场景融合

语聊房

iOS 2.X

2024-05-17

语聊房开发指导

更新时间:2024-05-17

欢迎使用融云语聊房 SDK RCVoiceRoomLib。

语聊房 SDK 是基于融云即时通讯（IM）和实时音视频（RTC）优势能力封装的场景化 SDK，参考主流语聊房应用功能进行设计，贴近场景，提供精简、高度封装的核心 API 与回调，帮助您降低学习成本，提升开发效率。

语聊房 SDK 支持包括麦位管理、房间管理、多人连麦、跨房间 PK 与混音在内的功能。

客户端 SDK

语聊房客户端 SDK 即 RCVoiceRoomLib，支持开箱即用。配合融云 IM 与 RTC 服务端 API 接口，可构建丰富的业务特性组合。

- **贴近业务：** API 调用和命名贴近语聊房业务场景与客户端功能特性。
- **使用简单：** 核心 API 数量不超过 20 个，核心回调数量不超过 5 个。
- **扩展性强：** 提供了丰富的扩展属性，不管是语音游戏，语音社交，还是 KTV 场景均可覆盖。
- **资源丰富：** 提供文档、教学视频，和全功能的开源语聊房 App demo 项目（RC RTC）。

功能列表

语聊房客户端 SDK RCVoiceRoomLib 的主要功能包括麦位管理、房间管理、多人连麦、跨房间 PK 与混音。更多功能请参见下表（包括但不限于以下内容）：

功能	描述
房间管理	支持从客户端创建、加入、退出房间。
用户上麦	支持房间内用户上指定麦位或自由上麦，最多支持 32 人同房间内连麦。
申请麦序	支持房间内用户申请上麦，App 开发者可根据业务需要实现由任意用户或特定用户处理批准或拒绝上麦。
静音麦位	支持房间内任何用户静音或取消静音任意指定麦位或所有麦位。
锁定麦位	支持房间内任何用户锁定任意指定麦位或所有麦位，其他观众无法上麦。注意，锁麦仅锁定麦位状态，暂不支持将被锁麦位上用户自动下麦。
动态修改麦位数量	在直播过程中，可动态增加或减少麦位数量。注意，修改后所有连麦者自动下麦。
实时监听麦克风音量	监听不同麦位的麦克风音量。
混音支持	支持背景音，伴唱，特效声等混音效果。 注意，需要直接调用 RTCLib 接口。
控制音频质量	内置房间音频质量支持人声、标清音质、高清音质。内置房间场景支持普通通话、音乐聊天室、音乐教室。可动态切换质量与场景，满足教学，K 歌等不同场景需求。
跨房间 1v1 PK	支持两个房间之间两个主播的跨房间 PK，可将对方主播在当前房间内静音。注意，PK 期间不支持房间内连麦。
房间属性可扩展	支持自定义房间扩展属性。注意，该字段建议使用 JSON 格式字符串。

功能	描述
麦位属性可扩展	支持自定义麦位扩展属性。注意，该字段建议使用 JSON 格式字符串。App 开发者可根据狼人杀，相亲房等具体业务场景，自行定义扩展数据结构，用于区分角色等。

> 语聊房示例 App（RC RTC）实现了语聊房 SDK 提供的主要功能，您可查看 主要功能演示 »

语聊房 App 服务端

语聊房客户端 SDK 仅依赖融云提供的 IM 和 RTC 能力。语聊房 App 服务端指您自己的 **App 业务服务器**（App server），即为您自身的业务提供接口的后端，您需要自行实现。

您可以使用 IM 服务端 API 与 RTC 服务端 API 所提供的全部能力构建您的语聊房 App 后台服务系统。

> 了解如何使用 IM + RTC 全部服务端能力，请前往 即时通讯服务端文档 · 实时音视频服务端文档 »

为协助您搭建语聊房 App 业务服务端，我们提供了丰富的参考资源。您可以参考语聊房服务端最佳实践，实现需要客户端和服务端配合的常见产品功能。此外，您可以查看我们提供的语聊房 App 服务端示例项目（Demo server），了解基本功能与最佳实践的具体实现。

> 查看最佳实践，请前往语聊房服务端开发指南 »

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。

> 音视频服务必须要从控制台开通后方可使用。参见控制台文档开通音视频服务 »

资源与支持

- **示例应用项目（Quickdemo）**

  预置了融云 App Key 和对应的测试服务器 URL，无需部署服务器即可运行。用于演示和体验语聊房业务。

  - Android Quickdemo (Github) · iOS Quickdemo (Github) · Web Quickdemo (Github)
  - Android Quickdemo (Gitee) · iOS Quickdemo (Gitee) · Web Quickdemo (Gitee)

- **RC RTC 应用**

  展示如何使用场景化 SDK 搭建语聊房、语音电台、视频直播、语音呼叫、视频呼叫等互动社交场景，提供 Android 端和 iOS 端的源码。

- RC RTC Android 源码 (Github) · RC RTC iOS 源码 (Github)
- RC RTC Android 源码 (Gitee) · RC RTC iOS 源码 (Gitee)

- **RC RTC 配套 App server**

  官网体验应用 RC RTC 配套的 App server，基于 java，提供登录/游客登录、创建房间、上/下麦位、房间音乐、礼物等相关接口。

  - RC RTC 配套 App server (Github)
  - RC RTC 配套 App server (Gitee)

主要功能演示

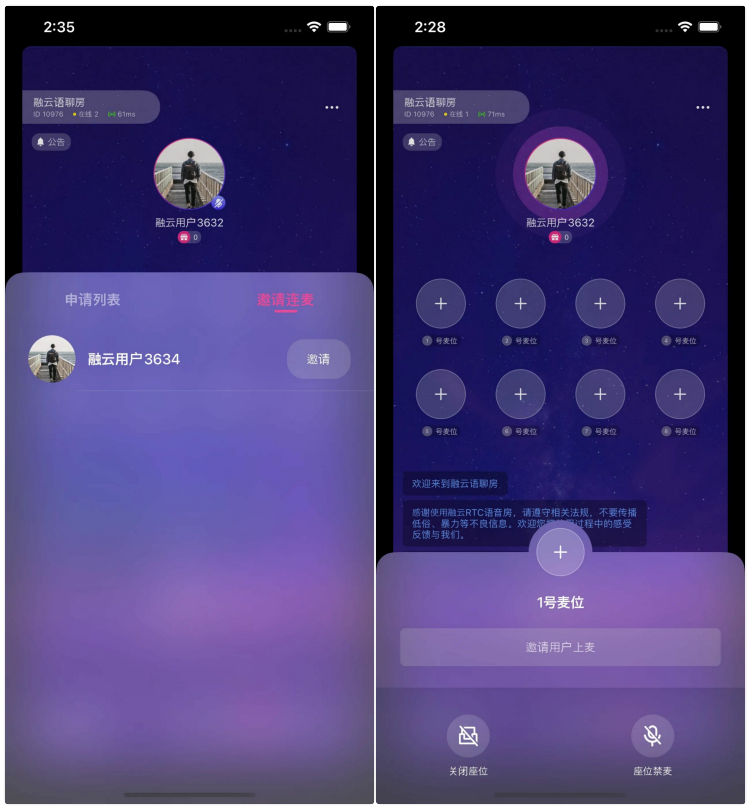
更新时间:2024-05-17

语聊房的主要功能包括麦位管理、房间管理、多人连麦、跨房间 PK 与混音。以下我们使用语聊房示例 App（RC RTC）的界面，简要介绍语聊房 SDK RCVoiceRoomLib 的主要功能。

