

实时音视频 通话 **CallLib/Kit** Android 5.X

2024-08-30

实时音视频开发指导

更新时间:2024-08-30

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

本页面简单介绍融云 RTC 服务能力和 SDK 产品。

客户端 SDK

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

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

SDK 适用场景

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

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

如何选择 SDK

不同 SDK 适用的业务场景差异较大，请您谨慎选择并决策。

- **CallPlus 与 CallLib/Kit** 用于实现通话（呼叫）功能的客户端库。封装了拨打、振铃、接听、挂断等一整套呼叫流程，支持一对一及群组内多人呼叫的通话能力。CallPlus、CallLib/Kit 均依赖 RTCLib，两者区别如下：
 - **【推荐】** CallPlus 是融云新一代针对音视频呼叫场景的 SDK，后续新的产品特性和持续迭代均以 CallPlus 为重点。
 - CallLib/Kit 是老版本的音视频通话 SDK，CallLib 不含任何 UI 界面组件，CallKit 提供了呼叫相关的通用 UI 组件库。
 - CallPlus 与 CallLib/Kit 使用完全不同的后端服务架构实现音视频通话（呼叫）功能，因此与 CallLib/Kit 并不互通。暂不支持从 CallLib/Kit 平滑迁移至 CallPlus。
- **RTCLib** 是融云音视频核心能力库。应用开发者可将 RTCLib 用于支持直播、会议业务场景。

具体选择建议如下：

- 不需要通话（呼叫）功能，可使用 RTCLib，即您仅需要融云为您的 App 提供实时音视频（RTC）核心能力。

- 需要开发支持通话（呼叫）的音视频应用，但不希望自行实现呼叫 UI，可使用 CallKit。直接利用融云提供的呼叫 UI，节省开发时间。
- 需要开发支持通话（呼叫）的音视频应用，不希望 SDK 带任何 UI 组件，可使用 CallPlus、CallLib，推荐您使用 CallPlus。
- 通过融云提供的独立功能插件扩展客户端 SDK 的功能。

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

高级和扩展功能

RTC 服务支持的高级与扩展功能，包括但不限于以下项目：

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

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

平台兼容性

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

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

版本支持

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

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

注 1：关于 Linux 平台的支持，请咨询融云的商务。

SDK 体积对比

Android 端

以下数据基于 RTC 5.X 版本。

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

iOS 端

以下数据基于 RongCloudRTC 5.X 版本。

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

实时音视频服务端

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

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

[前往融云服务端开发文档·集成必读](#) »

控制台

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

音视频服务必须要从控制台开通后方可使用。参见[开通音视频服务](#)。

实时音视频数据

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

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

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

运行示例项目 (Demo)

更新时间:2024-08-30

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

QuickDemo 按场景和功能分为多个模块，提供 callapp 和 rtcapp 两个应用，对主要功能进行演示。QuickDemo 开放源代码，您可以对感兴趣的部分进行代码改造，以便进一步了解细节。

环境要求

- 使用 Android Studio 3.0 或以上版本。如果您尚未安装，请在 [官网下载](#) 并安装。
- Android SDK 4.4 或以上版本，即 `minSdkVersion >= 19`。
- Android Build Tools 21 或以上版本。为获得更好的编译体验，建议使用最新版。
- JDK 1.7 或以上版本。

融云开发者账户

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

提示

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

- 如果仅为体验 QuickDemo 创建应用，建议选择国内数据中心。如果选择海外数据中心，则需要额外在 QuickDemo 中修改 SDK 连接的服务地址。配置方法可参见 [数据中心](#)。

开通音视频服务

开发环境下的每个应用均可享有 10000 分钟免费体验时长。如果在开发环境下开通音视频服务，可直接按照以下步骤，开通音视频服务即可开始免费体验和测试。免费体验时长用完即止。

如果在生产环境下开通音视频服务，则需要先预存费用，才可开通。详情请参考 [开通音视频服务](#)。

设置设备

示例应用必须部署到搭载 Android 4.4 或更高版本的 Android 设备或 Android 模拟器，鸿蒙系统也可以使用。

- 如要使用 Android 设备，请按照在 [硬件设备上运行应用](#) 中的说明进行操作。
- 如要使用 Android 模拟器，您可以使用 [Android Studio 附带的 Android 虚拟设备 \(AVD\) 管理器](#) 创建虚拟设备并安装模拟器。

运行 QuickDemo

在运行 QuickDemo 前请确保已完成上述步骤。以下是检查清单：

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

1. 克隆下载示例代码。

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

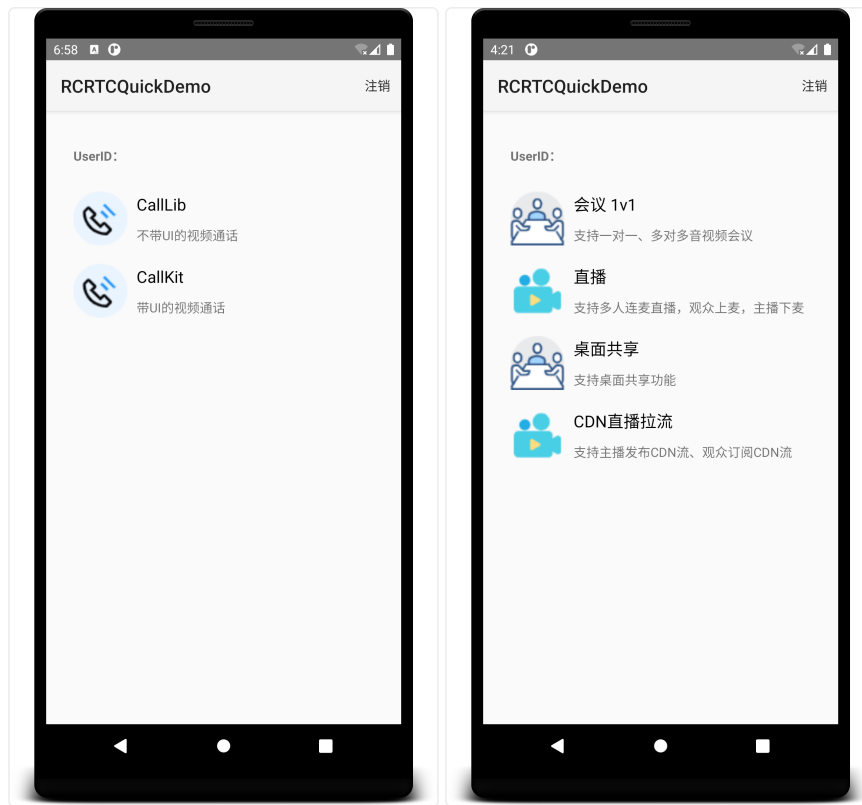
2. 在 Android Studio 中，选择 **Open an Existing Project**。等待导入完成。确保 Gradle 成功加载完 Maven 依赖库，此步视网络情况大约需要 2~5 分钟。
3. 打开 DemoApplication.java 文件，填入从控制台获取的 App Key 与 App Secret：

路径：**Project** 视图 <demo-app-name>/src/main/java/cn/rongcloud/demo/

```
/**
 * TODO: 请替换成您自己申请的 AppKey
 */
public static final String APP_KEY = "";

/**
 * TODO: 请替换成您自己 AppKey 对应的 Secret
 * 这里仅用于模拟从 App Server 获取 UserID 对应的 Token，开发者在上线应用时客户端代码不要存储该 Secret，
 * 否则有被用户反编译获取的风险，拥有 Secret 可以向融云 Server 请求高级权限操作，对应用安全造成恶劣影响。
 */
public static final String APP_SECRET = "";
```

4. 在 Android Studio 顶部选择 callapp 或 rtcapp，点击运行。
5. 运行成功后输入用户 ID，即可进入体验。建议在真实设备上运行。



⚠ 警告

本教程中直接在客户端代码中写入 APP_SECRET 的行为仅为演示目的。

APP_SECRET 可用于获取用户身份令牌 (Token)，以及实现人员禁言、房间踢人等高级能力。
存储在客户端代码里很容易被反编译导致泄露。

APP_SECRET 一旦泄露，攻击者就可以盗取 SDK 服务流量，或进行高权限破坏性操作。

正确的方式是将 APP_SECRET 存储在您的应用服务端，并提供面向应用客户端的接口。

更多详情请参见 [音视频服务端开发文档](#)。

导入 CallKit SDK

更新时间:2024-08-30

融云支持使用 Maven 远程仓库、本地库模块 (Module) 和源代码三种方式，将 CallKit SDK 导入到您的应用工程中。

环境要求

- (SDK \geq 5.6.3) 使用 Android 5.0 (API 21) 或更高版本
- (SDK < 5.6.3) 使用 Android 4.4 (API 19) 或更高版本

检查版本

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

Maven

使用 Gradle 添加对 CallKit、CallLib、IMKit、IMLib 四个模块的依赖关系。请注意使用 [融云的 Maven 仓库](#)。

1. 打开根目录下的 build.gradle (**Project** 视图下)，声明融云的 Maven 代码库。

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

2. 在应用的 build.gradle 中，添加如下远程依赖项。注意，融云 RTC 业务依赖 IM 通道，所以必须同时集成 IMLib。

```
dependencies {
    // x.y.z, 请填写具体的 SDK 版本号, 新集成用户建议使用最新版。
    implementation 'cn.rongcloud.sdk:call_kit:x.y.z'
    implementation 'cn.rongcloud.sdk:call_lib:x.y.z'
    implementation 'cn.rongcloud.sdk:im_kit:x.y.z'
    implementation 'cn.rongcloud.sdk:im_lib:x.y.z'

    implementation 'cn.rongcloud.sdk:face_beautifier:x.y.z' // 美颜扩展库 (可选)
}
```

提示

- 各个 SDK 的最新版本号可能不相同，还可能是 x.y.z.h，可前往 [融云官网 SDK 下载页面](#) 或 [融云的 Maven 代码库](#) 查询。
- 从 5.2.0 版本开始，CallKit/CallLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

Android 本地库模块 (Module)

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

1. 在 Android Studio 中打开工程后，依次点击 **File > New > Import Module**，找到下载的 Module 组件并导入。
2. 在应用的 build.gradle 中，添加本地库模块依赖项。

```
dependencies {
    implementation project(':CallKit')
    implementation project(':CallLib')
    implementation project(':IMKit')
    implementation project(':IMLib')
}
```

源码手动导入

融云提供 CallKit 源码，是为方便开发者根据 App 风格对呼叫 UI 做个性化的修改，比如色调搭配，按钮位置等，都可以自由定制。

CallKit Github 源代码地址：[GitHub](#) · [Gitee](#)

1. 先按照 Maven 导入或本地手动导入的方式，集成 CallLib、IMKit、IMLib 三个 CallKit 依赖库，并确保都是当时官网的最新版本，如下：

```
dependencies {
    implementation 'cn.rongcloud.sdk:call_lib:x.y.z'
    implementation 'cn.rongcloud.sdk:im_kit:x.y.z'
    implementation 'cn.rongcloud.sdk:im_lib:x.y.z'
}
```

提示

- 融云不提供老版本 CallKit 开源代码下载。用户配合 CallKit 所使用的 CallLib、IMKit、IMLib 版本需要是官网当前最新版本。
- 从 5.2.0 版本开始，CallKit/CallLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持

一致)。

2. 进入工程目录，克隆 CallKit 源码：

```
cd <ProjectFolder>
git clone https://github.com/rongcloud/callkit-android.git
```

3. 在 settings.gradle 文件中，添加引用：

```
include ':callkit-android'
```

4. 在应用的 build.gradle 中，添加依赖：

```
dependencies {
    ...
    implementation project(':callkit-android')
}
```

初始化

更新时间:2024-08-30

在使用 SDK 其它功能前，必须先进行初始化。本文将详细说明 CallKit 初始化的方法。

注意事项

- 必须在应用生命周期内调用初始化方法，只需要调用一次。
- 初始化后，会启动应用主进程、与应用包名相关的 IPC 进程、以及融云默认推送进程。详情请参考[关于多进程的说明](#)。

准备 App Key

您必须拥有正确的 App Key，才能进行初始化。

您可以[控制台](#)，查看您已创建的各个应用的 App Key。

如果您拥有多个应用，请注意选择应用名称（下图中标号 1）。另外，融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。在获取应用的 App Key 时，请注意区分环境（生产 / 开发，下图中标号 2）。

提示

- 如果您并非应用创建者，我们建议在获取 App Key 时确认页面上显示的数据中心是否符合预期。
- 如果您尚未向融云申请应用上线，仅可使用开发环境。



初始化之前

部分配置必须在初始化之前完成，否则 SDK 功能无法正常工作。

- 开通音视频服务：音视频服务需要手动开通。请根据应用的具体业务类型，开通对应的音视频服务。详细说明请参见[开通音视频服务](#)。

- **海外数据中心**：因为音视频业务依赖即时通讯业务 IMLib 提供信令通道，如果您的应用使用海外数据中心，必须在初始化之前修改 IMLib SDK 默认连接的服务地址为海外数据中心地址。否则 SDK 默认连接中国国内数据中心服务地址。详细说明请参见[配置海外数据中心服务地址](#)。

初始化接口

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

```
public class App extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        RongIM.init(this, "从控制台申请的 AppKey");
    }
}
```

最简单的情况，您可以使用 RongIM.init() 方法，在应用启动时进行初始化，并直接传入 App Key。

提示

- 请注意 IMKit 核心类为 RongIM。
- 必须在应用生命周期内调用初始化方法，只需要调用一次。

关于多进程的说明

融云 SDK 采用了多进程机制，初始化之后，应用会启动以下进程：

1. 应用的主进程
2. `<应用包名>:ipc`。此进程是 IM 通信的核心进程，和主进程任务相互隔离。
3. `io.rong.push`：融云默认推送进程。该进程是否启动由推送通道的启用策略决定。详细说明可参考[启用推送](#)。

实现音视频通话

更新时间:2024-08-30

CallKit 是在 CallLib 基础上，增加了一套默认呼叫界面的音视频呼叫功能 SDK，包含了单人、多人音视频呼叫的各种场景和功能，您可以快速的集成它来实现音视频呼叫场景。我们还提供了这个模块的开源代码（[GitHub](#) [Gitee](#)），您可以根据业务需要进行界面修改。

提示

房间人数上限

1. 考虑移动设备的带宽（主要是在多路视频情况下）和 UI 交互效果，建议单次通话或房间内，视频不超过 16 人，纯音频不超过 32 人。超过此上限可能影响通话效果。
2. CallKit 代码中已设置人数上限。默认发起视频呼叫时，最多可选 7 人。发起音频呼叫时，最多可选 20 人。如需调整，建议勿超过建议上限。
3. CallKit 为开源 SDK，可在源码中修改上限（修改位置：`CallSelectMemberActivity.java` 中的 `NORMAL_VIDEO_NUMBER` 与 `NORMAL_AUDIO_NUMBER`）。

步骤 1：服务开通

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

具体步骤请参阅 [开通音视频服务](#)。

提示

服务开通、关闭等设置完成后 30 分钟后生效。如需通过 SDK 判断您的 App 是否已成功开通服务，可使用 CallLib 的 `isVoIPEnabled` 方法。

步骤 2：SDK 导入

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

具体步骤请参阅 [导入 CallLib SDK](#)。

步骤 3：代码混淆

若开发者发布的 App 启用代码混淆，请务必在 `app/proguard-rules.pro` 文件添加如下配置：

```

-keepattributes Exceptions,InnerClasses

-keepattributes Signature
#RongRTCLib
-keep public class cn.rongcloud.** {*;};

#RongIMLib
-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.**

-ignorewarnings

```

步骤 4：权限配置

1. 在 AndroidManifest.xml 中声明 SDK 需要的所有权限。

```

<!-- 音视频需要网络权限 和 监听网络状态权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- 摄像头采集需要 -->
<uses-permission android:name="android.permission.CAMERA" />
<!-- 音频采集需要 -->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />

```

2. 如果您的应用需要支持 Android 6.0（API 级别 23）或更高版本的设备，您还需要在 App 用户使用对应功能时（例如发起呼叫、接听）请求摄像头（CAMERA）、麦克风（RECORD_AUDIO）权限。详见 Android 开发者官方文档[运行时权限](#)与[请求权限的工作流](#)。

步骤 5：初始化

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

- 示例代码：

```

public class App extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        RongIM.init(this, "从控制台申请的 AppKey");
    }
}

