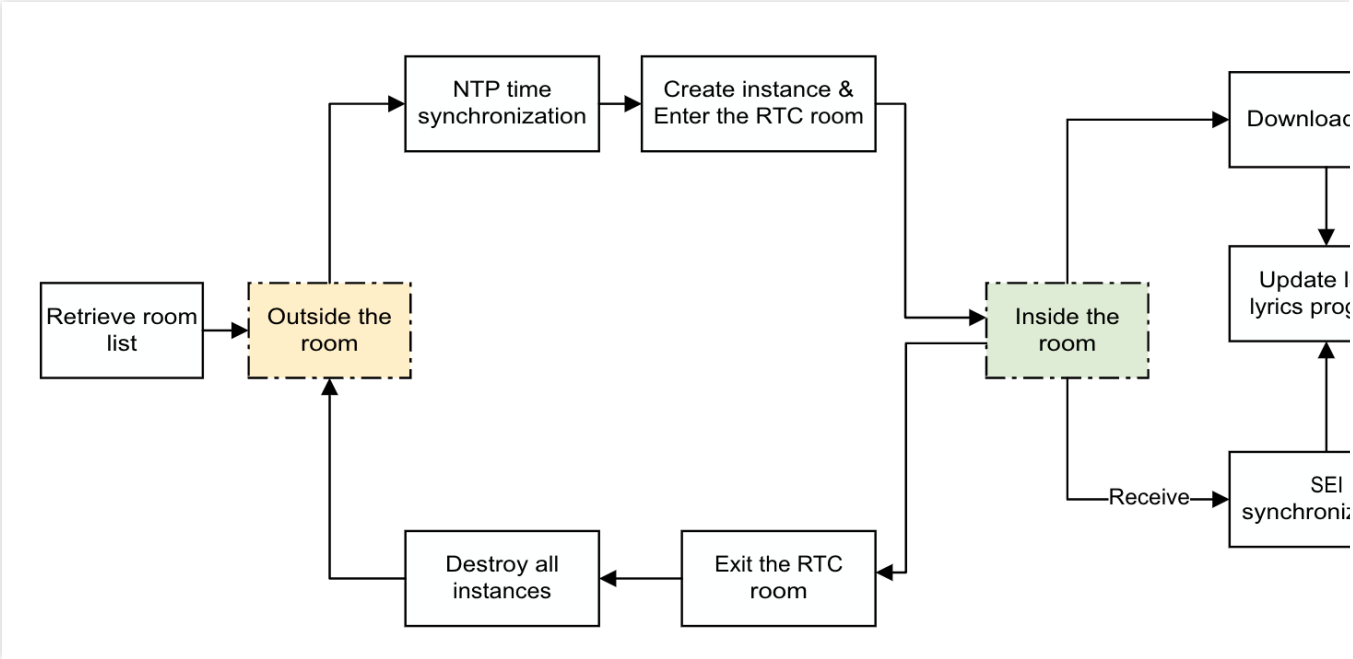
需要通过 SEI 发送歌曲进度，以便观众端同步歌词。

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。

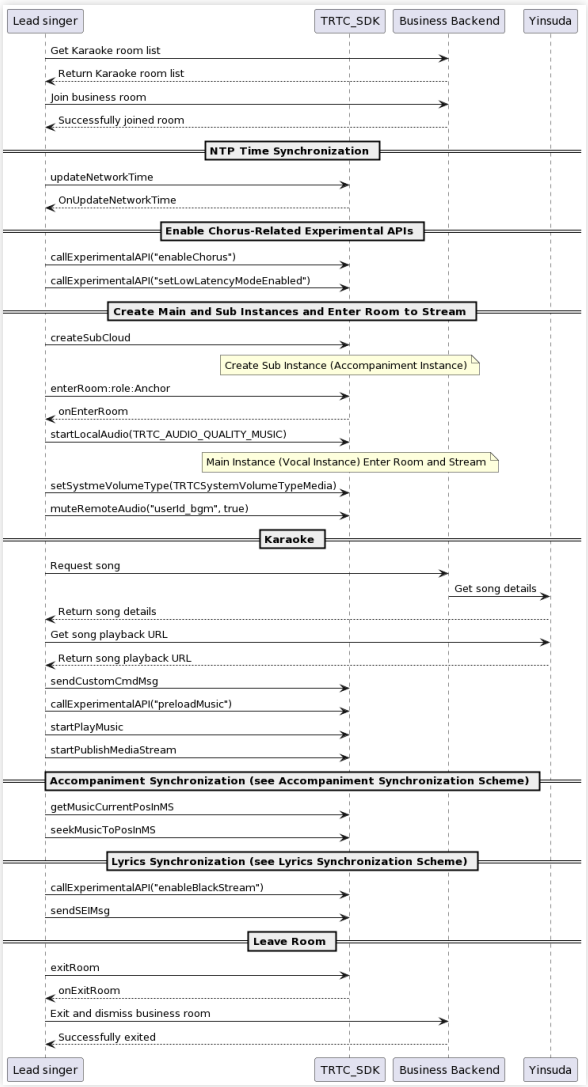


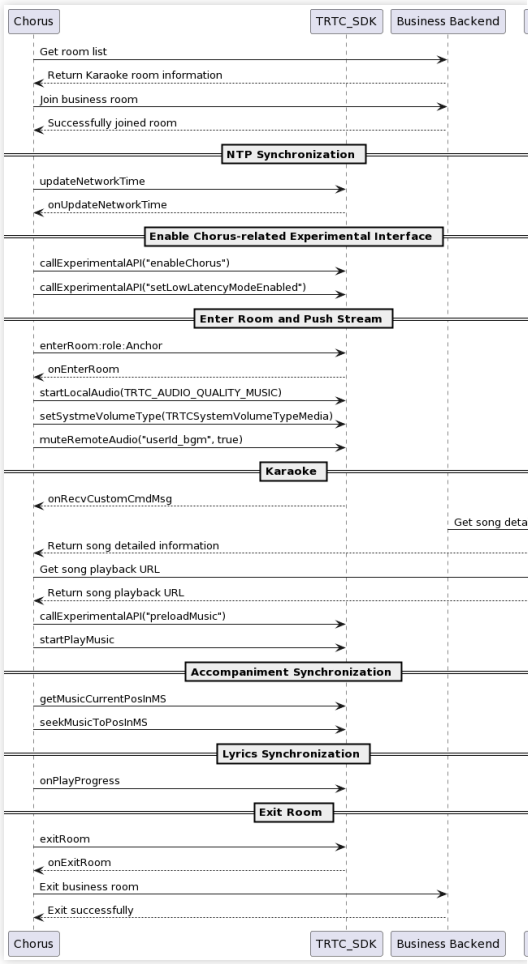
拉混流收听合唱。
解析混流中 SEI 的歌曲进度信息，用于同步歌词；
上麦后停止拉混流，转为拉麦上人声流，同时开启合唱模式。

（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

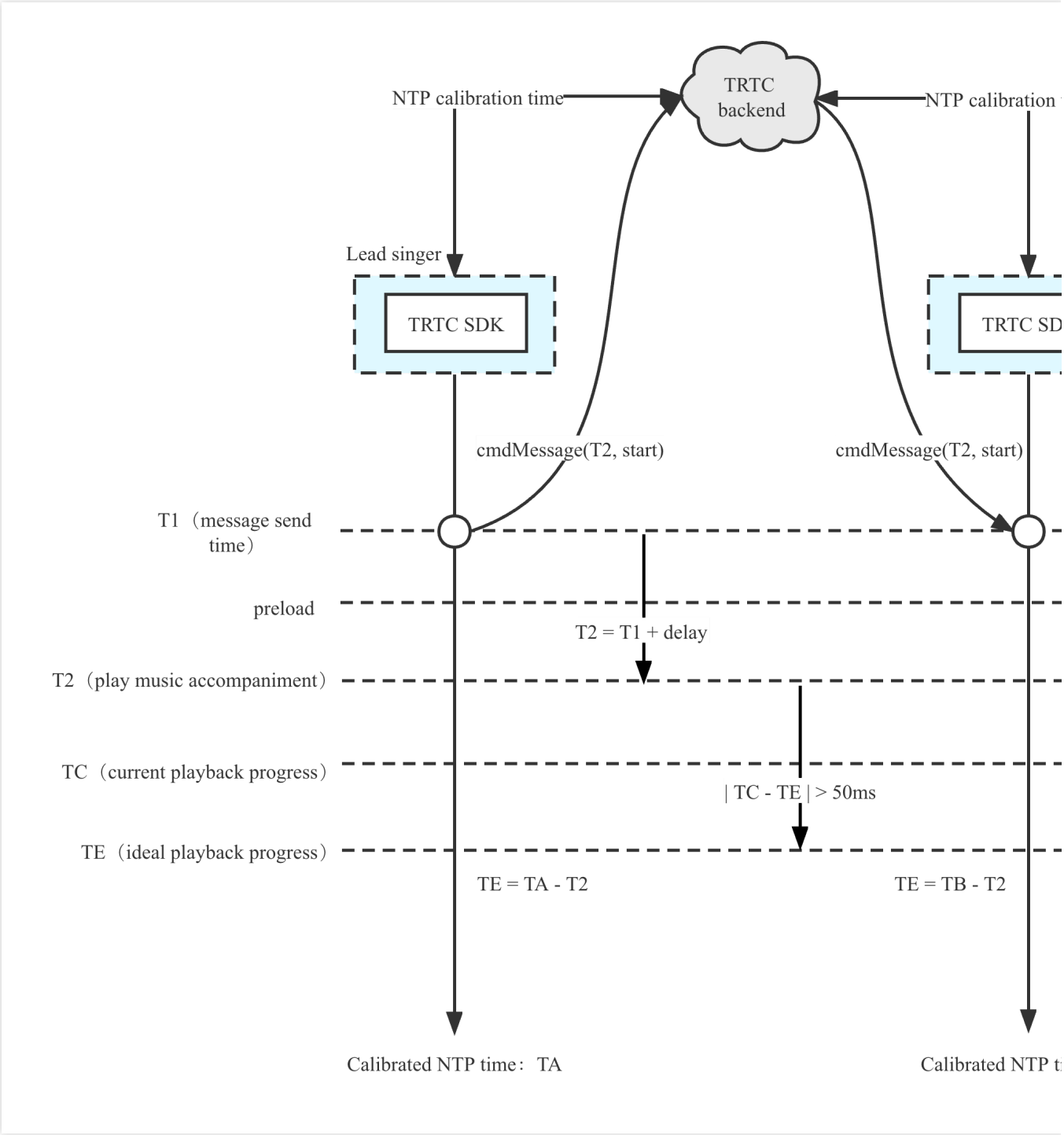
iOS

最近更新时间：2023-09-27 14:44:45

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。



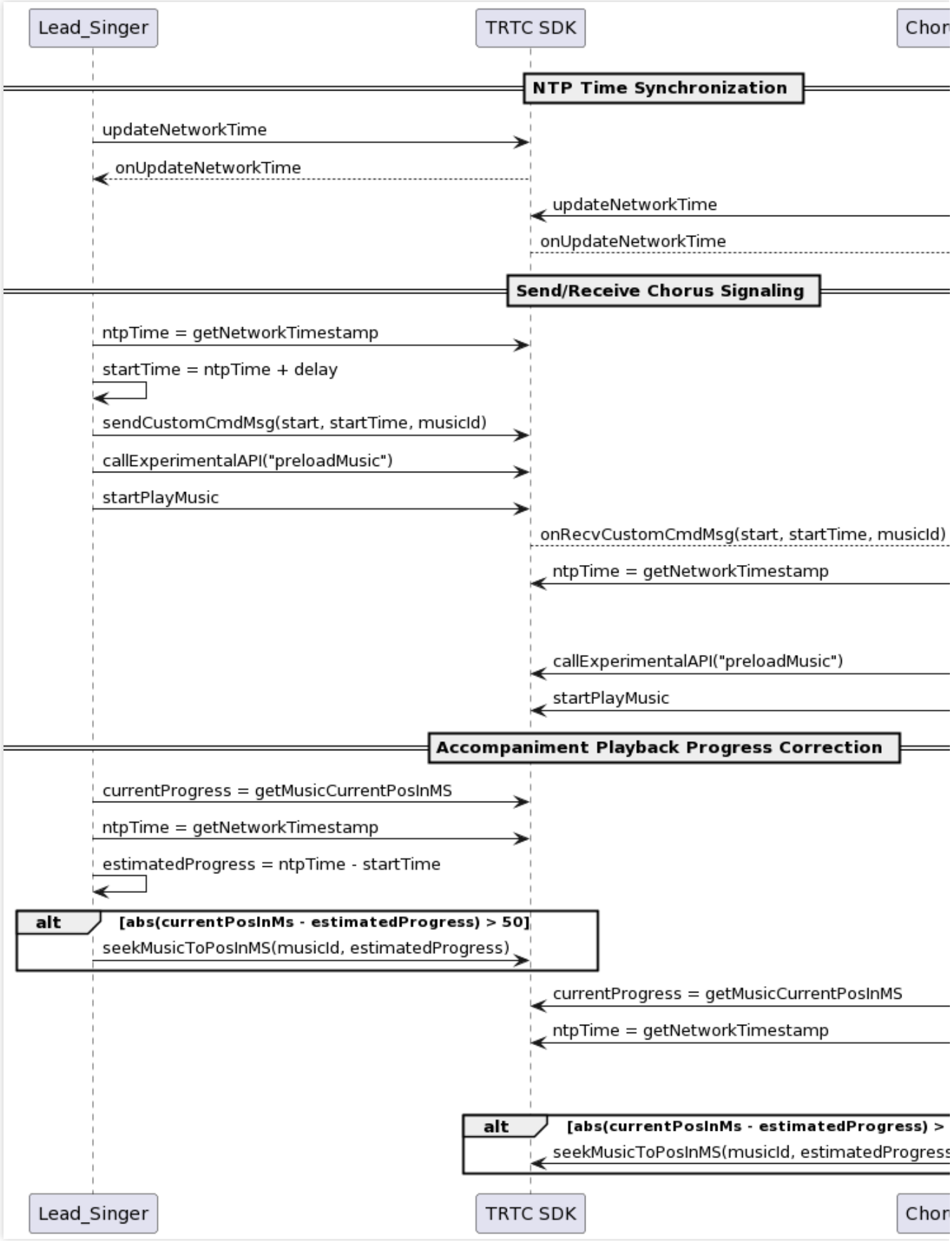
具体流程如下：

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。

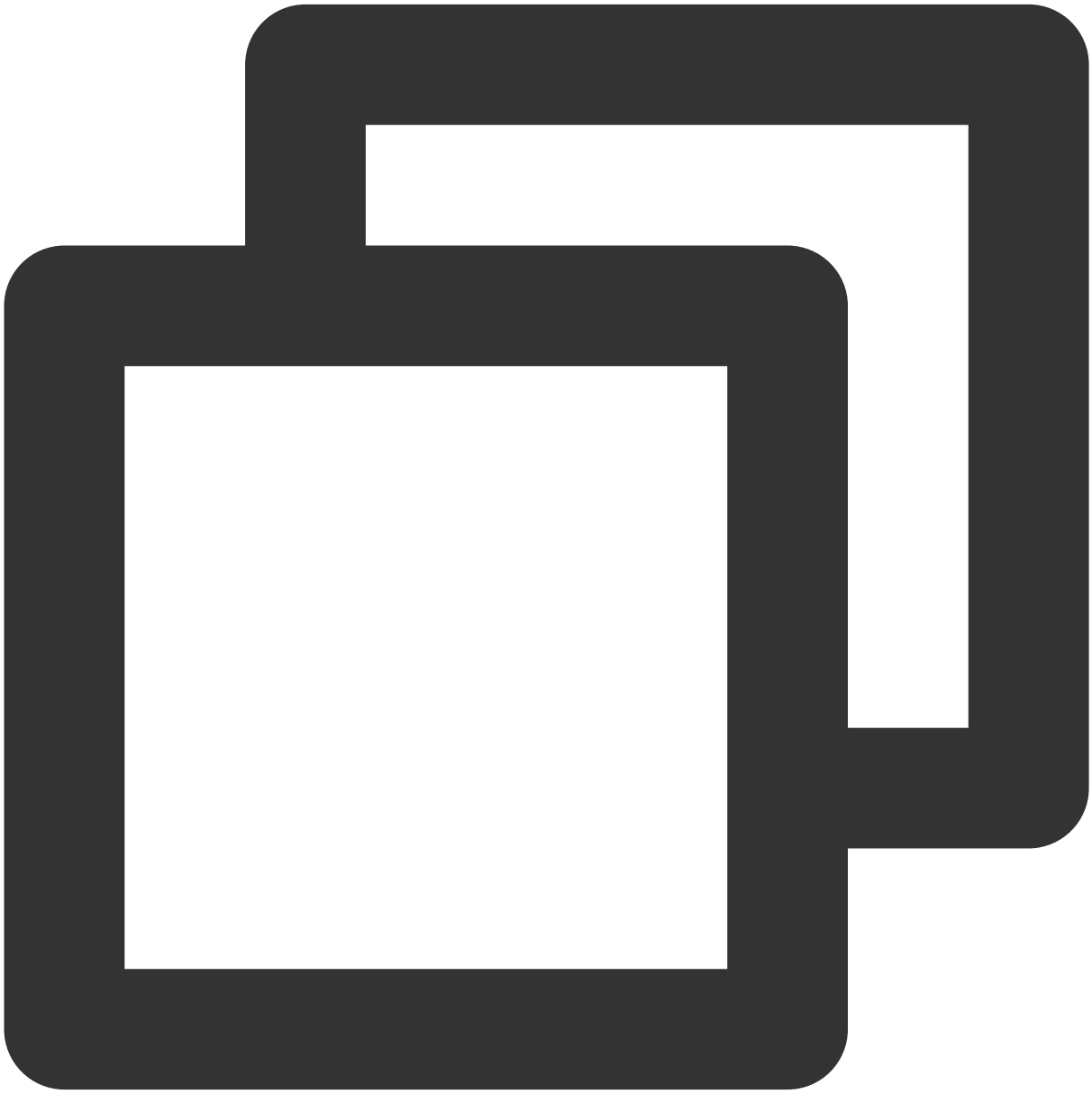
时序图


Lead_Singer
TRTC SDK
Chor
NTP Time Synchronization
updateNetworkTime
onUpdateNetworkTime
updateNetworkTime
onUpdateNetworkTime
Send/Receive Chorus Signaling
ntpTime = getNetworkTimestamp
startTime = ntpTime + delay
sendCustomCmdMsg(start, startTime, musicId)
callExperimentalAPI("preloadMusic")
startPlayMusic
onRecvCustomCmdMsg(start, startTime, musicId)
ntpTime = getNetworkTimestamp
callExperimentalAPI("preloadMusic")
startPlayMusic
Accompaniment Playback Progress Correction
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
alt
[abs(currentPosInMs - estimatedProgress) >
seekMusicToPosInMS(musicId, estimatedProgress
Lead_Singer
TRTC SDK
Chor

