场景融合

语聊房

Android 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

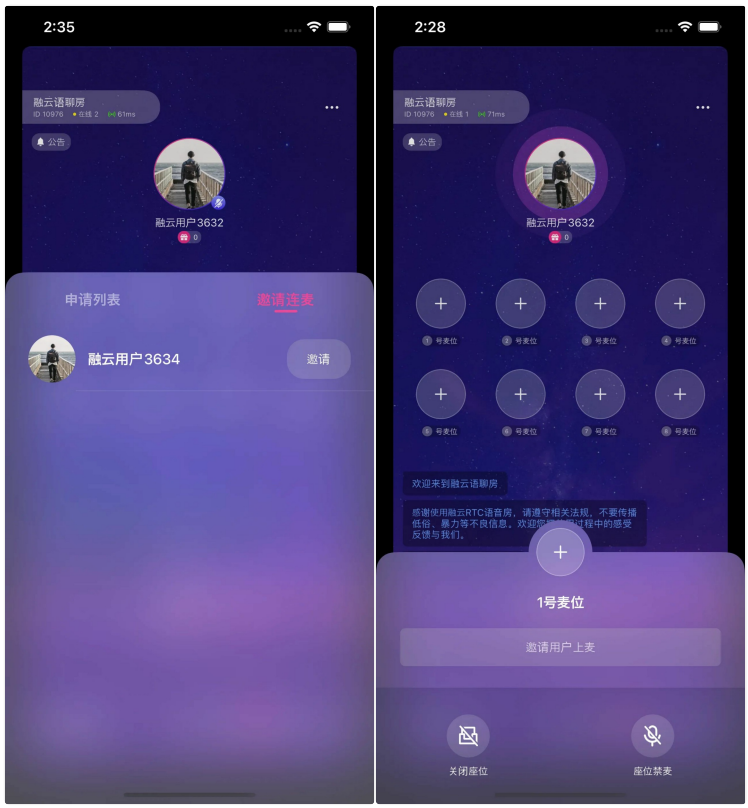
更新时间:2024-05-17

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。

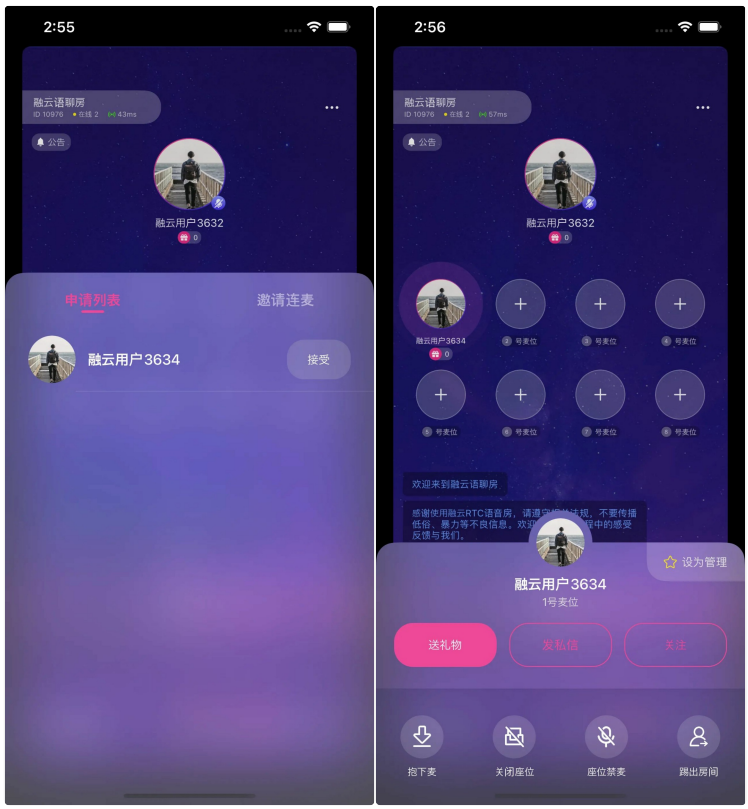


房间管理

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。

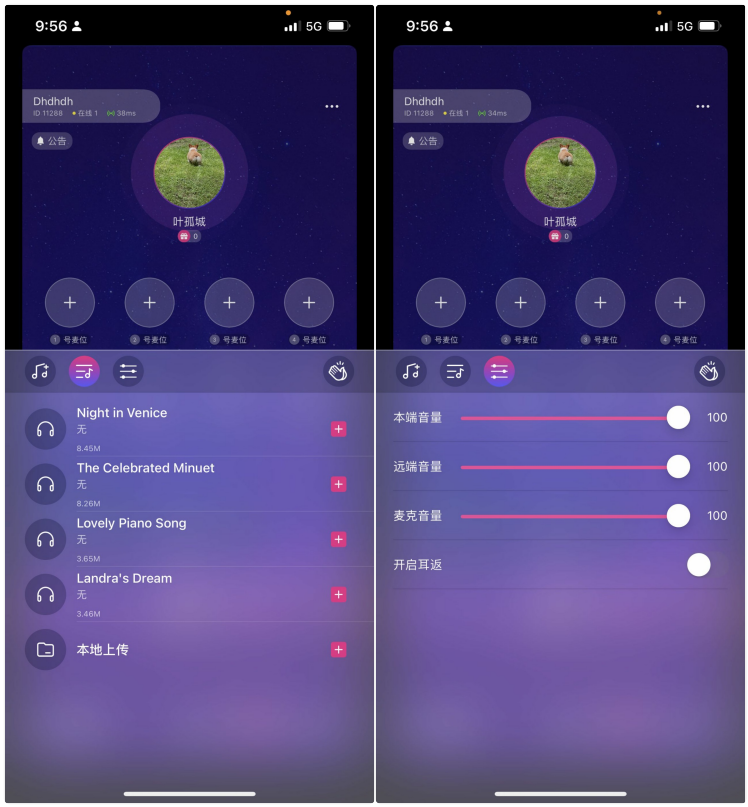


跨房间 PK

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。

集成 SDK 环境要求

更新时间:2024-05-17

- **AndroidStudio**：4.0+
- **Android**：5.0 及以上
- **Java**：java 8

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。

集成语聊房 SDK

1. 在 project 的 build.gradle 添加融云仓库

```
allprojects {
repositories {
// 融云 maven 仓库
maven { url "https://maven.rongcloud.cn/repository/maven-releases/" }
}
}
```

2. module 的 build.gradle 文件中的 dependencies 节点添加如下代码：

```
dependencies {
......
// im 建议替换最新版本
implementation 'cn.rongcloud.sdk:im_lib:5.1.7'
// rtc 建议替换最新版本
implementation 'cn.rongcloud.sdk:rtc_lib:5.1.14'
// 语聊房SDK 建议替换最新版本
implementation 'cn.rongcloud.sdk:voiceroom_lib:2.0.7.2'
......
}
```

3. **gradle 配置完成后，重新 build 项目。**

混淆配置

- **根据实际项目的依赖情况，选项对应的混淆配置：**
  1. **项目中依赖 imlib 和 rtclib，则添加 imlib，rtc 和 语聊房 SDK 的混淆配置即可**
  2. **项目中依赖 imkit 和 rtclib，则添加 imkit，rtc 和 语聊房 SDK 的混淆配置即可**
- **混淆配置有交叉重复的部分，可适当去重。**

1. IMLib 的混淆配置

```
# imkit 的混淆配置
-keepattributes Exceptions,InnerClasses
-keepattributes Signature
-keep class io.rong.** {*;}
-keep class cn.rongcloud.** {*;}
-keep class * implements io.rong.imlib.model.MessageContent {*;}
-dontwarn io.rong.push.**
-dontnote com.xiaomi.**
-dontnote com.google.android.gms.gcm.**
-dontnote io.rong.**

# 当代码中有继承 PushMessageReceiver 的子类时，需 keep 所创建的子类广播
# 把 io.rong.app.DemoNotificationReceiver 改成子类广播的完整类路径即可.
-keep class io.rong.app.DemoNotificationReceiver {*;}

-ignorewarnings
```

2. IMKit 的混淆配置

```
# imkit 的混淆配置
-keepattributes Exceptions,InnerClasses
-keepattributes Signature
-keep class io.rong.** {*;}
-keep class cn.rongcloud.** {*;}
-keep class * implements io.rong.imlib.model.MessageContent {*;}
-dontwarn io.rong.push.**
-dontnote com.xiaomi.**
-dontnote com.google.android.gms.gcm.**
-dontnote io.rong.**

# 下方混淆使用了Location包时才需要配置，可参考高德官网的混淆方
式:https://lbs.amap.com/api/android-sdk/guide/create-project/dev-attention
-keep class com.amap.api.**{*;}
-keep class com.amap.api.services.**{*;}
-keep class com.autonavi.**{*;}

# 当代码中有继承 PushMessageReceiver 的子类时，需 keep 所创建的子类广播
# 把 io.rong.app.DemoNotificationReceiver 改成子类广播的完整类路径即可.
-keep class io.rong.app.DemoNotificationReceiver {*;}

-ignorewarnings
```

