

会议 / 直播

iOS 5.X

2024-08-30

实时音视频开发指导

更新时间:2024-08-30

欢迎使用融云实时音视频（RTC）。RTC 服务基于房间模型设计，可以支持一对一音视频通信、和多人音视频通信。底层基于融云 IM 信令通讯，可保障在长时间音视频通话及弱网情况下保持正常连通。

本页面简单介绍融云 RTC 服务能力和 SDK 产品。

客户端 SDK

融云客户端 SDK 提供丰富的接口，大部分能力支持开箱即用。配合 RTC 服务端 API 接口，可满足丰富的业务特性要求。

提示

[前往融云产品文档](#) · [客户端 SDK 体系](#) · [RTCLib](#) · [CallKit](#) · [CallLib](#) · [CallPlus](#) >

SDK 适用场景

CallKit、CallLib、RTCLib 是融云 RTC 服务提供的三款经典的客户端 SDK。其中 CallKit、CallLib 用于开发音视频通话（呼叫）业务。RTCLib 是音视频基础能力库，可满足类似会议、直播等业务场景需求，具备较高的扩展与定制属性。

业务分类	适用的 SDK	流程差异	场景描述
通话（呼叫）	CallKit、CallLib	SDK 内部呼叫流程自动处理房间号	拨打音视频电话（类比微信音视频通话）
会议	RTCLib	与会者需要约定房间号，参会需进入同一房间	线上会议、小班课、在线视频面试、远程面签等
直播	RTCLib	支持区分主播、观众角色。观众可通过连麦进行发言。	直播社交、大型发布会、语聊房、线上大班课等

如何选择 SDK

不同 SDK 适用的业务场景差异较大，请您谨慎选择并决策。

- **CallKit 与 CallLib** 是用于实现通话（呼叫）功能的客户端库。封装了拨打、振铃、接听、挂断等一整套呼叫流程，支持一对一及群组内多人呼叫的通话能力。CallKit 与 CallLib 均依赖 RTCLib，两者区别如下：
 - CallKit 提供了呼叫相关的通用 UI 扩展库。
 - CallLib 不含任何 UI 界面组件。
- **RTCLib** 是融云音视频核心能力库。应用开发者可将 RTCLib 用于支持直播、会议业务场景。

具体选择建议如下：

- 不需要通话（呼叫）功能，可使用 RTCLib，即您仅需要融云为您的 App 提供实时音视频（RTC）核心能力。
- 需要开发支持通话（呼叫）的音视频应用，但不希望自行实现呼叫 UI，可使用 CallKit。直接利用融云提供的呼叫 UI，节省

开发时间。

- 需要开发支持通话（呼叫）的音视频应用，不希望 SDK 带任何 UI 组件，可使用 CallLib。
- 通过融云提供的独立功能插件扩展客户端 SDK 的功能。

在使用融云 SDK 进行开发之前，我们建议使用快速上手教程与示例项目进行评估。

高级和扩展功能

RTC 服务支持的高级与扩展功能，包括但不限于以下项目：

- 跨房间连麦：支持多主播跨房间连麦 PK 直播。
- 通话数据统计：按照指定的时间间隔上报通话的详细数据。
- 屏幕共享：通过自定义视频流的方式在房间内发起屏幕共享功能。
- 自定义加密：可选择对媒体流进行加密，从而保障用户的数据安全。
- 插件支持：支持通过插件实现美颜、CDN 播放器等功能。
- 云端录制：在音视频通话（呼叫）、直播、会议时分别录制每个参与者的音视频、或合并后进行录制。
- 内容审核：融云媒体服务器（RTC Server）把收到的音视频流转码后送审，审核结果返回应用服务器。

部分功能需配合 RTC 服务端 API 使用。具体支持的功能与平台相关。具体使用方法请参见客户端 SDK 开发文档或服务端开发文档。

平台兼容性

CallKit、CallLib、RTCLib 均支持主流移动操作平台，客户端功能在多端基本保持一致，支持多平台互通。CallPlus 暂仅支持 Android、iOS、Web 平台。

平台/框架	接口语种	支持架构	说明
Android	Java	armeabi-v7a、arm64-v8a、x86、x86-64	系统版本 4.4 及以上
iOS	Objective-C	---	系统版本 9.0 及以上
Windows	C++、Electron	x86、x86-64	Windows 7 及以上
Linux	C、Electron	---	推荐 Ubuntu 16.04 及以上；其他发行版需求请咨询商务
MacOS	Electron	---	系统版本 10.10 及以上
Web	Javascript	---	详见客户端文档「Web 兼容性」
Flutter	dart	---	Flutter 2.0.0 及以上
uni-app	Javascript	---	uni-app 2.8.1 及以上
React Native	Javascript	---	React Native 0.65 及以上
Unity	C#	Android(armeabi-v7a、arm64-v8a) iOS(arm64,armv7) Windows(x86、x86-64)	---

版本支持

RTC 服务客户端 SDK 针对各平台/框架提供的最新版本如下（--- 表示暂未支持）：

SDK/平台	Android	iOS	Web	Electron	Flutter	Unity	uni-app	小程序	React Native	Windows - C++	Linux - C
RTCLib	5.6.x	5.6.x	5.6.x	5.6.x	5.2.x	5.2.x	5.2.x	5.0.x	5.2.x	5.1.x	见注 ¹
CallLib	5.6.x	5.6.x	5.0.x	5.1.x	5.1.x	---	5.1.x	3.2.x	5.1.x	---	---
CallKit	5.6.x	5.6.x	---	---	---	---	---	---	---	---	---
CallPlus	2.x	2.x	2.x	---	---	---	---	---	---	---	---

注 1：关于 Linux 平台的支持，请咨询融云的商务。

SDK 体积对比

Android 端

以下数据基于 RTC 5.X 版本。

CPU 架构	集成 RTCLib 增量	集成 CallLib 增量	集成 CallKit 增量
armeabi	4.5MB	4.6MB	7.4MB
arm64-v8a	5.1MB	5.1MB	8.0MB
x86	5.4MB	5.4MB	8.3MB
全平台	17.2MB	17.2MB	20.1MB

iOS 端

以下数据基于 RongCloudRTC 5.X 版本。

CPU 架构	集成 RTCLib 增量	集成 CallLib 增量	集成 CallKit 增量
arm64	4.3M	4.4M	8.9M
arm64 + armv7	8.6M	8.9M	14.8M

实时音视频服务端


实时音视频服务端 API 可以协助您构建集成融云音视频能力的 App 后台服务系统。

您可以使用服务端 API 将融云服务集成到您的实时音视频服务体系中。例如，向融云获取用户身份令牌 (Token)，从应用服务端封禁用户、移出房间等。

 提示

[前往融云服务端开发文档 · 集成必读](#) >>

控制台

使用[控制台](#) ，您可以对开发者账户和应用进行管理，开通音视频服务，以及其他高级服务，查看应用数据报表，和计费数据。

 提示

音视频服务必须要从控制台开通后方可使用。参见[开通音视频服务](#)。

实时音视频数据

您可以前往控制台的[数据统计页面](#)，查询、查看音视频用量、业务健康检查等数据。开通相应服务后，还能获取如业务数据分析等数据。

融云还提供通话质量实时的监控工具，以图表形式展示每一通音视频通话的质量数据，帮助定位通话问题，提高问题解决效率。

融云不会利用客户的数据。同时融云提供完善的数据隐私保护策略。参见[SDK 隐私政策](#)。

运行示例项目 (Demo)

更新时间:2024-08-30

融云音视频产品提供一个功能体验 QuickDemo 示例应用项目 ([Github](#) · [Gitee](#))，集中演示了融云实时音视频产品 [音视频通话](#)、[音视频会议](#)、[低延迟直播](#) 在 iOS 端的功能，以便开发者体验产品，快速集成，实现单群聊、音视频通话、语音聊天室、娱乐直播、教学课堂、多人会议等场景需求。

QuickDemo 按场景和功能分为多个模块，对主要功能进行演示。

QuickDemo 开放源代码，您可以对感兴趣的部分进行代码改造，以便进一步了解细节。

前置条件

- [注册开发者账号](#)。注册成功后，控制台会默认自动创建您的首个应用，默认生成开发环境下的 App Key，使用国内数据中心。
- 获取开发环境的应用 App Key。如不使用默认应用，请参考 [如何创建应用](#)，并获取对应环境 App Key 和 App Secret。

提示

每个应用具有两个不同的 App Key，分别对应开发环境与生产环境，两个环境之间数据隔离。在您的应用正式上线前，可切换到使用生产环境的 App Key，以便上线前进行测试和最终发布。

- 开通音视频通话服务，以及音视频直播服务。开发环境下可免费开通，每个应用均可享有 10000 分钟免费体验时长，免费体验时长用完即止。生产环境下需要先预存费用才可开通。您需要同时开通以上两项服务方可体验 QuickDemo 的全部能力。服务开通后最长 30 分钟生效。详见 [开通音视频服务](#)。

运行 QuickDemo

在运行 QuickDemo 前请确保已完成上述步骤。以下是检查清单：

- 已注册融云开发者账户
- 已准备好 App Key 和 App Secret
- 已开通音视频服务免费体验，且已等待 30 分钟。

步骤 1：下载并加载 QuickDemo 代码

1. 克隆下载示例代码。

```
git clone https://github.com/rongcloud/rtc-quickdemo-ios.git
```

2. 进入 QuickDemo 文件夹，在终端执行以下命令，下载依赖库。

```
$ pod install
```

提示

如果出现找不到相关版本的问题，可先执行 `pod repo update`，再执行 `pod install`。

3. 使用 XCode 加载 xcworkspace 文件，即 QuickDemo 工程。
4. 找到 /RCRTCQuickDemo/Tool/Constant/Constant.m 里的 AppKey 和 AppSecret 并替换为从控制台获取到的具体内容，然后去掉 #error 提醒。

```
/*!
请填写 App Key
获取地址: https://console.rongcloud.cn/agile/formwork/app/appService
*/
NSString * const AppKey = @"";

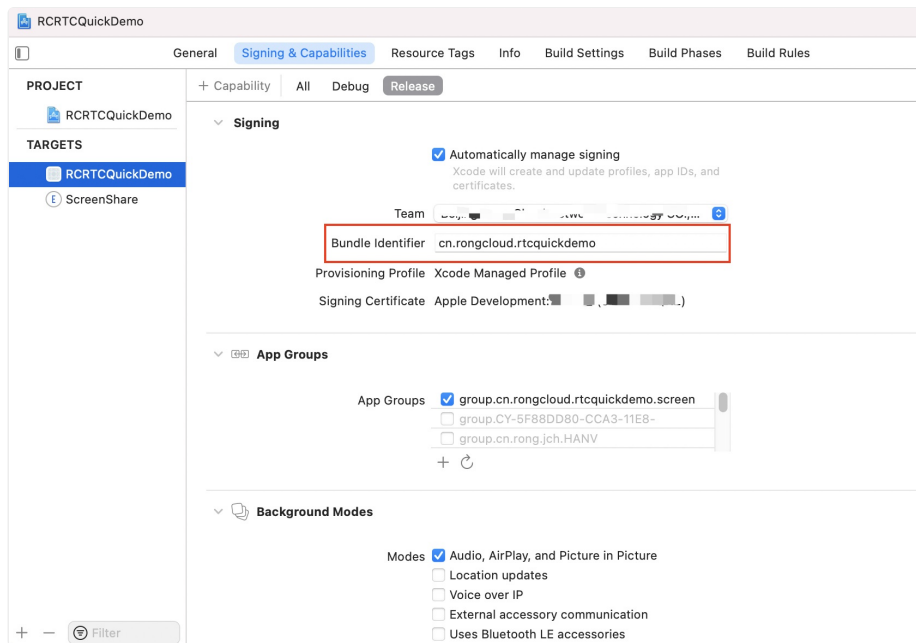
/*!
请填写 App Secret
获取Token 需要提供 App Key 和 App Secret
正式环境请勿在客户端请求 Token，您的客户端代码一旦被反编译，会导致您的 AppSecret 泄露。
请务必确保在服务端获取 Token。
*/
NSString * const AppSecret = @"";
```

提示

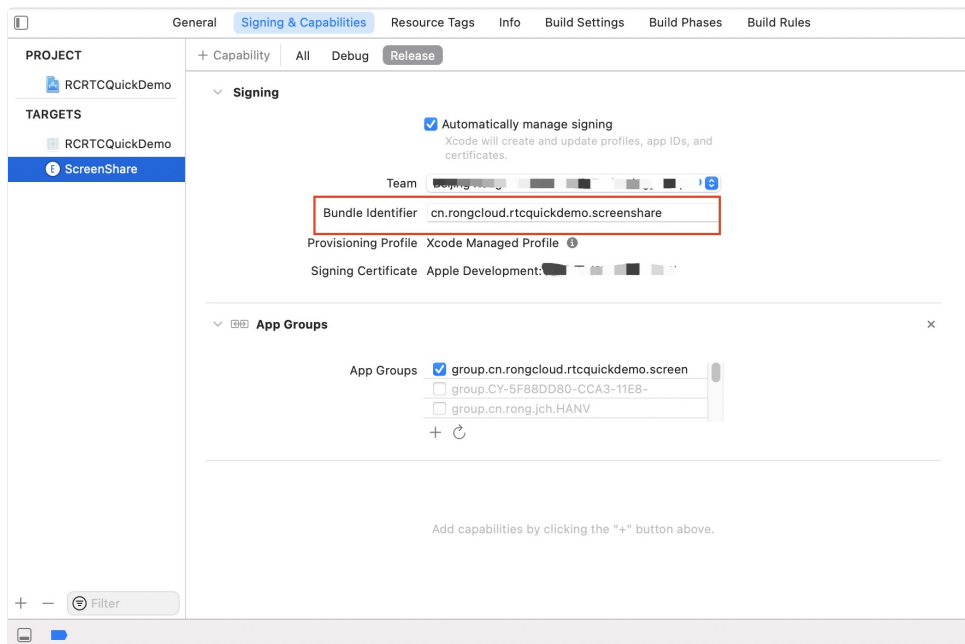
本教程中直接在客户端代码中写入 APP_SECRET 的行为仅为演示目的。APP_SECRET 可用来获取用户身份令牌 (Token)，以及实现人员禁言、房间踢人等高级能力。存储在客户端代码里很容易被反编译导致泄漏。APP_SECRET 一旦泄露，攻击者就可以盗取 SDK 服务流量，或进行高权限破坏性操作。正确的方式是将 APP_SECRET 存储在您的应用服务端，并提供面向应用客户端的接口。更多详情请参见音视频服务端开发文档。

步骤 2：修改 Bundle ID

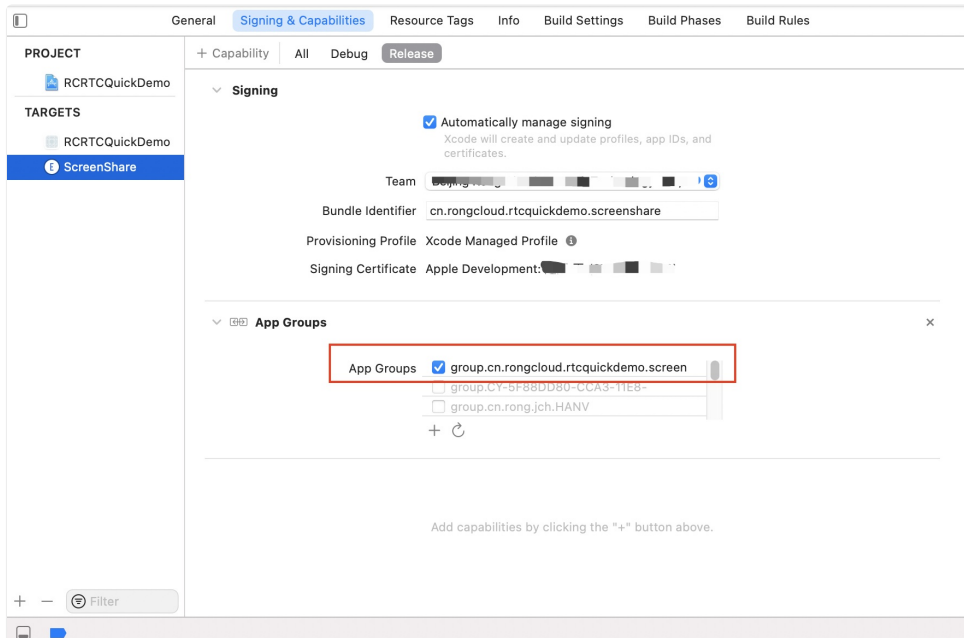
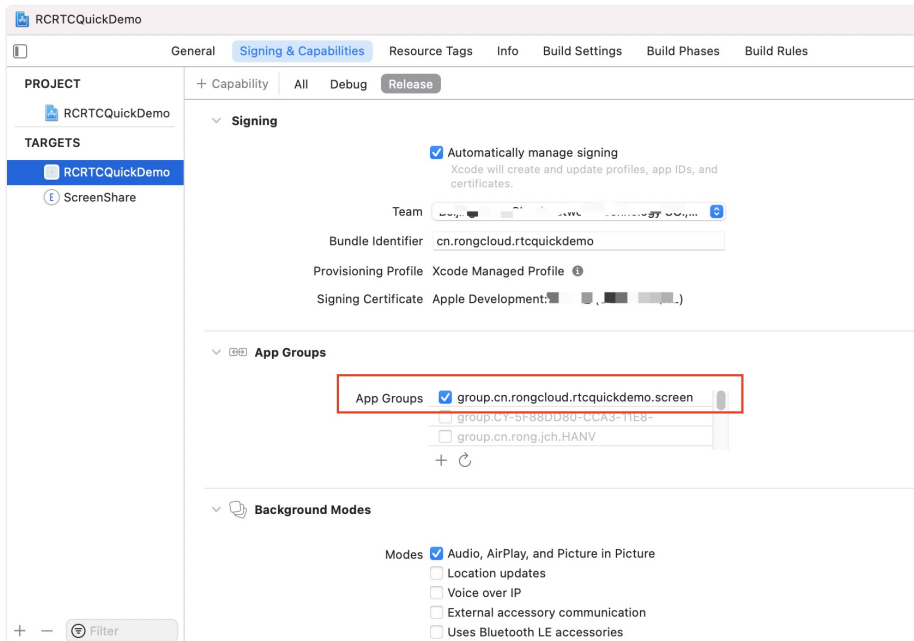
1. 修改工程中 RCRTCQuickDemo 的 Bundle Identifier，例如：cn.rongcloud.rtcquickdemo。



2. 修改工程中 ScreenShare 的 Bundle Identifier，保持跟前面一致，在最后添加 screenshare，例如：cn.rongcloud.rtcquickdemo.screenshare。



3. 修改工程中 RCRTCQuickDemo 和 ScreenShare 的 App Group 配置，修改成与 screenshare 相同的 Bundle Identifier。



步骤 3：编译运行

运行成功后输入用户 ID，即可进入体验。建议在真实设备上运行。

RongCloud RTC



User ID : 111

 **会议 1v1**
会议 1v1

 **Live**
直播

 **CallLib**
会议 1v1

 **CallKit**
会议 1v1

 **共享屏幕**
会议 1v1

 **CDN**
使用cdn拉流

导入 SDK

更新时间:2024-08-30

融云支持使用 CocoaPods 和本地手动导入两种方式，将 RTCLib SDK 导入到您的应用工程中。

环境要求

- iOS 9.0 及以上。
- Xcode 9.0 或以上版本。

检查版本

在导入 SDK 前，您可以前往 [融云官网 SDK 下载页面](#) 确认当前最新版本号。

CocoaPods 导入

1. 在 podfile 中添加如下内容：

```
pod 'RongCloudRTC/RongRTCLib', '~> x.y.z'  
pod 'RongCloudRTC/RongFaceBeautifier', '~> x.y.z' 美颜 (可选)  
pod 'RongCloudRTC/RongRTCPlayer', '~> x.y.z' 融云 CDN + 混音网络资源文件 (可选)  
pod 'RongCloudRTC/RongVoiceBeautifier', '~> x.y.z' 美声特效 (可选)
```

提示

- `x.y.z` 代表具体版本，各个 SDK 的最新版本号可能不同，在融云下载页或 CocoaPods 仓库能查询到。
- RTCLib 会自动依赖下载对应版本的 IMLib 库，一般不需要手动指定。
- RTCLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致。从 5.2.0 开始至 5.4.4（不含），要求前三位一致。从 5.4.4 开始，要求前两位保持一致。注意，RTCLib 5.4.4 不可匹配小于 5.4.4 的 IM SDK。

2. 请在终端中运行以下命令：

```
pod install
```

提示

如果出现找不到相关版本的问题，可先执行 `pod repo update`，再执行 `pod install`。

- 上一步完成后，会自动导入指定版本的融云 SDK，CocoaPods 会在您的工程根目录下生成一个 `xcworkspace` 文件，只需通过 XCode 打开该文件即可加载工程。

本地手动导入

提示

RTCLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致。从 5.2.0 开始至 5.4.4（不含），要求前三位一致。从 5.4.4 开始，要求前两位保持一致。注意，RTCLib 5.4.4 不可匹配小于 5.4.4 的 IM SDK。

在导入 SDK 前，您需要前往融云官网 SDK 下载页面 [SDK 下载页面](#)，将音视频通话（无 UI）SDK 下载到本地。

- 导入 `RongRTCLib.xcframework`，并将 Embed 设置为 Embed & Sign。
- 导入 `RongIMLibCore.xcframework`，并将 Embed 设置为 Embed & Sign。

工程配置

- 音视频通话需要用到摄像头和麦克风权限，请在工程的 `info.plist` 中添加如下键值：
 - Privacy - Microphone Usage Description
 - Privacy - Camera Usage Description
- 请将工程中 Target -> Signing & Capabilities -> Background Modes 如下内容勾选：
 - Audio, AirPlay, and Picture in Picture
 - Remote notifications

提示

SDK 5.1.1 及之前的版本，音视频通话中需要用到 HTTP 请求，请在工程的 `info.plist` 中添加如下键值：App Transport Security Settings，并在此键值下再添加 Allow Arbitrary Loads 并将 Value 设置为 YES。

实现音视频会议

更新时间:2024-08-30

融云开发者账户是使用融云 SDK 产品的必要条件。在开始之前，请先[前往融云官网注册开发者账户](#)。注册后，控制台将自动为你创建一个应用，默认为开发环境应用，使用国内数据中心。请获取该应用的 App Key，在本教程中使用。

首次使用融云音视频的用户，建议参考文档[运行示例项目](#)，以完成开发者账号注册、音视频服务开通等工作。

提示

房间人数上限

考虑移动设备的带宽（主要是在多路视频情况下）和 UI 交互效果，建议单次通话或房间内，视频不超过 16 人，纯音频不超过 32 人。超过此上限可能影响通话效果。

环境要求

- iOS 9.0 及以上。
- Xcode 9.0 或以上版本。

步骤 1：服务开通

您在融云创建的应用默认不会启用音视频服务。在使用融云提供的任何音视频服务前，您需要前往控制台，为应用开通音视频服务。

具体步骤请参见文档[开通音视频服务](#)。

提示

服务开通、关闭等设置完成后 30 分钟后生效。

步骤 2：初始化

RTCLib 是基于 IMLibCore 作为信令通道的，所以要先初始化 IMLibCore。如果不换 AppKey，在整个应用生命周期中，初始化一次即可。

```
[[RCCoreClient sharedCoreClient] initWithAppKey:@"从控制台申请的 AppKey"];
```

- 部分 RTC 引擎必须在初始化时提供，详见[引擎配置](#)。

步骤 3：连接 IM 服务

音视频用户之间的信令传输依赖于融云的即时通信（IM）服务，因此需要先调用 `connectWithToken` 与 IM 服务建立好 TCP 长连接。建议在功能模块的加载位置处调用，之后再行音视频呼叫业务。当模块退出后调用 `disconnect` 或 `logout` 断开该连接。

```
// 连接 IM 服务
[[RCCoreClient sharedCoreClient] connectWithToken:@"从您服务器端获取的 Token"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];
```

步骤 4：加入房间

指定房间 id 并加入房间，通过异步回调的 `code` 判断是否加入房间成功。

```
RCRTCVideoStreamConfig *videoConfig = [[RCRTCVideoStreamConfig alloc] init];
videoConfig.videoSizePreset = RCRTCVideoSizePreset720x480;
videoConfig.videoFps = RCRTCVideoFPS30;
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoConfig:videoConfig];

RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];
config.roomType = RCRTCRoomTypeNormal;
[[RCRTCEngine sharedInstance] enableSpeaker:YES];
[[RCRTCEngine sharedInstance] joinRoom:@"Your_Room_ID" config:config completion:^(RCRTCRoom *Nullable
room, RCRTCCode code) {
if (code != RCRTCCodeSuccess) {
// 加入房间失败处理
} else {
// 加入成功后进行资源的发布和订阅，见步骤 7
[self afterJoinRoom:room];
}
}
}];
```

步骤 5：发布资源

加入成功后开始采集本地视频并发布本地视频流。

```

- (void)afterJoinRoom:(RCRTCRoom *)room {
// 设置房间代理
self.room = room;
room.delegate = self;

// 开始本地视频采集
[[[RCRTCEngine sharedInstance] defaultVideoStream] setVideoView:self.localView];
[[[RCRTCEngine sharedInstance] defaultVideoStream] startCapture];

// 发布本地视频流
[room.localUser publishDefaultStreams:^(BOOL isSuccess, RCRTCCode desc) {
if (isSuccess && desc == RCRTCCodeSuccess) {
NSLog(@"本地流发布成功");
}
}]];

// 如果已经有远端用户在房间中，需要订阅远端流
if ([room.remoteUsers count] > 0) {
NSMutableArray *streamArray = [NSMutableArray array];
for (RCRTCRemoteUser *user in room.remoteUsers) {
[streamArray addObjectsFromArray:user.remoteStreams];
}
// 见步骤 8
[self subscribeRemoteResource:streamArray];
}
}

```

步骤 6：订阅资源

1. 如果已经有远端用户在房间中，需要订阅远端流。

```

- (void)subscribeRemoteResource:(NSArray<RCRTCInputStream *> *)streams {
[self.room.localUser subscribeStream:streams
tinyStreams:nil
completion:^(BOOL isSuccess, RCRTCCode desc) {
if (isSuccess && desc == RCRTCCodeSuccess) {
NSLog(@"无端流订阅成功");
}
}]];

// 创建并设置远端视频预览视图
for (RCRTCInputStream *stream in streams) {
if (stream.mediaType == RTCMediaTypeVideo) {
[[[RCRTCVideoInputStream *) stream] setVideoView:self.remoteView];
[self.remoteView setHidden:NO];
}
}
}
}

```

2. 监听房间事件。

```
#pragma mark - RCRTCRoomEventDelegate
// 远端用户发布资源通知
- (void)didPublishStreams:(NSArray<RCRTCInputStream *> *)streams {
    [self subscribeRemoteResource:streams];
}

// 远端用户取消发布资源通知
- (void)didUnpublishStreams:(NSArray<RCRTCInputStream *> *)streams {
    [self.remoteView setHidden:YES];
}

// 远端用户离开通知
- (void)didLeaveUser:(RCRTCRemoteUser *)user {
    [self.remoteView setHidden:YES];
}
```


实现低延迟直播

更新时间:2024-08-30

融云开发者账户是使用融云 SDK 产品的必要条件。在开始之前，请先[前往融云官网注册开发者账户](#)。注册后，控制台将自动为您创建一个应用，默认为开发环境应用，使用国内数据中心。请获取该应用的 App Key，在本教程中使用。

首次使用融云音视频的用户，建议参考文档[运行示例项目](#)，以完成开发者账号注册、音视频服务开通等工作。

步骤 1：服务开通

您在融云创建的应用默认不会启用音视频服务。在使用融云提供的任何音视频服务前，您需要前往控制台，为应用开通音视频服务。

具体步骤请参见文档[开通音视频服务](#)。

提示

服务开通、关闭等设置完成后 30 分钟后生效。

步骤 2：SDK 导入

您需要导入融云音视频核心能力库 RTCLib，和 RTC 业务所依赖的即时通讯能力库 IMLib。根据您的业务需求，可选择导入美颜扩展库和 CDN 扩展库。

融云推荐使用 CocoaPods 方式导入 SDK：

1. 配置 Podfile。

```
pod 'RongCloudRTC/RongRTCLib', '~> x.y.z'
```

提示

- `x.y.z` 代表具体版本，各个 SDK 的最新版本号可能不同，在[融云下载页](#)或 [CocoaPods 仓库](#)能查询到。
- RTCLib 会自动依赖下载对应版本的 IMLib 库，一般不需要手动指定。
- RTCLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致。从 5.2.0 开始至 5.4.4（不含），要求前三位一致。从 5.4.4 开始，要求前两位保持一致。注意，RTCLib 5.4.4 不可匹配小于 5.4.4 的 IM SDK。

2. 安装依赖项。

```
$ pod install
```

提示

如果出现找不到相关版本的问题，可先执行 `pod repo update`，再执行 `pod install`。

步骤详细说明及其他导入方式请参阅[导入 SDK](#)。

步骤 3：权限配置

1. 音视频通话需要用到摄像头和麦克风权限，请在工程的 `info.plist` 中添加如下键值：
 - Privacy - Microphone Usage Description
 - Privacy - Camera Usage Description
2. 请将工程中 Target -> Signing & Capabilities -> Background Modes 如下内容勾选：
 - Audio, AirPlay, and Picture in Picture
 - Remote notifications

提示

SDK 5.1.1 及之前的版本，音视频通话中需要用到 HTTP 请求，请在工程的 `info.plist` 中添加 `App Transport Security Settings` 键值，并在此键值下再添加 `Allow Arbitrary Loads` 并将 Value 设置为 YES。

步骤 4：初始化

音视频 SDK 是基于即时通信 SDK 作为信令通道的，所以要先初始化 IM SDK。如果不换 AppKey，在整个应用生命周期中，初始化一次即可。建议调用位置放在应用启动位置处，或在音视频功能模块的加载位置处。

```
[[RCCoreClient sharedCoreClient] initWithAppKey:@"从控制台申请的 AppKey"];
```

- 部分 RTC 引擎必须在初始化时提供，详见[引擎配置](#)。

步骤 5：连接 IM

音视频用户之间的信令传输依赖于融云的即时通信（IM）服务，因此需要先调用 `connectWithToken` 与 IM 服务建立好 TCP 长连接。建议在功能模块的加载位置处调用，之后再行音视频呼叫业务。当模块退出后调用 `disconnect` 或 `logout` 断开该连接。

```
[[RCCoreClient sharedCoreClient] connectWithToken:@"从您服务器端获取的 Token"  
dbOpened:^(RCDBErrorCode code) {}  
success:^(NSString *userId) {}  
error:^(RCConnectErrorCode status) {}  
tokenIncorrect:^{}];
```

主播端

多人之间想要发起音视频通话，需要加入同一个音视频房间。对于直播需求来讲，房间类型需选择 `RCRTCRoomTypeLive`，直播类型可根据业务需要选择 `RCRTCLiveTypeAudioVideo` 或 `RCRTCLiveTypeAudio`，即音视频直播间或纯音频直播间。加入房间的角色也分为 `RCRTCLiveRoleTypeBroadcaster` 和 `RCRTCLiveRoleTypeAudience`，即主播和观众。下面就这两种身份，分别进行说明。

步骤 6.1：加入房间

1. 构建 `RCRTCRoomConfig`，指定房间类型和主播身份：

```
// 设置房间、直播和角色类型  
RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];  
config.roomType = RCRTCRoomTypeLive;  
config.liveType = RCRTCLiveTypeAudioVideo;  
config.roleType = RCRTCLiveRoleTypeBroadcaster;
```

2. 调用 `RCRTCEngine` 下的 `joinRoom` 方法创建并加入一个直播房间。

```
[[RCRTCEngine sharedInstance] joinRoom:@"直播间 ID"  
config:config  
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {  
if (code != RCRTCCodeSuccess) {  
// 加入房间失败处理  
return;  
}  
// 进行发布订阅流操作  
}];
```

③ 提示

客户端通过 `joinRoom` 传入的直播间 ID 来加入不同房间。房间不需要客户端创建或销毁，融云服务若发现该房间不存在时会自动创建。当所有主播（只有观众不算）离开持续 24 小时后，服务会自动销毁该房间。

步骤 6.2：发布音视频流

加入房间后，开始摄像头采集并发布音视频流。

```

// 1. 初始化本地渲染视图
RCRTCVideoView *view = [[RCRTCVideoView alloc] initWithFrame:CGRectMake(100, 100, 100, 100)];
// 2. 设置视频流的渲染视图
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoView:view];
// 3. 添加渲染视图
[self.view addSubview:view];
// 4. 开始摄像头采集
[[RCRTCEngine sharedInstance].defaultVideoStream startCapture];
// 5. 发布本地流到房间
[self.room.localUser publishDefaultLiveStreams:^(BOOL isSuccess, RCRTCCode desc, RCRTLIVEINFO *
_Nullable liveInfo) {
if (desc == RCRTCCodeSuccess) {
// 可以用 liveInfo 配置合流信息
self.liveInfo = liveInfo;
}
}]];

```

观众端

步骤 7.1：加入房间

1. 指定房间 id 并加入房间。

```

// 设置房间、直播和角色类型
RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];
config.roomType = RCRTCRoomTypeLive;
config.liveType = RCRTLIVEINFO_AudioVideo;
config.roleType = RCRTLIVEINFO_RoleTypeAudience;

```

2. 调用 RCRTCEngine 下的 joinRoom 方法创建并加入一个直播房间。

```

[[RCRTCEngine sharedInstance] joinRoom:@"Your_Room_ID"
config:config
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {
if (code != RCRTCCodeSuccess) {
// 加入房间失败处理
return;
}
// 进行订阅流操作
}]];

```

步骤 7.2：观众观看直播

观众可通过加入房间后返回的 RCRTCRoom 下 localUser 中的订阅方法订阅多路音视频流。同一个流只能填写在 avStreams 或 tinyStreams 中的一个数组中。

```

NSArray *tinyStream = isTiny ? streams : @[];
NSArray *ordinaryStream = isTiny ? @[] : streams;
// 订阅房间中远端用户音视频流资源
[self.room.localUser subscribeStream:ordinaryStream
tinyStreams:tinyStream
completion:^(BOOL isSuccess, RCRTCCode desc) {
if (desc != RCRTCCodeSuccess) {
return;
}
// 创建并设置远端视频预览视图
NSInteger i = 0;
for (RCRTCInputStream *stream in liveStreams) {
if (stream.mediaType == RTCMediaTypeVideo) {
if (i==0) {
// 1.初始化渲染远端视频的 view
RCRTCRemoteVideoView *view = [[RCRTCRemoteVideoView alloc] initWithFrame:CGRectMake(100, 400, 100, 100)];
// 2.设置视频流的渲染视图
[(RCRTCVideoInputStream *)stream setVideoView:view];
// 3.添加渲染视图
[self.view addSubview:view];
}else{
// 其他远端视图逻辑
}
}
}
}
}];

```

引擎配置

引擎配置速览

更新时间:2024-08-30

RTC 引擎提供以下配置，可按需修改。

配置项	默认值
断线重连	默认开启
媒体流加密功能 (SRTP)	默认关闭
状态报表数据回调时间间隔	默认 1000ms
检测 IM SDK 和 RTC SDK 版本号是否一致	默认开启
音频初始化配置 - 与其它后台 App 进行混音	默认开启
音频初始化配置 - 立体声	默认开启
视频初始化配置 - 硬件高压缩编码	默认关闭

断线重连

断线重连功能默认开启，可以在引擎初始化时传入以下配置进行关闭：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.isEnableAutoReconnect = NO;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

状态报表数据回调时间间隔

状态数据报表回调默认时间间隔为 1000ms，最小时间间隔为 100ms。请注意，过小的时间间隔会影响性能。

可以在引擎初始化时传入以下配置进行修改：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
NSInteger myInterval = 2000 // 修改回调时间间隔为 2 秒
config.statusReportInterval = myInterval;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

媒体流加密功能 (SRTP)

SDK 内置 SRTP 安全实时传输协议，即协议层的标准加密方式。以开关形式提供，使用简单。媒体流加密功能 (SRTP) 功能默认关闭。请注意，开启该功能会对性能和用户体验有一定影响，如果没有该需求请不要打开。

可以在引擎初始化时传入以下配置进行开启：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.enableSRTP = YES;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

检测 IM SDK 和 RTC SDK 版本号是否一致

默认开启，用于在 Debug 模式下检测 IM SDK 和 RTC SDK 版本号是否一致。

可以在引擎初始化时传入以下配置进行关闭：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.enableVersionMismatch = NO;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

音频初始化配置

与其它后台 App 进行混音

默认开启，允许与其他后台 App 进行混音。请注意，如果该属性设置为 NO，切换到其它 App 操作麦克风或者扬声器时，会导致自己 App 麦克风采集和播放被打断。

可以在引擎初始化时传入以下配置进行关闭：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.enableMixWithOthers = NO;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

开启/关闭立体声

默认开启，可以在引擎初始化时传入以下配置进行关闭：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.enableStereo = NO;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

视频初始化配置

开启/关闭硬件高压压缩编码

默认关闭，使用BaseLine进行视频编码。您可以在引擎初始化时传入以下配置进行开启：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.enableHardwareEncoderHighProfile = YES;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

③ 提示

开启硬件高压压缩编码可能会引发兼容性问题，请谨慎使用。

基本操作

创建/加入房间

更新时间:2024-08-30

调用 RCRTCEngine 下的 joinRoom 方法加入房间，如果该房间之前不存在，则会在调用时自动创建并加入。

提示

每个房间在创建之初，会由融云服务生成一个在用户全网唯一的 SessionId，可用于后台业务查询或与融云进行问题沟通。当房间内的所有人退出或被服务器判定掉线后，此 Session 结束。之后即便再用相同的 RoomId 创建房间，SessionId 也会更新为不同值。

会议模式下，推荐使用不带 RCRTCRoomConfig 的 joinRoom:completion: 方法：

```
[[RCRTCEngine sharedInstance] joinRoom:@"Your_Room_ID" completion:^(RCRTCRoom * _Nullable room,
RCRTCCode code) {
if (code != RCRTCCodeSuccess) {
// 由于 SDK 未初始化，网络异常等原因，造成的加入房间失败后的逻辑处理 ...
// 失败原因参考 code 的具体含义。
// 处理逻辑可以是过段时间重新加入，或给用户弹通知等。
return;
}
// 成功后的逻辑处理 ...
}];
```

直播模式下，推荐使用带 RCRTCRoomConfig 的 joinRoom:config:completion: 方法：

```
// 创建房间配置对象，设置房间类型为音视频直播，角色身份为主播。
RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];
config.roomType = RCRTCRoomTypeLive;
config.liveType = RCRTCLiveTypeAudioVideo;
config.roleType = RCRTCLiveRoleTypeBroadcaster;
[[RCRTCEngine sharedInstance] joinRoom:@"Your_Room_ID" config:config completion:^(RCRTCRoom * _Nullable
room, RCRTCCode code) {
if (code != RCRTCCodeSuccess) {
// 由于 SDK 未初始化，网络异常等原因，造成的加入房间失败后的逻辑处理 ...
// 失败原因参考 code 的具体含义。
// 处理逻辑可以是过段时间重新加入，或给用户弹通知等。
return;
}
// 成功后的逻辑处理 ...
}];
```

参数说明如下：

参数	类型	说明
roomId	字符串	房间唯一 ID。roomId 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式 最长 64 个字符。
config	RCRTCRoomConfig	房间配置，包含以下属性： <ul style="list-style-type: none"> <code>roomType</code>: 房间场景。直播模式下请指定为 <code>RCRTCRoomTypeLive</code>。 <code>liveType</code>: 直播类型（纯音频、音视频）。仅在 <code>roomType</code> 选择 <code>RCRTCRoomTypeLive</code>（直播模式）下设置有效。 <code>roleType</code>: 直播角色（主播、观众）。仅在 <code>roomType</code> 选择 <code>RCRTCRoomTypeLive</code>（直播模式）下设置有效。
completion	异步回调代码块	结果回调，返回房间对象 room。可从中获取当前房间内的所有主播，以及其他属性

退出房间

1. 如果用户开启了视频采集，在调用 [leaveRoom](#) 之前必须手动调用 `[[RCRTCEngine sharedInstance].defaultVideoStream stopCapture]` 关闭视频采集。