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

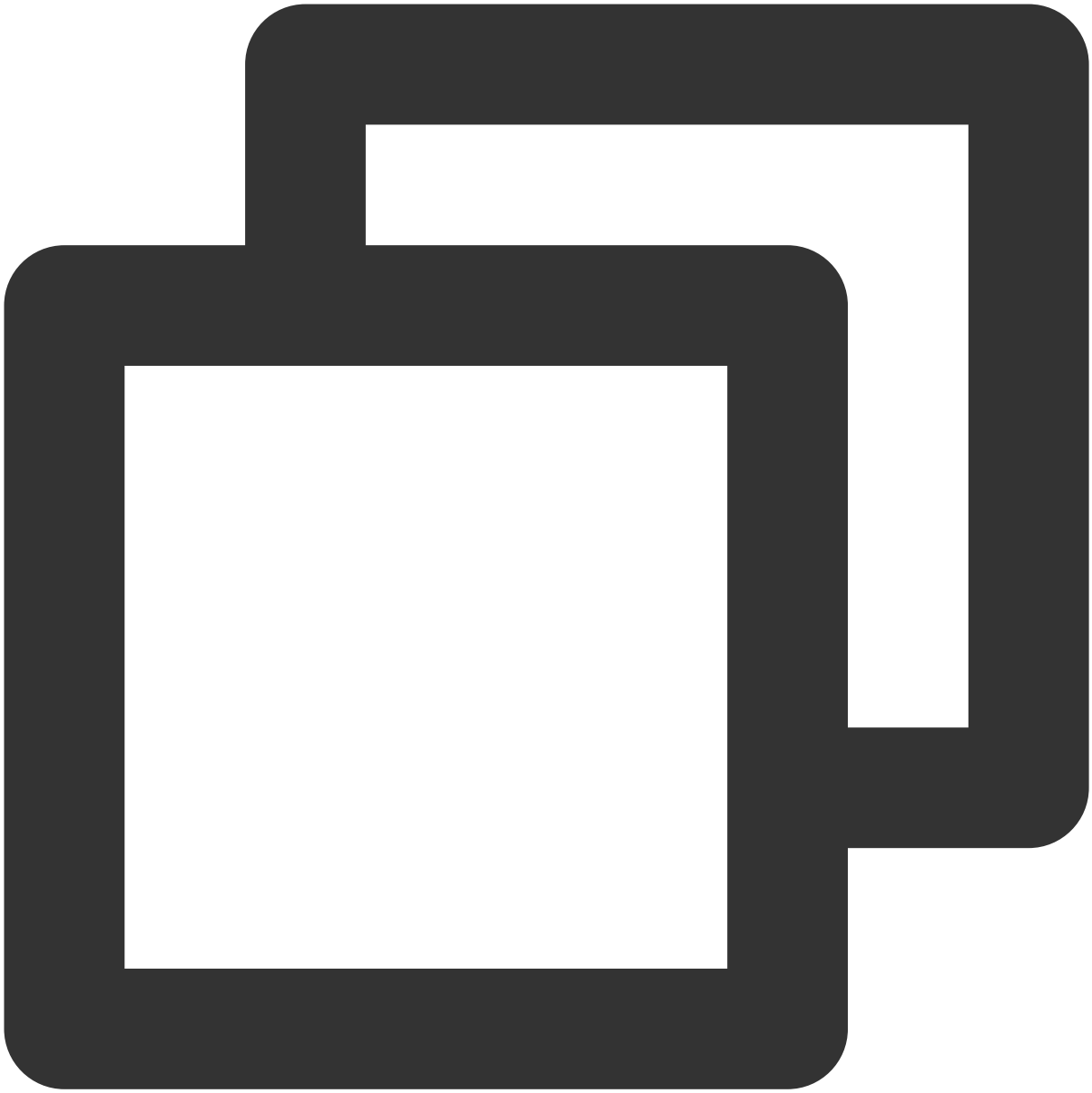
1. NTP 校时服务



```
//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

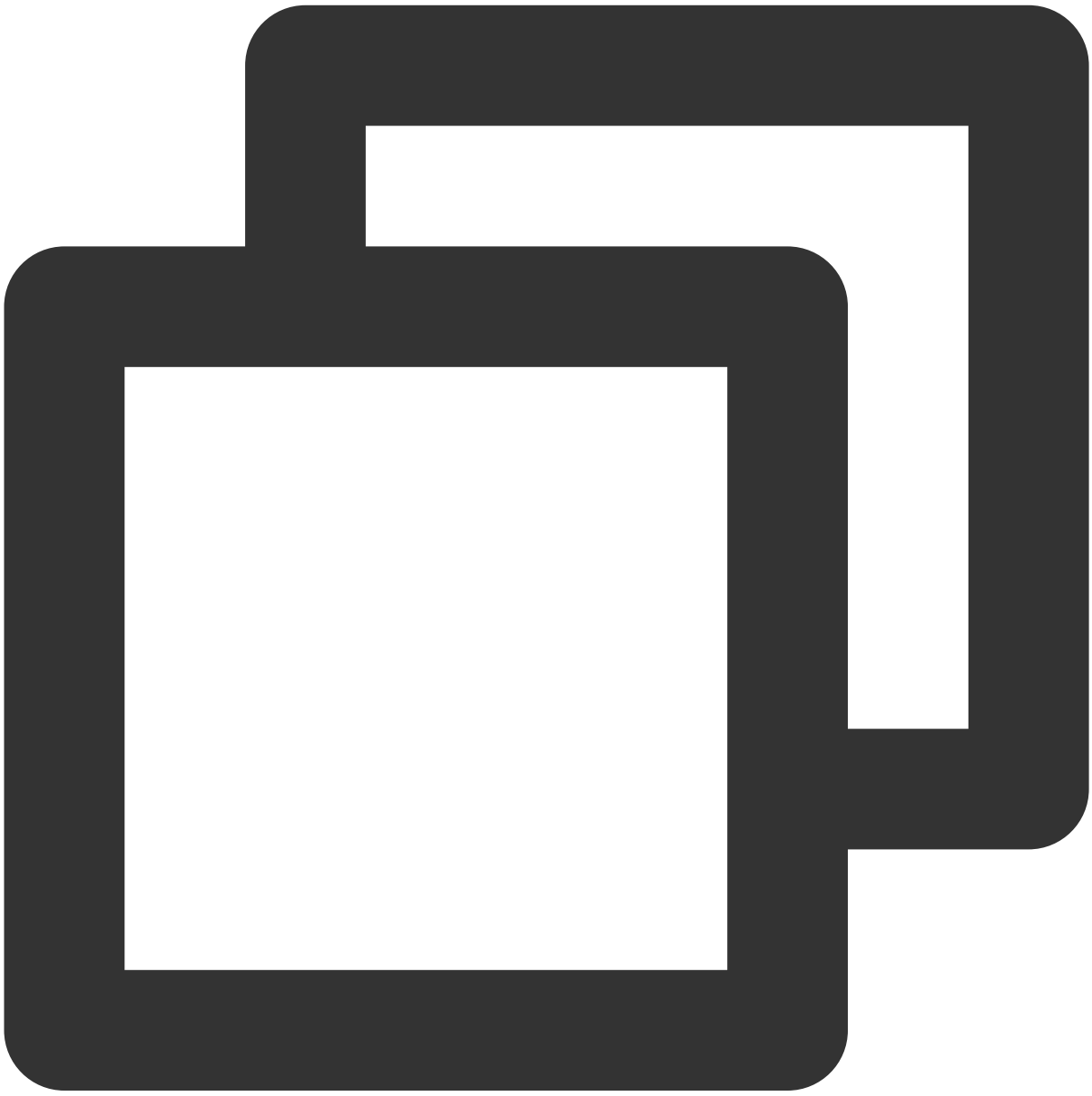


```
NSDictionary *json = @{
                        @"cmd": @"startChorus",
                        // 约定时间合唱
                        @"startPlayMusicTS": @(startTs),
                        @"musicId": @"musicId",
                        @"musicDuration": @(musicDuration),
                      };
NSString *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:(UIn

    NSString *msg = [[NSString alloc] initWithData:message encoding:NSUTF8StringEnc
    NSError *error;
```

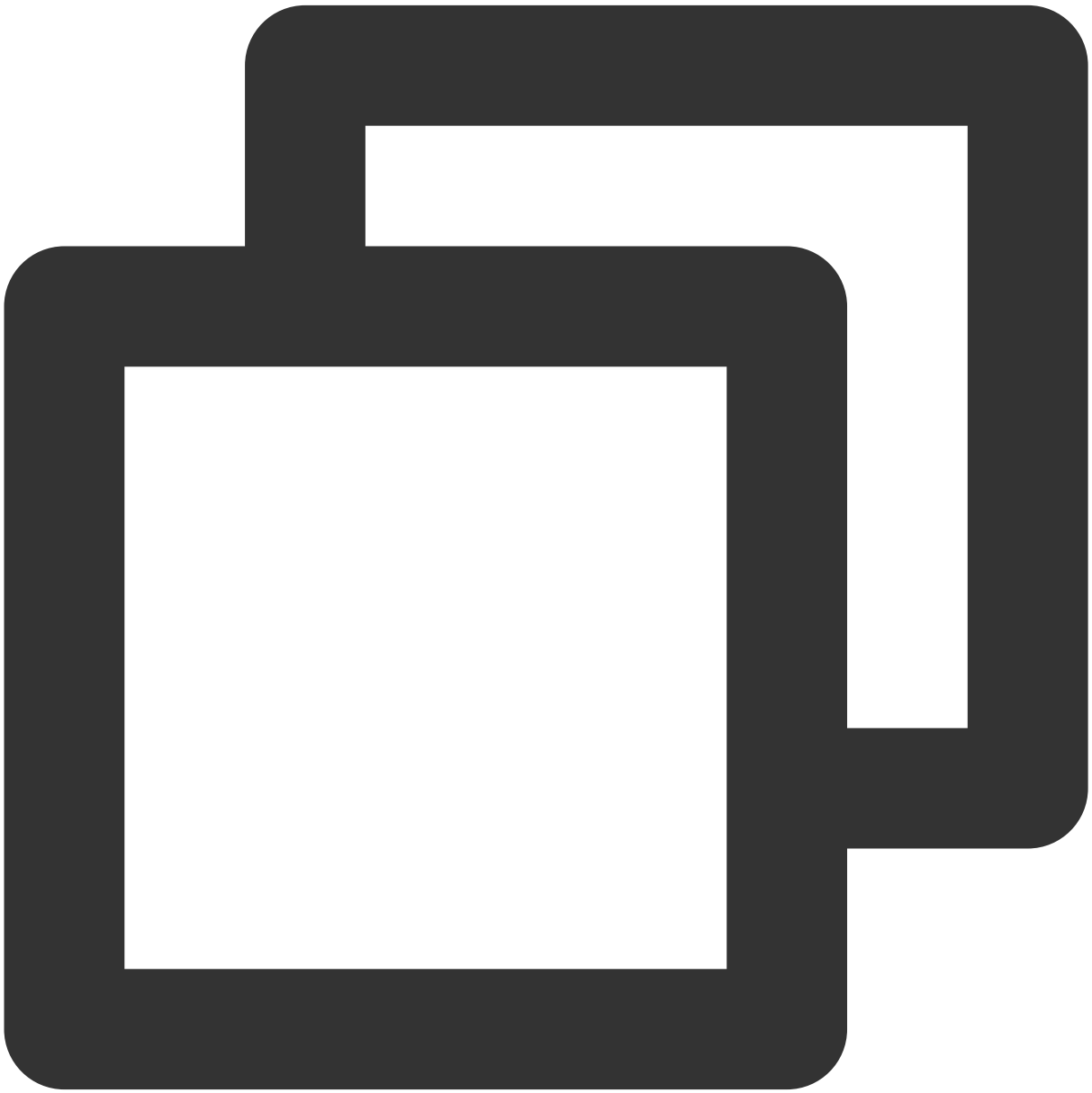
```
NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[msg dataUsingEnco
                                                      options:NSJSONReadingMutab
                                                        error:&error];

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).longLongValu
    // 合唱约定时间和当前时间差值
    NSInteger startDelayMS = labs(startPlayMusicTs - [TXLiveBase getNetworkTime
    // 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
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSU
    [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
        // 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.c
// 当前歌曲的理想播放时间进度
NSInteger estimatedProgress = [TXLiveBase getNetworkTimestamp] - self.startPlayChor
if (estimatedProgress >= 0 && labs(currentProgress - estimatedProgress) > 50) {
    // 当播放进度超过50ms，进行修正
    [[subCloud getAudioEffectManager] seekMusicToPosInMS:self.currentPlayMusicID pt
}
```

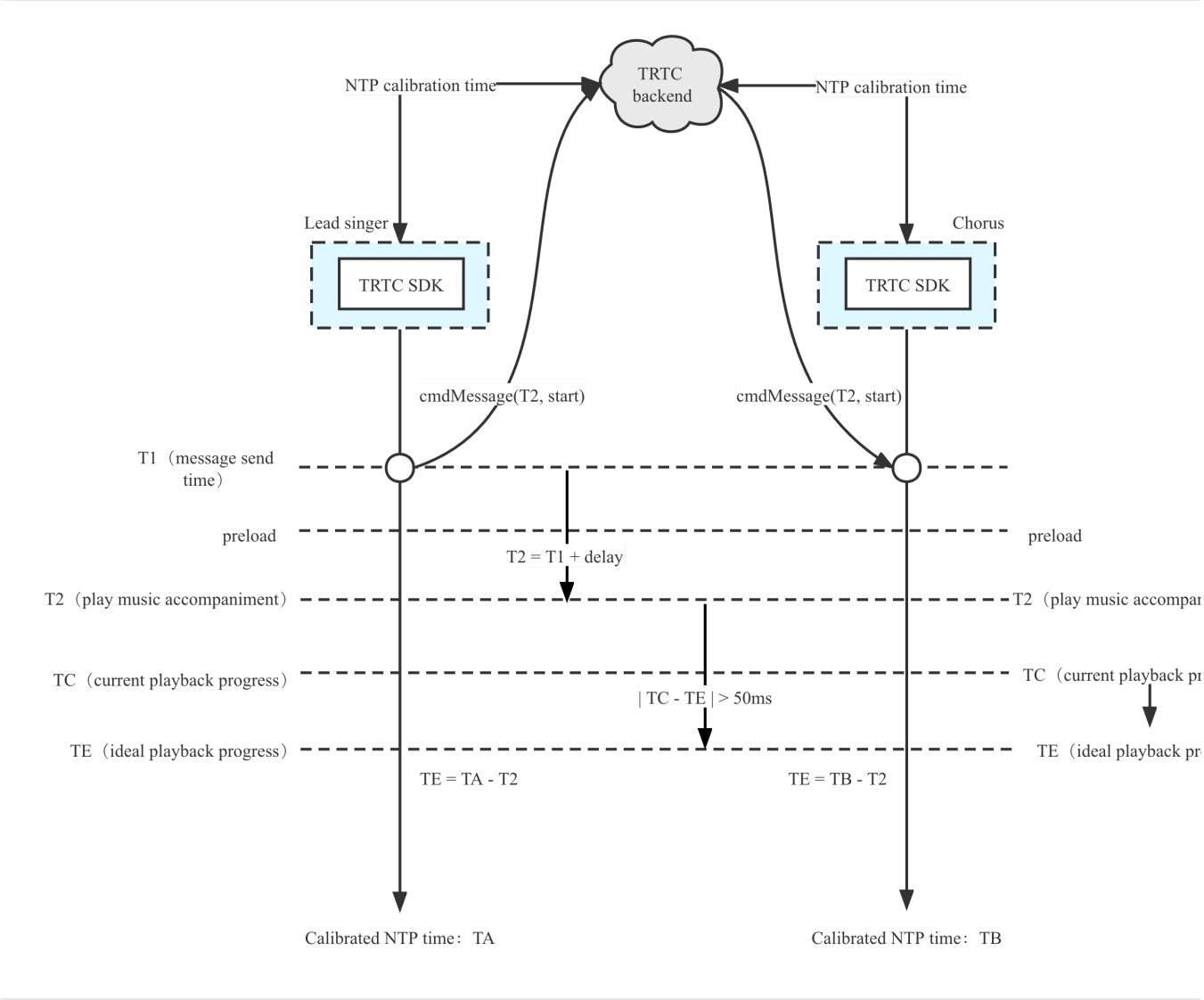
Android

最近更新时间：2023-09-26 16:45:26

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。



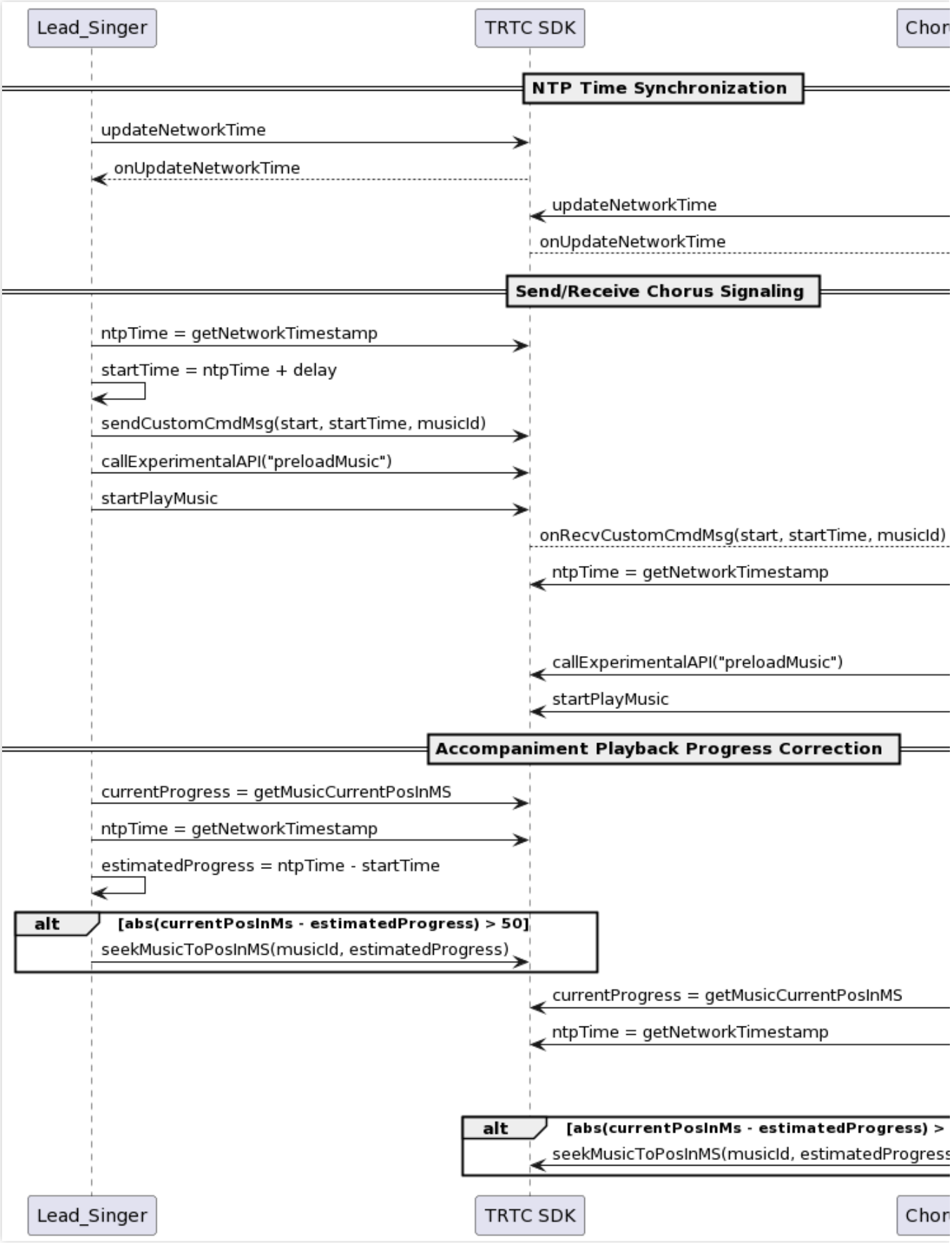
具体流程如下：

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。

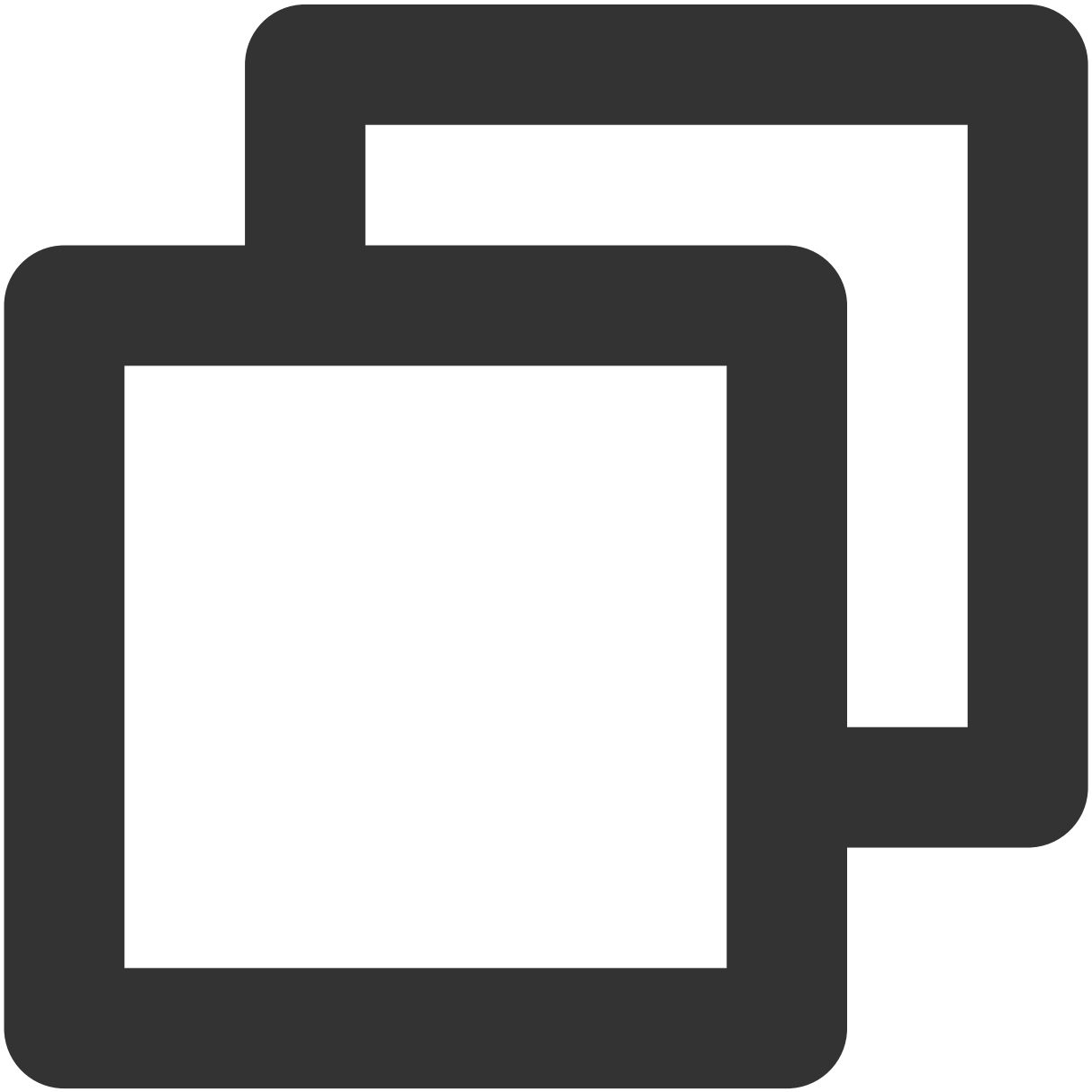
时序图


Lead_Singer
TRTC SDK
Chor
NTP Time Synchronization
updateNetworkTime
onUpdateNetworkTime
updateNetworkTime
onUpdateNetworkTime
Send/Receive Chorus Signaling
ntpTime = getNetworkTimestamp
startTime = ntpTime + delay
sendCustomCmdMsg(start, startTime, musicId)
callExperimentalAPI("preloadMusic")
startPlayMusic
onRecvCustomCmdMsg(start, startTime, musicId)
ntpTime = getNetworkTimestamp
callExperimentalAPI("preloadMusic")
startPlayMusic
Accompaniment Playback Progress Correction
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
estimatedProgress = ntpTime - startTime
alt
[abs(currentPosInMs - estimatedProgress) > 50]
seekMusicToPosInMS(musicId, estimatedProgress)
currentProgress = getMusicCurrentPosInMS
ntpTime = getNetworkTimestamp
alt
[abs(currentPosInMs - estimatedProgress) >
seekMusicToPosInMS(musicId, estimatedProgress
Lead_Singer
TRTC SDK
Chor

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

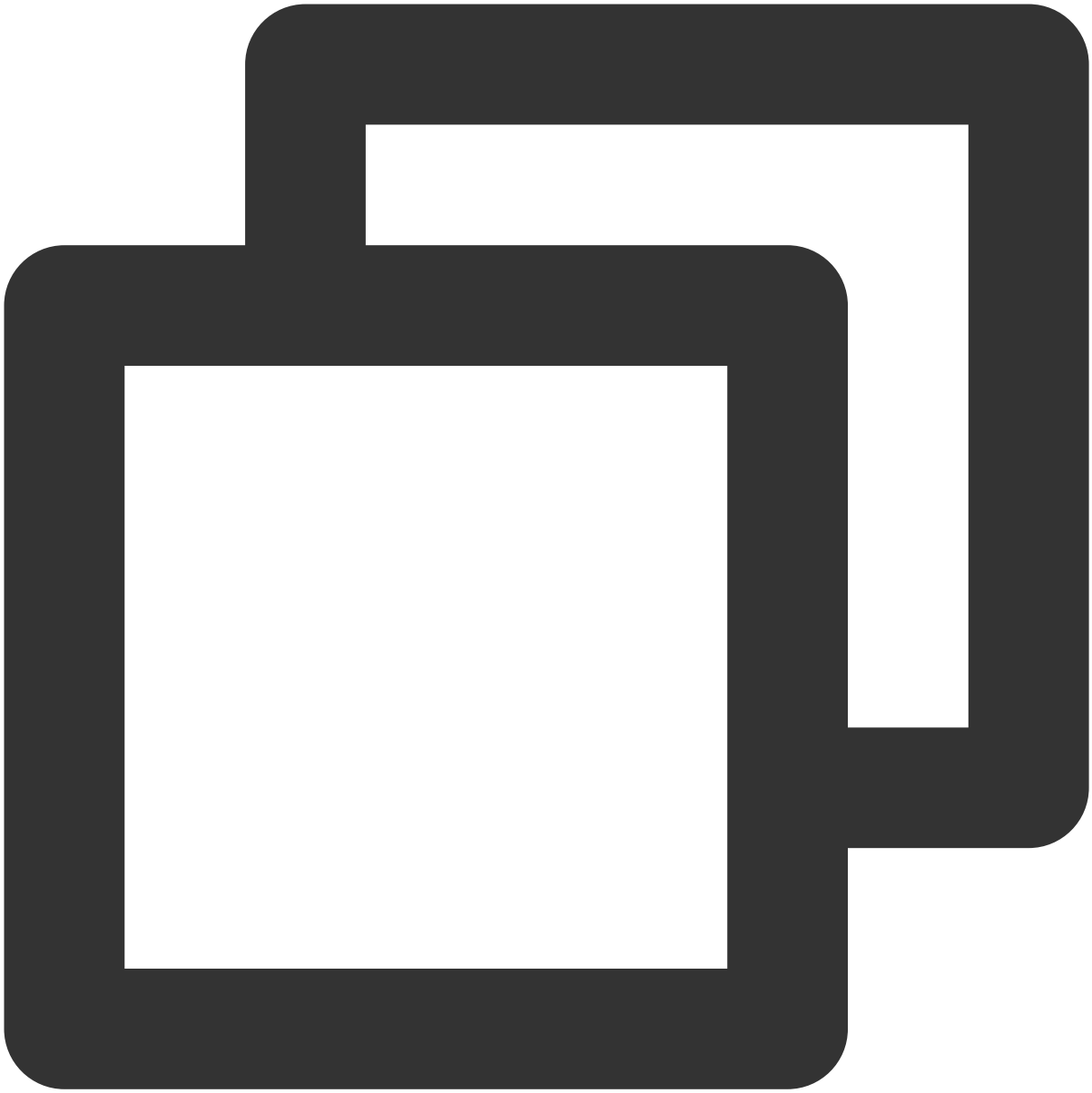
1. NTP 校时服务