麦位管理

麦位管理包含丰富的麦位相关操作：

- 闭麦： 关闭特定麦位的麦克风，在此麦位上麦时观众和其他主播听不到该麦位的声音。
- 锁麦： 锁定某麦位，使之无法上麦。
- 邀请上麦： 邀请特定用户上麦。

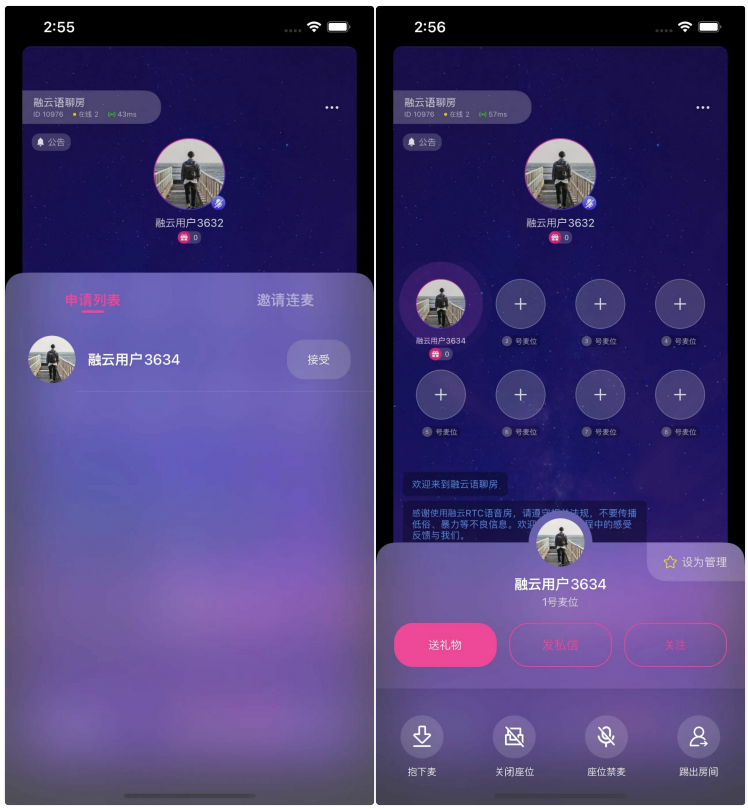


房间管理

语聊房包含丰富的房间属性，可定义扩展等。

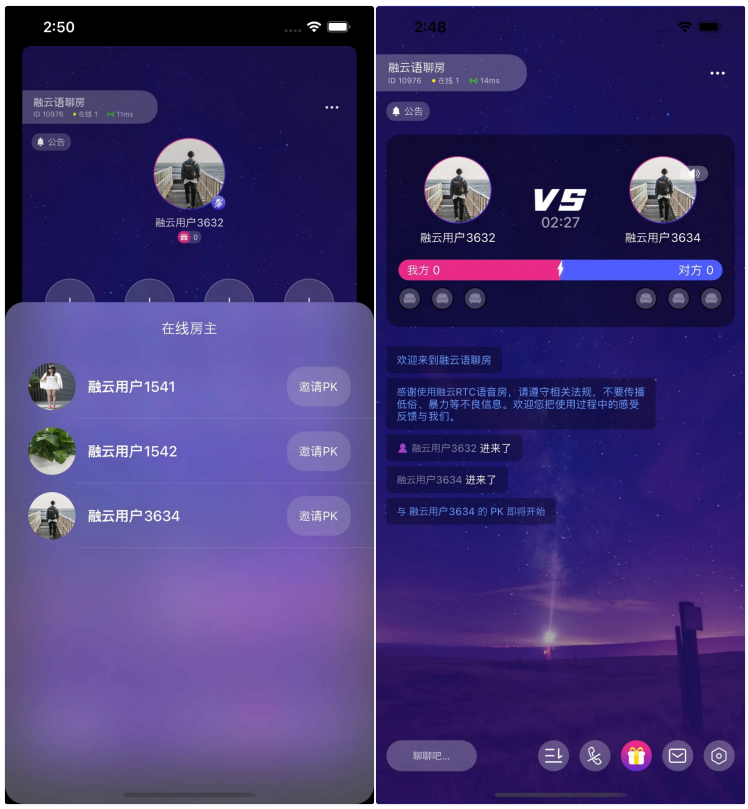
多人连麦

RCVoiceRoomLib支持最多 32 个麦位同时连麦。SDK 支持自由上麦和申请上麦两种上麦机制，并且支持强制下麦和将某个用户踢出房间等操作。



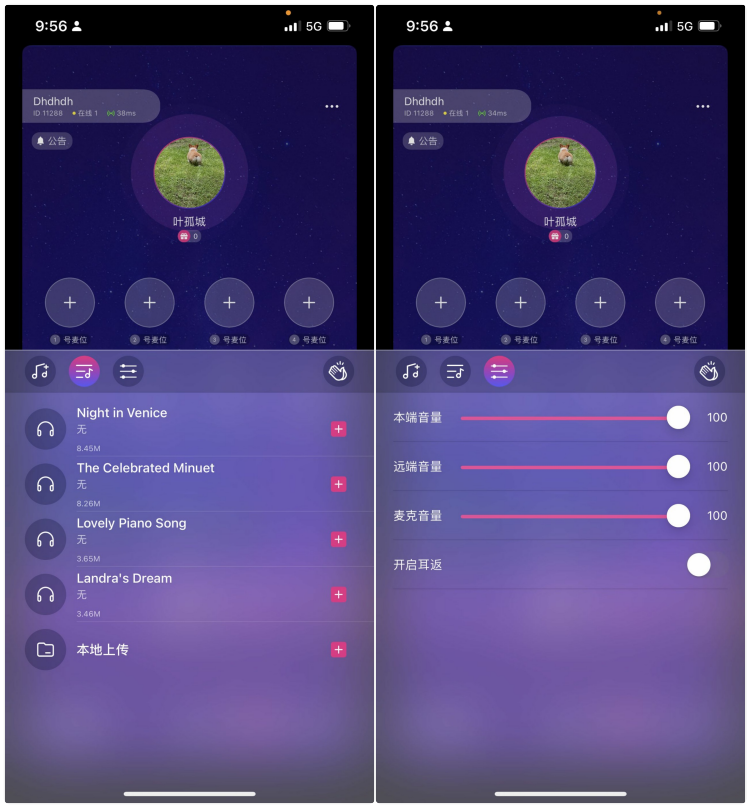
跨房间 PK

语聊房支持不同房间的房主进行 1v1 PK。



混音支持

多种音频场景和质量支持，满足教学，K 歌等不同场景音质。



开通服务 必须开通的服务

更新时间:2024-05-17

您在融云创建的应用默认不会启用语聊房业务依赖的所有服务。在使用语聊房 SDK，必须开通以下全部服务。

> 服务开通、关闭等设置完成后 30 分钟后生效。

语聊房 SDK 依赖以下业务：

业务类型	依赖说明	控制台服务名称	控制台链接
音视频通话	建立音视频通信	音视频通话	开通音视频通话服务
音视频直播	使用直播业务	音视频直播[注1]	开通音视频直播服务
聊天室自定义属性	依赖即时通讯（IM）的聊天室属性功能	聊天室自定义属性设置[注1]	开通聊天室自定义属性设置

注[1]：开通音视频直播服务前，需要先开通音视频通话服务。

免费体验时长

针对音视频服务，在开发环境下创建的每个应用均可享有 10000 分钟免费体验时长。

快速集成 环境要求

更新时间:2024-05-17

- **Xcode**：确保与苹果官方同步更新
- **CocoaPods**：1.10.0 及以上
- **iOS**：11.0 及以上
- **Objective-C**：2.0

开通音视频服务

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。

使用语聊房业务要求开通「音视频直播」服务，具体步骤请参阅 开通音视频服务 。

> 服务开通、关闭等设置完成后 30 分钟后生效。

使用 pod 集成语聊房 SDK

1. 在项目的 **Podfile** 中添加

```
pod 'RCVoiceRoomLib', '2.0.4'
```

2. 在终端中运行以下命令：

```
pod install
```

3. **Pod** 安装完成后，CocoaPods 会在您的工程根目录下生成一个 .xcworkspace 文件。您需要通过此文件打开您的工程，而不是之前的 .xcodeproj

集成后的包大小

集成语聊房 SDK 后，会增加一定的包大小。

- iOS 平台： 增量大约 4 MB

- Android 平台： 增量大约 7 MB

版本依赖说明

RCVoiceRoomLib 依赖融云 IMLib 与 RTCLib，依赖版本如下。使用 CocoaPods 安装时会自动处理依赖关系。

依赖组件	版本
IMLib	5.1.4 及以上
RTCLib	5.1.8 及以上

初始化

更新时间:2024-05-17

本节介绍如何快速初始化语聊房 SDK。

> 在初始化前，请确保已完成以下操作：
>
> - 您已开通融云开发者账号，并申请了融云 App Key。
> - 您已为 App Key 开通音视频服务。使用语聊房业务要求开通「音视频直播」服务。

初始化

RCVoiceRoomLib 依赖融云即时通讯 SDK 提供消息通讯能力。因此，需要使用即时通讯 SDK 的初始化方法。

- 如果您的项目未集成任何融云 SDK，请使用 IMLib 的初始化方法。请在 `application:didFinishLaunchingWithOptions:` 中执行下列方法初始化：

```
#import <RongIMLib/RongIMLib.h>
/// 使用 **IMLib** 初始化
/// appkey 即您申请的 appkey，需要开通音视频直播服务
/// token一般是您在登录自己的业务服务器之后，业务服务器返回给您的，可存在本地。
[[RCIMClient sharedRCIMClient] initWithAppKey:appkey];
[[RCIMClient sharedRCIMClient] connectWithToken:token timeLimit:5
dbOpened:^(RCDBErrorCode code) {
//消息数据库打开，可以进入到主页面
} success:^(NSString *userId) {
//连接成功
} error:^(RCConnectErrorCode status) {
if (status == RC_CONN_TOKEN_INCORRECT) {
//Token 错误，可检查客户端 SDK 初始化与 App 服务端获取 Token 时所使用的 App Key 是否一致
} else if(status == RC_CONNECT_TIMEOUT) {
//连接超时，弹出提示，可以引导用户等待网络正常的时候再次点击进行连接
} else {
//无法连接 IM 服务器，请根据相应的错误码作出对应处理
}
}];
```

- 如果您的项目已集成融云 IMLib 或 IMKit SDK，您无需做任何更改。

房间管理

更新时间:2024-05-17

语聊房 SDK 内部依赖即时通讯聊天室房间（ChatRoom）与音视频基础能力业务的音视频房间（RTCRoom）。

语聊房的生命周期依赖并完全等价于聊天室的生命周期（ChatRoom），受聊天室自动销毁机制影响，可通过保活机制控制销毁时机。

> 语聊房相关操作主要包括：
>
> 设置代理 · 创建房间 · 加入房间 · 离开房间 · 踢出房间 · 销毁房间

设置代理

创建一个语聊房首先需要设置设置用于接收语聊房业务的所有回调的代理。这样创建房间后监听才会全部回调。
连麦，麦位管理，PK 等涉及的业务回调都依赖这个代理设置，因此非常重要！

代码示例

在viewDidLoad里一般会这样设置。

```
- (void)viewDidLoad {
[super viewDidLoad];

// 设置语聊房代理
[RCVoiceRoomEngine.sharedInstance setDelegate:self];
}
```

创建房间

右边是内部大概的执行、回调顺序。

配置房间信息

创建语聊房房间信息，包含房间名称、麦位数量等必要配置。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 创建一个 RCVoiceRoomInfo 实例
RCVoiceRoomInfo *roomInfo = [[RCVoiceRoomInfo alloc] init];
/// 设置房间名称
roomInfo.roomName = roomName;
/// 设置麦位数量
roomInfo.seatCount = 9;
```

创建房间

获取房间 ID 之后调用 RCVoiceRoomLib 的 createAndJoinRoom 方法。