提示

离开房间接口（`leaveRoom`）不会自动关闭视频采集。如不主动关闭，可能会导致耗电量增加等问题。

2. 调用 RCRTCEngine 下的 `leaveRoom` 接口离开房间，离开时 SDK 内部会自动取消所有已发布和订阅的资源。

```
[[RCRTCEngine sharedInstance] leaveRoom:^(BOOL isSuccess, RCRTCCode code) {
}];
```

房间事件回调

更新时间:2024-08-30

应用程序可以通过 RCRTCRoom 对象的 delegate 属性设置注册 [RCRTCRoomEventDelegate](#) 代理委托。

设置代理后，App 可以监听房间的状态及资源变化。在会议模式下，参会的远端用户的状态变化都会触发通知。直播模式下，仅主播角色的远端用户会触发通知。

提示

[RCRTCRoomEventDelegate](#) 代理不会返回直播模式下观众相关的事件。原因如下：第一，观众只可订阅主播资源，不能发布资源，其操作对直播并无任何影响。第二，观众数量通常较多，如果应用针对观众进出直播间及订阅事件都进行处理，对应用程序的性能造成严重负面影响。

状态相关

1. 远端参会用户（或远端主播用户）加入通知：

当有远端参会用户（或远端主播用户）加入时触发。因主播加入房间后才能发布资源，该回调代表远端参会用户（或远端主播用户）刚刚加入，此时并无任何资源发布，所以当前也订阅不到该用户的任何媒体流。

```
- (void)didJoinUser:(RCRTCRemoteUser *)user;
```

2. 远端参会用户（或远端主播用户）离开通知：

当有远端参会用户（或远端主播用户）离开房间时触发，此时 SDK 会自动取消订阅该用户已发布的流，无需手动调用 unsubscribeStream。

```
- (void)didLeaveUser:(RCRTCRemoteUser *)user;
```

3. 远端参会用户（或远端主播用户）掉线通知：

当有远端参会用户（或远端主播用户）掉线时触发，代表该用户意外与融云服务断连超过 1 分钟。网络不好、App 意外崩溃或用户主动杀进程等情况，都会造成客户端与融云服务断连。如 1 分钟内没有恢复，则会被服务判定掉线，此时 SDK 会自动取消订阅该用户的所有资源，无需手动调用 unsubscribeStream。

```
- (void)didOfflineUser:(RCRTCRemoteUser *)user;
```

4. 远端参会用户（或远端主播用户）音频静默状态变更通知：

当远端参会用户（或远端主播用户）调用了音频流的 `isMute` 方法时触发。通过 `stream` 对象能拿到该用户 `userId`，参数 `mute` 为远端参会用户（或远端主播用户）更新后的值，YES 代表音频静默，NO 代表恢复正常。

```
- (void)stream:(RCRTCInputStream *)stream didAudioMute:(BOOL)mute;
```

5. 远端参会用户（或远端主播用户）视频静默状态变更通知：

当远端参会用户（或远端主播用户）调用了视频流的 `isMute` 方法时触发。参数 `mute` 为远端参会用户（或远端主播用户）更新后的值，YES 代表视频静默，NO 代表恢复正常。

```
- (void)stream:(RCRTCInputStream *)stream didVideoEnable:(BOOL)enable;
```

资源相关

1. 远端参会用户（或远端主播用户）资源发布通知：

当远端参会用户（或远端主播用户）发布资源时触发，`streams` 为该用户当前发布流的集合。从中可以获取发送人（`userId`），流标签（`tag`），媒体类型（`type`），当前状态（`state`）等信息，也可以调用 `subscribeStream` 接口，订阅其中的流。

```
- (void)didPublishStreams:(NSArray <RCRTCInputStream *> *)streams;
```

2. 远端参会用户（或远端主播用户）资源取消发布通知：

当远端参会用户（或远端主播用户）取消发布资源时触发，接收到后 SDK 会自动取消订阅这些流。开发者也可以根据这些流中的信息，来给用户做出相应的提示。

```
- (void)didUnpublishStreams:(NSArray<RCRTCInputStream *> *)streams;
```

数据相关

1. 第一个音视频关键帧到达通知：

```
- (void)didReportFirstFrame:(RCRTCInputStream *)stream mediaType:(RCRTCMediaType)mediaType;
```

2. 房间自定义消息到达通知：

```
- (void)didReceiveMessage:(RCMessage *)message;
```

自定义房间属性

更新时间:2024-08-30

[RCRTCRoom](#) 类提供在音视频房间属性扩展功能。在音视频房间中，可以设置、获取和删除扩展属性。

设置房间属性

使用 `setAttribute` 设置房间属性。

```
[self.room setAttribute:[info toJsonString]
forKey:@"属性 Key 值"
message:message
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

参数	类型	说明
attributeValue	NSString	属性值
key	NSString	属性名称
message	RCMessageContent	是否在设置属性的时候携带消息内容，传空则不往房间中发送消息，可以通过自定义消息通知其他端有新属性已经设置
completion	RCRTCOperationCallback	设置完成回调 block

获取房间属性

```
[self.room getAttributes:@{@"属性 Key 值"}
completion:^(BOOL isSuccess, RCRTCCode code, NSDictionary * _Nullable attr) {
}];
```

参数	类型	说明
attributeKeys	NSArray	属性 Key 值列表
completion	RCRTCAttributeOperationCallback	查询结果的回调 block

删除属性

```
[self.room deleteAttributes:@[@" Key 值"]
message:deleteMessage
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

参数	类型	说明
attributeKeys	NSArray	属性 Key 值列表
message	RCMessageContent ↗	是否在设置属性的时候携带消息内容，传空则不往房间中发送消息
completion	RCRTCOperationCallback	删除完成回调 block

属性变化回调

在设置和删除自定义房间属性时，房间内的其他用户在通过 [RCRTCRoomEventDelegate](#) [↗](#) 下面的代理方法，回调接收自定义的房间属性消息 [RCMessage](#) [↗](#)。

```
– (void)didReceiveMessage:(RCMessage *)message;
```

- 参数说明：

参数	类型	说明
message	RCMessage ↗	接收到其他人发送到 room 里的消息体，参考 IMLib 中 RCMessage

设备检测

更新时间:2024-08-30

在直播与会议开始前，用户可以通过设备检测接口，查看自己的设备是否都处在可用状态。

提示

尽量避免在直播或会议过程中调用设备音频测试接口测试，否则会影响正常业务。

检测麦克风 & 扬声器

从 v5.1.9 版本开始，SDK 支持在直播或会议前，通过 `startEchoTest:` 方法测试音频设备（麦克风、扬声器）。实现该接口后，用户可在直播前自行检测设备的音频是否正常。

开始前请确保已在项目中实现了基本的音视频功能，调用 `startEchoTest:` 方法。调用该方法时，您需要设置一个 `interval` 参数，表示获取本次测试结果的间隔时间。该参数单位为 s，取值范围为 [2 10]。10 表示 10 秒后播放本次测试录到的声音，获取测试结果。用户可以等待并检查是否可以听到自己的声音回放。

如果 `interval` 参数输入值 小于 2，按 2s 处理，大于 10 按 10s 处理。

```
[[RCRTCEngine sharedInstance] startEchoTest:10];
```

成功调用 `startEchoTest:` 方法后，引导用户先说一段话，如果声音在设置的时间间隔后回放出来，且用户能听到自己刚才说的话，则表示系统音频设备和网络连接都是正常的。获取音频设备测试结果后，调用 `stopEchoTest` 方法停止语音通话检测。

调用 `startEchoTest:` 后必须调用 `stopEchoTest` 以结束测试，否则不能进行下一次设备检测测试。

```
[[RCRTCEngine sharedInstance] stopEchoTest];
```


摄像头

更新时间:2024-08-30

[RCRTCCameraOutputStream](#) 对象中提供了摄像头的管理方法。

应用程序不能自己创建 [RCRTCCameraOutputStream](#) 对象。请在 IM 连接成功后调用 [RCRTCEngine](#) 中的 [defaultVideoStream](#) 属性获取 [RCRTCCameraOutputStream](#) 对象。

打开摄像头

使用 [startCapture](#) 方法打开摄像头。

```
[[RCRTCEngine sharedInstance].defaultVideoStream startCapture];
```

关闭摄像头

使用 [stopCapture](#) 方法关闭摄像头。

```
[[RCRTCEngine sharedInstance].defaultVideoStream stopCapture];
```

切换摄像头

使用 [switchCamera](#) 方法切换采集摄像头。

```
[[RCRTCEngine sharedInstance].defaultVideoStream switchCamera];
```

选择摄像头

使用 [cameraPosition](#) 方法切换采集摄像头。通过 [RCRTCDeviceCamera](#) 指定摄像头。

```
[RCRTCEngine sharedInstance].defaultVideoStream.cameraPosition = RCRTCDeviceFront;
```

手动对焦

使用 [isCameraFocusSupported](#) 方法判断摄像头是否支持区域对焦；如果支持，还可以继续调用 [setCameraFocusPositionInPreview](#) 传入对焦点的坐标，来对对焦区域进行设置，坐标原点在屏幕的左上角。

```
if ([[RCRTCEngine sharedInstance].defaultVideoStream isCameraFocusSupported])
{
CGPoint pt; // 需要对焦的点
[[RCRTCEngine sharedInstance].defaultVideoStream setCameraFocusPositionInPreview:pt];
}
```

区域测光

使用 [isCameraExposurePositionSupported](#) 方法判断设备是否支持区域测光；如果支持，还可以继续调用 [setCameraExposurePositionInPreview](#) 设置测光点，设置区域测光的坐标原点为视频区域的左上角。

```
if ([[RCRTCEngine sharedInstance].defaultVideoStream isCameraExposurePositionSupported])
{
CGPoint pt; // 需要测光的点
[[RCRTCEngine sharedInstance].defaultVideoStream setCameraExposurePositionInPreview:pt];
}
```

闪光灯

使用 [isCameraTorchSupported](#) 方法判断摄像头是否支持闪光灯开关；如果支持可调用 [enableCameraTorch](#) 传入 BOOL 值，进行闪光灯开启和关闭。

```
if ([[RCRTCEngine sharedInstance].defaultVideoStream isCameraTorchSupported])
{
//打开闪光灯
[[RCRTCEngine sharedInstance].defaultVideoStream enableCameraTorch:YES];
//关闭闪光灯
[[RCRTCEngine sharedInstance].defaultVideoStream enableCameraTorch:NO];
}
```

缩放比例

使用 [isCameraZoomSupported](#) 方法判断摄像头是否支持设置缩放比例；如果支持，还可以继续调用 [getCameraMaxZoomFactor](#) 和 [setCameraZoomFactor](#) 传入对应参数，来对修改视频效果。

```
if ([[RCRTCEngine sharedInstance].defaultVideoStream isCameraZoomSupported])
{
//获取最大缩放比例
float zoomFactor = [[RCRTCEngine sharedInstance].defaultVideoStream getCameraMaxZoomFactor];
//设置摄像头缩放比例
[[RCRTCEngine sharedInstance].defaultVideoStream setCameraZoomFactor:2.0];
}
```

视频采集方向

使用 [RCRTCCTCameraOutputStream](#) 的 [videoOrientation](#) 属性设置摄像头的采集方向，默认值为 `AVCaptureVideoOrientationPortrait`。

```
[RCRTCEngine sharedInstance].defaultVideoStream.videoOrientation = AVCaptureVideoOrientationPortrait;
```

采集镜像

使用 [RCRTC CameraOutputStream](#) 的 [isPreviewMirror](#) 属性设置摄像头是否镜像采集，前置摄像头默认镜像采集，后置摄像头默认非镜像采集。

```
//非镜像  
[RCRTCEngine sharedInstance].defaultVideoStream.isPreviewMirror = NO;  
//镜像  
[RCRTCEngine sharedInstance].defaultVideoStream.isPreviewMirror = YES;
```

麦克风

禁用麦克风

更新时间:2024-08-30

SDK 封装了麦克风输出音频流对象 `RCRTCMicOutputStream`，App 可在获取此对象后通过 `setMicrophoneDisable` 方法禁用麦克风设备采集。

```
- (void)setMicrophoneDisable:(BOOL)disable;
```

参数	类型	说明
disable	BOOL	采集运行中关闭或打开麦克风，YES 关闭 NO 打开

App 无法自行创建 [RCRTCMicOutputStream](#) 对象。请通过 `[RCRTCEngine sharedInstance].defaultAudioStream` 获取该对象后进行设置：

```
// 设置禁用麦克风采集  
[[RCRTCEngine sharedInstance].defaultAudioStream setMicrophoneDisable:YES];
```

关于如何设置麦克风采集音量，以及如何开启耳返功能、设置耳返音量，详见 [音量](#)。

扬声器

更新时间:2024-08-30

扬声器播放的是接收到的声音，此方法在 RCRTCEngine 中，可以无外接音频输出设备的情况下切换到外放播放声音。

提示

关于如何切换使用听筒与扬声器，详见[音频路由](#)。

```
/*!  
设置是否切换听筒为扬声器  
  
@param enable YES 使用扬声器 ; NO 不使用  
@discussion  
切换听筒/扬声器  
  
@remarks RCRTCEngine : 媒体流操作  
@return 接入外设时，如蓝牙音箱等返回 NO  
*/  
- (BOOL)enableSpeaker:(BOOL)enable;
```

参数说明：

参数	类型	说明
enable	BOOL	YES 使用扬声器 NO 使用听筒

示例代码：

```
[[RCRTCEngine sharedInstance] enableSpeaker:YES];
```

本地用户流 发布

更新时间:2024-08-30

开发者加入房间成功后，可以通过加入房间返回的 [RCRTCRoom](#) 对象发布本地默认音视频流，包括：麦克风采集的音频和摄像头采集的视频。发布自定义视频流，请参考：

```
[[RCRTCEngine sharedInstance].room.localUser publishDefaultStream:^(BOOL isSuccess, RCRTCCode desc) {  
}];
```

参数	类型	说明
completion	RCRTCOperationCallback	发布完成回调

取消发布

发布本地默认音视频流后，可以调用 `RCRTCLocalUser` 对象取消发布本地默认音视频流。退出房间时不需要调用此方法，SDK 内部会做取消发布的处理。

```
[[RCRTCEngine sharedInstance].room.localUser unpublishDefaultStream:^(BOOL isSuccess, RCRTCCode desc) {  
}];
```

参数	类型	说明
completion	RCRTCOperationCallback	取消发布完成回调

远端用户流

订阅

更新时间:2024-08-30

使用 `subscribeStreams` 方法订阅房间内其他用户的资源，可选择订阅大流或小流，大小流通过 `avStreams` 和 `tinyStreams` 区分。

以下两个时机需要处理订阅逻辑：

- 加入房间成功后通过 `RCRTCRoom` 对象取得用户资源列表，需要处理订阅逻辑。
- 资源发布代理触发后需要处理订阅逻辑。

```

- (void)subscribeRemoteResource:(NSArray<RCRTCInputStream*> *)streams {
    NSArray *tinyStream = isTiny ? streams : @[];
    NSArray *ordinaryStream = isTiny ? @[] : streams;

    // 订阅房间中远端用户音视频流资源
    [[RCRTCEngine sharedInstance].room.localUser subscribeStreams:ordinaryStream
    tinyStreams:tinyStream
    callback:^(BOOL isSuccess, RCRTCCode desc, NSArray<RCRTCInputStream *> * _Nullable subscribeErrorList) {

        if (desc != RCRTCCodeSuccess) {
            return;
        }

        if (subscribeErrorList.count) {
            // 您可以根据业务决定是否对 subscribeErrorList 中的流进行重新订阅，或仅作提示
        }

        // 创建并设置远端视频预览视图
        NSInteger i = 0;
        for (RCRTCInputStream *stream in streams) {
            if (stream.mediaType == RTCMediaTypeVideo) {
                BOOL isSubErrorStream = NO;
                for (RCRTCInputStream *subErrorStream in subscribeErrorList) {
                    if ([stream.streamId isEqualToString:subErrorStream.streamId] && stream.mediaType ==
                    subErrorStream.mediaType) {
                        isSubErrorStream = YES;
                        break;
                    }
                }
                if (isSubErrorStream) {
                    continue;
                }
                if (i==0) {
                    // 1.初始化渲染远端视频的 view
                    RCRTCRemoteVideoView *view = [[RCRTCRemoteVideoView alloc] initWithFrame:CGRectMake(100, 400, 100,
                    100)];
                    // 2.设置视频流的渲染视图
                    [(RCRTCVideoInputStream *)stream setVideoView:view];
                    // 3.添加渲染视图
                    [self.view addSubview:view];
                }else{
                    // 其他远端视图逻辑
                }
                i++;
            }
        }
    }];
}

```

参数	类型	说明
avStreams	RCRTCInputStream	普通流
tinyStreams	RCRTCInputStream	需要携带小流的流数组
callback	RCRTCSubscribeOperationCallback	订阅资源完成的回调。callback 中 subscribeErrorList 为服务器返回的订阅失败列表。如果 isSuccess 为 YES 但是 subscribeErrorList 不为空，表示部分订阅成功，您可以根据业务决定是否对 subscribeErrorList 中的流进行重新订阅

取消订阅

使用 `unsubscribeStream` 方法取消订阅一个或多个音视频流 `RCRTCInputStream`。

```
- (void)unsubscribeStream:(NSArray <RCRTCInputStream *> *)streams  
completion:(nonnull RCRTCOperationCallback)completion;
```

参数	类型	说明
streams	NSArray	发布的音视频流 <code>RCRTCInputStream</code> 数组
completion	RCRTCOperationCallback	取消订阅的音视频流的回调

主播端

准备发布

更新时间:2024-08-30

1. 主播端初始化 SDK 并连接融云服务器。

```
[[RCCoreClient sharedCoreClient] initWithAppKey:@"你的AppKey"];
[[RCCoreClient sharedCoreClient] connectWithToken:@"你的token"
dbOpened:nil
success:^(NSString *userId) {

NSLog(@"IM connect success,user ID : %@",userId);
...

} error:^(RCConnectErrorCode errorCode) {
NSLog(@"IM connect failed, error code : %ld", (long)errorCode);
}];
```

2. (可选) App 可以先将音频输出切换到扬声器。详见 [音频路由](#)。

```
// 1.设置切换听筒为扬声器
[[RCRTCEngine sharedInstance] enableSpeaker:YES];
```

3. 主播端需要配置房间信息并加入房间。

```
// 配置房间
RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];
config.roomType = RCRTCRoomTypeLive;
config.liveType = RCRTLIVETypeAudioVideo;
config.roleType = RCRTLIVERoleTypeBroadcaster;
[[RCRTCEngine sharedInstance] joinRoom:@"你的房间号"
config:config
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {
if (code != RCRTCCodeSuccess) {
}
// 可以通过实现 <RCRTCRoomEventDelegate> 的方法来监听房间相关事件回调
room.delegate = self;
}];
```

4. 主播端创建本地视频渲染的 view。

```
// 1.初始化本地渲染视图
RCRTCVideoView *view = [[RCRTCVideoView alloc] initWithFrame:CGRectMake(100, 100, 100, 100)];
// 2.设置视频流的渲染视图
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoView:view];
// 3.添加渲染视图
[self.view addSubview:view];
```

5. 开始摄像头采集。

```
[[RCRTCEngine sharedInstance].defaultVideoStream startCapture];
```

发布资源

① 提示

发布资源前，请确定已经以主播身份加入房间、开启摄像头采集以及设置完视频流的渲染视图并添加在本地视图上以供显示。

1. 当 [RCRTCRoomType](#) 为 [RCRTCRoomTypeLive](#) 时，主播在配置完房间并加入后可以发布默认音视频流。此接口仅直播模式的主播可用。

```
- (void)publishDefaultLiveStreams:(RCRTLIVEOperationCallback)completion;
```

```
[[RCRTCEngine sharedInstance].room.localUser publishDefaultLiveStreams:^(BOOL isSuccess, RCRTCCode desc, RCRTLIVEInfo * _Nullable liveInfo) {
// 可以通过拿到的liveInfo对象进行推流相关配置
}];
```

2. 如果主播想要单独发布默认音频流，视频流或者自定义视频流时，可以使用另外一个发布接口。此接口仅直播模式的主播可用。详情可跳转[视频流相关-发布自定义流](#)中查看。

```
- (void)publishLiveStream:(nonnull RCRTCOutputStream *)stream completion:(nonnull RCRTLIVEOperationCallback)completion;
```

参数	类型	说明
stream	RCRTCOutputStream	发布的音视频流
completion	RCRTCLiveOperationCallback	发布完成回调

示例代码：

```
// 创建本地渲染视图
RCRTCVideoView *localFileVideoView = [[RCRTCVideoView alloc] initWithFrame:CGRectMake(0, 0, 100, 100)];
localFileVideoView.fillMode = RCRTCVideoFillModeAspectFit;
localFileVideoView.frameAnimated = NO;
[self.view addSubview:localFileVideoView];

// 创建自定义视频流
NSString *tag = @"RongRTCFileVideo";
RCRTCVideoOutputStream *fileVideoOutputStream = [[RCRTCVideoOutputStream alloc]
initWithVideoOutputStreamWithTag:tag];

// 设置自定义视频流配置
RCRTCVideoStreamConfig *videoConfig = fileVideoOutputStream.videoConfig;
videoConfig.videoSizePreset = RCRTCVideoSizePreset640x480;
[fileVideoOutputStream setVideoConfig:videoConfig];
[fileVideoOutputStream setVideoView:localFileVideoView];

// 设置自定义视频流的视频源
NSString *path = [[NSBundle mainBundle] pathForResource:@"video_demo1_low"
ofType:@"mp4"];
RCRTCFileSource *fileCapturer = [[RCRTCFileSource alloc] initWithFilePath:path];
fileCapturer.delegate = self;
fileVideoOutputStream.videoSource = fileCapturer;
[fileCapturer addObserver:fileVideoOutputStream];

// 发布自定义视频流
[[RCRTCEngine sharedInstance].room.localUser publishLiveStream:fileVideoOutputStream
completion:^(BOOL isSuccess, RCRTCCode code, RCRTCLiveInfo * _Nullable liveInfo) {
if (code == RCRTCCodeSuccess) {
}
}
}];
```

取消发布

提示

主播发布本地资源后也可以取消发布，主播专用取消发布接口有 2 个。

1. 取消发布本地默认音视频流，此接口仅直播模式的主播可用，即 [RCRTCRoomType](#) 为 [RCRTCRoomTypeLive](#) 可用。

```
- (void)unpublishDefaultLiveStreams:(RCRTCOperationCallback)completion;
```

```
[[RCRTCEngine sharedInstance].room.localUser unpublishDefaultLiveStreams:^(BOOL isSuccess, RCRTCCode code) {
}];
```

- 取消发布本地指定音视频流，此接口仅直播模式的主播可用, 即 [RCRTCRoomType](#) 为 [RCRTCRoomTypeLive](#) 可用。

```
[[RCRTCEngine sharedInstance].room.localUser unpublishLiveStream:self.fileVideoOutputStream
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

参数	类型	说明
stream	RCRTCOutputStream	取消发布的音视频流
completion	RCRTCOperationCallback	取消发布的音视频流完成回调

获取资源

提示

对于主播身份，远端用户音视频流可以通过以下两种方式取得：

- 加入房间前远端用户已经发布资源，则在加入房间后取得的 [RCRTCRoom](#) 对象的 [RCRTCRemoteUser](#) 中的 `remoteStreams` 中取得。

```
[[RCRTCEngine sharedInstance] joinRoom:_roomId
config:config
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {

// 参与的远端用户
if (room.remoteUsers.count) {
NSMutableArray *streamArray = [NSMutableArray array];
for (RCRTCRemoteUser *user in room.remoteUsers) {

// 参与的远端用户的音视频流
if (user.remoteStreams.count) {

// 可以在此订阅远端音视频资源
}
}
}
}];
```

- 加入房间后远端用户才发布资源，可以通过房间代理方法取得音视频流数组。

```

- (void)didPublishStreams:(NSArray<RCRTCInputStream *> *)streams {
// 可以在此订阅远端音视频资源
}

```

订阅资源

远端用户发布音视频流后，可以通过加入房间后返回的 [RCRTCRoom](#) 中的订阅方法订阅多路远端指定音视频流。同一个流只能填写在 `avStreams` 或 `tinyStreams` 中的一个数组中。

```

NSArray *tinyStream = isTiny ? streams : @[];
NSArray *ordinaryStream = isTiny ? @[] : streams;
// 订阅房间中远端用户音视频流资源
[[RCRTCEngine sharedInstance].room.localUser subscribeStream:ordinaryStream
tinyStreams:tinyStream
completion:^(BOOL isSuccess, RCRTCCode desc) {
if (desc != RCRTCCodeSuccess) {
return;
}
// 创建并设置远端视频预览视图
NSInteger i = 0;
for (RCRTCInputStream *stream in streamArray) {
if (stream.mediaType == RTCMediaTypeVideo) {
if (i==0) {
// 1.初始化渲染远端视频的 view
RCRTCRemoteVideoView *view = [[RCRTCRemoteVideoView alloc] initWithFrame:CGRectMake(100, 400, 100, 100)];
// 2.设置视频流的渲染视图
[(RCRTCVideoInputStream *)stream setVideoView:view];
// 3.添加渲染视图
[self.view addSubview:view];
}else{
}
// 其他远端视图逻辑
}
}
}
}];

```

参数	类型	说明
<code>avStreams</code>	<code>RCRTCInputStream</code>	普通流
<code>tinyStreams</code>	<code>RCRTCInputStream</code>	需要携带小流的流数组
<code>completion</code>	<code>RCRTCOperationCallback</code>	订阅资源完成的回调

取消订阅

1. 通过代理收到远端用户取消发布音视频流后，可以调用 [RCRTCRoom](#) 中 [RCRTCLocalUser](#) 的取消订阅的方法，取消订阅该用户的音视频流。退出房间时不需要取消订阅，SDK 内部会处理。

```
[[RCRTCEngine sharedInstance].room.localUser unsubscribeStreams:streams
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

参数	类型	说明
streams	NSArray	音视频流集合
completion	RCRTCOperationCallback	取消订阅完成的回调

2. 也可以取消订阅远端指定音视频流。

```
[[RCRTCEngine sharedInstance].room.localUser unsubscribeStream:stream
completion:^(BOOL isSuccess,RCRTCCode desc) {
}];
```

参数	类型	说明
stream	RCRTCInputStream	音视频流
completion	RCRTCOperationCallback	取消订阅完成的回调

观众端

获取资源

更新时间:2024-08-30

直播模式中主播发布的音视频流，会在服务端另行合并生成一道音频合流和一道视频合流。观众既可以直接订阅原始音视频流（简称“分流”），也可订阅“合流”。下面将分别介绍这两种订阅方式的区别和适用场景。

提示

观众被定义为只能订阅不能发布，如需发布必须先转为主播身份，再进行资源的发布。

获取分流

分流订阅跟会议模式的订阅一样，适合小众玩法灵活的直播业务场景。比如观众被分成多种角色，需要选择性观看或收听部分主播发布的资源；又或者不同观众的展示布局并不相同，甚至可以随时切换布局的情况，SDK 提供视频展示 View 的创建和与流绑定接口，开发者自行编写展示逻辑。

观众的订阅需要在两个地方进行处理：一是在加入直播间的成功回调里，需要遍历得到房间内已经存在用户发布的流，并订阅；二是在收到主播刚刚发布流的通知，即 [didPublishStreams](#) 时订阅。

- 加入房间成功后，在回调中获取分流。

```
[[RCRTCEngine sharedInstance] joinRoom:@"你的房间id"
config:config
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {

if (room.remoteUsers.count) {
NSMutableArray *streamArray = [NSMutableArray array];
for (RCRTCRemoteUser *user in room.remoteUsers) {
if (user.remoteStreams.count) {
[streamArray addObjectFromArray:user.remoteStreams];
}
}
// 可以在此订阅streamArray
[self subscribeRemoteResource:streamArray];
}

}];
```

- 通过实现 [RCRTCRoomEventDelegate](#) 的 [didPublishStreams](#) 获取分流。

```
- (void)didPublishStreams:(NSArray<RCRTCInputStream * > *)streams {
//可以在此订阅streams
[self subscribeRemoteResource:streams];
}
```


获取合流

- 加入房间前远端直播合流已经发布，则通过加入房间后 [RCRTCRoom](#) 对象的 [getLiveStreams](#) 方法获得已经存在的直播合流。

```
[[RCRTCEngine sharedInstance] joinRoom:@"你的房间id"
config:config
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {

// 观众获取 live 合流
NSArray *liveStreams = [room getLiveStreams];
if (liveStreams.count) {

// 观众可以在此订阅合流
}
}];
```

- 加入房间后远端直播合流发布，可以通过房间代理方法取得音视频流数组。

```
- (void)didPublishLiveStreams:(NSArray<RCRTCInputStream*> *)streams {
// 观众可以在此订阅合流
}
```

订阅资源

远端用户发布音视频流后，可以通过加入房间后返回的 [RCRTCRoom](#) 中的 `subscribeStreams` 方法订阅多路远端指定音视频流。同一个流只能填写在 `avStreams` 或 `tinyStreams` 中的一个数组中。

```

- (void)subscribeRemoteResource:(NSArray<RCRTCInputStream*> *)streams {
    NSArray *tinyStream = isTiny ? streams : @[];
    NSArray *ordinaryStream = isTiny ? @[] : streams;

    // 订阅房间中远端用户音视频流资源
    [[RCRTCEngine sharedInstance].room.localUser subscribeStreams:ordinaryStream
    tinyStreams:tinyStream
    callback:^(BOOL isSuccess, RCRTCCode desc, NSArray<RCRTCInputStream *> * _Nullable subscribeErrorList) {

        if (desc != RCRTCCodeSuccess) {
            return;
        }

        if (subscribeErrorList.count) {
            // 您可以根据业务决定是否对 subscribeErrorList 中的流进行重新订阅，或仅作提示
        }

        // 创建并设置远端视频预览视图
        NSInteger i = 0;
        for (RCRTCInputStream *stream in streams) {
            if (stream.mediaType == RTCMediaTypeVideo) {
                BOOL isSubErrorStream = NO;
                for (RCRTCInputStream *subErrorStream in subscribeErrorList) {
                    if ([stream.streamId isEqualToString:subErrorStream.streamId] && stream.mediaType ==
                    subErrorStream.mediaType) {
                        isSubErrorStream = YES;
                        break;
                    }
                }
                if (isSubErrorStream) {
                    continue;
                }
                if (i==0) {
                    // 1.初始化渲染远端视频的 view
                    RCRTCRemoteVideoView *view = [[RCRTCRemoteVideoView alloc] initWithFrame:CGRectMake(100, 400, 100,
                    100)];
                    // 2.设置视频流的渲染视图
                    [(RCRTCVideoInputStream *)stream setVideoView:view];
                    // 3.添加渲染视图
                    [self.view addSubview:view];
                }else{
                    // 其他远端视图逻辑
                }
                i++;
            }
        }
    }];
}

```

参数	类型	说明
avStreams	RCRTCInputStream	普通流
tinyStreams	RCRTCInputStream	需要携带小流的流数组
callback	RCRTCSubscribeOperationCallback	订阅资源完成的回调。callback 中 subscribeErrorList 为服务器返回的订阅失败列表。如果 isSuccess 为 YES 但是 subscribeErrorList 不为空，表示部分订阅成功，您可以根据业务决定是否对 subscribeErrorList 中的流进行重新订阅

取消订阅

观众可以调用 [RCRTCRoom](#) 的 `unsubscribeStreams` 方法取消订阅，取消订阅该用户的音视频流。退出房间时不需要取消订阅，SDK 内部会处理。

```
[[RCRTCEngine sharedInstance].room.localUser unsubscribeStreams:streams
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

参数	类型	说明
streams	NSArray	音视频流 RCRTCInputStream 集合
completion	RCRTCOperationCallback	取消订阅完成的回调

也可以取消订阅远端指定音视频流 [RCRTCInputStream](#) 对象。

```
[[RCRTCEngine sharedInstance].room.localUser unsubscribeStream:stream
completion:^(BOOL isSuccess, RCRTCCode desc) {
}];
```

合流布局

合流布局

更新时间:2024-08-30

直播合流相关接口仅针对主播可用，分为视频合流和音频合流。

视频合流布局

视频合流布局分为 3 种：自定义布局、悬浮布局 和 自适应布局，主播调用接口 `publishLiveStream` 或 `publishDefaultLiveStreams` 发布资源成功后，通过 Block 回调会返回直播合流信息接口类对象 `RCRTCLiveInfo` (opens new window)，App 层需要缓存，以便设置合流布局使用，然后可以通过类 `RCRTCMixConfig` (opens new window)来进行合流布局配置，再通过类 `RCRTCLiveInfo` (opens new window)中 `setMixConfig` 方法进行设置。

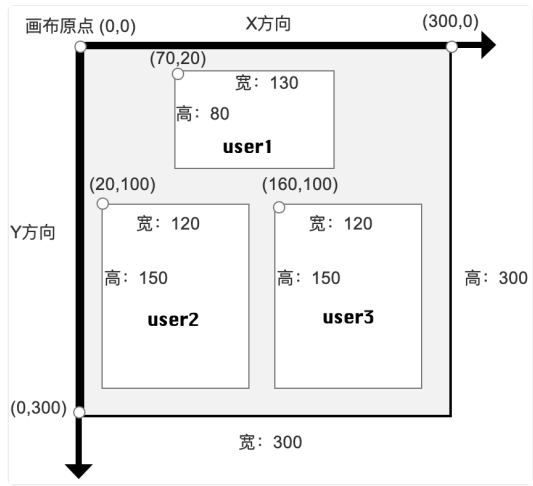
从 5.1.1 版本 (`RCRTCMixConfig.version == 2`) 开始，`RCRTCMixConfig` 中新增加了 `customMode` 属性，功能为自定义模式开关，打开后，用户可以更加灵活的控制合流布局，建议将其设置为 YES，新版本兼容老版本逻辑，升级 SDK 不需要修改代码。

```
// 自定义合流布局
- (void)streamlayoutMode:(RCRTCMixLayoutMode)mode {
RCRTCMixConfig *config = [LiveMixStreamTool setOutputConfig:mode];
[self.liveInfo setMixConfig:config completion:^(BOOL isSuccess, RCRTCCode code) {
if (isSuccess) {
//dosomething
} else {
NSLog(@"%ld", (long)code);
}
}];
}
```

视频自定义布局

视频自定义布局可以根据 `RCRTCMixConfig.customLayouts` 视频流列表，设置各个连麦者视图位置及大小，根据 `RCRTCMixConfig.mediaConfig.videoConfig.videoLayout` 和 `RCRTCMixConfig.mediaConfig.videoConfig.tinyVideoLayout`，分别设置合流后视频大小流属性（大流默认值是 360 * 640，25Fps，1200 kbps；小流默认值 180 * 320，15Fps，360kbps）。

例如需要合流三个用户的视频流，合流后整体视频尺寸 宽*高 = 300 * 300，那么就是以整体视频作为画布，画布的原点 (0,0) 在左上角，三个用户视频窗口按照 `RCRTCMixConfig.customLayouts` 列表顺序，分别相对原点的位置进行绘制，整体效果如下图所示：



```

// 布局配置类
RCRTCMixConfig *streamConfig = [[RCRTCMixConfig alloc] init];
// 选择混流模式，自定义布局开/关效果一致
streamConfig.customMode = YES;
// 选择模式
streamConfig.layoutMode = RCRTCMixLayoutModeCustom;
// 设置合流后视频参数：宽：300，高：300，视频帧率 20，视频码率 500，背景色 0x778899；
streamConfig.mediaConfig.videoConfig.videoLayout.width = 300;
streamConfig.mediaConfig.videoConfig.videoLayout.height = 300;
streamConfig.mediaConfig.videoConfig.videoLayout.fps = 20;
streamConfig.mediaConfig.videoConfig.videoLayout.bitrate = 500;
[streamConfig.mediaConfig.videoConfig setBackgroundColor:0x778899];
// 设置是否裁剪
streamConfig.mediaConfig.videoConfig.videoExtend.renderMode = RCRTCVideoRenderModeCrop;

// 设置合流视频列表
// 添加本地视频流 (user1)
RCRTCCustomLayout *inputConfigUser1 = [[RCRTCCustomLayout alloc] init];
inputConfigUser1.videoStream = [RCRTCEngine sharedInstance].defaultVideoStream;
// 坐标示例，具体根据自己布局设置
inputConfigUser1.x = 20;
inputConfigUser1.y = 70;
inputConfigUser1.width = 130;
inputConfigUser1.height = 80;
[streamConfig.customLayouts addObject:inputConfigUser1];

// 添加远端视频流 (user2, user3)，以下假设远端视频流有 2 个
NSMutableArray *remoteStreamArr = [NSMutableArray array];
NSArray<RCRTCRemoteUser * > *remoteUsers = [RCRTCEngine sharedInstance].room.remoteUsers;
for (RCRTCRemoteUser* remoteUser in remoteUsers) {
    for (RCRTCInputStream *inputStream in remoteUser.remoteStreams) {
        if (inputStream.mediaType == RTCMediaTypeVideo) {
            [remoteStreamArr addObject:inputStream];
        }
    }
}

RCRTCCustomLayout *inputConfigUser2 = [[RCRTCCustomLayout alloc] init];
inputConfigUser2.videoStream = remoteStreamArr[0];
// 坐标示例，具体根据自己布局设置
inputConfigUser2.x = 20;
inputConfigUser2.y = 100;
inputConfigUser2.width = 120;
inputConfigUser2.height = 150;
[streamConfig.customLayouts addObject:inputConfigUser2];

RCRTCCustomLayout *inputConfigUser3 = [[RCRTCCustomLayout alloc] init];
inputConfigUser3.videoStream = remoteStreamArr[1];
// 坐标示例，具体根据自己布局设置
inputConfigUser3.x = 160;
inputConfigUser3.y = 100;
inputConfigUser3.width = 120;
inputConfigUser3.height = 150;
[streamConfig.customLayouts addObject:inputConfigUser3];

```

视频悬浮布局

视频悬浮布局混流画布采用 `RCRTCMixConfig.hostVideoStream` 指定的视频流作为背景视频，如果没有设置采用用户第一个发布的视频，其它视频流是按照用户发布视频流的先后顺序进行悬浮小窗绘制（`RCRTCMixConfig.customMode == YES`，会筛选 `RCRTCMixConfig.customLayouts` 列表进行绘制），当有人离开时，系统会自动按照现有发布视频流的次序重新布局。根据 `RCRTCMixConfig.mediaConfig.videoConfig.videoLayout` 和 `RCRTCMixConfig.mediaConfig.videoConfig.tinyVideoLayout`，分别设

置合流后视频大小流属性（大流默认值是 360 * 640，25Fps，1200 kbps；小流默认值 180 * 320，15Fps，360kbps）。

例如需要合流自己本地视频流加上另外六个远端用户的视频流，合流后整体视频尺寸 宽*高 = 300 * 300，整体效果如下图所示：



```
// 布局配置类
RCRTCMixConfig *streamConfig = [[RCRTCMixConfig alloc] init];
// 选择混流模式
streamConfig.customMode = YES;
// 选择模式
streamConfig.layoutMode = RCRTCMixLayoutModeSuspension;
// 设置合流后视频参数：宽：300，高：300，视频帧率 20，视频码率 500，背景色 0x778899；
streamConfig.mediaConfig.videoConfig.videoLayout.width = 300;
streamConfig.mediaConfig.videoConfig.videoLayout.height = 300;
streamConfig.mediaConfig.videoConfig.videoLayout.fps = 20;
streamConfig.mediaConfig.videoConfig.videoLayout.bitrate = 500;
[streamConfig.mediaConfig.videoConfig setBackgroundColor:0x778899];
// 设置是否裁剪
streamConfig.mediaConfig.videoConfig.videoExtend.renderMode = RCRTCVideoRenderModeCrop;