```

融云即时通信 SDK 采用了后台进程方式来确保稳定性。运行后会发现以下三个进程：

1. App 进程，进程名为 App 的包名。
2. 融云 IM 进程，进程名为 ipc。
3. 融云推送进程，如集成了厂商推送，则不会存在此进程，进程名为 io.rong.push。

步骤 6：连接 IM 服务

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

```
RongIM.connect("用户 Token", new RongIMClient.ConnectCallback() {
    @Override
    public void onSuccess(String userId) {
        // 连接成功
    }

    @Override
    public void onError(RongIMClient.ConnectionErrorCode code) {
        // 连接失败
    }

    @Override
    public void onDatabaseOpened(RongIMClient.DatabaseOpenStatus databaseOpenStatus) {
        // 数据库打开失败
    }
});
```

提示

- 如调用此接口时，遇到网络不好导致连接失败，SDK 会自动启动重连机制进行最多 10 次重连，重连时间间隔分别为 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 秒。在这之后如果仍没有连接成功，还会在检测到设备网络状态变化，比如网络恢复或切换网络时再次尝试重连。
- 如 App 在被杀死后，接收到了推送通知，点击通知拉起应用时，需要再次调用 `connect` 方法进行连接。

步骤 7：呼叫方

通常 App 内呼叫和被叫方逻辑会同时存在，所以需要分别集成。

发起单人呼叫

```
RongCallKit.startSingleCall(MainActivity.this, "对方 UserId",
RongCallKit.CallMediaType.CALL_MEDIA_TYPE_VIDEO);
```


参数	类型	必填	说明
context	Context	是	上下文
targetId	String	是	对方的 UserId
mediaType	CallMediaType	是	会话媒体类型，包括纯音频呼叫和音视频呼叫。

发起多人呼叫

```
String targetId = "GroupId";
RongCallKit.CallMediaType mediaType = RongCallKit.CallMediaType.CALL_MEDIA_TYPE_VIDEO;
ArrayList<String> userIds = new ArrayList<>();
userIds.add("UserId1");
userIds.add("UserId2");
RongCallKit.startMultiCall(MainActivity.this, Conversation.ConversationType.GROUP, targetId, mediaType,
userIds);
```

参数	类型	必填	说明
context	Context	是	上下文
conversationType	Conversation.ConversationType	是	会话类型
targetId	String	是	被叫用户所在共同群组的 GroupId
mediaType	CallMediaType	是	会话媒体类型，包括纯音频呼叫和音视频呼叫。
userIds	ArrayList<String>	是	被叫用户 UserId 列表

步骤 8：接听方

接听呼叫

- App 在前台时，当收到邀请时会自动弹出呼叫界面。
- App 在后台，且 Android 系统 <= 10 的手机，仍然会自动弹出呼叫界面。
- App 在后台，且 Android 系统 > 10 的手机，因受到新规则的限制，只能在手机上方弹出横幅通知栏，提示用户接听或挂断。

呼叫相关回调

IRongCallListener 是呼叫状态的监听类，CallKit 的 RongCallProxy.java 已经实现了该监听，并且会回调到 BaseCallActivity 中的各个方法，您可以继承 BaseCallActivity，根据需要复写其中的对应方法，即可获取对应的呼叫回调。

```
public class MyCallActivity extends BaseCallActivity {
/**
 * 电话已拨出。
 * 主叫端拨出电话后，通过回调 onCallOutgoing 通知当前 call 的详细信息。
 *
 * @param callSession 通话实体。
 * @param localVideo 本地 camera 信息。
 */
@Override
public void onCallOutgoing(RongCallSession callSession, SurfaceView localVideo) {
super.onCallOutgoing(callSession, localVideo);
}
}
```

```

/**
 * 已建立通话。
 * 通话接通时，通过回调 onCallConnected 通知当前 call 的详细信息。
 *
 * @param callSession 通话实体。
 * @param localVideo 本地 camera 信息。
 */
@Override
public void onCallConnected(RongCallSession callSession, SurfaceView localVideo) {
    super.onCallConnected(callSession, localVideo);
}

/**
 * 通话结束。
 * 通话中，对方挂断，己方挂断，或者通话过程网络异常造成的通话中断，都会回调 onCallDisconnected。
 *
 * @param callSession 通话实体。
 * @param reason 通话中断原因。
 */
@Override
public void onCallDisconnected(RongCallSession callSession, RongCallCommon.CallDisconnectedReason reason) {
    super.onCallDisconnected(callSession, reason);
}

/**
 * 被叫端正在振铃。
 * 主叫端拨出电话，被叫端收到请求，发出振铃响应时，回调 onRemoteUserRinging。
 *
 * @param userId 振铃端用户 id。
 */
@Override
public void onRemoteUserRinging(String userId) {
    super.onRemoteUserRinging(userId);
}

/**
 * 被叫端加入通话。
 * 主叫端拨出电话，被叫端收到请求后，加入通话，回调 onRemoteUserJoined。
 *
 * @param userId 加入用户的 id。<br />
 * @param mediaType 加入用户的媒体类型，audio or video。<br />
 * @param userType 加入用户的类型，1:正常用户,2:观察者。<br />
 * @param remoteVideo 加入用户者的 camera 信息。如果 userType为2，remoteVideo对象为空；<br />
 * 如果对端调用{@link RongCallClient#startCall(int, boolean, Conversation.ConversationType, String, List, List, RongCallCommon.CallMediaType, String, StartCameraCallback)} 或
 * {@link RongCallClient#acceptCall(String, int, boolean, StartCameraCallback)}开始的音视频通话，则可以使用如下设置改变对端视频流的镜像显示:<br />
 * <pre class="prettyprint">
 * public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType, SurfaceView remoteVideo) {
 *     if (null != remoteVideo) {
 *         ((RongRTCVideoView) remoteVideo).setMirror( boolean);//观看对方视频流是否镜像处理
 *     }
 * }
 * </pre>
 */
@Override
public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType, SurfaceView remoteVideo) {
    super.onRemoteUserJoined(userId, mediaType, userType, remoteVideo);
}

/**
 * 通话中的某一个参与者，邀请好友加入通话，发出邀请请求后，回调 onRemoteUserInvited。
 * @param userId 被邀请者的ID ,可以通过

```

```

RongCallClient.getInstance().getCallSession().getObserverUserList().contains(userId) ，查看加入的用户是否在
观察者列表中
* @param mediaType
*/
@Override
public void onRemoteUserInvited(String userId, RongCallCommon.CallMediaType mediaType) {
    super.onRemoteUserInvited(userId, mediaType);
}

/**
 * 通话中的远端参与者离开。
 * 回调 onRemoteUserLeft 通知状态更新。
 *
 * @param userId 远端参与者的 id。
 * @param reason 远端参与者离开原因。
 */
@Override
public void onRemoteUserLeft(String userId, RongCallCommon.CallDisconnectedReason reason) {
    super.onRemoteUserLeft(userId, reason);
}

/**
 * 当通话中的某一个参与者切换通话类型，例如由 audio 切换至 video，回调 onMediaTypeChanged。
 *
 * @param userId 切换者的 userId。
 * @param mediaType 切换者，切换后的媒体类型。
 * @param video 切换者，切换后的 camera 信息，如果由 video 切换至 audio，则为 null。
 */
@Override
public void onMediaTypeChanged(String userId, RongCallCommon.CallMediaType mediaType, SurfaceView video)
{
    super.onMediaTypeChanged(userId, mediaType, video);
}

/**
 * 通话过程中，发生异常。
 *
 * @param errorCode 异常原因。
 */
@Override
public void onError(RongCallCommon.CallErrorCode errorCode) {
    super.onError(errorCode);
}

/**
 * 远端参与者 camera 状态发生变化时，回调 onRemoteCameraDisabled 通知状态变化。
 *
 * @param userId 远端参与者 id。
 * @param disabled 远端参与者 camera 是否可用。
 */
@Override
public void onRemoteCameraDisabled(String userId, boolean disabled) {
    super.onRemoteCameraDisabled(userId, disabled);
}

/**
 * 远端参与者 麦克风 状态发生变化时，回调 onRemoteMicrophoneDisabled 通知状态变化。
 *
 * @param userId 远端参与者 id。
 * @param disabled 远端参与者 Microphone 是否可用。
 */
@Override
public void onRemoteMicrophoneDisabled(String userId, boolean disabled) {
    super.onRemoteMicrophoneDisabled(userId, disabled);
}

```

```

/**
 * 接收丢包率信息回调
 *
 * @param userId 远端用户的ID
 * @param lossRate 丢包率：0-100
 */
@Override
public void onNetworkReceiveLost(String userId, int lossRate) {
    super.onNetworkReceiveLost(userId, lossRate);
}

/**
 * 发送丢包率信息回调
 *
 * @param lossRate 丢包率，0-100
 * @param delay 发送端的网络延迟
 */
@Override
public void onNetworkSendLost(int lossRate, int delay) {
    super.onNetworkSendLost(lossRate, delay);
}

/**
 * 收到某个用户的第一帧视频数据
 *
 * @param userId
 * @param height
 * @param width
 */
@Override
public void onFirstRemoteVideoFrame(String userId, int height, int width) {
    super.onFirstRemoteVideoFrame(userId, height, width);
}

/**
 * 本端音量大小回调
 *
 * @param audioLevel
 */
@Override
public void onAudioLevelSend(String audioLevel) {
    super.onAudioLevelSend(audioLevel);
}

/**
 * 对端音量大小回调
 *
 * @param audioLevel
 */
@Override
public void onAudioLevelReceive(HashMap<String, String> audioLevel) {
    super.onAudioLevelReceive(audioLevel);
}

/**
 * 远端用户发布了自定义视频流
 *
 * @param userId 发布了自定义视频流的用户
 * @param streamId 自定义视频流Id
 * @param tag 流标签
 * @param surfaceView
 */
@Override
public void onRemoteUserPublishVideoStream(String userId, String streamId, String tag, SurfaceView

```

```

surfaceView) {
super.onRemoteUserPublishVideoStream(userId, streamId, tag, surfaceView);
}

/**
 * 远端用户取消发布自定义视频流
 *
 * @param userId 取消发布自定义视频流的用户
 * @param streamId 自定义视频流Id
 * @param tag 流标签
 */
@Override
public void onRemoteUserUnpublishVideoStream(String userId, String streamId, String tag) {
super.onRemoteUserUnpublishVideoStream(userId, streamId, tag);
}
}

```

如果上述方法不适合，您还可以通过修改 RongCallProxy.java 的代码，实现自己应用的监听。示例如下：

```

public class RongCallProxy implements IRongCallListener {

private IRongCallListener mCallListener; // 增加一个监听。

/*设置自己应用的监听*/
public void setAppCallListener(IRongCallListener listener) {
this.mAppCallListener = listener;
}

/*修改对应的通话状态回调的方法，使其回调到您设置的应用自身的监听*/
@Override
public void onCallOutgoing(RongCallSession callSession, SurfaceView localVideo) {
if (mCallListener != null) {
mCallListener.onCallOutgoing(callSession, localVideo);
}
/*增加的代码，回调应用设置的监听*/
if(mAppCallListener != null) {
mAppCallListener.onCallOutgoing(callSession, localVideo);
}
}
... // 根据您的需要，同样的方式修改其它通话状态回调函数。
}

```

修改完上述方法后，在您的应用里调用 setAppCallListener() 设置您自己的监听。

步骤 9：获取来电通知

因 Android 系统多版本，多硬件厂商，面临碎片化等问题，在处理音视频来电通知时，SDK 根据 Android 版本、CallKit 版本、与 App 处于前台/后台的状态有不同的处理方式。

应用程序在前台时，CallKit 可在远端发起呼叫时启动来电界面。

应用程序处于后台时，CallKit 的处理方式如下：

- 如果当前设备为 Android 10 之前版本的手机，CallKit 可启动来电界面。如果无法弹出 CallKit 的呼叫界面，请检查您的设备是否允许 App 使用「后台弹出界面」权限（某些 Android 设备上要求用户去系统设置中手动为 App 打开“后台弹出界面”的权限）。

- 如果当前设备为 Android 10 及之后版本的手机，请注意从 Android 10 开始，系统禁止应用在后台启动 Activity（参见 [Android 官方说明](#)）。如果您集成的 CallKit $\geq 5.1.9$ ，SDK 会弹窗提示，并长响铃。如果您集成的 CallKit $< 5.1.9$ ，SDK 会弹出通知栏通知用户，提示音是通知音。App 用户点击通知后可打开通话页面。

应用程序长时间在后台可能被系统回收，或者 App 用户下线时，则必须集成远程推送才能收到来电推送通知。详细请参考 [Android 推送集成](#)。集成离线推送后，即使 App 已经被系统回收，也可以收到呼叫的推送通知。

提示

集成 FCM 推送时您可能需要自行实现音视频信令消息（呼叫邀请、挂断等）的通知弹出逻辑。

如遇到关于来电通知的问题，可参考以下知识库：

- [Android 音视频通话通知说明](#)
- [无法弹出被呼叫页面](#)
- [如何自定义 CallKit 中的来电、呼叫铃声](#)

接入扩展插件

- CallKit 可以接入官方美颜插件或相芯美颜插件。注意，相芯美颜插件要求 CallKit 版本 $\geq 5.4.0$ 。详见 CallLib 文档[美颜插件](#)。

导入 CallLib SDK

更新时间:2024-08-30

融云支持使用 Maven 远程仓库和本地库模块 (Module) 两种方式，将 CallLib SDK 导入到您的应用工程中。

环境要求

- (SDK \geq 5.6.3) 使用 Android 5.0 (API 21) 或更高版本
- (SDK < 5.6.3) 使用 Android 4.4 (API 19) 或更高版本

检查版本

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

Maven

使用 Gradle，声明 [融云的 Maven 代码库](#)，并添加对 CallLib、IMLib 等模块的依赖关系。

1. 打开根目录下的 build.gradle (**Project** 视图下)，声明融云的 Maven 代码库。

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

2. 在应用的 build.gradle 中，添加如下远程依赖项。注意，融云 RTC 业务依赖 IM 通道，所以必须同时集成 IMLib。

```
dependencies {
    // x.y.z, 请填写具体的 SDK 版本号, 新集成用户建议使用最新版。
    implementation 'cn.rongcloud.sdk:call_lib:x.y.z'
    implementation 'cn.rongcloud.sdk:im_lib:x.y.z'

    implementation 'cn.rongcloud.sdk:face_beautifier:x.y.z' // 美颜扩展库 (可选)
}
```

- 各个 SDK 的最新版本号可能不相同，还可能是 x.y.z.h，可前往 [融云官网 SDK 下载页面](#) 或 [融云的 Maven 代码库](#) 查询。
- 从 5.2.0 版本开始，CallLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

Android 本地库模块 (Module)

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

1. 在 Android Studio 中打开工程后，依次点击 **File > New > Import Module**，找到下载的 Module 组件并导入。
2. 在应用的 build.gradle 中，添加本地库模块依赖项。

```
dependencies {  
    implementation project(':CallLib')  
    implementation project(':IMLib')  
}
```

提示

从 5.2.0 版本开始，CallLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

初始化

更新时间:2024-08-30

在使用 SDK 其它功能前，必须先进行初始化。本文将详细说明 CallLib 初始化的方法。

注意事项

- 必须在应用生命周期内调用初始化方法，只需要调用一次。
- 初始化后，会启动应用主进程、与应用包名相关的 IPC 进程、以及融云默认推送进程。详情请参考[关于多进程的说明](#)。

准备 App Key

您必须拥有正确的 App Key，才能进行初始化。

您可以[控制台](#)，查看您已创建的各个应用的 App Key。

如果您拥有多个应用，请注意选择应用名称（下图中标号 1）。另外，融云的每个应用都提供用于隔离生产和开发环境的两套独立 App Key / Secret。在获取应用的 App Key 时，请注意区分环境（生产 / 开发，下图中标号 2）。

提示

- 如果您并非应用创建者，我们建议在获取 App Key 时确认页面上显示的数据中心是否符合预期。
- 如果您尚未向融云申请应用上线，仅可使用开发环境。



初始化之前

部分配置必须在初始化之前完成，否则 SDK 功能无法正常工作。

- 开通音视频服务：音视频服务需要手动开通。请根据应用的具体业务类型，开通对应的音视频服务。详细说明请参见[开通音视频服务](#)。

- **海外数据中心**：因为音视频业务依赖即时通讯业务 IMLib 提供信令通道，如果您的应用使用海外数据中心，必须在初始化之前修改 IMLib SDK 默认连接的服务地址为海外数据中心地址。否则 SDK 默认连接中国国内数据中心服务地址。详细说明请参见[配置海外数据中心服务地址](#)。

初始化接口

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

```
public class App extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        RongIMClient.init(this, "从控制台申请的 AppKey");
    }
}
```