```
// roomId 是您的业务服务器返回的
[[RCVoiceRoomEngine sharedInstance] createAndJoinRoom:roomId room:roomInfo success:^{
/// 创建成功后会自动加入语聊房
[SVProgressHUD showSuccessWithStatus:@"创建成功"];
} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
[SVProgressHUD showSuccessWithStatus:@"创建失败"];
}];
```

房间绑定

聊天室与音视频房间绑定成功后，只要音视频房间仍存在，则阻止上面描述的聊天室自动销毁的情况。
详见参考链接绑定音视频房间。

```
#import "RongIMLib/RCIMClient.h"
[[RCChatRoomClient sharedChatRoomClient] bindChatRoom:@"" withRTCRoom:@"" success:^{

} error:^(RCErrorCode nErrorCode) {

}]
```

加入房间

右边是内部大概的执行流程。

房间列表

可以通过您的业务服务器接口获取。

```
[WebService getRoomListWithSuccess:^(id responseObject) {
/// 得到对应的房间信息（房间id等）列表，用作业务UI层展示
} failure:^(NSError * _Nonnull error) {

}];
```

调用加入房间接口

调用 JoinRoom 接口加入房间，该接口要求语聊房已创建。

申请加入语聊房的用户需要提供房间 ID。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 这里的roomId是通过您获取自己业务服务器接口拿到的。
- (void)joinVoiceRoom:(NSString *)roomId {
[[RCVoiceRoomEngine sharedInstance] joinRoom:roomId success:^{
[SVProgressHUD showSuccessWithStatus:@"加入房间成功"];
} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
[SVProgressHUD showSuccessWithStatus:@"加入房间失败"];
}];
}
```

创建或加入语聊房后的回调

加入或者创建语聊房成功后，会依次触发同样的一些回调，如下：

房间信息更新回调

roomInfoDidUpdate 会在任何人修改房间属性时触发。

您可以在此回调处理房间属性相关的 UI 刷新。

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

/// 任何房间信息的修改都会触发此回调。
- (void)roomInfoDidUpdate:(RCVoiceRoomInfo *)roomInfo {
self.roomInfo = roomInfo;
[self updateRoomInfoView];
}
```

麦位信息变化回调

seatInfoDidUpdate 会在任一麦位变化时触发。返回值为麦位数组，包含了当前最新的麦位信息。

您可以在此回调处理所有麦位变动相关的 UI 刷新。

```
/// 任何麦位的变化都会触发此回调。
- (void)seatInfoDidUpdate:(NSArray<RCVoiceSeatInfo *> *)seatInfolist {
// 保存最新麦位信息，并刷新UI。
self.seatlist = seatInfolist;
[self.collectionView reloadData];
}
```

KV 准备完毕回调

roomKVDidReady 触发证明您的语聊房已经初始化完成。

部分语聊房有房主加入房间自动上麦的业务逻辑，可在该回调中完成。例如在该回调触发时，房主调用上麦方法进行上麦。

```
/// 房间信息初始化完毕，可在此方法进行一些初始化操作，例如进入房间房主自动上麦等
- (void)roomKVDidReady {
// 如果需要进入房间自动上麦，可在此方法中调用enterSeat
}
```

离开房间

主播/观众离开语聊房间，不再发布/收听语音。

调用离开方法

调用 leaveRoom 接口。

注意：调用该方法时，内部会先将主播下麦。如果主播此时正在麦位上，会自动下麦。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
[[RCVoiceRoomEngine sharedInstance] leaveRoom:^{
} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
}];
```

回调处理

通常在成功回调里，做移除 Controller 之类的销毁处理，以及其他业务处理代码。

---

iOS 目前的回调在子线程，因此如需处理 UI 相关的操作，需要回到主线程。

```
dispatch_async(dispatch_get_main_queue(), ^{
[SVProgressHUD showSuccessWithStatus:@"离开房间成功"];
[self popSelfToLeave];
});
dispatch_async(dispatch_get_main_queue(), ^{
[SVProgressHUD showErrorWithStatus:[NSString stringWithFormat:@"离开房间失败 code: %ld",(long)code]];
});
```

踢出房间

语聊房 SDK 将用户踢出房间的功能是通过发送消息实现的，需要业务逻辑配合处理：

调用踢出房间接口

传入 userId，用于指定需要被踢出语聊房的用户。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 将某个用户踢出房间
/// @param userId 踢出房间的userId
/// @param successBlock 成功回调
/// @param errorBlock 失败回调
[[RCVoiceRoomEngine sharedInstance] kickUserFromRoom:@"userId" success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

被踢的一方用户回调

被踢出的回调里，执行离开房间方法。

```


/// 用户被踢出房间的回调
/// @param targetId 被踢出房间的用户id
/// @param userId 执行踢某人出房间的用户id
- (void)userDidKickFromRoom:(nonnull NSString *)targetId byUserId:(nonnull NSString *)userId {
[[RCVoiceRoomEngine sharedInstance] leaveRoom:^{
dispatch_async(dispatch_get_main_queue(), ^{
[SVProgressHUD showSuccessWithStatus:@"离开房间成功"];
[self popSelfToLeave];
});

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
dispatch_async(dispatch_get_main_queue(), ^{
[SVProgressHUD showErrorWithStatus:[NSString stringWithFormat:@"离开房间失败 code: %ld",(long)code]];
});
}];
}
```

销毁房间

默认情况下，聊天室房间（ChatRoom）一个小时内无人加入且无新消息，就会触发自动销毁。音视频房间（RTCRoom）只要没人立即销毁。详见参考链接房间销毁。

房间销毁后离开房间

在房间关闭 roomDidClosed 回调里，调用离开房间的接口，如右边所示：

在房间销毁回调里必须主动调用离开房间接口。

```objc
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 房间已关闭
- (void)roomDidClosed {
[[RCVoiceRoomEngine sharedInstance] leaveRoom:^{
dispatch_async(dispatch_get_main_queue(), ^{
[SVProgressHUD showSuccessWithStatus:@"离开房间成功"];
[self popSelfToLeave];
});

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {
dispatch_async(dispatch_get_main_queue(), ^{
[SVProgressHUD showErrorWithStatus:[NSString stringWithFormat:@"离开房间失败 code: %ld",(long)code]];
});
}];
}
```

观众连麦

更新时间:2024-05-17

针对连麦，语聊房 SDK RCVoiceRoomLib 提供了上麦和下麦两个操作。您只需要调用函数，刷新 UI 即可。不用关心内部处理流的订阅、麦位的同步等操作的实现。

- 上麦（加入麦位）：调用 `enterSeat` 成功后，用户角色从观众切换为主播，并拥有发布音频流的能力。麦位状态需要更新并且同步语聊房内所有用户以更新 UI 和状态。
- 下麦（离开麦位）：调用 `leaveSeat` 成功后，用户角色从主播切换为观众，同时失去发布流的能力。麦位状态需要更新并且同步语聊房内所有用户以更新 UI 和状态。

> 麦位相关操作主要包括：
>
> - 上麦
>   自由上麦 · 申请上麦 · 邀请上麦
> - 下麦
>   主动离开麦位 · 强制下麦

下麦（离开麦位）

SDK 现拥有下面几种下麦模式：

- **主动离开麦位**
- **强制下麦**

主动离开麦位

离开麦位

调用该方法成功后，角色自动切换，并且自动取消了发布流的操作。该用户角色回归普通用户。

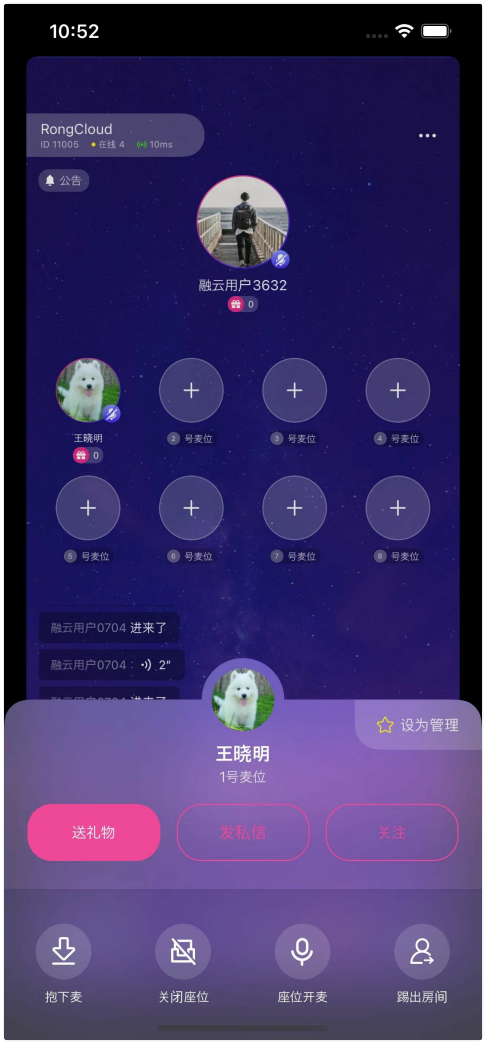
```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 用户主动下麦
/// @param successBlock 下麦成功
/// @param errorBlock 下麦失败
- (void)leaveSeatWithSuccess:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

强制下麦

强制下麦是指房主或其他有权限的用户通知房间里的某个观众强制下麦。通知强制下麦后，该观众会触发回调，需要在回调中调用 leaveSeat。大致流程如右边所示。

UI 表现形式

右边是 UI 演示房主将某麦上用户下麦的界面。



房主/管理员将某个用户下麦

让某个用户下麦离开麦位。

```
/// 将某个麦位的用户下麦
/// @param userId 下麦的用户id
/// @param successBlock 下麦成功
/// @param errorBlock 下麦失败
- (void)kickUserFromSeat:(NSString *)userId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

被强制下麦用户收到通知

被强制下麦后，下麦用户要做离开麦位的处理。

```
/// 收到自己被下麦的通知
- (void)kickSeatDidReceive:(NSUInteger)seatIndex;
```