```
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



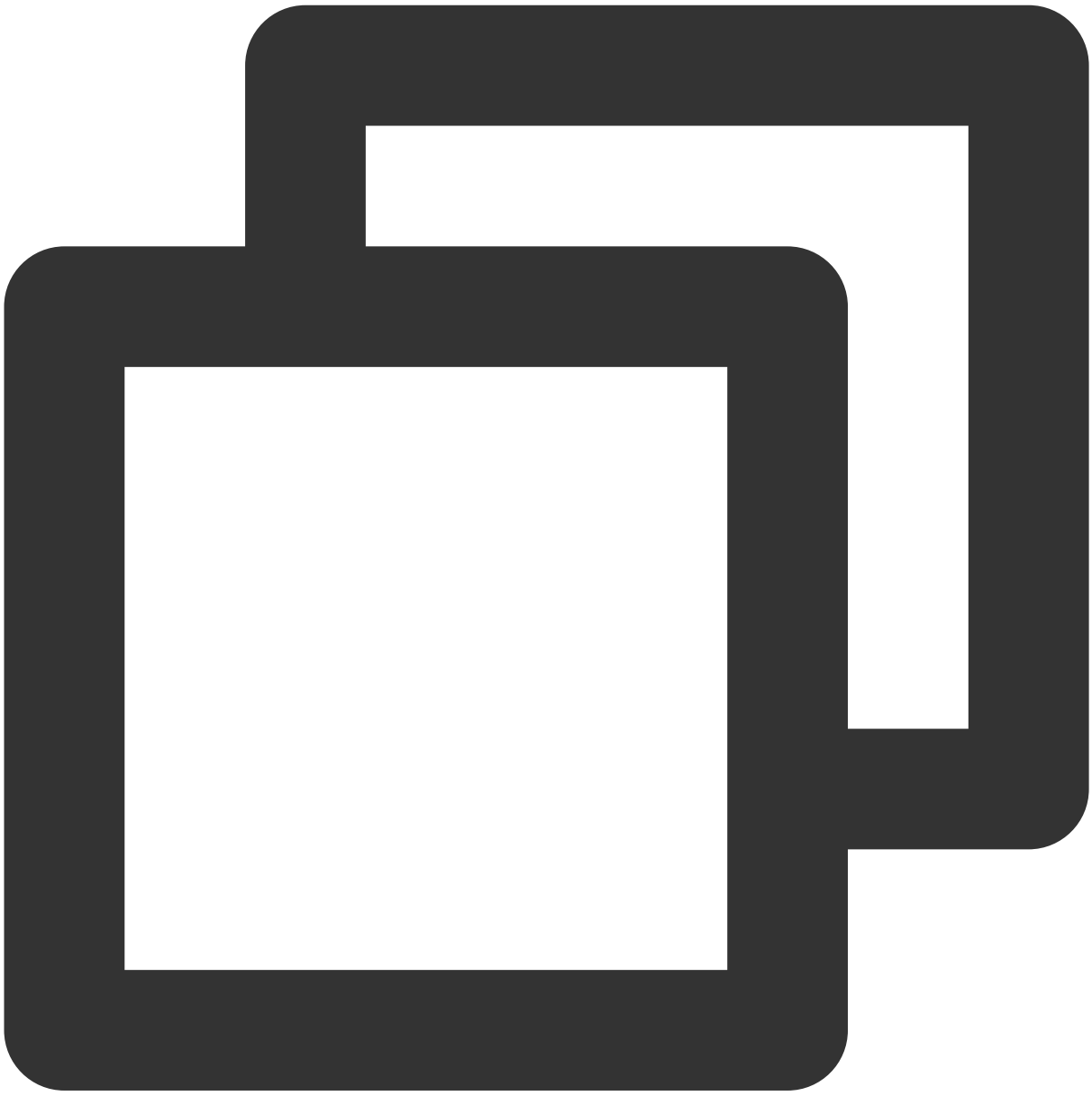
```
JSONObject jsonObject = new JSONObject();
jsonObject.put("cmd", "startChorus");
// 约定时间合唱
jsonObject.put("startPlayMusicTS", startTs);
jsonObject.put("musicId", "musicId");
String body = jsonObject.toString();
mTRTCCloud.sendCustomCmdMsg(0, body.getBytes(), false, false);
```

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



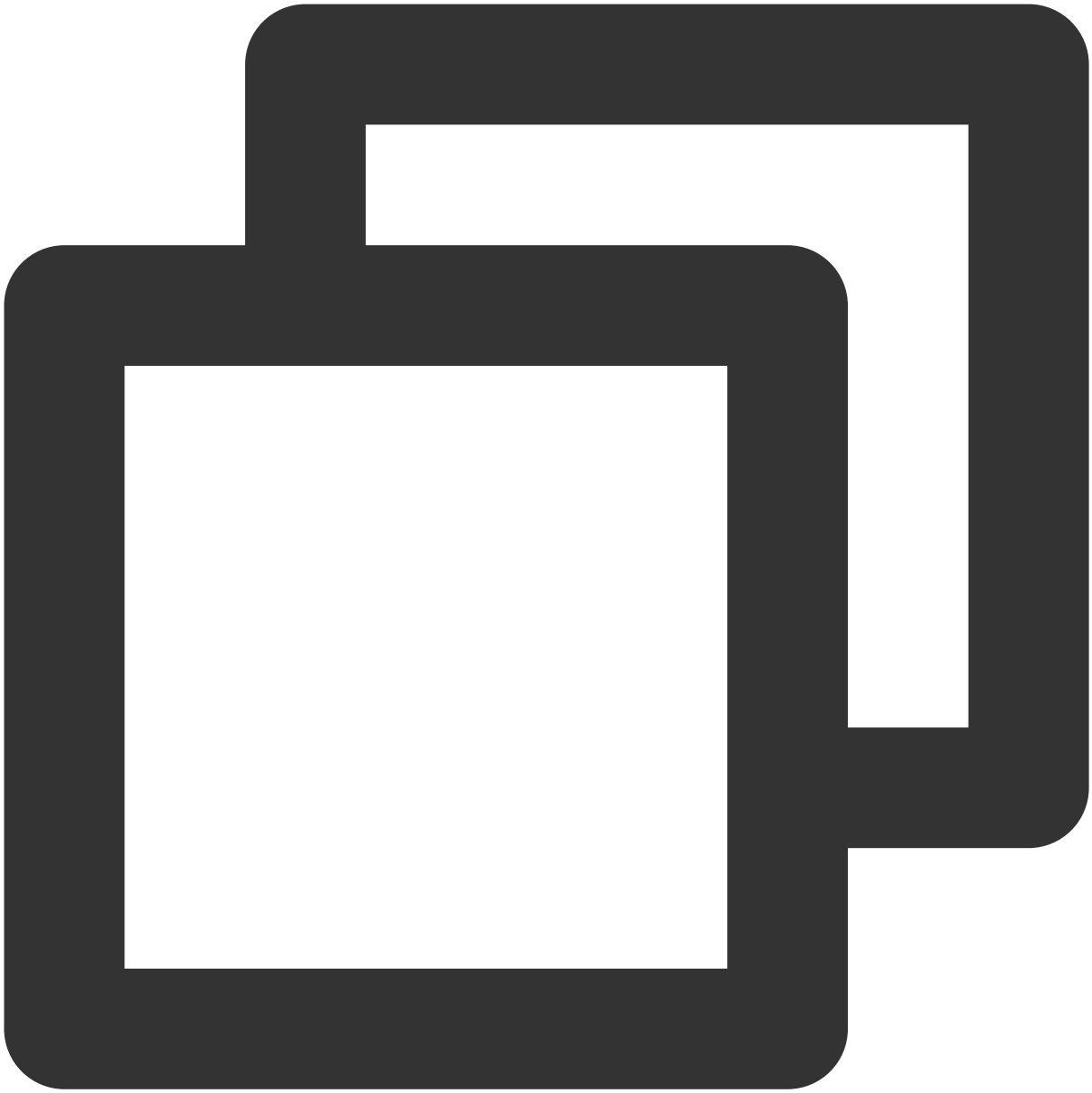
```
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[] message) {
    JSONObject json = new JSONObject(new String(message, "UTF-8"));
    String cmd = json.getString("cmd");
    // 合唱命令
    if (cmd.equals("startChorus")) {
    // 合唱开始时间
```

```
    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\\": {\\
    // 播放歌曲
    TXAudioEffectManager.AudioMusicParam param = new TXAudioEffectManager.AudioMusi
    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(musi
// 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) estimatedPr
}
```

歌词同步

iOS

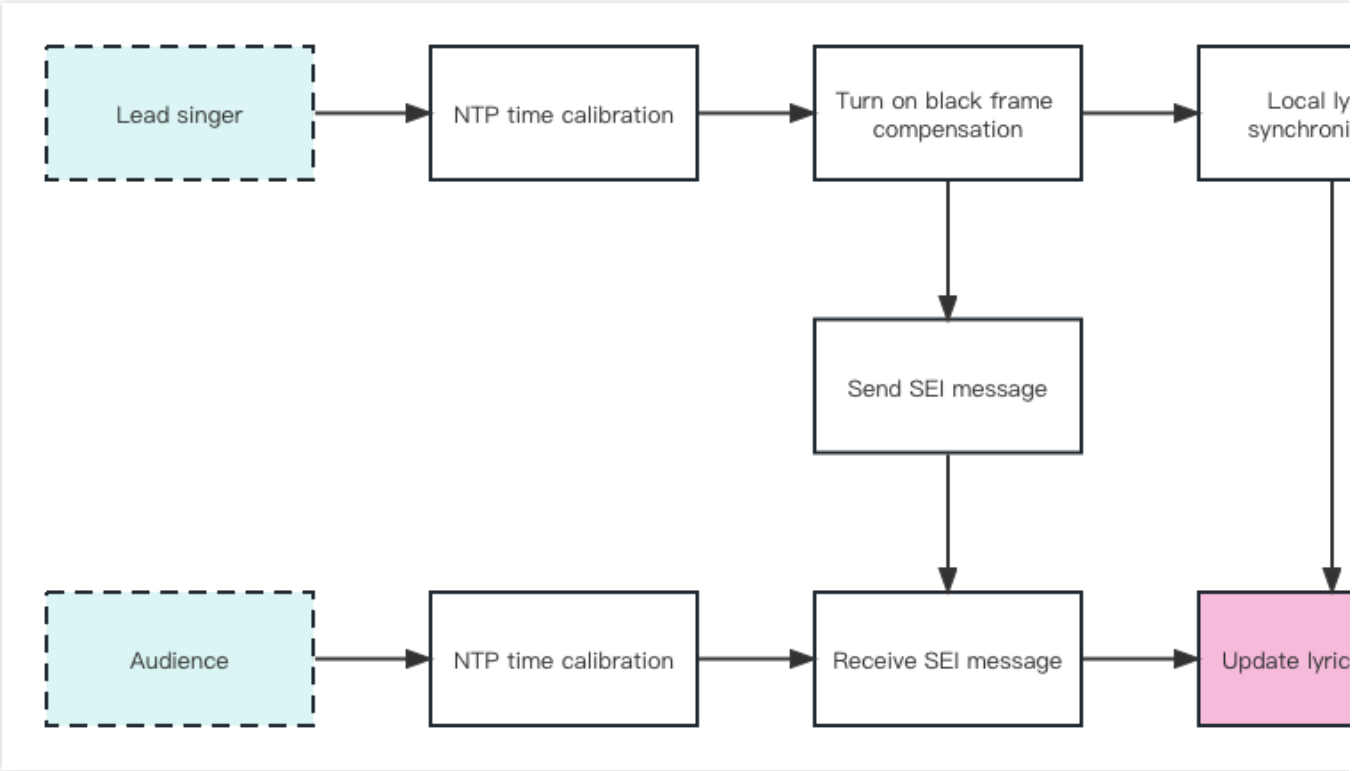
最近更新时间：2024-07-04 10:10:52

1.1 实现流程

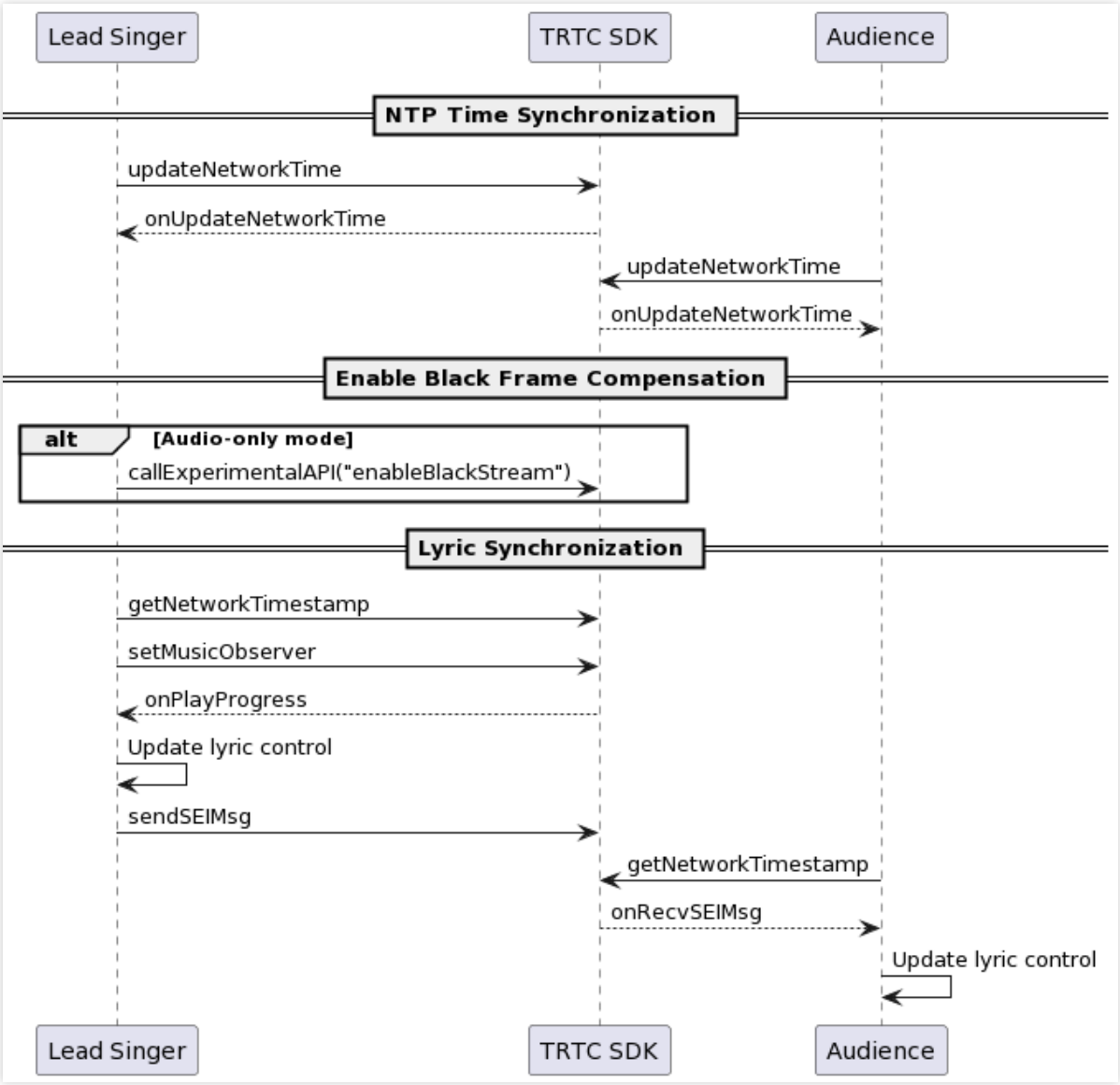
歌词同步方案中，三种不同角色的动作如下：

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。



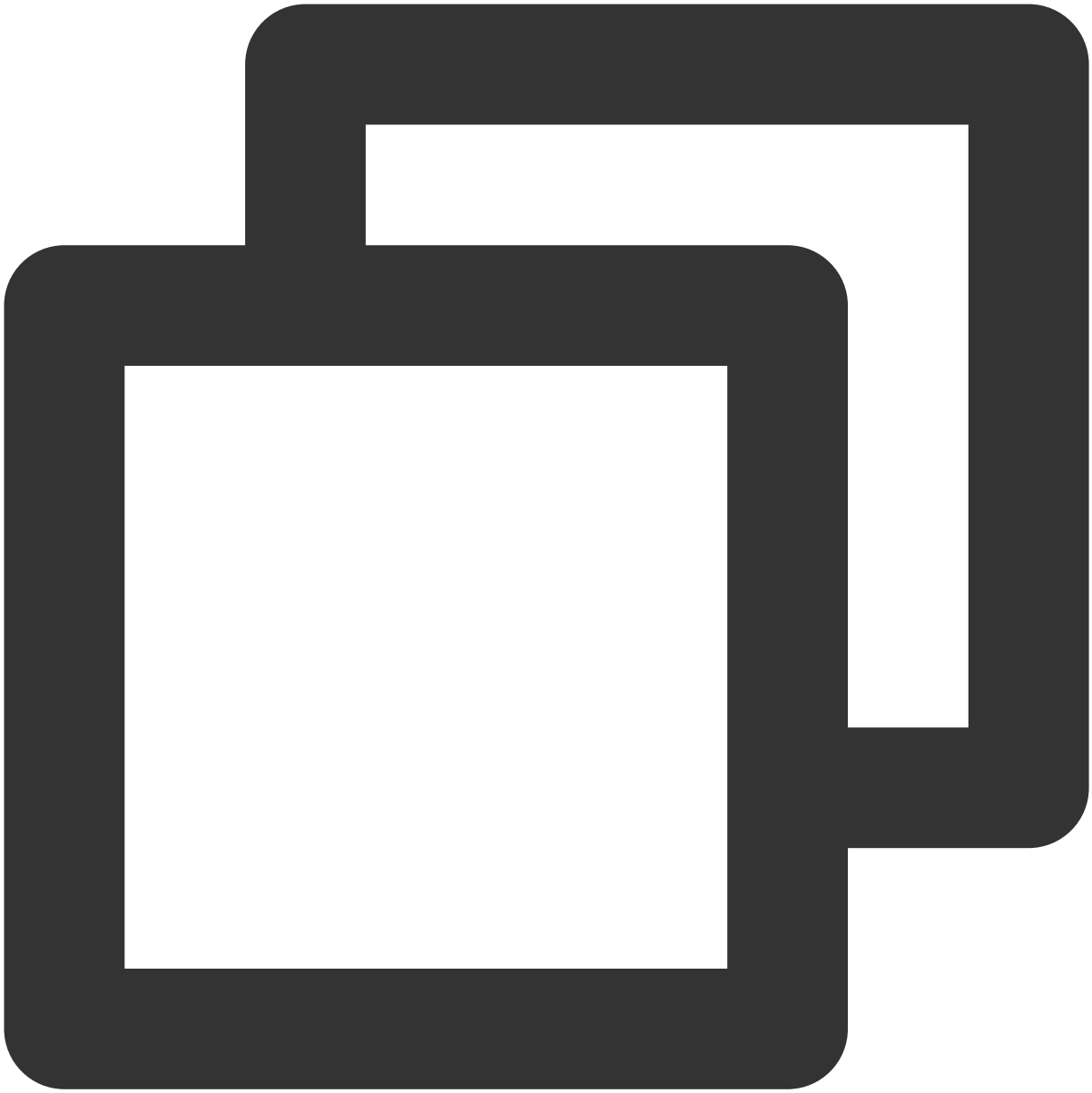
时序图



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:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[trtcCloud callExperimentalAPI:jsonString];
```

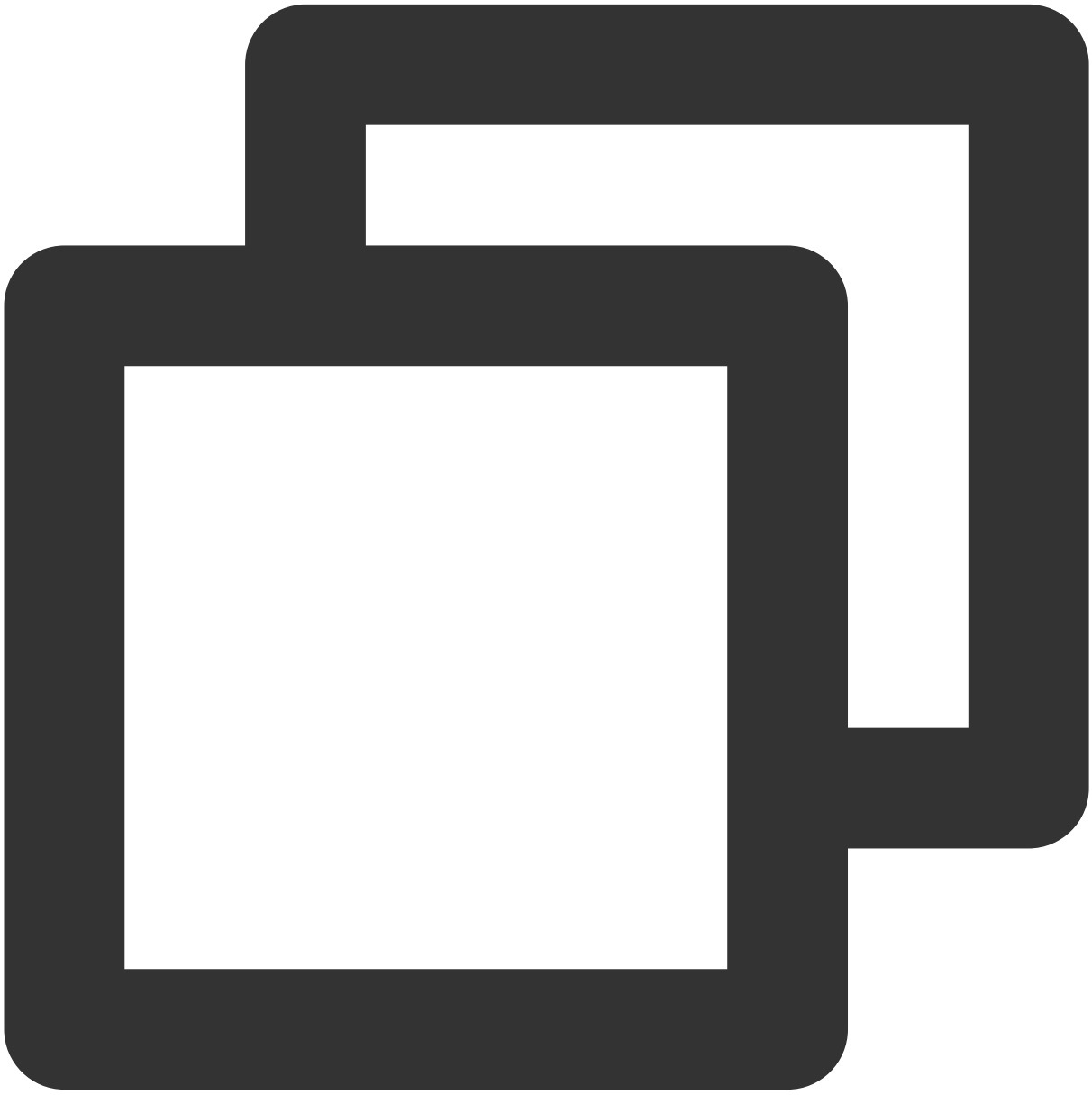
说明：

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 duratio
    //当前 ntp 时间
```

