实时音视频
通话 CallLib/Kit
Web 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。

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，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。

Web 兼容性

更新时间:2024-08-30

> CallLib SDK 的兼容性同 RTCLib SDK

Web 兼容性说明

Web 端 RTCLib 从 **v5.1.0** 开始支持大部分现代浏览器，但受限于各浏览器厂商对于 **WebRTC** 的支持情况不同，**RTCLib** 在各浏览器上的能力可能会存在差异，**这些差异在 H5 上表现尤为明显**。

已知问题说明

- **iOS 设备下各浏览器要求音视频资源的播放必须在用户操作事件回调中进行（如 click 事件回调中），否则会播放失败。**
- 部分华为 Andrdoid 设备在 Chrome 浏览器中无法正常使用 H.264 编解码能力，故无法支持视频功能。
- Vivo Android 设备内置浏览器不支持 WebRTC 基础能力，故无法使用 RTCLib。
- 小米 Android 设备内置小米浏览器不支持 WebRTC 基础能力，故无法使用 RTCLib。
- 对于在小米 11 设备上、Android 11 系统中通过微信内置浏览器使用 RTCLib 能力，需注意以下事项：
  - 先发布资源再订阅资源，会因为浏览器底层对远端声音资源解码问题导致无法播放远端音频，先订阅再发布则不受影响；
  - 建议在微信内置浏览器和 **WebView** 中仅做订阅资源。
- iOS 平台 QQ 浏览器在刷新页面后二次加入房间时，获取音视频流无法正常弹出授权提醒，因此不建议使用。

浏览器支持清单

	功能	收音频	发音频	收视频	发视频	发视频小流	发屏幕共享	发自定义音视频
平台	浏览器	---	---	---	---	---	---	---
Windows	Chrome	57+	57+	57+	57+	63+	72+	57+
	FireFox	56+	56+	56+	56+	56+	66+	56+
	Edge	79+	79+	79+	79+	79+	79+	79+
	Opera	76+	76+	76+	76+	不支持	76+	不支持
	QQ	10+	10+	10+	10+	10+	不支持	10+
	360	12+	12+	12+	12+	12+	12+	12+
MacOSX	Chrome	57+	57+	57+	57+	63+	72+	57+
	Safari	11+	11+	11+	11+	11+	11+	不支持
	FireFox	56+	56+	56+	56+	56+	66+	56+
	Edge	79+	79+	79+	79+	79+	79+	79+

	功能	收音频	发音频	收视频	发视频	发视频小流	发屏幕共享	发自定义音视频
	Opera	46+	46+	46+	46+	不支持	46+	不支持
	~~QQLite~~	不支持	不支持	不支持	不支持	不支持	不支持	不支持
iOS 14.3+	Safari	支持	支持	支持	支持	不支持	不支持	不支持
	Chrome	支持	支持	支持	支持	不支持	不支持	不支持
	FireFox	支持	支持	支持	支持	不支持	不支持	不支持
	微信内置浏览器	微信 6.5	微信 6.5	微信 6.5	微信 6.5	不支持	不支持	不支持
Android 6.0+	Chrome	90+	90+	90+	90+	不支持	不支持	不支持
	FireFox	87+	87+	87+	87+	不支持	不支持	不支持
	Opera	62+	62+	62+	62+	不支持	不支持	不支持
	WebView	支持	支持	支持	支持	不支持	不支持	不支持
	微信内置浏览器	支持	支持	支持	支持	不支持	不支持	不支持

运行示例项目（Demo）

更新时间:2024-08-30

融云音视频通话 QuickDemo（GitHub · Gitee）演示了融云产品音视频通话在 Web 端的功能，以便开发者体验产品，快速集成，实现单群聊、音视频通话等场景需求。

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

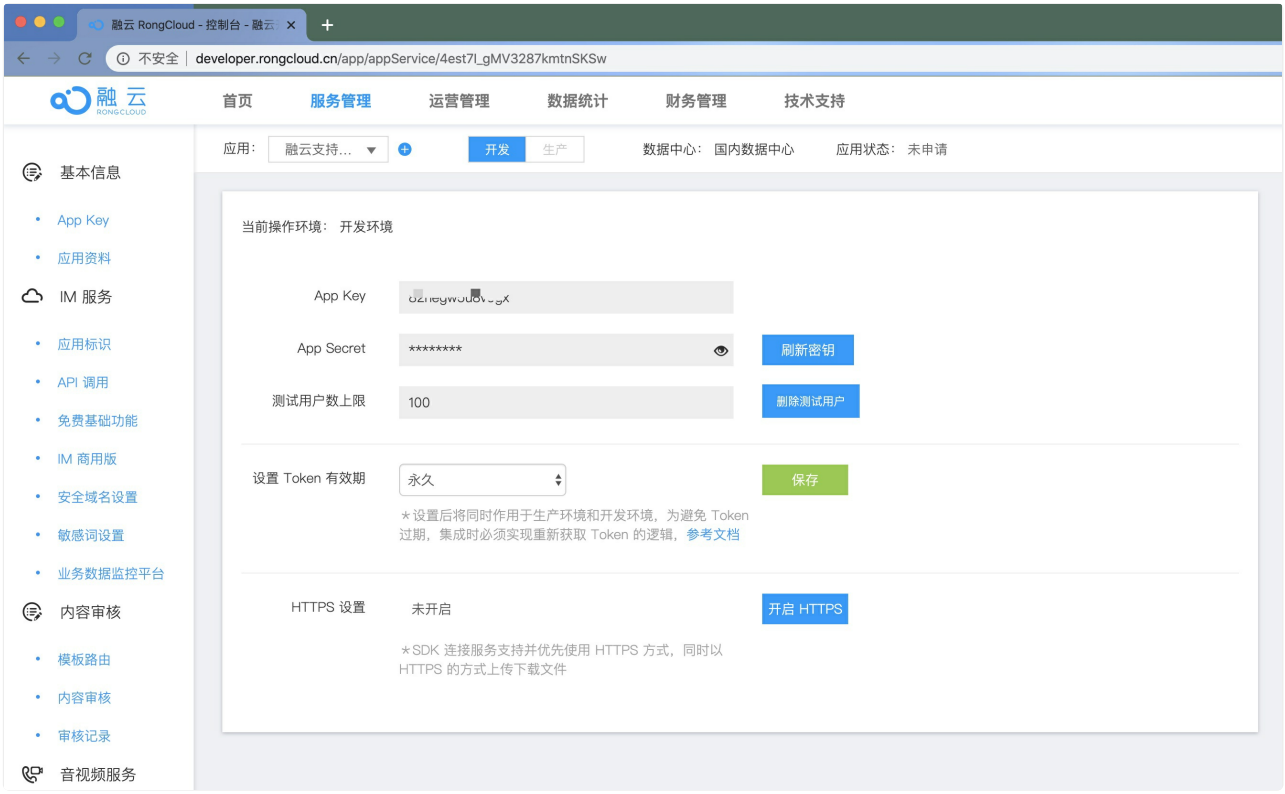
所有支持浏览器请参见Web 兼容性。推荐使用 Google Chrome 最新版体验 QuickDemo。

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。



您需要记录上图所示的应用 App Key，在本教程中使用。

应用的 App Key / Secret 是获取连接融云服务器身份凭证的必要条件。请注意不要泄露。

> App Secret 用于生成数据签名，仅在请求融云服务端 API 接口时使用。本教程中暂不涉及。

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。服务开通后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。