麦位管理

更新时间:2024-05-17

锁麦和闭麦是语聊房麦位管理中常见的操作。语聊房 SDK RCVoiceRoomLib 将这两种操作封装为两个函数。您不需要管理其他的状态就可实现想要的效果。

> 麦位控制主要包括：
>
> 锁定麦位 · 静音麦位（闭麦）

锁定麦位

锁定某个麦位。被锁上的麦位，任何人均不可使用。假设 1 号麦位被锁定，任何人调用 enterSeat 上 1 号麦时均会返回错误。大致流程如右边所示。

示例

锁麦在 UI 上可体现为麦位已锁定。右边是语聊房 SDK QuickDemo 的 UI 演示 1 号麦位被锁定。

锁定麦位

调用 lockSeat 成功后，所有用户都会收到 seatInfoDidUpdate 回调，在此刷新 UI 即可。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 锁定某个麦位
/// @param seatIndex 麦位序号
/// @param isLocked 是否锁麦位
/// @param successBlock 锁麦成功
/// @param errorBlock 锁麦失败
- (void)lockSeat:(NSUInteger)seatIndex
lock:(BOOL)isLocked
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

静音麦位（闭麦）

闭麦即将某个麦位静音，任何在麦位上的用户说话不会被此房间内的任何人听到。大致流程如右边所示。

示例

下面我们使用语聊房 SDK QuickDemo 的 UI 演示 1 号麦位被闭麦的界面。

静音麦位（闭麦）

调用 muteSeat 成功后，所有用户都会收到 seatInfoDidUpdate 回调，在此刷新 UI 即可。

```
/// 将某个麦位静音
/// @param seatIndex 麦位序号
/// @param isMute 是否静音
/// @param successBlock 静音成功
/// @param errorBlock 静音失败
- (void)muteSeat:(NSUInteger)seatIndex
mute:(BOOL)isMute
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

房间控制

更新时间:2024-05-17

语聊房的一些控制操作可以分为两类：

- 通过即时通讯聊天室属性（`ChatRoom` KV) 实现的功能操作。
- 与音视频房间（`RTCRoom`）相关的控制操作。

> 麦位控制主要包括：
>
> ChatRoom KV 相关的控制操作 · 音视频房间（RTCRoom）相关控制操作

聊天室属性（KV）相关控制操作

全部锁麦

利用 RCVoiceRoomInfo 的 isLockAll 属性 控制语聊房里面所有麦位的锁定状态。

全部锁麦后，所有麦位不能上麦；反之，可以全部解锁。YES 为全部锁定，NO 为全部解除锁定。

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

RCVoiceRoomInfo *roomInfo = self.currentRoomInfo;
roomInfo.isLockAll = YES;
[[RCVoiceRoomEngine sharedInstance] setRoomInfo:roomInfo success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

全部闭麦

利用 RCVoiceRoomInfo 的 isMuteAll 属性，控制语聊房里面所有麦位的打开/关闭状态。

全部闭麦后，所有麦位的上麦主播静音；反之，可以恢复全部为打开状态。YES 为全部闭麦，NO 为全部解除闭麦，即麦克风为打开状态。

```
RCVoiceRoomInfo *roomInfo = self.currentRoomInfo;
roomInfo.isMuteAll = YES;
[[RCVoiceRoomEngine sharedInstance] setRoomInfo:roomInfo success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

锁定其它麦位

锁定除自己之外其余所有的空闲麦位。

```
[[RCVoiceRoomEngine sharedInstance] lockOtherSeats:YES];
```

静音其它麦位

关闭除自己之外其余所有的麦位，使得其他麦位静音。

```
[[RCVoiceRoomEngine sharedInstance] muteOtherSeats:YES];
```

房间信息扩展

如需为语聊房添加与房间信息有关的自定义数据，可以使用此扩展方法。

该自定义数据会被同步给房间内所有人。

```
RCVoiceRoomInfo *roomInfo = self.currentRoomInfo;
roomInfo.extra = @"扩展信息，建议传json字符串";
[[RCVoiceRoomEngine sharedInstance] setRoomInfo:roomInfo success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

房间信息更新回调

调用以上操作 KV 的方法后，房间内其他人都会收到房间信息的改变，可以在这个回调里处理业务逻辑。

```
// 任何房间信息的修改都会触发此回调。
- (void)roomInfoDidUpdate:(RCVoiceRoomInfo *)roomInfo {

}
```

音视频房间（RTCRoom）相关控制操作

麦克风控制

开启/停止麦克风采集音频数据。

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

/// 停止本地麦克风收音
/// @param isDisable 是否停止
- (void)disableAudioRecording:(BOOL)isDisable;
```

扬声器控制

打开/关闭设备扬声器。

```

/// 设置是否使用扬声器
/// @param isEnable 是否使用
- (void)enableSpeaker:(BOOL)isEnable;
```

静音全部流

静音远端传过来的音频流。

```

/// 静音所有远程音频流
/// @param isMute 是否静音所有远程音频流
- (void)muteAllRemoteStreams:(BOOL)isMute;
```

音频质量和模式

根据自己业务的使用场景，设置语聊房音频质量以及不同的模式。

```
/// 设置房间音频质量和场景
/// @param quality 音频质量
/// @param scenario 音频场景
- (void)setAudioQuality:(RCVoiceRoomAudioQuality)quality
scenario:(RCVoiceRoomAudioScenario)scenario;
```

房间内通知

更新时间:2024-05-17

为了方便用户同步房间的一些操作（类似更改房间背景）我们设计了方便的房间通知接口，用于通知语聊房内所有用户。

房间通知的流程如下：

发送房间通知

房间通知实质上就是自定义一个消息体，这个消息体包含两个部分：消息名字和内容。

发送通知消息

您需要定义通知的 Name 与 Content。

例如，房主修改房间背景图片之后，需要房间其他用户刷新房间背景图，那么可以调用下述方法。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 通知房间所有用户消息
/// @param name 消息名称
/// @param content 内容
- (void)notifyVoiceRoom:(NSString *)name
content:(NSString *)content;

/// 如房主修改房间背景图片之后，要同步给所有房间内的用户一起刷新
[[RCVoiceRoomEngine sharedInstance] notifyVoiceRoom:@"refreshBackgroundImage" content:@"ImageUrl"];
```

房间内所有成员收到回调

发送通知成功后，房间其他人会触发 roomNotificationDidReceive 回调。

可在该回调中进行判断，并进行对应的操作。

```
/// 收到房间通知
/// @param name 通知的名称
/// @param content 通知的内容
- (void)roomNotificationDidReceive:(NSString *)name
content:(NSString *)content;
```

跨房间 PK

更新时间:2024-05-17

跨房间 PK 指不在同一个语聊房的主播进行跨房间连麦，目前支持 1 v 1 PK。

> 发起 PK 邀请的房间必须存在，用户必须在线。

下面我们使用语聊房 SDK QuickDemo 的 UI 展示一个典型的跨房间 PK 页面。

跨房间 PK 的大致流程如下。

> PK相关的操作主要包括：
>
> 发起 PK 邀请 · 取消 PK 邀请 · 响应 PK 邀请 · 静音 PK 对象 · 恢复 PK

发起 PK 邀请

跨房间 PK 的邀请者（简称为 Inviter）需要发送 PK 邀请给受邀者（简称为 Invitee）。SDK 内部会调用 RTCLib 提供的接口，具体行为是邀请者申请加入其它房间（在这里就是受邀者的音视频房间）。

邀请者发起邀请

发起邀请时，必须传入受邀者的房间 ID，受邀者的用户 ID。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 发送 PK 邀请
/// @param inviteeRoomId 被邀请用户所在的房间 id
/// @param inviteeUserId 被邀请人的用户id
/// @param successBlock 邀请发送成功回调
/// @param errorBlock 邀请发送失败回调
- (void)sendPKInvitation:(NSString *)inviteeRoomId
invitee:(NSString *)inviteeUserId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

受邀者收到通知

邀请发送成功后，Invitee 会触发回调。

```
/// 收到邀请 PK 的回调
/// @param inviterRoomId 邀请者的房间id
/// @param inviterUserId 邀请者的用户id
- (void)pkInvitationDidReceiveFromRoom:(NSString *)inviterRoomId byUser:(NSString *)inviterUserId;
```

邀请者取消 PK 邀请

邀请者（Inviter）可以取消 PK 邀请，即撤回之前对某个受邀者（Invitee）的 PK 邀请。

取消 PK 邀请

邀请者撤回邀请时，必须传入受邀者的房间 ID，受邀者的用户 ID。

```objc
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 撤回已发送的PK邀请
/// @param inviteeRoomId 被邀请用户所在的房间id
/// @param inviteeUserId 被邀请人的用户id
/// @param successBlock 邀请撤回发送成功
/// @param errorBlock 邀请撤回发送失败
- (void)cancelPKInvitation:(NSString *)inviteeRoomId
invitee:(NSString *)inviteeUserId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

受邀者收到取消通知

取消邀请成功后，受邀者就会收到通知，可以根据业务做一些处理。

```objc
/// 邀请者取消 PK 邀请回调
/// @param inviterRoomId 邀请者的房间id
/// @param inviterUserId 邀请者的用户id
- (void)cancelPKInvitationDidReceiveFromRoom:(NSString *)inviterRoomId byUser:(NSString *)inviterUserId;
```