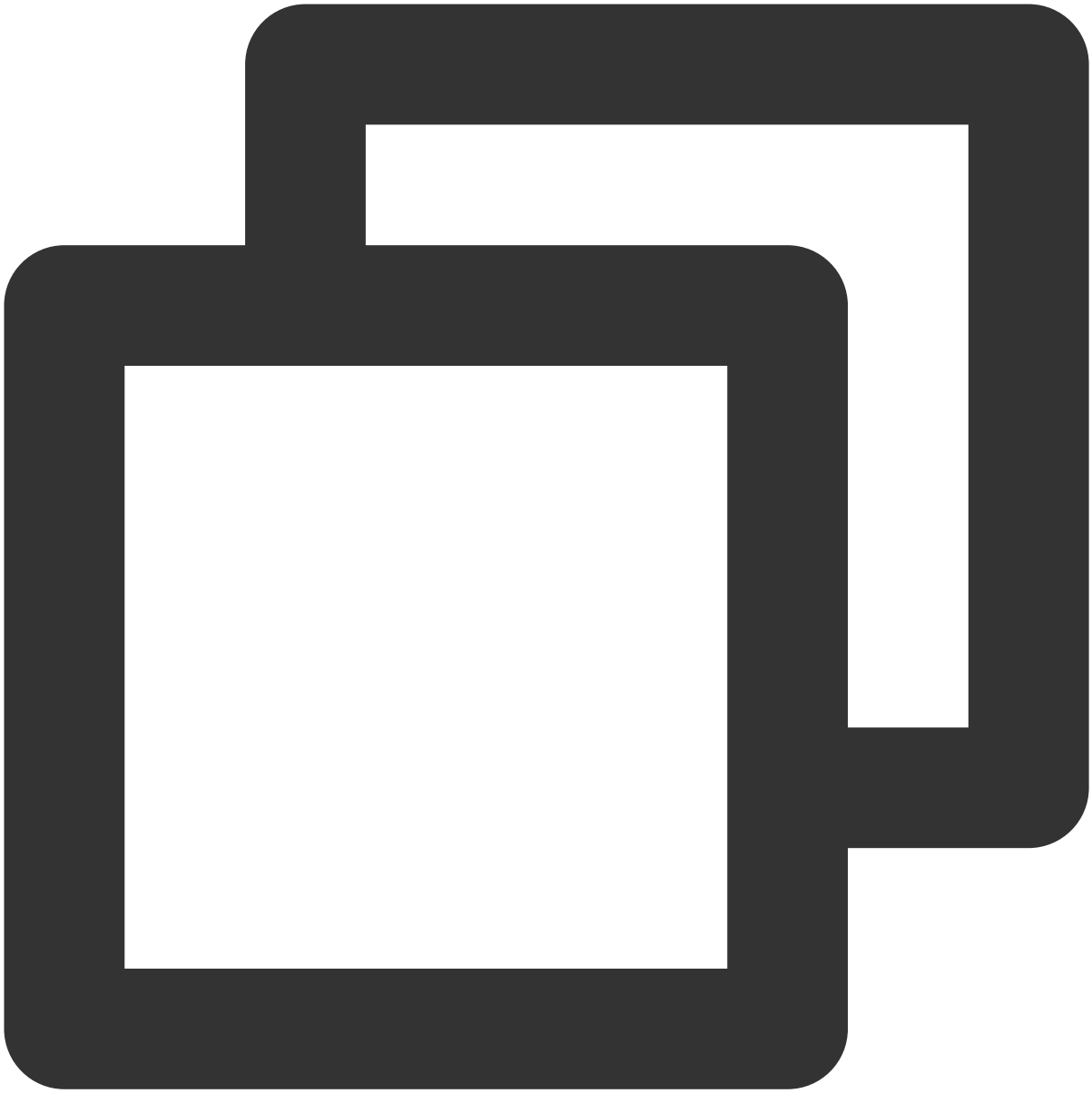
```
    NSInteger ntpTime = [TXLiveBase getNetworkTimestamp];
    //通知歌曲进度，用户会在这里进行歌词的滚动
    NSDictionary *progressMsg = @{
            @"bgmProgressTime":@(progressMs),
            @"ntpTime":@(ntpTime),
            @"musicId": @(musicId),
            @"duration": @(durationMs),
    };
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:progressMsg options:
    NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8S
    [trtcCloud sendSEIMsg:[jsonString dataUsingEncoding:NSUTF8StringEncoding] repea
};
```

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 duratio
    ...
    // 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:NSJ
    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。

Android

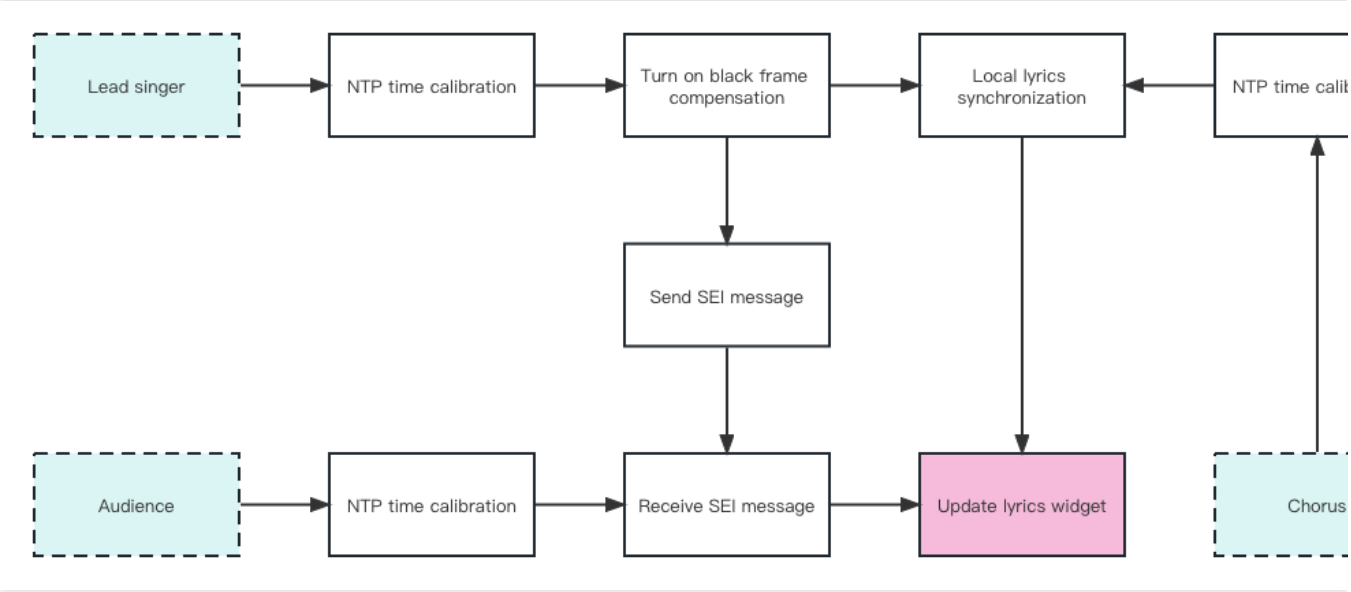
最近更新时间：2024-07-04 10:10:34

实现流程

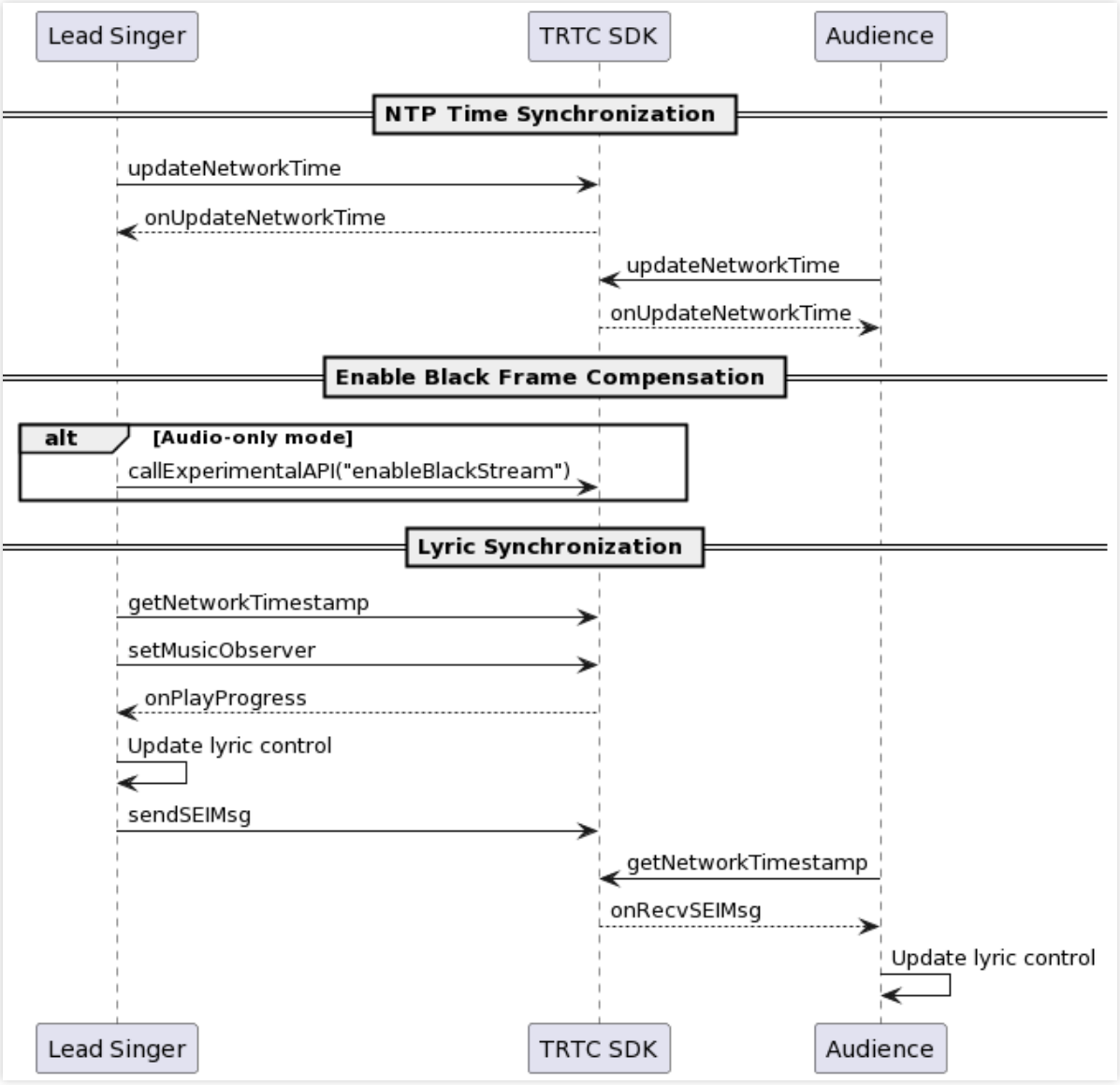
歌词同步方案中，三种不同角色的动作如下：

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。



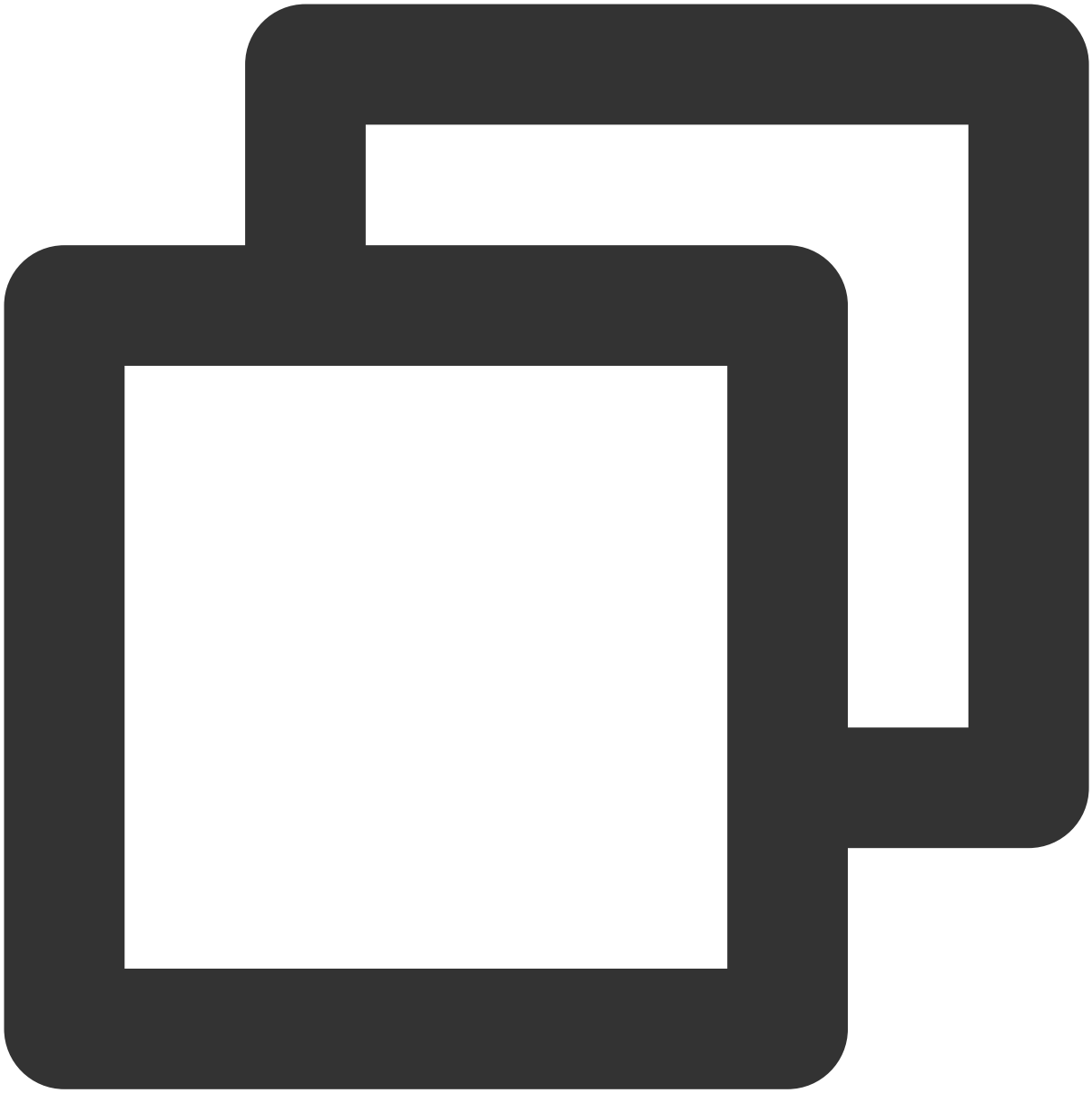
时序图



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\\": {\
```

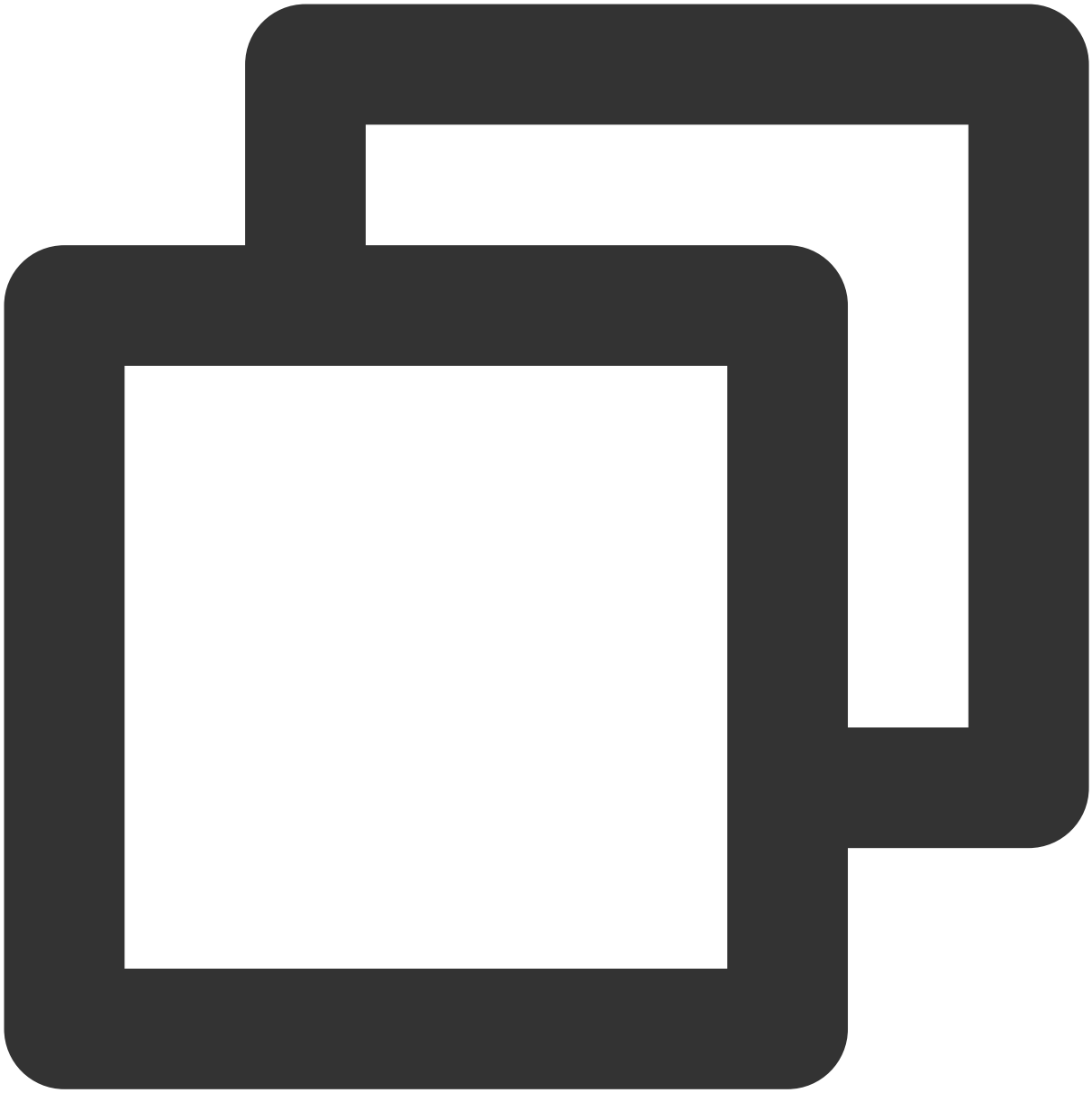
说明：

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.TXMu
    @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();
```

