场景融合

语聊房

Web 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、房间管理和多人连麦的功能。

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、房间管理和多人连麦。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。
播放音频资源	支持播放本地、网络音频资源。
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。
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。

  Web Quickdemo

- **RC RTC 应用**

  展示如何使用场景化 SDK 搭建语聊房场景，提供 Web 端的源码。

  RC RTC Web 在线体验 · RC RTC web 源码

- **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

主要功能演示

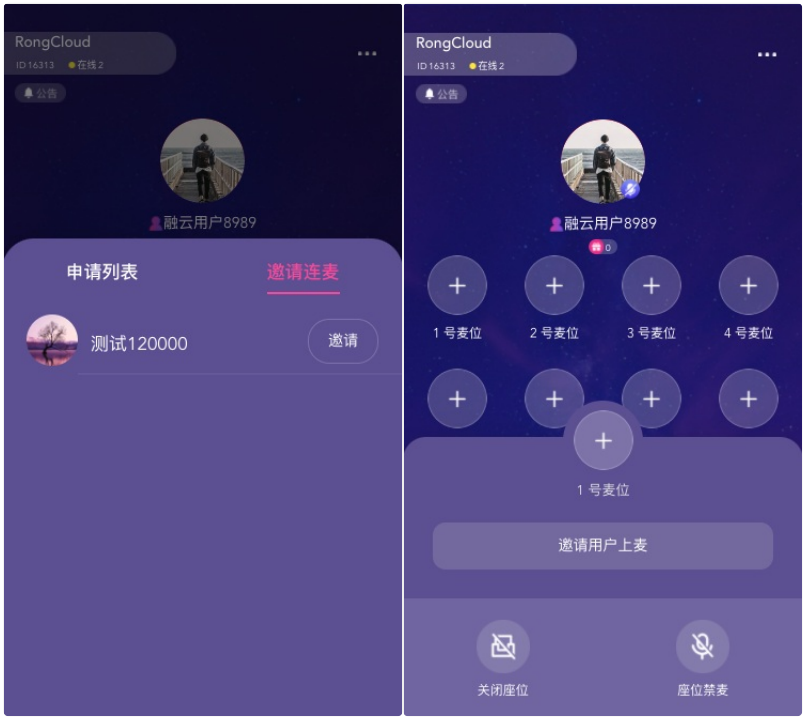
更新时间:2024-05-17

语聊房的主要功能包括麦位管理、房间管理、多人连麦。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。

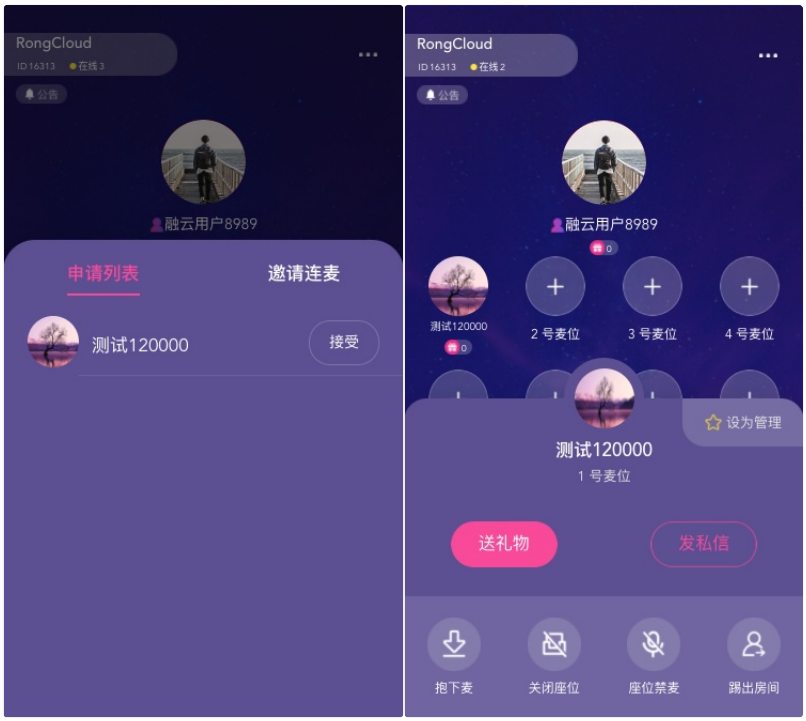


房间管理

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。



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。

集成 SDK 注册融云开发者账号

更新时间:2024-05-17

开通融云开发者服务请参考 开通服务

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。

导入 SDK

语聊房业务依赖即时通讯业务提供的信令通道。基于 RCVoiceRoomLib 开发应用，您必须安装以下库：

- 即时通讯基础能力库 Engine、IMLib
- 实时音视频基础能力库 RTCLib
- 语聊房能力库 RCVoiceRoomLib

> 从 2.0.4 版本开始，语聊房 SDK 依赖的 Engine 、 IMLib 和 RTCLib SDK 需要单独安装及初始化。如果您从 2.0.3 版本升级，请务必注意修改导入及初始化代码。

安装 Engine

要求安装 RongEngine 5.1.2 或之后版本。以下安装最新版：

```
# 安装 RongEngine v5
npm install @rongcloud/engine --save
```