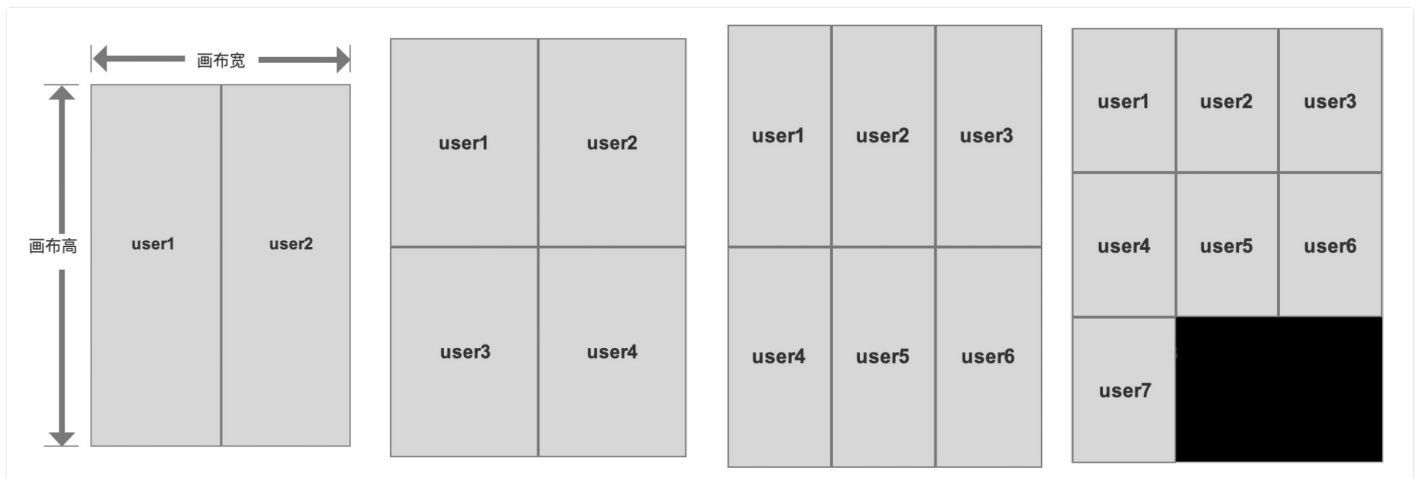
// 设置合流视频列表
// 添加本地视频流 (user0)
RCRTCCustomLayout *inputConfigUser0 = [[RCRTCCustomLayout alloc] init];
inputConfigUser0.videoStream = [RCRTCEngine sharedInstance].defaultVideoStream;
[streamConfig.customLayouts addObject:inputConfigUser0];
// 添加远端视频流 (user2, user3, user4, user5, user6)，以下假设远端视频流有6个
NSArray<RCRTCRemoteUser *> *remoteUsers = [RCRTCEngine sharedInstance].room.remoteUsers;
for (RCRTCRemoteUser* remoteUser in remoteUsers) {
    for (RCRTCInputStream *inputStream in remoteUser.remoteStreams) {
        if (inputStream.mediaType == RTCMediaTypeVideo) {
            RCRTCCustomLayout *inputConfig = [[RCRTCCustomLayout alloc] init];
            inputConfig.videoStream = inputStream;
            [streamConfig.customLayouts addObject:inputConfig];
        }
    }
}
```

视频自适应布局

视频自适应布局按照用户发布视频流的先后顺序进行绘制（RCRTCMixConfig.customMode == YES，会筛选 RCRTCMixConfig.customLayouts 列表进行绘制），绘制原则是按照具体人数平分整体视频区域，使之每个子窗口加载区域大小一致。当有人离开时，系统会自动按照现有发布视频流的次序重新布局。根据 RCRTCMixConfig.mediaConfig.videoConfig.videoLayout 和 RCRTCMixConfig.mediaConfig.videoConfig.tinyVideoLayout，分别设

置合流后视频大小流属性（大流默认值是 360 * 640，25Fps，1200 kbps；小流默认值 180 * 320，15Fps，360kbps）。

例如分别合流 2、4、6、7 个视频流，合流后整体视频尺寸 宽*高 = 300 * 300，整体效果如下图所示：



```
// 布局配置类
RCRTCMixConfig *streamConfig = [[RCRTCMixConfig alloc] init];
// 选择混流模式
streamConfig.customMode = YES;
// 选择模式
streamConfig.layoutMode = RCRTCMixLayoutModeAdaptive;
// 设置合流后视频参数：宽：300，高：300，视频帧率 20，视频码率 500，背景色 0x778899；
streamConfig.mediaConfig.videoConfig.videoLayout.width = 300;
streamConfig.mediaConfig.videoConfig.videoLayout.height = 300;
streamConfig.mediaConfig.videoConfig.videoLayout.fps = 20;
streamConfig.mediaConfig.videoConfig.videoLayout.bitrate = 500;
[streamConfig.mediaConfig.videoConfig setBackgroundColor:0x778899];
// 设置是否裁剪
streamConfig.mediaConfig.videoConfig.videoExtend.renderMode = RCRTCVideoRenderModeCrop;

// 设置合流视频列表
// 添加本地视频流 (user0)
RCRTCCustomLayout *inputConfigUser0 = [[RCRTCCustomLayout alloc] init];
inputConfigUser0.videoStream = [RCRTCEngine sharedInstance].defaultVideoStream;
[streamConfig.customLayouts addObject:inputConfigUser0];
// 添加远端视频流 (user2, user3, user4, user5, user6)，以下假设远端视频流有6个
NSArray<RCRTCRemoteUser *> *remoteUsers = [RCRTCEngine sharedInstance].room.remoteUsers;
for (RCRTCRemoteUser* remoteUser in remoteUsers) {
    for (RCRTCInputStream *inputStream in remoteUser.remoteStreams) {
        if (inputStream.mediaType == RTCMediaTypeVideo) {
            RCRTCCustomLayout *inputConfig = [[RCRTCCustomLayout alloc] init];
            inputConfig.videoStream = inputStream;
            [streamConfig.customLayouts addObject: inputConfig];
        }
    }
}
}
```

音频合流布局

音频合流布局的原理就是通过控制输入源列表（需要合流的音频流列表 RCRTCMixConfig.customMixAudios），设置输出音频配置（合流后的音频流配置 RCRTCMixConfig.mediaConfig.audioConfig），达到合并哪些音频流输出指定配置的音频流。

音频合流会根据音频合流列表 RCRTCMixConfig.customMixAudios 进行音频混音，设置音频合流列表后，观众订阅直播流，就会只听到音频合流列表中对应的用户。


```

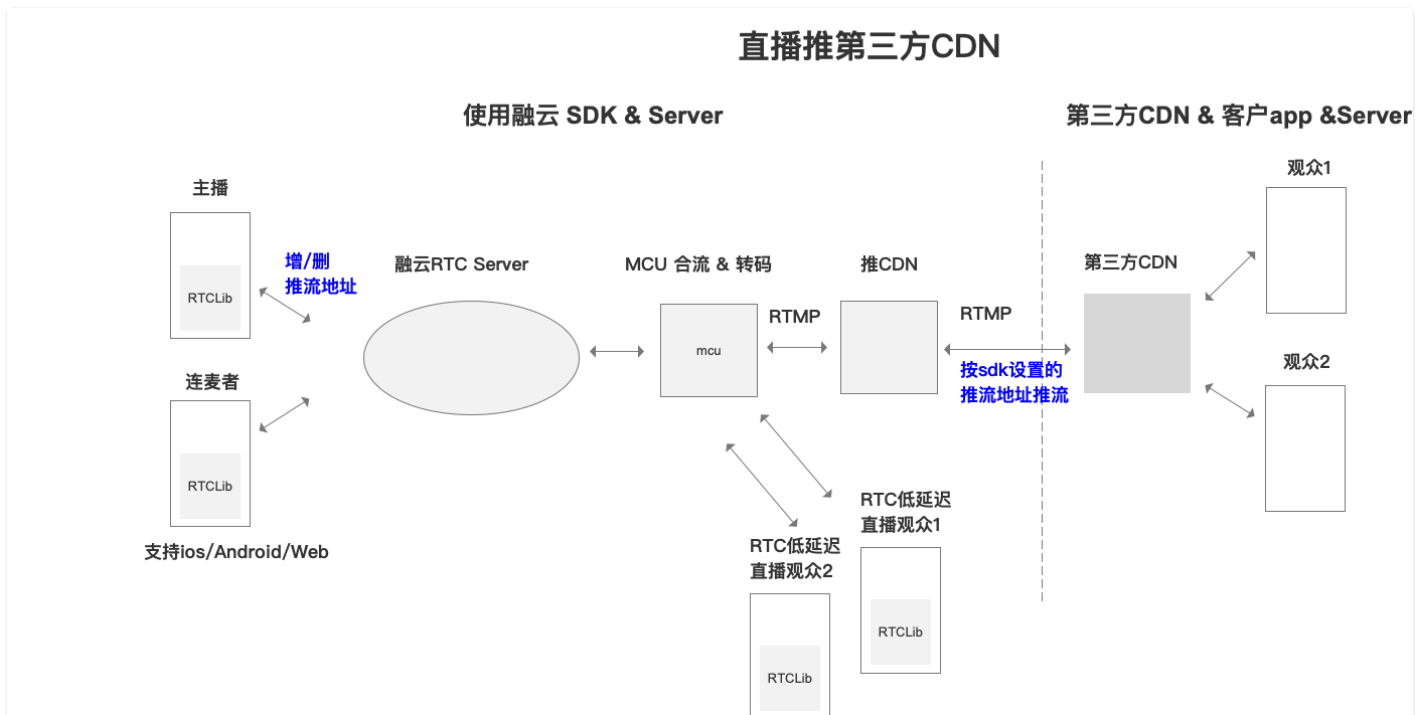
// 布局配置类
RCRTCMixConfig *streamConfig = [[RCRTCMixConfig alloc] init];
// 选择混流模式
streamConfig.customMode = YES;

// 设置合流音频列表
// 添加本地音频流 (user0)
RCRTCCustomMixAudio *mixAudioUser0 = [[RCRTCCustomMixAudio alloc] init];
mixAudioUser0.audioStream = [RCRTCEngine sharedInstance].defaultAudioStream;
[streamConfig.customMixAudios addObject: mixAudioUser0];
// 添加远端音频流 (user2, user3, user4, user5, user6), 以下假设远端音频流有6个
NSArray<RCRTCRemoteUser *> *remoteUsers = [RCRTCEngine sharedInstance].room.remoteUsers;
for (RCRTCRemoteUser* remoteUser in remoteUsers) {
for (RCRTCInputStream *inputStream in remoteUser.remoteStreams) {
if (inputStream.mediaType == RTCMediaTypeAudio) {
RCRTCCustomMixAudio *mixAudioUser = [[RCRTCCustomMixAudio alloc] init];
mixAudioUser.audioStream = [RCRTCEngine sharedInstance].defaultAudioStream;
[streamConfig.customMixAudios addObject: mixAudioUser];
}
}
}
}

```

旁路推流

融云直播支持在使用低延迟直播时转推第三方 CDN，您仅需在参与直播的 App 端调用添加第三方直播推流地址，来进行旁路推流。业务链路如下图所示：



配置直播 CDN 地址

当主播发布资源成功之后，主播就可设置 CDN 推流地址，设置 CDN 地址有以下几点要求：

1. 必须开通音视频服务和直播服务。
2. 房间模式必须为直播模式。
3. 设置的 CDN 地址不能为空。

4. 最多设置 5 个 CDN 地址，超出会抛出 RCRTCCodeCDNCountReachToLimit 错误。
5. 如果多次设置相同的地址，会直接返回成功。

```
[self.liveInfo addPublishStreamUrl:@"https://....." completion:completion];
```

参数	类型	说明
url	NSString	要设置的 CDN 地址
completion	block	设置 CDN 之后的回调，会返回所有设置过的 CDN 地址

移除配置过的 CDN 地址

当主播发布资源成功之后，主播可选择移除一个设置过的 CDN 推流地址，有以下几点要求：

1. 必须开通音视频服务和直播服务。
2. 房间模式必须为直播模式。
3. 移除的 CDN 地址不能为空。
4. 如果移除的地址，之前没有设置过，会直接返回成功。

```
[self.liveInfo removePublishStreamUrl:@"https://....." completion:completion];
```

参数	类型	说明
url	NSString	要移除的 CDN 地址
completion	block	移除 CDN 之后的回调，会返回所有设置过的 CDN 地址

同房间连麦

更新时间:2024-08-30

提示

5.1.9 版本之前如需实现上下麦，需要让观众先退出房间，设置新的角色身份，再重新加入房间。

观众上麦

观众上麦本质上是切换身份变成当前房间的主播，然后以主播身份继续直播相关操作。App 处理步骤如下：

1. （可选）App 可提前获取当前用户需要发布的流。如果在切换为主播时传入流数组，SDK 会在角色切换成功后自动发布。
2. App 调用 [RCRTCLocalUser](#) 的 [switchToBroadcaster](#) 接口，将当前登录用户（观众角色）切换为主播角色。角色切换后，RCRTCRoom 对象中数据会自动刷新。角色切换后会保留之前注册的 [RCRTCStatusReportDelegate](#)、[RCRTCEngineEventDelegate](#) 和 [RCRTCRoomEventDelegate](#) 等 RTC SDK 全局类型的监听，其他监听需要切换角色成功后重新注册才能使用。
3. SDK 内部会取消当前登录用户之前（作为观众）订阅的音视频流。因此，在成功切换为主播角色后，App 需要为当前用户调用 [subscribeStreams](#) 重新订阅。注意，主播角色一般仅订阅其他主播发布的流，不订阅合流。

```
NSArray *streams = @[self.engine.defaultAudioStream,self.engine.defaultVideoStream];

[self.room.localUser switchToBroadcaster:streams onSuccess:^(RCRTCLiveInfo * _Nonnull liveInfo) {
//TODO:切换到主播成功，订阅房间内流资源
NSMutableArray *streams = [NSMutableArray array];
for (RCRTCRemoteUser *user in self.room.remoteUsers) {
if (user.remoteStreams.count) {
[streams addObjectFromArray:user.remoteStreams];
}
}
if (streams.count) {
// 订阅资源
//[self subscribeStreams:streams];
}
} onFailure:^(RCRTCCode code) {
NSLog(@"OnFailed:%@",@(code));
} onKicked:^(
NSLog(@"OnKicked");
});
```

如果切换为主播时 `publishStreams` 为空（不发布资源），App 可以单独调用主播发布资源的方法。详见 [发布与订阅·主播端](#)。

参数	类型	说明
publishStreams	NSArray<RCRTCOutputStream *>	切换成主播，需要发布的流数组（可以传空）。
onSucceed	void (^)(RCRTLIVEInfo *liveInfo)	切换成功的回调，liveInfo 会携带发布资源后的 liveUrl
onFailed	void (^)(RCRTCCode code)	切换失败的回调，不会影响当前观众身份，用户可以继续音视频相关操作
onKicked	void (^)(void)	切换失败，被 SDK 主动踢出的回调，需要用户重新加入房间才能继续音视频的相关操作

- 观众上麦后，RTC 房间内的其他用户（主播、观众）同房间内的其他用户会通过 [RCRTCRoomEventDelegate](#) 协议里面的 `didSwitchRoleWithUser:roleType:` 方法监听到远端用户切换身份为 `RCRTLIVERoleTypeBroadcaster`。
- 观众上麦后，RTC 房间内的其他用户（主播、观众）会收到回调：
 - 其他主播角色用户通过 [RCRTCRoomEventDelegate](#) 协议里面的 `didSwitchRoleWithUser:roleType:` 方法监听到远端用户切换身份为 `RCRTLIVERoleTypeBroadcaster`。
 - 观众角色用户通过 [RCRTCRoomEventDelegate](#) 协议里面的 `didJoinUser` 回调收到通知。
- 观众上麦后，RTC 房间内的其他用户（主播、观众）可通过 [房间事件回调](#) 中的 `didPublishStreams` 回调收到发流通知。App 端需要自行处理资源订阅。

主播下麦

主播下麦本质上是切换身份变成当前房间的观众，然后以观众身份继续观看直播等相关操作。App 处理步骤如下：

- App 调用 [RCRTLIVELocalUser](#) 的 `switchToAudienceOnSucceed` 接口，将当前登录用户（观众角色）切换为主播角色。角色切换后，`RCRTCRoom` 对象中数据会自动刷新。角色切换后会保留之前注册的 [RCRTCStatusReportDelegate](#)、[RCRTCENGINEEventDelegate](#) 和 [RCRTCRoomEventDelegate](#) 等 RTC SDK 全局类型的监听，其他监听需要切换角色成功后重新注册才能使用。
- SDK 内部会取消当前登录用户之前（作为主播）发布、订阅的音视频流。因此，在成功切换为观众角色后，App 需要为当前用户调用 [subscribeStreams](#) 重新订阅。注意，观众角色一般仅订阅合流。

```
[self.room.localUser switchToAudienceOnSucceed:^(
//TODO: 观众订阅
NSArray *streams = [self.room getLiveStreams];
//[self subscribeStreams:streams];
} onFailed:^(RCRTCCode code) {
NSLog(@"OnFailed:%@",@(code));
} onKicked:^(
NSLog(@"OnKicked");
}];
```

参数	类型	说明
onSucceed	void (^)(void)	切换成功的回调
onFailed	void (^)(RCRTCCode code)	切换失败的回调，不会影响当前主播身份，用户可以继续音视频相关操作
onKicked	void (^)(void)	切换失败，被 SDK 主动踢出的回调，需要用户重新加入房间才能继续音视频的相关操作

提示

如果当前用户在跨房间连麦中加入了 RCRTCOtherRoom，切换为观众时，SDK 内部会帮其退出所有 RCRTCOtherRoom。不会结束本次连麦。如果需要结束连麦请在切换角色前调用 RCRTCEngine 的 leaveOtherRoom:notifyFinished:completion: 方法，notifyFinished 参数传 YES 即可。

3. 主播下麦后，RTC 房间内的其他用户（主播、观众）会收到回调：

- 其他主播角色用户通过 RCRTCRoomEventDelegate 协议里面的 didSwitchRoleWithUser:roleType: 方法监听到远端用户切换身份为 RCRTCLiveRoleTypeAudience。
- 观众角色用户通过 RCRTCRoomEventDelegate 协议里面的 didLeaveUser 回调收到通知。

监听房间用户切换角色事件

当房间内的用户使用切换角色方式上下麦时，同房间内用户均会收到回调。其中主播用户会通过 RCRTCRoomEventDelegate 协议里面的 didSwitchRoleWithUser:roleType: 方法监听到远端用户切换身份。

如果主播用户订阅的其他主播下麦，SDK 内部会主动取消订阅不存在的音视频流，但 App 需要在此回调方法中移除当前视频流视图等操作。

```

/*!
远端用户切换身份通知

@param user 切换身份的用户
@param roleType 该用户当前的身份

@discussion
有切换身份时的回调，当有用户切换身份的时候，当前房间内其他主播会收到该消息。
如果订阅了当前 user 的流，SDK 会主动取消订阅，不需要手动取消订阅。
added from 5.1.9

@remarks 代理
*/
- (void)didSwitchRoleWithUser:(RCRTCRemoteUser *)user roleType:(RCRTCLiveRoleType)roleType;

```

参数	类型	说明
user	RCRTCRemoteUser	切换身份的用户
roleType	RCRTCLiveRoleType	切换成功的目标角色类型

跨房间连麦 流程说明

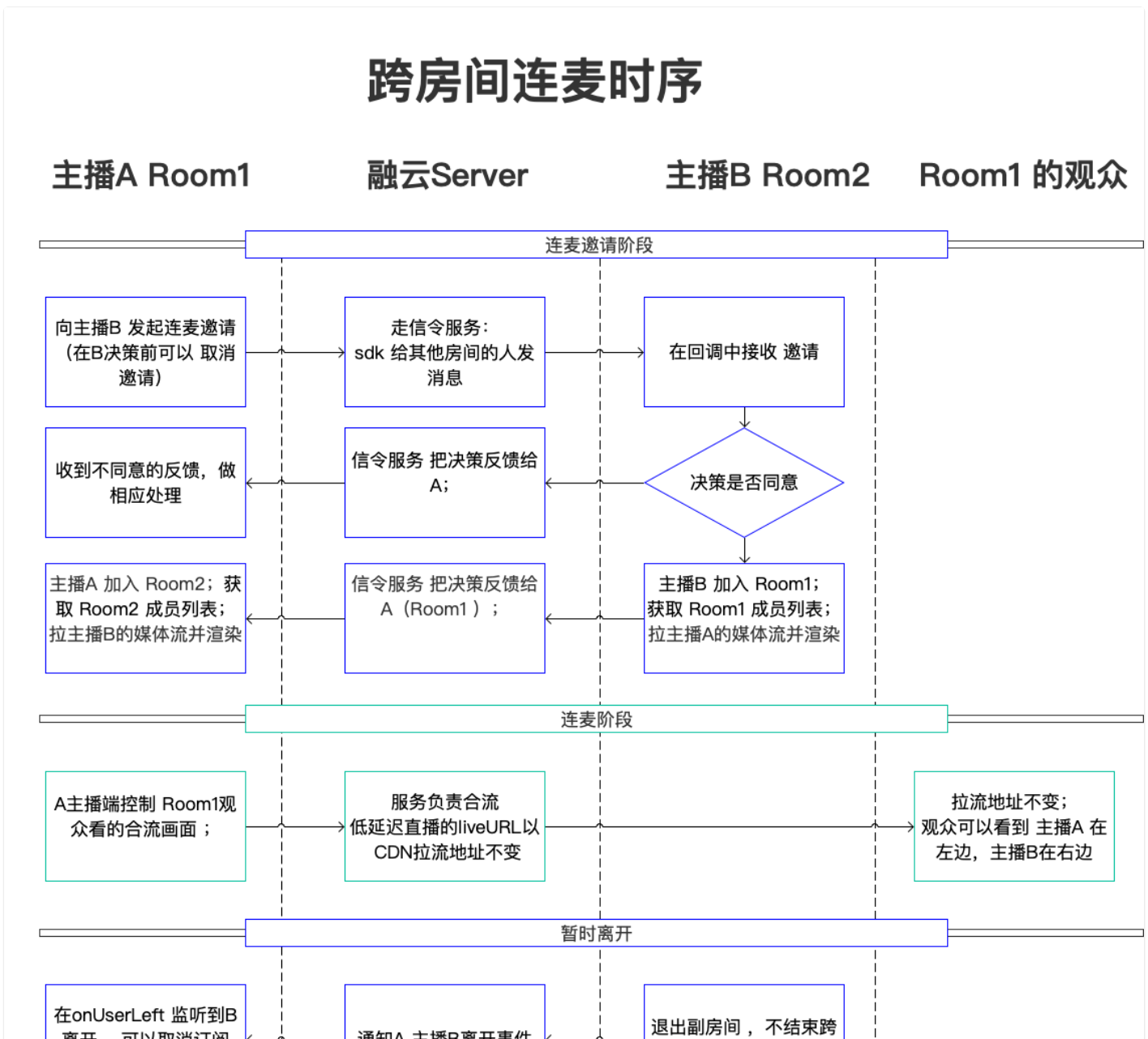
更新时间:2024-08-30

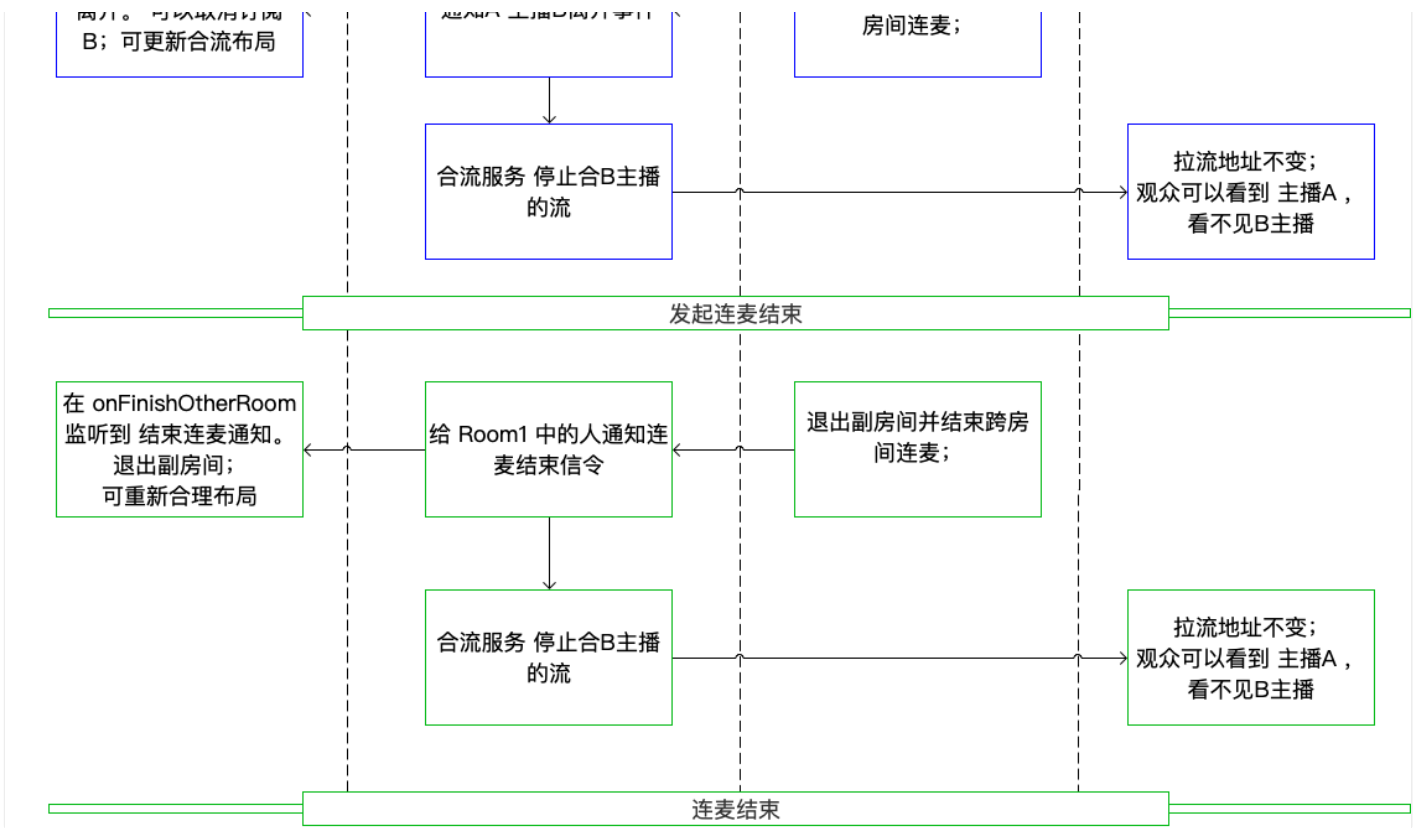
跨房间连麦功能中区分主房间和副房间。跨房间连麦前需要双方都已经加入自己的主房间，未加入主房间前无法进行连麦的邀请和被邀请。

主副房间定义如下：

- 主房间：本端在最开始加入的房间，即：`RCRTCRoom.h`
- 副房间：在接受邀请后加入的对方房间，即：`RCRTCOtherRoom.h`

两种房间类都有对应的代理 `RCRTCRoomEventDelegate` 和 `RCRTCOtherRoomEventDelegate` 用来区分不同类型房间的回调。





发起邀请

向指定用户发送跨房间连麦请求：

```

- (void)requestJoinOtherRoom:(NSString *)inviteeRoomId
  userId:(NSString *)inviteeUserId
  autoMix:(BOOL)autoMix
  extra:(NSString *)extra
  completion:(RCRTCOperationCallback)completion;

```

参数说明：

参数	类型	说明
inviteeRoomId	NSString	被邀请人所在房间号
inviteeUserId	NSString	被邀请人userId
autoMix	BOOL	是否将邀请人音视频资源发送到被邀请人房间中合流
extra	NSString	附加信息，可随消息发送给被邀请人
completion	RCRTCOperationCallback	动作的回调

示例代码：

```

[self.mainRtcRoom.localUser requestJoinOtherRoom:otherRoomId
  userId:otherRoomUserId
  autoMix:YES
  extra:@""
  completion:^(BOOL isSuccess, RCRTCCode code) {
}];

```

取消邀请

向指定用户发送跨房间连麦请求：

```
- (void)cancelRequestJoinOtherRoom:(NSString *)inviteeRoomId  
userId:(NSString *)inviteeUserId  
extra:(NSString *)extra  
completion:(RCRTCOperationCallback)completion;
```

参数说明：

参数	类型	说明
inviteeRoomId	NSString	被邀请人所在房间号
inviteeUserId	NSString	被邀请人userId
extra	NSString	附加信息，可随消息发送给被邀请人
completion	RCRTCOperationCallback	动作的回调

示例代码：

```
[self.mainRtcRoom.localUser cancelRequestJoinOtherRoom:otherRoomId  
userId:otherRoomUserId  
extra:@""  
completion:^(BOOL isSuccess, RCRTCCode code) {  
  
}];
```

应答邀请

被邀请方响应邀请方发出的跨房间连麦请求。

```
- (void)responseJoinOtherRoom:(NSString *)inviterRoomId  
userId:(NSString *)inviterUserId  
agree:(BOOL)agree  
autoMix:(BOOL)autoMix  
extra:(NSString *)extra  
completion:(RCRTCOperationCallback)completion;
```

参数说明：

参数	类型	说明
inviterRoomId	NSString	邀请人所在房间号
inviterUserId	NSString	邀请人userId
agree	BOOL	是否同意加入副房间
autoMix	BOOL	是否将被邀请人音视频资源发送到邀请人房间中合流
extra	NSString	附加信息，可随消息发送给被邀请人（选填）

参数	类型	说明
completion	RCRTCOperationCallback	动作的回调

示例代码

```
[self.mainRtcRoom.localUser responseJoinOtherRoom:otherRoomId
userId:otherRoomuserId
agree:YES
autoMix:YES
extra:@""]
completion:^(BOOL isSuccess, RCRTCCode code) {}];
```

加入副房间

邀请方与被邀请方连在麦邀请同意后加入副房间。

```
- (void)joinOtherRoom:(NSString *)roomId
completion:(void (^)(RCRTCOtherRoom * _Nullable room, RCRTCCode code))completion;
```

参数说明:

参数	类型	说明
roomId	NSString	副房间 Id，支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式 最长 64 个字符
completion	void (^)(RCRTCOtherRoom * _Nullable room, RCRTCCode code)	加入副房间的回调

示例代码:

```
[[RCRTCEngine sharedInstance] joinOtherRoom:roomId
completion:^(RCRTCOtherRoom * _Nullable room, RCRTCCode code) {
room.delegate = self;
}];
```

订阅资源

订阅多路远端指定音视频流。

```
- (void)subscribeStream:(NSArray <RCRTCInputStream *> *)avStreams
tinyStreams:(NSArray <RCRTCInputStream *> *)tinyStreams
completion:(nonnull RCRTCOperationCallback)completion;
```

参数说明:

参数	类型	说明
avStreams	NSArray <RCRTCInputStream *>	需要订阅的音频流和视频大流数组

参数	类型	说明
tinyStreams	NSArray <RCRTCInputStream *>	需要订阅的视频小流数组
completion	RCRTCOperationCallback	订阅完成的回调

示例代码:

```
[self.mainRTCRoom.localUser subscribeStream:streams
tinyStreams:@[]
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

取消订阅

取消订阅多路远端指定音视频流，远端用户离开与自己离开主、副房间时 RongRTCLib 内部会取消订阅，不需要上层处理。

```
- (void)unsubscribeStreams:(nonnull NSArray <RCRTCInputStream *> *)streams
completion:(nonnull RCRTCOperationCallback)completion;
```

参数说明:

参数	类型	说明
streams	NSArray <RCRTCInputStream *>	音视频流数组
completion	RCRTCOperationCallbac	取消订阅完成的回调

示例代码:

```
[self.mainRTCRoom.localUser unsubscribeStreams:streams
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

离开副房间

```
- (void)leaveOtherRoom:(NSString *)roomId
notifyFinished:(BOOL)isNotify
completion:(nonnull RCRTCOperationCallback)completion;
```

参数说明:

参数	类型	说明
roomId	NSString	副房间 Id
isNotify	BOOL	是否通知所有连麦用户结束跨房间连麦
completion	RCRTCOperationCallbac	离开副房间完成的回调

示例代码:

```

[[RCRTCEngine sharedInstance] leaveOtherRoom:otherRoomId
notifyFinished:YES
completion:^(BOOL isSuccess, RCRTCCode code) {
}
}];

```

场景说明

邀请连麦流程

场景一：

邀请方发起邀请后，邀请方又取消邀请。

此场景中被邀请方未做任何操作，仅被动收到两次代理回调。

邀请方	被邀请方
步骤1：调用 RCRTCLocalUser 类中的 requestJoinOtherRoom: userId: autoMix: extra: completion: 方法发起跨房间连麦邀请	步骤2：在 RCRTCRoomEventDelegate 中收到 didRequestJoinOtherRoom: inviterUserId: extra: 邀请回调
步骤3：调用 RCRTCLocalUser 类中的 cancelRequestJoinOtherRoom: userId: extra: completion: 方法结束之前发起的邀请	步骤4：在 RCRTCRoomEventDelegate 中收到 didCancelRequestOtherRoom: inviterUserId: extra: 取消受邀的回调

场景二：

邀请方发起邀请后，被邀请方应答同意。

此场景中被邀请方应答同意，只有被邀请方不会收到应答的回调，包含邀请人在内两个房间的所有非观众用户都会收到代理回调。

邀请方	被邀请方
步骤1：调用 RCRTCLocalUser 类中的 requestJoinOtherRoom: userId: autoMix: extra: completion: 方法发起跨房间连麦邀请	步骤2：在 RCRTCRoomEventDelegate 中收到 didRequestJoinOtherRoom: inviterUserId: extra: 邀请回调
步骤4：包含邀请人在内两个房间的所有非观众用户，不包含被邀请方，会在 RCRTCRoomEventDelegate 中收到 didResponseJoinOtherRoom: inviterUserId: inviteeRoomId: inviteeUserId: extra: 应答回调	步骤3：调用 RCRTCLocalUser 类中的 responseJoinOtherRoom: userId: agree: autoMix: extra: completion: 应答邀请，其中 agree 参数的值为 YES

场景三：

邀请方发起邀请后，被邀请方应答拒绝。

此场景中被邀请方应答拒绝，只有邀请方会收到应答拒绝的回调，其他所有用户都不会收到代理回调。

邀请方	被邀请方
步骤1：调用 RCRTCLocalUser 类中的 requestJoinOtherRoom: userId: autoMix: extra: completion: 方法发起跨房间连麦邀请	步骤2：在 RCRTCRoomEventDelegate 中收到 didRequestJoinOtherRoom: inviterUserId: extra: 邀请回调

邀请方	被邀请方
步骤4：只有邀请方会收到应答拒绝的回调，其他所有用户都不会收到 RCRTCRoomEventDelegate 中的 didResponseJoinOtherRoom: inviterUserId: inviteeRoomId: inviteeUserId: extra: 应答回调	步骤3：调用 RCRTCLocalUser 类中的 responseJoinOtherRoom: userId: agree: autoMix: extra: completion: 应答邀请，其中 agree 参数的值为 NO

加入副房间流程

场景四：

在 场景二 的前提下，

邀请方在收到应答同意的回调后，加入被邀请方的房间，即：邀请方加入的副房间。

被邀请方在调用应答同意后，加入邀请方的房间，即：被邀请方加入的副房间。

邀请方	被邀请方
步骤2：在收到被邀请方同意代理回调后，调用 RCRTCEngine 中 joinOtherRoom: completion: 方法加入副房间	步骤1：在应答邀请同意后，调用 RCRTCEngine 中 joinOtherRoom: completion: 方法加入副房间

场景五：

在加入主房间前，如果主房间的某用户已经与其他房间另一用户成功建立跨房间连麦。

可以通过加入主房间成功后得到的 RCRTCRoom 对象中检查 room.otherRoomIdArray 中是否存在房间中的用户已经加入的副房间。如果存在，可选择是否加入其他副房间。

```

/*!
主房间中主播已经加入的副roomId列表
*/
@property (nonatomic, strong, readonly) NSArray *otherRoomIdArray;

```

邀请方
步骤1：调用 RCRTCEngine 中 joinRoom: completion: 方法加入主房间
步骤2：检查发现 room.otherRoomIdArray 中存在其他副房间
步骤3：调用 RCRTCEngine 中 joinOtherRoom: completion: 方法加入副房间

订阅流程

在 场景四 或 场景五 的前提下：

场景六：

订阅副房间用户已经发布的音视频资源方式一：加入副房间后，通过遍历取得副房间中所有远端用户中的 inputStream 列表，再调用 RCRTCLocalUser 中 subscribeStream: tinyStreams: completion: 方法订阅副房间中远端用户已经发布的音视频资源。

```

[[RCRTCEngine sharedInstance] joinOtherRoom:roomId
completion:^(RCRTCOtherRoom * _Nullable pkRoom, RCRTCCode code) {
pkRoom.delegate = self;

if (code == RCRTCCodeSuccess) {
NSMutableArray *streamArray = [NSMutableArray array];
for (RCRTCRemoteUser *user in pkRoom.remoteUsers) {
for (RCRTCInputStream *stream in user.remoteStreams) {
if (stream.mediaType == RTCMediaTypeVideo) {
RCRTCVideoInputStream *videoInputStream = (RCRTCVideoInputStream *)stream;
RCRTCRemoteVideoView *remoteVideoView = [[RCRTCRemoteVideoView alloc] initWithFrame:CGRectMake(0, 0, 90,
120)];
[需要显示到的View addSubview:remoteVideoView];
[videoInputStream setVideoView:remoteVideoView];
}
[streamArray addObject:stream];
}
}

if (streamArray.count > 0) {
[self.mainRTCRoom.localUser subscribeStream:streamArray
tinyStreams:@[]
completion:^(BOOL isSuccess, RCRTCCode desc) {
if (completion) {
completion(isSuccess, desc);
}
}]];
}
}
}
}];
}];

```

订阅副房间用户刚发布的音视频资源方式二：加入副房间后，当有副房间中远端用户发布资源时，会回调 RCRTCOtherRoomEventDelegate 中 room: didPublishStreams: 方法，调用 RCRTCLocalUser 中 subscribeStream: tinyStreams: completion: 方法，使用代理方法中的 streams 参数订阅副房间中远端用户刚发布的音视频资源。

```

- (void)room:(RCRTCBaseRoom *)room didPublishStreams:(NSArray <RCRTCInputStream *> *)streams {
for (RCRTCInputStream *stream in streams) {
if (stream.mediaType == RTCMediaTypeVideo) {
RCRTCVideoInputStream *videoInputStream = (RCRTCVideoInputStream *)stream;
RCRTCRemoteVideoView *remoteVideoView = [[RCRTCRemoteVideoView alloc] initWithFrame:CGRectMake(0, 0, 90,
120)];
[videoInputStream setVideoView:remoteVideoView];
}
}

[self.mainRTCRoom.localUser subscribeStream:streams
tinyStreams:@[]
completion:^(BOOL isSuccess, RCRTCCode code) {
}]];
}
}

```

离开副房间流程

场景七：

已经加入副房间的用户，需要退出副房间时，可以调用 RCRTCEngine 中的 leaveOtherRoom: notifyFinished: completion: 方法。

此方法内部会取消订阅已经订阅的副房间所有用户音视频资源，上层无需自行取消订阅。

此方法中的 `isNotify` 参数为 `NO` 时，仅代表调用方离开副房间，其他用户会收到 `RCRTCOtherRoomEventDelegate` 中：

```
- (void)room:(RCRTCBaseRoom *)room didLeaveUser:(RCRTCRemoteUser *)user;
```

当 `isNotify` 参数为 `YES` 时，其他用户会首先收到 `RCRTCOtherRoomEventDelegate` 中：

```
- (void)room:(RCRTCBaseRoom *)room didLeaveUser:(RCRTCRemoteUser *)user;
```

回调方法，然后会收到 `RCRTCRoomEventDelegate` 中：

```
- (void)didFinishOtherRoom:(NSString *)roomId  
userId:(NSString *)userId;
```

回调方法，用来通知跨房间连麦中的所有用户，可以退出副房间结束连麦。

被服务端踢出房间流程

通过调用服务端 API 可以将指定房间的指定用户踢出该房间。

在踢出某个用户之前，建议先通过服务下发消息让此客户端执行结束跨房间连麦，然后再从主房间踢出违禁用户。

被踢出的房间是该用户加入的主房间时，`RongRTCLib` 内部会调用 `leaveRoom:` 方法先退出已经加入的副房间，再退出主房间并销毁资源。UI 层无需再次调用离开主、副房间的方法，只处理上层逻辑即可。

被踢出的房间是该用户加入的副房间时，`RongRTCLib` 内部会调用 `leaveOtherRoom: notifyFinished: completion:` 方法仅退出指定的副房间，并取消订阅已经订阅的该副房间中用户的音视频资源，UI 层无需再次调用离开副房间的方法，只处理上层逻辑即可。