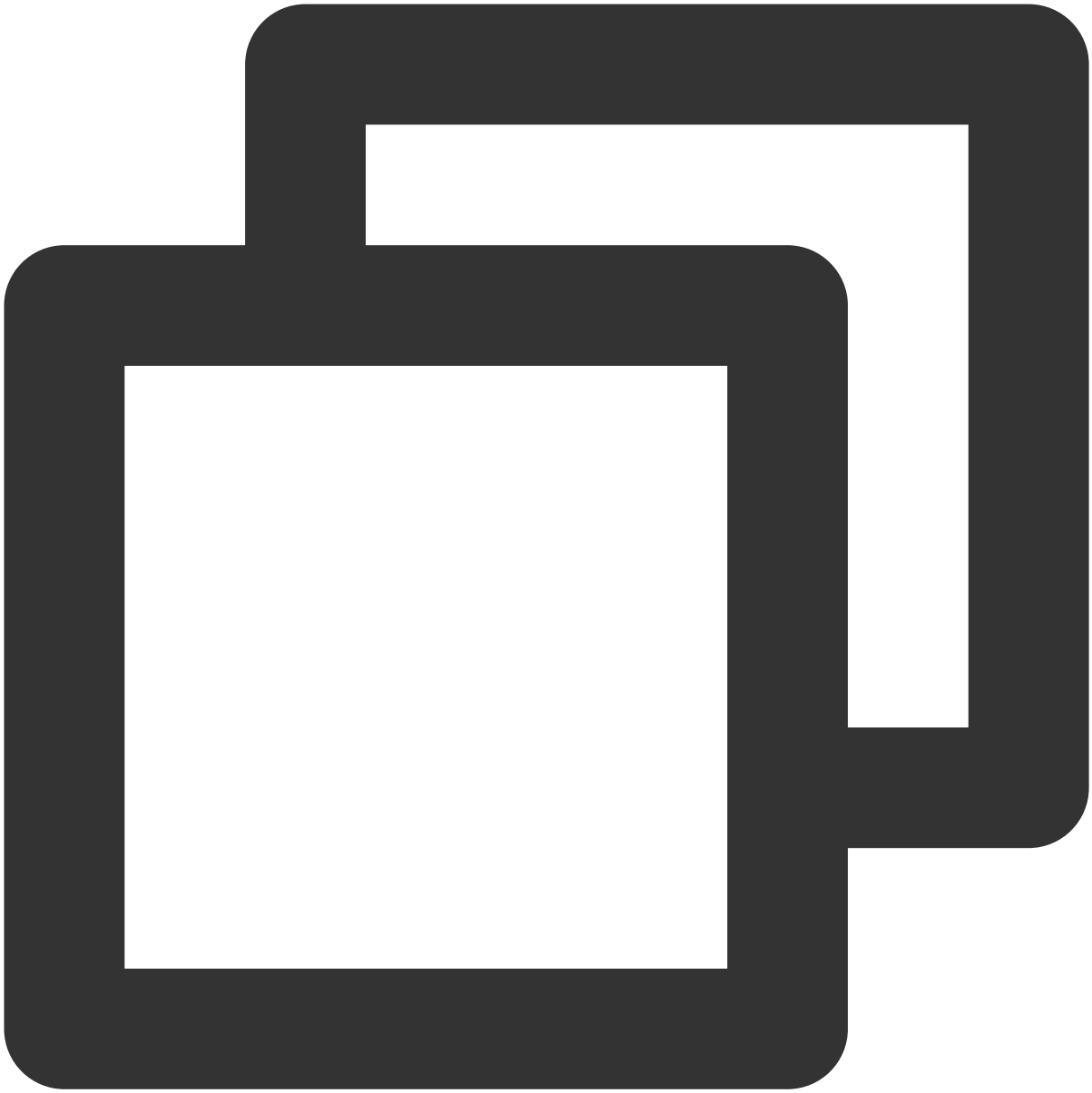
```
            mTRTCCloud.sendSEIMsg(jsonObject.toString().getBytes(), 1);
        }
    }
```

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.TXMu
    @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。

人声同步
iOS

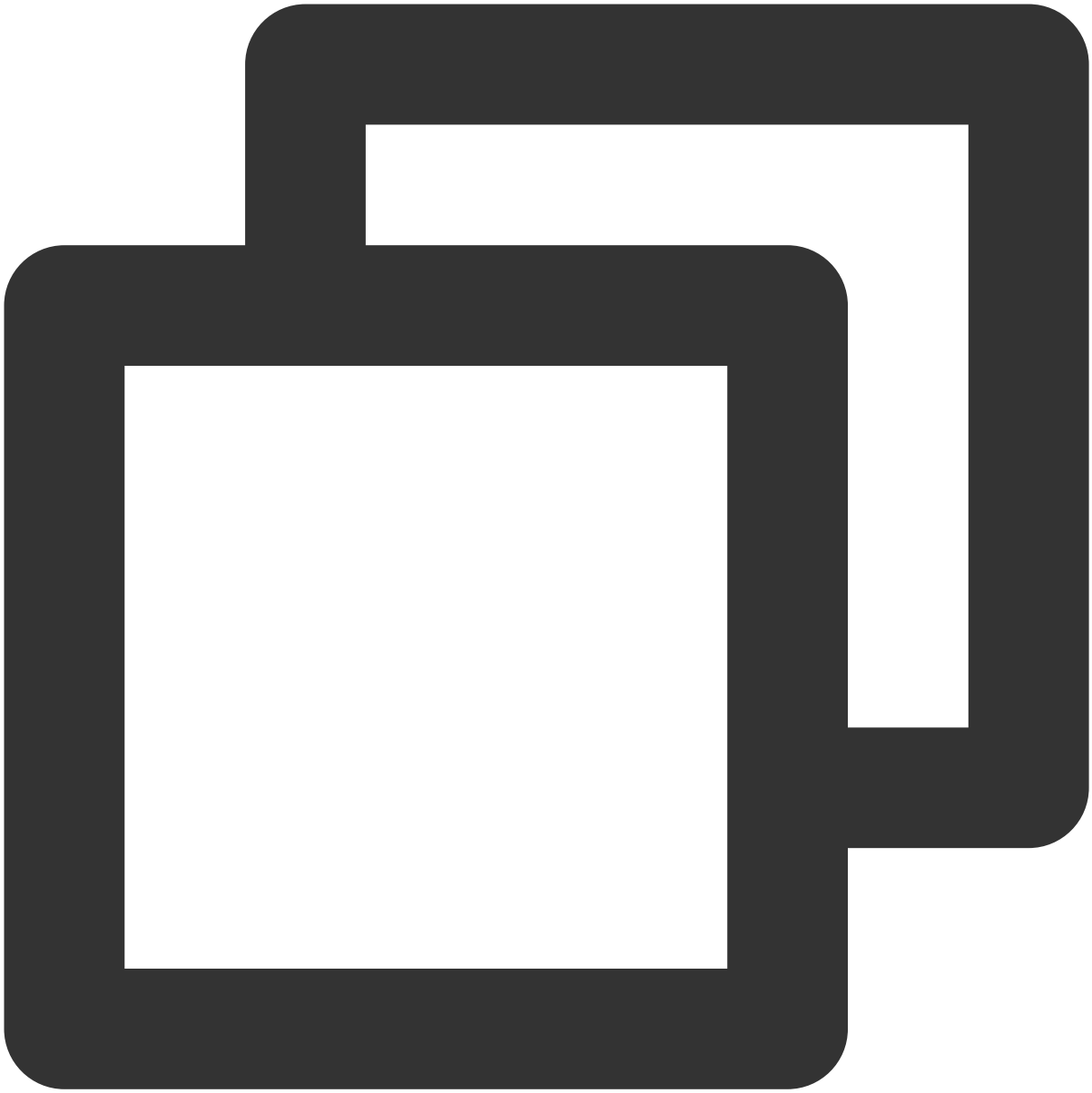
最近更新时间：2023-09-26 16:53:03

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:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[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:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
```

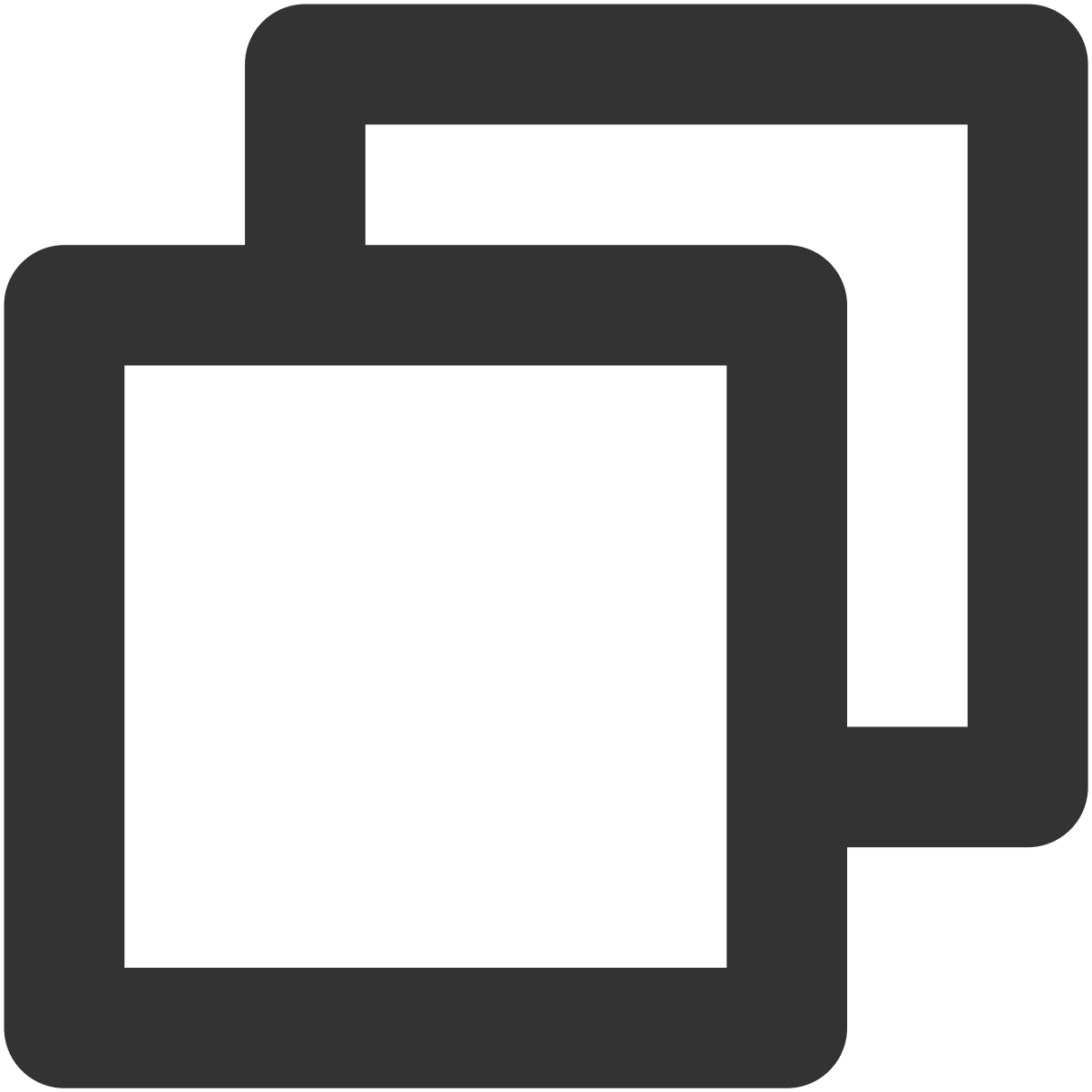
说明：

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:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[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:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[subCloud callExperimentalAPI:jsonString];
```

Android

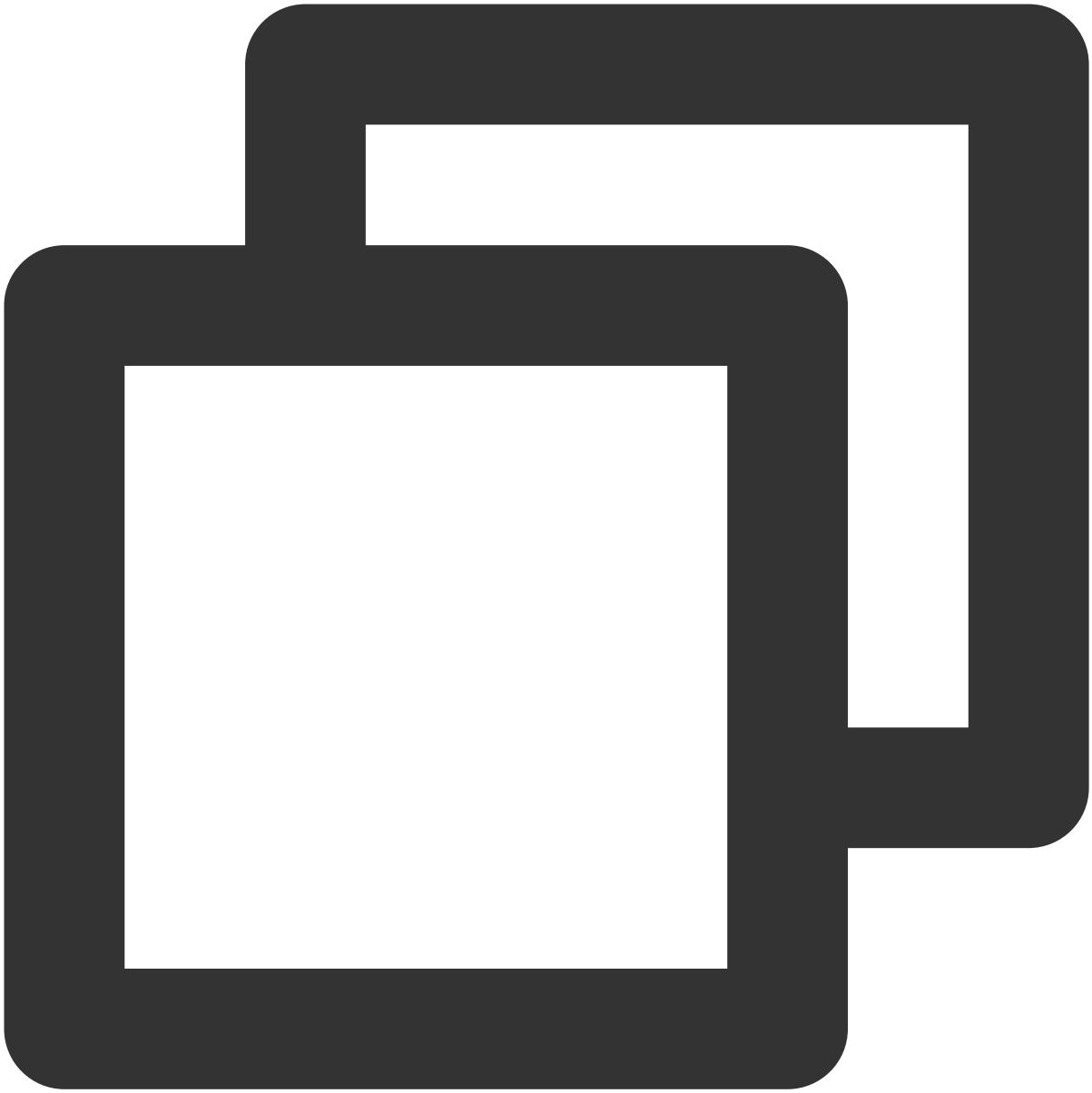
最近更新时间：2023-09-26 16:53:25

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\\":{\\"enab
// 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
```

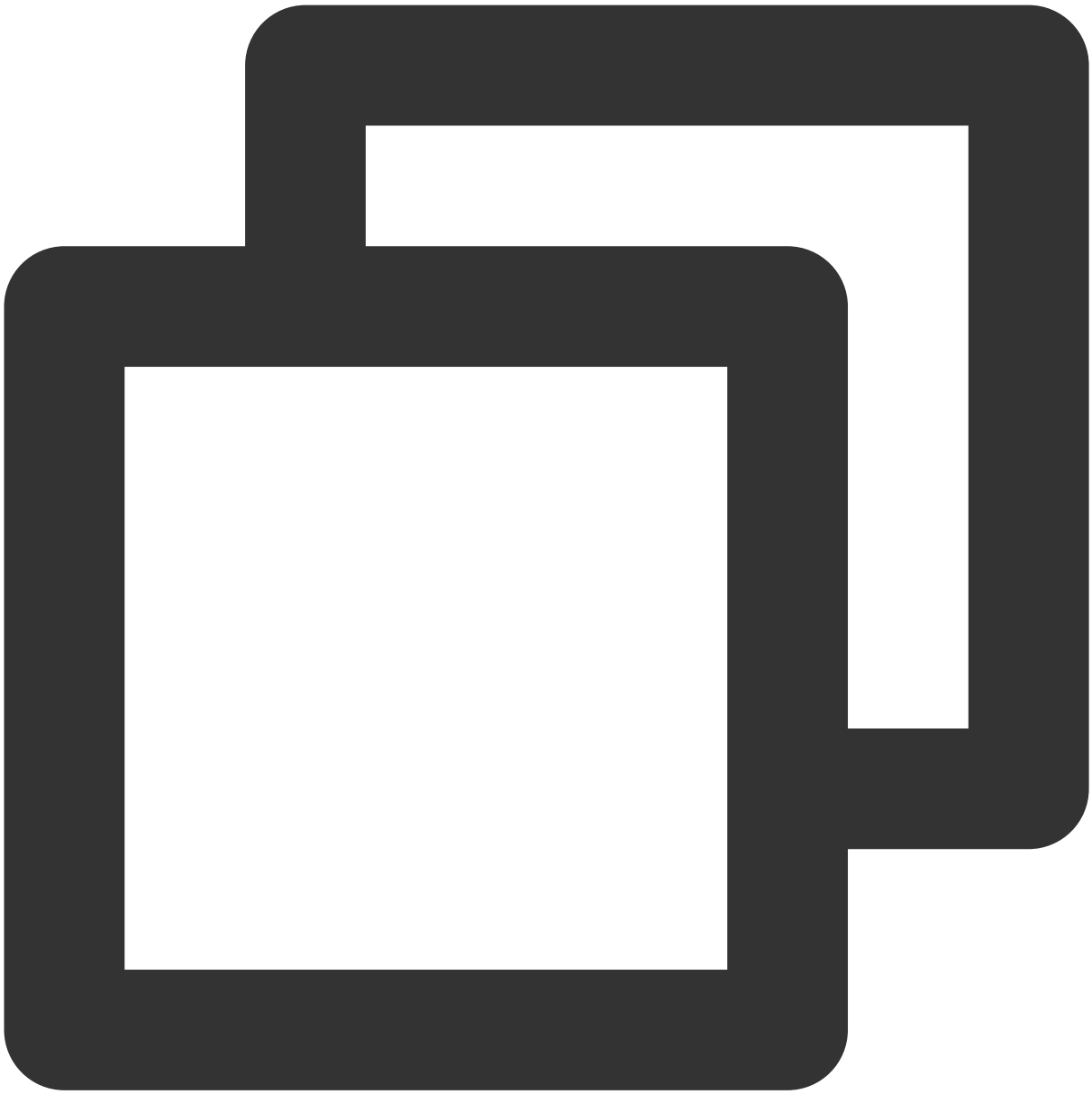
说明：

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
// 子实例（伴奏实例）开启高性能音频 AAudio
subCloud.callExperimentalAPI("{\\"api\\":\\"setLowLatencyModeEnabled\\",\\"params\\
```