3. RTC 混淆配置

```
# rtc 混淆配置
-keepattributes Exceptions,InnerClasses
-keepattributes Signature

-keep class io.rong.** {*;}
-keep class cn.rongcloud.** {*;}
-keep class * implements io.rong.imlib.model.MessageContent {*;}
-dontwarn io.rong.push.**
-dontnote com.xiaomi.**
-dontnote com.google.android.gms.gcm.**
-dontnote io.rong.**

# 下方混淆使用了Location包时才需要配置，可参考高德官网的混淆方
式:https://lbs.amap.com/api/android-sdk/guide/create-project/dev-attention
-keep class com.amap.api.**{*;}
-keep class com.amap.api.services.**{*;}
-keep class com.autonavi.**{*;}

# 当代码中有继承 PushMessageReceiver 的子类时，需 keep 所创建的子类广播
# 把 io.rong.app.DemoNotificationReceiver 改成子类广播的完整类路径即可.
-keep class io.rong.app.DemoNotificationReceiver {*;}

-ignorewarnings
```

4. 语聊房的混淆配置

```
# voiceroom 混淆配置
-keep class cn.rongcloud.voiceroom.api.** {*;}
-keep class cn.rongcloud.voiceroom.model.** {*;}
-keep class cn.rongcloud.voiceroom.utils.** {*;}
-keep class cn.rongcloud.messager.** {*;}
```

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。

依赖组件	版本
IMLib	5.1.6 及以上
RTCLib	5.1.12 及以上

初始化 SDK

更新时间:2024-05-17

依据应用内是否已集成融云 IMLib/IMKit，语聊房的初始化分为两种情况。

- 如果您是融云产品的新用户，或您的应用内没有集成过任何融云服务，请按照本文档步骤操作，使用 `RCVoiceRoomLib` 的初始化方式。
- 如果您的应用内已集成了融云 IMLib 或者 IMKit，请直接参见其他注意事项。

RCVoiceRoomLib 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。
> - 建议在 Application 中初始化。语聊房 SDK 依赖于 IM 的初始化，您可以使用 IMLib 或 IMKit 初始化，具体可以根据当前项目的实际依赖选择使用。

```
/**
* 初始化 appKey
* 注意：语聊房依赖于 IM 的初始化，您可以使用 im_lib 或 im_kit 初始化，具体可以根据当前项目的实际依赖选择使用 imlib 还是 imkit。
*
* @param context 当前应用的 application
* @param appKey 开发者申请的 appKey
*/
void init(Application context, String appKey);
```

代码示例

以下以 IMLib 初始化为例。

```
String process = UIKit.getCurrentProcessName();
if (!getPackageName().equals(process)) {
// 非主进程不初始化 避免过度初始化
return;
}
Log.d(TAG, "initVoiceRoom:process : " + process);
RongCoreClient.init(this, appKey);
```

连接融云服务

语聊房 SDK 依赖于 IMLib 或 IMKit。您可以使用 IMLib 或 IMKit 的连接方法，具体可以根据当前项目的实际依赖选择使用。

```
/**
* 连接融云服务器
* @param token 从融云服务器获取的 token
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
RongCoreClient connect(String token, ConnectCallback connectCallback)
```

示例代码

以下以 IMLib 连接方法为例。

```
RongCoreClient.connect(token, new IRongCoreCallback.ConnectCallback() {
@Override
public void onSuccess(String t) {
KToast.show("connect success");
}

@Override
public void onError(IRongCoreEnum.ConnectionErrorCode e) {
String info = "connect fail：\n【" + e.getValue() + "】" + e.name();
Log.e("ConnectActivity", info);
KToast.show(info);
}

@Override
public void onDatabaseOpened(IRongCoreEnum.DatabaseOpenStatus code) {

}
}
);
```

其他注意事项

- 已集成 **IMLib** 或者 **IMKit** 并设置过消息监听，如果 IMLib 版本低于 5.1.5，必须替换成语聊房的 addMessageReceiveListener，防止您的消息监听被覆盖，从而导致接收不到消息。对于版本高于 5.1.5 的用户，您也可以选择替换为语聊房的addMessageReceiveListener，便于代码阅读。

  以下示例说明如何替换您 app 中消息监听代码：

  ```
  RongCoreClient.setOnReceiveMessageListener(listener);
  // 或
  RongIM.addOnReceiveMessageListener(listener);

  // 若果app中有使用以上代码初始化的，建议修改如下：
  // 说明：
  // 1. 防止您的消息监听被替换，从而导致接收不到消息
  RCVoiceRoomEngine.getInstance().addMessageReceiveListener(listener)
  ```

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。

设置房间事件监听

创建一个语聊房首先需要设置房间事件监听，这样创建房间后监听才会全部回调。

```
// 设置房间事件监听，可以实现 RCVoiceRoomEventListener 监听房间内事件回调
RCVoiceRoomEngine.getInstance().setVoiceRoomEventListener(this);
```

配置 RTC 信息

RCRTCConfig 是初始化 RTC 引擎所需的配置信息，不传或传 null 内部会采用默认配置。

```
// 构建一个 RCRTCConfig，可根据自己需求配置信息。
RCRTCConfig config = RCRTCConfig.Builder.create().build();
```

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。

```
// 创建一个 RCVoiceRoomInfo 实例
RCVoiceRoomInfo rcVoiceRoomInfo = new RCVoiceRoomInfo();
// 设置房间名称
rcVoiceRoomInfo.setRoomName(roomName);
// 设置麦位数量
rcVoiceRoomInfo.setSeatCount(9);
// 设置上麦模式（ture是自由上麦 false为申请上麦）
rcVoiceRoomInfo.setFreeEnterSeat(false);
// 设置全麦锁座
rcVoiceRoomInfo.setLockAll(false);
// 设置全麦锁麦
rcVoiceRoomInfo.setMuteAll(false);
```

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.getInstance().createAndJoinRoom(config, roomId, rcVoiceRoomInfo, new
RCVoiceRoomCallback() {
@Override
public void onSuccess() {
//创建成功,自动加入房间
}

@Override
public void onError(int code, String message) {
//创建失败
}
});
```

加入房间

右图是内部大概的执行流程。

设置房间事件监听

加入一个语聊房首先需要设置房间事件监听，这样加入房间后监听才会全部回调。

```
// 设置房间事件监听，可以实现 RCVoiceRoomEventListener 监听房间内事件回调
RCVoiceRoomEngine.getInstance().setVoiceRoomEventListener(this);
```

配置 RTC 信息

RCRTCConfig 是初始化 RTC 引擎所需的配置信息，不传或传 null 内部会采用默认配置。

```
// 构建一个 RCRTCConfig，可根据自己需求配置信息。
RCRTCConfig config = RCRTCConfig.Builder.create().build();
```

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。

```
// roomId 是您的业务服务器返回的
RCVoiceRoomEngine.getInstance().joinRoom(config, roomId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {
}

@Override
public void onError(int code, String message) {
}
});
```

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

onRoomInfoUpdate 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。

```
// 任何房间信息的修改都会触发此回调。
public void onRoomInfoUpdate(RCVoiceRoomInfo rcVoiceRoomInfo){
// 可以刷新房间 UI 。
this.roomInfo = rcVoiceRoomInfo;
refreshRoomView();
}
```

麦位信息变化回调

onSeatInfoUpdate 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。
public void onSeatInfoUpdate(List<RCVoiceSeatInfo> list) {
// 保存最新麦位信息，并刷新 UI 。
self.seatlist = seatInfolist;
refreshSeatView();
}
```

KV准备完毕回调

onRoomKVReady 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
public void onRoomKVReady() {
// 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。

```
RCVoiceRoomEngine.getInstance().leaveRoom(new RCVoiceRoomCallback() {
@Override
public void onSuccess() {
}

@Override
public void onError(int code, String message) {
}
});
```

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

```
// 这里可以做销毁页面等逻辑
finish();
// 打印错误信息或其他操作
Log.e("leaveRoom", message);
```

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。

```
/**
* 踢出房间
* @param userId 要踢出的人的 id
*/
private void kickUser(String userId){
RCVoiceRoomEngine.getInstance().kickUserFromRoom(userId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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
*/
@Override
public void onUserReceiveKickOutRoom(String targetId, String userId) {
// 这里可以对比 targetId 和当前用户 id ，一致的话调用离开房间
if(TextUtils.equals(targetId, mUid)){
RCVoiceRoomEngine.getInstance().leaveRoom(null);
}
}
```

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 onRoomDestroy 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。

```
// 房间已关闭
@Override
public void onRoomDestroy() {
RCVoiceRoomEngine.getInstance().leaveRoom(new RCVoiceRoomCallback() {
@Override
public void onSuccess() {
finish();
}

@Override
public void onError(int i, String s) {
}
});
}
```

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 QuickDemo 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。

```
/**
* 上麦
* @param index 麦位号
*/
public void enterSeat(int index) {
RCVoiceRoomEngine.getInstance().enterSeat(index, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

处理上麦回调

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

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，如刷新 RecyclerView。

```
// 任何麦位的变化都会触发此回调。
@Override
public void onSeatInfoUpdate(List<RCVoiceSeatInfo> list) {
// 更新麦位 UI
}
```

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。