获取用户 Token

用户 Token 是与用户 ID 对应的身份验证令牌，是应用程序的用户在融云的唯一身份标识。应用客户端必须与融云建立 IM 连接，连接时必须传入 Token。

在实际业务运行过程中，应用客户端需要通过应用的服务端调用 IM Server API 申请取得 Token。详见 Server API 文档 注册用户。

在本教程中，为了快速体验和测试 SDK，我们将使用控制台「北极星」开发者工具箱，从 API 调试页面调用 获取 Token 接口，获取到 **userId** 为 1 的用户的 Token。提交后，可在返回正文中取得 Token 字符串。

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"code":200,"userId":"1","token":"gxld6GHx3t1eDxof1qtxxYrQcjkbhl1V@sgyu.cn.example.com;sgyu.cn.example.c
```

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
- 已开通音视频服务免费体验，且已等待 30 分钟
- 已获取用于体验的 Token

1. 克隆下载示例代码。

```
git clone https://github.com/rongcloud/web-quickdemo-calllib-v5.git
```

2. 使用浏览器直接打开 `index.html` 进行操作。

3. 运行成功后，请按照提示输入 App Key，Token即可进入体验。

安装 CallLib SDK

更新时间:2024-08-30

您可以使用 NPM 安装 CallLib 与其依赖的 IMLib 与 RTCLib，或者使用 CDN 方式安装。

实时音视频通话业务依赖即时通讯业务（IMLib）提供的信令通道，同时 CallLib 依赖 RTCLib 提供音视频基础能力，因此基于 CallLib 开发应用需要使用以下几个库：

- 即时通讯基础能力库 IMLib。推荐使用 IMLib 5.X。
- 实时音视频基础能力库 RTCLib 5.X 版本。
- 实时音视频通话能力库 CallLib 5.X 版本。

从 NPM 安装

您需要安装三个 NPM 模块。

如果是新集成融云 SDK 的客户，可略过以下步骤 1、2、3，直接在项目文件夹中打开终端窗口，运行以下命令安装全部模块：

```
npm install @rongcloud/engine@latest @rongcloud/imlib-next --save
npm install @rongcloud/plugin-rtc --save
npm install @rongcloud/plugin-call --save
```

步骤 1：安装 IMLib

兼容 IMLib 2.X、4.X、5.X 版本。IM 业务可与其他版本与平台的 IMLib SDK 互通。

新集成客户推荐使用 IMLib 5.X。如需要选用 IMLib 2.X 或 4.X 版本，推荐使用对应的 Adapter SDK。

- **安装 5.X 版本 IMLib（推荐）**

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

- **或 安装 IMLib 4.X 版本 Adapter SDK**

```
# @rongcloud/imlib-v4 已停止维护，推荐用 RongIMLib-v4-Adapter
# 旧版 imlib-v4 要求版本 ≧ 4.5 +
# npm install @rongcloud/imlib-v4 --save
#
# 安装 RongIMLib-v4-Adapter
npm install @rongcloud/engine@latest @rongcloud/imlib-v4-adapter --save
```

- **或 安装 IMLib 2.X 版本 Adapter SDK**

```
# @rongcloud/imlib-v2 已停止维护，推荐用 RongIMLib-v2-Adapter
# 旧版 imlib-v2 要求版本 ≧ 2.10 +
# npm install @rongcloud/imlib-v2 --save
#
# 安装 RongIMLib-v2-Adapter
npm install @rongcloud/engine@latest @rongcloud/imlib-v2-adapter@latest -S
```

> 提示
>
> 已集成 RongIMLib v3 的客户，必须将 RongIMLib v3 升级到 IMLib 4.X 版本 Adapter SDK 及以上版本。

步骤 2：安装 RTCLib

需要安装 RTCLib 5.2.0 或 5.2.0 以上版本

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

步骤 3：安装 CallLib

```
# 安装 CallLib
npm install @rongcloud/plugin-call --save
```

步骤 4：导入模块

全部下载安装完成后，即可在代码中导入 IMLib、RTCLib 与 CallLib 库。

1. 导入 IMLib 库。请根据您安装的 IMLib 版本导入。

```
//导入 IMLib 5.X
import * as RongIMLib from "@rongcloud/imlib-next";

//或，导入 @rongcloud/imlib-v4-adapter
import { IMClient, init } from "@rongcloud/imlib-v4-adapter";

//或，导入 @rongcloud/imlib-v2-adapter
import { RongIMClient, IMClient } from "@rongcloud/imlib-v2";
```

2. 导入 RTCLib 和 CallLib 库。

```
// 导入 RTCLib、CallLib
import { installer as rtcInstaller, RCRTCClient, RCTrack, RCFrameRate, RCResolution } from
"@rongcloud/plugin-rtc";

import { installer as callInstaller, RCCallClient, RCCallSession, RCCallErrorCode, ISessionListener,
IEndSummary, ISenderInfo, IMuteUser, IInvitedUsers, RCCallLanguage, RCCallEndReason, RCCallMediaType,
IOfflineRecord, RCCallSessionState } from "@rongcloud/plugin-call";
```

从 CDN 安装

CallLib 与其依赖的 IMLib、RTCLib 均支持使用 CDN 文件导入项目。

1. 安装 IMLib（推荐新集成客户使用 IMLib 5.X）。

```
//安装 RongIMLib 5.X（推荐）
<script src="https://cdn.ronghub.com/RongIMLib-5.9.5.prod.js"></script>

//或，安装 RongIMLib-v4-Adapter
<script src="https://cdn.ronghub.com/RongIMLib-v4-Adapter-5.9.5.prod.js"></script>

//或，安装 RongIMLib-v2-Adapter
<script src="https://cdn.ronghub.com/RongIMLib-v2-Adapter-5.9.5.prod.js"></script>
```

2. 安装 RTCLib 和 CallLib。

```
// RTCLib v5
<script src="https://cdn.ronghub.com/RCRTC-5.7.2.prod.js"></script>
// RongCallLib
<script src="https://cdn.ronghub.com/RCCall-5.2.10.prod.js"></script>
```

3. 全部安装完成后，即可在代码中使用 IMLib、RTCLib 与 CallLib 库。各个库的全局变量定义如下：

- IMLib 全局变量： `RongIMLib`
- RTCLib 全局变量： `RCRTC`
- CallLib 全局变量： `RCCall`

初始化客户端时您会用到以上全局变量。在实现音视频通话文档中亦有说明。

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。本文中将详细说明 IM 客户端、RTC 客户端、 CallLib 客户端初始化的方法。

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 IM、RTC、CallLib 客户端

CallLib 可与使用 IMLib 5.X、4.X、2.X 配合使用。IMLib 版本不同时，IM 客户端、RTC 客户端、CallLib 客户端初始化均不相同。请根据您安装的 IMLib 选择以下初始化方式。

CallLib 配合 IMLib 5.X

音视频通话依赖 IMLib 做作为信令通道，因此需要先调用 IMLib 的 init 方法，初始化 IM 客户端，随后调用 IMLib 的 installPlugin 方法初始化 RTC 客户端与 CallLib 客户端。

appkey 即您的融云应用的 App Key。