最简单的情况，您可以使用 RongIMClient.init() 方法，在应用启动时进行初始化，并直接传入 App Key。

提示

- 请注意 IMLib 核心类为 RongIMClient。
- 必须在应用生命周期内调用初始化方法，只需要调用一次。

关于多进程的说明

融云 SDK 采用了多进程机制，初始化之后，应用会启动以下进程：

1. 应用的主进程
2. `<应用包名>:ipc`。此进程是 IM 通信的核心进程，和主进程任务相互隔离。
3. `io.rong.push`：融云默认推送进程。该进程是否启动由推送通道的启用策略决定。详细说明可参考[启用推送](#)。

实现音视频通话

更新时间:2024-08-30

CallLib 是在 RTCLib 基础上，额外封装了一套音视频呼叫功能 SDK，包含了单人、多人音视频呼叫的各种场景和功能，通过集成它，您可以自由的实现音视频呼叫场景的各种玩法。

提示

房间人数上限

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

步骤 1：服务开通

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

具体步骤请参阅 [开通音视频服务](#)。

提示

服务开通、关闭等设置完成后 30 分钟后生效。如需通过 SDK 判断您的 App 是否已成功开通服务，可使用 CallLib 的 [isVoIPEnabled](#) 方法。

步骤 2：SDK 导入

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

具体步骤请参阅 [导入 CallLib SDK](#)。

步骤 3：代码混淆

若开发者发布的 App 启用代码混淆，请务必在 `app/proguard-rules.pro` 文件添加如下配置：

```

-keepattributes Exceptions,InnerClasses

-keepattributes Signature
#RongRTCLib
-keep public class cn.rongcloud.** {*;}

#RongIMLib
-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.**

-ignorewarnings

```

步骤 4：权限配置

1. 在 AndroidManifest.xml 中声明 SDK 需要的所有权限。

```

<!-- 音视频需要网络权限 和 监听网络状态权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- 摄像头采集需要 -->
<uses-permission android:name="android.permission.CAMERA" />
<!-- 音频采集需要 -->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />

```

2. 如果您的应用需要支持 Android 6.0（API 级别 23）或更高版本的设备，您还需要在 App 用户使用对应功能时（例如发起呼叫、接听）请求摄像头（CAMERA）、麦克风（RECORD_AUDIO）权限。详见 Android 开发者官方文档[运行时权限](#)与[请求权限的工作流](#)。

步骤 5：初始化

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

```

public class App extends Application {
@OVERRIDE
public void onCreate() {
super.onCreate();
RongIMClient.init(this, "从控制台申请的 AppKey");
}
}

```

融云即时通信 SDK 采用了后台进程方式来确保稳定性。运行后会发现以下三个进程：

1. App 进程，进程名为 App 的包名。
2. 融云 IM 进程，进程名为 ipc。
3. 融云推送进程，如集成了厂商推送，则不会存在此进程，进程名为 io.rong.push。

步骤 6：通话事件处理

SDK 提供针对来电、通话状态、通话记录的事件处理机制。

监听通话呼入

1. 调用 RongCallClient 中的 setReceivedCallListener 来监听通话呼入。

```
RongCallClient.setReceivedCallListener(new IRongReceivedCallListener() {  
    /**  
     * 来电回调  
     * @param callSession 通话实体  
     */  
    @Override  
    public void onReceivedCall(RongCallSession callSession) {  
    }  
  
    /**  
     * targetSDKVersion 大于等于 23 时检查权限的回调。当 targetSDKVersion 小于 23 的时候不需要实现。  
     * 在这个回调里用户需要使用Android6.0新增的动态权限分配接口通知用户授权，  
     * 然后根据用户授权或者不授权分别回调  
     * RongCallClient.getInstance().onPermissionGranted()和  
     * RongCallClient.getInstance().onPermissionDenied()来通知CallLib。  
     *  
     * @param callSession 通话实体  
     */  
    @Override  
    public void onCheckPermission(RongCallSession callSession) {  
    }  
});
```

2. 如果应用需要支持 Android 6.0（API 级别 23）或更高版本的设备，请在 onCheckPermission 回调中通过被叫用户，请求授予摄像头（CAMERA）、麦克风（RECORD_AUDIO）权限，并将结果通知 CallLib。

- 授权后，通知 CallLib。SDK 会触发电监听上的 onReceivedCall 回调。

```
RongCallClient.getInstance().onPermissionGranted();
```

- 拒绝授权后，通知 CallLib。此时主叫端会触发通话状态监听的 onCallDisconnected 方法结束呼叫。因为 REMOTE_REJECT(12) 对方拒绝。

```
RongCallClient.getInstance().onPermissionDenied()
```

监听通话状态变化

调用 `RongCallClient` 中的 `setVoIPCallListener` 来监听通话状态的变化。

```
RongCallClient.getInstance().setVoIPCallListener(new IRongCallListener() {  
/**  
* 电话已拨出。  
* 主叫端拨出电话后，通过回调 onCallOutgoing 通知当前 call 的详细信息。  
*  
* @param callSession 通话实体。  
* @param localVideo 本地 camera 信息。  
*/  
@Override  
public void onCallOutgoing(RongCallSession callSession, SurfaceView localVideo) {  
}  
  
/**  
* 已建立通话。  
* 通话接通时，通过回调 onCallConnected 通知当前 call 的详细信息。  
*  
* @param callSession 通话实体。  
* @param localVideo 本地 camera 信息。  
*/  
@Override  
public void onCallConnected(RongCallSession callSession, SurfaceView localVideo) {  
}  
  
/**  
* 通话结束。  
* 通话中，对方挂断，己方挂断，或者通话过程网络异常造成的通话中断，都会回调 onCallDisconnected。  
*  
* @param callSession 通话实体。  
* @param reason 通话中断原因。  
*/  
@Override  
public void onCallDisconnected(RongCallSession callSession, RongCallCommon.CallDisconnectedReason  
reason) {  
}  
  
/**  
* 被叫端加入通话。  
* 主叫端拨出电话，被叫端收到请求后，加入通话，回调 onRemoteUserJoined。  
*  
* @param userId 加入用户的 id。<br />  
* @param mediaType 加入用户的媒体类型，audio or video。<br />  
* @param userType 加入用户的类型，1:正常用户,2:观察者。<br />  
* @param remoteVideo 加入用户者的 camera 信息。如果 userType为2，remoteVideo对象为空；<br />  
* 如果对端调用{@link RongCallClient#startCall(int, boolean, Conversation.ConversationType, String, List,  
List, RongCallCommon.CallMediaType, String, StartCameraCallback)} 或  
* {@link RongCallClient#acceptCall(String, int, boolean, StartCameraCallback)}开始的音视频通话，则可以使用如  
下设置改变对端视频流的镜像显示：<br />  
* <pre class="prettyprint">  
* public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType,  
SurfaceView remoteVideo) {  
* if (null != remoteVideo) {  
* ((RongRTCVideoView) remoteVideo).setMirror( boolean);//观看对方视频流是否镜像处理  
* }  
* }  
* </pre>  
*/  
@Override  
public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType,  
SurfaceView remoteVideo) {  
if (null != remoteVideo) {  
((RongRTCVideoView) remoteVideo).setMirror( true);  
}  
}  
}
```

```

*/
@Override
public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType,
SurfaceView remoteVideo) {
}

/**
 * 通话中的远端参与者离开。
 * 回调 onRemoteUserLeft 通知状态更新。
 *
 * @param userId 远端参与者的 id。
 * @param reason 远端参与者离开原因。
 */
@Override
public void onRemoteUserLeft(String userId, RongCallCommon.CallDisconnectedReason reason) {
}
});

```

监听漏接电话

调用 RongCallClient 中的 setMissedCallListener 来监听漏接的通话，一般用于记录通话记录。

```

RongCallClient.setMissedCallListener(new RongCallMissedListener() {
/**
 * 漏接通话回调
 * @param callSession 通话实体
 * @param reason 远端参与者离开原因。
 */
@Override
public void onRongCallMissed(RongCallSession callSession, RongCallCommon.CallDisconnectedReason reason)
{
}
});

```

通话计时

CallLib SDK 无法直接获取通话时长，您可以在 IRongCallListener 的通话建立成功、音频首帧回调、或视频首帧回调方法中，使用当前时间减去通话起始时间，获取通话时长。

回调方法：

```

/**
 * 已建立通话。 通话接通时，通过回调 onCallConnected 通知当前 call 的详细信息。
 *
 * @param callSession 通话实体。
 * @param localVideo 本地 camera 信息。
 */
void onCallConnected(RongCallSession callSession, SurfaceView localVideo);

/**
 * 收到某个用户的第一帧视频数据
 *
 * @param userId
 * @param height
 * @param width
 */
void onFirstRemoteVideoFrame(String userId, int height, int width);

/**
 * 收到某个用户的第一帧音频数据
 *
 * @param userId
 */
void onFirstRemoteAudioFrame(String userId);

```

定时计算通话时长：

```

@Override
public void onCallConnected(final RongCallSession callSession, SurfaceView localVideo) {
    // TODO

    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            RongCallClient instance = RongCallClient.getInstance();
            if (instance == null) {
                return;
            }
            RongCallSession callSession = instance.getCallSession();
            if (callSession == null) {
                return;
            }
            // 获取通话建立成功时的时间戳
            long ConnectedTime = callSession.getActiveTime();
            // 获取当前本地系统时间戳
            long currentTime = System.currentTimeMillis();
            // 计算通话时长
            long callDuration = currentTime - ConnectedTime;
            // 格式化输出
            String formatTime = String.format("%02d:%02d:%02d", callDuration / 3600000, (callDuration % 3600000) / 60000, callDuration % 60000);
            // 弹出提示
            Toast.makeText(this, "通话时长：" + formatTime, Toast.LENGTH_SHORT).show();
            handler.postDelayed(this, 1000); // 循环获取通话时长
        }
    }, 1000); // 延迟 1 秒后执行一次
};

```


如需在应用程序的服务端进行通话计时，建议使用融云提供的服务端回调音视频房间状态同步 [🔗](#)。通过实时回调事件 event 11（成员加入音视频房间）和 event 12（成员加入音视频房间）记录计费的开始和结束时间。

步骤 7：连接 IM 服务

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

```
RongIMClient.connect("从您服务器端获取的 Token", new RongIMClient.ConnectCallback() {
@Override
public void onSuccess(String userId) {
// 连接成功
}

@Override
public void onError(RongIMClient.ConnectionErrorCode code) {
// 连接失败
}

@Override
public void onDatabaseOpened(RongIMClient.DatabaseOpenStatus databaseOpenStatus) {
// 数据库打开失败
}
});
```

提示

- 如调用此接口时，遇到网络不好导致连接失败，SDK 会自动启动重连机制进行最多 10 次重连，重连时间间隔分别为 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 秒。在这之后如果仍没有连接成功，还会在检测到设备网络状态变化，比如网络恢复或切换网络时再次尝试重连。
- 如 App 在被杀死后，接收到了推送通知，点击通知拉起应用时，需要再次调用 connect 方法进行连接。

步骤 8：发起呼叫

连接 IM 服务成功后，可调用 RongCallClient 中的 startCall 来方法发起通话。

发起单人呼叫

```
// 被叫用户 Id
String targetId = "UserB";
List<String> userIds = new ArrayList<>();
userIds.add(targetId);
RongCallClient.getInstance().startCall(Conversation.ConversationType.PRIVATE, targetId, userIds, null,
RongCallCommon.CallMediaType.VIDEO, null);
```

发起多人呼叫

```
// 被叫用户所在共同群组 Id
String targetId = "GroupX";
// 群组内的被叫用户 id 集合
List<String> userIds = new ArrayList<>();
userIds.add("UserB");
userIds.add("UserC");
RongCallClient.getInstance().startCall(Conversation.ConversationType.GROUP, targetId, userIds, null,
RongCallCommon.CallMediaType.VIDEO, null);
```

步骤 9：呼叫接听

在收到 `onReceivedCall(RongCallSession rongCallSession)` 回调之后，调用如下方法接听通话。

```
RongCallClient.getInstance().acceptCall(RongCallClient.getInstance().getCallSession().getCallId());
```

步骤 10：离线推送通知

集成离线推送后，即使 App 已经被系统回收，也可以收到呼叫的推送通知。详细请参考 [Android 推送集成](#)。

① 提示

集成 FCM 推送时您可能需要自行实现音视频信令消息（呼叫邀请、挂断等）的通知弹出逻辑。

主叫方 发起呼叫

更新时间:2024-08-30

调用 `RongCallClient.startCall` 方法发起单人或多人音视频通话，该方法默认打开前置摄像头。多人通话场景所有通话者必须在一个群组内。

- 参数说明：

参数	类型	必填	说明
conversationType	Conversation.ConversationType	是	会话类型
targetId	String	是	目标会话 ID，单人通话为对方 UserId，群组通话为 GroupId
userIds	List<String>	是	邀请参与通话的用户 ID 列表，不能为 null，必须包含 observerUserIds 中除发起者ID的所有用户ID
observerUserIds	List<String>	是	观察者列表，无观察者可传 null，当音视频发起方是观察者时，observerUserIds 需要包含发起方用户 ID
mediaType	RongCallCommon.CallMediaType	是	发起的通话媒体类型
extra	String	是	附加信息，透传至对端，对端通过 <code>RongCallSession.getExtra()</code> 获取

- 示例代码：

```

Conversation.ConversationType conversationType = Conversation.ConversationType.PRIVATE;
String targetId = "001";
List<String> userIds = new ArrayList<>();
userIds.add("002");
List<String> observerUserIds = new ArrayList<>();
observerUserIds.add("003");
RongCallCommon.CallMediaType mediaType = RongCallCommon.CallMediaType.VIDEO;
String extra = "";
RongCallClient.getInstance().startCall(conversationType, targetId, userIds, observerUserIds,
mediaType, extra);
    
```

指定摄像头发起呼叫

发起单人或多人音视频通话需要打开指定摄像头，请调用以下方法。

- 参数说明：

参数	类型	必填	说明
cameraId	int	是	摄像头 ID
mirror	boolean	是	是否镜像视频流，详细说明请查看下方提示
conversationType	Conversation.ConversationType	是	会话类型
targetId	String	是	目标会话 ID，单人通话为对方 UserID，群组通话为 GroupId
userIds	List<String>	是	邀请参与通话的用户 ID 列表，不能为空，必须包含 observerUserIds 中除发起者 ID 的所有用户 ID
observerUserIds	List<String>	是	观察者列表，无观察者可传 null，当音视频发起方是观察者时，observerUserIds 需要包含发起方用户 ID
mediaType	RongCallCommon.CallMediaType	是	发起的通话媒体类型
extra	String	是	附加信息，透传至对端，对端通过 RongCallSession.getExtra() 获取

提示

mirror 说明

在某些特殊设备上，后置摄像头被安装在设备的前面。为打开后置摄像头时可以正常显示，请使用镜像处理。mirror仅对本地有效（发送出去的数据依然是未处理数据）。

如果本地视频流做了镜像处理，为了对端观看体验可以在 IRongCallListener#onRemoteUserJoined(String, RongCallCommon.CallMediaType, int, SurfaceView) 回调中添加如下设置：

```
public void onRemoteUserJoined (String userId, RongCallCommon.CallMediaType mediaType, int
userType, SurfaceView remoteVideo){
    if (null != remoteVideo) {
        ((RongRTCVideoView) remoteVideo).setMirror(boolean); //观看对方视频流是否镜像处理
    }
}
```

- 示例代码：

```

int cameraId = 1;
boolean mirror = false;
Conversation.ConversationType conversationType = Conversation.ConversationType.PRIVATE;
String targetId = "001";
List<String> userIds = new ArrayList<>();
userIds.add("002");
List<String> observerUserIds = new ArrayList<>();
observerUserIds.add("003");
RongCallCommon.CallMediaType mediaType = RongCallCommon.CallMediaType.VIDEO;
String extra = "";
RongCallClient.getInstance().startCall(cameraId, mirror, conversationType, targetId, userIds,
observerUserIds, mediaType, extra, new StartCameraCallback() {
@Override
public void onDone(boolean b) {
}

@Override
public void onError(int i) {
}
});

```

挂断通话

调用 `RongCallClient.hangUpCall` 方法挂断通话，拒绝和挂断为同一个方法调用，SDK 内部会自动告知对方挂断、拒绝原因。

- 参数说明：

返回值	返回类型	说明
callId	String	呼叫 ID，可以从 RongCallSession.getCallId() 中获取

- 示例代码：

```

// im未连接或者不在通话中，RongCallClient 和 RongCallSession 为空
if (RongCallClient.getInstance() != null && RongCallClient.getInstance().getCallSession() != null) {
RongCallClient.getInstance().hangUpCall(RongCallClient.getInstance().getCallSession().getCallId());
}