安装 IMLib

要求安装 IMLib 5.1.2 之后版本。以下安装最新版：

```
# 安装 RongIMLib v5
npm install @rongcloud/imlib-next --save
```

安装 RTCLib

要求安装 RTCLib 5.3.1 或 5.3.1 以上版本。以下安装最新版：

```
# 安装 RTCLib
npm install @rongcloud/plugin-rtc --save
```

安装 RCVoiceRoomLib

```
# 安装 RCVoiceRoomLib
npm install @rongcloud/rcvoiceroomlib-v1 --save
```

代码集成

```
// 非 ESModule
const RongEngine = require('@rongcloud/engine')
const RongIMLib = require('@rongcloud/imlib-next')
const RongRTCLib = require('@rongcloud/plugin-rtc')
const RCVoiceRoomLib = require('@rongcloud/rcvoiceroomlib-v1')
// ESModule
import * as RongEngine from "@rongcloud/engine";
import * as RongIMLib from "@rongcloud/imlib-next";
import * as RongRTCLib from "@rongcloud/plugin-rtc";
import RCVoiceRoomLib from "@rongcloud/rcvoiceroomlib-v1";
```

> Web 端 RCVoiceRoomLib 依赖于融云Engine 、 IMLib 与 RTCLib，兼容参考 IMLib 兼容说明 和 RTCLib 浏览器支持清单 。

版本依赖说明

RCVoiceRoomLib 依赖融云 Engine 、 IMLib 与 RTCLib。

从 RCVoiceRoomLib v2.0.1 开始，依赖版本如下。

依赖组件	版本
Engine	5.1.2 及以上
IMLib	5.1.2 及以上
RTCLib	5.3.1 及以上

初始化 SDK

更新时间:2024-05-17

从 2.0.4 版本开始，语聊房 SDK 依赖的 IMLib、RTCLib SDK 需要单独安装及初始化。如果您从 2.0.3 版本升级，请务必注意修改导入及初始化代码。

初始化

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。

调用 RCVoiceRoomLib 的初始化接口时，务必保证 RongIMLib 、RongRTCLib 已初始化完毕，且 RCVoiceRoomLib 初始化接口在应用全生命周期内仅被调用一次。

```
// RongIMLib 初始化,请务必保证此过程只被执行一次
RongIMLib.init({appkey:'<Your-App-Key>'});
// RongRTCLib 初始化,请务必保证此过程只被执行一次
const rtcClient = RongIMLib.installPlugin(RongRTCLib.installer,{});
// RCVoiceRoomLib 初始化,请务必保证此过程只被执行一次
RCVoiceRoomLib.init({
RongIMLib: RongIMLib,
RongRTCLib: rtcClient,
});
```

建立连接

App Key 是应用的唯一性标识，Token 则是用户的唯一性标识，是用户连接融云 IM 服务所必需的身份凭证。Token 一般由开发者的应用服务器调用融云 Server API 获取 Token 接口获取之后，由应用服务器下发到应用客户端。

以下示例代码假定客户端已获取 Token。
从 2.0.4 版本开始，语聊房 SDK 改为使用 RongIMLib 的 `connect()` 方法。如果您从 2.0.3 版本升级，请注意修改连接代码。

调用 RongIMLib 的 `connect()` 方法：

```
RongIMLib.connect('<Your-Token>')
```

后续所有代码示例中的 `RCVoiceRoomLib` 均指通过初始化并连接服务后的 RCVoiceRoomLib 实例对象

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：
>
> 创建房间 · 加入房间 · 离开房间 · 踢出房间 · 销毁房间

创建房间

右图是内部大概的执行、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。

```
//配置房间信息
const RCVoiceRoomInfo = {}
// 设置房间名称
rcVoiceRoomInfo.roomName = `<your roomname>`;
// 设置麦位数量
rcVoiceRoomInfo.seatCount = 9;
```

创建房间

调用 RCVoiceRoomLib 的 createAndJoinRoom 方法。

```
/**
* 用户根据roomId创建并加入一个语聊房
* @param roomId 房间 Id
* @param RCVoiceRoomInfo 配置房间信息
* @param enableSeatStateLock 是否启用麦位状态锁，处理多端抢麦问题。SDK默认不启用。(非必传)
*/

RCVoiceRoomLib.createAndJoinRoom(roomId,RCVoiceRoomInfo,enableSeatStateLock);
```

加入房间

右图是内部大概的执行流程。

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。

```
/**
* 用户根据已存在roomId加入一个语聊房
*
* @param roomId 房间 Id
* @param enableSeatStateLock 是否启用麦位状态锁，处理多端抢麦问题。SDK默认不启用。(非必传)
*/
RCVoiceRoomLib.joinRoom(roomId,enableSeatStateLock);
```

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。

```
/**
* 房间信息变更回调
*
*/
RCVoiceRoomLib.on("roomInfoDidUpdate", (roomInfo) => {
console.log(roomInfo)
//业务代码
})
```

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。

```
/**
* 房间座位变更回调
*
*/
RCVoiceRoomLib.on("seatInfoDidUpdate", (seatInfoList) => {
console.log(seatInfoList)
//业务代码
})
```

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。

```
/**
* 离开房间
* @param roomId 房间 Id
*/
RCVoiceRoomLib.leaveRoom(roomId).then(()=>{
})
```