```
// IM 客户端初始化 (IMLib 5.X)
RongIMLib.init({
appkey: '<your-app-key>',
});

// RTC 客户端初始化
// RTCLib 全局变量定义为 RCRTC，使用 CDN 文件方式集成时，示例如下：
// const rtcClient = RongIMLib.installPlugin(RCRTC.installer, { /* 配置项 */ })
const rtcClient: RCRTCClient = RongIMLib.installPlugin(rtcInstaller, { /* 配置项 */ })

// CallLib 客户端初始化
// CallLib 全局变量定义为 RCCall，使用 CDN 文件集成时，示例如下：
// const caller = RongIMLib.installPlugin(RCCall.installer)
const caller: RCCallClient = RongIMLib.installPlugin(callInstaller, {
// rtcClient 实例 （必填）
rtcClient,
/**
* 被动收到邀请 （收到一个远端发起的新会话），会产生一个新的 session 对象 （必填）
*/
onSession(session: RCCallSession){

/**
* **收到新的 session 后需要立即注册事件监听**
*/
session.registerSessionListener({

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当远端用户同意接听
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当有远端用户挂断
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup(sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession){
```

```
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取
* @param track 本端资源或远端资源, track 不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady(track: RCTrack, session?: RCCallSession){
// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},
});
},

/**
* 以下三条只要满足一条，就会触发onSessionClose
* 1、本端用户自己主动挂断
* 2、服务端把本端用户踢出 RTC 房间
* 3、房间里小于2个人
*
* @param {RCCallSession} session 被结束的 session 对象
* @param summaryInfo 结束一个 session 的后汇总信息
*/
onSessionClose(session: RCCallSession, summaryInfo?: IEndSummary){

},

/**
* 接收 IM 离线期间收到的呼叫记录（按需监听）
*/
onOfflineRecord(record: IOfflineRecord){

},
});
```

CallLib 配合 IMLib 4.X

音视频通话依赖 IMLib 做作为信令通道，因此需要先调用 IMLib 的 init 方法，初始化 IM 客户端，随后调用 IMClient 的 install 方法初始化 RTC 客户端与 CallLib 客户端。

appkey 即您的融云应用的 App Key。

```
// IM 客户端初始化 (IMLib 4.X)
const imClient: IMClient = init({
appkey: 'your_app_key',
```

```
appkey: '<your-app-key>'
});

// RTC 客户端初始化
// RTCLib 全局变量定义为 RCRTC，使用 CDN 文件方式集成时，示例如下：
// const rtcClient = imClient.install(RCRTC.installer)
const rtcClient: RCRTCClient = imClient.install(rtcInstaller)

// CallLib 客户端初始化
// CallLib 全局变量定义为 RCCall，使用 CDN 文件集成时，示例如下：
// const caller = imClient.install(RCCall.installer)
const caller: RCCallClient = imClient.install(callInstaller, {

// rtcClient 实例 （必填）
rtcClient,
/**
* 被动收到邀请 （收到一个远端发起的新会话），会产生一个新的 session 对象 （必填）
*/
onSession(session: RCCallSession){

/**
* **收到新的 session 后需要立即注册事件监听**
*/
session.registerSessionListener({

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当远端用户同意接听
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当有远端用户挂断
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup(sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession){
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取
* @param track 本端资源或远端资源, track 不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady(track: RCTrack, session?: RCCallSession){
// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
```

```
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},
});
},

/**
* 以下三条只要满足一条，就会触发onSessionClose
* 1、本端用户自己主动挂断
* 2、服务端把本端用户踢出 RTC 房间
* 3、房间里小于2个人
*
* @param {RCCallSession} session 被结束的 session 对象
* @param summaryInfo 结束一个 session 的后汇总信息
*/
onSessionClose(session: RCCallSession, summaryInfo?: IEndSummary){

},


/**
* 接收 IM 离线期间收到的呼叫记录（按需监听）
*/
onOfflineRecord(record: IOfflineRecord){

},
});
```

CallLib 配合 IMLib 2.X

音视频通话依赖 IMLib 做作为信令通道，因此需要先调用 IMLib 的 init 方法，初始化 IM 客户端，随后调用 IMClient 的 install 方法初始化 RTC 客户端与 CallLib 客户端。

appkey 即您的融云应用的 App Key。

```
// IM 客户端初始化（IMLib 2.X）
RongIMClient.init('<your-app-key>');
const imClient: IMClient = RongIMClient.getInstance();

// RTC 客户端初始化
// RTCLib 全局变量定义为 RCRTC，使用 CDN 文件方式集成时，示例如下：
// const rtcClient = imClient.install(RCRTC.installer)
const rtcClient: RCRTCClient = imClient.install(rtcInstaller)

// CallLib 客户端初始化
// CallLib 全局变量定义为 RCCall，使用 CDN 文件集成时，示例如下：
// const caller = imClient.install(RCCall.installer)
const caller: RCCallClient = imClient.install(callInstaller, {

// rtcClient 实例 （必填）
rtcClient,
/**
* 被动收到邀请（收到一个远端发起的新会话），会产生一个新的 session 对象（必填）
```

```
* 被动收到邀请（收到一个远端发起的新会话），会产生一个新的 session 对象（必填）
*/
onSession(session: RCCallSession){

/**
* **收到新的 session 后需要立即注册事件监听**
*/
session.registerSessionListener({

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},
/**
* 当远端用户同意接听
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},
/**
* 当有远端用户挂断
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup(sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession){
const { userId } = sender;
},
/**
* 本端资源或远端资源已获取
* @param track 本端资源或远端资源，track 不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady(track: RCTrack, session?: RCCallSession){
// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

});
},

/**
* 以下三条只要满足一条，就会触发onSessionClose
* 1、本端用户自己主动挂断
* 2、服务端把本端用户踢出 RTC 房间
```

```
* 3、房间里小于2个人
*
* @param {RCCallSession} session 被结束的 session 对象
* @param summaryInfo 结束一个 session 的后汇总信息
*/
onSessionClose(session: RCCallSession, summaryInfo?: IEndSummary){

},

/**
* 接收 IM 离线期间收到的呼叫记录（按需监听）
*/
onOfflineRecord(record: IOfflineRecord){

},
});
```

实现音视频通话

更新时间:2024-08-30

在本教程中，您可以体验集成融云 Web 端音视频通话（呼叫）SDK CallLib 的集成流程。 CallLib 支持单人音视频呼叫和基于群组的多人呼叫场景。

示例项目

融云音视频通话 QuickDemo（GitHub · Gitee）演示了融云产品音视频通话在 Web 端的功能，以便开发者体验产品，快速集成，实现单群聊、音视频通话等场景需求。

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。具体实现详见运行示例项目。

前置条件

> 提示
>
> 浏览器页面地址必须为 **https** 协议地址，或使用 **localhost** 域名。

CallLib 概念说明

> 提示
>
> 通过 onSessionClose 监听函数获得通话结束。

步骤 1：开通服务

您在融云创建的应用不会默认启用音视频服务。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。

具体步骤请参阅控制台文档开通音视频服务。

> 提示
>
> 服务开通、关闭等设置完成后 30 分钟后生效。

步骤 2：导入 SDK

您可以使用 NPM 安装 CallLib 与其依赖的 IMLib 与 RTCLib，或者使用 CDN 方式安装。

CallLib 相关业务依赖 IMLib 作为信令通道。因此，开发音视频通话必须安装融云音视频核心能力库 RTCLib，即时通讯能力库

IMLib。您可选用 2.X 或 4.X 或 5.X 版本的 IMLib。

本部分仅针对新集成客户简要说明 **NPM 方式安装方法**。以下步骤中均以 IMLib 5.X 为例说明。

> 提示
>
> CallLib 配合使用 IMLib 2.X，4.X 的详细说明，以及 CDN 方式安装方法，请参阅 安装 CallLib SDK。

1. 安装 5.X 版本 IMLib（推荐新集成客户使用 IMLib 5.X）。