```

邀请通话

调用 `RongCallClient.addParticipants` 方法邀请用户加入当前通话（仅限群组），该方法必须在通话已经建立 (`IRongCallListener.onCallConnected`)之后调用有效。

- 参数说明：

参数	类型	必填	说明
callId	String	是	通话 ID
userIds	ArrayList<String>	是	邀请的用户 ID 列表，请一定包含邀请的观察者列表 中的人员
observerUserIds	ArrayList<String>	是	邀请的观察者列表，没有观察者可传 null

- 示例代码：

```
if (RongCallClient.getInstance() != null && RongCallClient.getInstance().getCallSession() != null) {  
    String callId = RongCallClient.getInstance().getCallSession().getCallId();  
    ArrayList<String> userIds = new ArrayList<>();  
    userIds.add("002");  
    userIds.add("003");  
    ArrayList<String> observerUserIds = new ArrayList<>();  
    observerUserIds.add("002");  
    observerUserIds.add("003");  
    RongCallClient.getInstance().addParticipants(callId, userIds, observerUserIds);  
}
```

被叫方 接听通话

更新时间:2024-08-30

默认接听

当收到来自 `onReceivedCall` 的远端通话请求时，可使用 `RongCallClient` 的 `acceptCall` 方法来接听。该方法默认打开前置摄像头。

- 示例代码：

```
public void onReceivedCall(RongCallSession session) {  
    RongCallClient.getInstance().acceptCall(session.getCallId());  
}
```

指定摄像头接听

来电监听中接收到来电请求后，调用如下方法接听通话，该方法可以打开指定 Id 摄像头。

- 参数说明：

参数	类型	必填	说明
<code>callId</code>	String	是	呼叫 ID，可以从 RongCallSession.getCallId() 中获取
<code>cameraId</code>	int	是	摄像头 ID
<code>mirror</code>	boolean	是	是否镜像视频流，详细说明请查看下方提示
<code>callback</code>	StartCameraCallback	是	打开摄像头是否成功回调

提示

某些特殊设备将后置摄像头安装在设备的前面时，当打开后置摄像头时为了正常显示，请使用镜像处理，

`mirror` 仅对本地有效(发送出去的数据依然是未处理数据)，

如果本地视频流做了镜像处理，为了对端观看体验可以

在 `IRongCallListener#onRemoteUserJoined(String,`

`RongCallCommon.CallMediaType, int, SurfaceView)` 回调中添加如下设置：

```
public void onRemoteUserJoined (String userId, RongCallCommon.CallMediaType mediaType,int userType,
SurfaceView remoteVideo){
if (null != remoteVideo) {
((RongRTCVideoView) remoteVideo).setMirror(boolean);//观看对方视频流是否镜像处理
}
}
```

- 返回参数说明：

返回值	返回类型	说明
callId	String	呼叫 ID ，可以从 RongCallSession.getCallId() 中获取

- 示例代码：

```
if (RongCallClient.getInstance() != null && RongCallClient.getInstance().getCallSession() != null) {
String callId = RongCallClient.getInstance().getCallSession().getCallId();
int cameraId = 0;
boolean mirror = false;
RongCallClient.getInstance().acceptCall(callId, cameraId, mirror, new StartCameraCallback() {
@Override
public void onDone(boolean isFront) {

}

@Override
public void onError(int errorCode) {

}
});
}
```

拒绝/挂断通话

调用 `RongCallClient.hangUpCall` 方法挂断通话，拒绝和挂断为同一个方法调用，SDK 内部会自动告知对方挂断、拒绝原因。

- 参数说明：

返回值	返回类型	说明
callId	String	呼叫 ID ，可以从 RongCallSession.getCallId() 中获取

- 示例代码：


```
// im未连接或者不在通话中，RongCallClient 和 RongCallSession 为空
if (RongCallClient.getInstance() != null && RongCallClient.getInstance().getCallSession() != null) {
RongCallClient.getInstance().hangUpCall(RongCallClient.getInstance().getCallSession().getCallId());
}
```

通话监听

更新时间:2024-08-30

融云 CallLib 库提供了 [IRongReceivedCallListener](#) 和 [IRongCallListener](#) 两个类, 用于处理呼叫相关的业务逻辑上报。

来电监听

需要设置 CallLib 的全局通话监听 [IRongReceivedCallListener](#) , 来监听通话呼入。

1. 调用 RongCallClient 中的 setReceivedCallListener 来监听通话呼入。

```
RongCallClient.setReceivedCallListener(new IRongReceivedCallListener() {
    /**
     * 来电回调
     * @param callSession 通话实体
     */
    @Override
    public void onReceivedCall(RongCallSession callSession) {
    }

    /**
     * targetSDKVersion 大于等于 23 时检查权限的回调。当 targetSDKVersion 小于 23 的时候不需要实现。
     * 在这个回调里用户需要使用Android6.0新增的动态权限分配接口通知用户授权，
     * 然后根据用户授权或者不授权分别回调
     * RongCallClient.getInstance().onPermissionGranted()和
     * RongCallClient.getInstance().onPermissionDenied()来通知CallLib。
     *
     * @param callSession 通话实体
     */
    @Override
    public void onCheckPermission(RongCallSession callSession) {
    }
});
```

2. 如果应用需要支持 Android 6.0 (API 级别 23) 或更高版本的设备, 请在 onCheckPermission 回调中通过被叫用户, 请求授予摄像头 (CAMERA)、麦克风 (RECORD_AUDIO) 权限, 并将结果通知 CallLib。

- 授权后, 通知 CallLib。SDK 会触发电听听的 onReceivedCall 回调。

```
RongCallClient.getInstance().onPermissionGranted();
```

- 拒绝授权后, 通知 CallLib。此时主叫端会触发通话状态监听的 onCallDisconnected 方法结束呼叫。因为 REMOTE_REJECT(12) 对方拒绝。

```
RongCallClient.getInstance().onPermissionDenied()
```

通话状态监听

设置通话状态的回调 [IRongCallListener](#) ，来监听通话状态的变化。

```
RongCallClient.getInstance().setVoIPCallListener(new IRongCallListener() {  
//  
});
```

具体支持监听的事件请参见下方介绍的回调方法。

通话建立、结束等状态相关的回调

- 电话已拨出，返回当前通话的详细信息。
- 已建立通话，返回当前通话的详细信息。
- 通话结束。对方挂断，己方挂断，或者通话过程网络异常造成的通话中断，都会通过同一个回调返回原因（[RongCallCommon.CallDisconnectedReason](#)）。
- 被叫端正在振铃，返回振铃用户的用户 ID。
- 被叫端加入通话，返回加入者的用户信息和摄像头信息。
- 通话中的某一个参与者，邀请好友加入通话。返回被邀请者的信息和媒体类型。
- 通话中的远端参与者离开，返回离开者的信息和离开原因（[RongCallCommon.CallDisconnectedReason](#)）。在多人通话与 1v1 通话中，对端挂断均会先回调 `onRemoteUserLeft`，再触发其他回调。
- 当通话中的某一个参与者切换通话类型，例如由视频切换至音频。返回切换操作者的信息、切换后的媒体类型等。
- 通话过程中，发生异常。返回错误码（[RongCallCommon.CallErrorCode](#)）。

```
/**  
 * 电话已拨出。  
 * 主叫端拨出电话后，通过回调 onCallOutgoing 通知当前 call 的详细信息。  
 *  
 * @param callSession 通话实体。  
 * @param localVideo 本地 camera 信息。  
 */  
@Override  
public void onCallOutgoing(RongCallSession callSession, SurfaceView localVideo) {}  
  
/**  
 * 已建立通话。  
 * 通话接通时，通过回调 onCallConnected 通知当前 call 的详细信息。  
 *  
 * @param callSession 通话实体。  
 * @param localVideo 本地 camera 信息。  
 */  
@Override  
public void onCallConnected(RongCallSession callSession, SurfaceView localVideo) {}  
  
/**  
 * 通话结束。
```

```

* 通话中，对方挂断，己方挂断，或者通话过程网络异常造成的通话中断，都会回调 onCallDisconnected。
*
* @param callSession 通话实体。
* @param reason 通话中断原因。
*/
@Override
public void onCallDisconnected(RongCallSession callSession, RongCallCommon.CallDisconnectedReason reason) {}

/**
* 被叫端正在振铃。
* 主叫端拨出电话，被叫端收到请求，发出振铃响应时，回调 onRemoteUserRinging。
*
* @param userId 振铃端用户 id。
*/
@Override
public void onRemoteUserRinging(String userId) {}

/**
* 被叫端加入通话。
* 主叫端拨出电话，被叫端收到请求后，加入通话，回调 onRemoteUserJoined。
*
* @param userId 加入用户的 id。<br />
* @param mediaType 加入用户的媒体类型，audio or video。<br />
* @param userType 加入用户的类型，1:正常用户,2:观察者。<br />
* @param remoteVideo 加入用户者的 camera 信息。如果 userType为2，remoteVideo对象为空；<br />
* 如果对端调用{@link RongCallClient#startCall(int, boolean, Conversation.ConversationType, String, List, List, RongCallCommon.CallMediaType, String, StartCameraCallback)} 或
* {@link RongCallClient#acceptCall(String, int, boolean, StartCameraCallback)}开始的音视频通话，则可以使用如下设置改变对端视频流的镜像显示：<br />
* <pre class="prettyprint">
* public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType,
* SurfaceView remoteVideo) {
* if (null != remoteVideo) {
* ((RongRTCVideoView) remoteVideo).setMirror( boolean);//观看对方视频流是否镜像处理
* }
* }
* </pre>
*/
@Override
public void onRemoteUserJoined(String userId, RongCallCommon.CallMediaType mediaType, int userType, SurfaceView remoteVideo) {}

/**
* 通话中的某一个参与者，邀请好友加入通话，发出邀请请求后，回调 onRemoteUserInvited。
* @param userId 被邀请者的ID ，可以通过
RongCallClient.getInstance().getCallSession().getObserverUserList().contains(userId) ，查看加入的用户是否在
观察者列表中
* @param mediaType
*/
@Override
public void onRemoteUserInvited(String userId, RongCallCommon.CallMediaType mediaType) {}

/**
* 通话中的远端参与者离开。
* 回调 onRemoteUserLeft 通知状态更新。
*
* @param userId 远端参与者的 id。
* @param reason 远端参与者离开原因。
*/
@Override
public void onRemoteUserLeft(String userId, RongCallCommon.CallDisconnectedReason reason) {}

/**
* 当通话中的某一个参与者切换通话类型，例如由 audio 切换至 video，回调 onMediaTypeChanged。

```

```

*
* @param userId 切换者的 userId。
* @param mediaType 切换者，切换后的媒体类型。
* @param video 切换者，切换后的 camera 信息，如果由 video 切换至 audio，则为 null。
*/
@Override
public void onMediaTypeChanged(String userId, RongCallCommon.CallMediaType mediaType, SurfaceView video)
{}

/**
* 通话过程中，发生异常。
*
* @param errorCode 异常原因。
*/
@Override
public void onError(RongCallCommon.CallErrorCode errorCode) {}

```

设备相关回调

- 远端参与者摄像头状态发生变化时，回调 onRemoteCameraDisabled 通知状态变化。
- 远端参与者麦克风状态发生变化时，回调 onRemoteMicrophoneDisabled 通知状态变化。

```

/**
* 远端参与者 camera 状态发生变化时，回调 onRemoteCameraDisabled 通知状态变化。
*
* @param userId 远端参与者 id。
* @param disabled 远端参与者 camera 是否可用。
*/
@Override
public void onRemoteCameraDisabled(String userId, boolean disabled) {}

/**
* 远端参与者 麦克风 状态发生变化时，回调 onRemoteMicrophoneDisabled 通知状态变化。
*
* @param userId 远端参与者 id。
* @param disabled 远端参与者 Microphone 是否可用。
*/
@Override
public void onRemoteMicrophoneDisabled(String userId, boolean disabled) {}

```

网络质量相关回调

- 接收丢包率信息回调，返回远端用户 ID 及丢包率。
- 发送丢包率信息回调，返回丢包率及发送端的网络延迟。
- 收到某个用户的第一帧视频数据，返回用户信息及宽高数据。

```

/**
 * 接收丢包率信息回调
 *
 * @param userId 远端用户的ID
 * @param lossRate 丢包率：0-100
 */
@Override
public void onNetworkReceiveLost(String userId, int lossRate) {}

/**
 * 发送丢包率信息回调
 *
 * @param lossRate 丢包率，0-100
 * @param delay 发送端的网络延迟
 */
@Override
public void onNetworkSendLost(int lossRate, int delay) {}

/**
 * 收到某个用户的第一帧视频数据
 *
 * @param userId
 * @param height
 * @param width
 */
@Override
public void onFirstRemoteVideoFrame(String userId, int height, int width) {}

```

音量相关回调

- 本端音量大小回调
- 对端音量大小回调

```

/**
 * 本端音量大小回调
 *
 * @param audioLevel
 */
@Override
public void onAudioLevelSend(String audioLevel) {}

/**
 * 对端音量大小回调
 *
 * @param audioLevel key:userId , value:音量等级
 */
@Override
public void onAudioLevelReceive(HashMap<String, String> audioLevel) {}

```

资源相关回调

- 远端用户发布了自定义视频流，返回视频流相关信息。
- 远端用户取消发布自定义视频流，返回取消发布的视频流的相关信息。

```
/**
 * 远端用户发布了自定义视频流
 * <p>调用RongCallClient.getInstance().publishCustomVideoStream(String tag, publishCallBack callBack) 方
法发布自定义视频流</p>
 * @param userId 用户 ID
 * @param streamId 自定义视频流 id
 * @param tag 自定义视频流 tag
 * @param surfaceView 自定义视频流视图
 */
@Override
public void onRemoteUserPublishVideoStream(String userId, String streamId, String tag, SurfaceView
surfaceView) {}

/**
 * 远端用户取消发布自定义视频流
 * @param userId 用户 ID
 * @param streamId 自定义视频流 id
 * @param tag 自定义视频流 tag
 */
@Override
public void onRemoteUserUnpublishVideoStream(String userId, String streamId, String tag) {}
```

分辨率/码率/帧率设置

更新时间:2024-08-30

在发起通话和接听通话前，可调用 [setVideoConfig] 设置音视频通话采用的分辨率、码率、和帧率。

设置分辨率

默认情况下，SDK 使用默认分辨率 RESOLUTION_480_640。

在发起通话和接听通话前，调用 [RCRTCVideoStreamConfig.Builder](#) 的 setVideoResolution 方法设置音视频通话采用的分辨率。

```
RCRTCVideoStreamConfig.Builder builder = RCRTCVideoStreamConfig.Builder.create();
builder.setVideoResolution(RCRTCVideoResolution.RESOLUTION_480_640);
builder.setMinRate(200);
builder.setMaxRate(900);
builder.setVideoFps(RCRTCVideoFps.Fps_15);

RongCallClient.getInstance().setVideoConfig(builder);
```

设置码率

默认情况下，SDK 根据当前分辨率进行匹配，自动适用对应的默认最小和最大码率设置。在通话过程中，实际视频码率在最小码率和最大码率之间根据网络情况浮动。

在发起通话和接听通话前，可以调整本端的最小和最大码率。调用 [RCRTCVideoStreamConfig.Builder](#) 的 setMinRate 设置最小码率。调用 setMaxRate 设置最大码率。码率单位为 kbps。

设置帧率

默认情况下，SDK 使用默认帧率 Fps_15。

在发起通话和接听通话前，可以调用 [RCRTCVideoStreamConfig.Builder](#) 的 setVideoFps 设置帧率，支持的帧率为 Fps_10、Fps_15、Fps_24、Fps_30。

摄像头设置

开启摄像头采集

更新时间:2024-08-30

在通话建立 (IRongCallListener.onCallConnected) 之后打开摄像头，对端 不会收到 IRongCallListener.onRemoteCameraDisabled 通知。

- 示例代码：

```
RongCallClient.getInstance().startCapture();
```

开关摄像头

在通话建立 (IRongCallListener.onCallConnected) 之后打开摄像头，对端 会收到 IRongCallListener.onRemoteCameraDisabled 通知。

- 示例代码：

```
RongCallClient.getInstance().setEnabledLocalVideo(true);
```

摄像头采集方向

在发起通话和接听通话前，使用 setCameraFrameOrientation 设置本地摄像头采集角度和视频编码使用的角度。

- 示例代码：

```
RongCallClient.getInstance().setCameraFrameOrientation(cameraOrientation, frameOrientation);
```