受邀者响应 PK 邀请

受邀者（Invitee）收到邀请后可以同意，拒绝，忽略邀请，该响应会回复给 Inviter。

被邀请者响应

Invitee 通过传入不同 RCPKResponseType 来向 Inviter 发送不同的回复。

- `RCPKResponseAgree`：接受 PK 邀请
- `RCPKResponseReject`：拒绝 PK 邀请
- `RCPKResponseIgnore`：忽略 PK 邀请

```objc
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 回复邀请人是否接受邀请
/// @param inviterRoomId 邀请人所在的房间 id
/// @param inviterUserId 邀请人的用户id
/// @param type 回应邀请者的类型，接受，拒绝或者忽略
/// @param successBlock 回复发送成功
/// @param errorBlock 回复发送失败
- (void)responsePKInvitation:(NSString *)inviterRoomId
inviter:(NSString *)inviterUserId
responseType:(RCPKResponseType)type
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

邀请者收到同意 PK

Inviter 收到同意 PK 的回复后，PK 就会开始。

```
/// PK 运行的回调，如果 PK 连接成功，或者进入正在进行PK的房间均会触发此回调
/// @param inviterRoomId 邀请 PK 的用户所在房间id
/// @param inviterUserId 邀请 PK 的用户id
/// @param inviteeRoomId 被邀请 PK 的用户所在房间id
/// @param inviteeUserId 被邀请 PK 的用户id
- (void)pkOngoingWithInviterRoom:(NSString *)inviterRoomId
withInviterUserId:(NSString *)inviterUserId
withInviteeRoom:(NSString *)inviteeRoomId
withInviteeUserId:(NSString *)inviteeUserId;
```

邀请者收到拒绝 PK

如果 Invitee 的响应是拒绝 PK 邀请，Inviter 会收到通知。

```
/// 被邀请者拒绝 PK 邀请回调
/// @param inviteeRoomId 被邀请者的房间 id
/// @param initeeUserId 被邀请者的用户 id
- (void)rejectPKInvitationDidReceiveFromRoom:(NSString *)inviteeRoomId byUser:(NSString *)initeeUserId;
```

邀请者收到忽略 PK

如果 Invitee 的响应是忽略 PK 邀请，Inviter 会收到通知。

```
/// 邀请者忽略 PK 邀请回调
/// @param inviteeRoomId 被邀请者的房间id
/// @param inviteeUserId 被邀请者的用户id
- (void)ignorePKInvitationDidReceiveFromRoom:(NSString *)inviteeRoomId byUser:(NSString *)inviteeUserId;
```

静音 PK 对象

PK 过程中可以控制关闭/打开对面主播的声音，SDK 内部是通过调用 RTC 音频合流配置实现。

设置 PK 对象静音

调用 `mutePKUser` 设置 PK 对象为静音状态。

`isMute` 可以为 YES/NO，对应是否屏蔽的情况。

```objc
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 屏蔽PK对象的语音
/// @param isMute 是否屏蔽
/// @param successBlock 调用成功
/// @param errorBlock 调用失败
- (void)mutePKUser:(BOOL)isMute
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

恢复 PK

恢复 PK 一般用于解决异常情况，如 APP 被杀进程，通常需要配合客户自己的业务服务端实现。内部实现是重新加入 PK 对方的音视频房间。

方法调用

恢复 PK 时，需要从自己的业务服务器上获取被中断的 PK 过程的如下信息：

- `inviterUserId`：邀请者用户 ID
- `inviterRoomId`：邀请者所在房间 ID
- `inviteeUserId`：受邀请者用户 ID
- `inviteeRoomId`：受邀请者所在房间 ID

```objc
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 恢复跨房间 PK
/// @param info PK 的信息
/// @param successBlock 恢复成功
/// @param errorBlock 恢复失败
- (void)resumePKWithPKInfo:(RCVoicePKInfo *)info
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

结束 PK

Inviter 和 Invitee 都可以发起结束 PK 的操作，但只能由一方发起，两者同时发起会报错。

调用结束方法

Inviter 或 Invitee 调用 `quitPK` 结束 PK 方法，可以终止 PK 。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
/// 退出PK
/// @param successBlock 退出成功
/// @param errorBlock 退出失败
- (void)quitPK:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

结束回调通知

成功调用 quitPK 后，对方收到 PK 结束的回调。

本端调用结束 PK 方法不会收到 pkDidFinish 回调。

```
/// 对方结束PK时会触发此回调
/// 收到该回调后会自动退出 PK 连接
- (void)pkDidFinish;
```

扩展功能

更新时间:2024-05-17

语聊房的提供一些扩展接口，用于一些特定需求，增强 SDK 接口功能的丰富性。

通过类扩展 RCVoiceRoomEngine (Plugin) 方式提供。

上麦（扩展）

`enterSeat` 可实现上麦的同时可修改目标麦位的 mute 属性 & extra 属性。

构建麦位信息

构建空麦位，并赋值 mute & extra 属性。

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

RCVoiceSeatInfo *seat = [[RCVoiceSeatInfo alloc] init];
seat.mute = NO;
seat.extra = @"lala";
```

上麦

上麦，并携带以上构建的麦位信息。

```

[[RCVoiceRoomEngine sharedInstance] enterSeat:index seatInfo:seat success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

麦位更新（扩展）

`updateSeat` 可实现修改目标麦位的 mute 属性 & extra 属性。

构建麦位信息

构建空麦位，并赋值 mute & extra 属性。注意：

- 如不修改 mute 属性，将会以默认值(false)赋值给目标麦位。

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

RCVoiceSeatInfo *seat = [[RCVoiceSeatInfo alloc] init];
seat.mute = NO;
seat.extra = @"lala";
```

更新

调用更新 api。

```
[[RCVoiceRoomEngine sharedInstance] updateSeatInfo:0 seatInfo:seat success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

切换麦位（扩展）

- `switchSeatTo(index,preSeat,targetSeat,callback)` 可实现切麦，且同时修改当前麦位和目标麦位的 mute 属性 & extra 属性。
- `switchSeatTo(index,switchMute,switchExtra,callback)` 可实现切麦，且可将当前麦位的 mute 属性 & extra 属性 携带到目标麦位，并将当前麦位的 mute 重置为 false ，extra 重置未null。

切换麦位并更新

构建空麦位，并赋值 mute & extra 属性。

```
#import <RCVoiceRoomLib/RCVoiceRoomLib.h>
// 重置当前麦位的 mute & extra
RCVoiceSeatInfo *preSeat = [[RCVoiceSeatInfo alloc] init];
preSeat.mute = YES;
preSeat.extra = @"preSeat extra info";

// 修改 targetSeatInfo 的 mute & extra
RCVoiceSeatInfo *targetSeat = [[RCVoiceSeatInfo alloc] init];
targetSeat.mute = NO;
targetSeat.extra = @"targetSeat extra info";

[[RCVoiceRoomEngine sharedInstance] switchSeatTo:index preSeat:preSeat targetSeat:targetSeat success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

切换麦位携带属性

切麦，并更新麦位信息。

```
/// 跳麦方法：用户已在麦位上，想切换麦位时调用
/// switchMute YES：跳麦后会把旧麦位信息的 mute 值赋给新麦位，同时旧麦位信息的 mute 置为 NO； NO: 不转移赋值
/// switchExtra 跳麦后会把旧麦位信息的 extra 值赋给新麦位，同时旧麦位信息的 extra 置为null； NO: 不转移赋值
[[RCVoiceRoomEngine sharedInstance] switchSeatTo:index switchMute:NO switchExtra:NO success:^{

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

状态相关（扩展）

- 是否启用SDK 麦位锁 KV ，默认：不启用；
- 是否启用内置的基于 KV 分发 PK 用户的音量，默认：不启用
- 清理异常麦位状态KV
- 麦克风禁用状态

启用麦位状态KV

`setSeatPlaceHolderStateEnable` 可修改 SDK 内部是否启用麦位的状态 KV 方案来解决多端同时抢麦的问题，SDK 默认不启用，注意：

- 1、若不启用，多用户同时抢麦时会出现最后一位抢麦的用户成功的问题。
- 2、若启用，服务端需要实现「异常退出的场景」的最佳实践，并移除状态锁的 KV 对。Key 示例：RCSPlaceHolderKey_seat_+ 麦位索引

```
[[RCVoiceRoomEngine sharedInstance] setSeatPlaceHolderStateEnable:YES];
```

清理异常麦位状态KV

`clearSeatState` 可清除当前异常的麦位状态KV，建议在 joinRoom 成功回调后执行，清除可能存在的因异常情况产生的麦位状态KV。

```
[[RCVoiceRoomEngine sharedInstance] clearSeatState:^(NSArray<NSString *> * _Nonnull clearKeys) {

} error:^(RCVoiceRoomErrorCode code, NSString * _Nonnull msg) {

}];
```

语聊房房间模型 RCVoiceRoomInfo

更新时间:2024-05-17

```objc
@interface RCVoiceRoomInfo : NSObject<NSCopying>

/// 房间名称，不可为空
@property (nonatomic, copy) NSString *roomName;

/// 房间麦位数量，不可为空
@property (nonatomic, assign) NSInteger seatCount;

/// 是否自由上麦
@property (nonatomic, assign) BOOL isFreeEnterSeat;

/// 房间麦位静音状态，状态标记，直接修改不会自动触发静音麦位操作
@property (nonatomic, assign) BOOL isMuteAll;

/// 房间麦位锁定状态，状态标记，直接修改不会自动触发锁麦操作
@property (nonatomic, assign) BOOL isLockAll;

/// 自定义属性，可以传入json等
@property (nonatomic, copy, nullable) NSString *extra;

- (instancetype)init;

/// 类方法，根据jsonString生成一个RCVoiceRoomInfo实例
/// @param jsonSting RCVoiceRoomInfo的jsonSting
+ (RCVoiceRoomInfo *)modelWithJSON:(NSString *)jsonSting;

/// 自动生成该类对应的json字符串
- (NSString *)jsonString;

@end
```

语聊房麦位模型 RCVoiceSeatInfo

更新时间:2024-05-17

```
/// 麦位状态是互斥的
typedef NS_ENUM(NSUInteger, RCSeatStatus) {
RCSeatStatusEmpty = 0,
RCSeatStatusUsing = 1,
RCSeatStatusLocking = 2
};