```
// 请求上麦
public void requestSeat() {
RCVoiceRoomEngine.getInstance().requestSeat(new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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
public void cancelRequestSeat() {
RCVoiceRoomEngine.getInstance().cancelRequestSeat(new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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 onRequestSeatListChanged 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
*/
@Override
public void onRequestSeatListChanged() {
// 房主或管理员接受处理排麦，根据业务确定
if(!isRoomOwner) return;
}
```

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。

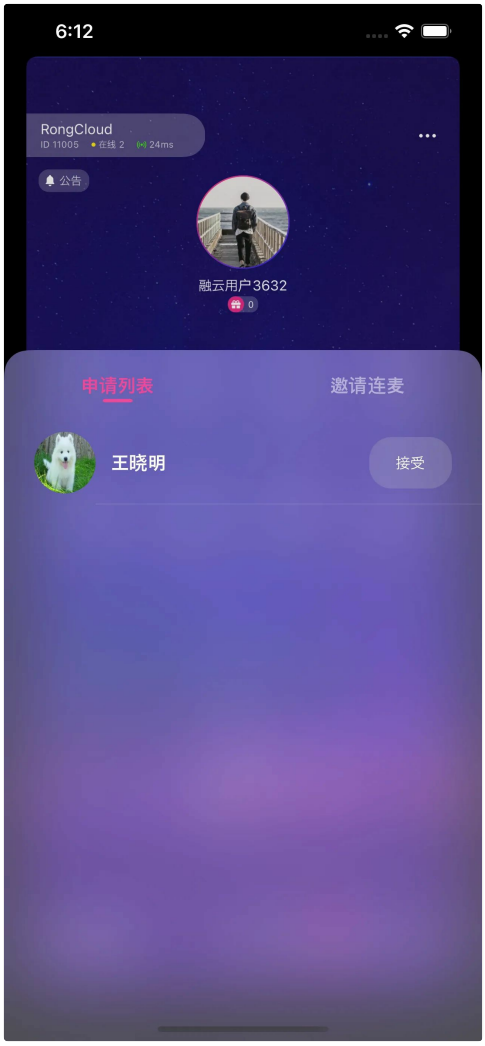
请根据您自身业务确定哪些用户有权限，可查询房间内最新的连麦申请列表。

```
// 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
public void getRequestSeatUserIds(){
RCVoiceRoomEngine.getInstance().getRequestSeatUserIds(new RCVoiceRoomResultCallback<List<String>>() {
@Override
public void onSuccess(List<String> strings) {

}

@Override
public void onError(int i, String s) {

}
});
}
```

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
*/
@Override
public void acceptRequestSeat(String userId) {
RCVoiceRoomEngine.getInstance().acceptRequestSeat(userId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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 `requestSeatDidAccept`。

在此方法中调用 enterSeat 即可成功上麦。

```
/**
* 用户申请连麦被接受
*/
@Override
public void onRequestSeatAccepted() {
// 房主或管理员同意上麦申请后，可以自己调用上麦操作
RCVoiceRoomEngine.getInstance().enterSeat(index, null);
}
```

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 请求排麦的用户id
*/
public void rejectRequestSeat(String userId) {
RCVoiceRoomEngine.getInstance().rejectRequestSeat(userId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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
@Override
public void onRequestSeatRejected() {

}
```

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 被邀请用户的 id
*/
public void pickUserToSeat(String userId) {
RCVoiceRoomEngine.getInstance().pickUserToSeat(userId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

被邀请用户收到通知

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

```
/**
* 用户被抱上麦 由 pickUserToSeat 触发
*
* @param userId 抱上麦操作的执行用户 Id
*/
@Override
public void onPickSeatReceivedFrom(String userId) {
// 这里可以弹框给被邀请用户，同意的话直接调用上麦，不同意可以不处理
new AlertDialog.Builder(context)
.setTitle("是否同意上麦邀请？")
.setPositiveButton("同意", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
// 同意直接上麦
RCVoiceRoomEngine.getInstance().enterSeat(index, null);
}
})
.setNegativeButton("拒绝", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

}
})
.show();
}
```

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
public void leaveSeat() {
RCVoiceRoomEngine.getInstance().leaveSeat(new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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。

```
/**
* 把用户踢下麦
* @param userId 要下麦的用户
*/
public void kickUserFromSeat(String userId){
RCVoiceRoomEngine.getInstance().kickUserFromSeat(userId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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。

```java
/**
* 被踢下麦回调
*
* @param index 麦位索引
*/
@Override
public void onKickSeatReceived(int index) {
RCVoiceRoomEngine.getInstance().leaveSeat(null);
}
```

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 onSeatInfoUpdate 回调，在此刷新 UI 即可。

```
/**
* 锁麦，注意：如果该麦位上有人，会被踢下麦
*
* @param seatIndex 麦位序号
* @param isLocked 是否锁麦位
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
public void lockSeat(int seatIndex, boolean isLocked) {
RCVoiceRoomEngine.getInstance().lockSeat(index, isLocked, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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 QuickDemo 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 `onSeatInfoUpdate` 回调，在此刷新 UI 即可。

```
/**
* 静麦，注意：
* 1、可以静麦自己也可以静麦其他人
* 2、muteSeat 和 disableAudioRecording 都是操作麦克风状态，后者只能操作自己的麦克风状态，而且修改麦克风状态，该状态不会同步给房间内的其他人
*
* @param seatIndex 麦位序号
* @param isMute 是否静音
* @param callback 结果回调
*/
public void muteSeat(int seatIndex, boolean isMute) {
RCVoiceRoomEngine.getInstance().muteSeat(seatIndex, isMute, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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。true 为全部锁定，false 为全部解除锁定。

```
private void setRoomInfo(RCVoiceRoomInfo roomInfo) {
roomInfo.setLockAll(true);
RCVoiceRoomEngine.getInstance().setRoomInfo(roomInfo, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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。true 为全部闭麦，false 为全部解除闭麦，即麦克风为打开状态。

```
private void setRoomInfo(RCVoiceRoomInfo roomInfo) {
roomInfo.setMuteAll(true);
RCVoiceRoomEngine.getInstance().setRoomInfo(roomInfo, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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
*/
public void lockOtherSeats(boolean isLock) {
RCVoiceRoomEngine.getInstance().lockOtherSeats(isLock, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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 是否静音
*/
public void muteOtherSeats(boolean isMute) {
RCVoiceRoomEngine.getInstance().muteOtherSeats(isMute, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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。

```
public void setRoomInfo(RCVoiceRoomInfo roomInfo){
roomInfo.setExtra("自定义数据");
RCVoiceRoomEngine.getInstance().setRoomInfo(roomInfo, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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。

```
@Override
public void onRoomInfoUpdate(RCVoiceRoomInfo room) {
// 这里做相关的业务处理
}
```

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。

```
// 停止本地麦克风收音
// @param isDisable 是否停止
RCVoiceRoomEngine.getInstance().disableAudioRecording(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
RCVoiceRoomEngine.getInstance().enableSpeaker(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
RCVoiceRoomEngine.getInstance().muteAllRemoteStreams(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
RCVoiceRoomEngine.getInstance().setAudioQuality(quality, 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。

```
// 通知房间所有用户消息
// @param name 消息名称
// @param content 内容
RCVoiceRoomEngine.getInstance().notifyVoiceRoom("refreshBackgroundImage", "https://abc.png", new
RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
```

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 onRoomNotificationReceived 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
* 该回调是由 notifyVoiceRoom api 触发
*
* @param name 自定义名称， 对应 notifyVoiceRoom 的 name
* @param content 自定义通知内容， 对应 notifyVoiceRoom 的 content
*/
@Override
public void onRoomNotificationReceived(String name, String 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
*/
public void sendPKInvitation(String inviteeRoomId,String inviteeUserId){
RCVoiceRoomEngine.getInstance().sendPKInvitation(inviteeRoomId, inviteeUserId, new RCVoiceRoomCallback()
{
@Override
public void onSuccess() {
}

@Override
public void onError(int code, String message) {
}
});
}
```

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
*/
@Override
public void onReceivePKInvitation(String inviterRoomId, String inviterUserId) {
// 收到 PK 邀请后可以弹框选择接受或者拒绝等操作，根据业务需求自己实现
}
```

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
*/
public void cancelPKInvitation(String inviteeRoomId, String inviteeUserId){
RCVoiceRoomEngine.getInstance().cancelPKInvitation(inviteeRoomId, inviteeUserId, new
RCVoiceRoomCallback() {
@Override
public void onSuccess() {
}

@Override
public void onError(int code, String message) {
}
});
}
```

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
*/
@Override
public void onPKInvitationCanceled(String inviterRoomId, String 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 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
- `PKResponse.ignore`：忽略 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
*/
public void responsePKInvitation(String inviterRoomId, String inviterUserId, PKResponse pkResponse){
RCVoiceRoomEngine.getInstance().responsePKInvitation(inviterRoomId, inviterUserId, pkResponse, new
RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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中字段
* inviterRoomId 邀请 PK 的用户所在房间id
* inviterUserId 邀请 PK 的用户id
* inviteeRoomId 被邀请 PK 的用户所在房间id
* inviteeUserId 被邀请 PK 的用户id
*/
@Override
public void onPKGoing(@NonNull RCPKInfo rcpkInfo) {

}
```

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
*/
@Override
public void onPKInvitationRejected(String inviteeRoomId, String inviteeUserId) {

}
```