```
# 安装 IMLib v5
npm install @rongcloud/engine@latest @rongcloud/imlib-next --save
```

2. 安装 RTCLib 5.X。

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

3. 安装 CallLib 5.X。

```
# 安装 CallLib
npm install @rongcloud/plugin-call --save
```

4. 全部下载安装完成后，即可在代码中导入 IMLib（5.X）、RTCLib 与 CallLib 库。

```
// 导入 IMLib 5.X
import * as RongIMLib from "@rongcloud/imlib-next";

// 导入 RTCLib
import { installer as rtcInstaller, RCRTCClient, RCTrack, RCFrameRate, RCResolution } from
"@rongcloud/plugin-rtc";

// 导入 CallLib
import { installer as callInstaller, RCCallClient, RCCallSession, RCCallErrorCode, ISessionListener,
IEndSummary, ISenderInfo, IMuteUser, IInvitedUsers, RCCallLanguage, RCCallEndReason, RCCallMediaType,
IOfflineRecord, RCCallSessionState } from "@rongcloud/plugin-call";
```

步骤 3：初始化

本部分仅针对新集成客户简要说明使用 IMLib 5.X 时，IM 客户端、RTC 客户端与 CallLib 客户端的初始化方式。

> 提示
>
> 注意，CallLib 配合使用 IMLib 2.X，4.X 时，IM、RTC 与 CallLib 客户端初始化方式均有不同。详见 初始化。

请依次初始化 IM 客户端、RTC 客户端与 CallLib 客户端。

1. 调用 IMLib 的 `init` 方法，初始化 IM 客户端（IMLib 5.X）。`appkey` 即您的融云应用的 App Key。

```
// IM 客户端初始化 (IMLib 5.X)
RongIMLib.init({
appkey: '<your-app-key>',
});
```

2. 初始化 RTC 客户端与 CallLib 客户端。在初始化 CallLib 客户端实例时，传入 `onSession` 函数监听来电。初始化完成后，可获取 `RCCallClient` 实例对象 `caller`。`RCCallClient` 主要用于发起呼叫。

```
// RTC 客户端初始化
// RTCLib 全局变量定义为 RCRTC，使用 CDN 文件方式集成时，示例如下：
// const rtcClient = RongIMLib.installPlugin(RCRTC.installer, { /* 配置项参数 */ })
const rtcClient: RCRTCClient = RongIMLib.installPlugin(rtcInstaller, { /* 配置项参数 */ });

// CallLib 客户端初始化
// CallLib 全局变量定义为 RCCall，使用 CDN 文件集成时，示例如下：
// const caller = RongIMLib.installPlugin(RCCall.installer)
const caller: RCCallClient = RongIMLib.installPlugin(callInstaller, {
// rtcClient 实例 （必填）
rtcClient,
/**
* 被动收到邀请 （收到一个远端发起的新会话），会产生一个新的 session 对象 （必填）
*/
onSession(session: RCCallSession){

/**
* **收到新的 session 后需要立即注册事件监听**
*/
session.registerSessionListener({

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当远端用户同意接听
* @param sender 远端用户
* @param session 当前的 session 对象
*/
```

```
onAccept(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当有远端用户挂断
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup(sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession){
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取
* @param track 本端资源或远端资源，track 不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady(track: RCTrack, session?: RCCallSession){
// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},
});
},

/**
* 以下三条只要满足一条，就会触发onSessionClose
* 1、本端用户自己主动挂断
* 2、服务端把本端用户踢出 RTC 房间
* 3、房间里小于2个人
*
* @param {RCCallSession} session 被结束的 session 对象
* @param summaryInfo 结束一个 session 的后汇总信息
*/
onSessionClose(session: RCCallSession, summaryInfo?: IEndSummary){

},

/**
* 接收 IM 离线期间收到的呼叫记录（按需监听）
*/
onOfflineRecord(record: IOfflineRecord){

},
});
```

步骤 4：建立 IM 连接

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，因此需要先与 IM 服务建立连接，之后再进行音视频呼叫业务。

```
//与 IM 服务建立连接（IMLib 5.X）
RongIMLib.connect('<user-token>').then(res => {
if (res.code === 0) {
console.log('链接成功，链接用户 id 为：', res.data.userId);
} else {
console.warn('链接失败，code:', res.code)
}
})
```

> 提示
>
> 注意，如 CallLib 配合使用 IMLib 2.X 或 4.X 版本，建立 IM 连接的方式如下：

```
// 与 IM 服务建立连接（IMLib 4.X）
imClient.connect({ token: '<user-token>' })
.then((user) => {
console.log("链接成功，链接用户 id 为：", user.id);

})
.catch((error) => {
console.log("链接失败：", error.code, error.msg);
});


// 与 IM 服务建立连接（IMLib 2.X）
RongIMClient.connect(('<user-token>', {
onSuccess: (userId) => {
console.log('连接成功，用户 ID 为', userId);
},
onTokenIncorrect: function() {
console.log('连接失败，失败原因：token 无效');
},
onError: function(errorCode) {
console.log('连接失败，失败原因：', errorCode);
}
});
```

步骤 5：发起呼叫

主动呼叫分为发起单人通话和发起多人通话，可根据实际需求调用。多人通话的场景必须在一个群组内。

发起单人通话

成功建立 IM 连接后，使用初始化 CallLib 时获取的 `RCCallClient` 实例的 call 方法发起单人通话。

```
/**
* 发起单人通话，如果成功后会产生一个新的session
* @param targetId 被呼叫一方的用户 id 必填
```

```
* @param mediaType 1->音频呼叫 or 2->音视频呼叫 必填
* @param listener session对象上注册的事件 （必填）
* @param constraints 获取音频或音视频资源时的参数 可选
* @param params.channelId 组织 Id 可选
*/
const { code, session } = await caller.call({
targetId: this.targetId,
mediaType,
listener: {

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求 （必填）
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当远端用户同意接听 （必填）
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当有远端用户挂断 （必填）
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup(sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession){
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取 （必填）
* @param track 本端资源或远端资源, track 不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady(track: RCTrack, session?: RCCallSession){

// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
'video' + user.userId
) as HTMLVideoElement;
track.play(video);
}
},
},
});
```