回调处理

通常在成功回调里，做移除监听，销毁页面等逻辑操作

```
// 业务代码
// 这里可以做销毁页面等逻辑
```

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。

```

/**
* 被踢出房间
* @param userId 被踢用户 Id
*/
RCVoiceRoomLib.kickUserFromRoom(userId);
```

被踢的一方用户回调

被踢出的回调里，执行 leaveRoom 离开房间。

```

/**
* 被踢出房间回调
*
* @param targetId 被踢用户的标识
* @param userId 发起踢人用户的标识
*/
RCVoiceRoomLib.on("RCKickUserOutRoomContent", (info) => {
//业务代码
console.log(info);
//执行 `leaveRoom` 离开房间
RCVoiceRoomLib.leaveRoom(roomId)
})
```

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 ChatroomDestroyed 回调里，调用离开房间的接口，如右图所示：

在房间销毁回调里必须主动调用离开房间接口。

```
/**
* 房间被销毁回调
*
*/
RCVoiceRoomLib.on("ChatroomDestroyed", () => {
//业务代码
//执行 `leaveRoom` 离开房间
RCVoiceRoomLib.leaveRoom(roomId)
})
```

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

上麦

SDK 现拥有下面几种上麦模式：

- **自由上麦**
- **申请上麦**
- **邀请上麦**

> 语聊房的麦位总数由创建时传入的房间信息 `RCVoiceRoomInfo` 对象的 `seatCount` 属性决定。因此一个语聊房的麦位序号范围为 [0, seatCount - 1]

UI 表现形式

在语聊房 UI 上，上麦一般可体现为麦位上显示某个用户的头像，表示该麦位已被该用户占用。

下面我们使用语聊房 SDK Demo 的 UI 截图展示主播麦位和 1 号麦位用户在麦上时的状态。

自由上麦

自由上麦 不需要经过房主或者管理员等通过。通常是在 UI 上点击某个麦位后直接调 enterSeat。大致流程如右图所示。

调用 enterSeat

调用成功后，用户角色自动切换为主播，并自动发布流。

房间内所有人会收到相应回调。

```
/**
* 用户主动上麦
*
* @param seatIndex 麦位序号 number
* @param seat (非必传)上麦同步设置的麦位信息，目前只支持设置 seat.isMute & seat.extra
*/
RCVoiceRoomLib.enterSeat(seatIndex,seat);
```

处理上麦回调

成功调用 enterSeat 后，房间所有用户都会触发回调 seatInfoDidUpdate。

您需要在回调中处理麦位 UI 的变化。常规做法就是直接在回调中更新数据源，刷新您的麦位 UI。

```
// 任何麦位的变化都会触发此回调。
/**
* 房间座位变更事件，包括自身上麦或下麦也会触发此事件
*
* @param seatInfoList 座位列表信息 {@link RCVoiceRoomInfo}
*/
RCVoiceRoomLib.on("seatInfoDidUpdate",(seatInfoList)=>{
//业务代码
consol.log(seatInfoList)
});
```

申请上麦

想要上麦的用户需先发起上麦申请，经房主或其他有权限的用户同意后，方可上麦。大致流程如右图：

用户发起连麦申请

语聊房最多支持 20 个连麦申请。

```
/**
* 请求排麦
*
*/
RCVoiceRoomLib.requestSeat();
```

用户取消连麦申请

取消连麦，就会空出一个申请名额。

```
// 取消排麦请求
RCVoiceRoomLib.cancelRequestSeat();
```

连麦申请列表变化通知

发起连麦，取消连麦，房间内其他用户都会触发 RequestSeatListChanged 回调，继而收到请求麦位列表变化的通知。

```
/**
* 排麦列表发生变化回调
*/
RCVoiceRoomLib.on("RequestSeatListChanged", () => {
//业务代码
})
```

查询最新连麦申请列表

收到请求麦位列表变化的通知后，调用 getRequestSeatUserIds。

请根据您自身业务确定哪些用户有权限，可查询房间内最新的连麦申请列表。

```
// 调用查询申请列表接口，可根据列表数据做展示，然后提供同意或拒绝操作
const RequestSeatUserIds = RCVoiceRoomLib.getRequestSeatUserIds();
```

展示连麦申请用户列表（UI 示例）

getRequestSeatUserIds 成功后，返回值包含最多 **20** 个此时正在要求排麦的用户 ID。

您可使用自身业务接口通过用户 ID 获取这些用户的信息，并显示在 UI 上。如右图所示：

同意连麦申请

房主/管理员（根据您自身业务确定哪些用户有权限）接受连麦申请，即同意让申请者上麦。

```
/**
* 同意用户排麦请求
*
* @param userId 请求排麦的用户 Id
*/
RCVoiceRoomLib.acceptRequestSeat(userId);
```

连麦申请者收到同意通知

房主/管理员同意连麦请求后，发起申请的用户会触发回调 RequestSeatAccepted。

在此方法中调用 enterSeat 即可成功上麦。

```
/**
* 发送的排麦请求得到房主或管理员同意
*/
RCVoiceRoomLib.on("RequestSeatAccepted"()=>{
//业务代码
})
```

拒绝连麦请求

房主/管理员（根据您自身业务确定哪些用户有权限）拒绝连麦申请，即拒绝发起申请的用户上麦。

```
/**
* 拒绝用户排麦请求
*
* @param userId 被拒绝用户 userId
*/
RCVoiceRoomLib.rejectRequestSeat(userId);
```