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 initeeUserId 被邀请者的用户 id
*/
@Override
public void onPKInvitationIgnored(String inviteeRoomId, String 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 `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
*/
public void mutePKUser(boolean isMute){
RCVoiceRoomEngine.getInstance().mutePKUser(isMute, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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
*/
public void resumePk(String otherRoomId, String otherUserId){
RCVoiceRoomEngine.getInstance().resumePk(otherRoomId, otherUserId, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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
*/
public void quitPK(){
RCVoiceRoomEngine.getInstance().quitPK(new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
}
```

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 onPKFinish 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
*/
@Override
public void onPKFinish(){

}
```

扩展功能

更新时间:2024-05-17

扩展接口（IVoicePlugin），是对语聊房引擎（RCVoiceRoomEngine）中提供功能扩展或增强而提供的一组 api 的集合。具见语聊房扩展功能 。

RCVoiceRoomEngine 中的静态方法 getPlugin() 获取扩展接口实例。

```
IVoicePlugin plugin = RCVoiceRoomEngine.getPlugin()
```

扩展 API 使用示例 上麦（扩展）

enterSeat(index,seat,callback) 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。

```
RCVoiceSeatInfo seat = new RCVoiceSeatInfo();
//默认为false，如不修改该属性，会将默认值false赋值给目标麦位
seat.setMute(mute);
seat.setExtra(extra);
```

上麦

上麦，并携带以上构建的麦位信息。

```
RCVoiceRoomEngine.getPlugin().enterSeat(seatIndex,seat, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
```

麦位更新（扩展）

`updateSeat(index,seat,callback)` 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。

```
RCVoiceSeatInfo seat = new RCVoiceSeatInfo();
//默认为false，如不修改该属性，会将默认值 false 赋值给目标麦位
seat.setMute(mute);
seat.setExtra(extra);
```

更新

调用更新 api。

```
RCVoiceRoomEngine.getPlugin().updateSeat(seatIndex,seat, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int code, String message) {

}
});
```

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 重置为 null。

切换麦位并更新

构建空麦位，并赋值 mute & extra 属性。

```
// 重置当前麦位的 mute & extra
RCVoiceSeatInfo preSeat = new RCVoiceSeatInfo();
preSeat.setMute(false);
preSeat.setExtra(null);
// 修改 targetSeatInfo 的 mute & extra
RCVoiceSeatInfo targetSeat = new RCVoiceSeatInfo();
targetSeat.setMute(ture); // 跳麦后静音
targetSeat.setExtra("user info");// 将当前用户信息赋值给 extra
RCVoiceRoomEngine.getPlugin().switchSeatTo(targetSeatIndex,preSeat,targetSeat, new RCVoiceRoomCallback()
{
@Override
public void onSuccess() {}

@Override
public void onError(int code, String message) {}
});
```

切换麦位携带属性

切麦，并更新麦位信息。

```
// 参数1：目标麦位索引
// 参数2：是否携带 mute 属性。 true：携带
// 参数3：是否携带 extra 属性。true：携带
RCVoiceRoomEngine.getPlugin().switchSeatTo(targetSeatIndex,true,true, new RCVoiceRoomCallback() {
@Override
public void onSuccess() {
}

@Override
public void onError(int code, String message) {
}
});
```

状态相关（扩展）

- 是否启用SDK 内部是否启用麦位的状态 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

enableSeatStateLocked(enable) 可修改 SDK 内部是否启用麦位的状态 KV 方案来解决多用户同时抢麦的问题，SDK 默认不启用，注意：

- 1.不启用，多用户同时抢一个麦位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

```
// 参数：是否启用麦位状态 KV 方案解决多端抢麦，true：启用
RCVoiceRoomEngine.getPlugin().enableSeatStateLocked(true);
```

启用分发PK用户状态

enableDispatchPKUserState(enable) 是否启用内置的基于 KV 分发 PK 用户的音量，默认：不启用。

```
// 参数：是否启用内置的基于 KV 分发 PK 用户的音量，true：启用
RCVoiceRoomEngine.getPlugin().enableDispatchPKUserState(true);
```

清理异常麦位状态KV

clearSeatState(callback) 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。

```
// 清理异常麦位状态KV
RCVoiceRoomEngine.getPlugin().clearSeatState(new RCVoiceRoomResultCallback<List<String>>() {
@Override
public void onSuccess(List<String> data) {
// data 被清理异常 KV 的 key
Logger.e(TAG, "clearSeatState: data = " + GsonUtil.obj2Json(data));
}
@Override
public void onError(int code, String message) {
}
});
```

麦克风禁用状态

isDisableAudioRecording(userId) 可用于获取指定用户（本地/远端）的麦克风的禁用状态。

```
// 参数：房间内指定主播的userId，如果传的是当前用户的userId，亦可获取当前用户的麦克风的状态，远端用户需是主播。
// 返回：用户的麦克风禁用状态
boolean audioRecordDisable = RCVoiceRoomEngine.getPlugin().isDisableAudioRecording(userId);
```

语聊房房间模型 RCVoiceRoomInfo

更新时间:2024-05-17

```
/**
* 语聊房封装实体
*/
public class RCVoiceRoomInfo extends BaseInfo implements Parcelable, Cloneable {

/**
* 房间名
*/
@SerializedName("roomName")
private String mRoomName;
/**
* 房间座位数
*/
@SerializedName("seatCount")
private int mSeatCount;

/**
* 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
*/
@SerializedName("isFreeEnterSeat")
private boolean isFreeEnterSeat;

/**
* 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
*/
@SerializedName("isLockAll")
private boolean isLockAll;

/**
* 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
*/
@SerializedName("isMuteAll")
private boolean isMuteAll;

/**
* 拓展字段
*/
@SerializedName("extra")
private String extra;

public String getRoomName() {
return mRoomName;
}

public void setRoomName(String roomName) {
mRoomName = roomName;
}

public int getSeatCount() {
return mSeatCount;
}

public void setSeatCount(int seatCount) {
```

```
mSeatCount = seatCount;
}


public boolean isFreeEnterSeat() {
return isFreeEnterSeat;
}

public void setFreeEnterSeat(boolean freeEnterSeat) {
isFreeEnterSeat = freeEnterSeat;
}


public String getExtra() {
return extra;
}

public boolean isLockAll() {
return isLockAll;
}

public void setLockAll(boolean lockAll) {
isLockAll = lockAll;
}

public boolean isMuteAll() {
return isMuteAll;
}

public void setMuteAll(boolean muteAll) {
isMuteAll = muteAll;
}

public void setExtra(String extra) {
this.extra = extra;
}

/**
* 实体转json
* @return json
*/
public String toJson() {
return JsonUtils.toJson(this);
}

@Override
public String toString() {
return "RCVoiceRoomInfo{" +
"mRoomName='" + mRoomName + '\'' +
", mSeatCount=" + mSeatCount +
", isFreeEnterSeat=" + isFreeEnterSeat +
", isLockAll=" + isLockAll +
", isMuteAll=" + isMuteAll +
", extra='" + extra + '\'' +
'}';
}

@Override
public boolean equals(Object o) {
if (this == o) return true;
if (o == null || getClass() != o.getClass()) return false;
RCVoiceRoomInfo that = (RCVoiceRoomInfo) o;
return mSeatCount == that.mSeatCount &&
isFreeEnterSeat == that.isFreeEnterSeat &&
isLockAll == that.isLockAll &&
```

```
isMuteAll == that.isMuteAll &&
Objects.equals(mRoomName, that.mRoomName) &&
Objects.equals(extra, that.extra);
}

@Override
public int hashCode() {
return Objects.hash(mRoomName, mSeatCount, isFreeEnterSeat, isLockAll, isMuteAll, extra);
}


public RCVoiceRoomInfo() {
}


public RCVoiceRoomInfo clone() {
try {
return (RCVoiceRoomInfo) super.clone();
} catch (CloneNotSupportedException e) {
e.printStackTrace();
}
return null;
}

@Override
public int describeContents() {
return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
dest.writeString(this.mRoomName);
dest.writeInt(this.mSeatCount);
dest.writeByte(this.isFreeEnterSeat ? (byte) 1 : (byte) 0);
dest.writeByte(this.isLockAll ? (byte) 1 : (byte) 0);
dest.writeByte(this.isMuteAll ? (byte) 1 : (byte) 0);
dest.writeString(this.extra);
}

public void readFromParcel(Parcel source) {
this.mRoomName = source.readString();
this.mSeatCount = source.readInt();
int tmpMAudioQuality = source.readInt();
this.isFreeEnterSeat = source.readByte() != 0;
int tmpMScenario = source.readInt();
this.isLockAll = source.readByte() != 0;
this.isMuteAll = source.readByte() != 0;
this.extra = source.readString();
}

protected RCVoiceRoomInfo(Parcel in) {
this.mRoomName = in.readString();
this.mSeatCount = in.readInt();
int tmpMAudioQuality = in.readInt();
this.isFreeEnterSeat = in.readByte() != 0;
int tmpMScenario = in.readInt();
this.isLockAll = in.readByte() != 0;
this.isMuteAll = in.readByte() != 0;
this.extra = in.readString();
}

public static final Creator<RCVoiceRoomInfo> CREATOR = new Creator<RCVoiceRoomInfo>() {
@Override
public RCVoiceRoomInfo createFromParcel(Parcel source) {
return new RCVoiceRoomInfo(source);
}
```

```
}

@Override
public RCVoiceRoomInfo[] newArray(int size) {
return new RCVoiceRoomInfo[size];
}
};
```