切换前后置摄像头

在通话建立 (IRongCallListener.onCallConnected) 之后，调用 switchCamera() 方法切换前后置摄像头，该方法适用于通过 SDK 打开默认摄像头的场景，配合 RongCallClient.startCall 使用，startCall 方法默认打开前置摄像头。

- 示例代码：

```
RongCallClient.switchCamera();
```

切换指定摄像头

获取到摄像头 Id 后，可调用 `switchCamera(int cameraId, boolean isMirror, CameraSwitchCallBack callback)` 方法切换指定摄像头。

- 示例代码：

```
RongCallClient.getInstance().switchCamera(0, false, new CameraSwitchCallBack() {  
    @Override  
    public void onCameraSwitchDone(boolean b) {  
    }  
  
    @Override  
    public void onCameraSwitchError(String s) {  
    }  
});
```

视频采集

视频采集

更新时间:2024-08-30

1. 在 `RongIM.connect` 连接成功之后，`RongCallClient.startCall` 或 `RongCallClient.acceptCall` 调用之前注册 `RongCallClient.registerVideoFrameListener` 监听，监听会根据设置的 [采集方式](#) 上报 YUV(NV21) 或 `texture` 类型的本地视频流数据。
2. 在通话结束前需调用 `RongCallClient.unregisterVideoFrameObserver` 取消注册。

- 参数说明：

参数	类型	必填	说明
listener	IVideoFrameListener	是	视频数据回调接口，用于开发者自定义美颜等视频处理

- 回调参数说明：

回调参数	回调类型	说明
callVideoFrame	CallVideoFrame	视频数据实体类

- 返回参数说明：

返回值	返回类型	说明
CallVideoFrame	CallVideoFrame	视频数据实体类

- 示例代码：

```
RongCallClient.getInstance().registerVideoFrameListener(new IVideoFrameListener() {  
  
    @Override  
    public CallVideoFrame processVideoFrame(CallVideoFrame callVideoFrame) {  
        //TODO 回调线程名:Camera SurfaceTextureHelper  
        return callVideoFrame;  
    }  
});
```

采集方式

在发起通话或接听通话前，设置摄像头采集数据类型。

- 参数说明：

参数	类型	必填	说明
textureAble	boolean	是	设置视频流是否采用 texture 采集，默认 true : texture 方式采集，false : yuv 方式采集。

- 示例代码：

```
RCRTCConfig.Builder builder = RCRTCConfig.Builder.create();
builder.enableEncoderTexture(true);
RongCallClient.getInstance().setRTCConfig(builder);
```

编解码器

软硬编码

更新时间:2024-08-30

在发起通话和接听通话前,使用如下方法设置设备是否采用硬编码。

- 参数说明：

参数	类型	必填	说明
hardWareEncode	boolean	是	是否使用 H264 硬编码, SDK 会根据硬件支持情况创建硬编码器, 如果创建失败则使用软编

- 代码示例：

```
RCRTCConfig.Builder builder = RCRTCConfig.Builder.create();
builder.enableHardwareEncoder(true);
RongCallClient.getInstance().setRTCConfig(builder);
```

硬编码等级设置

在发起通话和接听通话前,设置硬件编码器编码等级参数。

- 参数说明：

参数	类型	必填	说明
hardWareEncodeHighProfile	boolean	是	设置硬编码压缩等级是否为 MediaCodecInfo.CodecProfileLevel.AVCProfileHigh, ProfileHigh 比 AVCProfileBaseline 压缩率更高, 但是 AVCProfileBaseline 兼容性更好, AVCProfileHigh 压缩等级为 MediaCodecInfo.CodecProfileLevel.AVCLevel3, 默认值为 false, false 代表 MediaCodecInfo.CodecProfileLevel.AVCProfileBaseline

- 代码示例：

```
RCRTCConfig.Builder builder = RCRTCConfig.Builder.create();
builder.enableHardwareEncoderHighProfile(true);
RongCallClient.getInstance().setRTCConfig(builder);
```

软硬解码

在发起通话和接听通话前, 设置设备是否采用硬解码。

- 参数说明：

参数	类型	必填	说明
hardWareDecode	boolean	是	是否使用 H264 硬解码，默认是，SDK 会根据硬件支持情况创建硬解码器，如果创建失败会使用软解

- 代码示例：

```
RCRTCConfig.Builder builder = RCRTCConfig.Builder.create();
builder.enableHardwareDecoder(true);
RongCallClient.getInstance().setRTCConfig(builder);
```

视频转音频

视频转音频

更新时间:2024-08-30

当用户希望从视频通话转为音频时，可以调用 `RongCallClient` 的 `changeCallMediaType` 方法。目前仅支持视频单向往音频转换，即参数只能为 `RongCallCommon.CallMediaType.AUDIO`。

提示

转换前需要调用 `RongCallClient.getInstance().setEnabledLocalVideo(false)` 关闭摄像头采集。

- 示例代码：

```
RongCallClient.getInstance().changeCallMediaType(RongCallCommon.CallMediaType.AUDIO);
```

美颜处理

更新时间:2024-08-30

本文描述如何在融云音视频 SDK 基础上实现美颜功能。

官方美颜插件

您可以使用融云官方提供了基础美颜插件。

步骤 1：插件集成


集成要求使用的 CallLib 或 RTCLib 版本不小于 5.1.4。有以下两种集成方式：

Maven 集成

在 app/build.gradle 中填入：

```
dependencies {
    ...
    // x.y.z, 请填写具体的 SDK 版本号, 需与 CallLib 或 RTCLib 的版本号保持一致。
    implementation 'cn.rongcloud.sdk:face_beautifier:x.y.z' // 美颜扩展库 (可选)
}
```

本地集成

1. 融云 [官网下载](#)  手动集成，选择美颜选项 sdk 下载。
2. 将下载的 FaceBeautifier 模块引用到您的工程中。在使用音效的 Module 中添加依赖：

```
implementation project(':FaceBeautifier')
```

步骤 2：插件使用

目前融云 Android 音视频 SDK 只支持 Texture 纹理类型为 RGB 视频格式的美颜。即需要在代码中打开 enableEncoderTexture 设置，美颜功能才会生效：

```
RCRTCEngine.getInstance().init(getApplicationContext(),
RCRTCConfig.Builder.create().enableEncoderTexture(true).build());
```


美颜参数设置分为基础值设置和滤镜设置。详细值及接口说明请参考 [接口文档](#)。

美颜基础参数

基础参数目前包括：美白、磨皮、亮度、红润四个参数，取值范围为 [0-10]，0 代表无效果，10 代表最大效果。代码示例如下：

```
RCRTCBeautyOption beautyOption = RCRTCBeautyEngine.getInstance().getCurrentBeautyOption();
if (seekTypId == R.id.beauty_whiteness) {
    beautyOption.setWhitenessLevel(progress); // 设置美白参数
} else if (seekTypId == R.id.beauty_smooth) {
    beautyOption.setSmoothLevel(progress); // 设置磨皮参数
} else if (seekTypId == R.id.beauty_bright) {
    beautyOption.setBrightLevel(progress); // 设置亮度参数
} else if (seekTypId == R.id.beauty_ruddy) {
    beautyOption.setRuddyLevel(progress); // 设置红润参数
}
RCRTCBeautyEngine.getInstance().setBeautyOption(true, beautyOption); // true 是使用美颜，false 不使用美颜
```

美颜滤镜设置

滤镜目前包括：唯美、清新、浪漫三种风格，代码示例如下：

```
RCRTCBeautyFilter beautyFilter = RCRTCBeautyEngine.getInstance().getCurrentFilter();
switch (checkedId){
    case 0:{
        RCRTCBeautyEngine.getInstance().setBeautyFilter(RCRTCBeautyFilter.NONE); // 不使用美颜滤镜
        break;
    }
    case 1:{
        RCRTCBeautyEngine.getInstance().setBeautyFilter(RCRTCBeautyFilter.ESTHETIC); // 唯美
        break;
    }
    case 2:{
        RCRTCBeautyEngine.getInstance().setBeautyFilter(RCRTCBeautyFilter.FRESH); // 清新
        break;
    }
    case 3:{
        RCRTCBeautyEngine.getInstance().setBeautyFilter(RCRTCBeautyFilter.ROMANTIC); // 浪漫
        break;
    }
    default:{
        Log.e(TAG, "onCheckedChanged: [group, checkedId]" + checkedId);
        break;
    }
}
```

相芯美颜插件

提示

从 5.4.0 版本开始，融云 CallLib/CallKit SDK 支持相芯美颜插件。使用相芯美颜需要购买相关授权，详情请咨询融云商务。

相芯美颜插件支持美颜、滤镜、美形和美肤功能。后续版本将完善对相芯 SDK 的封装。如需实现更多特效，您可以直接集成相芯 SDK。

当前不支持同时使用官方美颜和相芯美颜插件。

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

集成相芯美颜插件

声明[融云 Maven 仓库](#)后，请在 app/build.gradle 文件中添加下面内容：

```
dependencies {  
    ...  
    // x.y.z, 请填写具体的 SDK 版本号, 请注意保持 CallLib 版本号保持一致。  
    implementation 'cn.rongcloud.sdk:fu_beautifier:x.y.z' // 相芯美颜扩展库 (可选)  
}
```

提示

x.y.z 代表当前 CallLib 的具体版本号。您可以在[融云下载页](#)或[融云 Maven 仓库](#)进行查询。如果集成了 CallKit，则必须保持 CallKit、CallLib、fu_beautifier 插件版本号前三位一致。

初始化

初始化时请提供有效的相芯美颜的授权文件（请联系融云商务购买）。应用运行期间只调用一次即可，建议在 Application#onCreate 中初始化。

```
RCRTCfUBeautiflerEngine.getInstance().register(null, authpack.A());
```

美颜开关

打开美颜开关后设置的美颜效果才会生效；关闭开关美颜会失效。

```
RCRTCfUBeautiflerEngine.getInstance().setBeautyEnable(enabled);
```

滤镜设置

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

```
RCRTCfUBeautiflerEngine.getInstance().setFilter(FaceBeautyFilterEnum.FENNEN_1, (float) val);
```

美型设置

美型可设置瘦脸程度、V脸程度、窄脸程度、小脸程度、瘦鼻程度、嘴巴调整程度、开眼角强度、眼睛间距、鼻子长度、人中

长度、微笑嘴角强度、瘦颧骨强度等效果。下面是设置瘦脸程度的接口调用示例，其他接口请参考相芯美颜插件 Javadoc 文档。

```
RCRTCfUBeautiflerEngine.getInstance().setCheekThinningLevel((float) val);
```

美肤设置

可设置磨皮、美白、红润、锐化、亮眼、美牙、去除黑眼圈、去除法令纹强度等。下面是设置磨皮强度的接口调用示例，其他接口请参考相芯美颜插件 Javadoc 文档。

```
// 设置磨皮强度  
RCRTCfUBeautiflerEngine.getInstance().setSkinBlurLevel((float) val);
```

重置所有美颜效果

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

```
RCRTCfUBeautiflerEngine.getInstance().reset();
```

麦克风设置

常规接口

更新时间:2024-08-30

打开/关闭麦克风

当通话中希望关闭麦克风，可调用 RongCallClient 的 `setEnableLocalAudio` 接口，传入 `false` 达到本地静音效果；当需要再次打开时，传入 `true` 即可。默认值为 `true`，即麦克风打开状态。

- 示例代码：

```
RongCallClient.getInstance().setEnableLocalAudio(true);
```

进阶接口

采集音源设置

当使用嵌入式设备或个别少数定制机型，出现麦克风采集异常时，可尝试设置音源接口

`RCRTCConfig.Builder.setAudioSource`，传入其他音源类型。此接口默认值为 `VOICE_COMMUNICATION`，建议在默认值无效情况下，优先尝试 `DEFAULT` 或 `MIC`。参数含义详情，请参考 [安卓官网](#)。

- 示例代码：

```
RCRTCConfig.Builder builder = RCRTCConfig.Builder.create();  
builder.setAudioSource(AudioSource.MIC);  
RongCallClient.getInstance().setRTCConfig(builder);
```

警告

一般普通手机开发，不需要用到此接口，请勿随意调用，以免造成声音采集异常。

扬声器设置

听筒/扬声器切换

更新时间:2024-08-30

当通话中希望切换声音播放是由扬声器还是听筒输出时，可调用 `RongCallClient` 的 `setEnabledSpeakerphone` 来设置。传入 `true` 代表使用扬声器播放；`false` 代表使用听筒播放。默认是 `false` 即使用听筒播放。

- 示例代码：

```
RongCallClient.getInstance().setEnabledSpeakerphone(false);
```

⚠ 警告

此接口在 `IRongCallListener.onCallConnected(RongCallSession, SurfaceView)` 之后调用才有效。

音频路由

更新时间:2024-08-30

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

- 设置默认音频路由（要求 SDK 版本 \geq 5.3.2）。在无外接设备时，使用默认的音频输出设备（设备内置的听筒或扬声器）。一旦接入外部设备，SDK 仅会使用外接设备。
- 在接入有线耳机、蓝牙耳机、蓝牙音响等外部设备时，自动根据设备连接顺序、可用状态切换当前输出设备。当有多个外接设备时，音频会通过最后一个接入的设备播放。

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

初始化音频路由管理类

使用 SDK 提供的音频路由管理功能需首先进行初始化，其中 context 参数推荐使用 Application。

```
// 初始化音频路由管理类
RCRTCAudioRouteManager.getInstance().init(context);
```

若要停止使用自动音频路由功能，将其进行反初始化即可。

```
// 反初始化音频路由管理类
RCRTCAudioRouteManager.getInstance().unInit();
```

默认音频路由

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

在不同的[音频模式](#)下，SDK 使用的默认音频路由如下所示：

- 音视频通话、会议（`AudioScenario.DEFAULT`）：听筒
- 语聊房，音乐播放场景（`AudioScenario.MUSIC_CHATROOM`）：扬声器
- 音乐教学场景（`AudioScenario.MUSIC_CLASSROOM`）：扬声器

更改默认音频路由

当 App 用户没有连接外部音频输出设备时，SDK 会使用默认音频路由。

加入房间前，调用 RCRTCEngine 下 `setDefaultAudioRouteToSpeakerphone` 可更改默认音频路由。通过该方法的 `defaultToSpeaker` 参数控制 SDK 是否使用扬声器播放音频。该接口在初始化音频路由前后都可以调用，但配置仅在音频路由管理类初始化完成后生效。

以下示例中默认音频路由被修改为听筒。

```
// 设置默认音频路由为设备内置听筒
RCRTEngine.getInstance().setDefaultAudioRouteToSpeaker(false);
```

更改当前音频路由

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

- 直接修改 SDK 默认音频路由配置，您可以在通信过程中在扬声器、听筒之间动态切换。
- 调用 [enableSpeaker](#) 方法可以在通信过程中在扬声器、听筒之间动态切换。设置为 `false` 时 SDK 会恢复使用听筒。该方法在加入房间前后调用均可生效。`enableSpeaker` 只切换当前的音频路由为扬声器或听筒，不会影响 SDK 的默认音频路由设置。此时如果将外置设备连接再全部移除，SDK 会恢复为使用默认音频路由。

以下示例中调用 `enableSpeaker` 方法，更改当前音频路由为扬声器。

```
// 设置音频路由为扬声器
RCRTEngine.getInstance().enableSpeaker(true);
```

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

1. 音频路由自动切换到 App 用户连接的外部音频输出设备。
2. 如果 App 用户先后连接了多个外部设备，则音频路由会自动切换到最后一个连接的音频输出设备。在同时连接有线耳机和连接蓝牙耳机的状态下，首次初始化或者重置音频路由状态会优先蓝牙输出。初始化完成之后会根据蓝牙和有线耳机的连接顺序进行选择，后连接者优先。
3. 如果 App 用户移除当前输出设备，则音频路由会自动切换到上一个连接的音频输出设备。
4. (SDK \geq 5.3.2) 如果 App 用户移除所有外接音频输出设备，SDK 切换到使用默认音频路由输出。

当移动设备连接到耳机或蓝牙音频设备时，无法将通过 [enableSpeaker](#) 更改为扬声器。

- 如果 SDK 版本 $<$ 5.3.2，SDK 会保留 `enableSpeaker` 设置的听筒和扬声器的输出状态，耳机断开之后会根据之前的状态选择听筒或者扬声器播放。
- 如果 SDK 版本 \geq 5.3.2，SDK 会在 App 用户移除所有外接音频输出设备后切换到使用默认音频路由输出。