连麦申请者收到拒绝通知

可在此方法中显示提示或进行其他操作，比如显示一条提醒「您的连麦申请被拒绝」。

```
// 发送的排麦请求被房主或管理员拒绝
RCVoiceRoomLib.on("RequestSeatRejected",()=>{
//业务代码
});
```

邀请上麦

邀请上麦是指房主或其他有权限的用户邀请房间里的某个观众上麦。邀请发送后，该观众会触发回调，可在回调中选择上麦或者拒绝邀请。大致流程如右图所示。

UI 表现形式

右图是 UI 演示房主邀请用户上麦的界面。

房主/管理员邀请某个用户上麦

主动发起让某个用户上麦的邀请。

```
/**
* 邀请用户上麦
*
* @param userId 用户 Id
*/
RCVoiceRoomLib.pickUserToSeat(userId);
```

被邀请用户收到通知

触发回调 RCPickerUserSeatContent，被邀请用户可自行决定是否上麦。

```
/**
* 被邀请上麦回调
*
* @param sendUserId 发起邀请用户 Id
* @param targetId 被邀请用户 Id
*/
RCVoiceRoomLib.on('RCPickerUserSeatContent',({sendUserId,targetId})=>{
//业务代码
console.log({sendUserId,targetId})
})
```

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。

```
/**
* 用户主动下麦
*
*/
RCVoiceRoomLib.leaveSeat();
```

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`。大致流程如右图所示。

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。

```
/**
* 将某个麦位下麦
*
* @param userId 用户 Id
*/
RCVoiceRoomLib.kickUserFromSeat(userId);
```

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。

```
/**
* 房间座位变更事件，包括自身上麦或下麦也会触发此事件
*
* @param seatInfoList 座位列表信息 {@link RCVoiceRoomInfo}
*/
RCVoiceRoomLib.on("seatInfoDidUpdate",(seatInfoList)=>{
//业务代码
consol.log(seatInfoList)
});
```

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。

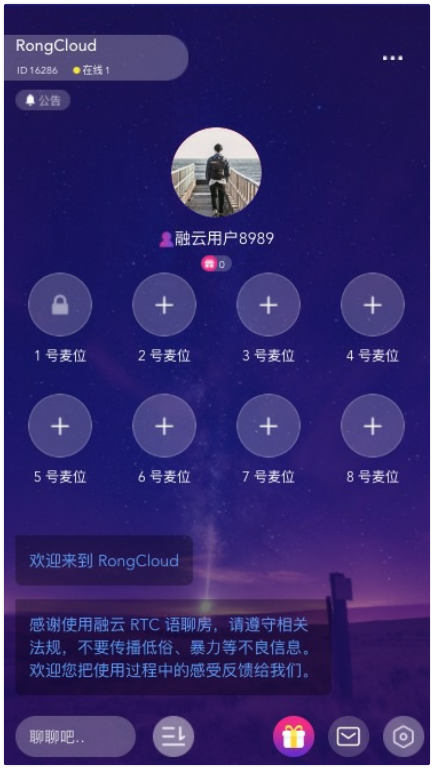
> 麦位控制主要包括：
>
> 锁定麦位 · 静音麦位（闭麦）

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 Demo 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。

```
/**
* 锁定某个麦位
*
* @param seatIndex 麦位序号
* @param isLocked 是否锁麦位
*/
RCVoiceRoomLib.lockSeat(seatIndex,isLocked);
```

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。大致流程如右图所示。

示例

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

闭麦方法

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
*/
RCVoiceRoomLib.muteSeat(seatIndex,isMute);
```

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

锁定其它麦位

锁定除自己之外其余所有的空闲麦位。

```
/**
* 锁定 / 解锁 其他空麦位
*
* @param isLock 是否锁麦
*/
RCVoiceRoomLib.lockOtherSeats(isLock);
```

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。

```
/**
* 静音除自己外的麦位
*
* @param isMute 是否静麦
*/
RCVoiceRoomLib.muteOtherSeats(isMute);
```

房间信息扩展

如果想要在房间内，加入与房间信息有关的自定义数据，同时同步给房间内所有人，可以使用此扩展方法。

```
/**
* 可更新房间信息的 extra 字段(自定义扩展信息，建议传json字符串)
* @param roomInfo.isFreeEnterSeat 是否自由上麦
* @param roomInfo.isLockAll 是否全麦锁座
* @param roomInfo.isMuteAll 是否全麦锁麦
* @param roomInfo.roomName 房间名称
* @param roomInfo.seatCount 房间座位数
* @param roomInfo.extra 拓展字段
*/
RCVoiceRoomLib.setRoomInfo(roomInfo: roomInfo);
```

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。

```
/**
* 房间信息变更回调
*
*/
RCVoiceRoomLib.on("roomInfoDidUpdate", (roomInfo) => {
console.log(roomInfo)
//业务代码
})
```

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。

```
/**
* 开启/停止麦克风采集音频数据
*
* @param isDisable
*/
RCVoiceRoomLib.disableAudioRecording(isDisable)
```

扬声器控制

打开扬声器(解决 mac safari 默认阻止播放 audio)。