语聊房麦位模型 RCVoiceSeatInfo

更新时间:2024-05-17

```
/**
* 语聊房麦位封装实体
*/
public class RCVoiceSeatInfo extends BaseInfo implements Parcelable, Cloneable {

public RCVoiceSeatInfo() {
}

/**
* 当前状态
*/
@SerializedName("status")
private RCSeatStatus mStatus = RCSeatStatus.RCSeatStatusEmpty;

/**
* 是否静音
*/
@SerializedName("mute")
private boolean mute;

/**
* 正在发言
*/
@SerializedName("speaking")
private boolean speaking;

/**
* 用户 Id
*/
@SerializedName("userId")
private String userId;

/**
* 拓展字段
*/
@SerializedName("extra")
private String extra;


public RCSeatStatus getStatus() {
return mStatus;
}

public void setStatus(RCSeatStatus status) {
mStatus = status;
}

public boolean isMute() {
return mute;
}

public void setMute(boolean mute) {
this.mute = mute;
}
```

```
public boolean isSpeaking() {
return speaking;
}

public void setSpeaking(boolean speaking) {
this.speaking = speaking;
}

public String getUserId() {
return userId;
}

public void setUserId(String userId) {
this.userId = userId;
}

public String getExtra() {
return extra;
}

public void setExtra(String extra) {
this.extra = extra;
}

/**
* 麦位状态枚举定义
*/
public enum RCSeatStatus {
/**
* 麦位空闲
*/
@SerializedName("0")
RCSeatStatusEmpty,

/**
* 麦位占用中
*/
@SerializedName("1")
RCSeatStatusUsing,

/**
* 麦位被锁
*/
@SerializedName("2")
RCSeatStatusLocking
}

@Override
public int describeContents() {
return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
dest.writeInt(this.mStatus == null ? -1 : this.mStatus.ordinal());
dest.writeByte(this.mute ? (byte) 1 : (byte) 0);
dest.writeByte(this.speaking ? (byte) 1 : (byte) 0);
dest.writeString(this.userId);
dest.writeString(this.extra);
}

public void readFromParcel(Parcel source) {
int tmpMStatus = source.readInt();
this.mStatus = tmpMStatus == -1 ? null : RCSeatStatus.values()[tmpMStatus];
this.mute = source.readByte() != 0;
```

```
this.mute = source.readByte() != 0;
this.speaking = source.readByte() != 0;
this.userId = source.readString();
this.extra = source.readString();
}


protected RCVoiceSeatInfo(Parcel in) {
int tmpMStatus = in.readInt();
this.mStatus = tmpMStatus == -1 ? null : RCSeatStatus.values()[tmpMStatus];
this.mute = in.readByte() != 0;
this.speaking = in.readByte() != 0;
this.userId = in.readString();
this.extra = in.readString();
}

public static final Creator<RCVoiceSeatInfo> CREATOR = new Creator<RCVoiceSeatInfo>() {
@Override
public RCVoiceSeatInfo createFromParcel(Parcel source) {
return new RCVoiceSeatInfo(source);
}

@Override
public RCVoiceSeatInfo[] newArray(int size) {
return new RCVoiceSeatInfo[size];
}
};

// 浅拷贝即可
public RCVoiceSeatInfo clone() {
try {
return (RCVoiceSeatInfo) super.clone();
} catch (CloneNotSupportedException e) {
e.printStackTrace();
}
return null;
}


@Override
public String toString() {
return "RCVoiceSeatInfo{" +
"mStatus=" + mStatus +
", mute=" + mute +
", speaking=" + speaking +
", userId='" + userId + '\'' +
", extra='" + extra + '\'' +
'}';
}

}
```

语聊房功能列表 IRCVoiceRoomEngine

更新时间:2024-05-17

```
/**
* 语聊房引擎接口
*/
public interface IRCVoiceRoomEngine {
/**
* 设置房间事件监听
* 注意：离开房间后内会自动将监听置空，因此再次加入房间时需要重新设置房间监听
* 建议1：避免加房间时的事件丢失，建议在加入房间前设置监听。
* 建议2：由于内部使用弱引用，故此处不能使用局部变量，离开房间时可不设置null，内部会自动置空，不影响内存回收。
*
* @param listener 事件监听
*/
void setVoiceRoomEventListener(RCVoiceRoomEventListener listener);

/**
* 设置rtc 的媒体服务地址，v2.1.0 新增
*
* @param url 服务地址
*/
void setMediaServerUrl(String url);

/**
* 增加消息监听
* 注意：若 IMLib 低于 5.1.5，该方法和 setMessageRecieveListener 会存在冲突；IMLib 在 5.1.5 及以上版本已做兼容。
* @param listener 接收消息回调 {@link IRongCoreListener.OnReceiveMessageListener}
*/
void addMessageReceiveListener(IRongCoreListener.OnReceiveMessageListener listener);

/**
* 移除消息监听
*
* @param listener 消息回调 {@link IRongCoreListener.OnReceiveMessageListener}
*/
void removeMessageReceiveListener(IRongCoreListener.OnReceiveMessageListener listener);

/**
* 创建并加入房间
* 注意：
* 1、不自动上麦，需根据业务确定是否手动上麦
* 2、如果房间存在，会重置房间的状态、KV、麦位信息
* 3、语聊房的生命周期内，只执行一次，如果多次执行，会重置之前的房间信息
*
* @param roomId 房间唯一标识
* @param roomInfo 房间信息{@link RCVoiceRoomInfo}
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
@Deprecated
void createAndJoinRoom(String roomId, RCVoiceRoomInfo roomInfo, RCVoiceRoomCallback callback);

/**
* 创建并加入房间
* 注意：
* 1、不自动上麦，需根据业务确定是否手动上麦
* 2、如果房间存在，会重置房间的状态、KV、麦位信息
```

```
* 3、语聊房的生命周期内，只执行一次，如果多次执行，会重置之前的房间信息
*
* @param rcrtcConfig RTC引擎初始化的配置信息
* @param roomId 房间唯一标识
* @param roomInfo 房间信息{@link RCVoiceRoomInfo}
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void createAndJoinRoom(RCRTCConfig rcrtcConfig, String roomId, RCVoiceRoomInfo roomInfo,
RCVoiceRoomCallback callback);

/**
* 加入语聊房，使用兼容耳返的默认配置初始化 RTC 引擎
* 注意：
* 1、如果房间不存在会报错
* 2、和 leaveRoom 成对调用
* 3、如果连续调用 joinRoom ，即中间没有调用 leaveRoom ，会报错状态异常：可查看 LogCat
* 4、如果已报状态异常，导致无法加入房间：可执行 RCRTCEngine.getInstance().unInit() 重置RTC的缓存状态
*
* @param roomId 房间唯一标识
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
@Deprecated
void joinRoom(String roomId, RCVoiceRoomCallback callback);

/**
* 加入语聊房，注意：
* 1、如果房间不存在会报错
* 2、和 leaveRoom 成对调用
* 3、如果连续调用 joinRoom，即中间没有调用 leaveRoom，会报错状态异常：可查看 LogCat
* 4、如果已报状态异常，导致无法加入房间：可在加入房间前 RCRTCEngine.getInstance().unInit() 重置RTC的缓存状态
*
* @param rcrtcConfig RTC 引擎初始化的配置信息
* @param roomId 房间唯一标识
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void joinRoom(RCRTCConfig rcrtcConfig, String roomId, RCVoiceRoomCallback callback);

/**
* 离开房间，注意：
* 1、和 joinRoom/createAndJoinRoom 成对调用
* 2、如果 joinRoom 后异常退出，导致无法调用 leaveRoom，再次重启 joinRoom 会报状态异常，
* 可在加入房间前RCRTCEngine.getInstance().unInit() 重置RTC的缓存状态
*
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void leaveRoom(RCVoiceRoomCallback callback);

/**
* 用户上麦
*
* @param seatIndex 麦位序号
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void enterSeat(int seatIndex, RCVoiceRoomCallback callback);

/**
* 尝试重启音频采集模块，并重新发布资源，v2.0.9 新增
* 注意：
* 1.麦上主播尝试调用才生效。
* 2.上麦前未申请权限，导致音频模块启动失败，导致上麦后未能正常发布资源。使用此 api 尝试重启音频采集模块，并重新发布资源
*
* @param callback 回调
*/
void republishStream(RCVoiceRoomCallback callback);

/**
```

```
/**
* 用户下麦
* 注意：和 enterSeat()成对调用
*
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void leaveSeat(RCVoiceRoomCallback callback);

/**
* 用户跳麦
* 注意：
* 1、只有已在麦位上的用户可跳麦 （即切换麦位）
* 2、不在麦位上的用户如果想上麦，仅可使用 enterSeat 方法
*
* @param seatIndex 需要跳转的麦位序号
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void switchSeatTo(int seatIndex, RCVoiceRoomCallback callback);

/**
* 抱用户上麦
*
* @param userId 用户 Id
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void pickUserToSeat(String userId, RCVoiceRoomCallback callback);

/**
* 踢人下麦
*
* @param userId 用户 Id
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void kickUserFromSeat(String userId, RCVoiceRoomCallback callback);

/**
* 将用户踢出房间
*
* @param userId 用户 Id
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void kickUserFromRoom(String userId, RCVoiceRoomCallback callback);

/**
* 锁麦，注意：如果该麦位上有人，会被踢下麦
*
* @param seatIndex 麦位序号
* @param isLocked 是否锁麦位
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void lockSeat(int seatIndex, boolean isLocked, RCVoiceRoomCallback callback);

/**
* 静麦，注意：
* 1、可以静麦自己也可以静麦其他人
* 2、muteSeat 和 disableAudioRecording 都是可理解为操作麦克风状态。但 disableAudioRecording 只能操作自己的麦克风
状态，而且对麦克风状态的修改不会同步给房间内的其他人
*
* @param seatIndex 麦位序号
* @param isMute 是否静音
* @param callback 结果回调
*/
void muteSeat(int seatIndex, boolean isMute, RCVoiceRoomCallback callback);

/**
* 将除自己外的所有麦位:静麦/取消静麦
```

```
* 注意：
* 1、如果 isMute = true，除自己外还有8个麦位，其中有3个已是静麦状态，此时只会静麦其他5个麦位，房间内的其他成员也只会接收到这5个麦位被静麦的回调
*
* @param isMute 是否静麦
*/
void muteOtherSeats(boolean isMute, RCVoiceRoomCallback callback);

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

/**
* 将除自己外的所有麦位:锁定/解锁
*
* @param isLock 是否锁麦
*/
void lockOtherSeats(boolean isLock, final RCVoiceRoomCallback callback);

/**
* 发送房间消息
*
* @param message 融云消息实体
* @param callback 结果回调
*/
void sendRoomMessage(MessageContent message, RCVoiceRoomCallback callback);

/**
* 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 {@link RCVoiceRoomInfo}
* @param callback 结果回调
*/
void setRoomInfo(RCVoiceRoomInfo roomInfo, RCVoiceRoomCallback callback);

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

/**
* 本地麦克风是否被禁用
*/
boolean isDisableAudioRecording();

/**
* 设置房间音频质量和场景
*
* @param audioQuality 音频质量
* @param scenario 音频场景
*/
void setAudioQuality(AudioQuality audioQuality, AudioScenario scenario);

/**
* 是否使用扬声器
*
* @param isEnable 是否使用
*/
void enableSpeaker(boolean isEnable);
```

```
/**
* 申请上麦
*
* @param callback 结果回调
*/
void requestSeat(RCVoiceRoomCallback callback);

/**
* 取消申请上麦
*
* @param callback 结果回调
*/
void cancelRequestSeat(RCVoiceRoomCallback callback);

/**
* 同意用户的上麦请求
*
* @param userId 请求排麦的用户 Id
* @param callback 结果回调
*/
void acceptRequestSeat(String userId, RCVoiceRoomCallback callback);

/**
* 拒绝用户上麦请求
*
* @param userId 被拒绝的申请人的Id
* @param callback 结果回调
*/
void rejectRequestSeat(String userId, RCVoiceRoomCallback callback);

/**
* 发送自定义请求
*
* @param content 发送的请求内容
* @param callback 结果回调
*/
void sendInvitation(String content, RCVoiceRoomResultCallback<String> callback);

/**
* 取消自己发出的请求
*
* @param invitationId 请求 Id
* @param callback 结果回调
*/
void cancelInvitation(String invitationId, RCVoiceRoomCallback callback);

/**
* 拒绝请求
*
* @param invitationId 请求的 Id
* @param callback 结果回调
*/
void rejectInvitation(String invitationId, RCVoiceRoomCallback callback);

/**
* 同意请求
*
* @param invitationId 请求 Id
* @param callback 结果回调
*/
void acceptInvitation(String invitationId, RCVoiceRoomCallback callback);

/**
* 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
* @param callback 结果回调
*/
void notifyVoiceRoom(String name, String content, RCVoiceRoomCallback callback);

/**
* 获取最近所有排麦申请的用户id
*
* @param callback 结果回调
*/
void getRequestSeatUserIds(RCVoiceRoomResultCallback<List<String>> callback);

/**
* 获取最新麦位信息
*
* @param resultCallback 结果回调
*/
void getLatestSeatInfo(final RCVoiceRoomResultCallback<List<RCVoiceSeatInfo>> resultCallback);

/**
* 更新指定麦位信息中的extra字段
*
* @param index 麦位索引
* @param extra extra字段值
*/
void updateSeatInfo(int index, String extra, RCVoiceRoomCallback callback);

/************************************
* voice room 2.0
************************************/

/**
* 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 callback 结果回调
*/
void sendPKInvitation(String inviteeRoomId, String inviteeUserId, RCVoiceRoomCallback callback);

/**
* 取消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 callback 结果回调
*/
void cancelPKInvitation(String inviteeRoomId, String inviteeUserId, RCVoiceRoomCallback callback);

/**
* 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
* @param callback 结果回调
*/
void responsePKInvitation(String inviterRoomId, String inviterUserId, PKResponse pkResponse,
RCVoiceRoomCallback callback);

/**
* 屏蔽PK对象的语音
*
* @param isMute 是否静音
* @param callback 结果回调
```

```
*/
void mutePKUser(boolean isMute, RCVoiceRoomCallback callback);

/**
* 退出PK
*/
void quitPK(RCVoiceRoomCallback callback);

/**
* 快速进入pk
* 注意：
* 1、此方法跳过邀请，直接进入pk阶段
* 2、执行该方法以后，只能执行quitPk 退出pk
*
* @param inviteeRoomId 被邀请用户所在的房间id
* @param inviteeUserId 被邀请人的用户id
*/
void resumePk(String inviteeRoomId, String inviteeUserId, RCVoiceRoomCallback callback);

}
```