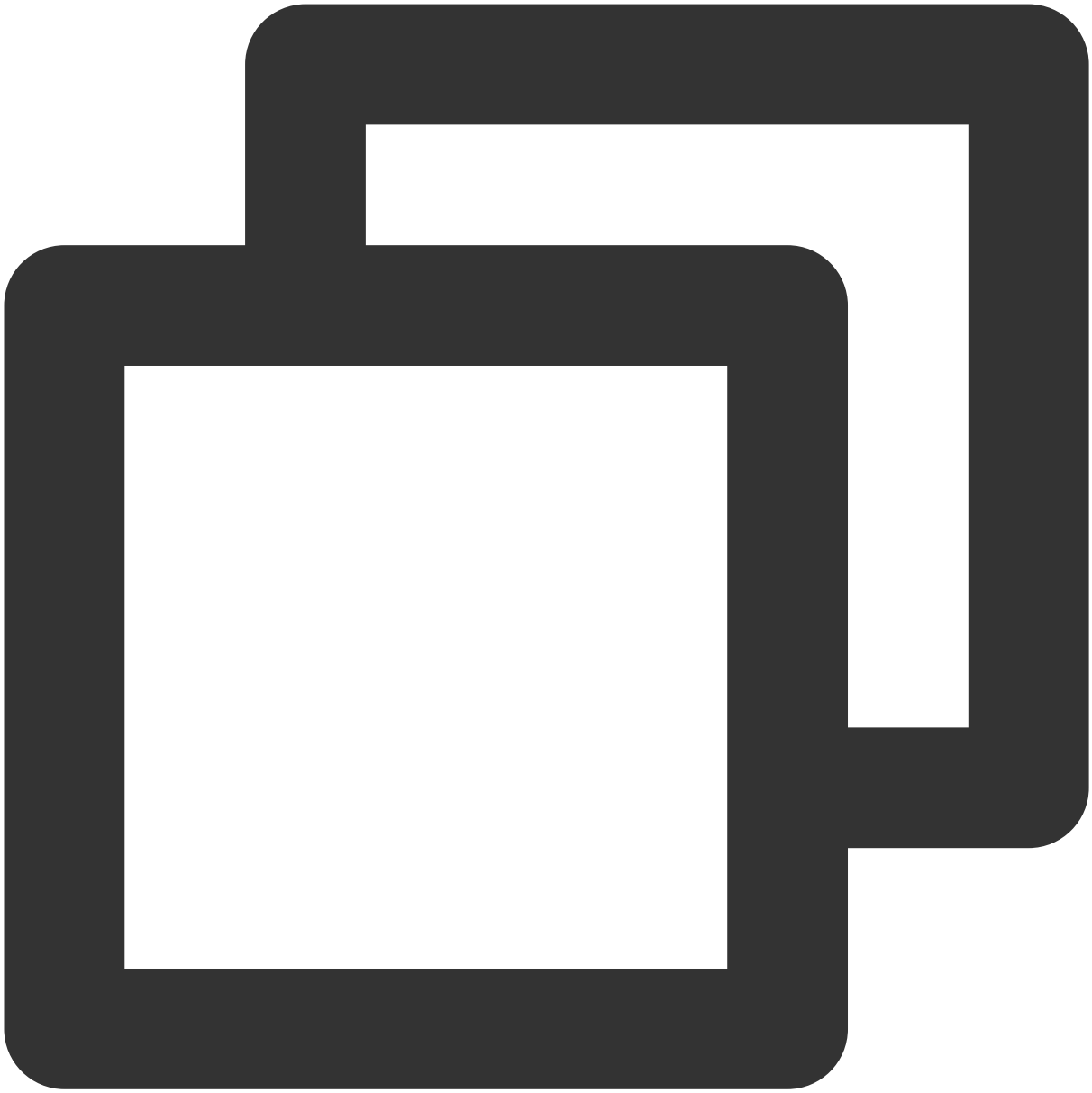
混流方案

iOS

最近更新时间：2023-09-26 16:53:49

具体代码实现

1. 创建主子实例

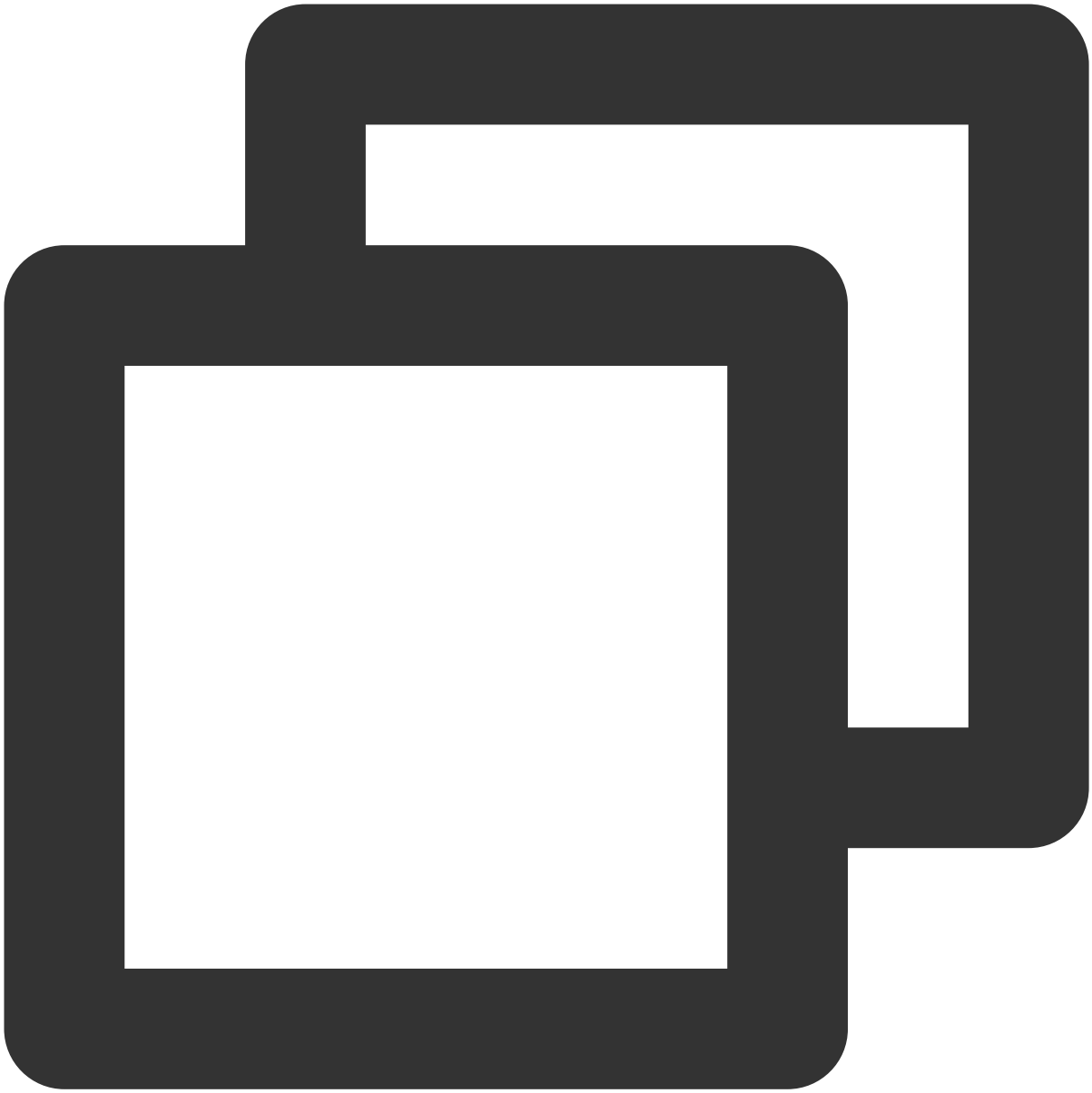


```
// 创建 TRTCCloud 主实例（人声实例）
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
// 创建 TRTCCloud 子实例（伴奏实例）
TRTCCloud *subCloud = [trtcCloud createSubCloud];
```

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];
```

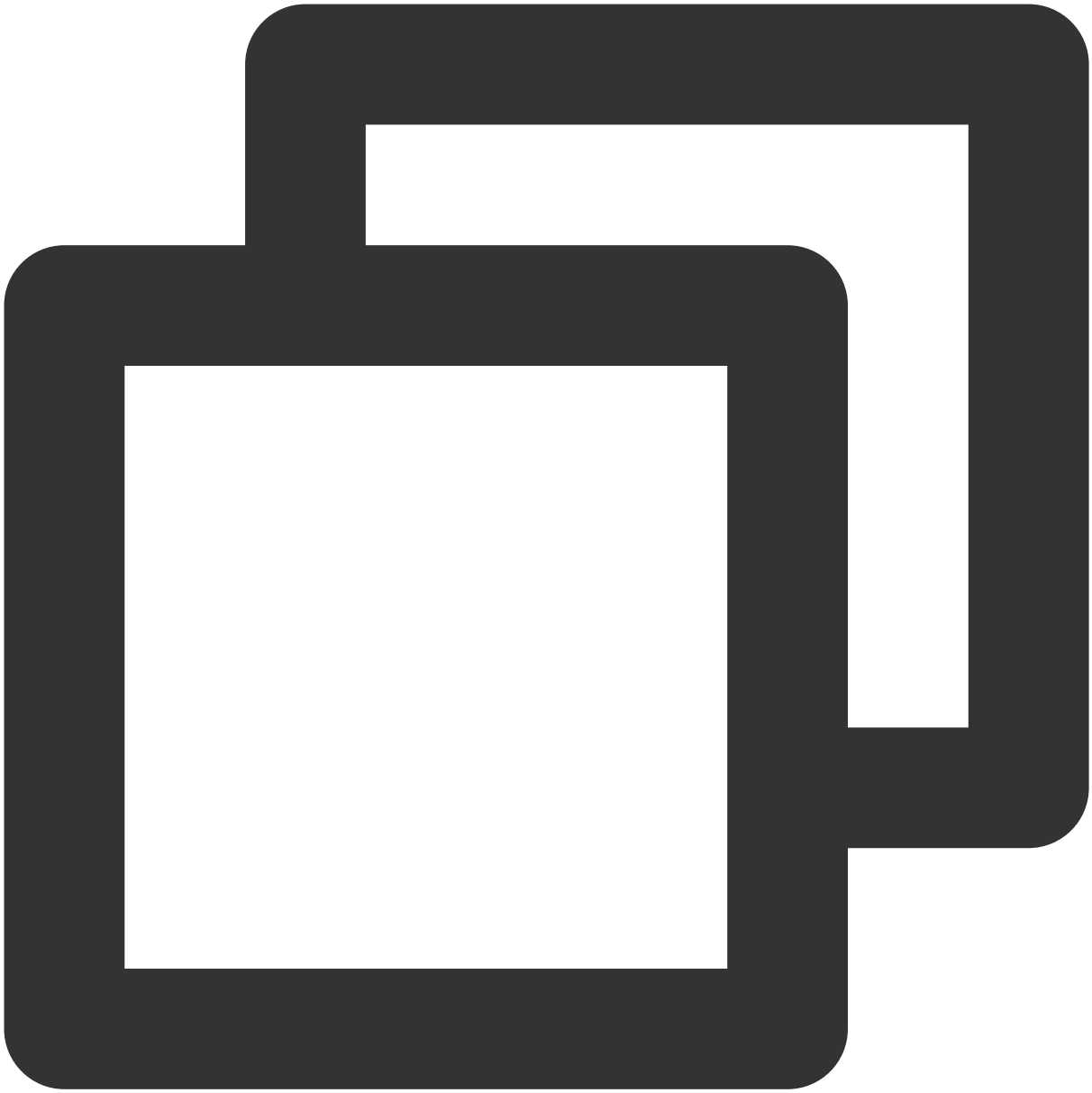
注意：

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
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
[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 onPr
```

注意：

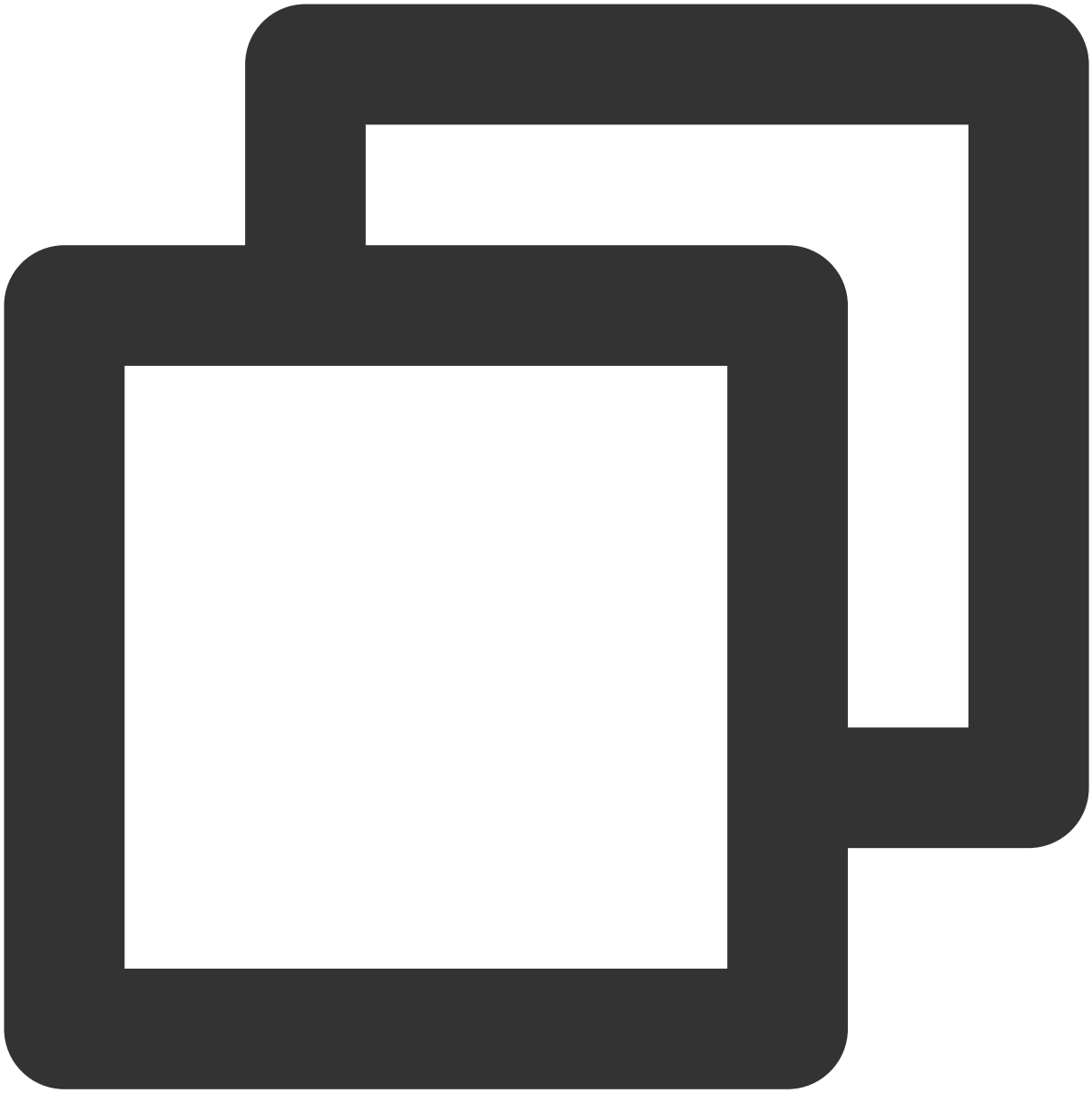
注意区分主实例和子实例的 userId，确保不重复且容易辨别。

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 mi
```

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。

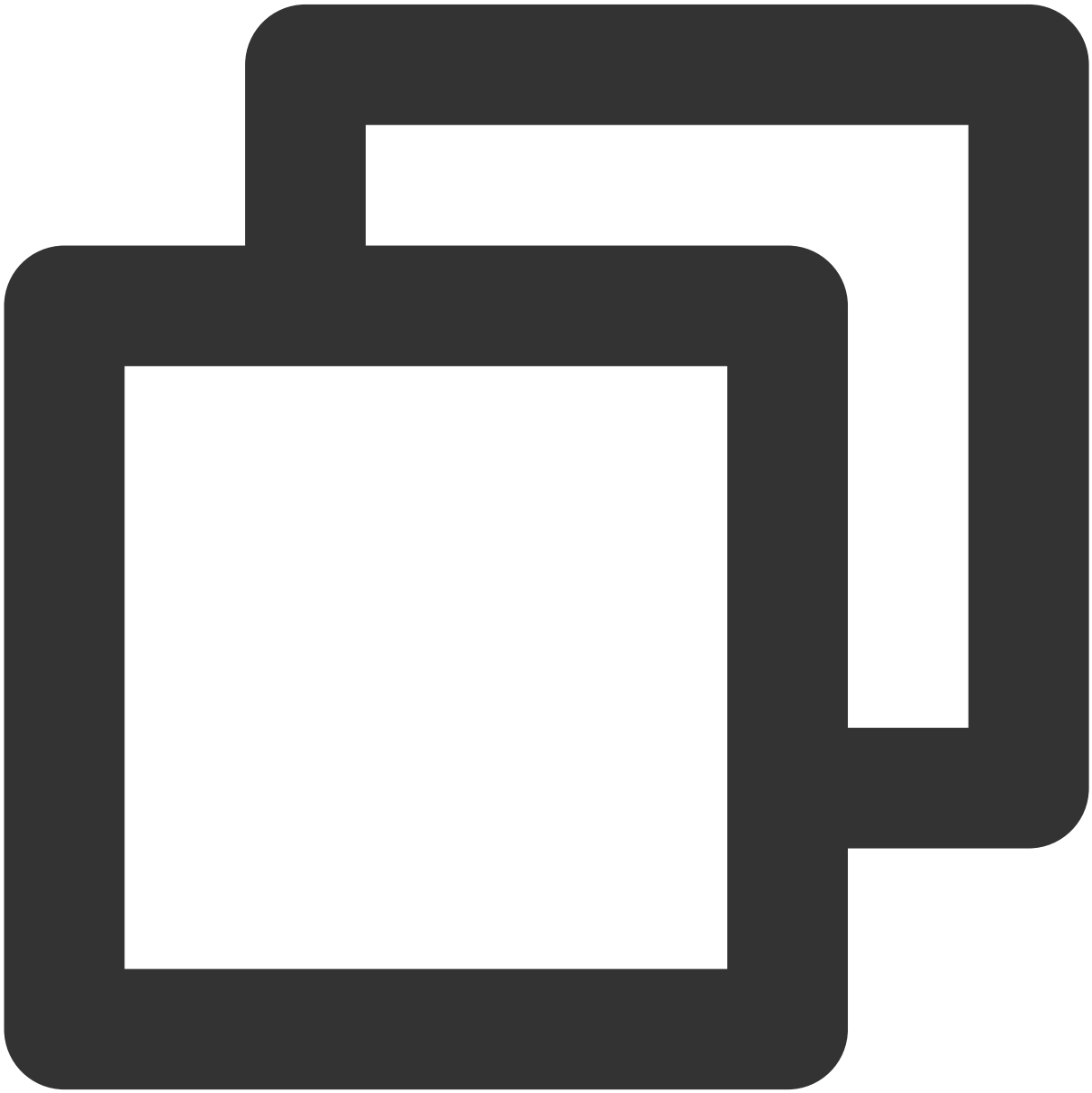
这里的混流转推方法 startPublishMediaStream 采用全新的后台架构，旧版应用需提供 SdkAppId 申请升级后方可使用。

Android

最近更新时间：2023-09-26 16:54:20

具体代码实现

1. 创建主子实例



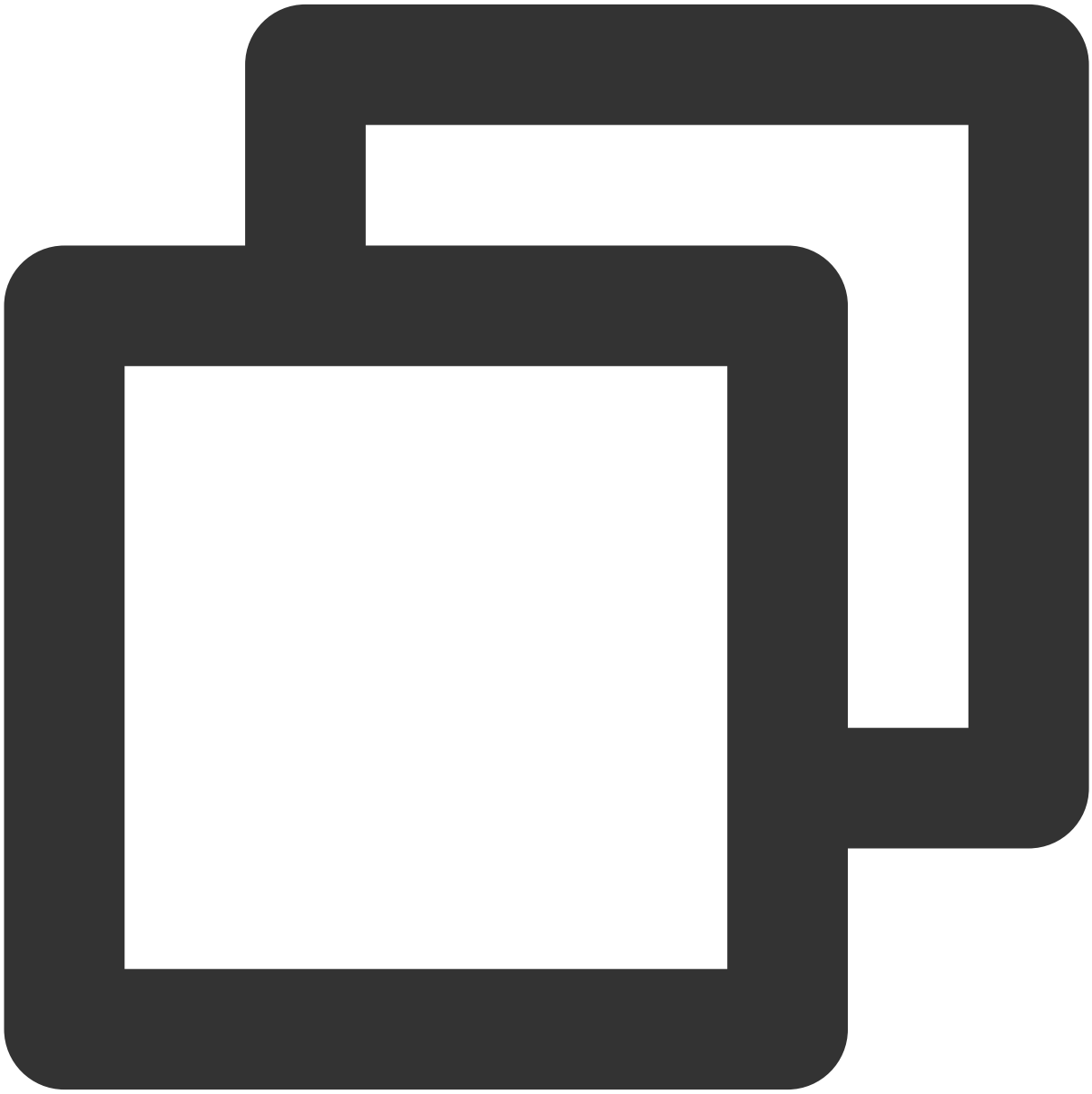
```
// 创建 TRTCCloud 主实例（人声实例）
```

```
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);
```

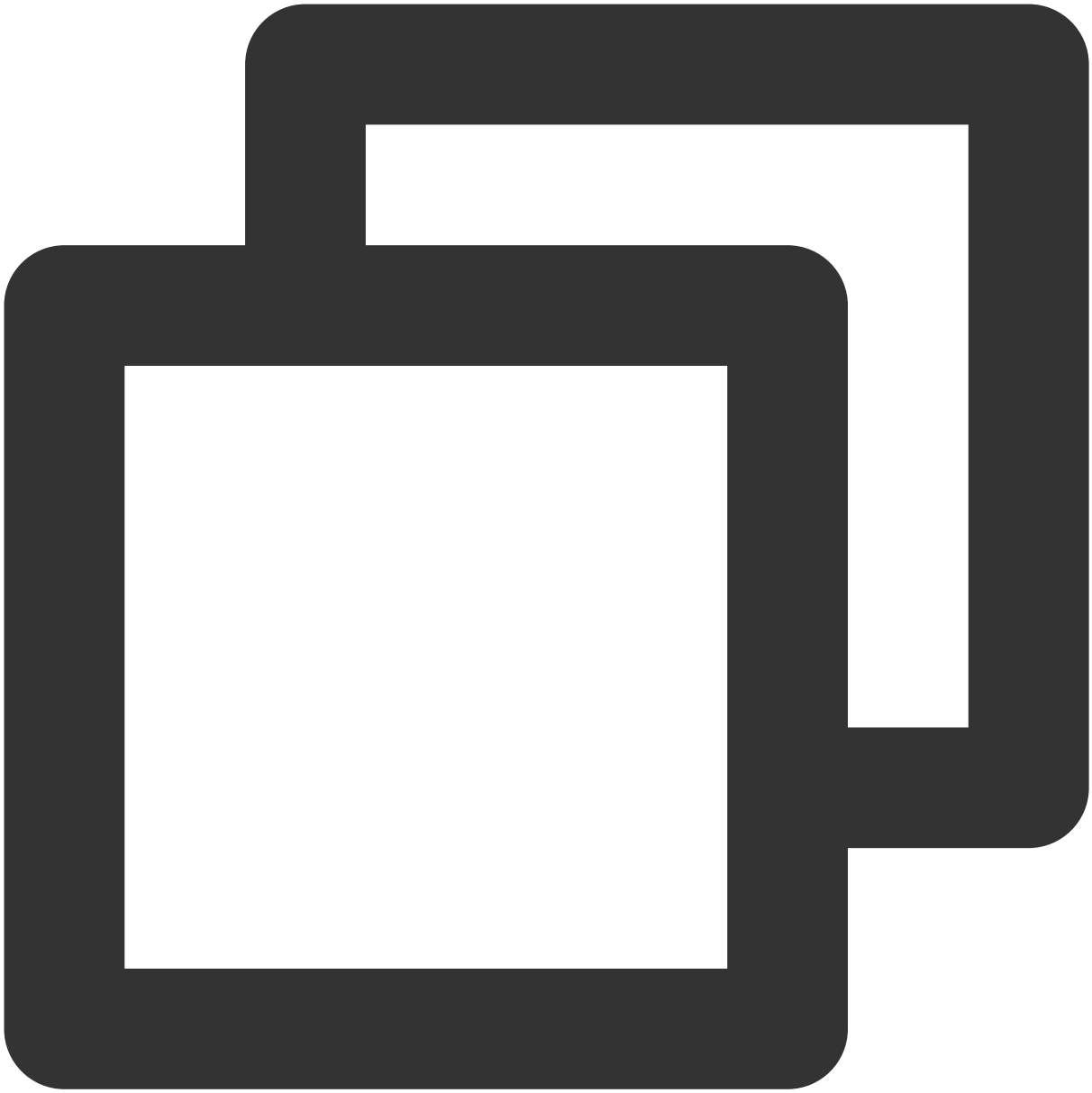
注意：

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\\": {\\"music
// 播放伴奏音乐并推流（在约定时间播放）
TXAudioEffectManager.AudioMusicParam param = new TXAudioEffectManager.AudioMusicPar
// 将伴奏音乐传到远端
param.publish = true;
subCloud.getAudioEffectManager().startPlayMusic(param);
```