@interface RCVoiceSeatInfo : NSObject<NSCopying>

/// 麦位状态，分为空置，有人使用，被锁定三种状态
@property (nonatomic, assign) RCSeatStatus status;

/// 是否静音
@property (nonatomic, assign, getter=isMuted) BOOL mute;

/// 若麦位被人使用，则userId为使用麦位的用户id
@property (nonatomic, copy, nullable) NSString *userId;

/// 其他信息
@property (nonatomic, copy, nullable) NSString *extra;

- (instancetype)init;

/// 类方法，根据jsonString生成一个RCVoiceSeatInfo实例
/// @param jsonSting RCVoiceSeatInfo的jsonSting
+ (RCVoiceSeatInfo *)modelWithJSON:(NSString *)jsonSting;

/// 自动生成该类对应的json字符串
- (NSString *)jsonString;

@end
```

语聊房功能列表 RCVoiceRoomEngine

更新时间:2024-05-17

```
@interface RCVoiceRoomEngine : NSObject

/// 语聊房SDK的单例
+ (RCVoiceRoomEngine *)sharedInstance;

/// 设置语聊房的Delegate
/// @param delegate 语聊房delegate
- (void)setDelegate:(id<RCVoiceRoomDelegate>)delegate;

/// 是否开启麦位锁定，用户防止多用户抢麦（2.1.1.1新增）
/// @param seatPlaceHolderStateEnable 是开启
- (void)setSeatPlaceHolderStateEnable:(BOOL)seatPlaceHolderStateEnable;

@end

@interface RCVoiceRoomEngine (Room)

/// 创建并加入语聊房
/// @param roomId 语聊房Id
/// @param roomInfo 语聊房信息，语聊房名字和麦位数量不可为空
/// @param successBlock 创建成功回调
/// @param errorBlock 创建失败回调
- (void)createAndJoinRoom:(NSString *)roomId
room:(RCVoiceRoomInfo *)roomInfo
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 用户根据roomId加入一个语聊房
/// @param roomId 房间id
/// @param successBlock 加入成功回调
/// @param errorBlock 加入失败回调
- (void)joinRoom:(NSString *)roomId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 离开房间
/// @param successBlock 离开房间成功
/// @param errorBlock 离开房间失败
- (void)leaveRoom:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 设置房间信息，房间的id必须与当前房间id一致
/// @param roomInfo 修改的房间信息
/// @param successBlock 设置成功
/// @param errorBlock 设置失败
- (void)setRoomInfo:(RCVoiceRoomInfo *)roomInfo
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 当前所在房间
- (nullable NSString *)roomId;

@end
```

```objc
@interface RCVoiceRoomEngine (Seat)

/// 获取最新的麦位信息
/// @param successBlock 麦位信息列表回调
/// @param errorBlock 失败回调
- (void)getLatestSeatInfo:(void (^)(NSArray<RCVoiceSeatInfo *>*))successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 用户主动上麦
/// @param seatIndex 麦位序号
/// @param successBlock 上麦成功
/// @param errorBlock 上麦失败
- (void)enterSeat:(NSUInteger)seatIndex
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 用户主动下麦
/// @param successBlock 下麦成功
/// @param errorBlock 下麦失败
- (void)leaveSeatWithSuccess:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 用户跳麦，在用户已经在麦位想切换麦位时调用
/// @param seatIndex 需要跳转的麦位序号
/// @param successBlock 跳麦成功
/// @param errorBlock 跳麦失败
- (void)switchSeatTo:(NSUInteger)seatIndex
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 抱用户上麦
/// @param userId 用户id
/// @param successBlock 抱麦成功
/// @param errorBlock 抱麦失败
- (void)pickUserToSeat:(NSString *)userId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 将某个麦位的用户下麦
/// @param userId 下麦的用户id
/// @param successBlock 下麦成功
/// @param errorBlock 下麦失败
- (void)kickUserFromSeat:(NSString *)userId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 清理麦位锁状态（2.1.1.1新增）
/// @param clearCompletion 成功回调
/// @param errorBlock 失败回调
- (void)clearSeatState:(RCSVoiceClearCompletion)clearCompletion
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 将某个用户踢出房间
/// @param userId 踢出房间的userId
/// @param successBlock 成功回调
/// @param errorBlock 失败回调
- (void)kickUserFromRoom:(NSString *)userId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 锁定某个麦位，如果该麦位有人，会将用户强制下麦
/// @param seatIndex 麦位序号
/// @param isLocked 是否锁麦位
/// @param successBlock 锁麦成功
/// @param errorBlock 锁麦失败
- (void)lockSeat:(NSUInteger)seatIndex
```

```
- (void)lockSeat:(NSUInteger)seatIndex
lock:(BOOL)isLocked
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 将某个麦位静音
/// @param seatIndex 麦位序号
/// @param isMute 是否静音
/// @param successBlock 静音成功
/// @param errorBlock 静音失败
- (void)muteSeat:(NSUInteger)seatIndex
mute:(BOOL)isMute
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 更新麦位extra属性
/// @param index 麦位序号
/// @param extra 更新的extra
/// @param successBlock 更新成功回调
/// @param errorBlock 更新失败回调
- (void)updateSeatInfo:(NSUInteger)index
withExtra:(NSString *)extra
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 将除了自身所在麦位的其余所有麦位静音或者取消静音
/// @param isMute 是否静音
- (void)muteOtherSeats:(BOOL)isMute;

/// 静音所有远程音频流
/// @param isMute 是否静音所有远程音频流
- (void)muteAllRemoteStreams:(BOOL)isMute;

/// 将所有麦位锁麦或者解除锁麦
/// 该方法不会锁定有人的麦位
/// @param isLock 是否锁麦
- (void)lockOtherSeats:(BOOL)isLock;

/// 请求排麦
/// @param successBlock 请求排麦发送成功
/// @param errorBlock 请求排麦发送失败
- (void)requestSeat:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 取消排麦请求
/// @param successBlock 取消排麦成功
/// @param errorBlock 取消排麦失败
- (void)cancelRequestSeat:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 同意用户排麦请求
/// @param userId 请求排麦的用户id
/// @param successBlock 同意请求成功
/// @param errorBlock 同意请求失败
- (void)acceptRequestSeat:(NSString *)userId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;


/// 拒绝用户排麦请求
/// @param userId 请求排麦的用户id
/// @param successBlock 拒绝请求成功
/// @param errorBlock 拒绝请求失败
- (void)rejectRequestSeat:(NSString *)userId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
```

```objc
/// 获取当前排麦的用户列表
/// @param successBlock 成功获取回调
/// @param errorBlock 获取失败回调
- (void)getRequestSeatUserIds:(void (^)(NSArray<NSString *>*))successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 发送请求
/// @param content 发送请求的内容
- (void)sendInvitation:(NSString *)content
success:(void (^)(NSString *))successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 拒绝请求
/// @param invitationId 请求id
- (void)rejectInvitation:(NSString *)invitationId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 同意请求
/// @param invitationId 请求id
- (void)acceptInvitation:(NSString *)invitationId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 取消自己发出的请求
/// @param invitationId 请求id
- (void)cancelInvitation:(NSString *)invitationId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 通知房间所有用户执行某个刷新操作
/// @param name 刷新操作的名称
/// @param content 刷新操作的内容
- (void)notifyVoiceRoom:(NSString *)name
content:(NSString *)content;

@end

@interface RCVoiceRoomEngine (Control)

/// 停止本地麦克风收音
/// @param isDisable 是否停止
- (void)disableAudioRecording:(BOOL)isDisable;

/// 获取麦克风状态
- (BOOL)isDisableAudioRecording;

/// 设置房间音频质量和场景
/// @param quality 音频质量
/// @param scenario 音频场景
- (void)setAudioQuality:(RCVoiceRoomAudioQuality)quality
scenario:(RCVoiceRoomAudioScenario)scenario;

/// 设置是否使用扬声器
/// @param isEnable 是否使用
- (void)enableSpeaker:(BOOL)isEnable;

@end

@interface RCVoiceRoomEngine (PK)

/// 发送PK邀请
/// @param inviteeRoomId 被邀请用户所在的房间id
/// @param inviteeUserId 被邀请人的用户id
/// @param successBlock 邀请发送成功回调
```

```
/// @param successBlock 邀请发送成功回调
/// @param errorBlock 邀请发送失败回调
- (void)sendPKInvitation:(NSString *)inviteeRoomId
invitee:(NSString *)inviteeUserId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 撤回已发送的PK邀请
/// @param inviteeRoomId 被邀请用户所在的房间id
/// @param inviteeUserId 被邀请人的用户id
/// @param successBlock 邀请撤回发送成功
/// @param errorBlock 邀请撤回发送失败
- (void)cancelPKInvitation:(NSString *)inviteeRoomId
invitee:(NSString *)inviteeUserId
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 回复邀请人是否接受邀请
/// @param inviterRoomId 邀请人所在的房间id
/// @param inviterUserId 邀请人的用户id
/// @param type 回应邀请者的类型，接受，拒绝或者忽略
/// @param successBlock 回复发送成功
/// @param errorBlock 回复发送失败
- (void)responsePKInvitation:(NSString *)inviterRoomId
inviter:(NSString *)inviterUserId
responseType:(RCPKResponseType)type
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 屏蔽PK对象的语音
/// @param isMute 是否屏蔽
/// @param successBlock 调用成功
/// @param errorBlock 调用失败
- (void)mutePKUser:(BOOL)isMute
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 恢复跨房间 PK
/// @param info PK 的信息
/// @param successBlock 恢复成功
/// @param errorBlock 恢复失败
- (void)resumePKWithPKInfo:(RCVoicePKInfo *)info
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 退出PK
/// @param successBlock 退出成功
/// @param errorBlock 退出失败
- (void)quitPK:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