语聊房扩展功能列表 IRCVoiceRoomEngine

更新时间:2024-05-17

```
/**
* 语聊房扩展接口
*
* 1、获取扩展接口实例
* RCVoiceRoomEngine.getPlugin();
* 2、使用示例 以 enterSeat为例：
* RCVoiceSeatInfo seat = new RCVoiceSeatInfo();
* seat.setMute(true); //默认为false，如不修改该属性，会将默认值false赋值给目标麦位
* seat.setExtra("麦上啦");
* RCVoiceRoomEngine.getPlugin().enterSeat(0,seat, null);
*/
public interface IVoicePlugin {
/**
* 获取当前房间RCRTCLiveInfo
* 注意：
* 1、供基于语聊房开发插件使用
*
* @return RCRTCLiveInfo
*/
RCRTCLiveInfo getLiveInfo();

/**
* 创建并加入房间
* 注意：
* 1、不自动上麦，需根据业务确定是否手动上麦
* 2、如果房间存在，会重置房间的状态、KV、麦位信息
* 3、语聊房的生命周期内，只执行一次，如果多次执行，会重置之前的房间信息
*
* @param roomId 房间唯一标识
* @param roomInfo 房间信息{@link RCVoiceRoomInfo}
* @param rtcConfig RTC引擎初始化的配置信息
* @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。
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void createAndJoinRoom(String roomId, RCVoiceRoomInfo roomInfo, RCRTCConfig rtcConfig, boolean
enableSeatStateLock, RCVoiceRoomCallback callback);

/**
* 加入语聊房，注意：
* 1、如果房间不存在会报错
* 2、和 leaveRoom 成对调用
* 3、如果连连续调用 joinRoom，即中间没有调用 leaveRoom，会报错状态异常：可查看 LogCat
* 4、如果已报状态异常，导致无法加入房间：可在加入房间前 RCRTCEngine.getInstance().unInit() 重置RTC的缓存状态
*
* @param roomId 房间唯一标识
* @param rtcConfig RTC 引擎初始化的配置信息
* @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。
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void joinRoom(String roomId, RCRTCConfig rtcConfig, boolean enableSeatStateLock, RCVoiceRoomCallback
callback);

/**
* 更新麦位信息，主要更新 seat.extra & seat.mute
```

```
* 注意：
* 1、供特定场景使用（不推荐）
* 2、seat.extra 不为空，才会同步设置，seat.extra 为空时会跳过，不设置。
* 3、seat.mute 无论 true 或 false 都会被设置
*
* @param index 麦位索引
* @param seat 待更新的麦位信息，目前只支持更新 seat.mute & seat.extra
* @param callback 结果回调
*/
void updateSeatInfo(int index, RCVoiceSeatInfo seat, RCVoiceRoomCallback callback);

/**
* 上麦，并更新目标麦位信息，目前只支持 seat.extra & seat.mute
* 注意：
* 1、供特定场景使用（不推荐）
* 2、seat.extra 不为空，才会同步设置，seat.extra 为空时会跳过，不设置。
* 3、seat.mute 无论true 或 false 都会被覆盖设置
*
* @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.mute & seat.extra
* @param callback 结果回调
*/
void enterSeat(int seatIndex, RCVoiceSeatInfo seat, RCVoiceRoomCallback callback);

/**
* 用户跳麦,并携带麦位信息
* 注意：
* 1、供特定场景使用（不推荐）
*
* @param seatIndex 目标麦位索引
* @param switchMute true：携带 mute，将当前麦位的 mute 赋值给目标麦位，且重置当前麦位的 mute 属性； false: 不携带
* @param switchExtra true：携带 extra，将当前麦位的 extra 赋值给目标麦位，且重置当前麦位的 extra 属性； false: 不
携带
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void switchSeatTo(int seatIndex, boolean switchMute, boolean switchExtra, RCVoiceRoomCallback callback);

/**
* 用户跳麦,并同时修改目标麦位和当前麦位的信息。
* 注意：
* 1、供特定场景使用（不推荐）
*
* @param seatIndex 目标麦位索引
* @param preSeat 待赋值给当前麦位的信息，切麦时会将 preSeat 的 mute & 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 mute & extra 赋值给目标麦位。
* @param callback 回调
*/
void switchSeatTo(int seatIndex, RCVoiceSeatInfo preSeat, RCVoiceSeatInfo targetSeat,
RCVoiceRoomCallback callback);

/**
* 下麦,并重置麦位信息
* 注意：
* 1、供特定场景使用（不推荐）
*
* @param resetMute true: extra 重置为 null； false: 不重置
* @param resetExtra true: mute重置为 false； false: 不重置
* @param callback 回调
*/
void leaveSeat(boolean resetMute, boolean resetExtra, RCVoiceRoomCallback callback);

/**
* 离开房间
* 注意：
* 1、供特定场景使用（不推荐）
*
```

```
* @param resetMute true: 重置extra为null ； false: 不修改
* @param resetExtra true: 重置mute 为false； false: 不修改
* @param callback 结果回调 {@link RCVoiceRoomCallback}
*/
void leaveRoom(boolean resetMute, boolean resetExtra, RCVoiceRoomCallback callback);

/**
* 是否启用麦位状态锁，处理多端抢麦问题。SDK默认不启用。
* 注意：
* 1、如果不启用，会有多端同事抢麦的问题
* 2、如果启用，需服务端接入异常退出的场景，并移除服状态锁的KV对，key：RCSPlaceHolderKey_seat_+ 麦位索引
*
* @param enable true: 启用，false: 不启用
*/
void enableSeatStateLocked(boolean enable);

/**
* 清理 麦位异常状态
* 注意：
* 1、只有加入房间成功后会生效
*
* @param callback 回调，返回被清理麦位状态KV的key的集合
*/
void clearSeatState(RCVoiceRoomResultCallback<List<String>> callback);

/**
* 是否启用内置 分发PK User state,默认false，不启用
*
* @param enable 是否启用
*/
void enableDispatchPKUserState(boolean enable);

/**
* 获取指定用户的麦克风是否禁用标识
*
* @param userId 指定用户的麦位禁用状态
*/
boolean isDisableAudioRecording(String userId);
}
```