```
/*!  
被服务端踢下线通知  
  
@param room 离开的房间  
@discussion  
如果用户在房间内，此时收到服务器封禁的通知，SDK 会关闭音视频连接，释放资源，  
将用户踢出房间，回调通知用户  
  
@remarks 代理  
*/  
- (void)didKickedByServer:(RCRTCBaseRoom *)room;
```

CDN 业务概述

更新时间:2024-08-30

当您使用 RTCLib SDK 做直播应用时，可以选用融云 CDN 服务或第三方 CDN 服务分发直播媒体流。

融云 CDN 服务

提示

- 融云 CDN 是付费增值服务，需要开通后才能使用。详见[融云 CDN 服务配置与音视频直播增值服务计费说明](#)。
- 客户端使用融云 CDN 服务依赖融云 CDN 插件（RongRTCPlayer）。

[融云 CDN](#) 服务能确保在不同区域、不同场景下加速直播内容的分发，提高资源访问速度。

直播主播端用 RTC 协议发布音视频资源，资源由融云服务端接收，转协议到 RTMP，并推给 CDN。视频编码协议为 h.264。

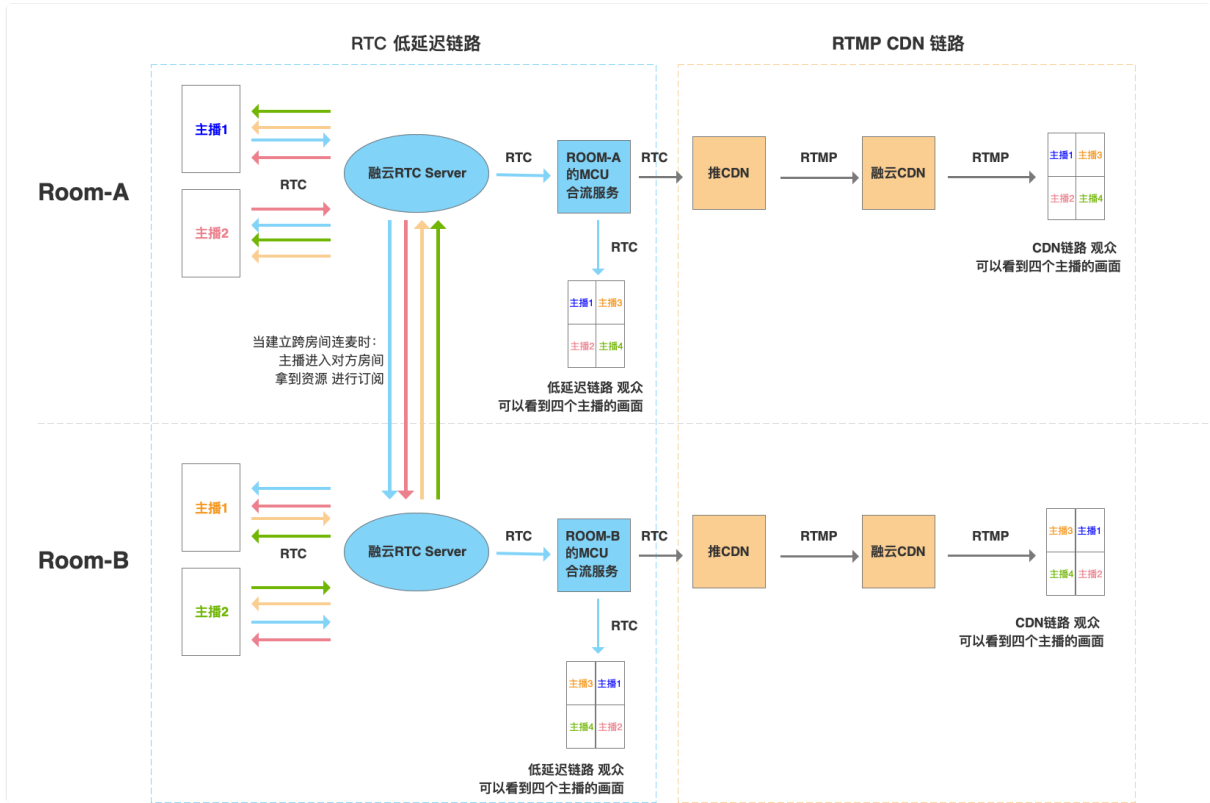
直播观众端可以感知到主播推流状态、服务端合流状态的变化。您无需在 App 的服务端（AppServer）做过多的任务管理。如果您选择由融云 RTC 服务转推融云 CDN 服务。

观众端可以通过 SDK 选择订阅 CDN 链路或者 RTC 低延迟链路的直播流。观众默认按 CDN 服务输出的视频格式拉流，也可以调接口设置指定的拉流分辨率、码率、帧率。非原始分辨率、码率会触发 CDN 转码服务，对计费有影响。

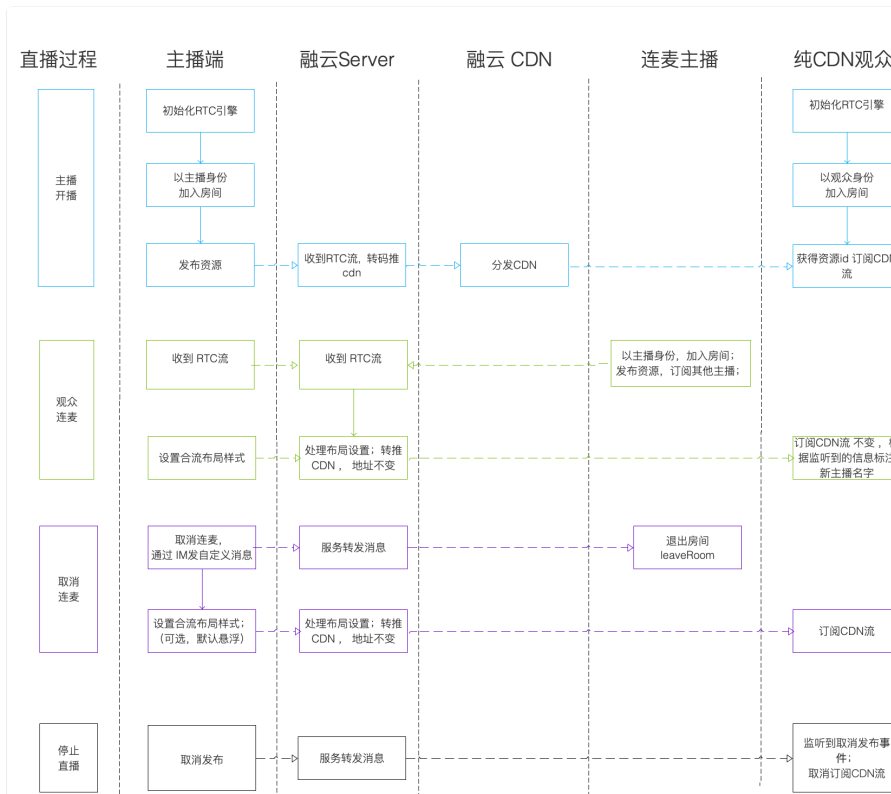
功能	说明	使用场景	开发文档
转推到融云 CDN 服务（开播自动转推）	要求在控制台开启开播自动推流。RTC 直播流先经过融云 RTC 服务器，再实时转推融云 CDN 服务。您可以使用融云 RTC 服务控制直播合流的布局、分辨率等。 <ul style="list-style-type: none">观众端可使用客户端 SDK 提供监听、订阅房间内 CDN 资源的方法。您也可以直接从融云 CDN 服务获取拉流地址，详见获取融云 CDN 拉流地址。	融云实现了与 CDN 资源相关的逻辑，如发布、监听、订阅、连麦等，使用简单，开发成本低。	融云 CDN 插件
转推到融云 CDN 服务（开播手动转推）	要求在控制台开启开播手动推流。与以上方案区别在于主播需要在客户端主动开启向融云 CDN 推流的开关。 <ul style="list-style-type: none">观众端可使用客户端 SDK 提供监听、订阅房间内 CDN 资源的方法。您也可以直接从融云 CDN 服务获取拉流地址，详见获取融云 CDN 拉流地址。	融云实现了与 CDN 资源相关的逻辑，如发布、监听、订阅、连麦等，使用简单，开发成本低。您可以自定决定何时向 CDN 推流。	融云 CDN 插件

功能	说明	使用场景	开发文档
直推融云 CDN	要求在控制台开启自行生成推拉流地址。RTC 直播流先经过融云 RTC 服务器，但不向融云 CDN 服务推流。App 须按照融云 CDN 服务提供的推拉流地址规则自行拼接推拉流地址，调用客户端或服务端的旁路推流接口，直接向融云 CDN 推流地址推流。观众端无法使用融云客户端 SDK 接口使用 CDN 资源。	不依赖融云 CDN 处理逻辑，观众端可直接通过 CDN 播放器拉流观看（推荐使用融云 CDN 插件提供的 CDN 播放器组件）。首屏打开时间较以上方案低，适用于对首屏打开时间敏感的直播业务。	客户端旁路推流、服务端旁路推流、CDN 播放器

融云 CDN 服务架构



融云 CDN 服务主要业务逻辑：



第三方 CDN 服务

融云客户端与服务端均提供旁路推流接口，可以将融云 RTC 服务中的直播流转推至第三方 CDN 服务。

功能	说明	使用场景	开发文档
转推到第三方 CDN 服务	RTC 直播流先经过融云 RTC 服务器，再实时转推融云 CDN 服务。您可以使用融云 RTC 服务控制直播合流的布局、分辨率等。观众端无法使用客户端 SDK 提供监听、订阅房间内 CDN 资源的方法。	开发者与第三方 CDN 有业务合作，想要使用原有的第三方 CDN 流媒体网络的分发服务的同时又想使用 SDK 进行实时连麦互动。观众端可直接通过 CDN 播放器拉流观看（推荐使用融云 CDN 插件提供的 CDN 播放器组件）	CDN 播放器

融云 CDN 服务配置

更新时间:2024-08-30

欢迎前往融云官网了解[直播 CDN 产品](#)。本文介绍如何开始配置融云 CDN 服务。

开通服务

融云 CDN 是付费增值服务，且需要开通后才能使用。您可以访问控制台的[融云 CDN](#)页面开通服务。



开通融云 CDN 服务后，可看到[域名配置](#)、[证书管理](#)、[防盗链配置](#)。

选择推拉流模式

融云 CDN 支持三种不同的推拉流地址模式：

- **开播自动推流**：在后台配置后，所有直播房间的直播流都会自动推流到 CDN，观众可在客户端加入房间时获取 CDN 信息，或者使用客户端 SDK 提供的监听房间内 CDN 资源的方法。如果您希望直接从融云服务端获取拉流地址，可从 App 服务端调用融云服务端 API，详见[获取融云 CDN 拉流地址](#)。
- **开播手动推流**：在直播主播开始推流后，您需要根据产品设计决定何时调接口让融云服务开始或停止推流到 CDN。观众可在客户端加入房间时获取 CDN 信息，或者使用客户端 SDK 提供的监听房间内 CDN 资源的方法。如果您希望直接从融云服务端获取拉流地址，可从 App 服务端调用融云服务端 API，详见[获取融云 CDN 拉流地址](#)。
- **自行生成推拉流地址**：融云服务端不负责生成推拉流地址，您需要在您的应用服务器自行生成 CDN 推拉流地址。



自行拼接推拉流地址

在自行生成推拉流地址模式下，可自行拼接生成 CDN 推拉流地址。需要拼接的字段包括推/拉流域名、自定义的 {AppName}

和自定义的 {StreamName}。您可以通过在 CDN 拉流地址中添加分辨率、码率参数 {with}、{height}、{fps}，拉取 CDN 转码流。

CDN 推流地址拼接规则

融云 CDN 仅支持 RTMP 协议的推流：

RTMP：rtmp://推流域名/{AppName}/{StreamName}

拼接推流地址后，可调用客户端 SDK 或服务端 API 的旁路推流接口，传入 CDN 推流地址进行推流。

CDN 拉流地址拼接规则

融云 CDN 支持使用 RTMP、FLV、HLS 协议进行拉流。

如需使用 CDN 流的原始分辨率、帧率：

- **RTMP**：rtmp://拉流域名/{AppName}/{StreamName}
- **FLV**：http(s)://拉流域名/{AppName}/{StreamName}.flv
- **HLS**：http(s)://拉流域名/{AppName}/{StreamName}.m3u8

如需拉取不同分辨率的 CDN 直播流（拉取非原始分辨率、码率的流会触发 CDN 转码服务，产生转码费用）：

- **RTMP**：rtmp://拉流域名/{AppName}/{StreamName}_{with}_{height}_{fps}
- **FLV**：http(s)://拉流域名/{AppName}/{StreamName}_{with}_{height}_{fps}.flv
- **HLS**：http(s)://拉流域名/{AppName}/{StreamName}_{with}_{height}_{fps}.m3u8

{with}、{height} 参数为要拉取的 CDN 转码流的分辨率宽高，参考下方枚举值。{fps} 为帧率参数，支持 10/15/24/30。

宽高限制列表：

```
[  
"176*144", "180*180", "256*144", "240*180", "320*180", "240*240", "320*240",  
"360*360", "480*360", "640*360", "480*480", "640*480", "720*480", "848*480",  
"960*720", "1280*720", "1920*1080", "144*176", "144*256", "180*240", "180*320",  
"240*320", "360*480", "360*640", "480*640", "480*720", "480*848", "720*960",  
"720*1280", "1080*1920"  
]
```

例如，拉取 flv 协议的 CDN 流，并指定宽x高为 720*920，帧率为 15，则拼接后的地址如下：

http(s)://yourdomain/appkey/roomid_720_960_15.flv

推荐使用融云 CDN 播放器进行播放。

③ 提示

- 如在控制台启用了防盗链，则您拼接的推拉流地址中还必须携带防盗链信息。防盗链地址的具体拼接规则请参见下文「防盗链配置」。
- 2022年6月前开通融云 CDN 服务的客户，如获取到的 HLS 拉流地址无法正常播放，请提交工单咨询。

域名配置

在域名配置标签下配置并保存了推流域名和拉流域名后，会看到完整的配置界面。

The screenshot shows the 'CDN 服务配置' (CDN Service Configuration) page. At the top, there's a toggle for '融云 CDN 服务状态: 已开通' (Rongyun CDN Service Status: Enabled) with a '关闭服务' (Close Service) button. Below that is the '推拉流模式' (Push/Pull Mode) set to '自行生成推拉流地址' (Generate push/pull stream addresses myself) with a '保存修改' (Save Changes) button. The '域名配置' (Domain Configuration) tab is active, with sub-tabs for '证书管理' (Certificate Management) and '防盗链' (Anti-leech). Under '域名配置', there are two rows for domain settings. The first row is for the push domain (*推流域名:), showing it is '已生效' (Effective) and has a 'cname 域名' (CNAME domain) with a '复制' (Copy) button. The second row is for the pull domain (*拉流域名:), also '已生效' (Effective) with a 'cname 域名' and '复制' (Copy) button. A red note states: '注意: CNAME 域名不能直接访问, 您需要在域名服务提供商处完成 CNAME 配置。配置生效后, 即可使用 CDN 链路进行直播拉流' (Note: CNAME domains cannot be accessed directly; you need to complete CNAME configuration at the domain service provider. After configuration is effective, you can use the CDN link for live streaming). Below the note, it says: '在 20:00 至次日 2:00 进行的配置不会立即生效, 会在 2 点后依次配置生效。在此时间段外进行配置会在 30 分钟后生效' (Configurations made between 20:00 and 02:00 the next day will not take effect immediately, but will take effect sequentially after 2:00. Configurations made outside this time period will take effect 30 minutes later). There is a '禁用' (Disable) button. The 'HTTPS 设置' (HTTPS Settings) section has a tip: '提示: FLV/HLS 协议的直播流默认使用 HTTP, 配置 SSL 证书后可以使用 HTTPS。如果您没有在融云平台上传过证书请先上传证书' (Tip: Live streams using the FLV/HLS protocol default to HTTP, but can use HTTPS after configuring an SSL certificate. If you haven't uploaded a certificate to the Rongyun platform, please upload one first). There is a '新增证书' (Add Certificate) button. At the bottom, there are two dropdown menus for '推流:' (Push) and '拉流:' (Pull), both set to '请选择' (Please select), with '绑定证书' (Bind Certificate) buttons next to them.

配置推拉流域名

推拉流域名都需要填写二级域名，请确保此域名没有在别的 App Key 下配置过。

提示

- 域名是成对的，推流和拉流有绑定关系，新增和修改时需要一并修改。
- 您需要确保一级域名已经备案过。如果因域名没有备案或涉及非法业务等原因导致推流或者拉流域名不可用，由此产生的任何后果由您自身承担。

推流与拉流域名配置生效后，系统会自动给该域名分配一个 CNAME。使用您的推流或拉流域名进行直播之前，需要您公司开发或服务运维人员在域名解析配置中添加对应的 CNAME 记录，让您的域名指向 CNAME。具体如何配置可以咨询域名供应商。

HTTPS 设置

HTTPS 设置是可选项。FLV/HLS 协议的直播流默认使用 HTTP，配置 SSL 证书后可以使用 HTTPS。

如果您没有在融云平台上传过证书，请点击新增证书，将证书上传。下图展示了提交证书的界面。

证书管理 / 新增 返回

提示:

- 不支持新增 MD5 加密签名算法证书。由于此类证书安全性较低,且部分主流浏览器已限制使用。
- 超过 44 KB 的 crt 文件在 IE11 浏览器上使用时存在高风险,请确保您上传的每个 crt 文件大小都小于 44 KB。

*证书名称:

请输入证书内容

*证书内容:

请输入私钥内容

*私钥内容:

请输入备注

备注:

防盗链

防盗链配置是可选项。配置后会提升推拉流域名的安全。

开启后一定要配置推流和拉流地址有效期时间和加密 URL 的密文 KEY。

域名配置 证书管理 **防盗链**

防盗链配置

推流: KEY: 时长: 秒

拉流: KEY: 时长: 秒

在 20:00 至次日 2:00 进行的配置不会立即生效,会在 2 点后依次配置生效,在此时间段外进行配置会在 30 分钟后生效

防盗链开启后,如果观众端遇到弱网(融云 SDK 内部帮您在网络恢复后重新订阅成功)会比没有防盗链多出一些恢复订阅时间。

自行生成防盗链推拉流地址

如果您选择的推拉流模式为自行生成推拉流地址,在您的服务端生成推拉流地址时在地址中加入防盗链相关信息。

相比普通推拉流地址,生成防盗链推拉流地址还需要用到以下防盗链参数:

防盗链参数	描述	补充说明
wsTime	生成推拉流地址时的 UNIX 时间。防盗链地址中直接携带该参数。	格式为 16 进制 UNIX 时间。防盗链地址中直接携带该参数。
KEY	MD5 计算方式的密钥,在控制台配置,仅用于计算 wsSecret。	可以自定义。
wsSecret	播放 URL 中的加密参数。防盗链地址中直接携带该参数。	值是通过将 KEY,URI,wsTime 依次拼接的字符串进行 MD5 加密算法得出,即 $wsSecret = MD5(wsTime + URI + KEY)$ 。

防盗链的推拉流地址生成示例:

假设原始 url (已包含推/拉流域名 + AppName + StreamName) 为: <http://cdn.rongcloud.cn/myappname/stream.flv>

1. 获取在控制台防盗链配置填入的 KEY,假设 KEY 为: rongcloud。

- 获取 wsTime，即生成推拉流地址时的 UNIX 时间，假设为 1601026312，转换成 16 进制 5f6db908。在计算 wsSecret 时需要用到。防盗链 URL 中也会直接携带该参数。
- 计算 wsSecret。防盗链 URL 中会直接携带该参数。
 - 获取计算 wsSecret 需要用到的 URI 参数值。需要用到您自定义的 AppName 与 StreamName。拼接规则为 `/{AppName}/{StreamName}`。从本例的原始 URL 可得出 URI 为 `/app/stream.flv`。
 - 依次拼接 wsTime + URI + KEY，获取签名字符串。在本例中该拼接字符串为：5f6db908/app/stream.flvrongcloud
 - 对签名字符串计算 md5hash，得到 wsSecret。即，`wsSecret = md5sum("5f6db908/app/stream.flvrongcloud") = 79aead3bd7b5db4adefb93a010298b5`
- 使用上述步骤中得到的 wsSecret 与 wsTime 参数，拼接成防盗链 URL：

<http://cdn.rongcloud.cn/app/stream.flv?wsSecret=79aead3bd7b5db4adefb93a010298b5&wsTime=5f6db908> 

当用户发起带时间戳防盗链的 url 请求后，CDN 服务器会对 url 内容进行校验，判断时间有效性及 MD5 值，两个值其中一个不符合要求，返回 403。

生成防盗链代码示例（golang）

```
package main

import (
    "crypto/md5"
    "encoding/hex"
    "fmt"
    "strconv"
    "strings"
    "time"
)

func main() {
    secret, t := GenWsSecret("/live/abc", "key")
    fmt.Println(secret)
    fmt.Println(t)
}

func GenWsSecret(uri, key string) (string, string) {
    uri = fmt.Sprintf("%s", strings.TrimLeft(uri, "/"))

    t := strconv.FormatInt(time.Now().Unix(), 16)

    ori := fmt.Sprintf("%s%s%s", t, uri, key)

    h := md5.New()
    h.Write([]byte(ori))

    return hex.EncodeToString(h.Sum(nil)), t
}
```

提示

- 在 20:00 至次日 2:00 进行的配置不会立即生效，会在 2 点后依次配置生效。在此时间段外进行配置会在 30 分钟后生效。

2. 新配置生效后，原有的配置即刻作废。建议开发者避开业务高峰时段，并给还在使用老配置的房间用户发送提醒通知，重新订阅新地址。

融云 CDN 插件

更新时间:2024-08-30

融云 CDN 插件 (RongRTCPlayer) 直接在 RTC 房间内提供了 [融云 CDN](#) 服务的订阅、播放等处理逻辑，降低了 App 业务侧的接入成本。

提示

融云 CDN 插件 (RongRTCPlayer) 从 5.3.0 版本开始提供 CDN 播放器组件 RCRTCMediaPlayer，可直接通过 CDN 地址播放 CDN 流、获取 SEI 数据、或直接获取音视频帧数据用于自行处理。详见 [CDN 播放器](#)。

本文主要描述如何通过融云 CDN 插件 (RongRTCPlayer) 操作和使用 RTC 房间内的 CDN 资源。

集成融云 CDN 插件

融云 CDN 插件导入方法请参见 [导入 SDK](#)。

集成融云 CDN 插件后请注意导入头文件：

```
#import <RongRTCPlayer/RongRTCPlayer.h>
```

推流到融云 CDN 服务

您可以通过 [融云 CDN](#) 服务的配置，由融云服务端控制，将 RTC 房间的直播流转推到融云 CDN。融云 CDN 服务获取直播流后，RTC 房间内将会新增 CDN 流资源，观众可以直接订阅。App 需要根据业务要求指定推流时机：

- 开播自动推流：所有直播房间的直播流都会自动推流到融云 CDN。观众可随时订阅。
- 开播手动推流：App 需要在直播主播开始推流后，通过客户端接口 `enableInnerCDN` 控制是否推流到融云 CDN 服务。

提示

如果推拉流模式为自行生成推拉流地址，融云服务端不负责将 RTC 房间的直播流转推到融云 CDN。您按照融云 CDN 服务的地址规则生成地址后，可通过客户端或服务端的旁路推流 API 将直播流传入融云 CDN 服务。

手动推流到融云 CDN

如果控制台推拉流模式为开播手动推流，必须在 App 客户端自行决定是否推流到融云 CDN 服务。在整个直播过程中都可以操作开关。

主播在 RTC 房间中发布资源后（详见[发布资源]），通过回调中返回的 [RCRTCLiveInfo](#) 对象操作是否启用向融云 CDN 推流。

以发布默认音视频流为例：

```
[[RCRTCEngine sharedInstance].room.localUser publishDefaultLiveStreams:^(BOOL isSuccess, RCRTCCode desc, RCRTCLiveInfo * _Nullable liveInfo) {  
    // 可以通过拿到的liveInfo对象进行推流相关配置  
}];
```

[RCRTCLiveInfo](#) 对象提供以下方法设置是否向融云 CDN 服务推流：

```
– (void)enableInnerCDN:(BOOL)enable  
completion:(void (^)(BOOL isSuccess, RCRTCCode code))completion
```

参数	类型	说明
enable	BOOL	是否开启
completion	(BOOL isSuccess, RCRTCCode code)	完成回调

```
[self.liveInfo enableInnerCDN:enable  
completion:^(BOOL isSuccess, RCRTCCode code) {  
    if (isSuccess) {  
        //TODO:  
    }  
}];
```

开启或关闭内置 CDN 时，观众均会收到对应回调。

- 开启内置 CDN 时：房间中的观众会收到 [RCRTCRoomEventDelegate](#) 中 `didPublishCDNStream:` 回调。
- 关闭内置 CDN 时：房间中的观众会收到 [RCRTCRoomEventDelegate](#) 中 `didUnpublishCDNStream:` 回调。

获取房间内 CDN 资源

采用开播自动推流或开播手动推流方式将直播流推到融云 CDN 服务后，RTC 房间内的观众可获取房间内的 CDN 资源。

观众可通过以下方式获取房间内 CDN 资源：

- 加入房间时获取：观众加入房间时，如果房间中已经存在 [RCRTCCDNInputStream](#) 流，可使用加入房间成功后获取到的 [RCRTCRoom](#) 对象中 `getCDNStream` 方法获取。
- 通过房间事件监听获取：已在房间内的观众可通过 [RCRTCRoom](#) 注册房间事件监听 [RCRTCRoomEventDelegate](#) 获取本房间中 [RCRTCCDNInputStream](#) 流的发布和取消发布回调事件。

```

/*!
远端CDN流发布资源通知

@param stream 发布的CDN资源
@discussion
Added from 5.1.5

@remarks 代理
*/
- (void)didPublishCDNStream:(RCRTCInputStream *)stream;

/*!
远端CDN流取消发布资源通知

@param stream 取消发布的CDN资源
@discussion
Added from 5.1.5

@remarks 代理
*/
- (void)didUnpublishCDNStream:(RCRTCInputStream *)stream;

```

订阅房间内 CDN 资源

观众可以调用 [RCRTCLocalUser](#) 类的 `subscribeStream:tinyStreams:completion:` 方法订阅房间中 [RCRTCInputStream](#) 资源。默认订阅 [RCRTCInputStream](#) 的原始分辨率、帧率，即 [getHighestResolution](#) 和 [getHighestFPS](#)。

```

- (void)subscribeStream:(NSArray <RCRTCInputStream *> *)avStreams
tinyStreams:(NSArray <RCRTCInputStream *> *)tinyStreams
completion:(nonnull RCRTCOperationCallback)completion;

```

参数	类型	说明
avStreams	NSArray <RCRTCInputStream *>	房间内直播流。
tinyStreams	NSArray <RCRTCInputStream *>	需要携带小流的流数组
completion	RCRTCOperationCallback	完成的回调

在 `avStreams` 参数中传入房间内的 CDN 流资源。 `tinyStreams` 为订阅 RTC 直播小流时的参数，可不传。

```

if (self.room == nil) {
return;
}

RCRTCDDNInputStream *rtmpStream = [self.room getCDNStream];
if (!rtmpStream) {
return;
}

RCRTCRemoteVideoView *cdnVideoView = [[RCRTCRemoteVideoView alloc] initWithFrame:self.view.bounds];
[self.sView addSubview:cdnVideoView];

[rtmpStream setVideoView:cdnVideoView];

[self.room.localUser subscribeStream:@[rtmpStream]
tinyStreams:@[]
completion:^(BOOL isSuccess, RCRTCCode code) {
if (isSuccess) {
//TODO:
}
}
]];

```

取消订阅

观众可以调用 [RCRTCLocalUser](#) 类的 `unsubscribeStream:completion:` 方法取消订阅 `RCRTCDDNInputStream` 资源。

```

- (void)unsubscribeStream:(nonnull RCRTCInputStream *)stream
completion:(nonnull RCRTCOperationCallback)completion;

```

参数	类型	说明
stream	RCRTCInputStream	房间内的 CDN 流
completion	RCRTCOperationCallback	取消订阅的结果回调

```

if (!self.room) {
return;
}

RCRTCDDNInputStream *rtmpStream = [self.room getCDNStream];
if (!rtmpStream) {
return;
}

[self.room.localUser unsubscribeStream:rtmpStream
completion:^(BOOL isSuccess, RCRTCCode code) {
if (isSuccess) {
//TODO:
}
}
]];

```

调整订阅分辨率

观众可以在订阅房间内 CDN 资源前后调整分辨率和帧率。

提示

- 观众订阅房间内 默认订阅 `RCRTCVideoSizePreset` 的原始分辨率、帧率，即 `getHighestResolution` 和 `getHighestFPS`。
- 调整分辨率和帧率会触发 CDN 转码服务，转码对计费有影响。详见 [音视频直播增值服务计费说明](#)。

调用订阅方法前，订阅成功后，均可以通过 `RCRTCVideoSizePreset` 中 `setVideoConfig:fpsValue:completion:` 方法设置本次订阅资源的分辨率、帧率。

```
/*!
设置分辨率和帧率 订阅之前设置
@param videoSizePreset 分辨率
@param fps 帧率
*/
- (void)setVideoConfig:(RCRTCVideoSizePreset)videoSizePreset
      fpsValue:(RCRTCVideoFPS)fps
      completion:(RCRTCConfigCallback)completion;
```

参数	类型	说明
videoSizePreset	RCRTCVideoSizePreset	分辨率枚举
fps	RCRTCVideoFPS	帧率枚举
completion	<code>RCRTCConfigCallback</code>	结果回调

```
if (!self.room) {
return;
}

RCRTCVideoSizePreset *rtmpStream = [self.room getCDNStream];
if (!rtmpStream) {
return;
}

[rtmpStream setVideoConfig:RCRTCVideoSizePreset1920x1080
fpsValue:RCRTCVideoFPS30
completion:^(BOOL isSuccess, RCRTCCode code) {
@StrongObj(self);
if (isSuccess) {
//TODO
}
}];
```

观众端禁用 CDN 流渲染

观众成功订阅 RTC 房间内的 CDN 流后，可调用 CDN 流 (`RCRTCVideoSizePreset`) 的 `isMute` 方法禁用或启用 CDN 流渲染。

```
@property (nonatomic, assign, readonly) BOOL isMute;
```

参数	类型	说明
isMute	BOOL	YES: 停止资源渲染，NO: 恢复资源渲染

```
if (!self.room) {  
    return;  
}  
  
RCRTCCDNInputStream *rtmpStream = [self.room getCDNStream];  
if (!rtmpStream) {  
    return;  
}  
  
[rtmpStream setIsMute:YES];
```

监听房间内 CDN 流的 SEI

提示

SDK 从 5.3.0 版本开始支持监听和接收房间内 CDN 流的 SEI 数据。

观众在订阅 RCRTCCDNInputStream 前，实现 RCRTCCDNInputStreamDelegate 协议的 `cdnStream:handleSEIData:` 代理方法，并设置 RCRTCCDNInputStream 中 `delegate` 代理，通过该协议方法监听 CDN 流的 SEI 数据。

```
@protocol RCRTCCDNInputStreamDelegate <NSObject>  
/*!  
SEI 数据监听回调  
  
@param cdnStream 当前 cdn 流对象  
@param data SEI 数据  
@discussion  
订阅 cdn 流成功后，通过 receiver 回调 cdn 流中包含 SEI 信息  
回调的频次取决于 cdn 流的 SEI 帧数据间隔  
*/  
- (void)cdnStream:(RCRTCCDNInputStream *)cdnStream handleSEIData:(NSString *)data;  
  
@end
```

参数	类型	说明
cdnStream	RCRTCCDNInputStream *	当前订阅的 cdn 流对象
data	NSString *	SEI 数据

```
if (!self.room) {
return;
}

RCRTCCDNInputStream *rtmpStream = [self.room getCDNStream];
//设置cdn流代理
rtmpStream.delegate = self;

#pragma mark - RCRTCCDNInputStreamDelegate
- (void)cdnStream:(RCRTCCDNInputStream *)cdnStream handleSEIData:(NSString *)data {
NSLog(@"cdn stream receive SEI:%@",data);
}
```

客户端旁路推流

更新时间:2024-08-30

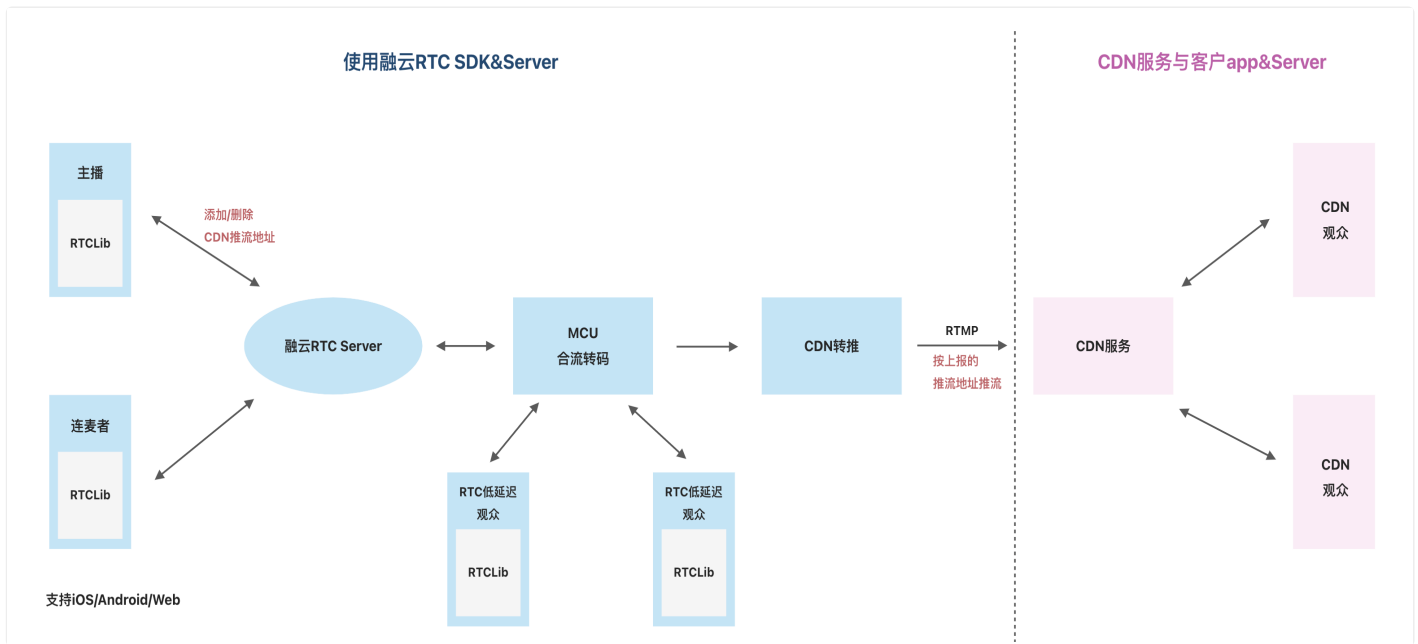
旁路推流：将融云实时音视频流转为 rtmp 流媒体协议，推流到 CDN 服务。

旁路推流支持两种控制方式，本文仅介绍方案一：

- 方案一：使用客户端接口进行控制，使用 `addPublishStreamUrl` 接口配置 CDN 地址。
- 方案二：使用服务端 API 的 `/rtc/mcu/config` 接口进行控制。本文不做介绍，具体请参见服务端文档[旁路推流](#)。

业务链路

融云直播支持将实时音视频流转推到 CDN 服务，业务链路如下图所示：



配置直播 CDN 推流地址

当主播发布资源成功之后，主播就可设置 CDN 推流地址，设置 CDN 地址有以下几点要求：

1. 必须开通音视频服务和直播服务。
2. 房间模式必须为直播模式。
3. 设置的 CDN 地址不能为空。
4. 最多设置 5 个 CDN 地址，超出会抛出 `RCRTCCodeCDNCountReachToLimit` 错误。
5. 如果多次设置相同的地址，会直接返回成功。

```
- (void)addPublishStreamUrl:(NSString *)url  
completion:(void (^)(BOOL isSuccess, RCRTCCode, NSArray *))completion;
```

- 输入参数：

参数	类型	必填	说明
url	NSString	是	要设置的 CDN 地址
completion	block	否	设置 CDN 之后的回调，会返回所有设置过的 CDN 地址

- 示例代码：

```
[self.liveInfo addPublishStreamUrl:@"https://....." completion:completion];
```

移除配置过的直播 CDN 地址

当主播发布资源成功之后，主播可选择移除一个设置过的 CDN 推流地址，有以下几点要求：

1. 必须开通音视频服务和直播服务。
2. 房间模式必须为直播模式。
3. 移除的 CDN 地址不能为空。
4. 如果移除的地址，之前没有设置过，会直接返回成功。

```
-(void)removePublishStreamUrl:(NSString *)url completion:(void (^)(BOOL isSuccess , RCRTCCode , NSArray *)completion);
```

- 输入参数：

参数	类型	必填	说明
url	NSString	是	要移除的 CDN 地址
completion	block	否	移除 CDN 之后的回调，会返回所有设置过的 CDN 地址

- 示例代码：

```
[self.liveInfo removePublishStreamUrl:@"123" completion:completion];
```

CDN 推流回调

融云在低延迟直播旁路推流到融云 CDN 或第三方 CDN 时，支持将推流的状态变化实时通知您的服务器。具体请参见音视频

服务端文档：

- [CDN 推流回调](#)

融云 CDN 播放器

更新时间:2024-08-30

提示

融云 CDN 插件 (RongRTCPlayer) 从 5.3.0 版本开始提供 CDN 播放器 RCRTCMediaPlayer。

融云 CDN 插件 RongRTCPlayer 中提供了 CDN 流播放组件 RCRTCMediaPlayer，可用于播放第三方 CDN 直播流、获取 SEI 数据、或直接获取音视频帧数据用于自行处理。

- 仅支持播放 CDN 直播流，不支持点播文件或网络点播资源。
- 支持的 CDN 直播流协议：RTMP、HTTP-FLV、HLS
- 仅支持播放 60fps 以下（不含）帧率的视频。

适用场景

融云 CDN 插件 (RongRTCPlayer) 直接在 RTC 房间内提供了 [融云 CDN](#) 服务的订阅、播放等处理逻辑，降低了 App 业务侧的接入成本。

CDN 流播放组件 RCRTCMediaPlayer 可直接输入 CDN 流地址进行播放，不依赖于用户加入 RTC 房，能有效降低 CDN 首屏打开延时，带来更优秀的用户体验。具体适用场景如下：

- 您的 App 自行接入了第三方 CDN 服务，希望通过融云 CDN 播放器组件播放第三方拉流地址。
- 您的 App 使用了 [融云 CDN](#) 服务，按照融云指定的规则自行拼接生成推流、拉流地址后（详见 [融云 CDN 服务配置](#)），通过融云 [客户端旁路推流](#) 或 [服务端旁路推流](#) 接口将 CDN 推流地址提交给融云。配置成功后，可在融云 CDN 播放器中输入您自行生成的 CDN 拉流地址进行播放。该方式要求您将融云 CDN 的「推拉流模式」配置为自行生成推拉流地址模式。
- 您的 App 使用了 [融云 CDN](#) 服务，并直接使用融云服务端接口 [获取融云 CDN 拉流地址](#)，您也可以通过融云 CDN 播放器组件进行播放。该方式要求您将融云 CDN 的「推拉流模式」配置为开播自动推流或开播手动推流模式。

集成 CDN 播放器组件

[RCRTCMediaPlayer](#) 在融云 CDN 插件 (RongRTCPlayer) 中提供。融云 CDN 插件导入方法请参见 [导入 SDK](#)。

集成融云 CDN 插件后请注意导入头文件：