```
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

发起多人通话

成功建立 IM 连接后，使用初始化 CallLib 时获取的 RCCallClient 实例的 callInGroup 发起多人通话。多人通话必须在同一群组中。

```
/**
* 发起多人通话， 如果成功后会产生一个新的session
* @param targetId 群组 Id （必填）
* @param userIds 被呼叫的群内成员 Id （必填）
* @param mediaType 0->音频呼叫 or 2->音视频呼叫 （必填）
* @param listener session对象上注册的事件 （必填）
* @param constraints 获取音频或音视频资源时的参数 （可选）
* @param channelId 组织 Id （可选）
*/
const { code, session } = await caller.callInGroup({
targetId: this.targetId,
userIds,
mediaType,
listener: {

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求 （必填）
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当远端用户同意接听 （必填）
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept(sender: ISenderInfo, session: RCCallSession){
const { userId } = sender;
},

/**
* 当有远端用户挂断 （必填）
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup(sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession){
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取 （必填）
* @param track 本端资源或远端资源, track 不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady(track: RCTrack, session?: RCCallSession){
// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
```

```
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

/**
* 群组通话中有其他人被邀请加入 （必填）
* @param sender 发送者
* @param invitedUsers 被邀请的用户列表
* @param session 当前的 session 对象
*/
onMemberModify: (sender: ISenderInfo, invitedUsers: IInvitedUsers[], session: RCCallSession) => {

}
},
});
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

步骤 6：接听

调用 accept 接通电话。如果通话类型为音视频通话，接通前可设置视频参数。接通之后，可将音视频通话转为纯音频通话，支持通话中切换音频输入设备。

通话媒体类型（纯音频或音视频）由发起者决定。

```
const { code } = await session.accept();
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

步骤 7：挂断

调用 hungup 挂断（或拒绝）通话。SDK 内部会自动告知对方挂断（或拒绝）的原因。

```
const { code } = await session.hungup();
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

主叫方 设置呼叫、挂断推送信息

更新时间:2024-08-30

主叫端在呼叫前调用

```
/**
* 设置呼叫、挂断推送数据
* @param callPushConfig 呼叫推送配置
* @param callPushConfig.pushTitle 呼叫推送标题
* @param callPushConfig.pushContent 呼叫推送内容
* @param hungupPushConfig 挂断推送配置
* @param hungupPushConfig.pushTitle 挂断推送标题
* @param hungupPushConfig.pushContent 挂断推送内容
*/
caller.setPushConfig(callPushConfig, hungupPushConfig);
```

发起呼叫

发起单人通话

完成初始化后，调用 `caller.call` 方法发起单人音视频通话。

API 参考：call

- 参数说明:

参数	类型	必填	说明
targetId	string	是	对方的userId
mediaType	`enum RCCallMediaType { number }`	是	发起的通话媒体类型。1 为音频呼叫；2 为 音视频呼叫
listener	`ISessionListener`	是	在 `session` 对象上注册的事件监听
constraints	`{ video?: ICameraVideoProfile, audio?: IMicphoneAudioProfile }`	否	音频或音视频资源的约束 video 为可选参数详见设置视频参数， audio为可选参数详见设置音频参数
extra	string	否	消息的扩展信息
pushTitle	string	否	通知的标题
pushContent	string	否	通知的内容

- 返回值:

返回值	返回类型	说明
code	RCCallErrorCode	10000表示成功，其余的表示失败，详见状态码
session	RCCallSession	session 对象

- 示例代码:

```
const { code, session } = await caller.call({
targetId: this.targetId,
mediaType,
listener: {

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求 （必填）
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 当远端用户同意接听 （必填）
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;

},

/**
* 当有远端用户挂断 （必填）
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup: (sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取， （必填）
* @param track 本端资源或远端资源, track不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady: (track: RCTrack, session?: RCCallSession) => {

// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
```

```
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

/**
* 群组通话中有其他人被邀请加入 （必填）
* @param sender 发送者
* @param invitedUsers 被邀请的用户列表
* @param session 当前的 session 对象
*/
onMemberModify: (sender: ISenderInfo, invitedUsers: IInvitedUsers[], session: RCCallSession) => {
},

/**
* 通话类型改变时触发，例如： 从视频变成音频通话 （必填）
* @param sender 发送者
* @param mediaType 1为音频通话 2为视频通话
* @param session 当前的 session 对象
*/
onMediaModify: (sender: ISenderInfo, mediaType: RCCallMediaType, session: RCCallSession) => {

},

/**
* 对方静音后触发 （必填）
* @param muteUser 已静音的用户
* @param session 当前的 session 对象
*/
onAudioMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

/**
* 对方禁用视频后触发 （必填）
* @param muteUser 已禁用视频的用户
* @param session 当前的 session 对象
*/
onVideoMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

},
});
if (code === RCCallErrorCode.SUCCESS) {

// do something
}
```

发起多人通话

完成初始化后，调用 `caller.callInGroup` 方法发起多人音视频通话。多人通话场景所有通话者必须在一个群组内。

API 参考：callInGroup

- 参数说明：

参数	类型	必填	说明
targetId	string	是	群组 Id
userIds	`Array<string>`	是	被呼叫的群内成员 Id
mediaType	`enum RCCallMediaType { number }`	是	发起的通话媒体类型 1->音频呼叫 or 2->音视频呼叫
listener	`ISessionListener`	是	在 session 对象上注册的事件监听
constraints	`{ video?: ICameraVideoProfile, audio?: IMicphoneAudioProfile }`	否	音频或音视频资源的约束 video 为可选参数详见设置视频参数， audio为可选参数详见设置音频参数
extra	string	否	消息的扩展信息
pushTitle	string	否	通知的标题
pushContent	string	否	通知的内容

- 返回值：

返回值	返回类型	说明
code	`RCCallErrorCode`	10000表示成功，其余的表示失败，详见状态码
session	`RCCallSession`	session对象

- 示例代码：

```
const { code, session } = await caller.callInGroup({
targetId: this.targetId,
userIds,
mediaType,
listener: {

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求 （必填）
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 当远端用户同意接听 （必填）
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;

},

/**
* 当有远端用户挂断 （必填）
```

```
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup: (sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取 （必填）
* @param track 本端资源或远端资源, track不可设置成 Vue 组件的响应式数据
* @param session 当前的 session 对象
*/
onTrackReady: (track: RCTrack, session?: RCCallSession) => {

// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id


// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

/**
* 群组通话中有其他人被邀请加入（必填）
* @param sender 发送者
* @param invitedUsers 被邀请的用户列表
* @param session 当前的 session 对象
*/
onMemberModify: (sender: ISenderInfo, invitedUsers: IInvitedUsers[], session: RCCallSession) => {
},

/**
* 通话类型改变时触发，例如： 从视频变成音频通话 （必填）
* @param sender 发送者
* @param mediaType 1为音频通话 2为视频通话
* @param session 当前的 session 对象
*/
onMediaModify: (sender: ISenderInfo, mediaType: RCCallMediaType, session: RCCallSession) => {

},

/**
* 对方静音后触发 （必填）
* @param muteUser 已静音的用户
* @param session 当前的 session 对象
*/
onAudioMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

/**
* 对方禁用视频后触发 （必填）
* @param muteUser 已禁用视频的用户
* @param session 当前的 session 对象
```

```
*/
onVideoMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

},
});
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

挂断通话

调用 session.hungup 方法挂断通话，SDK 内部会自动告知对方挂断、拒绝原因。

API 参考：hungup

- 返回值：

返回值	返回类型	说明
code	number	10000表示成功，其余的表示失败，详见状态码

- 示例代码：

```
const { code } = await session.hungup();
// do something
```

邀请通话

调用 session.invite(userIds) 方法邀请用户加入当前通话（仅限群组）。

API 参考：invite

- 参数说明：

参数	类型	必填	说明
userIds	Array<string>	是	邀请的用户 ID 列表
options	object	否	扩展字段

options 说明：

参数	类型	必填	说明
extra	string	否	消息的扩展信息
pushTitle	string	否	通知的标题
pushContent	string	否	通知的内容