@end

@interface RCVoiceRoomEngine (Analysis)

+ (NSString *)getVersion;

@end

@interface RCVoiceRoomEngine (Deprecated)

/// 发送信息
/// @param message 融云消息实体
/// @param successBlock 发送成功
/// @param errorBlock 发送失败
- (void)sendMessage: (RCMessageContent *)message
```

```
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock DEPRECATED_MSG_ATTRIBUTE("use IMLib/IMKit sendMessage instead");

@end
```

语聊房扩展功能列表 RCVoiceRoomEngine+Plugin （2.1.1新增）

更新时间:2024-05-17

```
@interface RCVoiceRoomEngine (Plugin)
/// 用户主动上麦
/// @param seatIndex 麦位序号
/// @param info 麦位信息(mute extra)
/// @param successBlock 上麦成功
/// @param errorBlock 上麦失败
- (void)enterSeat:(NSUInteger)seatIndex
seatInfo:(nullable RCVoiceSeatInfo *)info
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 更新某个麦位上的信息
/// @param index 麦位序号
/// @param info 麦位信息(mute extra)
/// @param successBlock 更新成功回调
/// @param errorBlock 更新失败回调
- (void)updateSeatInfo:(NSUInteger)index
seatInfo:(nullable RCVoiceSeatInfo *)info
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 用户跳麦，在用户已经在麦位想切换麦位时调用
/// @param seatIndex 需要跳转的麦位序号
/// @param preSeat 更新旧麦位信息
/// @param targetSeat 更新新麦位信息
/// @param successBlock 跳麦成功
/// @param errorBlock 跳麦失败
- (void)switchSeatTo:(NSUInteger)seatIndex
preSeat:(nullable RCVoiceSeatInfo *)preSeat
targetSeat:(nullable RCVoiceSeatInfo *)targetSeat
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;

/// 用户跳麦，在用户已经在麦位想切换麦位时调用
/// @param seatIndex 需要跳转的麦位序号
/// @param switchMute 跳麦携带mute，被携带后的麦位mute会被置为false； false: 不携带mute
/// @param switchExtra 跳麦携带extra，被携带后的麦位extra会被置为null； false: 不携带extra
/// @param successBlock 跳麦成功
/// @param errorBlock 跳麦失败
- (void)switchSeatTo:(NSUInteger)seatIndex
switchMute:(BOOL)switchMute
switchExtra:(BOOL)switchExtra
success:(RCVoiceRoomSuccessBlock)successBlock
error:(RCVoiceRoomErrorBlock)errorBlock;
@end
```

语聊房回调列表 RCVoiceRoomDelegate

更新时间:2024-05-17

```
@protocol RCVoiceRoomDelegate <NSObject>

@optional
/// 房间的信息和麦位信息初始化完成回调，用户可在此执行关于房间的其他初始化操作
- (void)roomKVDidReady;

/// 房间发生异常问题
- (void)roomDidOccurError:(RCVoiceRoomErrorCode)code __attribute__((deprecated("请使用使用
roomDidOccurErrorWithDetails方法")));

/// 房间发生异常问题
/// @param error 错误的详细信息
- (void)roomDidOccurErrorWithDetails:(id<RCVoiceRoomError>)error;

/// 房间信息变更回调
/// 第一次加入房间时也会触发回调
/// @param roomInfo 房间信息
- (void)roomInfoDidUpdate:(RCVoiceRoomInfo *)roomInfo;

/// 房间已关闭
- (void)roomDidClosed;

/// 房间座位变更回调，包括自身上麦或下麦也会触发此回调
/// @param seatInfolist 座位列表信息
- (void)seatInfoDidUpdate:(NSArray<RCVoiceSeatInfo *> *)seatInfolist;

/// 某个主播上麦回调，包含自己上麦也会触发此回调
/// @param seatIndex 麦位号
/// @param userId 用户Id
- (void)userDidEnterSeat:(NSInteger)seatIndex
user:(NSString *)userId;

/// 某个主播下麦回调，包含自己下麦也会触发此回调
/// @param seatIndex 麦位号
/// @param userId 用户Id
- (void)userDidLeaveSeat:(NSInteger)seatIndex
user:(NSString *)userId;

/// 座位静音状态回调
/// @param index 座位号
/// @param isMute 静音状态
- (void)seatDidMute:(NSInteger)index
isMute:(BOOL)isMute;

/// 座位关闭回调
/// @param index 座位号
/// @param isLock 是否关闭
- (void)seatDidLock:(NSInteger)index
isLock:(BOOL)isLock;

/// 观众进房回调
/// @param userId 观众Id
- (void)userDidEnter:(NSString *)userId;
```

```
/// 观众退房回调
/// @param userId 观众Id
- (void)userDidExit:(NSString *)userId;

/// 麦位麦克风状态变化回调
/// @param speaking 是否正在说话
/// @param index 麦位序号
/// @param level 音量值
- (void)seatSpeakingStateChanged:(BOOL)speaking
atIndex:(NSInteger)index
audioLevel:(NSInteger)level;

/// 用户麦克风状态变化回调
/// @param speaking 是否正在说话
/// @param userId 上麦用户Id
/// @param level 音量值
- (void)userSpeakingStateChanged:(BOOL)speaking
userId:(NSString *)userId
audioLevel:(NSInteger)level;


/// 收取信息回调
/// @param message 收到的消息
- (void)messageDidReceive:(RCMessage *)message;


/// 收到房间通知
/// @param name 通知的名称
/// @param content 通知的内容
- (void)roomNotificationDidReceive:(NSString *)name
content:(NSString *)content;

/// 收到自己被抱上麦的请求
- (void)pickSeatDidReceiveBy:(NSString *)userId;

/// 收到自己被下麦的通知
- (void)kickSeatDidReceive:(NSUInteger)seatIndex;

/// 发送的排麦请求得到房主或管理员同意
- (void)requestSeatDidAccept;

/// 发送的排麦请求被房主或管理员拒绝
- (void)requestSeatDidReject;

/// 排麦列表发生变化
- (void)requestSeatListDidChange;

/// 接受邀请
/// @param invitationId 邀请的Id
/// @param userId 发送邀请的用户
/// @param content 邀请内容，用户可以自定义
- (void)invitationDidReceive:(NSString *)invitationId
from:(NSString *)userId
content:(NSString *)content;

/// 邀请被接受回调
/// @param invitationId 邀请Id
- (void)invitationDidAccept:(NSString *)invitationId;

/// 邀请被拒绝回调
/// @param invitationId 邀请Id
- (void)invitationDidReject:(NSString *)invitationId;

/// 邀请被取消回调
/// @param invitationId 邀请Id
- (void)invitationDidCancel:(NSString *)invitationId;
```

```objc
- (void)invitationDidCancel:(NSString *)invitationId;

/// 用户被踢出房间的回调
/// @param targetId 被踢出房间的用户id
/// @param userId 执行踢某人出房间的用户id
- (void)userDidKickFromRoom:(NSString *)targetId byUserId:(NSString *)userId;


/// 网络延迟
/// @param rtt 单位ms
- (void)networkStatus:(NSInteger)rtt;

/// PK 如果连接成功，会触发此回调
/// @param inviterRoomId 邀请 PK 的用户id
/// @param inviterUserId 邀请 PK 的用户房间id
/// @param inviteeRoomId 被邀请 PK 的用户id
/// @param inviteeUserId 被邀请 PK 的用户房间id
- (void)pkOngoingWithInviterRoom:(NSString *)inviterRoomId
withInviterUserId:(NSString *)inviterUserId
withInviteeRoom:(NSString *)inviteeRoomId
withInviteeUserId:(NSString *)inviteeUserId;

/// 对方结束PK时会触发此回调
/// 收到该回调后会自动退出 PK 连接
- (void)pkDidFinish;

/// 收到邀请 PK 的回调
/// @param inviterRoomId 邀请者的房间id
/// @param inviterUserId 邀请者的用户id
- (void)pkInvitationDidReceiveFromRoom:(NSString *)inviterRoomId byUser:(NSString *)inviterUserId;

/// 邀请者取消 PK 邀请回调
/// @param inviterRoomId 邀请者的房间id
/// @param inviterUserId 邀请者的用户id
- (void)cancelPKInvitationDidReceiveFromRoom:(NSString *)inviterRoomId byUser:(NSString *)inviterUserId;

/// 被邀请者拒绝 PK 邀请回调
/// @param inviteeRoomId 被邀请者的房间id
/// @param initeeUserId 被邀请者的用户id
- (void)rejectPKInvitationDidReceiveFromRoom:(NSString *)inviteeRoomId byUser:(NSString *)initeeUserId;

/// 邀请者忽略 PK 邀请回调
/// @param inviteeRoomId 被邀请者的房间id
/// @param inviteeUserId 被邀请者的用户id
- (void)ignorePKInvitationDidReceiveFromRoom:(NSString *)inviteeRoomId byUser:(NSString *)inviteeUserId;