```
#import <RongRTCPlayer/RongRTCPlayer.h>
```

使用 CDN 播放器组件

1. 创建 RCRTCMediaPlayer 实例，同时设置播放事件 RCRTCMediaPlayerEventDelegate 的代理。注意，设置代理必

须在加载资源 (openWithUrl) 调用之前。

```
self.mediaPlayer = [RCRTCMediaPlayer alloc] init];
self.mediaPlayer.delegate = self;
```

2. 加载资源。资源缓冲完成后，不会自动播放。

```
NSURL *url = [NSURL URLWithString:@"rtmps://....."];
int res = [self.mediaPlayer openWithUrl:url];
if (res != 0) {
    NSLog(@"openWithUrl failed:%@",@(res));
}
```

3. 播放资源。

```
int res = [self.mediaPlayer play];
if (res != 0) {
    NSLog(@"play failed:%@",@(res));
}
```

4. 获取渲染视图。如果在 openWithUrl 加载之前调用渲染视图，该接口返回为 nil。

 提示

当设置 dataHandle 代理时播放器不执行内部渲染，该接口返回为 nil。

```
UIView *videoView = [self.mediaPlayer videoView];
[self.view addSubview:videoView];
```

5. (可选) 设置渲染模式。如不设置，默认为 RCRTCVideoFillMode.RCRTCVideoFillModeAspectFit，等比例填充黑边，直到一个维度到达区域边界。

```
//详细见 RCRTCVideoFillMode 枚举
int res = [self.mediaPlayer setRenderMode:RCRTCVideoFillModeAspectFit];
if (res != 0) {
    NSLog(@"setRenderMode failed:%@",@(res));
}
```

6. 销毁当前播放资源。App 用户可能需要切换到另一个主播的 CDN 流，或者切换到不同分辨率的 CDN 流地址时，此时必须先调用 `destroy`，再加载另一个资源 `openWithURL`。

```
int res = [self.mediaPlayer destroy];
if (res != 0) {
    NSLog(@"destroy failed:%@", @(res));
}
```

播放控制

- 在播放过程中可暂停。暂停后调用 `play` 方法可播放资源。

```
int res = [self.mediaPlayer pause];
if (res != 0) {
    NSLog(@"pause failed:%@", @(res));
}
```

- 设置播放音量。如不设置，使用默认值 100。

```
///播放器支持设置播放音量，以及取值当前音量，取值范围 [0,100]，默认值 100。
int res = [self.mediaPlayer setVolume:100];
if (res != 0) {
    NSLog(@"setVolume failed:%@", @(res));
}
```

- 获取当前音量

```
float volume = [self.mediaPlayer volume];
```

获取流信息数据

在加载资源后可以获取流信息数据 [RCRTCMediaStreamInfo](#)，例如音视频的帧率、码率、视频尺寸、音频采样率等。

```
NSArray *infos = [self.mediaPlayer getMediaStreamInfos];
NSLog(@"infos:%@", infos);
```

播放事件回调

通过设置 [RCRTCMediaPlayerEventDelegate](#) 代理，接收播放相关的事件。

- 播放状态变更回调

```
///当播放器的状态发生转变时，会触发以下回调。
- (void)onPlayer:(RCRTCMediaPlayer *)player didChangedState:(RCRTCPlaybackState)state {
    NSString *desc = nil;
    switch (state) {
        case RCRTCPlaybackStateIdle: {
            desc = @"didChangedState: Idle";
        }
        break;
        case RCRTCPlaybackStateLoading: {
            desc = @"didChangedState: Loading";
        }
        break;
        case RCRTCPlaybackStatePlayAble: {
            desc = @"didChangedState: PlayAble";
        }
        break;
        case RCRTCPlaybackStatePlaying: {
            desc = @"didChangedState: Playing";
        }
        break;
        case RCRTCPlaybackStatePausing: {
            desc = @"didChangedState: Pausing";
        }
        break;
        case RCRTCPlaybackStateError: {
            desc = @"didChangedState: Error";
        }
        break;
        default:
            break;
    }
    if (desc.length) {
        NSLog(@"%@", desc);
    }
}
```

- 视频分辨率变更通知

 提示

接收到视频首帧时，会通知上层当前帧分辨率，后续在推流分辨率发生变化时，也会通过该回调通知到上层。开发者可以通过此通知来适当调整当前视图尺寸。

```
- (void)onPlayer:(RCRTCMediaPlayer *)player didChangedVideoSize:(CGSize)size {
    NSLog(@"%@", [NSString stringWithFormat:@"didChangedVideoSize:%@", NSStringFromCGSize(size)]);
}
```

- 音频首帧渲染

```
- (void)onFirstAudioFrameRenderWithPlayer:(RCRTCMediaPlayer *)player {
    NSLog(@"onFirstAudioFrameRenderWithPlayer");
}
```

- 视频首帧渲染

```
- (void)onFirstVideoFrameRenderWithPlayer:(RCRTCMediaPlayer *)player {
    NSLog(@"onFirstVideoFrameRenderWithPlayer");
}
```

- 获取 SEI 数据

📌 提示

- 播放器支持解析拉流资源的媒体增强补充信息（SEI）在接收到 SEI 数据时，会通过以下回调通知到上层。回调的频次和 SEI 帧率一致。
- 配合融云的旁路推流，使用 `RCRTCMediaPlayer` 拉流时，默认会回调拉流房间内多个视频流的合流布局信息。回调频次以及数据结构与 `RCRTCRoomEventDelegate` 的 `didReceiveLiveStreamSEI` 方法相同。

```
- (void)onPlayer:(RCRTCMediaPlayer *)player handleSEIData:(NSString *)data {
    NSLog(@"handleSEIData:%@", data);
}
```

- 发生错误通知

📌 提示

当播放器在播放的过程中出现播放错误时，通过以下回调通知上层。开发者可以适当添加日志记录方便错误排查。

```
- (void)onPlayer:(RCRTCMediaPlayer *)player didOccurError:(NSInteger)errorCode {
    NSLog(@"errorCode:%@", @(errorCode));
}
```

音视频数据回调

播放器提供音视频帧数据回调，需要用户在 `openWithUrl` 调用前设置 [RCRTCMediaPlayerDataHandle](#) 的 `dataHandle` 代理。调用播放时，播放器会上抛音视频帧。

获取视频帧数据

设置 `dataHandle` 代理时后，CDN 播放器不执行内部视频渲染。App 可自行处理视频渲染。

```
- (void)onPlayer:(RCRTCMediaPlayer *)player handleVideoFrame:(RCRTCMediaPlayerVideoFrame *)videoFrame;
```

RCRTCMediaPlayerVideoFrame 参数说明：

参数	类型	说明
<code>pixelFormatType</code>	<code>OSType</code>	当前视频帧编码格式 NV12 或者 i420
<code>width</code>	<code>int</code>	视频的宽度(px)
<code>height</code>	<code>int</code>	视频的高度(px)
<code>yStride</code>	<code>int</code>	对 YUV 数据，表示 Y 缓冲区的行跨度
<code>uStride</code>	<code>int</code>	对 YUV 数据，表示 U 缓冲区的行跨度
<code>vStride</code>	<code>int</code>	对 YUV 数据，表示 V 缓冲区的行跨度
<code>yBuffer</code>	<code>uint8_t *</code>	对 YUV 数据，表示 Y 缓冲区的指针
<code>uBuffer</code>	<code>uint8_t *</code>	对 YUV 数据，表示 U 缓冲区的指针
<code>vBuffer</code>	<code>uint8_t *</code>	对 YUV 数据，表示 V 缓冲区的指针

```

- (void)onPlayer:(RCRTCMediaPlayer *)player handleVideoFrame:(RCRTCMediaPlayerVideoFrame *)videoFrame {
// NV12
if (videoFrame.pixelFormatType == kCVPixelFormatType_420YpCbCr8BiPlanarFullRange) {
CVReturn result = noErr;
NSMutableDictionary *pixelAttributes = @{@"(id)kCVPixelBufferIOSurfacePropertiesKey": @{};};
CVPixelBufferRef pixelBufferRef;
result = CVPixelBufferCreate(kCFAllocatorDefault,
videoFrame.width,
videoFrame.height,
kCVPixelFormatType_420YpCbCr8BiPlanarFullRange,
(__bridge CFDictionaryRef)pixelAttributes,
&pixelBufferRef);
if (result != noErr || !pixelBufferRef) {
NSLog(@"CVPixelBufferCreate failed:%@",@(result));
return;
}

{
CVPixelBufferLockBaseAddress(pixelBufferRef, 0);

void *yDestPlane = CVPixelBufferGetBaseAddressOfPlane(pixelBufferRef, 0);
void *uvDestPlane = CVPixelBufferGetBaseAddressOfPlane(pixelBufferRef, 1);
size_t heightY = CVPixelBufferGetHeightOfPlane(pixelBufferRef, 0);
size_t heightUV = CVPixelBufferGetHeightOfPlane(pixelBufferRef, 1);
size_t dstStrideY = CVPixelBufferGetBytesPerRowOfPlane(pixelBufferRef, 0);
size_t dstStrideUV = CVPixelBufferGetBytesPerRowOfPlane(pixelBufferRef, 1);
if (videoFrame.yBuffer) {
memcpy(yDestPlane, videoFrame.yBuffer, dstStrideY * heightY);
}
if (videoFrame.uBuffer) {
memcpy(uvDestPlane, videoFrame.uBuffer, dstStrideUV * heightUV);
}
CVPixelBufferUnlockBaseAddress(pixelBufferRef, 0);
}
// pixelBufferRef to do something

CVBufferRelease(pixelBufferRef);
}
}

```

获取音频帧数据

如需自行处理音频播放，可将 CDN 播放器的音量置为 0。

```

- (void)onPlayer:(RCRTCMediaPlayer *)player handleAudioFrame:(RCRTCMediaPlayerAudioFrame *)audioFrame;

```

RCRTCMediaPlayerAudioFrame 参数说明：

参数	类型	说明
sampleRateHz	UInt32	采样率
channels	UInt32	声道数量(如果是立体声，数据是交叉的)
samplesPerChannel	UInt32	每个声道的采样点数
bytesPerSample	UInt32	每个采样点的字节数: 对于 PCM 来说，一般使用 16 bit，即两个字节
buffer	void*	缓存区数据起始地址

参数	类型	说明
bufferSize	UInt32	缓存区数据大小 bufferSize = samplesPerChannel × channels × bytesPerSample
renderTimeMs	UInt64	外部音频帧的渲染时间戳

```
- (void)onPlayer:(RCRTCMediaPlayer *)player handleAudioFrame:(RCRTCMediaPlayerAudioFrame *)audioFrame {
    AudioBuffer buffer;
    buffer.mData = audioFrame.buffer;
    buffer.mDataByteSize = audioFrame.bufferSize;
    buffer.mNumberChannels = audioFrame.channels;
    AudioBufferList abl;
    abl.mNumberBuffers = 1;
    abl.mBuffers[0] = buffer;
    // AudioBufferList to do something
}
```

音量

更新时间:2024-08-30

本文介绍如何设置音频采集音量、耳返播放的音量以及如何对播放音频进行静音。

本文不介绍混音音量控制。请另行参见「混音」文档。

设置采集音量

采集是指音频信号由采集设备（麦克风）采集，然后传输到发送端的过程。App 可通过 [RCRTCmicOutputStream](#) 的 `recordingVolume` 设置麦克风为音频源的音频输出流音量大小。

- 如果 SDK 版本 < 5.3.4，采集音量大小范围为 0-100。默认 100。
- 如果 SDK 版本 \geq 5.3.4，采集音量大小范围为 0-200。默认 100。

```
/*!  
麦克风的音量，范围：0~200，默认值：100  
*/  
@property (nonatomic, assign) NSInteger recordingVolume;
```

[RCRTCmicOutputStream](#) 对象不能自行创建。App 通过 `[RCRTCEngine sharedInstance].defaultAudioStream` 获取后 `RCRTCmicOutputStream` 对象可进行设置：

```
[RCRTCEngine sharedInstance].defaultAudioStream.recordingVolume = 100;
```

设置耳返音量

耳返是指播放采集设备输出音频的过程。SDK 支持通过 [RCRTCAudioEffectManager](#) 管理耳返功能。

App 通过 `[RCRTCEngine sharedInstance].audioEffectManager` 获取 `RCRTCAudioEffectManager` 后，可打开或关闭耳返功能，调整耳返音量。音量取值范围 0~100。

```
// 打开或关闭耳返功能  
[[RCRTCEngine sharedInstance].audioEffectManager enableInEarMonitoring:YES];  
  
// 设置耳返音量  
[[RCRTCEngine sharedInstance].audioEffectManager setInEarMonitoringVolume:100];
```

设置播放音量

- 如果 SDK 版本 < 5.3.4，不支持设置播放音量值，仅支持设置指定或全部远端音频流为静默状态，实现静音效果。
- 如果 SDK 版本 \geq 5.3.4，可调节远端播放音量。

调节远端播放音量

提示

SDK 从 5.3.4 版本开始支持调节远端播放音量。

从 5.3.4 开始，App 可以通过 RCRTCEngine 的 remotePlaybackVolume 属性调节远端播放音量。音量范围为 [0-200]，0 表示静音。加入房间前后均可调节音量。该方法调节的是本地播放的所有远端用户混音后的音量。

```
// 设置远端音频播放音量
[[RCRTCEngine sharedInstance].remotePlaybackVolume = 150];
```

静音本地音频流

媒体流对象都可以调用 RCRTCStream 的 isMute 属性设置是否静默。

对于本地音频流，如果 isMute 为 YES 则不再发送本地资源，也不能播放，但不影响音频数据采集。

```
// 设置本地音频流为静默状态
[[RCRTCEngine sharedInstance].defaultAudioStream setIsMute:YES];
```

静音远端音频流

媒体流对象都可以调用 RCRTCStream 的 isMute 属性设置是否静默。

对于远端音频流，如果 isMute 为 YES 则不再播放远端音频，但不影响远端音频数据接收。

静音房间内全部远端音频流

远端流静音是将房间内接收到所有远端音频流静音。默认不开启。App 需要在用户加入房间后，在当前房间对象 RCRTCRoom 上调用 muteAllRemoteAudio。

提示

当前静音方式只是不播放接收的音频数据。

```
/*!  
设置所有远端用户是否静音  
  
@param mute 是否静音所有远端用户, YES 禁止 NO 允许  
@discussion  
将所有远端用户静音, 注: 该功能只是不播放接收到的音频数据  
  
@remarks 音频流处理  
*/  
- (void)muteAllRemoteAudio:(BOOL)mute;
```

参数	类型	说明
mute	BOOL	YES (开启静音) NO (默认不静音)

```
//self.room 为 sdk 加入的房间对象  
[self.room muteAllRemoteAudio:YES];
```

音频路由

更新时间:2024-08-30

提示

SDK 从 5.2.3 开始提供音频路由功能。

SDK 提供音频路由功能，用于管理 App 播放音频时的输出设备。主要功能如下：

- 设置默认音频路由。在无外接设备时，可设置默认使用的设备内置音频输出设备输出（听筒或扬声器）。一旦接入外部设备，SDK 仅会使用外接设备。
- 在接入有线耳机、蓝牙耳机、蓝牙音响等外部设备时，自动根据设备连接顺序、可用状态切换当前输出设备。当有多个外接设备时，音频会通过最后一个接入的设备播放。

本文主要描述了 SDK 在不同场景下的默认音频路由，默认音频路由修改方式，和音频路由监听方法。

默认音频路由

默认音频路由是指 App 所在设备的默认音频输出设备，例如移动设备上的听筒或扬声器。SDK 默认音频路由为扬声器。

更改默认音频路由

当 App 用户没有连接外部音频输出设备时，SDK 会使用默认音频路由。

加入房间前，调用 [setDefaultAudioRouteToSpeaker](#) 可更改默认音频路由。通过该方法的 `defaultToSpeaker` 参数控制 SDK 是否使用扬声器播放音频。

以下示例中将默认音频路由修改为听筒。

```
[[RCRTCEngine sharedInstance] setDefaultAudioRouteToSpeaker:NO];
```

更改当前音频路由

在 App 用户未接入任何外接音频输出设备时，SDK 会使用默认音频路由的设置。如需更改当前音频路由，有两种方式：

- 直接修改 SDK 默认音频路由配置，您可以在通信过程中在扬声器、听筒之间动态切换。
- 在 App 已将 SDK 默认音频路由设置为听筒的前提下，可调用 [EnableSpeaker](#) 方法并设置 `YES`，实现将音频从听筒转为扬声器输出的效果。设置为 `NO` 时 SDK 会恢复使用默认音频路由的配置。该方法在加入房间前后调用均可生效。该方法只切换当前的音频路由为扬声器，不会影响 SDK 的默认音频路由设置。

以下示例中调用 [EnableSpeaker](#) 方法，更改当前音频路由为扬声器。

```
[[RCRTCEngine sharedInstance] enableSpeaker:YES];
```

在 App 用户接入外接音频输出设备后，SDK 会自动管理音频路由。具体行为如下：

1. 音频路由自动切换到 App 用户连接的外部音频输出设备。
2. 如果 App 用户先后连接了多个外部设备，则音频路由会自动切换到最后一个连接的音频输出设备。
3. 如果 App 用户移除当前输出设备，则音频路由会自动切换到上一个连接的音频输出设备。
4. 如果 App 用户移除所有外接音频输出设备，SDK 切换到使用默认音频路由输出。

提示

因为 iOS 系统限制，当移动设备连接到耳机或蓝牙音频设备时，无法将音频路由更改为扬声器。

获取当前音频路由

对音频路由的任何更改都会触发 [RCRTCEngineEventDelegate](#) 中的 [didAudioRouteChanged](#) 回调。您可以使用此回调来获取当前的音频路由。

```
/// 音频路由变更回调
/// @param routeType 当前音频路由类型
- (void)didAudioRouteChanged:(RCRTCAudioRouteType)audioRouteType {
/// 扬声器
RCRTCAudioRouteTypeSpeaker,
/// 听筒
RCRTCAudioRouteTypeReceiver,
/// 耳机
RCRTCAudioRouteTypeHeadphone,
/// 蓝牙
RCRTCAudioRouteTypeBluetooth,
}
```

音频模式

更新时间:2024-08-30

为了来满足不同场景对音频设置的需求，同时降低使用复杂度，融云 RongRTC Lib 从 5.1.0 开始，对音频码率（5.1.0 音频通话质量）和音频模式（5.1.0 音频通话模式）进行了接口合并封装，并重新设计，对外提供音频质量 + 音频模式的接口，推荐使用 5.1.0 最新接口。

场景和音质选择

SDK 提供了音频场景 [RCRTCAudioScenario](#) 与音频质量 [RCRTCAudioQuality](#) 两个枚举类，推荐使用方式如下：

推荐使用场景	AudioScenario 场景枚举	AudioQuality 音质枚举	码率
通话，会议场景（默认）	DEFAULT	SPEECH	人声音质，编码码率最大值为 32Kbps
语聊房，音乐播放场景	MUSIC_CHATROOM	MUSIC	标清音乐音质，编码码率最大值为 64Kbps
音乐教学场景	MUSIC_CLASSROOM	MUSIC_HIGH	高清音乐音质，编码码率最大值为 128Kbps

设置音频通话质量和模式

SDK 在 `RCRTCMicOutputStream` 类中提供了 `setAudioQuality:Scenario:` 接口，用于设置音频场景 [RCRTCAudioScenario](#) 与音频质量 [RCRTCAudioQuality](#)。您可以在加入房间前或者加入房间后，通过 `[RCRTCEngine sharedInstance].defaultAudioStream` 实例调用 `setAudioQuality:Scenario:` 接口进行设置。

音频通话质量可以和音频通话模式进行任意组合，达到特殊场景需求，以下示例代码为几种常见场景推荐值。

```

/*!
普通通话模式（普通音质模式），满足正常音视频场景，人声音质，编码码率最大值为32Kbps
*/
[[RCRTCEngine sharedInstance].defaultAudioStream setAudioQuality:RCRTCAudioQualitySpeech
Scenario:RCRTCAudioScenarioDefault];

/*!
音乐教室模式，提升声音质量，适用对乐器演奏音质要求较高的场景，高清音乐音质，编码码率最大值为128Kbps
*/
[[RCRTCEngine sharedInstance].defaultAudioStream setAudioQuality:RCRTCAudioQualityMusicHigh
Scenario:RCRTCAudioScenarioMusicClassRoom];

/*!
音乐聊天室模式，提升声音质量，适用对音乐演唱要求较高的场景，高清音乐音质，编码码率最大值为128Kbps
*/
[[RCRTCEngine sharedInstance].defaultAudioStream setAudioQuality:RCRTCAudioQualityMusicHigh
Scenario:RCRTCAudioScenarioMusicChatRoom];

```

获取音频通话质量和模式

您可以通过 `[RCRTCEngine sharedInstance].defaultAudioStream` 实例的属性 `audioQuality` 和 `audioScenario` 获取音频场景 [RCRTCAudioScenario](#) 与音频质量 [RCRTCAudioQuality](#)：

```
/*!
音频通话质量
默认：人声音质，RCRTCAudioQualitySpeech
*/
@property (nonatomic, assign, readonly) RCRTCAudioQuality audioQuality;

/*!
音频通话模式
默认：普通通话模式，RCRTCAudioScenarioDefault
*/
@property (nonatomic, assign, readonly) RCRTCAudioScenario audioScenario;
```


混音 前提条件

更新时间:2024-08-30

从网络音源混音的能力依赖 RongRTCPlayer 插件。SDK 需要使用以下插件将网络文件下载后播放。如有需要，请集成以下插件：

```
// x.y.z，请填写具体的 SDK 版本号，新集成用户建议使用 SDK 和插件的最新版。
pod 'RongCloudRTC/RongRTCPlayer', '~> x.y.z' 融云 CDN + 混音网络资源文件（可选）
```

开始混音

混音功能支持将用户自定义的音频数据与本地麦克风采集的音频数据进行混合，加入房间并发布默认的音频流后调用 `startMixingWithURL:playback:mixerMode:loopCount:` 方法混音指定的音频文件。

```
- (BOOL)startMixingWithURL:(NSURL *)fileURL
  playback:(BOOL)isPlay
  mixerMode:(RCRTCMixerMode)mode
  loopCount:(NSUInteger)count;
```

参数	类型	说明
fileURL	NSURL	本地文件或者网络资源 URL。注意，参数 fileURL 为网络资源时，在线资源混音需要依赖 RongRTCPlayer 插件。
isPlay	BOOL	是否在本本地播放
mode	RCRTCMixerMode 🔗	混音方式
count	NSUInteger	循环混音或者播放次数

混音成功后，返回 YES。

```
//NSURL *mediaUrl = [NSURL URLWithString:@"https://xxxx.mp3"];
NSString *audioFilePath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"mp3"];
NSURL *mediaUrl = [NSURL fileURLWithPath:audioFilePath];
[[RCRTCAudioMixer sharedInstance] startMixingWithURL:mediaUrl
  playback:YES
  mixerMode:RCRTCMixerModeMixing
  loopCount:NSUIntegerMax];
```

切换左右声道

📌 提示

SDK 从 5.1.13 开始支持切换左右声道。仅支持在与本地文件混音时切换声道。不支持网络资源文件。

与本地文件混音时，可以调用 `setAudioDualMonoMode` 单独选择混音的左右声道。如未调用默认模式为使用立体声混音。

例如调用 `setAudioDualMonoMode` 设置为左声道混音时，SDK 会拷贝当前音频的左声道数据填充右声道，进而两个声道都是左声道数据，开始混音后用户听到的就是当前左声道混音的声音。您可以通过左右声道切换实现原唱、伴唱切换，以满足卡拉 OK 场景下的需求（通过声道切换原唱、伴唱要求音源支持，即原唱、伴唱音轨分别位于两个独立声道）。

```
/*!
设置混音声道模式
@param mode 声道模式 0 立体声混音, 1 左声道混音, 2 右声道混音
@discussion
只针对本地文件资源混音播放产生效果, 不支持网络资源的 url
Add from 5.1.13

@remarks 音频配置
@return 设置是否成功
*/
- (BOOL)setAudioDualMonoMode:(RCRTCAudioDualMonoMode)mode;
```

mode 为 SDK 提供的混音模式 ([RCRTCAudioDualMonoMode](#))，支持选择左右声道或者立体声。

设置成功后返回 YES，设置即时生效。

```
RCRTCAudioDualMonoMode mode = RCRTCAudioDualMonoLeft;//RCRTCAudioDualMonoRight
[[RCRTCAudioMixer sharedInstance] setAudioDualMonoMode:mode]
```

控制混音状态（结束、暂停、恢复）

- 混音结束或结束混音时需要调用以下方法结束混音。

```
[[RCRTCAudioMixer sharedInstance] stop];
```

- 暂停混音

```
[[RCRTCAudioMixer sharedInstance] pause];
```

- 恢复混音

```
[[RCRTCAudioMixer sharedInstance] resume];
```

- 获取当前混音状态

```
if(RTCMixEngineStatusPlaying == [RCRTCAudioMixer sharedInstance].status) {  
    // 处理正在播放混音的逻辑  
}
```

设置混音播放位置

提示

从 5.2.5.1 开始，SDK 支持先设置播放位置（`setPlayProgress`），再调用开始混音（`startMixingWithURL`）。

在混音开始前后均可以设置播放位置。如果希望从混音文件指定位置开始混音，建议先设置播放位置，再调用开始混音。

```
[[RCRTCAudioMixer sharedInstance] setPlayProgress:progress];
```

参数	类型	说明
progress	float	设置播放进度 取值范围 [0,1]

音量控制

SDK 支持控制混音音源输入的音量和混音在本地播放的音量。

调节混音输入音量

设置音频文件混音时的输入音量，取值范围 [0,100]，默认值 100。

```
// 设置音频文件混音输入音量为 50  
[RCRTCAudioMixer sharedInstance].mixingVolume = 50;
```

调节混音本地播放音量

如果设置混音时选择了本地播放，您可以调节音频文件本地播放音量，取值范围 [0,100]，默认值 100。

```
// 设置音频文件本地播放音量为 50  
[RCRTCAudioMixer sharedInstance].playingVolume = 50;
```

获取音频文件时长

SDK 支持获取混音音频文件时长。网络资源 URL 在播放后才能拿到音频时长。

```
// 获取音频文件时长  
Float64 duration = [RCRTCAudioMixer durationOfAudioFile:url];
```

参数	类型	说明
url	NSURL	文件 URL 音频时长（网络资源 URL 在播放后才能拿到音频时长）

返回音频文件时长，单位为秒。

混音事件回调

设置混音播放代理，通过代理用户可以得到 播放/混音进度 和 播放/混音状态。

```
[RCRTCAudioMixer sharedEngine].delegate = self;
```

```

#pragma mark - RCRTCAudioMixerAudioPlayDelegate - AudioMixer 的播放代理

/*!
当前播放进度

@param progress 播放进度 range [0,1]
@discussion
当前播放进度

@remarks 代理
*/
- (void)didReportPlayingProgress:(float)progress{
NSLog(@"当前播放进度 - %@", @(progress));
}

/*!
混音状态

@param mixingState 混音状态
@param mixingReason 混音状态改变的原因
@discussion
当前混音状态
Add from 5.1.3

@remarks 代理
*/
- (void)didAudioMixingStateChanged:(RCRTCAudioMixingState)mixingState
reason:(RCRTCAudioMixingReason)mixingReason{
if (mixingState == RCRTCMixingStatePlaying){
// 开始混音，可能的原因有：
if (mixingReason == RCRTCMixingReasonStartedByUser){
// 调用了开始混音方法 startMix
}else if (mixingReason == RCRTCMixingReasonStartNewLoop){
// 调用了 startMix 传入的loopCount < 0 (无限循环) 或 > 1时，自动开始下一次混音。
}else if (mixingReason == RCRTCMixingReasonResumedByUser){
// 调用了 resume 方法继续开始混音
}else if (mixingReason == RCRTCMixingReasonFileLoaded){
// 本地或网络混音文件已加载完成 (SDK 必须等待资源加载完成后才能播放混音文件)，该回调要求 SDK 版本 ≥ 5.2.5.1。
}
}else if (mixingState == RCRTCMixingStateStop){
// 混音完成，可能的原因有：
if (mixingReason == RCRTCMixingReasonAllLoopsCompleted){
// 调用 startMix方法时，传入的 loopCount > 0，并且 loopCount 次数的混音已经完成
}else if (mixingReason == RCRTCMixingReasonOneLoopCompleted){
// 调用 startMix方法时，传入的 loopCount < 0 (无限循环) 或 > 1，混音完成一次。接下来会继续自动开始下一次混音。
}else if (mixingReason == RCRTCMixingReasonStoppedByUser){
// 调用 stopMix 方法停止混音
}
}else if (mixingState == RCRTCMixingStatePause){
// 暂停了混音，mixingReason 为 RCRTCMixingReasonPausedByUser
}else if (mixingState == RCRTCMixingStateFailed){
// 加载失败，mixingReason 为 RCRTCMixingReasonCanNotOpen
}
}

@end

```

使用自定义音频数据进行混音

您可以通过写入自定义音频数据的方式进行混音。自定义音频数据可以使用 SDK 内置的格式要求或使用您自定义的格式。

使用内置音频格式写入自定义混音数据

1. 通过以下该属性，获取方法 `writeAudioBufferList:frames:sampleTime:playback:` 写入 `AudioBufferList` 要求的格式。

```
@property (nonatomic, readonly, class) AudioStreamBasicDescription writeAsbd;
```

2. 向即将发送的音频数据中混合自定义音频数据。

- 方法

```
- (void)writeAudioBufferList:(const AudioBufferList*)abl  
frames:(UInt32)frames  
sampleTime:(SInt64)sampleTime  
playback:(BOOL)isPlay;
```

`abl` 为 `AudioBufferList` 类型布局，必须使用上一步中获取的格式。

参数	类型	说明
<code>abl</code>	<code>AudioBufferList</code>	音频数据
<code>frames</code>	<code>UInt32</code>	音频帧个数
<code>sampleTime</code>	<code>SInt64</code>	音频帧时间戳
<code>isPlay</code>	<code>BOOL</code>	是否在本地图播放

- 示例代码：

```

SInt64 sampleTime = 0;
while (true) {
CMSampleBufferRef sample = [_outAudioTrack copyNextSampleBuffer];
if (!sample) {
break;
}
AudioBufferList abl = {0};
CMBlockBufferRef blockBuffer;
CMSampleBufferGetAudioBufferListWithRetainedBlockBuffer(sample,
NULL,
&abl,
sizeof(abl),
NULL,
NULL,
0,
&blockBuffer);
CMItemCount count = CMSampleBufferGetNumSamples(sample);
[[RCRTCAudioMixer sharedEngine] writeAudioBufferList:&abl
frames:(UInt32)count
sampleTime:sampleTime
playback:YES];
sampleTime += count;
CFRelease(blockBuffer);
CFRelease(sample);
if (_status == RongRTCFileVideoCapturerStatusStopped) {
break;
}
}
}

```

使用自定义格式写入混音数据

提示

SDK 从 5.2.3 版本开始支持该功能。

如果您希望自行指定 `AudioBufferList` 数据的格式，可以使用以下方法写入混音数据。

```
/*!
写入自定义音频数据
@param abl 音频数据，格式为 PCM
@param frames 音频帧个数
@param sampleTime 音频帧时间戳
@param asbd 音频格式描述
@param isPlay 是否在本地播放

@discussion
写入自定义音频数据

@remarks 音频流处理
*/
- (void)writeAudioBufferList:(const AudioBufferList *)abl
frames:(UInt32)frames
sampleTime:(SInt64)sampleTime
asbd:(AudioStreamBasicDescription)asbd
playback:(BOOL)isPlay;
```


流处理

本地音频流处理

更新时间:2024-08-30

1. 获取最终编码发送的音频数据

此回调返回的数据是最终编码发送的音频数据，比如经过SDK默认的3A(增益、回声、降噪)处理，如果还使用了混音、音效、自定义视频等功能，那么返回的数据中同时包含了以上效果的声音。

[RCRTCMicOutputStream](#) 中 `willSendAudioBufferCallback`

```
@property (nonatomic, copy, nullable) RCRTCAudioDataCallback willSendAudioBufferCallback;
```

属性说明:

参数	类型	说明
willSendAudioBufferCallback	RCRTCAudioDataCallback	将要发送的音频 pcm 数据的回调，用户可以直接处理该音频数据

回调说明:

```
typedef void(^RCRTCAudioDataCallback)(UInt32 inNumberFrames,  
AudioBufferList * _Nonnull ioData,  
const AudioTimeStamp * _Nonnull inTimeStamp);
```

返回参数:

参数	类型	说明
<code>inNumberFrames</code>	UInt32	帧个数
<code>ioData</code>	AudioBufferList	音频 pcm 数据
<code>inTimeStamp</code>	AudioTimeStamp	音频时间戳

2. 获取麦克风采集的原始音频数据 (v5.1.6版本新增)

此回调返回的数据是采集的原始音频数据，不经过SDK的任何处理。

[RCRTCMicOutputStream](#) 中 `recordAudioDataCallback`

```
@property (nonatomic, copy, nullable) RCRTCAudioFrameCallback recordAudioDataCallback;
```

属性说明:

参数	类型	说明
recordAudioDataCallback	RCRTCAudioFrameCallback	麦克风音频 pcm 数据的回调，用户可以直接处理该音频数据

回调说明：

```
typedef void (^RCRTCAudioFrameCallback)(RCRTCAudioFrame *_Nonnull frame);
```

返回参数：

参数	类型	说明
frame	RCRTCAudioFrame*	音频数据

远端音频

1. 获取订阅的所有远端用户混音后的音频数据

接收后播放前的远端音频，上报的远端音频数据是将收到的所有远端音频进行混音后的数据。此数据从 RCRTCRoom 中的如下代理方法回调。

```
@property (nonatomic, copy, nullable) RCRTCAudioDataCallback receivedAudioBufferCallback;
```

属性说明：

参数	类型	说明
receivedAudioBufferCallback	RCRTCAudioDataCallback	麦克风音频 pcm 数据的回调，用户可以直接处理该音频数据

返回参数：

参数	类型	说明
frame	RCRTCAudioFrame*	音频数据
inNumberFrames	UInt32	帧个数
ioData	AudioBufferList	音频 pcm 数据
inTimeStamp	AudioTimeStamp	音频时间戳

2. 获取订阅的指定远端用户音频数据（v5.1.6版本新增）

订阅的远端流音频数据。此数据从 RCRTCAudioInputStream 中的如下代理方法回调。

```
@property (nonatomic, copy, nullable) RCRTCAudioFrameCallback audioDataCallback;
```

属性说明：

参数	类型	说明
audioDataCallback 	RCRTCAudioFrameCallback	麦克风音频 pcm 数据的回调，用户可以直接处理该音频数据

参数:

参数	类型	说明
frame	RCRTCAudioFrame*	音频数据

RCRTCAudioFrame对象的各字段:

字段	类型	说明
bytes	uint8_t*	音频数据
length	int32_t	音频数据长度 (字节数)
channels	int32_t	声道数
sampleRate	int32_t	采样率
bytesPerSample	int32_t	位深
samples	int32_t	帧数
renderTimeMs	uint64_t	时间戳

美声特效

更新时间:2024-08-30

融云音视频 SDK 以插件库的形式 (RongVoiceBeautifier) 提供了多种美声特效选项。本插件会对麦克风采集的数据和耳返数据进行处理，本地与远端用户均会听到处理后的效果。

提示

CallKit、CallLib、RTCLib 从 5.1.6 版本开始，支持美声插件。

集成美声插件

在导入 SDK 前，您可以前往 [融云官网 SDK 下载页面](#) 确认当前最新版本号。

通过 CocoaPods 管理依赖

1. 请在 Podfile 文件中添加下面内容。

```
pod 'RongCloudRTC/RongVoiceBeautifier', '~> x.y.z'
```

提示

x.y.z 代表当前 RTCLib 的具体版本号。您可以在 [融云官网 SDK 下载页面](#) 或 [CocoaPods 仓库](#) 进行查询。

手动集成

1. 前往 [融云官网 SDK 下载页面](#)，下载声音特效库：
2. 导入以下内容：
 - RongVoiceBeautifier.bundle
 - RongVoiceBeautifier.framework。
3. 请将工程中 Target > Build Settings > Other Linker Flags 在此项中添加 -ObjC。为避免造成编译失败，禁止使用 -all_load 和 -force_load。
4. 请务必确保资源包 RongVoiceBeautifier.bundle 正确添加到项目中，否则声音音效不会生效。

使用美声插件

将 RongCloudRTC 升级到 5.1.6 及以后版本，在需要美颜功能的文件中导入声音特效插件头文件：

```
#import <RongVoiceBeautifier/RCRTCVoiceBeautifierEngine.h>
```

设置美声特效

确保在设置美声特效之前已经初始化 RCRTCEngine 引擎。

1. 使用 RCRTCVoiceBeautifierEngine 单例的 setPreset 方法设置声音特效。RCVoiceBeautifierType [🔗](#) 定义了预设的音效。例如，设置为男孩音效（RCVoiceBeautifierTypeEffectBoy）。

```
// 设置声音特效
[[RCRTCVoiceBeautifierEngine sharedInstance] setPreset:RCVoiceBeautifierTypeEffectBoy];
```

2. 使用 RCRTCVoiceBeautifierEngine 单例的 enable 方法设置声音特效是否启用。通过返回值可判断是否成功启用，返回 0 表示成功，返回 -1 表示失败。

```
// 设置声音特效是否开启
BOOL Success = [[RCRTCVoiceBeautifierEngine sharedInstance] enable:YES];
```

3. 检查美声特效是否已启用。

```
- (BOOL)isEnabled;
```

取消美声特效

设置美声特效后，会在当前客户端持续生效。如有需要，可在切换房间时修改、重置美声效果，直接关闭美声特效功能。

📌 提示

因为 RCRTCVoiceBeautifierEngine 类是单例对象，所以会保留开发者设置的声音特效参数。

```
// 例如，退出房间时重置美声特效为使用原声
[[RCRTCVoiceBeautifierEngine sharedInstance] setPreset:RCVoiceBeautifierTypeNone];
```

预置声音音效

[RCVoiceBeautifierType](#) 定义了预设的音效效果，提供美声、变声、混响三类效果。

声音类型	特效类别	说明
RCVoiceBeautifierTypeNone	原声	关闭变声
RCVoiceBeautifierTypeAcousticsFull	美声	饱满
RCVoiceBeautifierTypeAcousticsLow	美声	低沉
RCVoiceBeautifierTypeAcousticsHyperactivity	美声	高亢
RCVoiceBeautifierTypeMagnetic	美声	磁性(男) (SDK \geq 5.4.1)
RCVoiceBeautifierTypeFresh	美声	清新(女) (SDK \geq 5.4.1)
RCVoiceBeautifierTypeVitality	美声	活力(女) (SDK \geq 5.4.1)
RCVoiceBeautifierTypeMellow	美声	圆润 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeClear	美声	清澈 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeRinging	美声	嘹亮 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeVigorous	美声	浑厚 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectFalsetto	变声	假声
RCVoiceBeautifierTypeEffectBoy	变声	小男孩
RCVoiceBeautifierTypeEffectBoyToMan	变声	男孩转中年
RCVoiceBeautifierTypeEffectOldMan	变声	老年人
RCVoiceBeautifierTypeEffectGirl	变声	小女孩
RCVoiceBeautifierTypeSister	变声	小姐姐 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectGirlToWoman	变声	女孩转中年
RCVoiceBeautifierTypePigKing	变声	猪八戒 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectHulk	变声	绿巨人
RCVoiceBeautifierTypeAcousticsKTV	混响	KTV
RCVoiceBeautifierTypeAcousticsVocalConcert	混响	演唱会
RCVoiceBeautifierTypeEffectStudio	混响	录音棚 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectPhonograph	混响	留声机 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectStereo	混响	立体声 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectValley	混响	空旷 (SDK \geq 5.4.1)
RCVoiceBeautifierTypeEffectLobby	混响	3D 环绕 (SDK \geq 5.4.1)

性能与功耗

声音特效插件 `RCRTCVoiceBeautifier.framework` 自身包体积为 37.7M。添加声音特效插件后：

- CPU 消耗增加 5% 左右。
- Memory 消耗增加 3M 左右。
- 打包应用包体积增加 13.6M 左右。

大小流

更新时间:2024-08-30

大小流模式是指在发布资源时上传一大一小两道视频流。

SDK 默认为摄像头视频流打开发布大小流功能，即每个用户在发布视频资源时自动发布大小两个视频流。小流的分辨率默认跟随大流。

提示

在多人音视频通话过程中，大小流模式可有效减少下行带宽占用。订阅方可按需订阅小流。

小视频流与大视频流的分辨率对应关系如下:

大流分辨率	小流分辨率	比例
176X144	176X144	11:9
180X180	180X180	1:1
256X144	256X144	16:9
240X180	240X180	4:3
320X180	256X144	16:9
240X240	180X180	1:1
320X240	240X180	4:3
360X360	180X180	1:1
480X360	240X180	4:3
640X360	256X144	16:9
480X480	180X180	1:1
640X480	240X180	4:3
720X480	240X180	3:2
848X480	256X144	9:5
960X720	240X180	4:3
1280X720	256X144	16:9
1920X1080	256X144	16:9

发布方开关大小流

发布本地视频流之前，可以通过 `[RCRTCEngine sharedInstance].defaultVideoStream` 取得将要发布的视频流。

设置 `defaultVideoStream` 对象中的 `enableTinyStream` 属性可以控制是否发布默认视频的小流视频，如下：