语聊房回调列表 RCVoiceRoomEventListener

更新时间:2024-05-17

```
/**
* 语聊房事件监听
*/
public interface RCVoiceRoomEventListener {
/**
* 房间KV准备就绪
*/
void onRoomKVReady();

/**
* 房间销毁回调
*/
void onRoomDestroy();

/**
* 房间信息变更
*
* @param roomInfo 房间信息 {@link RCVoiceRoomInfo}
*/
void onRoomInfoUpdate(RCVoiceRoomInfo roomInfo);

/**
* 房间座位变更
* 注意：自身上麦或下麦也会触发此回调
*
* @param seatInfoList 座位列表信息 {@link RCVoiceRoomInfo}
*/
void onSeatInfoUpdate(List<RCVoiceSeatInfo> seatInfoList);

/**
* 主播上麦
* 注意：自己上麦也会触发此回调
*
* @param seatIndex 麦位号
* @param userId 用户Id
*/
void onUserEnterSeat(int seatIndex, String userId);

/**
* 主播下麦
* 注意：自己下麦也会触发此回调
*
* @param seatIndex 麦位号
* @param userId 用户Id
*/
void onUserLeaveSeat(int seatIndex, String userId);

/**
* 座位静音状态回调
*
* @param index 座位号
* @param isMute 静音状态
*/
void onSeatMute(int index, boolean isMute);
```

```java
/**
* 座位锁定状态回调
*
* @param index 座位号
* @param isLock 是否关闭
*/
void onSeatLock(int index, boolean isLock);

/**
* 用户进房回调
* 注意:调用 joinRoom 会触发房间内其他成员的这个回调
*
* @param userId 观众 Id
*/
void onAudienceEnter(String userId);

/**
* 用户退房回调
*
* @param userId 观众 Id
*/
void onAudienceExit(String userId);

/**
* 用户音量变动回调
*
* @param seatIndex 麦位序号
* @param audioLevel 音量 index > 0 可认为在说话
*/
void onSpeakingStateChanged(int seatIndex, int audioLevel);

/**
* 用户音量变动回调,v2.1.1 新增
*
* @param userId 用户Id
* @param audioLevel 音量 index > 0 可认为在说话
*/
default void onUserSpeakingStateChanged(String userId, int audioLevel) {
}

/**
* 用户音量变动回调（批量）, v2.1.2 新增
*
* @param audioLevels 音量信息集合 {@link AudioLevel}
*/
default void onSpeakingStateChanged(List<AudioLevel> audioLevels) {
}

/**
* 收取消息回调
*
* @param message 收到的消息
*/
void onMessageReceived(Message message);

/**
* 房间通知回调
* 该回调是由 notifyVoiceRoom api 触发
*
* @param name 自定义名称， 对应 notifyVoiceRoom 的 name
* @param content 自定义通知内容， 对应 notifyVoiceRoom 的 content
*/
void onRoomNotificationReceived(String name, String content);

/**
```

```
/**
* 用户被抱上麦 由 pickUserToSeat 触发
*
* @param userId 发起抱上麦操作的用户 ID。注意，非被抱上麦的用户 ID
*/
void onPickSeatReceivedFrom(String userId);

/**
* 被下麦回调 由 kickUserFromSeat 触发
*
* @param index 麦位序号
*/
void onKickSeatReceived(int index);

/**
* 房主或管理员同意用户的上麦申请
* 由 requestSeat 发出的上麦请求被同意时，触发此回调
*/
void onRequestSeatAccepted();

/**
* 发送的上麦请求被房主或管理员拒绝
* 由 requestSeat 发出的排麦请求被拒绝时，触发此回调
*/
void onRequestSeatRejected();

/**
* 上麦请求列表发生变化
* 由 requestSeat 发出的排麦请求列表有变更时，触发此回调
*/
void onRequestSeatListChanged();

/**
* 收到自定义邀请回调
*
* @param invitationId 邀请标识 Id
* @param userId 发送邀请用户的标识
* @param content 邀请内容 （用户可以自定义）
*/
void onInvitationReceived(String invitationId, String userId, String content);

/**
* 自定义邀请被接受回调
*
* @param invitationId 邀请标识 Id
*/
void onInvitationAccepted(String invitationId);

/**
* 自定义邀请被拒绝回调
*
* @param invitationId 邀请标识 Id
*/
void onInvitationRejected(String invitationId);

/**
* 自定义邀请被取消回调
*
* @param invitationId 邀请标识 Id
*/
void onInvitationCancelled(String invitationId);

/**
* 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
*/
void onUserReceiveKickOutRoom(String targetId, String userId);

/*************************************
* voice room 2.0
*************************************/

/**
* 网络延迟
*
* @param delayMs 延迟时间 单位:MS
*/
void onNetworkStatus(int delayMs);

/**
* 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
*/
void onPKGoing(RCPKInfo rcpkInfo);

/**
* 结束PK时会触发此回调
*/
void onPKFinish();

/**
* 收到邀请PK邀请回调
*
* @param inviterRoomId 邀请者的房间id
* @param inviterUserId 邀请者的用户id
*/
void onReceivePKInvitation(String inviterRoomId, String inviterUserId);

/**
* pk邀请被邀请者取消回调
* 邀请和被邀请双方都会触发
*
* @param inviterRoomId 邀请者的房间id
* @param inviterUserId 邀请者的用户id
*/
void onPKInvitationCanceled(String inviterRoomId, String inviterUserId);

/**
* PK邀请被受邀请者拒绝回调
*
* @param inviteeRoomId 被邀请者的房间id
* @param inviteeUserId 被邀请者的用户id
*/
void onPKInvitationRejected(String inviteeRoomId, String inviteeUserId);

/**
* PK邀请被忽略回调
*
* @param inviteeRoomId 被邀请人的房间id
* @param inviteeUserId 被邀请者的用户id
*/
void onPKInvitationIgnored(String inviteeRoomId, String inviteeUserId);

/**
* 用户的麦克风禁用状态变更回调,目前在PK时，只有PK双方能接收到对方麦克风状态变更, v2.1.2 新增
*
* @param roomId 用户所在的房间Id
* @param userId 用户Id
* @param disable 麦克风禁用状态 true:禁用 false:启用
```

```
* @param disable 麦克风禁用状态 true：禁用 false：启用
*/
void onUserAudioRecordingDisable(String roomId, String userId, boolean disable);
}
```