在初始化 SDK 提供的音频路由管理类之后，不推荐 App 再调用系统的 `AudioManager` 修改音频输出通道，否则可能导致 SDK 内部的音频路由由管理类状态错误。若必须使用，请在相关业务处理完成之后调用 `resetAudioRouteState` 方法重置音频路由由管理类状态。

```
// 重置音频路由由管理类状态
RCRCAudioRouteManager.getInstance().resetAudioRouteState();
```

获取当前音频路由

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

App 可通过 `setOnAudioRouteChangeListener` 设置监听。在使用完成之后请及时将 `listener` 设置为 `null`，避免造成内存泄漏。

```
// 设置音频输出设备改变监听
IRCRTCAudioRouteManager.getInstance().setOnAudioRouteChangeListener(new IRCRTCAudioRouteListener() {
    @Override
    public void onRouteChanged(RCAudioRouteType type) {

    }
});

// 使用完成之后，请及时设置为 null，避免造成内存泄漏
IRCRTCAudioRouteManager.getInstance().setOnAudioRouteChangeListener(null);
```

RCAudioRouteType 类型	说明
SPEAKER_PHONE	扬声器
EARPIECE	听筒
HEADSET	有线耳机
HEADSET_BLUETOOTH	蓝牙耳机

CallLib 3.X 升级到 5.X

更新时间:2024-08-30

本文描述 CallLib SDK 的升级步骤。

升级概述

CallLib SDK 5.X 是基于 AndroidX 开发的新版 SDK，功能更丰富，更稳定，并在之前版本上修复了大量问题，建议尽早升级至新版 CallLib SDK。

前置条件

- CallLib SDK 依赖 IMLib 请您确保已将 IMLib 升级至 5.X
- 已遵照 IMLib 升级要求将您的工程升级至 AndroidX。

修改依赖方式

以下仅介绍 maven CallLib 还支持本地依赖方式，请参考[导入 CallLib SDK](#)。

修改 Maven 仓库地址

```
maven {url "https://dl.bintray.com/rongcloud/maven"} // 3.X
```

```
maven {url "https://maven.rongcloud.cn/repository/maven-releases/"} // 5.X
```

修改依赖命名

```
dependencies {  
// x.y.z，请填写具体的 SDK 版本号，新集成用户建议使用最新版。  
implementation 'cn.rongcloud.sdk:call_lib:x.y.z'  
implementation 'cn.rongcloud.sdk:im_lib:x.y.z'  
}
```

提示

- 各个 SDK 的最新版本号可能不相同，还可能是 x.y.z.h，可前往 [融云官网 SDK 下载页面](#) 或 [融云的 Maven 代码库](#) 查询。
- 从 5.2.0 版本开始，CallKit/CallLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

接口变化

设置音视频引擎服务器地址信息

如果您的应用使用海外数据中心或私有化部署，请检查是否使用 `setEngineServerInfo`

请遵照[数据中心](#) 替换为以下两个方法：

```
/**
 * 设置 海外 数据中心的导航服务器和媒体服务器地址。
 * 此方法要在 {@link #init(Context, String)} 前使用
 *
 * @param naviServer 海外数据中心的导航服务器地址。
 * @param fileServer 海外数据中心的媒体服务器地址，即文件和图片的上传地址。使用独立数据中心时必须填写。
 */
RongIMClient.setServerInfo(final String naviServer, final String fileServer);
// 设置音视频媒体服务器地址
RCRTCEngine.getInstance().setMediaServerUrl(String mediaServerUrl);
```

修改视频属性设置

3.X 的 `RongRTCConfig` 包含了视频属性，音频属性，硬件采集属性设置。在 5.X 中视频属性单独使用 `RCRTCVideoStreamConfig` 配置

```
// 5.X设置分辨率，码率，帧率
RCRTCVideoStreamConfig.Builder rcrtcVideoStreamConfig = RCRTCVideoStreamConfig.Builder.create()
    .setVideoResolution(RCRTCVideoResolution.RESOLUTION_480_640)
    .setVideoFps(RCRTCVideoFps.Fps_15)
    .setMaxRate(1000)
    .setMinRate(350);
RongCallClient.getInstance().setVideoConfig(rcrtcVideoStreamConfig);
```

在 5.X 中音频属性单独使用 `RCRTCAudioStreamConfig` 配置

```
RCRTCAudioStreamConfig.Builder builder = RCRTCAudioStreamConfig.Builder.create()
    .setNoiseSuppression(RCRTCParamsType.NSMode.NS_MODE3);
RongCallClient.getInstance().setAudioConfig(builder);
```

3.X `RongRTCConfig` 中的其他配置可以使用 5.X 的 `RCRTCConfig` 进行替换。详见[引擎配置](#)。

```
/**
 * 视频流采集方式，设置视频流是否采用 texture 采集。一般安卓 5.0 以下系统建议使用 YUV 采集，以避免低版本系统 texture 的
 * 兼容性问题。
 *
 * @param enabled 默认为 true，即 texture 方式采集；当为 false 时，即 yuv 方式采集。
 */
RCRTCConfig.Builder rcrtcConfig = RCRTCConfig.Builder.create()
    .enableEncoderTexture(false);

RongCallClient.getInstance().setRTCConfig(rcrtcConfig);
```

CallKit 3.X 升级到 5.X

更新时间:2024-08-30

本文只描述 CallKit SDK 的升级步骤。如果您的 App 工程中调用了 CallLib RongCallClient 下的方法，请在完成本文档升级步骤后继续前往 CallLib 升级文档。

升级概述

CallKit SDK 5.X 是基于 AndroidX 开发的新版 SDK，功能更丰富，更稳定，并在之前版本上修复了大量问题，建议尽早升级至新版 CallKit SDK。

前置条件

- CallKit SDK 依赖 IMKit 请您确保已将 IMKit 升级至 5.X
- 已遵照 IMKit 升级要求将您的工程升级至 AndroidX。

评估升级工作量

- 如果您 App 未修改 CallKit 源码（例如采用 maven 依赖）直接按照以下方式修改依赖即可完成升级。
- 如果您对 CallKit 代码做了少量修改，您可以依照集成后的报错提示，进行修改。
- 如果您对 CallKit 代码做了大量修改，建议您查询您的修改记录，使用 5.X CallKit 重新实现个性化需求。

修改依赖方式

以下仅介绍 maven 集成方式。CallKit 还支持本地依赖方式，请参考[导入 CallKit SDK](#)。

修改 Maven 仓库地址

```
maven {url "https://dl.bintray.com/rongcloud/maven"} // 3.X
```

```
maven {url "https://maven.rongcloud.cn/repository/maven-releases/"} // 5.X
```

修改依赖命名

```
dependencies {  
    // x.y.z，请填写具体的 SDK 版本号，新集成用户建议使用最新版。  
    implementation 'cn.rongcloud.sdk:call_kit:x.y.z'  
    implementation 'cn.rongcloud.sdk:call_lib:x.y.z'  
    implementation 'cn.rongcloud.sdk:im_kit:x.y.z'  
    implementation 'cn.rongcloud.sdk:im_lib:x.y.z'  
}
```

④ 提示

- 各个 SDK 的最新版本号可能不相同，还可能是 x.y.z.h，可前往 [融云官网 SDK 下载页面](#) 或 [融云的 Maven 代码库](#) 查询。
- 从 5.2.0 版本开始，CallKit/CallLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

CallKit 5.X 升级

升级到 CallKit 5.4.0

更新时间:2024-08-30

在 CallKit 5.4.0 版本中，SDK 内部移除了对 IMLib 的依赖。

如果您从低版本升级至 CallKit 5.4.0，请遵照官方对 SDK 版本的要求，保持 CallKit/CallLib/IMKit/IMLib 均为 5.4.0，可实现平滑升级。

注意：如果需要升级项目依赖的 SDK 到 5.4.0，但无法同时升级 CallKit（通常因项目中使用并修改大量了 CallKit 源码），项目可能无法正常运行。如果遇到问题，您可以遵照以下列表，在项目中替换项目中部分代码：

- 替换 `io.rong.imlib.RongIMClient.MediaType` 为 `io.rong.imlib.IRongCoreEnum.MediaType`
- 替换 `io.rong.imlib.RongIMClient.ErrorCode` 为 `io.rong.imlib.RongCoreClient.ErrorCode`
- 替换 `io.rong.imlib.RongIMClient.ConnectionStatusListener.ConnectionStatus` 为 `io.rong.imlib.IRongCoreListener.ConnectionStatusListener.ConnectionStatus`

建议及时完善 App 项目，遵照官方指导保持所有 SDK 版本一致性，以享受最新修复及功能。

CallLib 升级 CallPlus

更新时间:2024-08-30

本文描述使用 CallLib 切换 CallPlus 的升级步骤。

升级概述

使用 CallLib 用户需要切换 CallPlus 且不做接口调用上的修改，需要参照此文档进行无缝升级切换。

本次升级方案中新增了 call_plus_wrapper 适配层插件，您只需要删除已经集成的 CallLib SDK，重新集成 call_plus_wrapper，CallPlus 即可完成切换升级。

前置条件

- 要求使用 CallLib 版本在 5.8.2 以上。
- 要求使用 CallPlus 版本在 2.1.4 以上。

Maven 集成

1. 删除您工程中集成的 CallLib Moudle 本地依赖，或者删除 CallLib Maven 依赖。
2. 在应用的 build.gradle 中，添加对 IMLibCore、CallPlus、RTCLib、call_plus_wrapper 的远程依赖项。融云 CallPlus 业务依赖 IM 通道，所以必须同时集成 IMLibCore SDK。

```
dependencies {  
    // x.y.z，请填写具体的 SDK 版本号，新集成用户建议使用最新版。  
    implementation 'cn.rongcloud.sdk:im_libcore:x.y.z' // 即时通讯基础能力库  
    implementation 'cn.rongcloud.sdk:callplus_lib:x.y.z' // 音视频呼叫能力库  
    implementation 'cn.rongcloud.sdk:rtc_lib:x.y.z' // 音视频通话基础能力库  
    implementation 'cn.rongcloud.sdk:call_plus_wrapper:x.y.z' // calllib 适配层插件  
}
```

③ 提示

各个 SDK 的最新版本号可能不相同，还可能是 x.y.z.h，可前往 [融云官网 SDK 下载页面](#) 或 [融云的 Maven 代码库](#) 查询。

Android 本地库模块

在导入 SDK 前，您需要前往[融云官网 SDK 下载页面](#)，将音视频通话能力-CallPlusLib及其依赖的即时通讯能力库-IMLib和音视频基础能力-RTCLib下载到本地。

1. 删除您工程中集成的 CallLib Moudle 本地依赖，或者删除 CallLib Maven 依赖。
2. 将 `rong_call_plus_wrapper_x.x.x.jar`, `rong_callPlus_2.x.x.aar` 文件复制到 Android Studio 项目的 `libs` 目录下，在应用的 `build.gradle` 中，添加 `*.aar` 依赖。
3. 依次点击 **File > New > Import Module**，将 `imlib` 和 `rtclib` 的 Module 组件导入。

```
dependencies {
implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
implementation project(path: ':imlib')
implementation project(path: ':rtclib')
}
```

提示：更详细的Call Plus 集成文档可前往融云官网 [Call Plus 集成页面](#)。

接口替换说明

本次升级主要针对使用 CallLib 能力通话用户，升级后使用的接口还是 CallLib 现有接口，无需调整。
初次集成需要参照 [实现音视频通话](#)

使用 CallKit 切换 CallPlus

本质上还是 CallLib 切换 CallPlus 的过程，注意上述过程中对 CallLib & CallPlus 的版本要求。

客户端 API

RTCLib

更新时间:2024-08-30

以下是 RTCLib 5.x 的 API 参考文档：

- [RTCLib 音视频会议、低延迟直播 \(无 UI\)](#) [↗](#)

RTCLib 5x 支持多种插件。以下是插件的 API 参考文档：

- [FaceBeautifier 美颜](#) [↗](#)

CallLib

以下是 CallLib 5.x 的 API 参考文档：

- [CallLib 音视频通话 \(不含 UI\)](#) [↗](#)

CallKit

以下是 CallKit 5.x 的 API 参考文档：

- [CallKit 音视频通话 \(含 UI\)](#) [↗](#)

更新日志 5.8.2

更新时间:2024-08-30

发布日期：2024/06/05

- **RTCLib SDK**

- 提升了 SDK 版本号为 5.8.2，无新增特性与修复。

- **CallLib SDK**

- 提升了 SDK 版本号为 5.8.2，无新增特性与修复。

- **CallKit SDK**

- 提升了 SDK 版本号为 5.8.2，无新增特性与修复。

5.8.0

发布日期：2024/03/26

- **RTCLib SDK:**

- 优化了域名解析逻辑。
- 修复了蓝牙设备重启导致 `NullPointerException` 异常崩溃的问题。

5.6.9

发布日期：2024/01/31

- **RTCLib SDK:**

- 修复：OPPO Reno 11 手机上硬解码崩溃的问题

- **CallKit SDK:**

- 修复：CallKit 悬浮窗不支持左右移动的问题。
- 修复：用户 ID 带下划线导致多人通话小窗不显示用户昵称的问题。

5.6.8

发布日期：2023/12/29

- **RTCLib SDK:**

- 修复：播放在线混音文件时，因混音文件缺少 channel 信息导致的崩溃问题。

5.6.7

发布日期：2023/11/30

• RTCLib SDK:

- 修复：修复部分内部已知问题。

5.6.5

发布日期：2023/10/12

• RTCLib SDK:

- 修复：跨房间连麦场景下副房间 ping 超时，onKicked 回调方法中 roomId 参数错误的问题。
- 修复：将 syncRoomResources 同步房间数据接口改用 pullKV 方式并补充同步副房间数据的逻辑。
- 修复：断线重连后如果房间 SessionId 发生变化应主动退出房间并通知应用层。
- 修复：小米 12S PRO MIUI 14 MediaPlayer 播放 -19 问题。

5.6.4

发布日期：2023/09/25

• RTCLib SDK:

- 新增：适配 Camera2，暂仅支持（Texture），可以通过 RCRTConfig 中的 enableCamera2 方法启用
- 修复：去除 logcat 中的敏感信息，如 UserID、token 等。

5.6.3

发布日期：2023/08/31

• RTCLib SDK:

- 修复：多人视频通话页面中 webview 使用方式检测有漏洞
- 修复：适配 Android12及以上音视频通话中开启悬浮窗权限导致录音被中断的问题
- 修复：由通知栏接听进入多人音视频通话页面会显示重复的参会人的问题

• CallKit SDK:

- 修复：Callkit 中 roomType 转换异常

5.6.2

发布日期：2023/08/11

- **RTCLib SDK:**

- 修复：主播切换为观众角色时未清空任务队列，导致执行队列中的重连任务引起空指针异常的问题
- 修复：`PeerConnectionFactory` 中的埋点方法 `recordFirstFrameEncodeDecodeInfo` 空指针异常
- 修复：`RongRTCSessionManager` 未初始化导致空指针异常

- **CallKit SDK:**

- 修复：视频通话中，添加用户通话，用户连接中时头像显示黑屏
- 修复：`VoIPBroadcastReceiver` 中未找到跳转目标 Activity 造成崩溃
- 修复：挂断后离线推送还在响铃，并且使用听筒而非外放播放铃声
- 修复：通知栏通知只有铃声没有震动

5.6.1

发布日期：2023/07/14

- **RTCLib SDK:**

- 修复使用 `startEchoTest` 硬件检测功能偶现关闭后还会播放的问题

- **CallKit SDK:**

- 修复 Google FCM 离线音视频推送会收到两条推送的问题

5.6.0

发布日期：2023/07/03

- **RTCLib SDK:**

- 修复：`MixAudioTrack` 多线程问题导致的 `RejectedExecutionException` 异常崩溃
- 修复：网络探测地址列表多线程数据不同步导致 `ArrayIndexOutOfBoundsException` 异常崩溃

5.5.0

发布日期：2023/09/08

为配合 IM SDK 5.5.0 稳定 (stable) 版本使用，提升 SDK 版本号为 5.5.0。功能基于 5.4.6 最新 Hotfix 版本，无新增特性与修复。

5.4.6

发布日期：2023/06/15

- **CallKit SDK:**

1. 优化：统一了 CallKit 头像样式

- **RTCLib SDK:**

1. 修复：订阅列表中有失败资源列表，由于端上处理异常导致空指针问题，出现订阅全部失败问题
2. 修复：订阅合流后无法解析各个主播的音量大小的问题
3. 修复：AudioTrack play 方法崩溃的问题

5.4.5

发布日期：2023/05/29

- **CallKit SDK:**

1. 修复：`SingleCallActivity` 中 `addRemoteVideoView` 方法导致 `NullPointerException` 崩溃的问题
2. 修复：Android 13 手机上多人音频通话场景下，切到悬浮窗后再切到选择成员列表页面，返回后会造成通话页面头像重复的问题