注意：

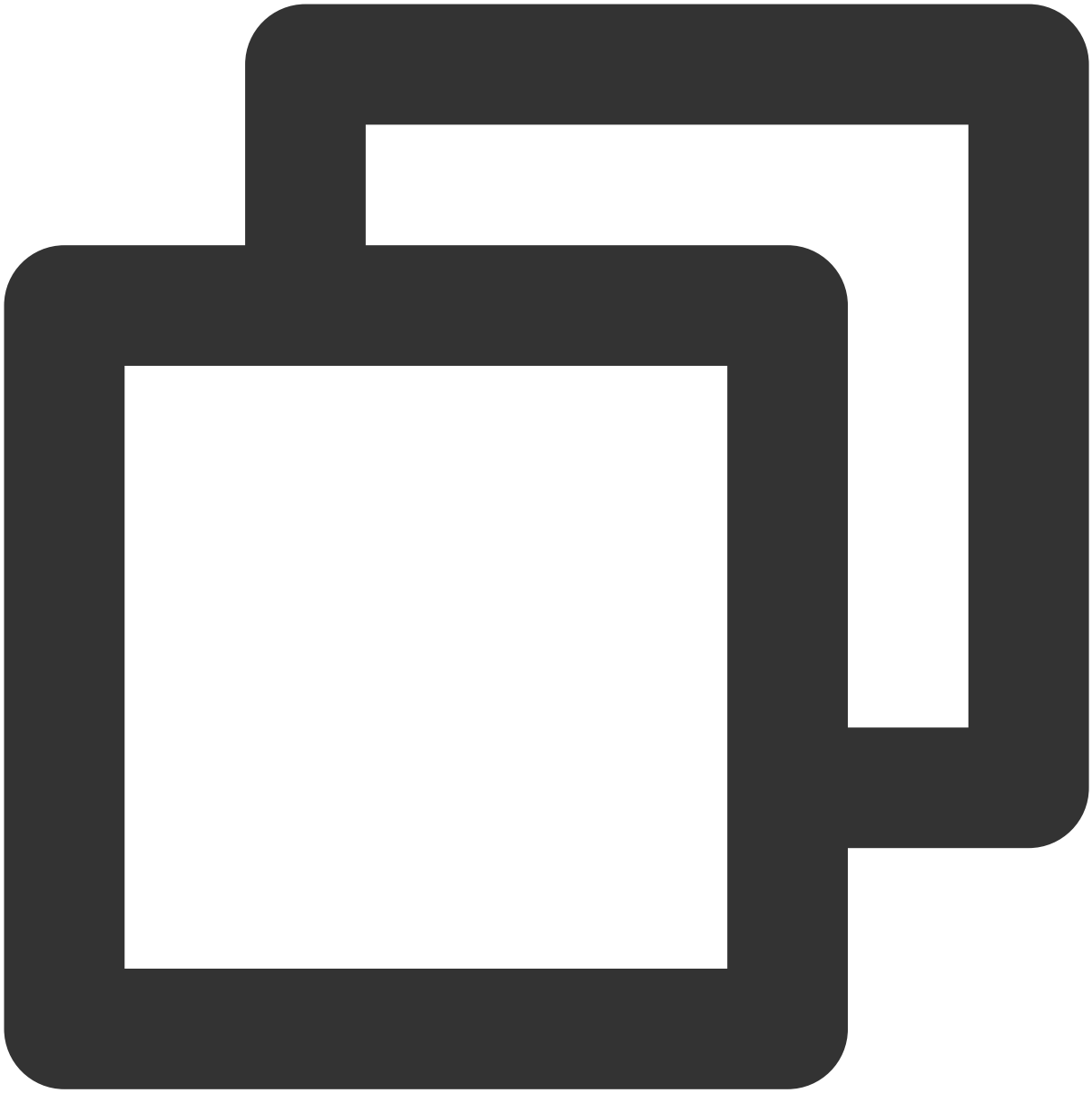
注意区分主实例和子实例的 userId，确保不重复且容易辨别；

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.TRTCS
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.TRTCS
// 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, trtcStreamMixing
```

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）

TRTCKaraoke(iOS)

最近更新时间：2022-07-22 15:13:44

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 Karaoke（iOS）。

- 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。

API	描述
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）。

API	描述
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。

API	描述
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。

API	描述
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。

API	描述
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 TRTC
KaraokeRoom#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，默认发送动主线程 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:a
```

```
ccompanyUrl:));
```

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)callbac
k 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:(ActionCallbac
k _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(setAudioCaptureVolu
me(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:user
Id: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 _Nullabl
e)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

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:invite
e:));
```

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。

参数	类型	含义
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。

TRTCKaraoke(Android)

最近更新时间：2022-07-22 15:13:44

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 Karaoke（Android）。

- 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。

API	描述
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）。

API	描述
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。

API	描述
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。

API	描述
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

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

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.Acti
onCallback 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。

参数	类型	含义
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：

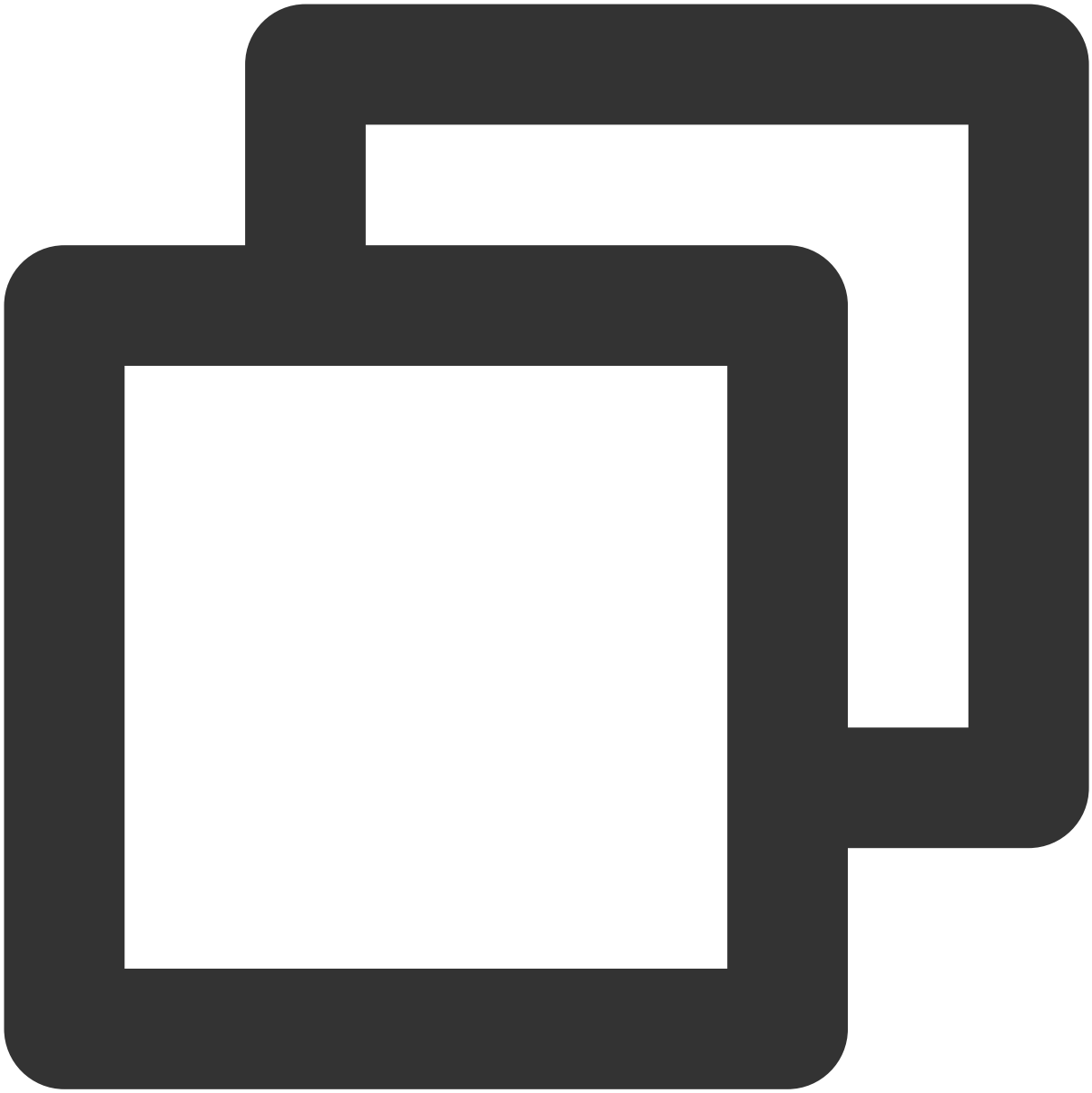
参数	类型	含义
musicID	int	播放时传入的 musicID。

常见问题（TUIKaraoke）
iOS

最近更新时间：2023-09-26 17:01:20

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？



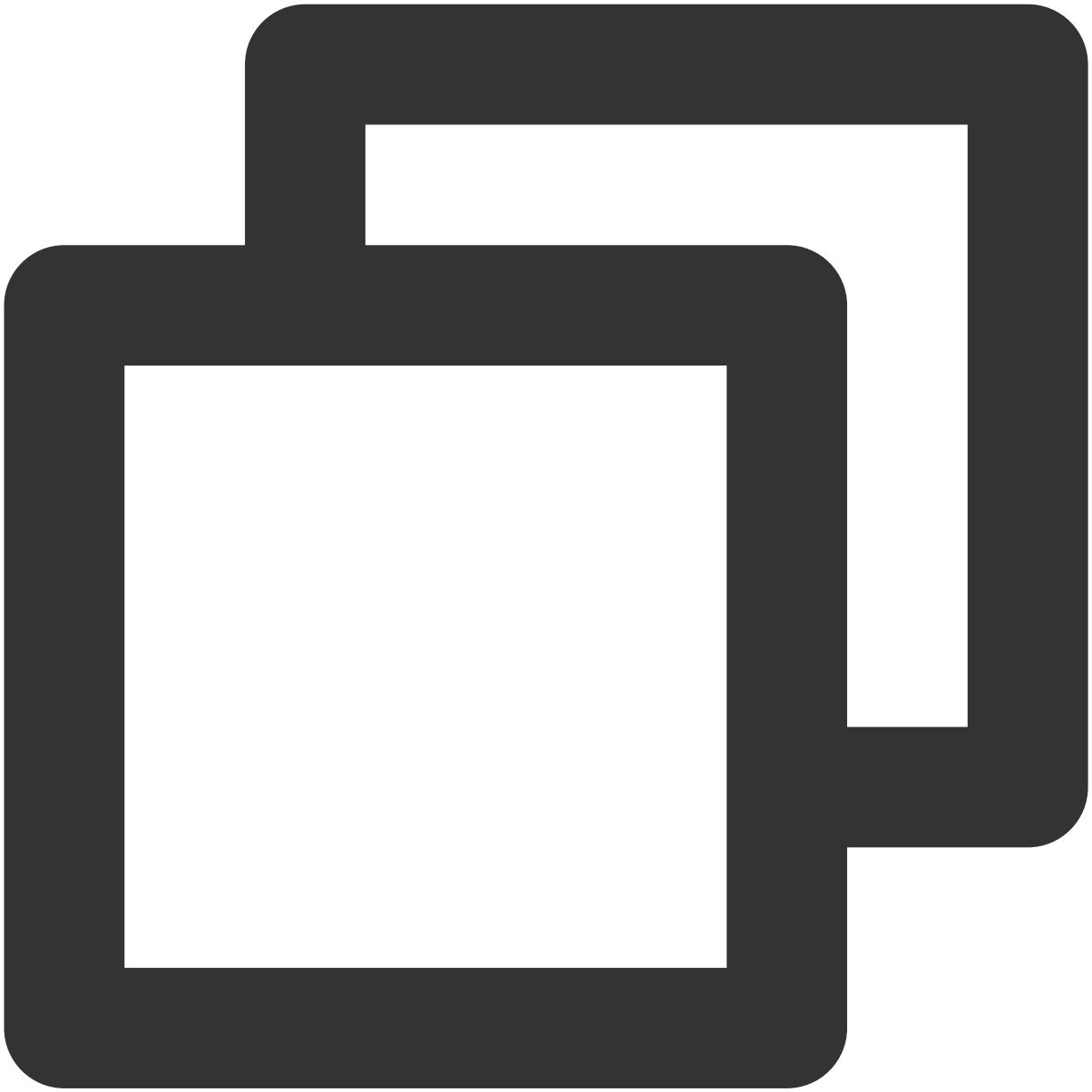
```
[[trtcCloud getAudioEffectManager] enableVoiceEarMonitor:YES];
```

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:NSJSONWr
NSString *jsonString = [[NSString alloc] initWithData:jsonData encoding:NSUTF8Strin
```

```
[trtcCloud callExperimentalAPI:jsonString];
```

NTP 校时问题

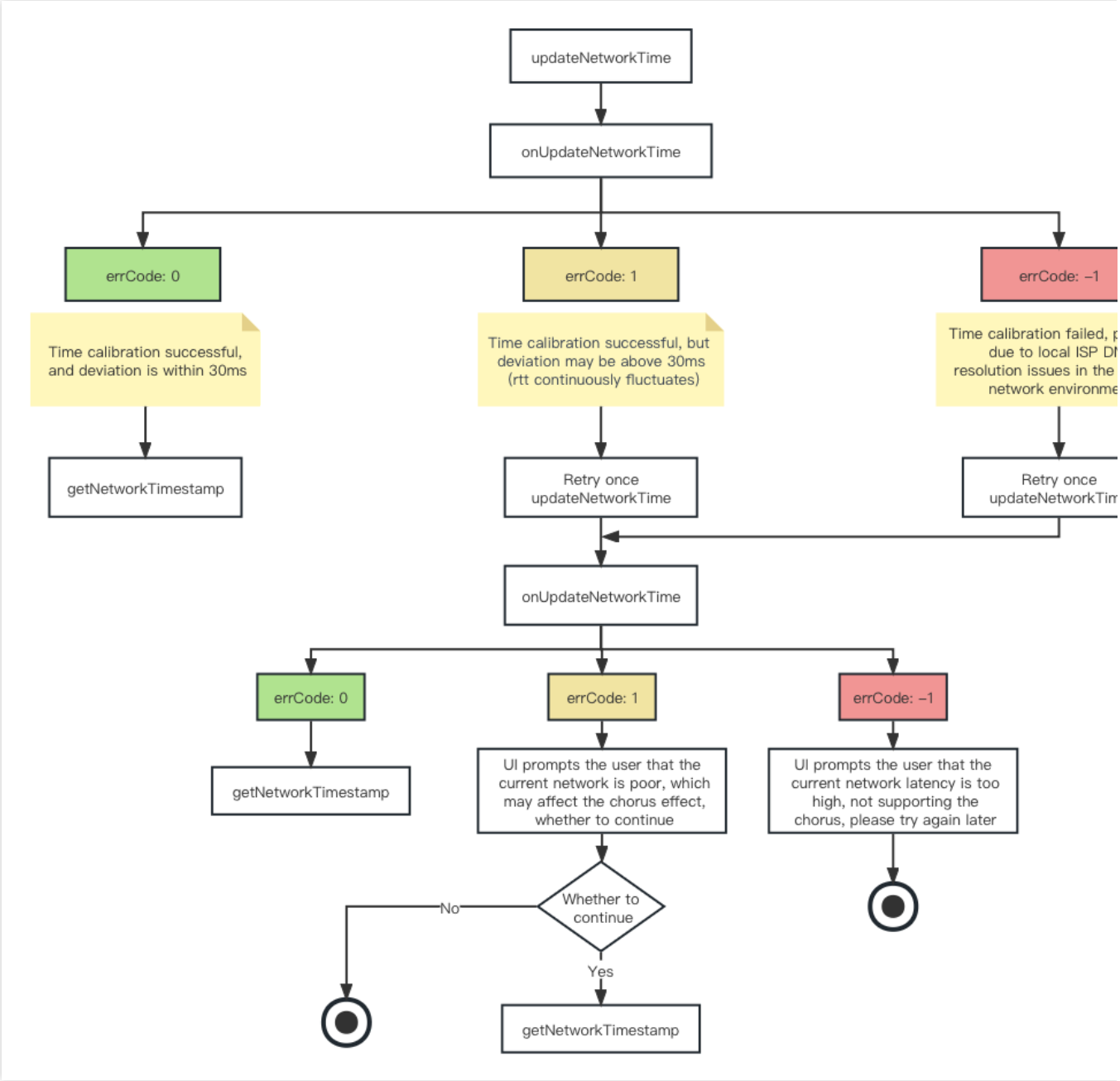
提醒：“NTP time sync finished, but result maybe inaccurate”？

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。

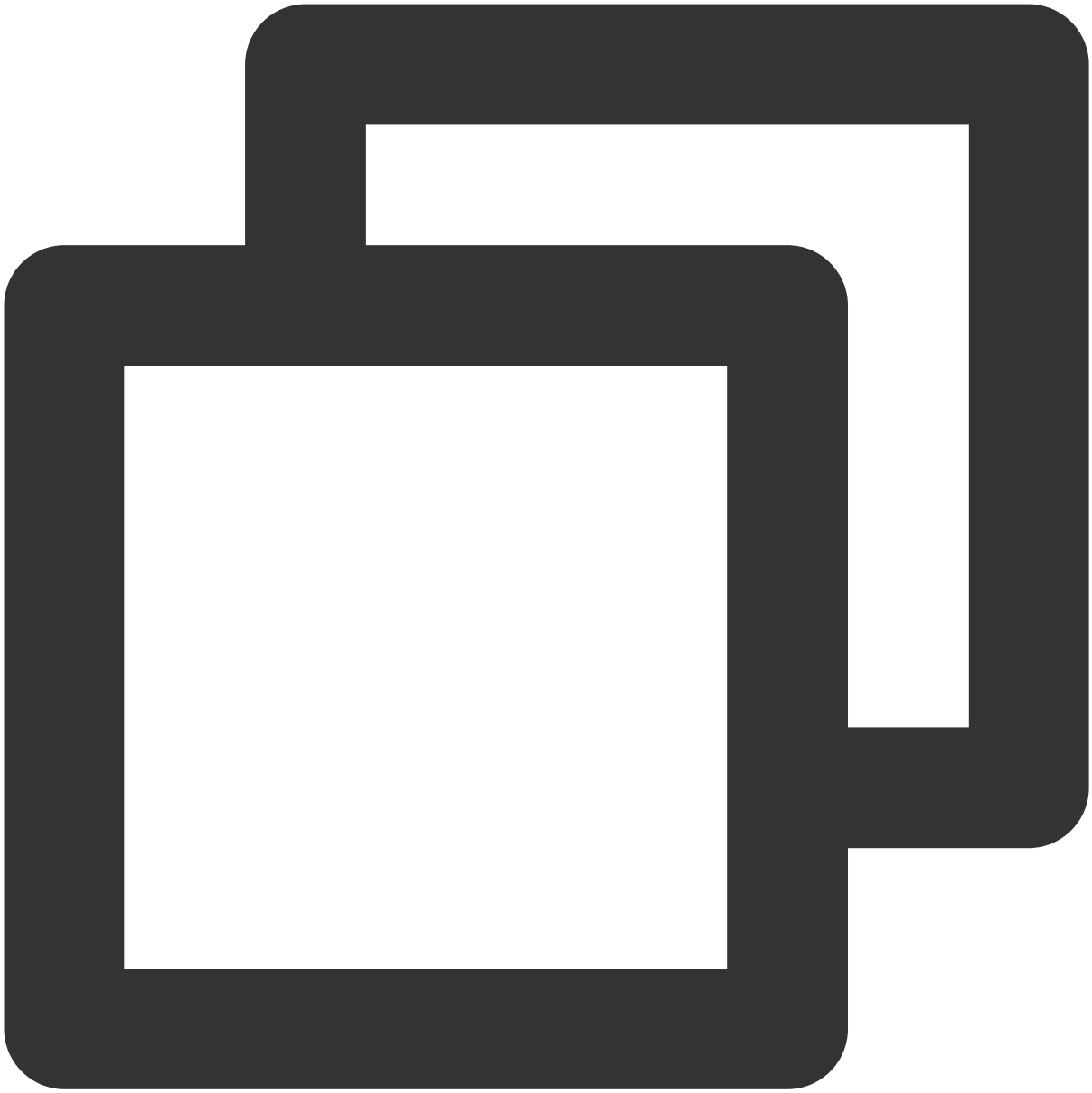
NTP 服务重试处理逻辑？



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。

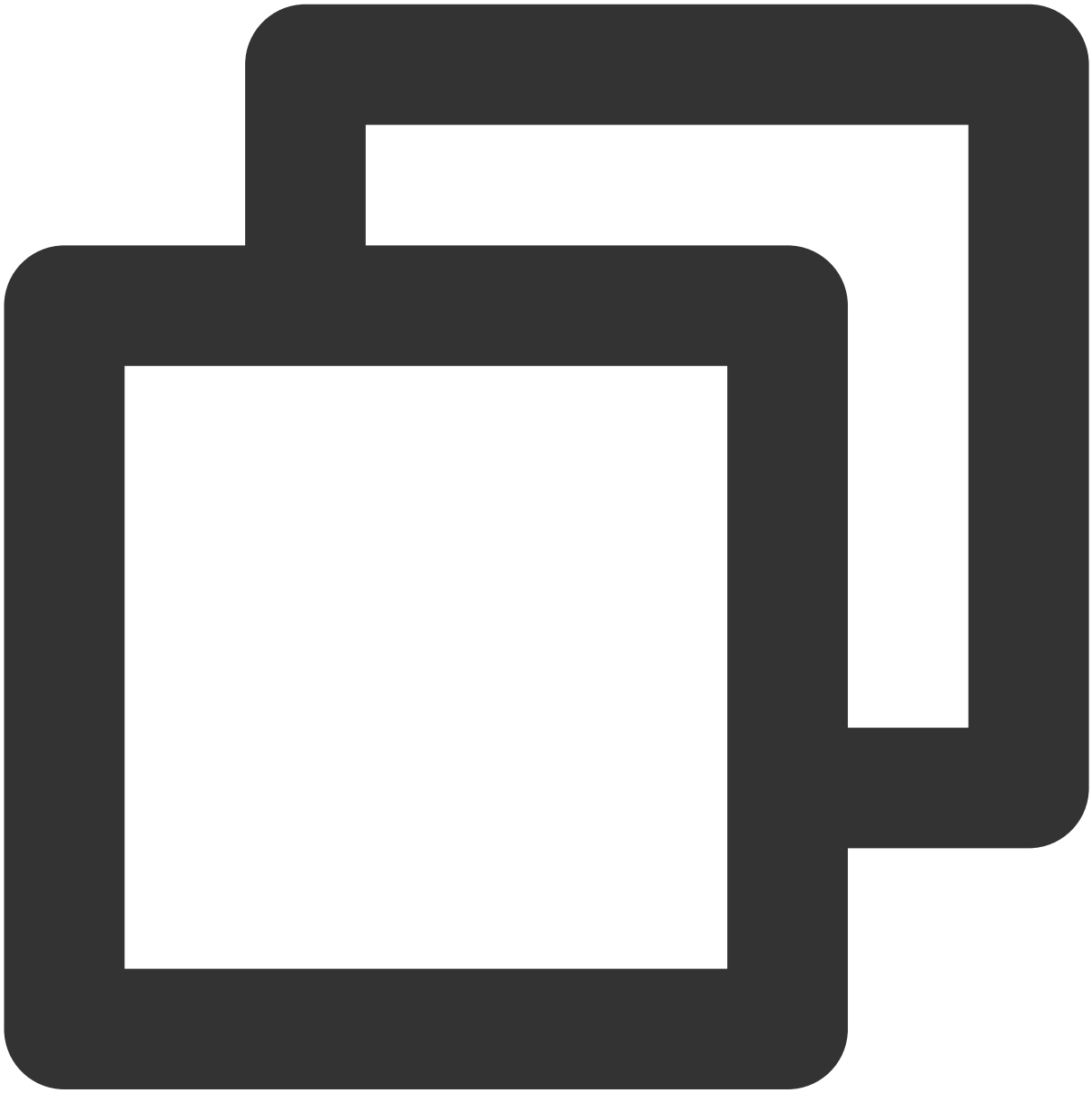
TRTC SDK 网络测速的发起：



```
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", re
        [mTextTestResult addObject:[NSString stringWithFormat:@"上行丢包率：%f \\n",
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行丢包率：%f \\n",
        [mTextTestResult addObject:[NSString stringWithFormat:@"网络延迟：%u ms \\n",
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行带宽：%ld kbps \\
        [mTextTestResult addObject:[NSString stringWithFormat:@"上行带宽：%ld kbps \\
        [mTextTestResult addObject:[NSString stringWithFormat:@"下行带宽：%@ \\n", tq
    } else {
        [mTextTestResult addObject:@"测速成功！\\n"];
        [mTextTestResult addObject:[NSString stringWithFormat:@"errMsg：%@ \\n", res
    }
}
```