调试日志

更新时间:2024-05-17

语聊房内部打印了 Log 信息，用户可以选择在 Debug 模式下开启日志，方便查询 SDK 内部运行顺序及错误处理。

配置

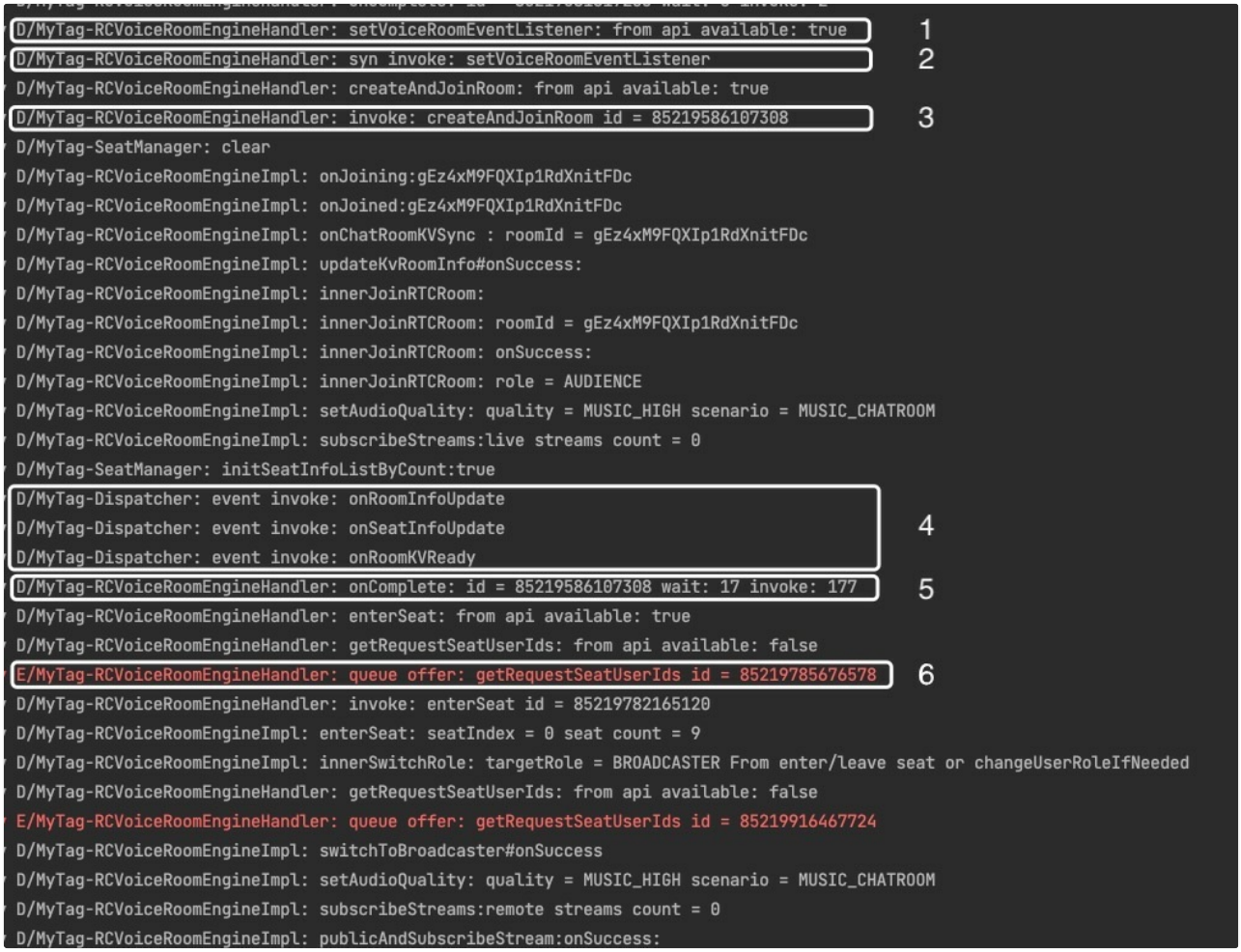
以下方法，用户可以根据具体情况进行配置。

```
// 默认 true，true：打印 log 信息，false：不打印
VMLog.setDebug(true);
// 自定义 Tag 前缀
VMLog.setCustomerPreTag("MyTag-");
// 设置日志级别，默认 DetailLevel.msg
// DetailLevel.msg ：简单的 log 信息
// DetailLevel.strack ：带堆栈的 log 信息
// DetailLevel.detailed ：带堆栈及设备版本信息等
VMLog.setDetailLevel(DailLevel.msg);
// 打开 Api 执行记录，默认 false
VMLog.setOpenInvokeRecord(true);
// 实时打印 Api 调用记录
VMLog.showInvokeRecords("ApiRecords");
// 将语聊房日志写到本地SD卡上，保存目录：sd卡/Android/data/包名/files/logger/
VMLog.setLogToLocal(true);
```

日志格式

打开 Api 执行记录后，可过滤设置的前缀，得到下图的 Log 信息。语聊房 SDK 内部维护了一个线程和队列，用户调用 Api 时，如果当前线程没任务执行，则立即执行，否则先加入队列，等待线程空闲后从队列里取出再执行。

示例图：



1. Api 调用，图中 ① 。
   - `setVoiceRoomEventListener` 为调用的 Api 名称。
   - `from api available: true` 当前任务是否可执行。
2. Api 执行，图中 ② ③ 。
   - ② `sync invoke` 标记同步执行，后面为 Api 名称。
   - ③ `invoke` 标记异步执行，后面为 Api 名称和 Id，和 ⑤ 有对应关系。
3. SDK 内部回调，图中 ④ 。
   - `event invoke` 标记内部回调，后面为回调的名称。
4. 异步执行结果，图中 ⑤ 。
   - `onComplete` 标记结束，后面为任务 Id 。
   - 和 ③ 是对应关系，即 ③ ~ ⑤ 为一个 Api 调用完整流程。
5. 任务放入队列等待执行，图中 ⑥ 。
   - `queue offer` 标记任务被放入队列，后面为 Api 名称和 Id。

更新日志 2.1.2.1

更新时间:2024-05-17

> **发布日期：**2022/11/17

1. 创建房间时添加绑定 RTC 房间。
2. 修复注册消息类型、KV 监听、消息监听后，可能因 IMLib 的 `unInit()` 导致的丢失问题。

2.1.1.1

> **发布日期：**2022/10/12

优化：

1. 添加 监听用户掉线或离开房间移除麦位状态 KV

新增：

1. 扩展 clearSeatState(callback)
2. 扩展 enableSeatStateLocked(enable);

v2.1.1

> **发布日期：**2022/9/26

新增：

1. RCVoiceRoomEngine 类中添加扩展接口 VoicePlugin 实例的静态方法 getPlugin()
2. 扩展 updateSeatInfo(index, seat, callback)
3. 扩展 enterSeat(index, seat, callback)
4. 扩展 switchSeatTo(index,preSeat,targetSeat,callback)
5. 扩展 switchSeatTo(index,switchMute,switchExtra,callback)

v2.1.0.1

> **发布日期：**2022/9/20

优化：

1. 添加 onUserSpeakingStateChange() 回调,可用于处理pk双方的音量回调
2. 移除 音频路由管理类（RCRTCAudioRouteManager） 的 init 和 unInit

修复：

1. 修复因房间内主播离开房间导致的 onPKFinish 回调
2. 修复getInstance()偶现的ANR

v2.1.0

> 发布日期：2022/9/09

优化：

1. 优化多端同时抢麦

新增：

1. 添加setMediaUrl()

v2.0.9

> 发布日期：2022/8/02

优化：

1. 优化logcat输出的错误提示信息，包含必要参数和解决方案步骤 和 底层 im/rtc 的错误码 和错误描述信息。

新增：

1. setLogToLocal（toLocal)
2. republishStream()

v2.0.8.4

> 发布日期：2022/6/28

优化：

1. 优化上麦

v2.0.8.3

> **发布日期：**2022/6/01

优化：

1. 优化上麦逻辑，移除切换为主播成功后的重复发布流。
2. 移除 2.0.8.2 版本添加的发布流成功后设置音频质量和场景。

修复：

1. 修改房间内主播回调两次麦位更新回调的问题

v2.0.8.2

优化：

1. 为优化资源占用，移除耳返兼容配置
2. 兼容 rtc 5.1.17 版本，发布流成功后设置音频质量和场景
3. log 系统中，默认关闭 API 执行记录

修复：

1. 修改因首次设置空的房间监听引起的控制指针问题
2. 修改因本地流为空引起的空指针问题

v2.0.8.1

> **发布日期：**2022/3/30

1. 内部添加阻塞队列
2. 修复多端抢麦的状态异常

v2.0.8

> **发布日期：**2022/2/23

1. 统一日志输出
2. 修复 pk 双方一端异常退出后 pk 不能结束

v2.0.7.2

> 发布日期：2022/1/17

新增接口

1. `isDisableAudioRecording()`: 获取本地麦克风的状态
2. `onRoomDestroy()`: 房间销毁回调

变更接口

1. 参数变更：`onSpeakingStateChanged(int seatIndex, boolen speaking)` 变更为 `onSpeakingStateChanged(int seatIndex, int audioLevel)`
2. 拼写变更：`onReveivePKInvitation()` 变更为 `onReceivePKInvitation()`
3. 拼写变更：`onPKgoing()` 变更为 `onPKGoing()`

移除接口

1. initWithAppKey()
2. connectWithToken()
3. disconnect()

v2.0.3

> 发布日期：2021/12/2

1. Android 编译环境适配 Gradle 7+

v2.0.2

> 发布日期：2021/11/16

1. 将房间事件的回调从 API 执行线程切回到 UI 线程。

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
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	禁麦其他人失败
70040	锁麦其他人失败
70041	恢复 PK 失败
70042	加入时获取 rtc role 失败
70043	响应 PK 邀请失败
70044	重新发布资源失败

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