```
/**
* 主动打开扬声器(解决mac safari默认阻止播放audio)
*
*/
RCVoiceRoomLib.enableSpeaker();
```

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
*/
RCVoiceRoomLib.muteAllRemoteStreams(isMute)
```

播放音乐

主播播放音乐后，房间内所有人都会接收并播放音乐。

```
/**
*
* 发布自定义音频流
* @param file 本地 MP3 file实例
*/
RCVoiceRoomLib.createTrack(file)
```

停止播放音乐

主播停止播放音乐。

```
/**
*
* 停止发布自定义音频流(调用默认已发布自定义音频流)
*
*/
RCVoiceRoomLib.stopCreateTrack()
```

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。

```
/**
* 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
*/
RCVoiceRoomLib.notifyVoiceRoom(name, content);
```

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 RoomNotificationReceived 回调。

可在该回调中进行判断，并进行对应的操作。

```
/**
* 房间通知回调
*
* @param name 名称
* @param content 通知内容
*/
RCVoiceRoomLib.on("RoomNotificationReceived", (info) => {
//业务代码
console.log(info);
})
```

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。

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 · 结束 PK · 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。

```
/**
* 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
*/
RCVoiceRoomLib.sendPKInvitation(inviteeRoomId,inviteeUserId)
```

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
*/
RCVoiceRoomLib.on("ReceivePKInvitation", (inviterRoomId, inviterUserId) => {
console.log(inviterRoomId, inviterUserId)
//业务代码
})
```

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。

```
/**
* 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
*/
RCVoiceRoomLib.cancelPKInvitation(inviteeRoomId,inviteeUserId)
```

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。

```
/**
* 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
*/
RCVoiceRoomLib.on("PKInvitationCanceled", (inviterRoomId, inviterUserId) => {
console.log(inviterRoomId, inviterUserId)
//业务代码
})
```

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 PKResponse 来向 Inviter 发送不同的回复。

- `PKResponse.accept`：接受 PK 邀请
- `PKResponse.reject`：拒绝 PK 邀请

```
/**
* 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 pkResponse pk邀请的响应状态 同意/拒绝
*/
RCVoiceRoomLib.responsePKInvitation(inviteeRoomId,inviteeUserId,pkResponse)
```

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
* rcpkInfo.inviterRoomId 邀请 PK 的用户所在房间id
* rcpkInfo.inviterUserId 邀请 PK 的用户id
* rcpkInfo.inviteeRoomId 被邀请 PK 的用户所在房间id
* rcpkInfo.inviteeUserId 被邀请 PK 的用户id
*/
RCVoiceRoomLib.on("PKGoing", (rcpkInfo) => {
console.log(rcpkInfo)
//业务代码
})
```

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
*/
RCVoiceRoomLib.on("PKInvitationRejected", (inviteeRoomId, inviteeUserId) => {
console.log(inviteeRoomId, inviteeUserId)
//业务代码
})
```

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 `true` / `false` ，对应是否屏蔽的情况。

```
/**
* 屏蔽PK对象的语音
* @param isMute 是否屏蔽
*/
RCVoiceRoomLib.mutePKUser(isMute)
```

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的时候，需要从自己的业务服务器上获取被中断的 PK 过程的信息，拿到对方房间的 ID 和 对方用户 ID 。

```
/**
* 恢复跨房间 PK
* @param otherRoomId 对方的房间 id
* @param otherUserId 对方的用户 id
*/
RCVoiceRoomLib.resumePk(otherRoomId, otherUserId)
```

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 。

```
/**
* 结束 PK
*/
RCVoiceRoomLib.quitPK()
```

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 `PKFinish` 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
*/
RCVoiceRoomLib.on("PKFinish", () => {
//业务代码
})
```

语聊房房间模型 RCVoiceRoomInfo

更新时间:2024-05-17

```
interface roomInfo{
/**
* 房间名,不可为空
*/
roomName:string;

/**
* 是否可以自由上麦，状态标记，直接修改不会自动触发锁麦操作
*/
isFreeEnterSeat:boolean;

/**
* 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
*/
isLockAll:boolean;

/**
* 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
*/
isMuteAll:boolean;

/**
* 房间座位数
*/
seatCount:number;

/**
* 拓展字段
*/
extra?: string;
}
```

语聊房麦位模型 RCVoiceSeatInfo

更新时间:2024-05-17

```
/// 麦位状态是互斥的
/**
* 麦位状态
*/
interface seatInfo{
/**
* 当前状态
*/
status: RCSeatStatus;

/**
* 是否静音
*/
mute: boolean;

/**
* 正在发言
*/
speaking: boolean;

/**
* 用户 Id
*/
userId?:string;

/**
* 拓展
*/
extra?:string;
}

/**
* 麦位状态枚举定义
*/
const RCSeatStatus = {
/**
* 麦位空闲
*/
RCSeatStatusEmpty:0,

/**
* 麦位占用中
*/
RCSeatStatusUsing:1,

/**
* 麦位被锁
*/
RCSeatStatusLocking:2
}
```