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

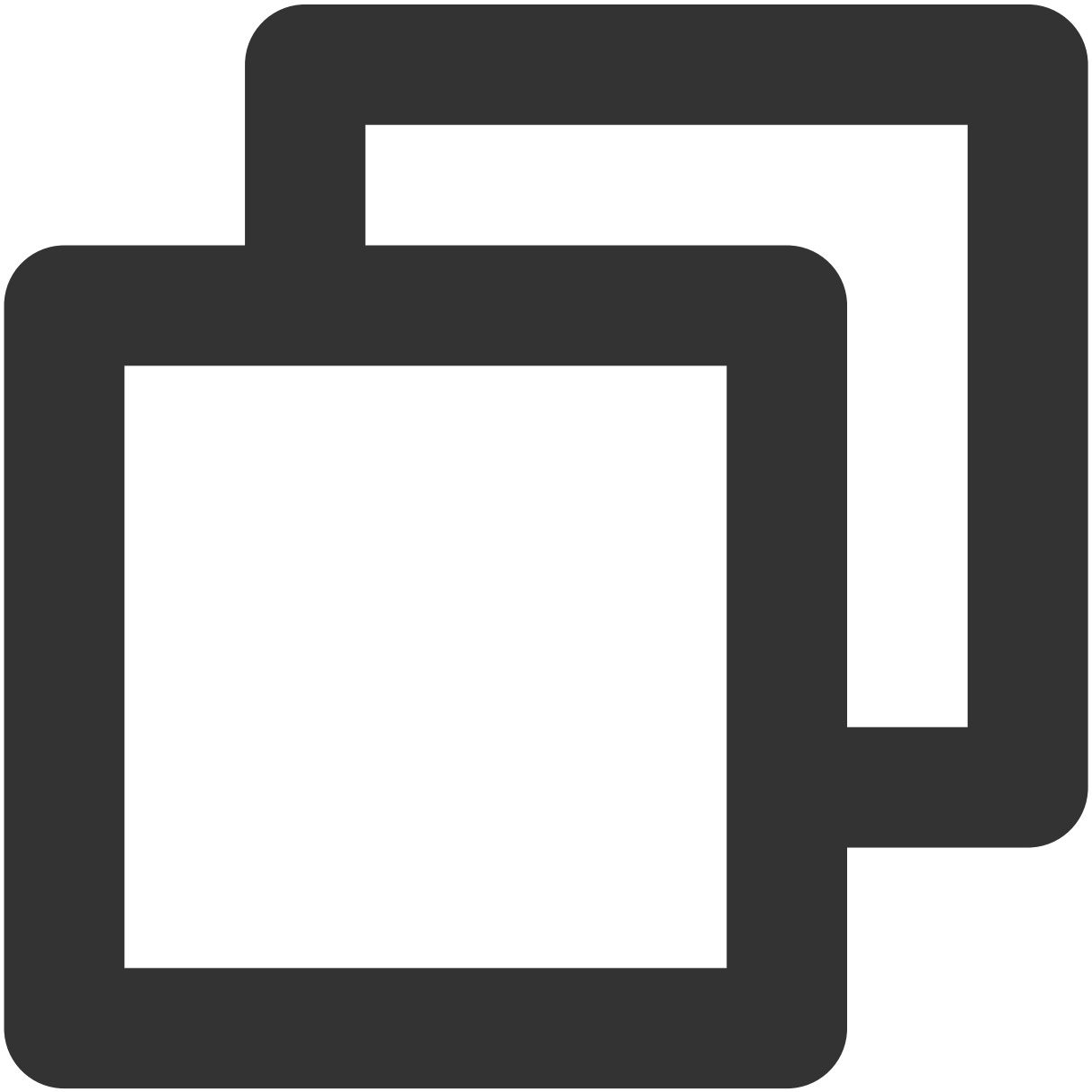
以上关键动作涉及的具体实现流程及代码实现详见歌曲同步、歌词同步、人声同步文档。

Android

最近更新时间：2023-09-27 11:28:24

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？



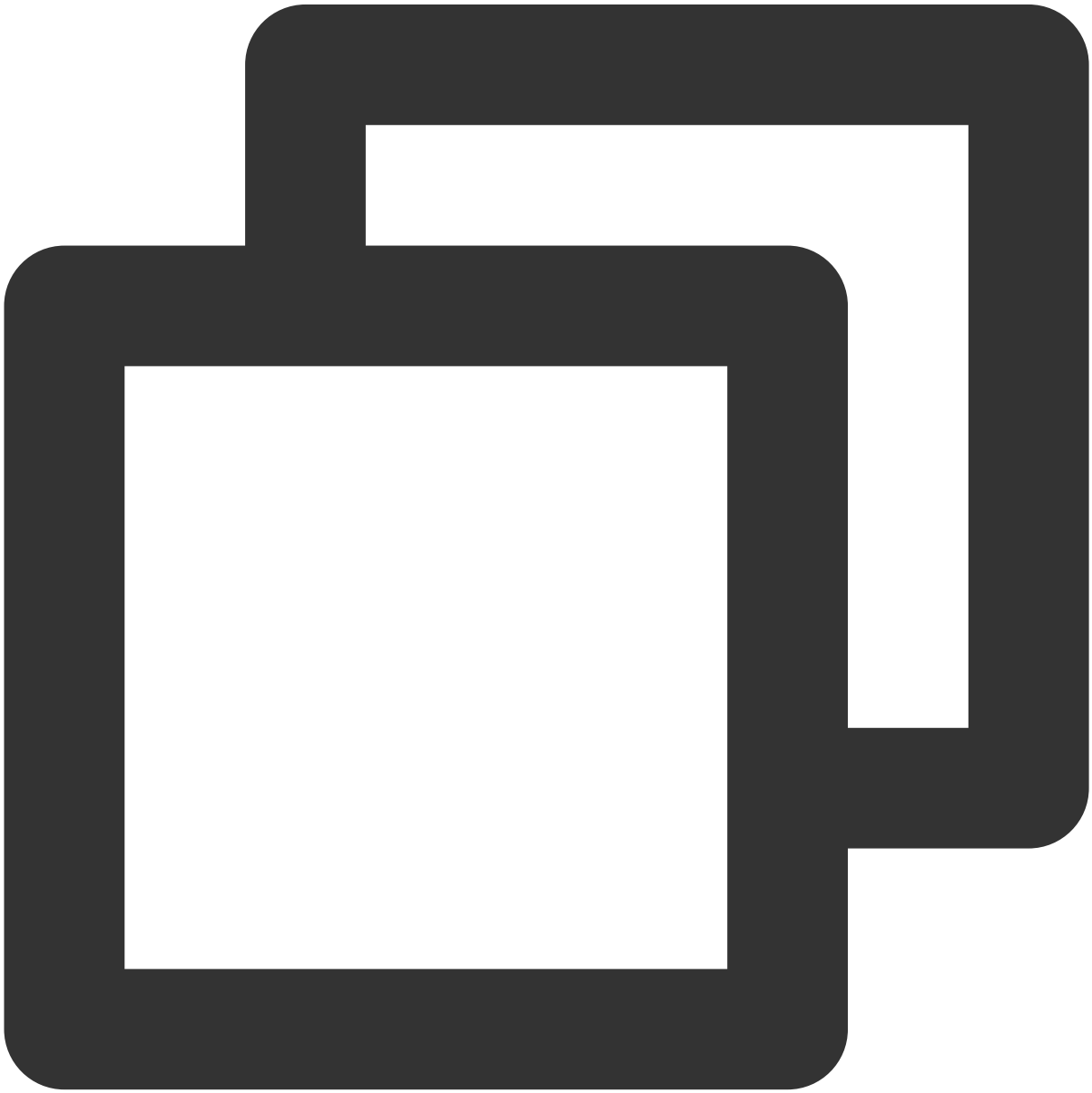
```
mTRTCCloud.getAudioEffectManager().enableVoiceEarMonitor(true)
```

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\\", \\"param
```

NTP 校时问题

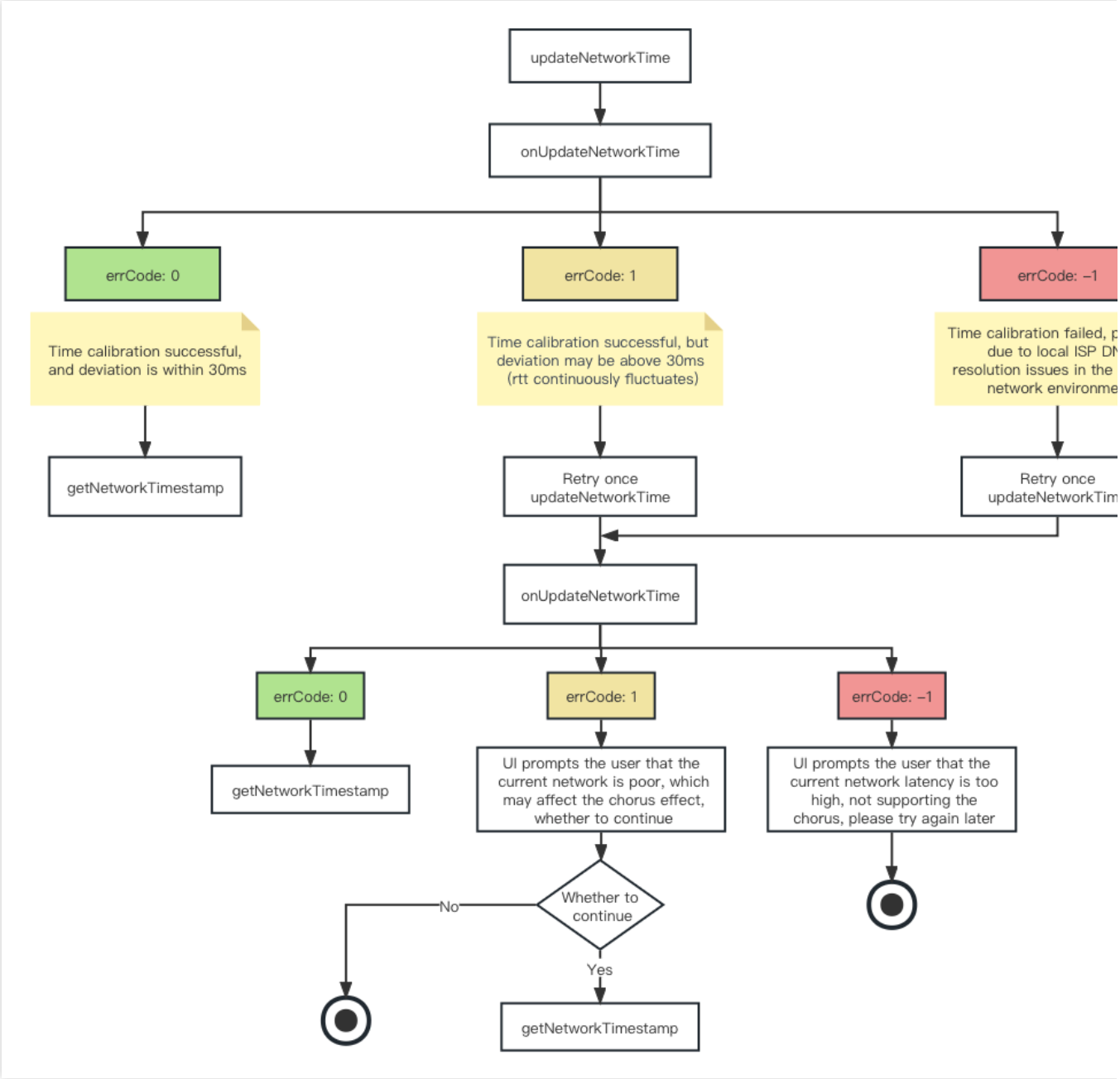
提醒：“NTP time sync finished, but result maybe inaccurate”？

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。

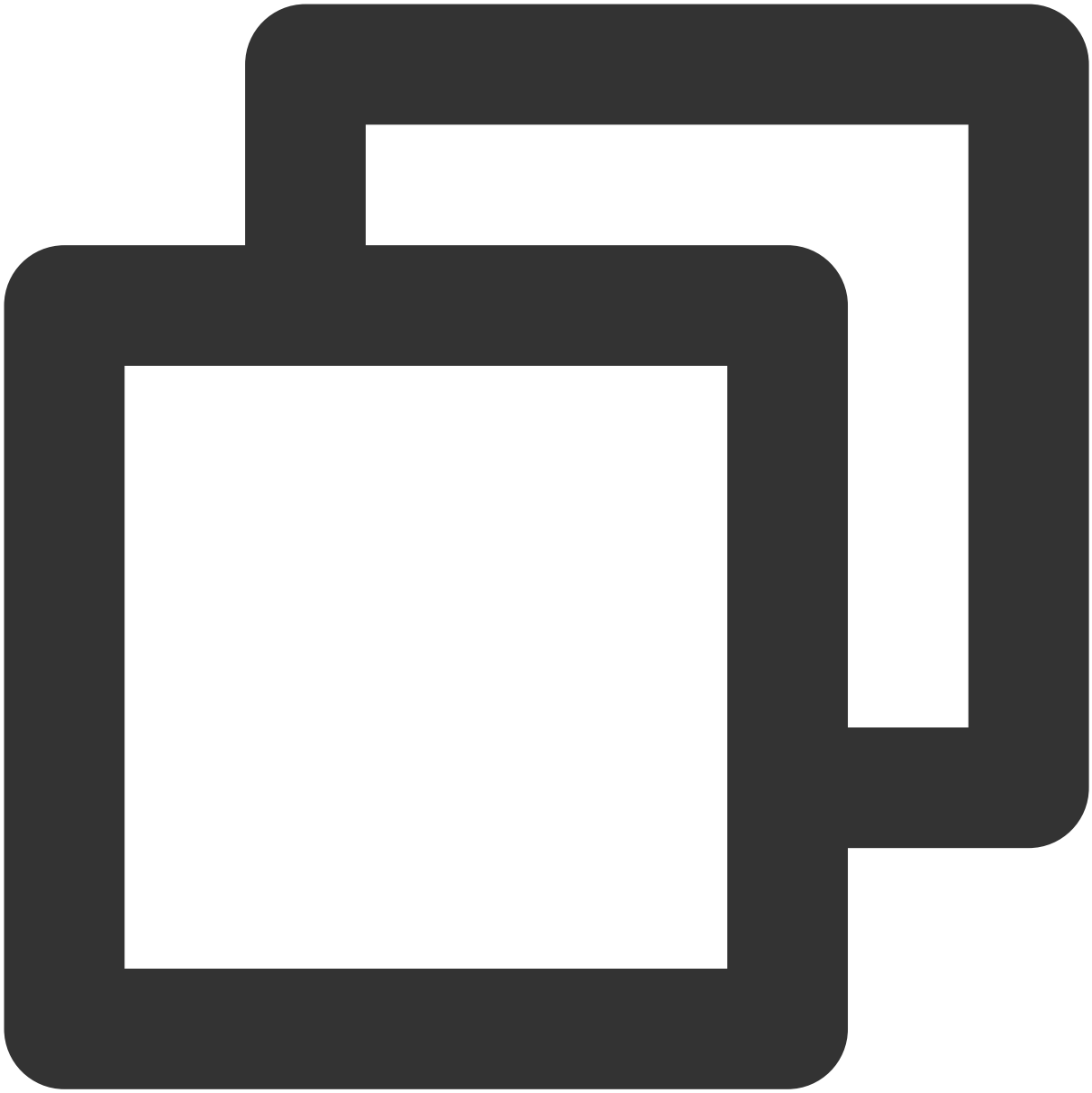
NTP 服务重试处理逻辑？



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。

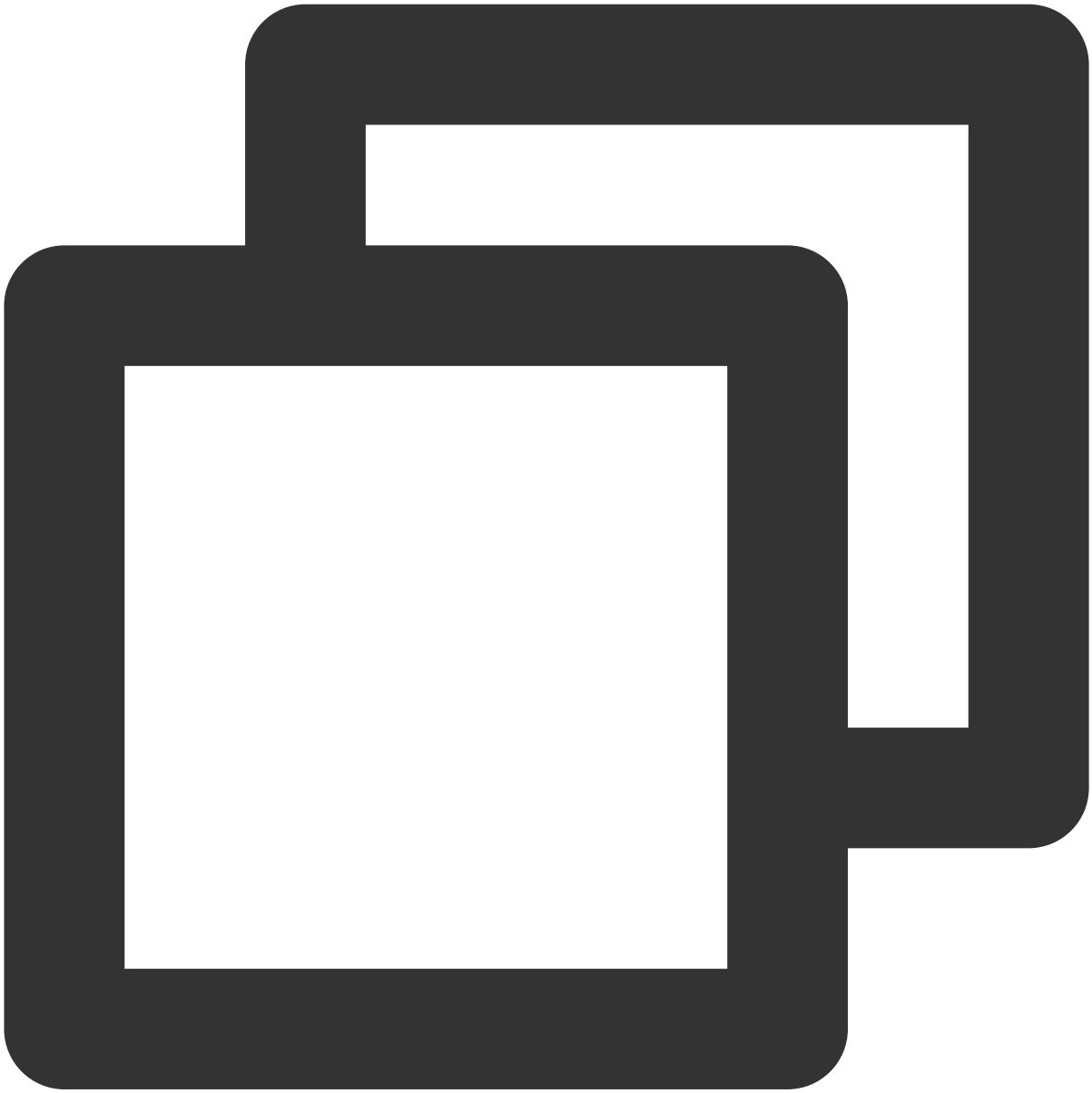
TRTC SDK 网络测速的发起：



```
TRTCCloudDef.TRTCSpeedTestParams speedTestParams = new TRTCCloudDef.TRTCSpeedTestPa
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\\
            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。