```
@property (nonatomic, assign, readwrite) BOOL enableTinyStream;
```

参数	类型	说明
enableTinyStream	BOOL	是否启用视频小流。

如果摄像头视频流，默认为开启。如果自定义视频流，默认为关闭。

```
[RCRTCEngine sharedInstance].defaultVideoStream.enableTinyStream = YES;
```

订阅方切换大小流

在订阅远端视频时或视频通话过程中，可通过 `tinyStreams` 和 `avStreams` 参数订阅或切换远端用户的大小视频流，同一个流只能填写在 `avStreams` 或 `tinyStreams` 中的一个数组中。

```
- (void)subscribeStream:(nullable NSArray<RCRTCInputStream *> *)avStreams  
tinyStreams:(nullable NSArray<RCRTCInputStream *> *)tinyStreams  
completion:(nullable RCRTCOperationCallback)completion;
```

参数	类型	说明
avStreams	NSArray	视频大流数组
tinyStreams	NSArray	视频小流数组
completion	RCRTCOperationCallback	订阅完成后回调

```
// avStreams 和 tinyAVStreams 中的 `RongInputAVStream` 对象可从上报的远端发布流的代理方法中取得  
[[RCRTCEngine sharedInstance].room.localUser subscribeAVStream:avStreams  
tinyStreams:tinyAVStreams  
completion:^(BOOL isSuccess, RCRTCCode desc) {  
}];
```


分辨率/码率/帧率设置

更新时间:2024-08-30

您可以通过 `RCRTCVideoStream` 父类属性 `videoConfig` 设置视频参数。

提示

`RCRTCVideoStream` 对象需要通过 `[RCRTCEngine sharedInstance].defaultVideoStream` 获取。

视频参数

`RCRTCVideoStreamConfig` 定义了视频流的分辨率、码率、与帧率参数:

```
@property (nonatomic, strong) RCRTCVideoStreamConfig *videoConfig;
```

参数	类型	说明	默认值
<code>videoSizePreset</code>	RCRTCVideoSizePreset	摄像头输出的视频分辨率	<code>RCRTCVideoSizePreset640x480</code>
<code>videoFps</code>	RCRTCVideoFPS	视频发送帧率	<code>RCRTCVideoFPS15</code>
<code>maxBitrate</code>	<code>NSUInteger</code>	最大码率，默认 640x480 分辨率时	900 kbps
<code>minBitrate</code>	<code>NSUInteger</code>	最小码率，默认 640x480 分辨率时	200 kbps

在通话过程中，实际视频码率在最小码率和最大码率之间根据网络情况浮动。

设置大流的视频参数

在发布本地摄像头视频资源之前，可以通过 `[RCRTCEngine sharedInstance].defaultVideoStream` 对象设置发送大流的视频参数。

参数说明：

参数	类型	说明
<code>videoConfig</code>	RCRTCVideoStreamConfig	视频参数采集参数

示例代码：

```
RCRTCVideoStreamConfig *videoConfig = [[RCRTCVideoStreamConfig alloc] init];
videoConfig.videoSizePreset = RCRTCVideoSizePreset720x480;
videoConfig.videoFps = RCRTCVideoFPS30;
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoConfig:videoConfig];
```

设置小流的视频参数

如果开启了本地摄像头视频小流，可在发布本地摄像头视频资源之前，通过 [RCRTCEngine sharedInstance].defaultVideoStream 对象设置发送的摄像头小流视频参数。

```
- (BOOL)setTinyVideoConfig:(RCRTCVideoStreamConfig *)config;
```

- 参数说明：

参数	类型	说明
videoConfig	RCRTCVideoStreamConfig	视频参数采集参数

- 示例代码：

```
RCRTCVideoStreamConfig *videoConfig = [[RCRTCVideoStreamConfig alloc] init];
videoConfig.videoSizePreset = RCRTCVideoSizePreset720x480;
videoConfig.videoFps = RCRTCVideoFPS30;
[[RCRTCEngine sharedInstance].defaultVideoStream setTinyVideoConfig:videoConfig];
```

流处理

更新时间:2024-08-30

SDK 支持应用程序获取视频流数据，自行进行美颜、录像等处理后，再把数据再返回给 SDK 进行发送或渲染。

在发送前处理本地视频流

应用程序可以在 SDK 发送视频流前处理视频流数据，修改会对发送到远端以及本地视频显示的视频数据生效。请在 `RCRTCVideoOutputStream` 类中的 `videoSendBufferCallback` 中，处理后同步返回的视频 `CMSampleBufferRef`。如果用户传正常数据，则内部会自行释放 `CFRelease(CMSampleBufferRef)` 对象，上层不需要再考虑释放问题。

```
[RCRTCEngine sharedInstance].defaultVideoStream.videoSendBufferCallback =
^CMSampleBufferRef _Nullable(BOOL valid, CMSampleBufferRef _Nullable sampleBuffer) {
//直接修改 sampleBuffer 内存数据或者返回一个新生成的 CMSampleBufferRef
};
```

参数	类型	说明
<code>videoSendBufferCallback</code>	<code>RCRTCVideoCMSampleBufferCallback</code>	引擎底部开始视频编码并发送之前会往上层抛一个回调，用户可以修改和调整 <code>CMSampleBufferRef</code> 数据，然后返回一个 <code>CMSampleBufferRef</code> 数据，如果返回空或者没有实现该回调，则会使用默认视频数据传输

在本地预览前处理视频流

应用程序可以在 SDK 本地显示视频流前处理视频流，修改仅作用于本地视频显示，SDK 仍然发送摄像头采集的原始数据。请在 `videoDisplayBufferCallback` 中，处理后同步返回的 `CMSampleBufferRef`。如果用户传正常数据，则内部会自行 `CFRelease CMSampleBufferRef` 对象，上层不需要再考虑释放问题。


```
[RCRTCEngine sharedInstance].defaultVideoStream.videoDisplayBufferCallback =
^CMSampleBufferRef _Nullable(BOOL valid, CMSampleBufferRef _Nullable sampleBuffer) {
//直接修改 sampleBuffer 内存数据或者返回一个新生成的 CMSampleBufferRef
};
```

参数	类型	说明
<code>videoDisplayBufferCallback</code>	<code>RCRTCVideoCMSampleBufferCallback</code>	本地摄像头采集的视频在即将预览前会往上层抛一个视频帧回调，用户可以处理视频帧数据之后然后回传给 RTC，RTC 使用用户处理的视频帧进行预览

处理收到的远端流

应用程序可以在 SDK 收到远端视频流后处理视频流。远端用户流 [RCRTCVideoInputStream](#) 中定义了 [RCRTCVideoInputStreamDelegate](#) 代理，实现此代理后可以将远端用户流中的视频 `CVPixelBufferRef` 数据返回。

```
- (void)willRenderCVPixelBufferRef:(CVPixelBufferRef)ref stream:(RCRTCInputStream *)stream;
```

参数	类型	说明
ref	CVPixelBufferRef	远端视频数据
stream	RCRTCInputStream 	远端视频流

发布自定义流

更新时间:2024-08-30

用户可自定义视频流，通过发布资源方法发布到房间中。远端用户可使用订阅方法订阅此自定义视频流。

1. 创建本地渲染视图。

```
// 创建本地渲染视图
RCRTCVideoView *localFileVideoView = [[RCRTCVideoView alloc] initWithFrame:CGRectMake(0, 0, 100, 100)];
localFileVideoView.fillMode = RCRTCVideoFillModeAspectFit;
localFileVideoView.frameAnimated = NO;
[self.view addSubview:localFileVideoView];
```

2. 创建自定义视频流配置 [RCRTCVideoStreamConfig](#)。

```
// 设置自定义视频流配置
RCRTCVideoStreamConfig *videoConfig = [[RCRTCVideoStreamConfig alloc] init];
videoConfig.videoSizePreset = RCRTCVideoSizePreset720x480;
```

3. 调用 [createFileVideoOutputStream](#) 创建一个自定义视频流 [RCRTCFileVideoOutputStream](#) 对象，并自定义 tag。支持自行选择是否在本端播放音频，是否要用文件中的音频替换麦克风数据。

注意，tag 不能包含 _ 和 RongCloudRTC。

```
// 创建自定义视频流
NSString *path = [[NSBundle mainBundle] pathForResource:@"video_demo1_low"
ofType:@"mp4"];
NSString *tag = @"RongRTCFileVideo";
RCRTCFileVideoOutputStream *fileVideoOutputStream = [[RCRTCEngine sharedInstance]
createFileVideoOutputStream:path
replaceAudio:NO
playback:YES
tag:tag
config:videoConfig];
```

4. 渲染自定义视频流。

```
// 渲染自定义视频流
[fileVideoOutputStream setVideoView:localFileVideoView];
fileVideoOutputStream.delegate = self;
```

5. 使用加入房间成功后返回的 `RCRTCRoom` 对象中的 `RCRTCLocalUser` 中的方法发布。

会议模式下，需要使用 [publishStream](#) 方法

```
// 会议模式下，发布自定义视频流
[[RCRTCEngine sharedInstance].room.localUser publishStream:fileVideoOutputStream
completion:^(BOOL isSuccess, RCRTCCode code) {
}];
```

直播模式下，需要使用 [publishLiveStream](#) 方法

```
// 直播模式下，发布自定义视频流
[[RCRTCEngine sharedInstance].room.localUser publishLiveStream:fileVideoOutputStream
completion:^(BOOL isSuccess, RCRTCCode code, RCRTCLiveInfo * _Nullable liveInfo) {
if (code == RCRTCCodeSuccess) {
}
}];
```

美颜插件

官方美颜插件

更新时间:2024-08-30

提示

从 5.1.4 及之后版本开始，融云 RTCLib SDK 支持美颜和滤镜功能。美颜与滤镜功能以 RongFaceBeautifier 插件库形式提供，并已集成于 SDK 中。

集成使用

将 RongCloudRTC 升级到 5.1.4 及以后版本，在需要美颜功能的文件中导入美颜插件头文件：

```
#import <RongFaceBeautifier/RongFaceBeautifier.h>
```

确保在设置美颜和滤镜之前已经初始化 RCRTCEngine 引擎。

使用 RCRTCBeautyEngine 单例的 setBeautyOption:option: 和 setBeautyFilter: 方法设置美颜参数和滤镜。

示例代码

```
// 获取当前美颜参数
RCRTCBeautyOption *option = [[RCRTCBeautyEngine sharedInstance] getCurrentBeautyOption];
// 修改参数
option.whitenessLevel = 0;
option.smoothLevel = 0;
option.ruddyLevel = 0;
option.brightLevel = 5;
// 设置美颜
[[RCRTCBeautyEngine sharedInstance] setBeautyOption:YES option:option];
// 设置滤镜
[[RCRTCBeautyEngine sharedInstance] setBeautyFilter:RCRTCBeautyFilterEsthetic];
```

设置美颜

```
– (BOOL)setBeautyOption:(BOOL)enable option:(RCRTCBeautyOption *)option;
```

参数	类型	必填	说明
enabled	BOOL	是	是否开启美颜
option	RCRTCBeautyOption 🔗	否	支持开发者设置参数 whitenessLevel (美白)、smoothLevel (磨皮)、ruddyLevel (红润) 和 brightLevel (亮度)，当 enabled 为 NO 时可以传 nil

设置滤镜

```
-(BOOL)setBeautyFilter:(RCRTCBeautyFilter)filter;
```

参数	类型	必填	说明
filter	RCRTCBeautyFilter	是	滤镜类型，支持设置唯美、清新和浪漫三种基础滤镜。

重置参数

因为 RCRTCBeautyEngine 类是单例对象，所以会保留开发者设置的美颜参数或滤镜类型，在需要重置美颜的时候（例如重复进入房间时）需要调用 reset 方法重置所有美颜参数和滤镜。

```
[[RCRTCBeautyEngine sharedInstance] reset];
```

相芯美颜插件

提示

从 5.2.5 及之后版本开始，融云 RTCLib SDK 支持相芯美颜插件。使用相芯美颜需要购买相关授权，详情请咨询融云商务。

插件已经对美颜、滤镜、美形和美肤等功能接口进行了封装，后续版本将继续完善。

当前不支持同时使用官方美颜和相芯美颜插件。

插件对应的相芯 SDK 的版本号为 8.3.0。暂不支持加载用户自定义的 bundle。

集成相芯美颜插件

通过 CocoaPods 集成，请在 Podfile 文件中添加以下内容：

```
pod 'RongFUFaceBeautifier', '~> x.y.z'
```

提示

x.y.z 代表当前 RTCLib 的具体版本号（插件版本号需要与 RTCLib 版本号一致）。您可以在融云下载页或 CocoaPods 仓库进行查询。

导入头文件

将 RongCloudRTC 升级到 5.2.5 及以后版本，在需要美颜功能的文件中导入美颜插件头文件：


```
#import <RongFUFaceBeautifier/RongFUFaceBeautifier.h>
```

初始化

应用运行期间只调用一次即可。

```
// auth_package 为相芯美颜授权文件  
[[RCRTCFCUBeautyEngine sharedInstance] registerWithAuthPackage:&auth_package  
authSize:sizeof(auth_package)];
```

美颜开关

打开美颜开关后设置的美颜效果才会生效；关闭开关美颜会失效。

```
[[RCRTCFCUBeautyEngine sharedInstance] setBeautyEnable:YES];
```

滤镜设置

滤镜可设置自然、白亮、冷色调、粉嫩、黑白、暖色调、蜜桃等效果。参数分两种：滤镜类别、滤镜级别。

```
[[RCRTCFCUBeautyEngine sharedInstance] setFilterWithName:name level:level];
```

美型设置

美型可设置瘦脸程度、V脸程度、窄脸程度、小脸程度、瘦鼻程度、嘴巴调整程度、开眼角强度、眼睛间距、鼻子长度、人中长度、微笑嘴角强度、瘦颧骨强度等效果。下面是设置瘦脸程度的接口调用示例，其他接口请参考插件头文件。

```
// 设置瘦脸程度  
[[RCRTCFCUBeautyEngine sharedInstance] setCheekThinningIntensity:level];
```

美肤设置

可设置磨皮、美白、红润、锐化、亮眼、美牙、去除黑眼圈、去除法令纹强度等。下面是设置磨皮强度的接口调用示例，其他接口请参考插件头文件。

```
// 设置磨皮强度  
[[RCRTCFCUBeautyEngine sharedInstance] setBlurIntensity:level];
```

重置所有美颜效果

执行此方法后，会重置所有美颜美形效果，并释放素材和模型资源，但不会关闭美颜引擎。下次启用美颜需要重新调用 setBeautyEnable: 方法。

```
[[RCRTCFUBeautyEngine sharedInstance] reset];
```

水印处理

更新时间:2024-08-30

融云支持在视频流上添加水印。添加水印有两种控制方式，本文仅介绍方案一：

- 方案一：使用客户端 SDK 提供的 `setWaterMark:position:` 方法添加图片水印。客户端发布的视频流即带有图片水印，因此订阅分流或合流的直播观众均会看到带水印的视频流。
- 方案二：使用服务端 API 的 `/rtc/mcu/config` 接口，在服务端处理，添加时间戳水印、文字水印或图片水印。这种方式支持为单人视频流或合流视频添加水印，但只有订阅合流的观众可看到带水印的视频流。本文不介绍服务端的处理方案，如有需要，请参见服务端文档[直播合流](#)。

方案一适用于实现 App 客户端用户自主添加个性化水印；方案二更适用于由 App 添加统一风格的水印。

客户端与服务端添加的水印相互独立。如果同时使用，则订阅合流的观众可能会看到水印叠加。

设置水印

提示

SDK 从 5.1.16 版本开始，提供内置水印接口。

RongCloudRTC 内置了设置水印的接口，通过调用 `RCRTCVideoOutputStream` 的 `setWaterMark:position:` 方法即可实现为相机/自定义文件/共享桌面采集到的视频流添加水印的功能。每一道视频流水印设置是独立的。

```
- (BOOL)setWaterMark:(nullable UIImage *)image position:(CGRect)position;
```

参数	类型	必填	说明
image	UIImage	否	水印图片，传入 nil 时则清除水印。
position	CGRect	是	水印的位置和尺寸参数。注意：参数取值范围 0 ~ 1，SDK 内部会根据视频分辨率计算水印实际的像素位置和尺寸。

示例代码

```
// 添加水印
// x, y, width 参数取值范围是 0 ~ 1
// 不用设置 height，内部会根据水印图片的宽高比自动计算一个合适的高度。
UIImage *waterMarkImage = [UIImage imageNamed:@"chat_water_mark"];
[[RCRTCEngine sharedInstance].defaultVideoStream setWaterMark:waterMarkImage
position:CGRectMake(0.1, 0.1, 0.14, 0)];

// 清除水印
[[RCRTCEngine sharedInstance].defaultVideoStream setWaterMark:nil
position:CGRectZero];
```

对接第三方插件

更新时间:2024-08-30

提示

在基于 SDK 建立通话基础上，使用 GPUImage 库可以实现本地视频滤镜和水印等视频纹理的渲染处理。

对接三方插件（实现美颜、水印）（ $\geq 5.2.3$ ）

在 [RCRTCVideoOutputStream](#) 的父类 [RCRTCVideoOutputStream](#) 中的属性 `streamEventDelegate` 中 `outputVideoStream:captureVideoFrame:` 方法，会同步返回采集的视频帧 [RCRTCVideoFrame](#)，开发者可以直接对该视频帧的内存数据 `pixelBuffer` 进行美颜、水印处理。

1. 设置代理。

```
[RCRTCEngine sharedInstance].defaultVideoStream.streamEventDelegate = self;
```

2. 在 `outputVideoStream:captureVideoFrame:` 方法中对该视频帧的内存数据 `pixelBuffer` 进行美颜、水印处理。

```
/*!
采集视频数据回调

@param videoFrame 采集的视频数据
@param stream 采集的视频帧数据所属接收到的流
@discussion
采集的视频数据，如果修改该视频帧，会影响本地渲染和编码发送的视频帧数据，不支持修改数据格式

@remarks 代理
added from 5.2.3
*/
- (void)outputVideoStream:(RCRTCVideoOutputStream *)stream captureVideoFrame:(nullable RCRTCVideoFrame *)videoFrame {
//美颜、水印处理 videoFrame 中的 pixelBuffer
}
```

输入参数	类型	说明
stream	RCRTCVideoOutputStream	当前流对象
videoFrame	RCRTCVideoFrame *	当前采集的视频帧

对接三方插件（实现美颜、水印）（< 5.2.3）

在 [RCRTC CameraOutputStream](#) 类中的 `videoSendBufferCallback` 中，处理后同步返回的视频 `CMSampleBufferRef` 会作用于本地视频显示和发送的视频数据。如果用户传正常数据，则内部会自行释放 `CFRelease(CMSampleBufferRef)` 对象，上层不需要再考虑释放问题。

参考 [RTC Quick Demo](#) ([GitHub](#) · [Gitee](#)) 下 `/RCRTCQuickDemo/Living/GPUImage/GPUImageHandle.m` 对美颜的处理。

```
@property (nonatomic, copy, nullable) RCRTCVideoCMSampleBufferCallback videoSendBufferCallback;
```

输入参数

参数	类型	说明
<code>videoSendBufferCallback</code>	<code>RCRTCVideoCMSampleBufferCallback</code>	引擎底部开始视频编码并发送之前会往上层抛一个回调，用户可以修改和调整 <code>CMSampleBufferRef</code> 数据，然后返回一个 <code>CMSampleBufferRef</code> 数据，实现美颜与水印效果。如果返回空或者没有实现该回调，则会使用默认视频数据传输

示例代码

```
[RCRTCEngine sharedInstance].defaultVideoStream.videoSendBufferCallback =  
^CMSampleBufferRef _Nullable(BOOL valid, CMSampleBufferRef _Nullable sampleBuffer) {  
    if ( !strongSelf.openBeauty&&!strongSelf.openWaterMark ) {  
        return sampleBuffer;  
    }  
    // 返回处理后的 CMSampleBufferRef 数据  
    CMSampleBufferRef processedSampleBuffer = [strongSelf.gpuImageHandler onGPUFilterSource:sampleBuffer];  
    return processedSampleBuffer ?: sampleBuffer;  
};
```

网络质量探测

更新时间:2024-08-30

自 5.1.17 起, RTCLib SDK 提供了一个 `startRTCProbeTest:` 方法, 支持用户在加入房间前进行网络质量探测, 然后通过代理回调将当前网络质量的相关数据, 包括往返时延、上下行丢包率、上下行网络带宽等数据返回给上层应用。

提示

请在加入房间前完成网络质量探测。SDK 不支持在音视频通话过程中进行网络质量探测。

设置 RTC 网络探测回调事件

通过 `RCRTCEngine` 设置 `probeTestDelegate` 代理, 开始探测后网络质量探测结果在 `RCRTCProbeTestDelegate` 协议方法里回调。

示例代码:

```
//设置探测结果回调代理
[[RCRTCEngine sharedInstance] setProbeTestDelegate:self];

//实现探测结果回调协议
#pragma mark - RCRTCProbeTestDelegate

/*!
汇报探测状态数据
*/
- (void)didReportProbeForms:(NSArray <RCRTCProbeStatusForm *>*)forms {
//开始探测后, 每 2s 回调一次报表数据, 持续 30s。
NSLog(@"%@@", forms);
}

/*!
探测完成
*/
- (void)didRTCProbeComplete {
NSLog(@"探测完成");
}

/*!
探测中断
*/
- (void)didRTCProbeInterrupt:(RCRTCCode)errorCode {
NSLog(@"探测中断");
}
```

开启 RTC 网络探测

加入房间前，调用 `startRTCProbeTest`：开启网络质量探测。开启探测成功后，会通过 `RCRTCProbeTestDelegate` 协议方法回调来反馈探测结果。每隔约 2 秒回调一次上下行网络的带宽、丢包、网络抖动和往返时延等数据。

探测总时长约为 30 秒，30 秒后会自动停止探测并调用 `didRTCProbeComplete` 回调方法通知上层应用探测结束。

提示

在探测自动结束 `didRTCProbeComplete` 前，如调用 `joinRoom` 加房间操作会打断当前正在进行的 RTC 网络探测。

示例代码：

```
[[RCRTCEngine sharedInstance] startRTCProbeTest:^(BOOL isSuccess, RCRTCCode code) {
if (isSuccess) {
NSLog(@"调用成功 code:%@",@(code));
}
else {
NSLog(@"调用失败 code:%@",@(code));
}
}];
```

停止 RTC 网络探测

开启探测成功约 30 秒后，会自动停止探测，如果不想等待自动结束，可主动调用 `stopRTCProbeTest` 方法强制停止探测。

```
[[RCRTCEngine sharedInstance] stopRTCProbeTest:^(BOOL isSuccess, RCRTCCode code) {
if (isSuccess) {
NSLog(@"停止成功 code:%@",@(code));
}
else {
NSLog(@"停止失败 code:%@",@(code));
}
}];
```


屏幕共享

更新时间:2024-08-30

融云 RTCLib SDK 为开发者提供了屏幕共享插件，屏幕共享功能以插件库形式提供，开发者可自行选择是否向项目中集成 RongRTCReplayKitExt 插件库来实现屏幕共享功能。RongRTCReplayKitExt 插件用于拓展 target 上使用。

提示

- 屏幕共享功能在 RongRTCLib 5.1.8.1 及以后支持。
- 屏幕共享库 RongRTCReplayKitExt 依赖的 RongRTCLib 版本必须大于等于 5.1.8.1

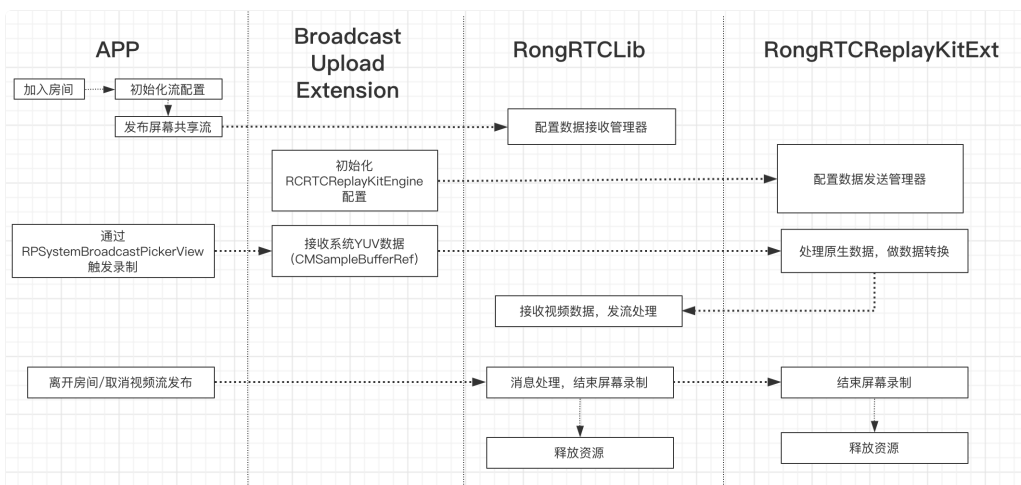
环境要求

- iOS: iOS12 及以上。
- RongRTCLib: 5.1.8.1 及以上。
- RongRTCReplayKitExt: 5.1.8.1 及以上。

实现方式

我们可以通过苹果的 Extension 来实现屏幕共享技术。这里我们将通过使用 iOS11 之后新增的系统级别的录屏能力，来实现录制自身 App 以外，手机屏幕内容的效果（受 iOS 系统的 ReplayKit 库限制，iOS12 前在 App 中调用 Extension 启动屏幕共享时，只能作用于 App 内，如果退出 App 则无法得到屏幕内容）。

iOS 端的屏幕共享是通过在 Extension 中使用 RongRTCReplayKitExt 框架实现录制屏幕流的接收，然后将屏幕共享流传输到 RTCLib 实现的。由于 Apple 不支持 Extension 进程与主 app 进程通信，因此您需要为屏幕共享流单独创建一个进程。



实现屏幕共享的主要步骤如下：

1. 创建 App Group，并在 XCode 中进行配置。目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。

2. 创建一个 Broadcast Upload Extension 用于开启屏幕共享的进程。
3. 宿主 APP 开启屏幕共享并发布屏幕共享流
4. 屏幕共享 extension 收到消息，初始化 RongRTCReplayKitExt 库，接收系统录制数据。

集成说明

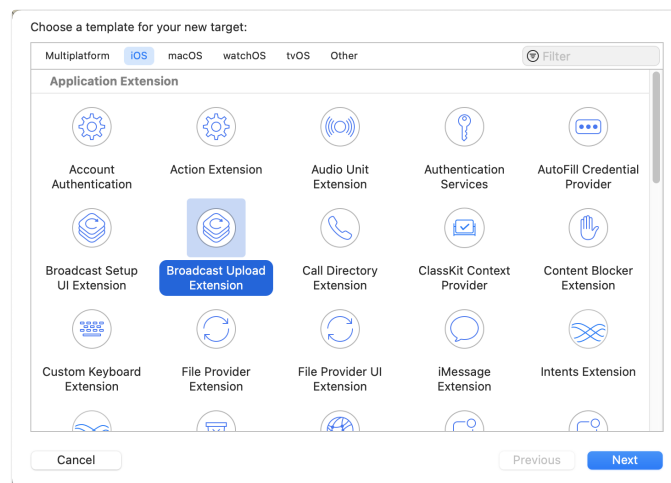
创建 App Group:

使用您的帐号登录 [苹果控制台](#)，进行以下操作，注意完成后需要重新下载对应的 Provisioning Profile。

1. 单击【Certificates, IDs & Profiles】。
2. 在右侧的界面中单击加号。
3. 选择【App Groups】，单击【Continue】。
4. 在弹出的表单中填写 Description 和 Identifier, 其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击【Continue】。
5. 回到 Identifier 页面，左上边的菜单中选择【App IDs】，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中【App Groups】并单击【Edit】。
7. 在弹出的表单中选择您之前创建的 App Group，单击【Continue】返回编辑页，单击【Save】保存。
8. 重新下载 Provisioning Profile 并配置到 XCode 中。

创建 Broadcast Upload Extension:

1. 在 Xcode 菜单依次单击【File】、【New】、【Target...】，选择【Broadcast Upload Extension】，创建类型为 Broadcast Upload Extension 的新 target。



2. 在弹出的对话框中填写相关信息，不用勾选【Include UI Extension】，单击【Finish】完成创建。
3. 在您的 屏幕共享 Target - Build Phases - Compile Sources 中添加您可能在屏幕共享 Extension 里引用类的 .m 文件名。
4. 受限于 RongIMLib 库中的默认 2 分钟断开连接的限制，需要修改如下 pod 路径下的 .plist 配置文件:

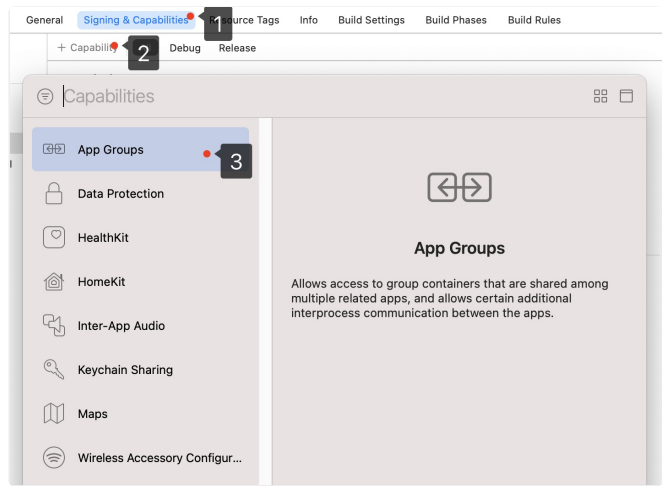
您的工程名/Pods/RongCloudIM/IMLibCore/RConfig.plist

在此文件中添加:

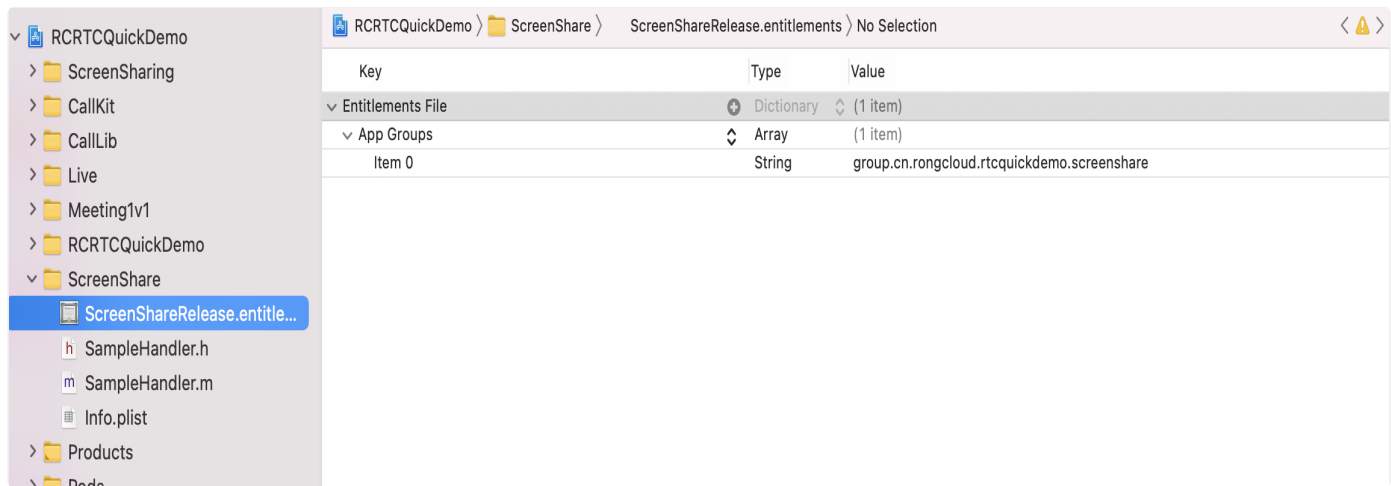
```
<key>Connection</key>
<dict>
<key>ForceKeepAlive</key>
<true/>
</dict>
```

其中: Connection 和 ForceKeepAlive 类型为 Key 值, ture 的类型为 Bool。

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



操作完成后, 会在文件列表中生成一个以您自己创建的 Target 命名的 Target名.entitlements 的文件, 如下图所示, 选中该文件并单击 + 号填写上述步骤中的 App Group 即可。



6. 选中宿主 App 的 Target, 并按照上述步骤对宿主 App 的 Target 做同样的处理。

导入 SDK

SDK 支持通过 CocoaPods 或手动依赖 Framework 的方式导入到项目中。

CocoaPods 自动导入 (推荐)

1. 请运行以下命令更新本地的 CocoaPods 仓库列表:

```
pod repo update
```

2. 请在 podfile 中添加如下内容:

```
target '主进程APP Target' do
  use_frameworks!
  pod 'RongCloudRTC/RongRTCLib', '~> 5.1.8.1'
end

target '您的屏幕共享的 target' do
  use_frameworks!
  pod 'RongCloudRTC/RongRTCReplayKitExt', '~> 5.1.8.1'
end
```

3. 请在终端中运行以下命令:

```
pod install
```

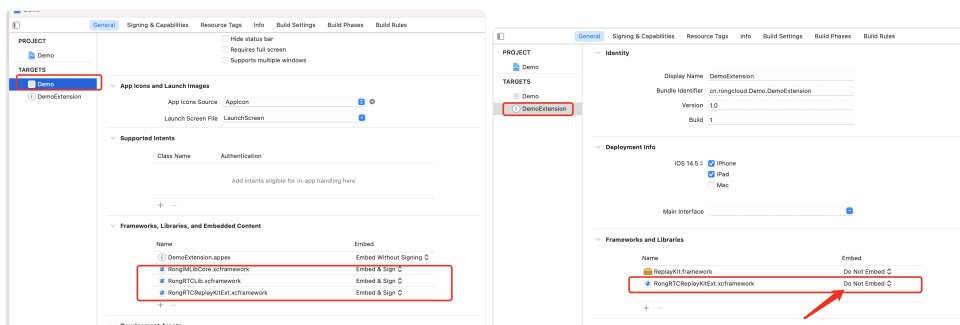
4. pod install 完成后, 会自动导入指定版本的 RongIMLib, CocoaPods 会在您的工程根目录下生成一个 .xcworkspace 文件, 打开即可。

手动依赖 Framework

1. 在融云官网下载实时音视频库进行集成。[下载地址](#)

2. 在主进程 APP Target 中 导入

RongRTCLib.xcframework、RongIMLibCore.xcframework、RongRTCReplayKitExt.xcframework。在您的屏幕共享的 target 中导入 RongRTCReplayKitExt.xcframework。



3. 权限配置

1. 音视频通话需要用到摄像头和麦克风权限, 请在工程的 info.plist 中添加如下键值:

- Privacy - Microphone Usage Description
- Privacy - Camera Usage Description

2. 请将工程中 Target -> Signing & Capabilities -> Background Modes 如下内容勾选：

- Audio, AirPlay, and Picture in Picture

共享实现

iOS 系统上的跨应用屏幕分享，需要增加 Extension 录屏进程以配合宿主 App 进程进行推流。Extension 录屏进程由系统在需要录屏的时候创建，并负责接收系统采集的 CMSampleBufferRef 数据。

宿主 App

宿主 App 在加入房间后在需要屏幕共享时，通过点击录制按钮通知屏幕共享 target、并发布屏幕共享流。

添加系统录制按钮

提示

此按钮需 iOS12 及以上可用。

- 示例代码：

```
#import <ReplayKit/ReplayKit.h>

// 添加录制按钮
RPSystemBroadcastPickerView *systemBroadcastPickerView = [[RPSystemBroadcastPickerView alloc]
initWithFrame:CGRectMake(0, 64, 50, 80)];
systemBroadcastPickerView.preferredExtension = @"您的屏幕共享 target 的 Bundle Identifier";
systemBroadcastPickerView.backgroundColor = [UIColor colorWithRed:53.0/255.0 green:129.0/255.0
blue:242.0/255.0 alpha:1.0];
systemBroadcastPickerView.showsMicrophoneButton = NO;
self.navigationItem.rightBarButtonItem = [[UIBarButtonItem
alloc] initWithCustomView:systemBroadcastPickerView];
```

初始化并连接

初始化 SDK 并连接融云服务器。

- [获取 App Key](#)
- [获取 Token](#)

示例代码：

```

[[RCCoreClient sharedCoreClient] initWithAppKey:@"您的AppKey"];
// 连接 IM
[[RCCoreClient sharedCoreClient] connectWithToken:@"您的token" dbOpened:^(RCDBErrorCode code) {
} success:^(NSString *userId) {
// 可以在此处加入房间
} error:^(RCConnectErrorCode errorCode) {
}]];

```

监听屏幕共享流的发布与取消

在发布屏幕共享流之前，可以先设置 RCRTCEngine 的代理，通过对应的代理方法来获取发布与取消屏幕共享流的时机。

- 示例代码：

```

[RCRTCEngine sharedInstance].delegate = self;

// 屏幕共享拓展 结束消息回调 Added from 5.1.8
- (void)screenShareExtentionFinished {
}

// 屏幕共享拓展 开始消息回调 Added from 5.2.0
- (void)screenShareExtentionStarted {
}

```

加入房间

连接 SDK 成功后，配置房间信息并加入房间。

- 示例代码：

```

[[RCRTCEngine sharedInstance] joinRoom:@"您的房间号"
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {
room.delegate = self;
// 发布资源
[self publishScreenStream];
}]];

```

发布资源

加入房间成功后可以发布 RCRTCScreenShareOutputStream 流。直播与会议场景发布流的方法不同，请注意区分。

- 获取 RCRTCScreenShareOutputStream 方式

```

/*!
获取屏幕共享所需流

@param groupId groupId 苹果开发者账号后台申请

@remarks RCRTCEngine:RCRTCScreenShareOutputStream get 接口
added from 5.1.8.1
*/
- (RCRTCScreenShareOutputStream *)getDefaultVideoStreamWith:(NSString *)groupId;

```

- 示例代码：

在会议场景下，请使用 `publishStream:completion` 方法。

```

RCRTCScreenShareOutputStream *videoOutputStream = [[RCRTCEngine sharedInstance]
getScreenShareVideoStreamWithGroupId:@"您的屏幕共享 Extension 的 Group ID"];

RCRTCVideoStreamConfig *videoConfig = videoOutputStream.videoConfig;
videoConfig.videoSizePreset = RCRTCVideoSizePreset1280x720;
videoConfig.videoFps = RCRTCVideoFPS24;
[videoOutputStream setVideoConfig:videoConfig];

[self.room.localUser publishStream:videoOutputStream
completion:^(BOOL isSuccess, RCRTCCode desc) {}];

```

在直播场景下，请使用 `publishLiveStream:completion` 方法。

```

RCRTCScreenShareOutputStream *videoOutputStream = [[RCRTCEngine sharedInstance]
getScreenShareVideoStreamWithGroupId:@"您的屏幕共享 Extension 的 Group ID"];