语聊房功能列表 IRCVoiceRoomEngine

更新时间:2024-05-17

```
/**
* @param AppKey
*/
interface initItem {
AppKey: string;
}

/**
*
* @param isMute 是否静音
* @param extra 拓展字段
*/
interface seat {
isMute: boolean;
extra: string;
}

/**
*
* @param isFreeEnterSeat 是否自由上麦
* @param isLockAll 是否全麦锁座
* @param isMuteAll 是否全麦锁麦
* @param roomName 房间名称
* @param seatCount 房间座位数
* @param extra 拓展字段
*/
interface roomInfo {
isFreeEnterSeat?: boolean;
isLockAll?: boolean;
isMuteAll?: boolean;
roomName: string;
seatCount: number;
extra?: string;
}


export default RCVoiceRoomLib = {
/**
* 初始化 RCVoiceRoomLib
*
* @param item 参数
* @param RongIMLib IM 实例对象
* @param RongRTCLib RTC 实例对象
*/
init(item: initItem)

/**
* 连接融云服务器
* 注意：
* @param token 用户 token
*/
connect(token: string)

/**
* 创建并加入房间 (v2.0.5 新增,代替 createAndJoinRoom(option:creatAndJoinOption))
* 注意：
* 不自动上麦，需根据业务确定是否手动上麦
* @param roomId 房间唯一标识
* @param roomInfo 房间信息
* @param enableSeatStateLock 是否启用麦位状态锁，处理用户同时抢麦问题。SDK默认不启用。
*/
createAndJoinRoom(roomId:string,roomInfo:roomInfo,enableSeatStateLock?:boolean);
```

```
/**
* 加入语聊房
*
* @param roomId 房间唯一标识
* @param enableSeatStateLock 是否启用麦位状态锁，处理多用户同时抢麦问题。SDK默认不启用。
*/
joinRoom(roomId:string,enableSeatStateLock?:boolean);

/**
* <p>离开当前房间</p>
* <p>注意：和joinRoom成对调用</p>
*
*/
leaveRoom(roomid: string);

/**
* 用户上麦
*
* @param seatIndex 麦位序号
* @param seat 上麦同步设置的麦位信息，目前只支持设置 seat.isMute & seat.extra
*/
enterSeat(seatIndex: number,seat?:seat);

/**
* <p>用户下麦</p>
* <p>注意：和 enterSeat()成对调用</p>
*
*/
leaveSeat();

/**
* <p>用户跳麦</p>
* <p>注意：在用户已经在麦位想切换麦位时调用</p>
*
* @param seatIndex 需要跳转的麦位序号
* @param preSeat 待赋值给当前麦位的信息，切麦时会将 preSeat 的 isMute & extra 赋值给当前所在麦位。
* @param targetSeat 待赋值给目标麦位的信息，切麦时会将 targetSeat 的 isMute & extra 赋值给目标麦位。
*/
switchSeatTo(seatIndex: number,preSeat?:seat,targetSeat?:seat);

/**
* 邀请用户上麦
*
* @param userId 用户 Id
*/
pickUserToSeat(userId: string);

/**
* 下麦 指定索引的麦位
*
* @param userId 用户 Id
*/
kickUserFromSeat(userId: string);

/**
* 将用户踢出房间
*
* @param userId 用户 Id
*/
kickUserFromRoom(userId: string);

/**
* 锁定指定索引的麦位
*
* @param seatIndex 麦位序号
* @param isLocked 是否锁麦位
*/
lockSeat(seatIndex: number, isLocked: boolean);

/**
* 静音指定索引的麦位
*
* @param seatIndex 麦位序号
```

```
* @param seatIndex 麦位序号
* @param isMute 是否静音
*/
muteSeat(seatIndex: number, ismute: boolean);

/**
* 将所有麦位静麦或取消静麦
*
* @param isMute 是否静麦
*/
muteOtherSeats(isMute: boolean);

/**
* 静音所有远程音频流
*
* @param isMute 是否静音
*/
muteAllRemoteStreams(isMute: boolean);

/**
* 将所有麦位锁麦或者解除锁麦
*
* @param isLock 是否锁麦
*/
lockOtherSeats(isLock: boolean);

/**
* 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
*/
setRoomInfo(roomInfo: roomInfo);

/**
* 停止本地麦克风收音
*
* @param isDisable 是否停止
*/
disableAudioRecording(isDisable: boolean);

/**
*
* 获取本地本地麦克风状态
*
*/
isDisableAudioRecording()

/**
* 主动打开扬声器(解决mac safari默认阻止播放audio)
*
*/
enableSpeaker();

/**
* 请求排麦
*
*/
requestSeat();

/**
* 取消排麦请求
*
*/
cancelRequestSeat();

/**
* 同意用户排麦请求
*
* @param userId 请求排麦的用户 Id
*/
acceptRequestSeat(userId: string);

/**
* 拒绝用户排麦请求
```

```
*
* @param userId
* @param
*/
rejectRequestSeat(userId: string);

/**
* 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
*/
notifyVoiceRoom(name: any, content: any);

/**
* 获取最近所有排麦申请的用户id
*
*/
getRequestSeatUserIds();

/**
* 获取最新麦位信息
*
* @
*/
getLatestSeatInfo();

/**
* 更新指定麦位信息中的extra字段 (v2.0.5 新增,代替 updateSeatInfoExtra(index: number, extra: any))
*
* @param index 麦位索引
* @param extra 拓展字段
* @param isMute 是否静麦
*/
updateSeatInfo(index: number, extra: string, isMute?:boolean )

/**
*
* 发布自定义音频流
* @param file 本地 MP3 file实例
*/
createTrack(file: any)

/**
*
* 停止发布自定义音频流(调用默认已发布自定义音频流)
*
*/
stopCreateTrack()

/**
* 是否启用麦位状态锁，处理多用户同时抢麦问题。SDK 默认不启用。(v2.0.5 新增)
* 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
* @param enable true: 启用，false: 不启用
*/
enableSeatStateLocked(enable:boolean);

/**
* 清理 麦位异常状态 (v2.0.5 新增)
* 注意：
* 1、只有加入房间成功后会生效
*/
clearSeatState();

/**************************************
* voice room pk (v2.0.8 新增)
**************************************/

/**
* 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
*/
sendPKInvitation(inviteeRoomId: string,inviteeUserId: string)

/**
* 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
*/
cancelPKInvitation(inviteeRoomId: string,inviteeUserId: string)

/**
* 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 pkResponse pk邀请的响应状态 同意/拒绝
*/
responsePKInvitation(inviteeRoomId: string,inviteeUserId: string,pkResponse:boolean)

/**
* 屏蔽PK对象的语音
* @param isMute 是否屏蔽
*/
mutePKUser(isMute:boolean)

/**
* 结束 PK
*/
quitPK()

/**
* 恢复跨房间 PK
* @param otherRoomId 对方的房间 id
* @param otherUserId 对方的用户 id
*/
resumePk(otherRoomId: string, otherUserId: string)

}
```