- 返回值 ：

返回值	返回类型	说明
code	number	10000表示成功，其余的表示失败，详见状态码

- 示例代码：

```
const { code } = await session.invite(userIds);
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

被叫方 设置呼叫、挂断推送信息

更新时间:2024-08-30

被叫端在挂断前调用

```
/**
* 设置呼叫、挂断推送数据
* @param callPushConfig 呼叫推送配置
* @param callPushConfig.pushTitle 呼叫推送标题
* @param callPushConfig.pushContent 呼叫推送内容
* @param hungupPushConfig 挂断推送配置
* @param hungupPushConfig.pushTitle 挂断推送标题
* @param hungupPushConfig.pushContent 挂断推送内容
*/
caller.setPushConfig(callPushConfig, hungupPushConfig);
```

监听来电

在初始化 CallLib 客户端实例时，传入 onSession 函数监听来电。

从 onSession 的参数可以取到来电通话的 session 实例，需要注册通话过程中的事件监听。详见通话事件说明。

extra 为远端发起呼叫或邀请时传入的字符串类型的消息扩展信息。

示例代码：

```
/**
* calllib 客户端初始化
*/
const caller: RCCallClient = RongIMLib.installPlugin(callInstaller, {

/**
* 监听来电，会产生一个新的 session 对象 （必填）
*/
onSession: (session: RCCallSession, extra?: string) => {
/**
* 通话的 session 实例，需要注册通话过程中的事件监听
* session.registerSessionListener({...})
*/
},

......

})
```

接听通话

使用 session 对象的 accept 方法来接听。

API 参考：accept

- 返回值

返回值	返回类型	说明
code	enum RCCallErrorCode { number }	10000表示成功，其余的表示失败，详见状态码

- 示例代码：

```
const { code } = await session.accept();
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

拒绝/挂断通话

调用 session.hungup 方法挂断通话，SDK 内部会自动告知对方挂断、拒绝原因。

API 参考：hungup

- 返回值：

返回值	返回类型	说明
code	enum RCCallErrorCode { number }	10000表示成功，其余的表示失败，详见状态码

- 示例代码：

```
const { code } = await session.hungup();
// do something
```

通话事件说明

更新时间:2024-08-30

有 3 种情况需要注册通话事件监听：

- 主动发起单人通话时，给 `caller.call` 方法传入 `listener` 参数。
- 主动发起多人通话时，给 `caller.callInGroup` 方法传入 `listener` 参数。
- 触发来电监听后，从 `onSession` 函数的参数里拿到 `session` 实例后注册。

主动发起呼叫时注册

发起单人通话

`caller.call` 方法的 `listener` 参数对应要监听的事件，CallLib SDK 内部会把这些事件监听注册到 `caller.call` 方法产生的 `session` 实例上。

API 参考：call

```
/**
* 单呼，发起呼叫，如果成功后会产生一个新的session
* @param targetId 被呼叫一方的用户 id（必填）
* @param mediaType 1->音频呼叫 or 2->音视频呼叫（必填）
* @param listener session对象上注册的事件，主动发起 call 成功后会产生一个新的session，这个 session 上要注册的事件（必填）
* @param constraints 获取音频或音视频资源时的参数 可选
* @param params.channelId 组织 Id 可选
*/
const { code, session } = await caller.call({
targetId: this.targetId,
mediaType,
listener: {

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求 （必填）
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 当远端用户同意接听 （必填）
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;

},
```

```
/**
* 当有远端用户挂断 （必填）
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup: (sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取，track为本地音频或音视频, track不可设置成 Vue 组件的响应式数据 （必填）
* @param track 本端资源或远端资源
* @param session 当前的 session 对象
*/
onTrackReady: (track: RCTrack, session?: RCCallSession) => {

// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

/**
* 群组通话中有其他人被邀请加入
* @param sender 发送者 （必填）
* @param invitedUsers 被邀请的用户列表
* @param session 当前的 session 对象
*/
onMemberModify: (sender: ISenderInfo, invitedUsers: IInvitedUsers[], session: RCCallSession) => {
},

/**
* 通话类型改变时触发，例如： 从视频变成音频通话， （必填）
* @param sender 发送者
* @param mediaType 1为音频通话 2为视频通话
* @param session 当前的 session 对象
*/
onMediaModify: (sender: ISenderInfo, mediaType: RCCallMediaType, session: RCCallSession) => {
},

/**
* 对方静音后触发 （必填）
* @param muteUser 已静音的用户
* @param session 当前的 session 对象
*/
onAudioMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

/**
* 对方禁用视频后触发 （必填）
* @param muteUser 已禁用视频的用户
* @param session 当前的 session 对象
```

```
*/
onVideoMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

},
});
if (code === RCCallErrorCode.SUCCESS) {

// do something
}
```

发起多人通话

caller.callInGroup 方法的 listener 参数对应要监听的事件，CallLib SDK 内部会把这些事件监听注册到 caller.callInGroup 方法产生的 session 实例上。

API 参考：callInGroup

```
/**
* 发起群组呼叫， 如果成功后会产生一个新的session
* @param targetId 群组 Id （必填）
* @param userIds 被呼叫的群内成员 Id （必填）
* @param mediaType 0->音频呼叫 or 2->音视频呼叫 （必填）
* @param listener 新 session 对象上注册的事件, 主动发起 call 成功后会产生一个新的session, 这个 session 上要注册的事件 （必填）
* @param constraints 获取音频或音视频资源时的参数 （可选）
* @param channelId 组织 Id （可选）
*/
const { code, session } = await caller.callInGroup({
targetId: this.targetId,
userIds,
mediaType,
listener: {

/**
* 当远端用户已开始响铃，表示对方已收到呼叫请求 （必填）
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 当远端用户同意接听 （必填）
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;

},

/**
* 当有远端用户挂断 （必填）
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup: (sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession) => {
```

```
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取，track为本地音频或音视频, track不可设置成 Vue 组件的响应式数据 （必填）
* @param track 本端资源或远端资源
* @param session 当前的 session 对象
*/
onTrackReady: (track: RCTrack, session?: RCCallSession) => {
// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

/**
* 群组通话中有其他人被邀请加入
* @param sender 发送者 （必填）
* @param invitedUsers 被邀请的用户列表
* @param session 当前的 session 对象
*/
onMemberModify: (sender: ISenderInfo, invitedUsers: IInvitedUsers[], session: RCCallSession) => {
},

/**
* 通话类型改变时触发，例如： 从视频变成音频通话， （必填）
* @param sender 发送者
* @param mediaType 1为音频通话 2为视频通话
* @param session 当前的 session 对象
*/
onMediaModify: (sender: ISenderInfo, mediaType: RCCallMediaType, session: RCCallSession) => {

},

/**
* 对方静音后触发 （必填）
* @param muteUser 已静音的用户
* @param session 当前的 session 对象
*/
onAudioMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

/**
* 对方禁用视频后触发 （必填）
* @param muteUser 已禁用视频的用户
* @param session 当前的 session 对象
*/
onVideoMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

},
});
if (code === RCCallErrorCode.SUCCESS) {
```

```
// do something
}
```