RCRTCVideoStreamConfig *videoConfig = videoOutputStream.videoConfig;
videoConfig.videoSizePreset = RCRTCVideoSizePreset1280x720;
videoConfig.videoFps = RCRTCVideoFPS24;
[videoOutputStream setVideoConfig:videoConfig];

[[self.room.localUser publishLiveStream:videoOutputStream
completion:^(BOOL isSuccess, RCRTCCode code, RCRTCLiveInfo *_Nullable liveInfo) {
}];

```

屏幕共享 Extension

屏幕共享 target 在收到录制通知后，需要初始化 `RongRTCReplayKitExt` 库，并接收系统源数据。

SampleHandler 文件处理

在系统对应函数里这里初始化 `sdk`，接收系统 `CMSampleBufferRef` 源数据。

- 示例代码：

```
#import "SampleHandler.h"
#import <RongRTCReplayKitExt/RongRTCReplayKitExt.h>

static NSString *const ScreenShareGroupID = @"您的屏幕共享 Extension 的 Group ID";

@implementation SampleHandler

- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *, NSObject *> *)setupInfo {
//初始化sdk
[[RCRTCReplayKitEngine sharedInstance] setupWithAppGroup:ScreenShareGroupID delegate:self];
}

- (void)broadcastPaused {
}

- (void)broadcastResumed {
}

- (void)broadcastFinished {
//结束处理
[[RCRTCReplayKitEngine sharedInstance] broadcastFinished];
}

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer
withType:(RPSampleBufferType)sampleBufferType API_AVAILABLE(ios(10.0)) {
switch (sampleBufferType) {
case RPSampleBufferTypeVideo:
//接收系统数据回调
[[RCRTCReplayKitEngine sharedInstance] sendSampleBuffer:sampleBuffer
withType:RPSampleBufferTypeVideo];
break;
case RPSampleBufferTypeAudioApp:
// Handle audio sample buffer for app audio
break;
case RPSampleBufferTypeAudioMic:
// Handle audio sample buffer for mic audio
break;

default:
break;
}
}
```

③ 提示

Broadcast Upload Extension 的内存使用限制为 50 MB，请确保屏幕共享的 Extension 内存使用不超过 50 MB。可以参考 [RTC Quick Demo](#)（[GitHub](#) · [Gitee](#)）内屏幕共享的实现。

自定义信令

创建信令

更新时间:2024-08-30

创建信令类需继承 RongIMLib 库中 RCMMessageContent 类，并实现以下几个方法：

- 示例代码：

```
- (NSData *)encode {
    NSData *data = [NSJSONSerialization dataWithJSONObject:self.whiteBoardDict
    options:NSUTF8WritingPrettyPrinted
    error:nil];
    return data;
}

- (void)decodeWithData:(NSData *)data {
    if (data) {
        self.messageDict = [NSJSONSerialization JSONObjectWithData:data
        options:NSJSONReadingMutableContainers
        error:nil];
    }
}

+ (RCMessagePersistent)persistentFlag {
    // 必须为 MessagePersistent_STATUS 的状态消息，否则会导致消息发送失败
    return MessagePersistent_STATUS;
}

+ (NSString *)getObjectname {
    return @"唯一的消息 Key 值";
}
```

注册消息

在使用前需要注册信令类，否则无法发送和接收。

- 示例代码：

```
[[RCCoreClient sharedCoreClient] registerMessageType:自定义消息类.class];
```

发送消息

[RCRTCRoom](#) 中提供了发送信令的方法，发送时已经在房间内的用户可以接收到此信令。

```
- (RCMessage *)sendRTCMessage:(RCMessageContent *)content
success:(void (^)(long messageId))successBlock
error:(void (^)(RCErrorCode nErrorCode, long messageId))errorBlock;
```

- 输入参数：

参数	类型	说明
content	RCMessageContent	信令的内容
successBlock	(void (^)(long messageId))	信令发送成功的回调 [messageId:消息的 ID]
errorBlock	(void (^)(RCErrorCode nErrorCode, long messageId))	信令发送失败的回调 [nErrorCode:发送失败的错误码 messageId:信令的 ID]

- 返回参数：

类型	说明
RCMessage	发送的消息实体

- 示例代码：

```
[self.room sendRTCMessage:message
success:^(long messageId) {}
error:^(RCErrorCode nErrorCode, long messageId) {
}];
```

接收消息

实现 [RCRTCRoomEventDelegate](#) 中的如下方法后，可通过此代理方法接收信令。

```
- (void)didReceiveMessage:(RCMessage *)message;
```

- 返回参数：

类型	说明
RCMessage	接收到的消息实体，参考 RongIMLib 中 RCMessage

直播数据统计

流状态数据上报

更新时间:2024-08-30

音视频库会每秒一次上报直播流的详细数据，上层依据此数据可进行提示，状态判断等处理。SDK 提供了 [RCRTCStatusReportDelegate](#) 代理可以取得直播流的详细数据 [RCRTCStatusForm](#)。

```
– (void)didReportStatusForm:(RCRTCStatusForm*) form;
```

RCRTCStatisticalForm 说明

属性	类型	说明
cpuUsage	float	CPU使用率，取值: 0 ~ 100
cpuUsageOfOS	float	系统CPU使用率，取值: 0 ~ 100
totalRecvBitRate	float	接收的所有下行码率，单位: kbps
totalSendBitRate	float	发送的所有上行码率，单位: kbps
networkType	NSString	当前使用的网络类型
rtt	NSInteger	往返时间，单位: 毫秒
ipAddress	NSString	IP地址
availableReceiveBandwidth	float	可接收带宽
availableSendBandwidth	float	可发送带宽
sendStats	NSArray<RCRTCStreamStat * >	发送轨道数据，见下方 RCRTCStreamStat 说明
recvStats	NSArray<RCRTCStreamStat * >	接收轨道数据，见下方 RCRTCStreamStat 说明

• RCRTCStreamStat 说明

属性	类型	说明
trackId	NSString	音/视频流ID
audioLevel	NSInteger	音频流中的音量，视频流中为0
bitRate	float	码率，单位: kbps
frameHeight	NSInteger	视频帧高度，音频流中为 -1
frameWidth	NSInteger	视频帧宽度，音频流中为 -1
mediaType	NSString	媒体类型，音频为: audio，视频为: video
packetLoss	float	发送丢包率，取值: 0 ~ 1
rtt	NSInteger	往返时间，单位: 毫秒

属性	类型	说明
frameRate	NSInteger	视频帧率，音频流中为 -1
jitterReceived	NSInteger	网络抖动，单位: 毫秒
codecName	NSString	音/视频编解码器
state	NSInteger	流是否可用状态，随麦克风开关变化，1 可用，2 不可用，-1 无法确定

直播合流音量上报

从 SDK 5.1.11 版本开始，SDK 支持在直播模式下实时上报音频合流的声音状态，您可单独获取每个主播的音量。

例如当前音频合流由 A B C 三个主播音频流产生，此时 A 在发声，SDK 会通过 didReportLiveAudioStatus：回调 audioStatus 数组，里面包含 A 主播的 audioLevel。

```
– (void)didReportLiveAudioStatus:(NSArray<RCRTCLiveAudioStatus *> *)audioStatus;
```

参数	类型	说明
audioStatus	NSArray<RCRTCLiveAudioStatus *> *	汇报音频合流数组，见下方 RCRTCLiveAduioStatus 说明

RCRTCLiveAduioStatus 说明：

属性	类型	说明
userId	NSString	用户Id
streamId	NSString	流Id
audioLevel	NSInteger	音量大小，0 - 9表示音量高低

媒体补充增强信息 (SEI)

更新时间:2024-08-30

提示

RTCLib SDK 从 5.3.0 版本开始正式支持媒体补充增强信息 (SEI)。

在音视频流媒体应用中，可以使用流 SEI (Supplemental Enhancement Information, 媒体补充增强信息) 将文本信息与音视频内容打包在一起，从推流端推出，并从拉流端接收，以此实现文本数据与音视频内容的精准同步的目的。一般可用于视频画面的精准布局、远端歌词同步、直播答题等应用场景。

自 5.3.0 起，RTCLib SDK 提供了 sendSEI 方法，支持在发布音视频时添加媒体补充增强信息 (SEI)。

SEI 开关

SDK 支持在加入房间后动态开启或者关闭发送 SEI 的能力。加入房间后，通过调用 RCRTCLocalUser.h 的 setEnableSEI 方法开启或者关闭发送 SEI 能力。该方法支持在发布流前后调用。

提示

会议模式下所有参会用户均可以发流，直播模式下主播角色用户可以发流。注意，直播模式下的观众角色不发布资源，因此开启发送 SEI 能力对于观众角色无效。观众角色调用该 API 后会返回对应的错误码。

```
// 加入房间后，通过 RCRTCRoom 对象持有的 RCRTCLocalUser 对象作为调用入口。  
// 开启 SEI 能力  
[self.room.localUser setEnableSEI:YES completion:^(BOOL isSuccess, RCRTCCode errCode) {  
NSLog(@"isSuccess:%@, errCode:%@",@(isSuccess),@(errCode));  
}];
```

发送 SEI 数据

用户如果在发布流数据之前已经开启 SEI 开关，通过 RCRTCLocalUser.h 头文件的 sendSEI 方法发送 SEI 数据。发送的结果通过返回值确定，不满足发送条件会返回对应的错误码。

提示

考虑到有可能因为网络问题导致丢失 SEI 数据帧，可以在频率限制范围内多次发送 SEI 信息。频率限制：1 秒钟不要超过 30 次。SEI 数据长度限制为 4096 字节。

```

//用户自定义数据
NSString *seiJson = @"xxx";
RCRTCCode result = [self.room.localUser sendSEI:seiJson];
if (result != RCRTCCodeSuccess) {
NSLog(@"发送失败:%@",@(code));
}

```

接收 SEI 数据

用户订阅房间内的流后，可以接收当前房间的 SEI 数据。

接收房间内其他用户的 SEI 数据

在会议模式下，订阅房间内远端用户发布的视频流之后，可以使用 `RCRTCRoomEventDelegate` 的方法接收当前房间的 SEI 数据。在直播模式下，如果进行同房间连麦，主播角色用户之间需要互相订阅。如果需要接收其他主播用户的 SEI 数据，可使用 `RCRTCRoomEventDelegate` 的 `didReceiveSEI:userId` 方法。

```

#pragma mark - RCRTCRoomEventDelegate
/*!
接收到远端用户发送的 SEI 通知

@param SEI sei 数据
@param userId 用户id
@discussion 监听远端用户发送的 SEI 内容，通过 userId 区分。
Added from 5.3.0

@remarks 代理
*/
- (void)didReceiveSEI:(NSString *)SEI userId:(NSString *)userId {
NSLog(@"userId:%@ didReceiveSEI:%@", userId, SEI);
}

```

- `didReceiveSEI:userId` 接收当前房间中指定用户发送的 SEI 数据。会议模式下，需要接收指定参会用户的 SEI 数据。直播模式下，主播角色用户之间如果进行连麦，则需要互相订阅，此时需要接收其他主播用户的 SEI 数据。
- `didReceiveLiveStreamSEI`（直播模式专用）接收当前直播房间中直播合流的 SEI 数据。直播模式下，观众一般订阅合流，因此需要接收的是合流的 SEI 内容。

直播模式下支持跨房间连麦。如果当前房间与其他房间连麦，用户可以使用 `RCRTCOtherRoomEventDelegate` 的以下方法接收指定主播用户发送的 SEI 数据。

- `room:didReceiveSEI:userId`（直播模式专用）接收当前直播房间中或跨房间连麦副房间中指定主播角色用户发布的 SEI 数据。

接收直播合流的 SEI 数据

直播场景下，观众身份的用户一般订阅合流（liveStreams），可以通过 `RCRTCRoomEventDelegate` 的 `didReceiveLiveStreamSEI` 方法接收合流的 SEI 内容。如果多个远端主播发送 SEI，可以通过此回调统一接收数据。

接收直播合流的 SEI 数据

RCRTCRoomEventDelegate 的 `didReceiveLiveStreamSEI:(NSString *)SEI` 方法专用于直播模式。直播模式下，观众一般订阅合流（liveStreams），因此需要接收的是接收当前直播房间中直播合流的 SEI 数据。

```
/*!
观众接收到合流 SEI 通知

@param SEI sei 数据
@discussion 观众角色订阅 liveStreams 后，该回调会接收以下两种类型的数据。
1.MCU server 会主动通过该接口回调主播合流布局的信息 {"mcuRoomState":"xxx"}
2.如果远端主播有发送 SEI，可以通过此回调接收数据。
Added from 5.3.0

@remarks 代理
*/
- (void)didReceiveLiveStreamSEI:(NSString *)SEI {
NSLog(@"liveStreamSEI:%@", SEI);
}
```

提示

`didReceiveLiveStreamSEI` 不仅接收远端主播发送 SEI 数据，还会接收融云 MCU 服务器发送的主播合流布局的信息，格式为：`{"mcuRoomState":"xxx"}`。

接收副房间内远端用户的 SEI 数据

直播模式下支持跨房间连麦。如果当前房间与其他房间建立跨房间连麦，用户可以使用副房间事件代理 `RCRTCOtherRoomEventDelegate` 的 `room:didReceiveSEI:userId` 方法接收指定主播用户发送的 SEI 数据。

```
#pragma mark - RCRTCOtherRoomEventDelegate

/*!
接收到副房间内远端用户发送的 SEI 通知

@param room 事件所在房间
@param SEI sei 数据
@param userId 用户id
@discussion 监听远端用户发送的 SEI 内容，通过 userId 区分。
Added from 5.3.0

@remarks 代理
*/
- (void)room:(RCRTCBaseRoom *)room didReceiveSEI:(NSString *)SEI userId:(NSString *)userId {
NSLog(@"room:%@ userId:%@ didReceiveSEI:%@", room, userId, SEI);
}
```

接收 CDN 流的 SEI 信息

- 如果您的观众端订阅房间内的融云 CDN 流，请参见[融云 CDN 插件](#)。
- 如果您的观众端使用 `RCRTCMediaPlayer` 直接播放 CDN 流，请参见[CDN 播放器](#)。

3.X 升级到 5.X

更新时间:2024-08-30

本文描述 RTCLib SDK 会议场景的升级步骤。

升级概述

RTCLib SDK 5.X 对比 3.X，适配了 iPhone 新机型与新系统，功能更丰富，更稳定，并在之前版本上修复了大量问题，建议尽早升级至新版 RTCLib SDK。

前置条件

- 5.X RTCLib SDK 依赖 RongIMLibCore 请您确保已将 RongIMLibCore 升级至 5.X。
- RongIMLib SDK 5.X 做了拆分，请遵照 IMLib 的升级文档先将使用对应功能的 SDK 进行升级集成。
- 已遵照 IMLib 升级要求将您的 Cocoapod 版本升级到 1.10.0 以上。

手动集成

1. 将原先 SDK 依赖的系统库全部去掉，如果 App 或者其他 SDK 有依赖的系统库除外

- AssetsLibrary
- MapKit
- ImageIO
- Security
- SystemConfiguration
- QuartzCore
- AVFoundation
- GLKit
- OpenGL ES
- CoreGraphics
- CoreLocation
- CoreTelephony
- CoreFoundation
- CoreMedia
- CoreAudio
- CoreVideo
- CFNetwork
- AudioToolbox

- VideoToolBox
 - AVFoundation
 - libc++.tbd
 - libz.tbd
 - libbz2.tbd
 - libiconv.tbd
 - libsqlite3.tbd
2. 将 3.X 的 RongRTCLib.framework、RongIMLib.framework、libopencore-amrnb.a (该库包含在 RongIMLib 软件包中) 删除并替换为 5.X 的 RongRTCLib.xcframework、RongIMLibCore.xcframework
 3. General -> Frameworks, Libraries, and Embedded Binaries 中将 RongXX.framework 的 Embed 设置为 Embed & Sign

pod 集成

1. 在 podfile 中添加如下内容：

```
# pod 'RongRTCLib', '~> 3.2.2' # 注释 3.X 的内容  
pod 'RongCloudRTC/RongRTCLib', '~> x.y.z'
```

2. 请在终端中运行以下命令：

```
pod install
```

如果出现找不到相关版本的问题，可先执行 `pod repo update`，再执行 `pod install`。

接口替换说明

对象命名的改变

RTCLib SDK 从 3.X 升级到 5.X 将对象的命名规则由 RongXXX 变更为 RCXXX，您可以全局针对 RTCLib 的对象进行统一的替换。

例如加入房间的方法名称没有变动，方法返回的实例对象名称由 RongRTCRoom 变更为 RCRTCRoom：

```
- (void)joinRoom:(NSString *)roomId  
completion:(nullable void (^)(RCRTCRoom *_Nullable room, RCRTCCode code))completion;
```

初始化接口

将原有 AppKey 初始化方法进行替换

```
#import <RongIMLibCore/RongIMLibCore.h>

[[RCCoreClient sharedCoreClient] initWithAppKey:@"从控制台申请的 AppKey"];
```

连接接口

将原有连接 IM 方法进行替换

```
// 连接 IM 服务
[[RCCoreClient sharedCoreClient] connectWithToken:@"从您服务器端获取的 Token"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];
```

音视频引擎初始化

将原有 sharedEngine 方法替换为

```
RCRTCEngine *engine = [RCRTCEngine sharedInstance];
```

状态监听接口

将原有的 RongRTCActivityMonitorDelegate 状态监视器代理替换为

```
@property (nonatomic, weak, nullable) id<RCRTCStatusReportDelegate> statusReportDelegate;
```

房间事件代理回调接口

将原有的 RongRTCRoomDelegate 房间事件代理替换为

```
/*!
房间事件代理
*/
@property (nonatomic, weak, nullable) id<RCRTCRoomEventDelegate> delegate;
```

代理中对应的方法变动对照 RCRTCRoomEventDelegate 中的说明替换即可

获取当前房间

将原有的当前加入房间 currentRoom 属性替换为

```
@property (nonatomic, strong, readonly, nullable) RCRTCRoom *room;
```

断线重连设置

断线重连功能默认开启，如果 3.X 有使用 `setReconnectEnable` 设置为 NO；升级后需要在引擎初始化时传入以下配置进行关闭：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];
config.isEnableAutoReconnect = NO;
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

离开房间接口

将原有的 `leaveRoom:completion:` 方法替换为

```
- (void)leaveRoom:(nullable RCRTCOperationCallback)completion;
```

发布与取消发布资源接口

加入房间成功后 3.X 是通过加入房间返回的 `RongRTCRoom` 对象管理本地默认音视频流；5.X 变动为通过加入房间返回的 `RCRTCRoom` 中的 `RCRTCLocalUser` 对象管理本地默认音视频流

```
// 发布本地默认音视频资源
[[RCRTCEngine sharedInstance].room.localUser publishDefaultStream:^(BOOL isSuccess, RCRTCCode desc) {
}];
// 取消发布本地默认音视频资源
[[RCRTCEngine sharedInstance].room.localUser unpublishDefaultStream:^(BOOL isSuccess, RCRTCCode desc) {
}];
```

订阅与取消订阅资源接口

加入房间成功后 3.X 是通过加入房间返回的 `RongRTCRoom` 对象管理对端的音视频流；5.X 变动为通过加入房间返回的 `RCRTCRoom` 中的 `RCRTCLocalUser` 对象管理对端音视频流

```
// 订阅对端音视频资源
[[RCRTCEngine sharedInstance].room.localUser subscribeStream:avStreams
tinyStreams:nil
completion:^(BOOL isSuccess, RCRTCCode desc) {
}];
// 取消订阅对端音视频资源
[[RCRTCEngine sharedInstance].room.localUser unsubscribeStream:streams
completion:^(BOOL isSuccess, RCRTCCode desc) {
}];
```

创建渲染视图

3.X 的创建渲染视图分为本地渲染视图 `RongRTCLocalVideoView` 与远端渲染视图 `RongRTCRemoteVideoView`，5.X 统一将创建渲染视图调整为 `RCRTCVideoView`

```
// 初始化本地渲染视图
RCRTCVideoView *view = [[RCRTCVideoView alloc] initWithFrame:CGRectMake(100, 100, 100, 100)];
```

渲染视频流

将 3.X 的渲染视图接口 setVideoRender: 接口替换为

```
// 渲染本地视图
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoView:@"创建的 RCRTCVideoView 视图"];
// 渲染远端视图
[[RCRTCVideoInputStream *)stream setVideoView:@"创建的 RCRTCVideoView 视图"];
```

摄像头管理

将原有的摄像头相关接口调用替换为

```
// 打开摄像头
[[RCRTCEngine sharedInstance].defaultVideoStream startCapture];
// 关闭摄像头
[[RCRTCEngine sharedInstance].defaultVideoStream stopCapture];
// 切换摄像头
[[RCRTCEngine sharedInstance].defaultVideoStream switchCamera];
```

麦克风管理

将原有的开关麦克风接口调用替换为

```
// 设置禁用麦克风采集
[[RCRTCEngine sharedInstance].defaultAudioStream setMicrophoneDisable:YES];
```

扬声器管理

将原有的扬声器/听筒切换接口调用替换为

```
// 设置使用扬声器
[[RCRTCEngine sharedInstance] enableSpeaker:YES];
```

设置视频属性

将原有的 setVideoCaptureParam: 接口调用替换为

```
RCRTCVideoStreamConfig *videoConfig = [[RCRTCVideoStreamConfig alloc] init];
videoConfig.videoSizePreset = RCRTCVideoSizePreset720x480;
videoConfig.videoFps = RCRTCVideoFPS30;
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoConfig:videoConfig];
```

视频大小流

将原有的控制是否开启大小流的接口替换为

```
[RCRTCEngine sharedInstance].defaultVideoStream.enableTinyStream = YES;
```

本地视频流处理

原有的获取本地流数据 `RongRTCVideoCMSampleBufferCallback` 方法已经废弃，升级后获取本地流与远端流的视频数据需要设置代理 `RCRTCVideoOutputStreamEventDelegate` 并实现对应代理方法

```
[RCRTCEngine sharedInstance].defaultVideoStream.streamEventDelegate = self;
/*!
采集视频数据回调

@param videoFrame 采集的视频数据，类型为 RCRTCVideoFrameFormatNV12
@param stream 采集的视频帧数据所属接收到的流
@discussion
采集的视频数据，如果修改该视频帧，会影响本地渲染和编码发送的视频帧数据，不支持修改数据格式

@remarks 代理
added from 5.2.3
*/
- (void)outputVideoStream:(RCRTCVideoOutputStream *)stream captureVideoFrame:(nullable RCRTCVideoFrame *)videoFrame;
```

远端视频流处理

原有的获取本地流数据 `willRenderCVPixelBufferRef:stream:` 方法已经废弃，升级后获取远端流的视频数据需要设置代理 `RCRTCVideoInputStreamEventDelegate` 并实现对应代理方法

```
// 获取到远端流后设置代理
((RCRTCVideoInputStream *)stream).streamEventDelegate = self;
/*!
即将渲染视频帧的数据回调

@param videoFrame 即将渲染的视频帧数据
@param stream 即将渲染的视频帧数据所属接收到的流
@discussion
即将渲染视频帧数据，如果修改该视频帧，会影响本地渲染远端用户的视频帧，不支持修改数据格式

@remarks 代理
added from 5.1.14
*/
- (void)inputVideoStream:(RCRTCVideoInputStream *)stream willRenderVideoFrame:(nullable RCRTCVideoFrame *)videoFrame;
```

进阶功能升级说明

混音设置

3.X 的混音功能只支持混音本地资源，5.X 支持混音在线网络音频资源。升级 5.X 后混音方法名称没有变动，需要注意的是方法调用对象由 RongRTCAudioMixer 变为 RCRTCAudioMixer

```
// 开始混音本地音频
NSString *audioFilePath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"mp3"];
NSURL *mediaUrl = [NSURL fileURLWithPath:audioFilePath];
[[RCRTCAudioMixer sharedInstance] startMixingWithURL:mediaUrl
 playback:YES
 mixerMode:RCRTCMixerModeMixing
 loopCount:NSUIntegerMax];
```

更多具体的方法调用参考 5.X 的[混音设置文档](#)

设置音频属性

为了来满足不同场景对音频设置的需求，同时降低使用复杂度，融云 RongRTCLib 从 5.1.0 开始，对音频码率（5.1.0 音频通话质量）和音频模式（5.1.0 音频通话模式）进行了接口合并封装，并重新设计，对外提供音频质量 + 音频模式的接口。具体的参数选择参考 5.X 的[设置音频属性文档](#)

音频流处理

SDK 5.X 依旧提供了本地音频流发送前和远端视音频流接收后的上报，数据回调接口的所属类对象以及名称对比 3.X 有所变动，返回的数据仍为 PCM 数据。具体可以参考对应 5.X 的[音频流处理文档](#)替换对应回调。

发布自定义流

发布自定义流的逻辑可以直接通过 5.X 的[自定义流文档](#)进行修改适配

发布屏幕共享

屏幕共享的升级方案可以通过 5.X 的[屏幕共享文档](#)进行修改适配

3.X 升级到 5.X

更新时间:2024-08-30

本文描述 RTCLib SDK 直播场景的升级步骤。

升级概述

- RTCLib SDK 5.X 对比 3.X，适配了 iPhone 新机型与新系统，功能更丰富，更稳定，并在之前版本上修复了大量问题，建议尽早升级至新版 RTCLib SDK。
- 3.X 版本的观众没有加房间的功能，只能通过 `subscribeLiveAVStream:liveType:handler:` 这个接口订阅 liveUrl 合流；5.X 版本的观众会先加入房间，拿到房间内的合流来订阅。

前置条件

- 5.X RTCLib SDK 依赖 RongIMLibCore 请您确保已将 RongIMLibCore 升级至 5.X。
- RongIMLib SDK 5.X 做了拆分，请遵照 IMLib 的升级文档先将使用对应功能的 SDK 进行升级集成。
- 已遵照 IMLib 升级要求将您的 Cocoapod 版本升级到 1.10.0 以上。

手动集成

1. 将原先 RTCLib SDK 依赖的系统库全部去掉，如果 App 或者其他 SDK 有依赖的系统库除外

- AssetsLibrary
- MapKit
- ImageIO
- Security
- SystemConfiguration
- QuartzCore
- AVFoundation
- GLKit
- OpenGL ES
- CoreGraphics
- CoreLocation
- CoreTelephony
- CoreFoundation
- CoreMedia
- CoreAudio
- CoreVideo
- CFNetwork

- AudioToolbox
 - VideoToolBox
 - AVFoundation
 - libc++.tbd
 - libz.tbd
 - libbz2.tbd
 - libiconv.tbd
 - libsqlite3.tbd
2. 将 3.X 的 RongRTCLib.framework、RongIMLib.framework、libopencore-amrnb.a (该库包含在 RongIMLib 软件包中) 删除并替换为 5.X 的 RongRTCLib.xcframework、RongIMLibCore.xcframework
 3. General -> Frameworks,Libraries,and Embedded Binaries 中将 RongXX.framework 的 Embed 设置为 Embed & Sign

pod 集成

1. 在 podfile 中添加如下内容：

```
# pod 'RongRTCLib', '~> 3.2.2' # 注释 3.X 的内容
pod 'RongCloudRTC/RongRTCLib', '~> x.y.z'
```

2. 请在终端中运行以下命令：

```
pod install
```

如果出现找不到相关版本的问题，可先执行 `pod repo update`，再执行 `pod install`。

接口替换说明

对象命名的改变

RTCLib SDK 从 3.X 升级到 5.X 将对象的命名规则由 RongXXX 变更为 RCXXX，您可以全局针对 RTCLib 的对象进行统一的替换。

例如加入房间的方法名称没有变动，方法返回的实例对象名称由 RongRTCRoom 变更为 RCRTCRoom：

```
- (void)joinRoom:(NSString *)roomId
config:(RCRTCRoomConfig *)config
completion:(nullable void (^)(RCRTCRoom *_Nullable room, RCRTCCode code))completion;
```

初始化接口

将原有 AppKey 初始化方法进行替换

```
#import <RongIMLibCore/RongIMLibCore.h>

[[RCCoreClient sharedCoreClient] initWithAppKey:@"从控制台申请的 AppKey"];
```

连接接口

将原有连接 IM 方法替换为

```
// 连接 IM 服务
[[RCCoreClient sharedCoreClient] connectWithToken:@"从您服务器端获取的 Token"
dbOpened:^(RCDBErrorCode code) {}
success:^(NSString *userId) {}
error:^(RCConnectErrorCode status) {}];
```

音视频引擎初始化

将原有 sharedEngine 方法替换为

```
RCRTCEngine *engine = [RCRTCEngine sharedInstance];
```

状态监听接口

将原有的 RongRTCActivityMonitorDelegate 状态监视器代理替换为

```
@property (nonatomic, weak, nullable) id<RCRTCStatusReportDelegate> statusReportDelegate;
```

房间事件代理回调接口

将原有的 RongRTCRoomDelegate 房间事件代理替换为

```
/*!
房间事件代理
*/
@property (nonatomic, weak, nullable) id<RCRTCRoomEventDelegate> delegate;
```

代理中对应的方法变动对照 RCRTCRoomEventDelegate 中的说明替换即可

获取当前房间

将原有的当前加入房间 currentRoom 属性替换为

```
@property (nonatomic, strong, readonly, nullable) RCRTCRoom *room;
```

断线重连设置

断线重连功能默认开启，如果 3.X 有使用 `setReconnectEnable`：设置为 NO；升级后需要在引擎初始化时传入以下配置进行关闭：

```
RCRTCConfig *config = [[RCRTCConfig alloc] init];  
config.isEnableAutoReconnect = NO;  
[[RCRTCEngine sharedInstance] initWithConfig:config];
```

离开房间接口

将原有的 `leaveRoom:completion:` 方法替换为

```
-(void)leaveRoom:(nullable RCRTCOperationCallback)completion;
```

房间配置接口

原有的 `RongRTCRoomConfig` 的配置类变更为 `RCRTCRoomConfig`，并且新增了参数 `RCRTCLiveRoleType` 用来标识加入房间的用户身份是主播还是观众

```
RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];  
config.roomType = RCRTCRoomTypeLive; //房间类型为直播  
config.liveType = RCRTCLiveTypeAudioVideo; //直播类型为音视频直播  
config.roleType = RCRTCLiveRoleTypeBroadcaster; //以主播身份加入房间
```

主播发布与取消发布资源接口

加入房间成功后 3.X 是通过加入房间返回的 `RongRTCRoom` 对象管理本地默认音视频流；5.X 变动为通过加入房间返回的 `RCRTCRoom` 中的 `RCRTCLocalUser` 对象管理本地默认音视频流

```
// 发布本地默认音视频资源  
[[RCRTCEngine sharedInstance].room.localUser publishDefaultLiveStreams:^(BOOL isSuccess, RCRTCCode desc,  
RCRTCLiveInfo * _Nullable liveInfo) {  
// 可以通过拿到的liveInfo对象进行推流相关配置  
}];  
// 取消发布本地默认音视频资源  
[[RCRTCEngine sharedInstance].room.localUser unublishDefaultLiveStreams:^(BOOL isSuccess, RCRTCCode  
code) {  
}];
```

主播订阅与取消订阅资源接口

加入房间成功后 3.X 是通过加入房间返回的 `RongRTCRoom` 对象管理对端的音视频流；5.X 变动为通过加入房间返回的

```
// 订阅对端音视频资源
[[RCRTCEngine sharedInstance].room.localUser subscribeStream:avStreams
tinyStreams:nil
completion:^(BOOL isSuccess, RCRTCCode desc) {
}];
// 取消订阅对端音视频资源
[[RCRTCEngine sharedInstance].room.localUser unsubscribeStream:streams
completion:^(BOOL isSuccess, RCRTCCode desc) {
}];
```

创建渲染视图

3.X 的创建渲染视图分为本地渲染视图 RongRTCLocalVideoView 与远端渲染视图 RongRTCRemoteVideoView，5.X 统一将创建渲染视图调整为 RCRTCVideoView

```
// 初始化本地渲染视图
RCRTCVideoView *view = [[RCRTCVideoView alloc] initWithFrame:CGRectMake(100, 100, 100, 100)];
```

渲染视频流

将 3.X 的渲染视图接口 setVideoRender: 接口替换为

```
// 渲染本地视图
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoView:@"创建的 RCRTCVideoView 视图"];
// 渲染远端视图
[[RCRTCVideoInputStream *]stream setVideoView:@"创建的 RCRTCVideoView 视图"];
```

摄像头管理

将原有的摄像头相关接口调用替换为

```
// 打开摄像头
[[RCRTCEngine sharedInstance].defaultVideoStream startCapture];
// 关闭摄像头
[[RCRTCEngine sharedInstance].defaultVideoStream stopCapture];
// 切换摄像头
[[RCRTCEngine sharedInstance].defaultVideoStream switchCamera];
```

麦克风管理

将原有的开关麦克风接口调用替换为

```
// 设置禁用麦克风采集
[[RCRTCEngine sharedInstance].defaultAudioStream setMicrophoneDisable:YES];
```

扬声器管理

将原有的扬声器/听筒切换接口调用替换为

```
// 设置使用扬声器
[[RCRTCEngine sharedInstance] enableSpeaker:YES];
```

设置视频属性

将原有的 setVideoCaptureParam: 接口调用替换为

```
RCRTCVideoStreamConfig *videoConfig = [[RCRTCVideoStreamConfig alloc] init];
videoConfig.videoSizePreset = RCRTCVideoSizePreset720x480;
videoConfig.videoFps = RCRTCVideoFPS30;
[[RCRTCEngine sharedInstance].defaultVideoStream setVideoConfig:videoConfig];
```

视频大小流

将原有的控制是否开启大小流的接口替换为

```
[RCRTCEngine sharedInstance].defaultVideoStream.enableTinyStream = YES;
```

本地视频流处理

原有的获取本地流数据 RongRTCVideoCMSampleBufferCallback 方法已经废弃，升级后获取本地流与远端流的视频数据需要设置代理 RCRTCVideoOutputStreamEventDelegate 并实现对应代理方法

```
[RCRTCEngine sharedInstance].defaultVideoStream.streamEventDelegate = self;
/*!
采集视频数据回调

@param videoFrame 采集的视频数据，类型为 RCRTCVideoFrameFormatNV12
@param stream 采集的视频帧数据所属接收到的流
@discussion
采集的视频数据，如果修改该视频帧，会影响本地渲染和编码发送的视频帧数据，不支持修改数据格式

@remarks 代理
added from 5.2.3
*/
- (void)outputVideoStream:(RCRTCVideoOutputStream *)stream captureVideoFrame:(nullable RCRTCVideoFrame *)videoFrame;
```

远端视频流处理

原有的获取本地流数据 willRenderCVPixelBufferRef:stream: 方法已经废弃，升级后获取远端流的视频数据需要设置代理 RCRTCVideoInputStreamEventDelegate 并实现对应代理方法

```

// 获取到远端流后设置代理
((RCRTCVideoInputStream *)stream).streamEventDelegate = self;
/*!
即将渲染视频帧的数据回调

@param videoFrame 即将渲染的视频帧数据
@param stream 即将渲染的视频帧数据所属接收到的流
@discussion
即将渲染视频帧数据，如果修改该视频帧，会影响本地渲染远端用户的视频帧，不支持修改数据格式

@remarks 代理
added from 5.1.14
*/
- (void)inputVideoStream:(RCRTCVideoInputStream *)stream willRenderVideoFrame:(nullable RCRTCVideoFrame *)videoFrame;

```

观众逻辑变动说明

观众订阅流程

5.X 观众订阅流程替换为

```

// 1、设置房间类型
RCRTCRoomConfig *config = [[RCRTCRoomConfig alloc] init];
config.roomType = RCRTCRoomTypeLive; //房间类型
config.liveType = RCRTLIVETypeAudioVideo; //直播类型
config.roleType = RCRTLIVERoleTypeAudience; //观众角色

// 2、以观众身份加入房间
[[RCRTCEngine sharedInstance] joinRoom:@"房间号"
config:config
completion:^(RCRTCRoom * _Nullable room, RCRTCCode code) {
if (code != RCRTCCodeSuccess) {
// 错误处理
return;
}
// 设置房间事件代理
self.room = room;
self.room.delegate = self;

// 3、加入房间时获取房间已经存在的 live 合流，直接订阅
NSArray *liveStreams = [room getLiveStreams];
if (liveStreams.count) {
// 从房间拿到合流去订阅
[room.localUser subscribeStream:liveStreams tinyStreams:nil completion:^(BOOL isSuccess, RCRTCCode desc)
{
// to do something
}
}
}];
}];

```

观众上麦流程

3.X 的观众的上麦流程需要先取消订阅，然后再加入房间发布资源；5.X 的观众上麦流程替换为

```

// 1、获取需要发布的资源
NSArray *streams = @[[RCRTCEngine sharedInstance].defaultAudioStream, [RCRTCEngine
sharedInstance].defaultVideoStream];
// 2、切换身份变为主播
[[RCRTCEngine sharedInstance].room.localUser switchToBroadcaster:streams onSuccess:^(RCRTLIVEInfo *
_Nonnull liveInfo) {
//TODO:切换到主播成功，订阅房间内流资源
NSMutableArray *streams = [NSMutableArray array];
for (RCRTCRemoteUser *user in self.room.remoteUsers) {
if (user.remoteStreams.count) {
[streams addObjectFromArray:user.remoteStreams];
}
}
if (streams.count) {
// 订阅资源
//[self subscribeStreams:streams];
}
} onFailure:^(RCRTCCode code) {
NSLog(@"OnFailed:%@",@(code));
} onKicked:^(
NSLog(@"OnKicked");
}]);

```

主播下麦流程

3.X 的主播的下麦流程需要先离开房间，然后再订阅合流资源；5.X 的主播下麦流程替换为

```

[[RCRTCEngine sharedInstance].room.localUser switchToAudienceOnSucceed:^(
//TODO:观众订阅
NSArray *streams = [self.room getLiveStreams];
//[self subscribeStreams:streams];
} onFailure:^(RCRTCCode code) {
NSLog(@"OnFailed:%@",@(code));
} onKicked:^(
NSLog(@"OnKicked");
}]);

```

更详细的说明可以参考 5.X 的[同房间连麦文档](#)

进阶功能升级说明

混音设置

3.X 的混音功能只支持混音本地资源，5.X 支持混音在线网络音频资源。升级 5.X 后混音方法名称没有变动，需要注意的是方法调用对象由 RongRTCAudioMixer 变为 RCRTCAudioMixer