语聊房回调列表 RCVoiceRoomEventListener

更新时间:2024-05-17

```
/**
* 语聊房事件监听
*/
/**
* 房间信息变更回调
*
*/
RCVoiceRoomLib.on("roomInfoDidUpdate", () => {
//业务代码
})

/**
* 房间座位变更回调
*
*/
RCVoiceRoomLib.on("seatInfoDidUpdate", () => {
//业务代码
})

/**
* 座位是否被静麦回调
* @param isMute 是否被静麦
*
*/
RCVoiceRoomLib.on("SeatMute", (isMute) => {
console.log(isMute)
//业务代码
})

/**
* 房间人数发生改变(有人进来)回调
* @param userCount 是否被静麦
*/
RCVoiceRoomLib.on("AudienceEnter", (userCount) => {
console.log(userCount)
//业务代码
})

/**
* 房间人数发生改变(有人离开)回调
* @param userCount 是否被静麦
*/
RCVoiceRoomLib.on("AudienceExit", (userCount) => {
console.log(userCount)
//业务代码
})

/**
* 收取消息回调
*
* @param message 收到的消息
*/
RCVoiceRoomLib.on("MessageReceived", (message:object) => {
//业务代码
console.log(message);
})

/**
* 房间通知回调
*
* @param name 名称
* @param content 通知内容
```

```
*/
RCVoiceRoomLib.on("RoomNotificationReceived", (info:object<{name:any,content:any}>) => {
//业务代码
console.log(info);
})

/**
* 自主播被下麦回调
*/
RCVoiceRoomLib.on("KickSeatReceived", () => {
//业务代码
})

/**
* 房主或管理员 接受同意 用户的排麦申请回调
*/
RCVoiceRoomLib.on("RequestSeatAccepted", () => {
//业务代码
})

/**
* 发送的排麦请求被房主或管理员拒绝回调
*/
RCVoiceRoomLib.on("RequestSeatRejected", () => {
//业务代码
})

/**
* 排麦列表发生变化回调
*/
RCVoiceRoomLib.on("RequestSeatListChanged", () => {
//业务代码
})


/**
* 收到上麦邀请回调
* @param targetId 被邀请上麦的用户id
* @param sendUserId 邀请上麦动作发起者id
*/
RCVoiceRoomLib.on("RCPickerUserSeatContent", (info:object<{targetId:string,sendUserId:string}>) => {
//业务代码
console.log(info);
})

/**
* 被踢出房间回调
*
* @param targetId 被踢用户的标识
* @param userId 发起踢人用户的标识
*/
RCVoiceRoomLib.on("RCKickUserOutRoomContent", (info:object<{targetId:string,userId:string}>) => {
//业务代码
console.log(info);
})

/**
* 账号被顶掉回调
*
*/
RCVoiceRoomLib.on("ConnectioBreakOff", () => {
//业务代码
})

/**
* 房间被销毁回调
*
*/
RCVoiceRoomLib.on("ChatroomDestroyed", () => {
//业务代码
})

/**
* etc 断开连接 主播端
```

```
* rtc 断开连接 主播端
*
*/
RCVoiceRoomLib.on("RTCPeerConnectionCloseByException", () => {
//业务代码
})

/************************************
* voice room pk (v2.0.8 新增)
************************************/

/**
* 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
*/
RCVoiceRoomLib.on("ReceivePKInvitation", (inviterRoomId, inviterUserId) => {
console.log(inviterRoomId, inviterUserId)
//业务代码
})

/**
* 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
*/
RCVoiceRoomLib.on("PKInvitationCanceled", (inviterRoomId, inviterUserId) => {
console.log(inviterRoomId, inviterUserId)
//业务代码
})

/**
* 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 rcpkInfo 返回pk信息
* rcpkInfo.inviterRoomId 邀请 PK 的用户所在房间id
* rcpkInfo.inviterUserId 邀请 PK 的用户id
* rcpkInfo.inviteeRoomId 被邀请 PK 的用户所在房间id
* rcpkInfo.inviteeUserId 被邀请 PK 的用户id
*/
RCVoiceRoomLib.on("PKGoing", (rcpkInfo) => {
console.log(rcpkInfo)
//业务代码
})

/**
* 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
*/
RCVoiceRoomLib.on("PKInvitationRejected", (inviteeRoomId, inviteeUserId) => {
console.log(inviteeRoomId, inviteeUserId)
//业务代码
})

/**
* 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
*/
RCVoiceRoomLib.on("PKFinish", () => {
//业务代码
})
```