接到通话邀请时注册

触发来电监听后，从 onSession 的参数可以取到来电通话的 session 实例，需要注册通话过程中的事件监听。

API 参考：registerSessionListener

参数说明：

参数	类型	必填	说明
onRinging	function	是	当远端用户已开始响铃，该函数有 2 个参数：sender 是发送者，session 是通话实例。这时用户可以在业务层做响铃的 UI 展示。
onAccept	function	是	当远端用户已同意接听，该函数有 2 个参数：sender 是发送者，session 是通话实例。这时用户可以把 UI 的‘响铃’变成‘通话中’。
onHungup	function	是	当有远端用户挂断，该函数有 3 个参数：sender 是发送者，reason 是挂断原因，session 是通话实例。这时用户可以在 UI 层提示‘xxx已挂断’。
onTrackReady	function	是	当本端资源或远端资源已获取，该函数有 2 个参数：track 是本端资源或远端资源，session 是通话实例。这时用户可以用拿到的 track 播放音频、视频。
onMemberModify	function	是	群组通话中有其他人被邀请加入，该函数有 3 个参数：sender 是发送者，invitedUsers 是被邀请的用户列表，session 是通话实例。这时用户可以在 UI 层提示‘xxx加入通话’。
onMediaModify	function	是	通话类型改变时触发，该函数有3个参数：sender 是发送者，mediaType 通话类型，session 是通话实例。这时用户可以在 UI 层提示‘已降级成音频通话’。
onAudioMuteChange	function	是	对方静音后触发，该函数有2个参数：muteUser 是已静音的用户，session 是通话实例。这时用户可以在 UI 层提示‘xxx已静音’。
onVideoMuteChange	function	是	对方禁用视频后触发，该函数有2个参数：muteUser 是已禁用视频的用户，session 是通话实例。这时用户可以在 UI 层提示‘xxx已禁用视频’。

以下是事件监听函数的参数说明：

- sender 参数说明：

参数	类型	必填	说明
sender.userId	string	是	发送者 userId
sender.name	string	否	用户名
sender.portraitUri	string	否	用户头像地址
sender.extra	string	否	用户信息中的附加信息

- invitedUsers 参数类型为 Array<IInvitedUsers>。IInvitedUsers 的参数说明同 sender。
- muteUser 参数说明：

参数	类型	必填	说明
muteUser.userId	string	是	发送者 userId
muteUser.muted	boolean	是	资是否禁用
muteUser.kind	string	否	被禁用的资源类型
muteUser.trackId	string	否	被禁用的资源ID

- reason 参数详见挂断通话的原因

示例代码：

```
/**
* 在 session 实例上注册事件监听
*/
session.registerSessionListener({

/**
* 当远端用户已开始响铃
* @param sender 已响铃的用户
* @param session 当前的 session 对象
*/
onRinging: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 当远端用户已同意接听
* @param sender 远端用户
* @param session 当前的 session 对象
*/
onAccept: (sender: ISenderInfo, session: RCCallSession) => {
const { userId } = sender;

},

/**
* 当有远端用户挂断
* @param sender 远端用户
* @param reason 挂断的原因
* @param session 当前的 session 对象
*/
onHungup: (sender: ISenderInfo, reason: RCCallEndReason, session: RCCallSession) => {
const { userId } = sender;
},

/**
* 本端资源或远端资源已获取，track为本地音频或音视频, track不可设置成 Vue 组件的响应式数据
* @param track 本端资源或远端资源
* @param session 当前的 session 对象
*/
onTrackReady: (track: RCTrack, session?: RCCallSession) => {

// track.isLocalTrack() 是否为本地资源
// track.isAudioTrack() 是否为音频
// track.isVideoTrack() 是否为视频
// track.getUserId() 产生该 track 的用户id

// 播放音频。如果为远端音频，建议直接播放。如为本端音频，建议不播放，以减少回音。
if (track.isAudioTrack() && !track.isLocalTrack()) {
```

```
if (track.isAudioTrack() && !track.isLocalTrack()) {
track.play();
}

// 视频在对应的容器里播放
if (track.isVideoTrack()) {
const video = document.getElementById(
"video" + user.userId
) as HTMLVideoElement;
track.play(video);
}
},

/**
* 群组通话中有其他人被邀请加入
* @param sender 发送者
* @param invitedUsers 被邀请的用户列表
* @param session 当前的 session 对象
*/
onMemberModify: (sender: ISenderInfo, invitedUsers: IInvitedUsers[], session: RCCallSession) => {
},

/**
* 通话类型改变时触发，例如： 从视频变成音频通话，
* @param sender 发送者
* @param mediaType 1为音频通话 2为视频通话
* @param session 当前的 session 对象
*/
onMediaModify: (sender: ISenderInfo, mediaType: RCCallMediaType, session: RCCallSession) => {

},

/**
* 对方静音后触发
* @param muteUser 已静音的用户
* @param session 当前的 session 对象
*/
onAudioMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

/**
* 对方禁用视频后触发
* @param muteUser 已禁用视频的用户
* @param session 当前的 session 对象
*/
onVideoMuteChange: (muteUser: IMuteUser, session: RCCallSession) => {
},

});
```

监听通话结束

需要在初始化 CallLib 客户端实例时，传入 onSessionClose 函数监听通话结束，从 onSessionClose 的参数里可以取到以下：

1. 结束通话的 session 实例
2. 通话结束后的汇总信息 (summaryInfo)

API 参考：install

summaryInfo 参数字段说明:

参数	类型	必填	说明
conversationType	enum ConversationType { number }	是	通话类型 1 单聊 3 群组
channelId	string	是	组织 ID
targetId	string	是	目标 ID
mediaType	enum RCCallMediaType { number }	是	通话媒体类型。 1 为音频通话 2 为视频通话
beginTimestamp	number	是	通话开始时间戳
beginTimestamp	number	是	通话开始时间戳
endTimestamp	number	是	通话结束时间戳
duration	number	是	通话时长 单位：ms
endReason	enum RCCallEndReason { number }	是	通话结束原因

示例代码：

```
/**
* calllib 客户端初始化
*/
const caller: RCCallClient = RongIMLib.installPlugin(callInstaller, {
/**
* 监听通话结束
* 以下三条只要满足一条，就会触发onSessionClose
* 1、本端用户自己主动挂断
* 2、服务端把本端用户踢出 RTC 房间
* 3、房间里小于2个人
*
* @param {RCCallSession} session 被结束的 session 对象
* @param summaryInfo 结束一个 session 的后汇总信息
*/
onSessionClose: (session: RCCallSession, summaryInfo?: IEndSummary) => {

},

......

})
```

通话信息

更新时间:2024-08-30

SDK 通过 RCCallSession 标识一个通话。主叫、被叫在通话过程中都会支持一个 RCCallSession 实例。

- 主叫方成功发起通话时，会返回 `RCCallSession` 实例。
- 被叫方需要在初始化 CallLib 客户端实例时，传入 `onSession` 函数监听来电，通过 `onSession` 的回调参数取到来电通话的 `RCCallSession` 实例。