- **RTCLib SDK:**

1. 修复：日志分割异常导致的崩溃问题
2. 修复：荣耀系列手机降低分辨率编码导致绿屏的问题
3. 优化：为小语种增加是否检查权限接口
4. 修复：关闭摄像头时崩溃报错 `:RuntimeException:getParameters failed (empty parameters)` 的问题
5. 修复：`MixAudioTrack` 中 `stop audioTrack` 造成 `IllegalStateException` 崩溃异常的问题

5.4.4

发布日期：2023/05/11

- **CallKit SDK:**

1. CallKit 接收 FCM 离线推送后可弹出通知

- **RTCLib SDK:**

1. 修改 RTCLib 与 IM SDK 版本匹配规则。从 5.4.4 开始，要求前两位保持一致。注意，RTCLib 5.4.4 不可匹配小于 5.4.4 的 IM SDK。
2. 修复结束通话没有走 VOIP，挂断推送中英文问题
3. 修复探测模块数组越界问题
4. 修复通话 session 为空（表示通话已经结束），但依然启动了本页面，导致页面无法销毁的问题
5. 修复 AudioEffectManager 类 AudioTrack 空指针问题
6. 修复 General Mobile G312 手机再进前台，通话未结束问题

5.4.3

发布日期：2023/04/21

问题修复：

1. 紧急修复 5.4.2 版本中与导航服务地址相关的问题。

5.4.2

发布日期：2023/04/20

• RTCLib SDK:

1. 修复：调用 Server API 踢出直播间观众后，观众端未接收到 onKicked 回调的问题
2. 修复：在 Android 13 上禁止音视频权限后，再次点击音视频页面闪烁，没有弹起权限申请的问题

• CallLib SDK:

1. 新增：音视频信消息推送默认设置 vivo category 参数为 IM
2. 修复：修复主播下麦时偶现的崩溃问题
3. 修复：iOS 发送的音视频挂断消息经华为推送接收后显示为空的问题
4. 修复：在判断是否重启录音 audioRecorder 时的空指针问题

• CallKit SDK:

1. 优化：单群聊音视频推送的文案显示
2. 修复：群聊音视频通话过程中，将悬浮框拉到最底部，弹起系统键盘，悬浮框被遮挡的问题

5.4.1

发布日期：2023/04/07

• RTCLib SDK:

1. 新增：美声插件新增部分美声特效
2. 修复：修复主播下麦时偶现的崩溃问题

• CallKit SDK:

1. 优化: 对齐 Android、iOS 的多语言提示文案
2. 优化：在 CallKit 的邀请通话推送内容中去掉 username 字段
3. 修复：音视频通话的过程中，当前用户的小屏画面在切换到手机桌面前后位置发生变更的问题
4. 修复：7人视频通话中，切换部分其他用户为大屏后，会先显示大屏画面黑屏再恢复正常显示的问题
5. 修复：摄像头关闭后，切换本端画面为小屏，一直显示黑屏的问题

6. 修复：音视频通话中，关闭自己摄像头切为小屏后打开摄像头，小屏不显示画面仍显示本端用户头像的问题
7. 修复：在与 PC 端 1v1 视频通话中，PC 端关闭摄像头后将其切换为大屏，PC 端此时打开摄像头导致 Android 端崩溃的问题
8. 修复：在与 PC 端 1v1 视频通话中，PC 端关闭摄像头后移动端卡在最后一帧的问题
9. 修复：在 OPPO reno6 手机上，音视频群组通话中小窗不显示用户名，切换大小窗口后才能显示的问题

5.4.0

发布日期：2023/03/03

• CallLib SDK:

1. 新增：音视频信消息推送默认设置华为 category 参数为 VOIP
2. 修复：群组通话再次邀请已在通话中的人员进行通话，从发起端对已在通话中的人员进行过滤
3. 新增：CallLib 可接入相芯美颜插件

• CallKit SDK:

1. 新增：CallKit 可接入相芯美颜插件
2. 优化：RTCLib、CallLib、CallKit 去除 IMLib 库引用，改用 IMLibCore 库。部分导入 CallKit 源码客户可能需要修改项目，详见升级 CallKit 文档。

• RTCLib SDK:

1. 优化：相芯美颜插件初始化不再依赖依赖 RCRTCEngine 初始化
2. 优化：RTCLib 初始化方法，增加 ErrorCode 返回值
3. 优化：RTCLib 适配 Android 13 读写权限
4. 修复：AudioTrack 创建失败导致崩溃问题

5.3.8

发布日期：2023/07/07

为配合 IM SDK 5.3.8 稳定 (stable) 版本使用，提升 SDK 版本号为 5.3.8。功能基于 5.3.5 版本，无新增特性与修复。

5.3.6

发布日期：2023/05/11

为配合 IM SDK 5.3.6 稳定 (stable) 版本使用，提升 SDK 版本号为 5.3.6。功能基于 5.3.5 版本，无新增特性与修复。

5.3.5

发布日期：2023/02/10

• CallKit SDK:

1. 修复：在某些手机上，单聊视频通话，切换大小显示窗口时，小窗口会出现在屏幕左上角闪烁的问题

5.3.4

发布日期：2023/01/17

• RTCLib SDK:

1. 新增：订阅流接口支持在订阅流失败时返回订阅失败的资源列表
2. 新增：新增调节远端资源的播放音量的接口 `adjustRemotePlaybackVolume`
3. 新增：支持本地采集音量增益，调节范围由 [0-100] 改为 [0-200]
4. 修复：切换混音模式影响混音背景音乐声音大小的问题

• CallLib SDK:

1. 修复：在 Android 端与 iOS 端通话过程中，iOS 接听系统来电会导致 Android 端崩溃的问题

• CallKit SDK:

1. 优化：音视频通话过程中，若本端接听 SIM 来电，则断开音视频通话。拒绝SIM来电，音视频通话保持正常
2. 修复：在通话过程中，PC 端关闭摄像头邀请 Android 端通话，Android 端接听后 PC 端画面显示为透明的问题
3. 修复：在群多人通话中，Android 端将已关闭摄像头的 PC 端用户切换为大屏画面，此时 PC 端用户打开摄像头会导致显示不正常的问题
4. 修复：Android 12 及以上手机连接蓝牙后，接听音视频通话会外放本端和对端声音的问题
5. 修复：接听前预览导致接听群组视频通话时崩溃的问题

5.3.3

发布日期：2022/12/28

• RTCLib SDK:

1. 修复：弱网情况下 CDN 播放器播放 CDN 直播流延迟持续增加的问题
2. 修复：卡顿率首次上报时间戳错误

• CallLib SDK:

1. 修复：群组音视频通话时，邀请其他人加入通话必现崩溃
2. 修复：因 `IRongCallListener#onRemoteUserJoined` 和 `onRemoteUserPublishVideoStream` 回调执行顺序差异，导致老版本 CallKit 显示黑屏的问题
3. 修复：主叫方看被叫方黑屏的问题

5.3.2

发布日期：2022/12/02

• RTCLib SDK:

1. 新增: 恢复/暂停音频模块接口 `pauseAudioModule()/resumeAudioModule()`，用于处理音频打断
2. 新增：默认音频路由接口 `setDefaultAudioRouteToSpeakerphone()`
3. 新增：支持设置耳返音量
4. 优化：完善高频收到拉取房间状态 notify 的情况
5. 优化: 离开房间时取消未及时执行的 RTC 信令
6. 修复：蓝牙耳机播放音乐上下麦时短暂声音异常问题
7. 修复：同时使用有线耳机、蓝牙耳机音频路由状态乱序问题

5.3.1

发布日期：2022/11/18

• RTCLib SDK:

1. 新增: 网络质量探测较差时，本端发布资源自适应调整分辨率
2. 优化：加载美颜插件改到子线程，避免可能发生的 ANR 问题
3. 修复：开黑模式下部分机型采集到第三方软件音乐问题
4. 修复：OPPO Reno8 订阅音频资源时崩溃问题
5. 修复：在 Web 端主播用户断线重连后重新加入房间但未重新发布资源时，Android 端会错误解析成用户取消发布资源动作。
6. 修复偶现的订阅任务没有回调导致任务队列阻塞
7. 修复大型会议场景下的视频屏幕闪烁问题
8. 修复红米 9A 蓝牙耳机音乐卡顿、sco 模式下不能采集人声的问题

• CallKit SDK:

1. 修复：发起端无 Camera 设备，对端无法更新 CallKit 的连接状态

5.3.0

发布日期：2022/11/04

• RTCLib SDK:

1. 新增：融云 CDN 插件新增 CDN 播放器组件，支持播放外部 URL
2. 新增：正式支持媒体补充信息（SEI）功能
3. 优化：短音效接口不依赖身份，同时支持本地播放
4. 优化：优化音频 3A 效果优化，降低回声
5. 优化：优化了内置 CDN 拉取首屏速度
6. 优化：在使用相机、录音设备、存储时，增加权限检查。如未授予权限，则抛出错误，主流程继续执行
7. 优化：提高了蓝牙耳机通话中断后重连的体验
8. 修复：armV7 架构下偶现的崩溃问题 `crash for not found RCRTC_ffmpeg`
9. 修复：Android 12 及以上机型初始化音频路由未授予 BLUETOOTH_CONNECT 权限导致崩溃
10. 修复：观众调用 `switchRole` 成为主播并发布资源后，小流上行分辨率一直为 0
11. 修复：小流设置帧率不生效
12. 修复：调节混音本端音量，远端也受到了影响

5.2.5

发布日期：2022/09/09

• RTCLib SDK:

1. 新增：相芯美颜插件
2. 优化：各端对齐大小流分辨率
3. 修复：蓝牙耳机空指针问题
4. 优化：Android 12 以上动态申请蓝牙连接权限
5. 修复：屏幕共享黑屏问题
6. 修复：开启耳返、扬声器出现的扬声器无效问题

• CallLib SDK:

1. 修复：CallLib 通话过程中被踢出房间后，生成的通话记录不显示通话时长问题

• CallKit SDK:

1. 去掉 RongCallKit 中 `checkEnvironment` 相关逻辑

5.2.4

发布日期：2022/07/22

• RTCLib SDK:

1. 修复：修复了5.1.7版本后ffmpeg和三方库冲突的问题;

- **CallLib SDK:**

1. 新增：Call Lib新增音频首帧回调;

5.2.3

发布日期：2022/06/01

- **RTCLib SDK:**

1. 增加：支持了 x64 架构;
2. 修复：修复了手动对焦模式不生效问题;
3. 修复：修复了水印功能不生效问题;

5.2.2

发布日期：2022/05/01

- **RTCLib SDK:**

1. 增加：修复了一些内部的 BUG;

5.2.1

发布日期：2022/04/01

- **RTCLib SDK:**

1. 增加：适配 Android 12;
2. 修复：修复了接入美颜模块后，纹理转换导致的gl1282崩溃;
3. 修复：修复了重复开关屏幕共享引起的崩溃问题;

5.2.0

发布日期：2022/03/01

从 5.2.0 版本开始，CallKit/CallLib/RTCLib 必须与其依赖的 IMKit/IMLib SDK 保持版本一致（前三位必须保持一致）。

- **RTCLib SDK:**

1. 增加：音频首帧回调;
2. 修复：客户端获取了服务端开启软解码的配置，由于客户端解析错误导致软解码未生效问题;

3. 修复：屏幕共享流在发布前设置分辨率，空指针崩溃;

5.1.17

发布日期：2022/01/26

• RTCLib SDK:

1. 增加：新增通话/会议/直播前检测网络状态的接口 `startLastmileProbeTest(LastmileProbeConfig config)`;
2. 增加：跨房间连麦支持主播占位图功能;
3. 优化：音乐聊天室模式和音乐教学模式下麦克风采集存在一定噪音的问题;
4. 修复：Server 配置了帧率分辨率等信息后，客户端未生效;
5. 修复：首次设置屏幕共享流的分辨率不生效;

• CallLib SDK:

1. 修复：alreadyhangupid 未清空导致群组通话无法挂断;

5.1.16

发布日期：2022/01/13

• RTCLib SDK:

1. 增加：在直播/会议场景下，支持在主播或参会者的视频图像上添加水印;
2. 优化：升级美颜模块，解决 Google 商店扫描加密漏洞的问题;
3. 优化：获取远端用户列表无序的问题;
4. 修复：发布自定义文件流时传入错误地址导致应用崩溃的问题;

• CallLib SDK:

1. 修复：通话过程中视频模式转语音模式之后获取 MediaType 数值错误的问题;

• CallKit SDK:

1. 修复：SDK 内部缺失 `androidx.media:media` 依赖导致应用在后台时不显示来电呼入的问题;
2. 修复：单人通话时先使用悬浮窗再切回界面时扬声器按钮状态错误的问题;
3. 修复：多人视频通话时关闭摄像头后，再次邀请他人会自动打开摄像头的问题;

5.1.15

发布日期：2021/12/24

• RTCLib SDK:

1. 增加：加入房间接口支持携带可扩展的用户属性信息，方便客户进行业务信息传递;

2. 增加：在会议和直播场景下，屏幕共享可采集其他应用的音频（系统要求：Android 10+）；
3. 修复：修复在使用 SetAudioQuality 设置音乐模式后华为手机开启耳返，耳返效果不生效问题；

• **CallKit SDK:**

1. 修复：修复 CallKit 在悬浮窗模式下通话，被服务端踢掉不显示通话时长的问题；

5.1.14

发布日期：2021/12/10

• **RTCLib SDK:**

1. 增加：多端加入RTC房间时支持设置互踢策略，可选择策略包括：RCRTCJoinType.KICK(顶掉其他端)、RCRTCJoinType.REFUSE（当前端加入失败）；
2. 优化：优化部分手机通话时声音调整到最大依然小的问题；
3. 修复：增加断线重连(因超时被踢出房间前)后重新同步房间内人员和资源的逻辑；
4. 修复：增加在没有摄像头权限下开启视频预览的错误回调，错误码：OPEN_CAMERA_NO_PERMISSION；
5. 修复：Pixel 手机升级到 Android 12 后某些（例如 360x480、480x480）分辨率下出现绿屏现象；

5.1.13

发布日期：2021/11/29

• **RTCLib SDK:**

1. 增加：混音功能支持左右声道切换，切换声道后同时将数据拷贝到另一个声道，以实现两个声道的数据同步，可用于支持K歌场景下原声与伴唱切换的功能；
2. 修复：修复部分手机机型切换蓝牙耳机失败的问题；

• **CallLib SDK:**

1. 修复：无法在通话接听前的预览中开关摄像头和麦克风的问题；

5.1.12

发布日期：2021/11/12

• **RTCLib SDK:**

1. 增加：音效插件（VoiceBeautifier）支持美声、变声、混响音效功能
 - 美声：低沉、饱满、高亢
 - 变声：假声、绿巨人、小男孩、小女孩、成熟男性、老年男性、老年女性

- 混响：KTV、演唱会

5.1.11

发布日期：2021/11/02

• RTCLib SDK:

1. 增加：支持在直播、会议时播放在线文件 (支持 HTTP、HTTPS、RTMP 和 RTSP 协议的 AVI、MP4、MKV、FLV 格式)，详情请参见「发布自定义流」功能文档。
2. 增加：观众订阅 MCU 合流成功时支持回调通知每个主播的音量。
3. 增加：MCU 合流支持主播设置占位图。
4. 修复：以关闭麦克风的状态退出房间，下次加入房间发布资源后，对端收到的音频资源状态依然是关闭的。
5. 修复：连续调用音频路由模块的 init 和 uninit 方法偶现的空指针问题。

• CallLib SDK:

1. 修复：群组聊天中，当邀请多位群组成员进行音视频通话时，若其中某人拒绝接受邀请，那么再次邀请时，该成员无法正常收到邀请。问题举例：A 邀请 B 和 C 进行音视频通话，但 B 拒绝接听；后续 A 再次邀请 B 时，B 收不到通话邀请。

5.1.10

发布日期：2021/10/15

• RTCLib SDK:

1. 增加：支持双声道模式，包括混音模块（目前支持连线耳机，暂不支持蓝牙耳机和听筒、扬声器）。
2. 增加：远端音频 frame 中增加时间戳。
3. 修复：IM 多端登录互踢时，观众端未主动退出。

• CallKit SDK:

1. 优化：权限申请，去掉不必要的权限申请，并区分音频和视频通话分别申请必要权限。
2. 修复：群组通话邀请人页面点击搜索弹出软键盘，选人后回到通话页面，软键盘未收起。

5.1.9

发布日期：2021/09/28

• RTCLib SDK:

1. 增加了观众/主播角色切换功能。

2. 增加了通话前音频设备检测功能。
3. 增加了自定义视频流视频数据回调接口。
4. 增加了摄像头、麦克风硬件资源抢占处理功能。
5. 优化了Android 10 及以上应用挂后台时弹窗提示，并长响铃(原来是弹出通知栏，通知音)。
6. 优化了耳返功能，开发者直接调用开启耳返的接口即可。
7. 修复了偶现的在线音频混音卡死的问题。

• **CallLib SDK:**

1. 增加了 CallKit 支持自定义设置头像背景功能。

• **CallKit SDK:**

1. 修复了 CallLib 多次收到推送，点击通知栏消息不能唤起呼叫页面的问题。

5.1.8

发布日期：2021/09/10

• **RTCLib SDK:**

1. 增加了 屏幕共享插件。
2. 增加了 弱网下根据带宽动态调整分辨率的能力。
3. 增加了 Stream 和状态报告中的流类型属性。
4. 优化了 猎户座处理器硬编码能力。
5. 修改了 远端视频数据回调的 YUV 数据为 i420 格式。
6. 修复了 推送竖屏的自定义视频时，效果出现旋转的问题。
7. 修复了 主播调用 setAudioQuality 接口设置音质时，内部设置混流码率引起的覆盖主播 MixConfig 问题。

5.1.7

发布日期：2021/08/27

• **RTCLib SDK:**

1. 增加了获取麦克风硬 3A 后音频数据的接口。
2. 增加了获取本端混音后音频数据的接口。
3. 增加了获取远端单路音频数据回调接口。
4. 增加了观众端 IM 重连后主动拉取 KV 的逻辑。
5. 增加了耳机连接、断开事件监听。
6. 优化了 maven 版本 SDK 支持查看 java doc 注释。

5.1.6

发布日期：2021/08/11

• **RTCLib SDK:**

1. 增加了在线音频文件混音功能。
2. 修复了华为耳返不生效的 BUG。
3. 修复了自定义视频编码错误 BUG。

5.1.5

发布日期：2021/07/09

• **RTCLib SDK:**

1. 增加了闪光灯开关，摄像头变焦功能。

• **CDNPlayer Plugin:**

1. 增加了 CDN 播放器插件 (RongRTCPlayer) 功能。

5.1.4

发布日期：2021/07/07

• **FaceBeautifier Plugin:**

1. 增加了美颜模块，包含美白、磨皮、红润、亮度、滤镜三款（浪漫，清新，唯美）。
2. RTCLib SDK、CallLib SDK、CallKit SDK 均可配合美颜模块使用。

5.1.3

发布日期：2021/07/01

• **RTCLib SDK:**

1. 增加了观众可以订阅主播分流。
2. 增加了按照 room id /stream id 进行合流布局选择策略的接口。
3. 重构了混音接口，为每一种混音策略提供状态详细回调和混音进度回调。
4. 修复了自定义加密在某些手机上的崩溃 (HUAWEI AMN-LX9) BUG。
5. 修复了相机旋转角度为 0 时，远端镜像功能不生效 (默认横屏的设备，后置摄像头) BUG。
6. 修复了双声道下开启 opensl es 高频崩溃 (内存越界) BUG。

5.1.2

发布日期：2021/05/21

• **RTCLib SDK:**

1. 修复了订阅状态错误的问题。
2. 完善 clusterID 逻辑，私有云模式下保存上次请求成功的地址。
3. 新增了本地视频编码镜像。
4. 优化了观众加房间逻辑。

• **CallLib SDK:**

1. 增加收到消息的日志埋点。
2. 增加被对端拉黑名单时的情况处理。
3. 修复设置的分辨率等参数在加入房间前生效。

5.1.1

发布日期：2021/04/09

• **RTCLib SDK:**

1. Android 静音模式释放硬件资源。
2. 默认使用双声道。
3. 优化会议场景下的噪音问题。
4. 客户端弱网优化。
5. 修复了小流码率设置超出上限的问题。
6. 修复了视频输入流设置预览窗口跨线程调用可能会导致异常的 Bug。

5.1.0

发布日期：2021/03/05

• **RTCLib SDK:**

1. 增加音频 PCMU 编解码。
2. 增加直播模板下观众加房间接口。
3. 增加小流码率、分辨率、帧率设置功能。
4. 增加通话过程中用户退出的异常处理，并把被踢和异常断开的回调方法放到 EngineListener 中。
5. 修复被踢或是异常断开时，可能出现的回调不执行问题。
6. 修复获取网络方法在高版本系统中不存在引起的崩溃。
7. 优化用户在房间中异常退出或被踢操作使用 IRCRTCEngineEventListener 抛出回调。

• **CallLib SDK:**

1. 修复了 A 邀 B，B 未登录情况下 A 挂断，B 登录后又弹接听界面的问题。

5.0.0

发布日期：2021/01/18

• RTCLib SDK:

1. 视频纹理回调数据结构中增加 matrix。
2. 优化 RCRTCVideoView 切换时的闪烁效果。
3. 优化混音算法。
4. 针对地不同的流支持不同的加密方式。
5. 修复了硬编切换软编过程可能存在的内存泄漏问题。

• CallLib SDK:

1. 邀请/挂断消息，支持设置推送模板 ID，模板 ID 及模板中内容在“控制台-自定义推送文案”中进行创建。设置后根据目标用户通过 RongIMClient 中的 setPushLanguageCode 设置的语言环境，匹配模板中设置的语言内容进行推送，未匹配成功时使用融云默认内容进行推送。

• CallKit SDK:

1. 修复 iQOO 手机收不到厂商推送。
2. 默认使用 AndroidX 架构。

4.1.0 Dev

发布日期：2020/12/17

• RTCLib SDK:

1. 支持跨房间连麦功能。
2. 修复重复调用 setVideoView 引起的黑屏问题。
3. 修复 RTCLib 由 init 状态切换到 uninit 状态时，IM 切换用户后未更新 UserId 的问题。

4.0.3 Stable

发布日期：2020/11/23

• RTCLib SDK:

1. IM SDK 切换 UserID 后，RTCLib 未更新内部 UserID 的问题。
2. 关闭了 jitter buffer 加速。

4.0.4 Dev

发布日期：2020/11/12

• **RTCLib SDK:**

1. 支持厂商（华为、vivo）耳返和低延迟耳返。
2. 增加了设置合流布局背景颜色的接口。
3. 修复了发布资源前操作硬件设备，发布的资源状态不正确的问题。
4. 修复了加入房间成功后拿到的远端用户列表未按顺序排列。
5. 修复了偶现多线程启动相机造成的崩溃。

• **CallLib SDK:**

1. 增加呼叫和被叫的角色信息，使用 inner 数据。
2. 增加了呼叫、挂断音视频时设置厂商通知属性功能。

• **CallKit SDK:**

1. 群组通话时，退出当前通话后，其他人再次邀请当前用户，当前用户无法弹出被叫页面。

4.0.3.2 Dev

发布日期：2020/09/29

• **RTCLib SDK:**

1. IM SDK 切换 UserID 后，RTCLib 未更新内部 UserID 的问题。
2. 关闭了 jitter buffer 加速。

4.0.3.1 Dev

发布日期：2020/09/22

• **RTCLib SDK:**

1. 修复 Bug，增强稳定性。

4.0.3 Dev

发布日期：2020/09/18

• **RTCLib SDK:**

1. 修复当丢包率小于 100 时，音视频北极星上显示不正确的问题。
2. 修复 Android4.4 手机发送自定义音频出现卡顿的问题。
3. 根据用户手机配置判断不支持 HTTP 时转为 HTTPS。

4. 内部逻辑进行优化，提升稳定性。

• **CallKit SDK:**

1. 修复当前在添加成员页面时，音视频断开后，页面应关闭回到会话窗口页面。
2. 修复呼叫多人的情况下，其中有一端未接听，当呼入端挂断后，未接听端没有自动挂断的问题。

4.0.2 Dev

发布日期：2020/08/19

• **RTCLib SDK:**

1. 新增音视频数据支持自定义加密功能。
2. 新增播放音效功能。
3. 优化通话过程中出现硬编解码错误时，切换到软编解码。
4. 修复退出房间后，立即重新 init，再次加入房间本地黑屏问题。

• **CallLib SDK:**

1. 修复通话未接通挂断，相机未释放，导致后续打开相机失败问题。

4.0.1 Dev

发布日期：2020/07/21

• **RTCLib SDK:**

1. 修复了发布或取消发布音频，报资源不存在的问题。
2. 修复了获取本地视频 bitmap 方法，图片内容被放大。
3. 修复了自定义视频重复释放引起的卡死问题。
4. 优化了 HTTPS 逻辑，支持 SNI 模式。

4.0.0.1 Dev

发布日期：2020/06/19

• **RTCLib SDK:**

1. 直播功能支持了 CDN 推流。
2. 直播 MCU 合流接口支持自定义视频流。
3. 增加了动态切换视频分辨率接口 setVideoConfig。
4. SDK 支持了对焦/曝光功能。
5. 自定义视频支持了发布没有音频轨的视频文件。
6. 对 SDK 接口进行了重构，并统一各端返回错误码。

7. 优化了 SDK 与 Media server 的交互逻辑。

3.2.2 Dev

发布日期：2020/05/08

• RTCLib SDK:

1. 支持了分辨率和帧率分别设置，使设置更加灵活。
2. 增加了自定义视频设置帧率、分辨率方法。
3. 实现 Web、iOS、Android 多端码率设置对齐，统一 SDK 内置默认分辨率对应的码率。
4. 视频默认使用 Baseline 编码方式。

3.2.1 Dev

发布日期：2020/04/10

• RTCLib SDK:

1. 优化了 HTTP / HTTPS 切换逻辑。

3.2.0 Dev

发布日期：2020/02/20

• RTCLib SDK:

1. 支持了视频直播功能。
2. 支持了视频局部缩放功能。
3. 修复了麦克风静音功能偶现影响自定义音频发送的问题。

3.1.7 Dev

发布日期：2020/01/10

• RTCLib SDK:

1. 增加了选择断网重连逻辑的方法：支持一直重连和尝试 1 分钟退出两种方式。
2. 修复了部分设备和 iOS 视频通话花屏问题。
3. 修复了反复发布/取消发布视频资源，导致远端听不到声音的问题。
4. 优化了 Media Server 地址选择逻辑，避免因 DNS 解析引起的漂移数据中心。

3.1.6 Dev

发布日期：2019/12/05

• **RTCLib SDK:**

1. 修复 Bug ，增强稳定性。

3.1.5 Dev

发布日期：2019/11/22

• **RTCLib SDK:**

1. 新增了全员禁音方法，关闭所有远端用户的声音。

3.1.4 Dev

发布日期：2019/10/30

• **RTCLib SDK:**

1. 增加了调节混音音量功能接口。
2. 修复了内存泄漏问题，提升了 SDK 稳定性。

3.1.3 Dev

发布日期：2019/09/20

• **RTCLib SDK:**

1. 修复 Bug ，增强稳定性。

3.1.2 Dev

发布日期：2019/08/31

• **RTCLib SDK:**

1. 修复 Bug ，增强稳定性。

3.1.1 Dev

发布日期：2019/08/16

• **RTCLib SDK:**

1. 修复部分情况下引起的崩溃问题。
2. 修复了发布自定义视频情况下，关闭摄像头失效。

3.1.0 Dev

发布日期：2019/08/06

• **RTCLib SDK:**

1. 修复 Bug，增强稳定性。

3.0.8 Dev

发布日期：2019/07/19

• **RTCLib SDK:**

1. 修复 Bug，增强稳定性。

3.0.7 Dev

发布日期：2019/06/28

• **RTCLib SDK:**

1. 修复 Bug，增强稳定性。

3.0.6 Dev

发布日期：2019/06/06

• **RTCLib SDK:**

1. 增加了对 x86 库的支持。
2. 增加了发送纹理视频数据、YUV 视频数据、音频 PCM 数据功能接口。
3. 修复了自定义视频 Tag 解析方式不对的问题。
4. 修复了多人视频通话，房间内有人员进出、视频卡顿问题。

3.0.5 Dev

发布日期：2019/05/17

- **RTCLib SDK:**

1. 修复了金立 GN9800 和 360F4 手机视频通话花屏问题。
2. 修复了多 mediaServer 地址切换失败问题。
3. 修复了房间有人退出偶现的崩溃问题。
4. 去掉了 READ_PHONE_STATE 的权限判断。

3.0.4 Dev

发布日期：2019/05/09

- **RTCLib SDK:**

1. 修复了一些 Bug 增强了稳定性。
2. 添加了对展讯处理器硬编解码的支持。
3. 对中教视通 Z3 板子进行了适配。

3.0.3 Dev

发布日期：2019/04/26

- **RTCLib SDK:**

1. 增加了多 MediaServer 地址动态切换逻辑。
2. 修复了多人视频通话时偶现的崩溃问题。
3. 优化了回声消除的算法。

3.0.2 Dev

发布日期：2019/04/22

- **RTCLib SDK:**

1. 修复 Bug ，增强稳定性。

3.0.1 Dev

发布日期：2019/04/11

- **RTCLib SDK:**

1. 修复 Bug ，增强稳定性。

状态码

更新时间:2024-08-30

状态码	说明
-1	未知错误
0	成功
1	IM 错误
2	HTTP 错误
40001	操作的用户已经不在该房间
40003	房间不存在错误
40016	未开通视频直播服务，参见控制台文档 如何开通音视频服务
40017	未开通音频直播服务，参见控制台文档 如何开通音视频服务
40018	生成 token 失败
40021	用户被封禁，请确认用户当前的状态。
40022	连麦邀请的房间不存在
40023	连麦邀请的人不在房间内
40024	连麦的人正在邀请中
40025	连麦取消邀请，但是邀请信息不存在
40026	应答邀请时，但邀请信息不存在

状态码	说明
40027	应答邀请时，sessionID 不一致
40029	房间已经存在
40030	房间类型不支持
40031	直播身份切换失败
40032	已经加入了房间
40033	(跨App通信场景) 允许互通App或网关地址未配置
50000	初始化失败, IM Server 未连接
50001	调用方法时输入参数错误
50002	加入相同房间错误，表示用户在客户端重复加入相同的房间
50003	调用房间内接口时，不在房间中
50004	未开通音视频服务，参见控制台文档 如何开通音视频服务 
50007	状态异常，
50010	HTTP 请求超时
50011	HTTP 响应错误 (含 500，404，405 等错误)
50012	网络不可用
50020	重复发布资源错误
50021	初步会话协商错误，没有消息的音视频参数
50022	会话协商错误，协商数据不匹配或者其他问题

状态码	说明
50023	发布的资源个数已经到达上限
50024	取消发布不存在的资源
50030	订阅不存在的音视频资源
50031	资源重复订阅
50032	取消订阅不存在的音视频资源
50065	RTC connection is null
50066	发布的 Stream 为空
50067	设置远端 SDP Error
50068	设置本地 SDP Error
50069	解析 JSON 错误
50070	服务端返回的 LiveInfo 为空
50071	PeerConnection 添加 Stream 失败
50073	RTC Token 无效
50074	调用相关接口，没有加入主房间错误
50075	操作的副房间号码和主房间号码一致错误
50076	取消的跨房间连麦请求不存在
50077	响应的跨房间连麦请求不存在
50079	服务端返回的信息异常

状态码	说明
50080	CDN 地址配置数量到达上限（最大为10个）
50100	自动重连出现异常
51000	视频硬编码初始化失败，SDK内部默认切换到软编码
51001	视频硬编码过程中失败，SDK内部默认切换到软编码
51002	视频硬解码初始化失败，SDK内部默认切换到软解码
51003	视频硬解码过程中失败，SDK内部默认切换到软解码
51004	无效的 Camera Id
51005	未找到 Camera Device
51006	打开 Camera 失败
51007	初始化 RCRTCEngine 超时
51100	创建 SDP 中的 Answer 失败
51200	CameraManager 已被释放
51201	操作被取消
51202	AudioManager 已被释放
54001	订阅 CDN 流时未找到 CDN Player 模块
54002	CDN 服务宕机，视频中断，一般是视频源异常或者不支持的视频类型
54008	CDN 数据连接中断或音视频源格式不支持
54009	初始化 CDN Player 模块异常

状态码	说明
54010	获取屏幕共享权限失败
56001	网络探测未开始，不需要调用停止探测方法
56002	网络探测已开始，不允许重复调用开始探测方法
56003	RTC 网络连接通道断开
56004	网络探测被中断，如：调用 joinRoom 或 uninit 等 api 方法都会导致探测被中断结束
56005	当前正在加房间中，不允许调用此操作