更新日志 v2.0.8

更新时间:2024-05-17

> 发布日期：2022/11/16

修复问题：

1. 修复在更新麦位信息后，「静音所有远程音频流」状态失效的问题

新增功能：

1. 新增跨房间 pk 功能相关 api

v2.0.5

> 发布日期：2022/10/17

修复问题：

1. 修复加入客户端创建的房间时，同步 seatinfolist 信息中存在 empty 的问题
2. 修复了调用 api 接口异常时，无错误信息上报的问题

功能优化：

1. 主动修改麦位信息时，可监听 seatInfoDidUpdate 更新麦位列表，优化同步异常的问题。
2. 优化了创建并加入房间时，监听 seatInfoDidUpdate 事件多次的问题
3. 优化了加入房间时，监听 seatInfoDidUpdate 事件多次的问题

接口变更：

1. 删除接口 `createAndJoinRoom(option:creatAndJoinOption);`，升级到 2.0.5 时需要替换该方法为 `createAndJoinRoom(roomId:string,roomInfo:roomInfo,enableSeatStateLock?:boolean);`。
2. 新增接口 `createAndJoinRoom(roomId:string,roomInfo:roomInfo,enableSeatStateLock?:boolean);`。`enableSeatStateLock` 参数控制是否启用麦位状态锁，用于处理多用户同时抢麦时最后一位用户抢麦成功的问题。
3. 删除接口 `updateSeatInfoExtra(index: number, extra: any)`，升级到 2.0.5 时需要替换该方法为 `updateSeatInfo(index: number, extra: string, isMute?:boolean )`。
4. 新增 `updateSeatInfo(index: number, extra: string, isMute?:boolean )`。`isMute` 参数控制可麦位的静音属性。
5. 变更接口 `joinRoom()`，新增了参数 `enableSeatStateLock`。
6. 接口 `enterSeat()` 新增 `seat` 参数，用于在上麦时设置麦位信息，目前支持设置 `seat.mute` & `seat.extra`。
7. 接口 `switchSeatTo()` 新增 `preSeat`（待赋值给当前麦位的信息）和。`targetSeat`（待赋值给目标麦位的信息）。
8. 新增 `clearSeatState()` 用于清理麦位异常锁状态。
9. 新增 `enableSeatStateLocked()` 控制是否启用麦位状态锁，用于处理多用户同时抢麦时最后一位用户抢麦成功的问题。

v2.0.4

> **发布日期：**2022/06/21

修复问题：

1. 修复进入房间时更新 KV 数据异常报错问题
2. 修复被踢出房间时状态异常问题

功能优化：

1. 进入房间时更新 SDK 内存数据优化
2. 新增过滤非当前房间错误 KV 信息能力，修复异常 KV 更新问题

接口变更：

1. 方法 `init()` 只注册场景化语聊房 SDK 能力。IMLib、RTCLib 功能需要单独安装并注册
2. 删除 `connect()` 方法,连接需调用 IMLib 的 `connect()` 方法

v2.0.3

> **发布日期：**2022/04/07

1. 修复切换座位数引起的麦位信息错误问题
2. 修复全麦锁座、全麦静音更新异常问题

v2.0.1

> **发布日期：**2022/03/14

新增接口：

1. 方法 `createTrack(file)` :发布自定义音频流
2. 方法 `topCreateTrack()` :停止发布自定义音频流(调用默认已发布自定义音频流)
3. 回调 `ConnectioBreakOff` :账号被顶掉通知
4. 回调 `ChatroomDestroyed` :房间被销毁通知
5. 回调 `RTCPeerConnectionCloseByException` :主播端 rtc 断开连接通知

变更接口：

1. 删除方法 `sendMessage()` :发送房间消息
2. 删除回调 `onVoiceRoomAgreeManagePick` :邀请被接受通知
3. 删除回调 `onVoiceRoomRejectManagePick` :邀请被拒绝通知
4. 回调 `onMessageReceived` 变更为 `MessageReceived`
5. 回调 `onRCPickerUserSeatContent` 变更为 `RCPickerUserSeatContent`
6. 回调 `onRoomNotificationReceived` 变更为 `RoomNotificationReceived`

7. 回调 `SeatMute` 和 `SeatUnMute` 被合并为 `SeatMute`
8. 回调 `changeUserCount` 被拆分为 `AudienceEnter` 和 `AudienceExit`
9. 方法 `updateSeatInfo()` 变更为 `updateSeatInfoExtra()`

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，请确认连接中的参数是否正确
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
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
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

状态码	说明
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
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
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

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