@end
```

状态码

更新时间:2024-05-17

状态码	说明
70000	操作成功
70001	连接服务器失败，请确定 connectWithToken中的参数是否正确
70002	麦位序号不正确，有可能是创建房间时未设置麦位数量，也有可能是房间已销毁导致 KV 无数据
70003	该上麦用户此时已经在麦位上
70004	用户不在麦位上
70005	切换的麦位和当前所在麦位一样
70006	麦位不是空置状态，无法上麦
70007	抱人上麦不能选择自己
70008	发送上麦邀请失败
70009	不能踢自己下麦
70010	加入语聊房失败，请确认是否在控制台开通了音视频直播和音视频通话功能，并且账户并非处于欠费状态
70011	离开语聊房失败，请确定连接正常，并且自己在房间中
70012	获取房间信息时失败。一般因房间自动销毁导致，如果您的房间需要在房主离开后依然保持存在，可通过「绑定音视频房间」进行房间保活。参见即时通讯 Android/iOS/Web 聊天室业务下的「绑定音视频房间」文档或即时通讯服务端 API「绑定音视频房间」。
70013	已经在排麦中
70014	排麦人数过多，目前 SDK 内置的排麦最大总数为20 个，如果您需要更多的排麦数量，可以利用自身业务服务器建立类似接口

状态码	说明
70015	申请排麦发送失败。
70016	取消排麦发送失败。
70017	同意排麦发送失败
70018	拒绝排麦发送失败
70019	麦位信息同步失败，原因与获取房间信息失败70012类似，请保证房间之前没被自然销毁，保证创建房间时正确设置了麦位数量
70020	同步房间信息失败，原因可能是房间已销毁，或者创建房间未成功
70021	更新麦位信息失败，请确定网络正常并且在房间中
70022	获取排麦用户列表失败
70023	发送聊天室消息失败
70024	发送自定义Invitation失败
70025	取消已发送的自定义Invitation失败
70026	同意自定义Invitation 发送失败
70027	拒绝自定义Invitation 发送失败
70028	创建语聊房失败，原因与 70010 类似
70029	当前用户 Id 为空，请确保正确连接了融云
70030	获取最新的麦位信息失败
70031	PK 开始失败
70032	退出 PK 失败

状态码	说明
70033	发送 PK 邀请失败
70034	取消 PK 邀请失败
70035	静音 PK 失败
70036	创建房间信息不正确，请确定创建房间是设置了房间名称和麦位数量
70037	已经在 PK 中
70038	切换角色失败
70039	更新麦位信息失败

更新日志 2.1.2

更新时间:2024-05-17

- 新增麦位锁定功能开关
- 新增跨房间 PK 时，主播音量和麦克风状态同步功能

2.1.1.1

优化

1. 用户离开房间/掉线，移除未清理的麦位锁KV，防止锁被占用，别的用户无法上麦

新增

1. `clearSeatState:error:`
2. `setSeatPlaceHolderStateEnable:`

2.1.1

新增

1. 添加扩展RCVoiceRoomEngine (Plugin)
2. `enterSeat:seatInfo:success:error:`
3. `updateSeatInfo:seatInfo:success:error:`
4. `switchSeatTo:preSeat:targetSeat:success:error:`
5. `switchSeatTo:switchMute:switchExtra:success:error:`

2.0.9.1

修复问题

1. 麦位切换的时候当前和目标麦位信息分开更新导致回调不安全，改为合并批量更新 KV

2.0.9

优化

1. 本地错误日志描述

修复问题

1. SDK 同时支持 arm64 & armv7。

2.0.8.4

修复问题

1. 修复离开房间接口同时多次调用没有回调结果问题

2.0.8.3

优化

1. 音频模式默认为音乐聊天室
2. 开放 SDK 依赖版本，不再限制 IM 和 RTC 版本

2.0.8.2

修复问题

1. 修复 SDK 内部文件路径错误问题

2.0.8

修复

1. 修复 PK 期间，对方退出或掉线导致无法正常结束 PK 的问题
2. 修复更改房间麦位数，麦位信息更新不正常的问题

新增

1. 错误信息打印，当 SDK 调用出错时，会打印出 SDK 调用栈详细信息

2.0.7.2

1. 修复主播加入说话没声音

2.0.7.1

1. 修复 SDK 依赖的 IMKit 改为 IMLib

2.0.7

优化

1. 优化麦位锁定的方法，使用rtc提供的 `[[RCRTCEngine sharedInstance] defaultAudioStream].isMute`，与麦克风的控制完全隔离
2. `userDidEnter/userDidExit` 接口不再返回在线人数 `userCount`，缩短进入/离开房间的耗时

新增接口

1. `- (BOOL)isDisableAudioRecording;` 获取麦克风状态
2. `RCVoiceRoomDelegate` 增加 `- (void)roomDidClosed;` 房间已关闭回调

常见问题

更新时间:2024-05-17

连接融云服务器失败

通常有两个原因导致：

- initWithAppKey 后立即执行 connectWithToken 方法。

  解决方案：init 一般在 application 中，connect 一般是在登录以后。如果 init 和 connect 必须依次执行，connect 建议延迟 200ms。

- 重复 connect，即 connect 以后没有执行 disconnect，又再次执行 connect 操作。

  解决方案：在 connect 执行前先调用 disconnect。

加入语聊房返回失败

这通常是由于您没有开通 Appkey 的音视频直播功能，或者是免费时长用完时发生的错误。可以通过控制台查看您是否已经开通音视频服务。

申请上麦时，谁有权限通过或拒绝申请？

在语聊房 SDK 中，并没有权限的概念，也就是说，当房间某个用户申请上麦时，任何人都可以接收申请麦位变化的回调，您需要根据自己业务的需求，确定哪些人可以处理申请。

语聊房 SDK 是否有 Server 端？

首先，要区分「服务端」概念，如下：

- **您自己的业务服务器，即为您自身的业务提供接口的后端**
- **融云提供 IM 和 RTC 服务的后端**

而语聊房客户端 SDK 只依赖于融云的 IM 和 RTC 服务器。语聊房 SDK 只是基于融云 IM 和 RTC 能力的一层封装。

也就是说，如果您需要了解任何后端的支持，只需要查看融云的即时通讯（IM）和实时音视频（RTC）的服务端文档即可，融云并未提供专属的语聊房服务端和对应文档。

语聊房 SDK 是否包含权限的概念？

语聊房 SDK 所包含的所有 API 接口，全都没有所谓权限的概念。

当我们说语聊房 SDK 只有角色划分，而没有没有权限划分，是什么意思呢?

即语聊房 SDK 内只定义主播和观众角色，而将具体的权限设计与控制交给 App 业务逻辑自行掌控。

- **主播**：即在麦位上，能否发布音频流的人。
- **观众**：即不在麦位上，只能听音频流的人。

语聊房 SDK 内没有权限划分，意味着所有 API 的调用对房间内的所有人一视同仁。所有人都可以调用。

在真实业务场景中，请您根据具体业务逻辑来判断调用权限，即哪些用户可以或不可以调用部分 API，并自行实现权限控制。

语聊房在什么情况下需要保活？

从留存/销毁业务逻辑上划分，语聊房大致有两种设计：

1. 房主直播时创建语聊房，退播后销毁房间。

   这一种业务相对较简单。如果采用这一种设计，**不需要服务端做额外的保活措施**。

2. 创建语聊房后，无论主播在线与否，均保持房间留存，房主退播后不会销毁房间。

   如果采用这一种设计，您可能需要对房间进行保活处理。

融云语聊房的销毁行为与融云即时通讯（IM）聊天室的销毁机制相关。融云 IM 聊天室的销毁有两种机制：

1. 主动调用 IM Server API 提供的销毁聊天室接口，主动销毁聊天室。
2. 聊天室 1 个小时内没有人说话（时间可配置，最长 24 小时），且没有人加入聊天室时，会把聊天室内所有成员踢出聊天室，并触发自动销毁聊天室。

如果您的业务逻辑设计要求语聊房长时间持续存在，您可以参考我们的语聊房保活最佳实践进行实现。

如果语聊房不保活会发生什么？

如果不保活，可能会导致您的聊天室已经被融云销毁，这样您就丢失了保存在聊天室属性里的所有麦位信息和房间信息。在下次直播时，可能会发现房间没有麦位。业务上出现错误。

所以请您按照自己的具体业务来判断是否需要进行房间保活。

房间内麦位上出现影子用户

影子用户：用户虽已离开房间，但是麦位上还显示用户的信息，且房间内的其他人不能上到这个麦位上。
原因：因异常原因（例如：断网操过1分钟 或 用户直接杀掉应用进程 等）导致用户没有正常调用离开房间接口，而直接被迫离开房间。
推荐处理方案：此种情况，需依赖服务端，将用户的信息从麦位上清除掉，即将麦位的 userId 属性重置为 null，state 属性重置为 0。您可以参考我们的语聊房异常退出最佳实践进行实现。

多用户同时抢麦

现象：多个用户同时抢占同一个麦位，只有最后一个抢麦的用户成功，其他用户被迫下麦。
原因：麦位信息是通过语聊房关联的聊天室属性（KV）维护的。多个用户同时抢占同一个麦位，实际是几乎同时去更新同一个聊天室的 KV（融云服务端可处理毫秒级的 KV 更新请求），因此导致聊天室属性的 Value 依次被覆盖，最终只显示最后一

个更新 KV 的用户抢麦成功，其他人则被踢下麦。
推荐处理方案：

- 语聊房 SDK 升级到 2.1.1+（Android/iOS）和 2.0.5+（Web），并启用麦位的状态 KV方案来处理多用户同时抢麦。
- 服务端需接入语聊房异常退出最佳实践，并添加清理麦位的状态 KV的逻辑
- 麦位的状态 KV: 为处理多用户同时抢麦额外添加的一组 KV，用于标识麦位的所属状态，只有设置它的用户才能修改它。Key 格式：”RCSPlaceHolderKey_seat_“ + 麦位索引。例如：上麦/跳麦时，先尝试修改麦位的状态 KV，如果修改失败则表示麦位上已有用户，且并不是当前用户，故会回调 onError；如果修改成功，则表示麦位上没有用户，执行上麦位逻辑。
- 清理麦位的状态KV：获取聊天室内所有的属性 KV ，根据融云的 RTC 服务回调的异常退出事件类型（包括 roomId & userId）中的 userId 遍历获取对应的麦位信息的 KV，取出 Key（格式：”RCSeatInfoSeatPartPrefixKey_“ + 麦位索引 ） 解析出后缀就是麦位的索引，然后删除 key = ”RCSPlaceHolderKey_seat_“ + 麦位索引 的KV对