```

// 开始混音本地音频
NSString *audioFilePath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"mp3"];
NSURL *mediaUrl = [NSURL URLWithString:audioFilePath];
[[RCRTCAudioMixer sharedInstance] startMixingWithURL:mediaUrl
playback:YES
mixerMode:RCRTCMixerModeMixing
loopCount:NSUIntegerMax];

```

更多具体的方法调用参考 5.X 的[混音设置文档](#)

设置音频属性

为了来满足不同场景对音频设置的需求，同时降低使用复杂度，融云 RongRTClib 从 5.1.0 开始，对音频码率（5.1.0 音频通话质量）和音频模式（5.1.0 音频通话模式）进行了接口合并封装，并重新设计，对外提供音频质量 + 音频模式的接口。具体的参数选择参考 5.X 的[设置音频属性](#)

音频流处理

SDK 5.X 依旧提供了本地音频流发送前和远端视音频流接收后的上报，数据回调接口的所属类对象以及名称对比 3.X 有所变动，返回的数据仍为 PCM 数据。具体可以参考对应 5.X 的[音频流处理文档](#)替换对应回调。

发布自定义流

发布自定义流的逻辑可以直接通过 5.X 的[自定义流文档](#)直接进行修改适配

发布屏幕共享

屏幕共享的升级方案可以通过 5.X 的[屏幕共享文档](#)进行修改适配

合流布局设置

合流布局设置的升级方案可以通过 5.X 的[合流布局文档](#)进行适配

更新日志

5.8.2

更新时间:2024-08-30

发布日期：2024/06/05

- **RTCLib SDK**

- 提升了 SDK 版本号为 5.8.2，无新增特性与修复。

- **CallLib SDK**

- 提升了 SDK 版本号为 5.8.2，无新增特性与修复。

- **CallKit SDK**

- 提升了 SDK 版本号为 5.8.2，无新增特性与修复。

5.8.1

发布日期：2024/04/03

- 提升 SDK 版本号为 5.8.1，无新增特性与修复。

5.8.0

发布日期：2024/03/29

- 提升 SDK 版本号为 5.8.0，无新增特性与修复。

5.6.10

发布日期：2024/03/20

- **RTCLib SDK**

- 完善了包含 `PrivacyInfo.xcprivacy` 的 Framework。详见 [关于 2024 春季 iOS 的隐私清单的通知](#)。

5.6.9

发布日期：2024/01/31

- **RTCLib SDK:**

- 新增：提供包含 `PrivacyInfo.xcprivacy` 的 Framework。详见 [关于 2024 春季 iOS 的隐私清单的通知](#)。

- 修复：PK 偶现不成功的问题。

- **CallKit SDK:**

- 新增：提供包含 `PrivacyInfo.xcprivacy` 的 Framework。详见 [关于 2024 春季 iOS 的隐私清单的通知](#)。

5.6.8

发布日期：2023/12/29

- **RTCLib SDK:**

- 修复：修复部分内部已知问题。

5.6.7

发布日期：2023/11/30

- **RTCLib SDK:**

- 修复：偶现的崩溃问题。

- **CallLib SDK:**

- 修复：偶现的崩溃问题。

5.6.5

发布日期：2023/10/12

- **RTCLib SDK:**

- 修复：跨房间连麦场景下断网重连后黑屏问题。

- **CallLib SDK:**

- 修复：断网离开房间后，后续对端无法呼入问题。

5.6.4

发布日期：2023/09/25

- **RTCLib SDK:**

- 修复：修复退出房间不结束云端录制问题。

5.6.3

发布日期：2023/08/31

提升 SDK 版本号为 5.6.3，无新增特性与修复。

5.6.2

发布日期：2023/08/11

提升 SDK 版本号为 5.6.2，无新增特性与修复。

5.6.1

发布日期：2023/07/14

提升 SDK 版本号为 5.6.1，无新增特性与修复。

5.6.0

发布日期：2023/07/03

提升 SDK 版本号为 5.6.0，无新增特性与修复。

5.5.0

发布日期：2023/09/08

为配合 IM SDK 5.5.0 稳定（stable）版本使用，提升 SDK 版本号为 5.5.0。功能基于 5.4.6 最新 Hotfix 版本，无新增特性与修复。

5.4.6

发布日期：2023/06/15

• CallKit SDK:

1. 优化：CallKit 头像跟随 IMKit 配置

• CallLib SDK:

1. 修复：上抛本地渲染帧与上抛发送帧回调接口错误的问题

5.4.5

发布日期：2023/05/29

- 修复了 CallLib、RTCLib 偶现的崩溃问题。

5.4.4

发布日期：2023/05/11

• RTCLib SDK:

1. 兼容 iOS 16.4.1 上行码率低导致对端视频画面模糊问题
2. 修复预热逻辑出现多线程问题导致的崩溃
3. 修复 Ping 探测工具类断言导致的崩溃
4. 修改 RTCLib 与 IM SDK 版本匹配规则。从 5.4.4 开始，要求前两位保持一致。注意，RTCLib 5.4.4 不可匹配小于 5.4.4 的 IM SDK。

5.4.2

发布日期：2023/04/20

• CallLib SDK:

1. 新增：音视频信消息推送默认设置 vivo category 参数为 IM

• CallKit SDK:

1. 优化：单群聊音视频推送的文案显示

5.4.1

发布日期：2023/04/07

• RTCLib SDK:

1. 新增：美声插件新增部分美声特效
2. 修复：用户跨房间连麦场景下，调用离开副房间接口引起的对方主播自动取消订阅的问题

• CallLib SDK:

1. 修复：改进来电呼入场景下杀死 App 后的挂断原因。修改后，该场景下挂断原因为“已挂断”
2. 修复：设置视频分辨率 1080p 格式不生效的问题;

• CallKit SDK:

1. 优化: 对齐 Android、iOS 的多语言提示文案
2. 修复：语音通话接收端前台铃声从听筒发出的问题
3. 修复：修复接听方弹出通话页面后网络断开场景下，点击接听按钮没有反应的问题
4. 修复：阿拉伯语环境下 UI 适配的问题

5.4.0

发布日期：2023/03/03

• **RTCLib SDK:**

1. 修复：快速点击播放多个音效后，连续调用停止单个或所有音效接口，偶现接听端持续听到杂音
2. 优化：更新 webRTC 库，防止偶现带宽探测引起的崩溃问题

• **CallLib SDK:**

1. 新增：音视频信消息推送默认设置华为 category 参数为 VOIP
2. 修复：群组通话再次邀请已在通话中的人员进行通话，从发起端对已在通话中的人员进行过滤
3. 新增：CallLib 可接入相芯美颜插件

• **CallKit SDK:**

1. 修复：iPhone 14 pro max 群组视频通话 "连接中" 文字显示被遮挡
2. 优化：对齐安卓端群组通话被其他端处理情况下的显示文案，小灰条显示「其他设备已处理」
3. 新增：CallKit 可接入相芯美颜插件

5.3.6

发布日期：2023/05/11

为配合 IM SDK 5.3.6 稳定 (stable) 版本使用，提升 SDK 版本号为 5.3.6。功能基于 5.3.5 版本，无新增特性与修复。

5.3.5

发布日期：2023/02/10

• **CallKit SDK:**

1. 修复：群组三人音频通话时，发起方挂断后其他人显示头像为发起方的问题
2. 修复：用户发起群组呼叫时（呼叫两人以上），被邀请方在未接听的情况下，底部显示的头像不正确的问题
3. 修复：用户在群组通话中，再次进入邀请成员选择界面时候，已选择人数显示不对的问题
4. 修复：群通话中未接通用户 UI 显示“...”而非“连接中”，以及部分场景下显示此状态不对的问题

5.3.4

发布日期：2023/01/17

• **RTCLib SDK:**

1. 新增：订阅流接口支持在订阅流失败时返回订阅失败的资源列表
2. 新增：新增调节远端资源的播放音量的接口 `remotePlaybackVolume`
3. 新增：支持本地采集音量增益，调节范围由 [0-100] 改为 [0-200]
4. 修复：不销毁音视频引擎的情况下，切换不同 AppKey 连接没有更新对应的 logServerUrl 问题
5. 优化：AudioSession 设置根据不同系统版本选择不同 API，避免高版本出现设置 mode 导致 categoryOptions 变化
6. 优化：SEI 视频帧异步子线程处理，防止同步队列死锁

• **CallLib SDK:**

1. 修复：设置挂断推送模板 ID 实际设置为邀请模板 ID 的问题
2. 修复：在多端登录时，如果已接听的通话在其他设备上挂断处理
3. 修复：相同 UserId 多设备登录，后面登录的设备需要忽略离线 RC:VCInvite 类型消息

• **CallKit SDK:**

1. 修复：针对齐刘海机型 loading 图错位修改
2. 修复：修复群组音频呼入状态，连接成功后，对端头像不高亮问题
3. 修复：接通状态显示 loading 状态的 bug
4. 修复 发起群组通话，没有赋值 apnsCollapseId，导致挂断推送无法覆盖邀请推送

5.3.3

发布日期：2022/12/28

• **RTCLib SDK:**

1. 修复：弱网情况下 CDN 播放器播放 CDN 直播流延迟持续增加的问题
2. 修复：iOS16 或者带着蓝牙耳机情况下混音没声音的问题
3. 修复：在线混音子线程多次调用 stopMix，导致 SDK work 线程和 onlinefile 中队列死锁问题
4. 优化：unInit 的 resetAudioDevice 函数异步处理，防止主线程调用时被阻塞
5. 修复：CDN 播放器 Wi-Fi 切换移动网络时画面卡住的问题

• **CallKit SDK:**

1. 修复：多人通话选人界面勾选按钮 UI 布局 Y 轴不居中问题
2. 修复：多人语音选人界面 (collectionView) 删除不存在的 item 导致崩溃的问题

5.3.2

发布日期：2022/12/02

• RTCLib SDK:

1. 优化：适配车载蓝牙，观众模式下，AudioSession Category 可以使用 PlayBack 替代 PlayAndRecord
2. 优化：直播场景下，主播切换为观众时，关闭摄像头
3. 优化：被踢时，没有进行任务队列任务取消操作，影响被踢离开房间的时间
4. 修复：优化离开房间逻辑，及时取消任务队列中任务，避免出现离开房间还能听到声音的问题
5. 修复：ijkPlayer 累计延迟问题
6. 修复：MicOutputStream setIsMute = YES 静音本地流，然后切换蓝牙设备，错误地将本地流静音关闭的问题

5.3.1

发布日期：2022/11/18

• RTCLib SDK:

1. 优化：优化初始速率控制，帧率控制，降低卡顿
2. 修复：观众上麦未对服务器返回的心跳超时进行设置，导致上麦后的主播永远不会超时被踢出房间的问题
3. 修复：音频设备启动（AUGraphStart）返回 -50，启动失败（没有声音）问题
4. 修复：多线程访问远端用户列表导致的崩溃问题
5. 修复：离开房间未清空任务队列，导致仍能听到房间内声音的问题

5.3.0

发布日期：2022/11/04

• RTCLib SDK:

1. 新增：融云 CDN 插件新增 CDN 播放器组件，支持播放外部 URL
2. 新增：正式支持媒体补充信息（SEI）功能
3. 优化：加入房间、发布和取消资源的操作重试逻辑
4. 优化：根据听感改进音量线性调节方式
5. 优化：完善 AVCaptureSession 的错误处理逻辑
6. 优化：优化了内置 CDN 拉取首屏速度
7. 优化：SDP 长度太大日志输出
8. 修复：在 IM 断开时 `RCRTCModifyVideoResourceOperation` 无法正常结束的问题
9. 修复：处理远端用户开关摄像头/麦克风时，异步到了主线程处理 AU 资源，会导致 AU 中资源不同步问题

5.2.5

发布日期：2022/09/09

• RTCLib SDK:

1. 增加：发布相芯美颜插件（封装相芯美颜）。
2. 增加：本地背景音乐和短音效支持访问手机媒体库资源（APP 需要导入 MediaPlayer.framework）。
3. 增加：添加音频设备启动失败状态回调。
4. 优化：多端对齐大小流分辨率。
5. 优化：音频模块重构，减少 CPU 功耗。
6. 优化：缩短 setRemoteSDP 耗时。
7. 优化：3A 模块支持背景音乐送参考，消除麦克风采集的扬声器播放出来的背景音乐。
8. 优化：支持背景音乐和短音效独立播放，不依赖身份和是否加入房间。
9. 优化：内置华为 VQE，支持隐藏切换 WebRTC 3A 和 VQE。
10. 优化：RTCLib 内置采样率监测逻辑，防止用户层面自己修改采样和 SDK 内部不一致导致声音问题。
11. 优化：增加订阅资源校验，防止出现错误订阅不应该订阅的资源。。
12. 修复：iPhone 8 屏幕共享产生绿边问题。
13. 修复：屏幕共享发布取消发布，再次发布，视频被裁剪。

• **CallLib SDK:**

1. 修复：RongCallLib，PC 关闭摄像头（取消发布），iOS 错误将用户类型置成观察者。

5.2.4

发布日期：2022/07/22

• **RTCLib SDK:**

1. 优化：禁止主播订阅合流操作，增加逻辑健壮性;
2. 优化：优化了拉取CDN直播流的首屏速度;
3. 修复：修复了切换音频场景导致双通道效果失效问题;
4. 修复：修复了远端音频流回调数据错误问题;
5. 修复：修复了前置摄像头切换后置摄像头时，设置摄像头指定分辨率失败，错误使用了摄像头默认分辨率的问题;
6. 修复：修复了低端机（高端机不容易出现）使用1080P运行一段时间，内存不断增长最终导致崩溃问题;
7. 修复：修复了RongRTCReplayKitExt插件停止屏幕共享时点击系统弹窗中“停止”按钮，主APP收不到结束共享通知的问题;

• **CallKit SDK:**

1. 修复：修复了voip下AudioSession启动失败导致通话听不到对端声音问题;

5.2.3

发布日期：2022/06/01

• **RTCLib SDK:**

1. 新增：添加音频路由功能；

2. 修复：修复了偶发的观众听到主播声音失真的问题；
3. 修复：修复了一些偶现的线程阻塞问题；
4. 修复：修复了一些偶现的崩溃问题

5.2.2

发布日期：2022/05/01

• RTCLib SDK:

1. 修复：修复了一些内部的 BUG;

5.2.1

发布日期：2022/04/01

• RTCLib SDK:

1. 修复：修复了一些内部的 BUG;

5.2.0

发布日期：2022/03/01

从 5.2.0 版本开始，CallKit/CallLib/RTCLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

• RTCLib SDK:

1. 增加：uinit 接口，当直播结束后，释放资源;
2. 增加：水印接口支持 BGRA 格式;
3. 增加：跨房间连麦支持主播占位图功能;
4. 优化：ijkplayer 内存管理;
5. 修复：变声插件初始化不生效问题;

5.1.17

发布日期：2022/01/26

• RTCLib SDK:

1. 增加：屏幕共享时支持采集其他 App 的声音，作为一路混音输入源进行发送;
2. 增加：新增通话/会议/直播前检测网络状态的接口 `startLastmileProbeTest(LastmileProbeConfig config)`;
3. 增加：跨房间连麦支持主播占位图功能;

4. 优化：CDN 首次拉流慢的问题;
5. 优化：音乐聊天室模式和音乐教学模式下麦克风采集存在一定噪音的问题;
6. 修复：修复 RongRTCPlayer 插件内存泄漏问题;

- **CallLib SDK:**

1. 修复：离线状态下收到呼叫并未接通，再次上线后仍显示为呼入页面的问题。问题修复后，再次上线时可正常显示为呼入+挂断的两条配对的历史记录;

5.1.16

发布日期：2022/01/13

- **RTCLib SDK:**

1. 修复：调用混音接口时，混音停止状态和混音暂停状态回调出错的问题;

- **CallLib SDK:**

1. 修复：群组呼叫时场景下，当群 ID 和用户 ID 一样时，会呼叫失败的问题;

- **CallKit SDK:**

1. 修复：不勾选 Audio, AirPlay, and Picture in Picture，应用从后台切到前台，扬声器没有声音的问题;
2. 修复：群聊双方都在选人界面时，一方发起通话，另一方没弹窗的问题;

5.1.15

发布日期：2021/12/24

- **RTCLib SDK:**

1. 增加：在直播/会议场景下，支持在主播或参会者的视频图像上添加水印;
2. 修复：先调用 RCRTCVideoView 中 SetFillMode 设置 FillMode，再调用 CDNInputStream 中 SetVideoView 绑定视图，会导致 VideoView FillMode 不生效的问题;
3. 修复：调用 RCRTCCameraOutputStream 中 SwitchCamera，镜像功能失效;
4. 修复：Stream 第二次调用 SetVideoView，对应的 VideoView 会不显示;
5. 修复：音乐模式下 AirPods 蓝牙耳机音量小的问题;
6. 修复：客户端发送了不准确的 ResolutionInfo，导致云端录像回调参数有误;
7. 修复：单独调节某一个气氛音效（如鼓掌、叫好）的音量不起作用的问题;

5.1.14

发布日期：2021/12/10

- **RTCLib SDK:**

1. 增加：加入房间接口可以携带扩展信息，方便客户进行业务信息传递;
2. 增加：多端加入RTC房间时支持设置互踢策略，可选择策略包括：RCRTCJoinType.KICK(顶掉其他端)、RCRTCJoinType.REFUSE（当前端加入失败）；
3. 优化：如果在通话/直播/会议中进程被电话打断，挂断电话后（在 APP 进程被系统回收之前）可以恢复音视频通话;
4. 修复：关闭手机内置麦克风后再连接蓝牙耳机，会导致 SDK 获取到的麦克风数据有问题，此时对端会听到滋啦声。问题修复后，如果获取到的麦克风数据有问题，即将数据置为 0;

• **CallKit SDK:**

1. 修复：使用 UIWindow 同时使用 UIAlertView 导致的崩溃;

5.1.13

发布日期：2021/11/29

• **RTCLib SDK:**

1. 增加：混音功能支持左右声道切换，切换声道后同时将数据拷贝到另一个声道，以实现两个声道的数据同步，可用于支持 K 歌场景下原声与伴唱切换的功能;
2. 优化：优化通话过程打断逻辑。支持在闹钟打断后通话后及时恢复音频设备采集和播放;
3. 优化：支持 APP 退到后台之后，Background Modes 不需要获取 Audio 权限;

5.1.12

发布日期：2021/11/12

• **RTCLib SDK:**

1. 增加：音效插件 (VoiceBeautifier) 支持美声、变声、混响音效功能;
 - 美声：低沉、饱满、高亢
 - 变声：假声、绿巨人、小男孩、小女孩、成熟男性、老年男性、老年女性
 - 混响：KTV、演唱会
2. 优化：音频模式下语聊房、音乐播放场景噪音问题，提升用户听觉体验;

5.1.11

发布日期：2021/11/02

• **RTCLib SDK:**

1. 增加：支持在直播、会议时播放在线文件 (支持 HTTP、HTTPS、RTMP 和 RTSP 协议的 AVI、MP4、MKV、FLV 格式)，详情请参见「发布自定义流」功能文档。

2. 增加：观众订阅 MCU 合流成功时支持回调通知每个主播的音量。
3. 增加：MCU 合流支持主播设置占位图。
4. 修复：LeaveRoom 在加入房间成功前调用，导致无法离开房间。
5. 修复：有线耳机在切换音频场景时偶现双通道失效。
6. 修复：Player 插件回调播放进度时崩溃。
7. 修复：AudioDevice 偶现资源竞争导致死锁。

• **CallLib SDK:**

1. 修复：「多端进行音视频通话」时，iOS 端接收到的 Web 端消息中没有 MediaId 导致的群组邀请人黑屏和崩溃问题。

5.1.10

发布日期：2021/10/15

• **RTCLib SDK:**

1. 增加：支持双声道模式，包括混音模块（1.在会议和直播场景时且在音乐模式下;2.目前支持连线耳机和听筒、扬声器，暂不支持蓝牙耳机）。
2. 增加：远端音频数据回调的时间戳。
3. 优化：自定义本地视频流数据旋转矫正适配。
4. 优化：视频数据 sampleBuffer 释放策略。
5. 优化：设备启动耗时（直播首帧耗时优化）。
6. 优化：远端流视图和本地流视图渲染方式统一。

• **CallLib SDK:**

1. 修复：问题修复后，多端登录同一 UserID 时，其中一端通话结束后生成的通话记录可通过服务端的呼叫消息同步功能顺利同步到其他端；

• **CallKit SDK:**

1. 优化：iOS15 的 UI 适配

5.1.9

发布日期：2021/09/28

• **RTCLib SDK:**

1. 增加了 观众/主播角色切换功能
2. 增加了 通话前音频设备检测功能

5.1.8

发布日期：2021/09/10

• RTCLib SDK:

1. 增加了 屏幕共享插件。
2. 修复了 偶现的音频设备启动失败的问题。

5.1.7

发布日期：2021/08/27

• RTCLib SDK:

1. 增加了获取麦克风硬 3A 后音频数据的接口。
2. 增加了获取本端混音后音频数据的接口。
3. 增加了获取远端单路音频数据回调接口。

5.1.6

发布日期：2021/08/11

• RTCLib SDK:

1. 增加了在线音频文件混音功能。

5.1.5

发布日期：2021/07/09

• RTCLib、 CallLib、 CallKit SDK:

1. 增加了 CDN 播放器插件 (RongRTCPlayer) 功能。
2. 增加了闪光灯开关，摄像头变焦功能。

5.1.4

发布日期：2021/07/07

• RTCLib、 CallLib、 CallKit SDK:

1. 增加了美颜模块，包含美白、磨皮、红润、亮度、滤镜三款（浪漫，清新，唯美）。

2. RTCLib SDK、CallLib SDK、CallKit SDK 均可配合美颜模块使用。

5.1.3

发布日期：2021/07/01

• RTCLib SDK:

1. 增加了观众可以订阅主播分流。
2. 增加了按照 Room Id /Stream Id 进行合流布局选择策略的接口。
3. 重构了混音接口，为每一种混音策略提供状态详细回调和混音进度回调。
4. 修复了语聊房切换音乐模式之后出现的 Bug。
5. 修复了耳返功能的 Bug。

• RTCLib、CallLib、CallKit SDK:

1. 优化了编译部分，支持 BitCode。
2. 优化了 SDK 提供形式，以 XCFramework 提供。

5.1.2

发布日期：2021/05/21

• RTCLib SDK:

1. 适配 4.0 主播发布资源，5.0 观众无法订阅问题。
2. IM 与 RTC 解耦。
3. 添加远端视频 View 镜像开关，视频发送数据镜像开关。
4. 移除 Http 请求，全部使用 Https。
5. 修复观众与 Media Server 交互接口，传递 RoomId 参数错误。
6. 修复摘带 AirPods，对端能听到“滋啦”一声。

5.1.1

发布日期：2021/04/09

• RTCLib SDK:

1. RCRTCStreamStat 废弃 trackId，使用 streamId 替代。
2. 修复离开房间后，有概率音频设备不被释放问题。
3. 修复音频设备（扬声器，麦克风）有概率启动失败问题。
4. 修复摄像头被多线程操作时候，启动失败问题。

5.1.0

发布日期：2021/03/05

• **RTCLib SDK:**

1. 增加直播模板下观众加房间接口。
2. 增加小流码率、分辨率、帧率设置功能。
3. 增加通话过程中用户退出的异常处理，并把被踢和异常断开的回调方法放到 `RCRTCEngineEventDelegate` 中。
4. 优化用户在房间中异常退出或被踢操作使用 `RCRTCEngineEventDelegate` 抛出回调。
5. 优化音频相关内容。

5.0.0

发布日期：2021/01/18

• **RTCLib SDK:**

1. 修复分辨率切换时，小流分辨率没有随着大流分辨率比例切换的问题。
2. 修复远端开关麦克风没有通知本端的问题。
3. 修复直播模式观众从订阅音频切换视频订阅，View 不渲染问题。
4. 适配 Siri 语音时中断音视频通话，结束后通话恢复的逻辑。

• **CallLib SDK:**

1. 音视频呼叫功能，支持设置推送模板 ID，模板 ID 及模板中在“控制台-自定义推送文案”中进行设置。设置后根据目标用户通过 `RCPushProfile` 中的 `setPushLanguageString` 设置的语言环境，匹配模板中设置的语言内容进行推送，未匹配成功时使用融云默认内容进行推送。
2. 修复会议模式收到音频呼叫，本地摄像头被关闭的问题。

• **CallKit SDK:**

1. 适配 RTL 模式。

4.1.0 Dev

发布日期：2020/12/17

• **RTCLib SDK:**

1. 支持跨房间连麦功能。
2. 适配音视频通话被打断场景。
3. 支持 1080P 分辨率。
4. 修复连续 2 次加入房间，后一次失败会把前一次成功状态清除，导致后续其它操作失败的问题。
5. 修复断线重连不生效问题。

- **CallLib SDK:**

1. 修复视频通话默认使用了听筒的问题。
2. 修复群组通话无法显示忙碌的问题。

- **CallKit SDK:**

1. 将使用的资源文件从 IMKit 中分离出来存储到 CallKit.framework。
2. 修复快速点击扬声器开关两次，导致扬声器开关无法再点击的问题。

4.0.3 Stable

发布日期：2020/11/23

- **RTCLib SDK:**

1. 关闭 jitter buffer 加速。
2. 修复观众订阅失败，无法再进行主播接口操作。
3. 修复正常房间模式下取消订阅流，无法再次进行订阅操作的问题。
4. 修复后台仅开通音频情况下，观众调用订阅接口无法订阅成功的问题。
5. 修复音频设备切换问题。
6. 修复部分场景状态数据中 RTT 为 0 的问题。
7. 修复用户离开或者掉线，回调中用户对象为空的问题。
8. 修复远端用户关闭音频，本地记录状态错误的问题。

- **CallLib SDK:**

1. 修复 CallLib SDK 扬声器状态的问题。

4.0.4 Dev

发布日期：2020/11/12

- **RTCLib SDK:**

1. 增加耳返功能。
2. 增加 setCameraPosition 设置摄像头位置功能接口。
3. 增加直播混流背景色设置接口。
4. 优化 OutputStream 创建条件。
5. 修复音频设备切换问题。
6. 修复部分场景状态数据中 RTT 为 0 的问题。
7. 修复用户离开或者掉线，回调中用户对象为空的问题。
8. 修复远端用户关闭音频，本地记录状态错误的问题。

- **CallLib SDK:**

1. 增加呼叫、挂断音视频时设置厂商通知属性功能。

2. 修复 CallLib SDK 扬声器状态的问题。

• **CallKit SDK:**

1. 适配 iOS14。

4.0.3.2 Dev

发布日期：2020/09/29

• **RTCLib SDK:**

1. 关闭 jitter buffer 加速。
2. 修复观众订阅失败，无法再进行主播接口操作。
3. 修复正常房间模式下取消订阅流，无法再次进行订阅操作的问题。
4. 修复后台仅开通音频情况下，观众调用订阅接口无法订阅成功的问题。

4.0.3.1 Dev

发布日期：2020/09/22

• **RTCLib SDK:**

1. 修复 Bug，增强稳定性。

4.0.3 Dev

发布日期：2020/09/18

• **RTCLib SDK:**

1. 修复直播模式发布默认资源成功后，未将 outputStream 流添加到 localUser 中，导致数据不全引起后序使用错误的问题。
2. 修复第一次订阅后，第二次直接减量订阅引起的数据未对齐的问题。
3. 修复发布默认音视频后，单独取消发布视频，会导致音频也会被取消发布的问题。
4. 修复本地视频 setIsMute 时对小流不起作用。
5. 修复 setVideoOrientation 接口，设置摄像头采集角度，再通过 videoSendBufferCallback 获取到的数据和设置的不一致问题。
6. RCRTCRemoteVideoView 中添加 setHidden，覆盖原始实现，优化用户使用体验。

4.0.2 Dev

发布日期：2020/08/19

- **RTCLib SDK:**

1. 音视频数据支持自定义加密功能。
2. 增加播放音效功能。
3. 修复观众和主播模式切换，有概率失败的问题。
4. 修复错误处理断线重连后房间内数据问题。
5. 修复在未加入房间时操作扬声器和麦克风不起作用的问题。
6. 优化统一观众端 UserId 为连接 IM 时使用的 UserId。

- **CallKit SDK:**

1. 修复视频时点击大屏错误进行了大小屏切换操作的问题。

4.0.1 Dev

发布日期：2020/07/21

- **RTCLib SDK:**

1. 修复混音功能中的 Bug。
2. 优化 HTTP/HTTPS 逻辑支持 SNI。
3. 修复 SDK 开关麦克风没有通知的问题。
4. 修复 SDK 取消发布资源后，再发布资源不成功的问题。

4.0.0.1 Dev

发布日期：2020/06/19

- **RTCLib SDK:**

1. 直播功能支持 CDN 推流。
2. 直播 MCU 合流接口支持自定义视频流。
3. 增加动态切换视频分辨率接口 setVideoConfig。
4. SDK 支持了对焦/曝光功能。
5. 对 SDK 接口进行了重构，并统一各端返回错误码。
6. 优化 SDK 与 Media server 的交互逻辑。

3.2.2 Dev

发布日期：2020/05/08

- **RTCLib SDK:**

1. 发布自定义流可支持设置最大、最小码率。

2. 支持了音视频流分别发布能力。
3. 实现 Web、iOS、Android 多端码率设置对齐，统一 SDK 内置默认分辨率对应的码率。
4. 优化了直播场景下断线重连逻辑。

3.2.1 Dev

发布日期：2020/04/10

• RTCLib SDK:

1. 优化 HTTP / HTTPS 切换逻辑。
2. 适配蓝牙耳机设备。
3. 优化摄像头处理逻辑。

3.2.0 Dev

发布日期：2020/02/20

• RTCLib SDK:

1. 支持视频直播功能。
2. 优化 SDK 连接，HTTP、HTTPS 切换重试连接逻辑。
3. 修复一些 Bug 提升了稳定性。

3.1.7 Dev

发布日期：2020/01/10

• RTCLib SDK:

1. 修复同 Web 端通话时，获取不到 UserID 的问题。
2. 优化连接逻辑，提高连通率。
3. 优化 Media Server 地址选择逻辑，避免因 DNS 解析引起的漂移数据中心。

3.1.6 Dev

发布日期：2019/12/05

• RTCLib SDK:

1. 修复 SDK Bug 增强了稳定性。

3.1.5 Dev

发布日期：2019/11/22

• **RTCLib SDK:**

1. 新增全员禁音方法，关闭所有远端用户的声音。
2. 优化 RTC 不在房间中的错误返回逻辑，服务端和客户端错误码分开。
3. 增加判断是否切换摄像头成功的逻辑，避免出现坏摄像头导致 App 崩溃的问题。

3.1.4 Dev

发布日期：2019/10/30

• **RTCLib SDK:**

1. 增加调节混音音量功能接口。
2. 修复因为 Tag 的原因导致开关摄像头不生效的问题。
3. 修复断网两分钟之后音视频不会恢复的问题。

3.1.3 Dev

发布日期：2019/09/20

• **RTCLib SDK:**

1. 修复部分情况下引起的 Bug，增强服务稳定性。

3.1.2 Dev

发布日期：2019/08/31

• **RTCLib SDK:**

1. 修复部分情况下引起的 Bug，增强服务稳定性。

3.1.1 Dev

发布日期：2019/08/16

• **RTCLib SDK:**

1. 修复蓝牙耳机直接关机时 SDK 状态判断不准的问题。
2. 修复 UIKit 系统库名字错误。
3. 修复 SDK 同时使用美颜和水印导致的内存泄漏。

4. 优化摄像头方向重复设置逻辑。

3.1.0 Dev

发布日期：2019/08/06

- **RTCLib SDK:**

1. 修复部分情况下引起的 Bug，增强服务稳定性。

3.0.8 Dev

发布日期：2019/07/19

- **RTCLib SDK:**

1. 修改第一个关键回调通知多次的 Bug。
2. 修复只发送音频时，视频状态错误的 Bug。

3.0.7 Dev

发布日期：2019/06/28

- **RTCLib SDK:**

1. 修复 SDK 部分 Bug，增强了稳定性。
2. 修复 iOS 10.3 及以下系统频繁进出房间偶现的崩溃问题。

3.0.6 Dev

发布日期：2019/06/06

- **RTCLib SDK:**

1. 增加外置 Mic 和网络音频流输入源混音功能。
2. 增加本地文件输入源混音功能。
3. 修复由于内存泄漏导致的视频绘制 context 崩溃的问题。
4. 优化人员进出 SDP 协商造成的视频和音频卡顿问题。
5. 自定义视频逻辑优化，音视频同步逻辑优化。
6. 优化底层数据协商逻辑。

3.0.5 Dev

发布日期：2019/05/17

• **RTCLib SDK:**

1. 修复因加入房间时序不对导致的内存泄露问题。
2. 修复偶现的由于多线程访问和修改 RongRTCRemoteView 导致的崩溃问题。
3. 修复 turnOnCamera 属性自动打开的问题。
4. 修复偶现的无法准确获取远端用户的 Bug。
5. 修复 App 突然杀掉又进入房间时远端画面黑屏的问题。
6. 优化底层 track 绑定逻辑。

3.0.4 Dev

发布日期：2019/05/09

• **RTCLib SDK:**

1. 修复 WIFI 和 4G 网络切换时音视频不通的问题。
2. 修复有人离开房间获取房间 remoteUsers 数据不准确的问题。
3. 修复杀掉 APP 再进入房间时远端用户画面黑屏的问题。

3.0.3 Dev

发布日期：2019/04/26

• **RTCLib SDK:**

1. 增加多 MediaServer 地址动态切换逻辑。
2. 修复发布订阅流程中增量订阅逻辑出错导致的取消发布资源出错的问题。

3.0.2 Dev

发布日期：2019/04/22

• **RTCLib SDK:**

1. 优化 SDK 断网重连机制，增强了稳定性。

3.0.1 Dev

发布日期：2019/04/11

- **RTCLib SDK:**

1. 修复 SDK 部分 Bug ，增强了稳定性。

客户端 API

RTCLib

更新时间:2024-08-30

以下是 RTCLib 5.x 的 API 参考文档 (appledoc) :

- [RTCLib 音视频会议、低延迟直播 \(无 UI\)](#) 

RTCLib 5x 支持多种插件。以下是插件的 API 参考文档 (appledoc) :

- [RongRTCReplayKitExt 屏幕共享](#) 
- [FaceBeautifier 美颜](#) 
- [VoiceBeautifier 美声](#) 

CallLib

以下是 CallLib 5.x 的 API 参考文档 (appledoc) :

- [CallLib 音视频通话 \(不含 UI\)](#) 

CallKit

以下是 CallKit 5.x 的 API 参考文档 (appledoc) :

- [CallKit 音视频通话 \(含 UI\)](#) 

状态码

即时通讯

更新时间:2024-08-30

状态码	描述
20106	该用户处于单聊禁言状态，禁止发送单聊消息。
20604	发送消息频率过高，1 秒钟最多只允许发送 5 条消息
20605	信令被封禁。如遇到该错误，请提交工单。
20606	操作不支持。仅私有云有效，服务端禁用了该操作
20607	请求超出了调用频率限制，请稍后再试
21406	当前用户不在该讨论组中
21501	发送的消息中包含敏感词（发送方发送失败，接收方不会收到消息）
21502	消息中敏感词已经被替换（接收方可以收到被替换之后的消息）
22406	当前用户不在该群组中
22408	当前用户在群组中已被禁言
23406	当前用户不在该聊天室中
23408	当前用户在该聊天室中已被禁言
23409	当前用户已被踢出并禁止加入聊天室。被禁止的时间取决于服务端调用踢出接口时传入的时间
23410	聊天室不存在
23411	聊天室成员超限，默认聊天室成员没有人数限制，但是开发者可以提交工单申请针对 App Key 进行聊天室人数限制

状态码	描述
23412	聊天室接口参数无效
23414	聊天室云存储业务未开通
23423	超过聊天室的最大状态设置数，1 个聊天室默认最多设置 100 个
23424	聊天室中非法覆盖状态值，状态已存在，没有权限覆盖
23425	超过聊天室中状态设置频率，1 个聊天室 1 秒钟最多设置和删除状态 100 次
23426	未开通聊天室属性自定义设置。请在控制台免费基础功能页面开通聊天室属性自定义设置。
23427	聊天室状态值不存在
25101	撤回消息参数无效，请确认撤回消息参数是否正确的填写
25102	未开通历史消息云存储。请开通单群聊消息云端存储或聊天室消息云端存储服务。
26001	push 设置参数无效。请确认是否正确的填写了 push 参数
26002	操作跟服务端同步时出现问题，有可能是操作过于频繁所致。如果出现该错误，请延时 0.5s 再试
26004	用户标签个数超限，最多支持添加 20 个标签
29102	公众号默认已关注，针对会话类型：ConversationType_APPSERVICE
29103	公众号已关注，针对会话类型：ConversationType_APPSERVICE
29104	公众号默认已取消关注，针对会话类型：ConversationType_APPSERVICE
29105	公众号已经取消关注，针对会话类型：ConversationType_APPSERVICE
29106	公众号未关注，针对会话类型：ConversationType_APPSERVICE
29201	公众号非法类型，针对会话类型：ConversationType_APPSERVICE

状态码	描述
29202	公众号默认已关注，针对会话类型：ConversationType_PUBLICSERVICE
29203	公众号已关注，针对会话类型：ConversationType_PUBLICSERVICE
29204	公众号默认已取消关注，针对会话类型：ConversationType_PUBLICSERVICE
29205	公众号已经取消关注，针对会话类型：ConversationType_PUBLICSERVICE
29206	公众号未关注，针对会话类型：ConversationType_PUBLICSERVICE
30001	当前连接不可用（连接已经被释放）
30002	当前连接不可用
30003	客户端发送消息请求，融云服务端响应超时
30016	消息大小超限，消息体（序列化成 json 格式之后的内容）最大 128k bytes
31002	请检查您使用的 AppKey 是否正确
31004	Token 无效
31005	App 校验未通过（开通了 App 校验功能，但是校验未通过）
31007	请检查您 App 的 BundleID 是否正确
31008	AppKey 被封禁或已删除
31009	连接失败，一般因为用户已被封禁。请检查您使用的 Token 是否正确，以及对应的 UserId 是否被封禁
31010	当前用户在其他设备上登录，此设备被踢下线
31011	用户在线时被封禁导致连接断开
31023	重连过程中当前用户在其它设备上登录

状态码	描述
31024	连接超过并发限定值
33000	将消息存储到本地数据时失败。发送或插入消息时，消息需要存储到本地数据库，当存库失败时，会回调此错误码。
33001	SDK 没有初始化
33002	数据库错误
33003	开发者接口调用时传入的参数错误
33003	请检查接口调用时传入的参数类型和值
33007	历史消息云存储业务未开通，可以在控制台中开启该服务
33009	聊天室被重置
33100	标签不存在
33101	标签已存在
33102	会话中不存在对应标签
34001	连接已存在，或正在重连中
34002	小视频时间长度超出限制，默认小视频时长上限为 2 分钟
34003	GIF 消息文件大小超出限制，默认 GIF 文件大小上限是 2 MB
34004	聊天室状态未同步完成
34005	连接环境不正确（融云公有云 SDK 无法连接到私有云环境）
34006	当调用 connectWithToken 接口，timeLimit 为有效值时，SDK 在 timeLimit 时间内还没连接成功返回此错误
34007	查询的公共服务信息不存在

状态码	描述
34008	消息不能被扩展
34009	消息扩展失败
34010	消息扩展大小超出限制，默认消息扩展字典 key 长度不超过 32，value 长度不超过 64，单次设置扩展数量最大为 20，消息的扩展总数不能超过 300
34011	媒体消息媒体文件 http 上传失败
34012	指定的会话类型不支持标签功能，会话标签仅支持单群聊会话、系统会话
34013	批量处理指定标签的会话个数超限，批量处理会话个数最大为 1000
34014	群已读回执版本不支持
34015	视频消息压缩失败

音视频

状态码	描述
40001	不在 RTC room 中
40002	server 内部错误
40003	没有匹配的 RTC room
40004	非法的用户 id
40005	重复加入已经存在的 RTC room
50000	初始化失败，信令服务（IM Server）未连接
50001	参数错误
50002	加入相同房间错误，表示用户在客户端重复加入相同的房间
50003	不在房间中
50004	请检查是否开通音视频服务
50006	RTC token为空，请查看是否还在房间内或者房间是否销毁
50007	SDK 内部状态错误
50008	观众加聊天室成功，kv 回调超时(30s)
50009	观众加聊天室调用 api 失败
50010	HTTP 请求超时
50011	HTTP 错误（含 500，404，405 等错误）
50020	发布重复资源
50021	设置本地 SDP 错误

状态码	描述
50022	设置远端 SDP 错误
50023	发布的流的个数已经到达上限
50024	取消发布不存在的资源
50025	创建本地 Offer 失败
50026	创建本地 Answer 失败
50030	订阅不存在的音视频资源
50031	资源重复订阅
50032	取消订阅不存在的音视频资源
50065	RTConnection 为空
50069	解析 Json 串出错
50074	未加入主房间
50075	操作的副房间号码和主房间号码一致错误
50076	取消的跨房间连麦请求不存在
50077	响应的跨房间连麦请求不存在
50079	发布时mediaServer返回的mcu流为空
50080	cdn 地址配置数量到达上限（最大为10个）
50081	帧时间戳非法
50082	解码视频帧失败
50090	音效文件数量已经到达最大数量
50091	处理非法的 soundId，如停止播放没有播放过的音效文件 id，此音效 ID 没有预设或者播放过
50100	自动重连异常
50101	观众加聊天室成功，pullData KV 没有值
51202	订阅流时，cdn流订阅失败
52000	音频设备启动失败
54001	没有集成player SDK
54002	CDN内部错误
54008	数据连接中断 或 音视频源格式不支持
54009	订阅 CDN 流时，初始化 player 模块异常
55001	切换的角色和当前角色相同错误
56002	RTC 网络探测未开始
56003	RTC 网络探测被内部强制停止
56004	与 media server ICE 断开
56005	加房间操作打断探测
56006	SEI 数据长度超出限制 4096 个字节
56007	SEI 通道未建立,请检查是否开启 SEI,或者发布音视频
56008	SEI 发送失败
56009	SEI 频率超出限制, 1秒内不超过 30 次