获得通话唯一标识

API 参考：getSessionId

```
/**
* 返回 string 类型
*/
session.getSessionId()
```

获取人员状态

API 参考：getUserState

```
/**
* 返回 RCCallUserState 枚举值 0表示用户不存在于通话中 1表示等待接听 2表示通话中
* @parm {string} userId
*/
session.getUserState(userId)
```

获得 session 的状态

API 参考：getState

```
/**
* 返回枚举值, 表示 session 的当前状态 0 等待建立连接 1 会话维持中 2会话已结束
*/
session.getState()
```

获取房间当前会话Id

API 参考：getRTCSessionId

```
/**
* 返回 string 类型
*/
session.getRTCSessionId()
```

获取目标 ID

单呼时为对方人员 ID，群呼时为群组 ID。

API 参考：getTargetId

```
/**
* 返回 string 类型
*/
session.getTargetId()
```

获取会话类型

API 参考：getConversationType

```
/**
* 返回 number 类型 1表示单聊 3表示群组
*/
session.getConversationType()
```

获取组织 ID

API 参考：getChannelId

```
/**
* 返回 string 类型
*/
session.getChannelId()
```

获取房间人员列表，不包含本端信息

API 参考：getRemoteUsers

```
/**
* 返回Array<IUserData>类型
*/
session.getRemoteUsers()
```

挂断通话的原因

更新时间:2024-08-30

枚举值	说明
1	己方取消已发出的通话请求
2	己方拒绝收到的通话请求
3	己方挂断
4	己方忙碌
5	己方未接听
6	己方不支持当前音视频引擎
7	己方网络错误
8	己方摄像头资源获取失败，可能是权限原因
9	己方资源发布失败
10	己方订阅资源失败
11	对方取消发出的通话请求
12	对方拒绝收到的通话请求
13	通话过程中对方挂断
14	对方忙碌
15	对方未接听
16	对方引擎不支持
17	对方网络错误
18	对方摄像头资源获取失败，可能是权限原因
19	远端资源发布失败
20	远端订阅资源失败
21	己方其他端已加入新通话
22	己方其他端已在通话中
23	己方被禁止通话
24	己端接听系统通话（移动端接听系统来电）
31	远端其他端已加入新通话
32	远端其他端已在通话中
33	远端被禁止通话
34	远端接听系统通话（移动端接听系统来电）
101	其他端接听
102	其他端挂断
103	己方被对方加入黑名单

枚举值	说明
104	音视频服务未开通

更换输入设备

更新时间:2024-08-30

> 暂不支持通话中更改视频摄像头

通话中更换音频输入设备

`session.changeAudioDevice(audioConstraints)`

`audioConstraints` 是可选参数，不传参数时取默认设备。

API 参考：changeAudioDevice

- 参数说明:

参数	类型	必填	说明
audioConstraints.micphoneId	sting	否	音频输入设备的ID
audioConstraints.sampleRate	number	否	采样率

- 代码示例：

```
function addDevicesChangeListener(session: RCCallSession){
navigator.mediaDevices.ondevicechange = null;

// 添加设备变动的监听
navigator.mediaDevices.ondevicechange = () => {

// 取到变动后的设备
navigator.mediaDevices.enumerateDevices().then((devices) => {

// 这里用默认的音频输入设备举例
session.changeAudioDevice();
});
}
};

// 在获得 session 后执行
addDevicesChangeListener(session)
```

视频管理 设置视频参数

更新时间:2024-08-30

参数	类型	必填	说明
cameraId	string	否	摄像头的 `deviceId`
frameRate	`enum RCFrameRate { string }`	是	帧率。默认值 'FPS_15'，可取值：'FPS_10'、'FPS_15'、'FPS_24'、'FPS_30'
resolution	`enum RCResolution { string }`	是	分辨率。默认值 'W640_H480'，可取值：'W176_H132'、'W176_H144'、'W256_H144'、'W320_H180'、'W240_H240'、'W320_H240'、'W480_H360'、'W640_H360'、'W480_H480'、'W640_H480'、'W720_H480'、'W1280_H720'、'W1920_H1080'

发起单人通话时设置

`constraints.video`

API 参考：call

```
const { code, session } = await caller.call({
targetId: this.targetId,
mediaType,
listener: {
......
},

constraints: {
// 视频参数设置
video: {

// 摄像头的deviceId, 以“xxxx”举例
cameraId: "xxxx",

// 默认帧率为 15
frameRate: RCFrameRate.FPS_15,

// 默认分辨率为 640 * 480
resolution: RCResolution.W640_H480
}

}

});
```

发起多人通话时设置

`constraints.video`

API 参考：callInGroup

```
const { code, session } = await caller.callInGroup({
targetId,
mediaType,
userIds,
listener: {
......
},

constraints: {
// 视频参数设置
video: {

// 摄像头的deviceId，以“xxxx”举例
cameraId: "xxxx",

// 默认帧率为 15
frameRate: RCFrameRate.FPS_15,

// 默认分辨率为 640 * 480
resolution: RCResolution.W640_H480
}

}

});
```

接听时设置

video

API 参考：accept

```
const { code } = await session.accept({
// 视频参数设置
video: {

// 摄像头的deviceId，以“xxxx”举例
cameraId: "xxxx",

// 默认帧率为 15
frameRate: RCFrameRate.FPS_15,

// 默认分辨率为 640 * 480
resolution: RCResolution.W640_H480
}

});
```

降级通话

降级通话是指从视频通话转为音频通话，目前仅支持视频往音频单向转换。

API 参考：descendAbility

```
/**
* 返回Promise<{code: RCCallErrorCode}>
*/
const { code } = await session.descendAbility()
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

禁用本地视频

API 参考：disableVideoTrack

```
/**
* 返回Promise<{code: RCCallErrorCode}>
*/
const { code } =await session.disableVideoTrack()
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

启用本地视频

API 参考：enableVideoTrack

```
/**
* 返回Promise<{code: RCCallErrorCode}>
*/
const { code } = await session.enableVideoTrack()
if (code === RCCallErrorCode.SUCCESS) {
// do something
}
```

音频管理 设置音频参数

更新时间:2024-08-30

参数	类型	必填	说明
micphoneId	string	否	麦克风的 deviceId
sampleRate	number	否	采样率

发起单人通话时设置

constraints.audio

API 参考：call

```
const { code, session } = await caller.call({
targetId: this.targetId,
mediaType,
listener: {
......
},

constraints: {
// 音频参数设置
audio: {

// 麦克风的deviceId, 这里以“xxxx”举例
micphoneId: 'xxxx',

// 采样率
sampleRate: 48,
}

}

});
```

发起多人通话时设置

constraints.audio

API 参考：callInGroup

```
const { code, session } = await caller.callInGroup({
targetId,
mediaType,
userIds,
listener: {
......
},

constraints: {
// 音频参数设置
audio: {

// 麦克风的deviceId, 这里以“xxxx”举例
micphoneId: 'xxxx',

// 采样率
sampleRate: 48,
}

}

});
```

接听时设置

audio

API 参考：accept

```
const { code } = await session.accept({
// 音频参数设置
audio: {

// 麦克风的deviceId, 这里以“xxxx”举例
micphoneId: 'xxxx',

// 采样率
sampleRate: 48,
}

});
```

禁用本地音频

API 参考：disableAudioTrack

```
await session.disableAudioTrack()
```

启用本地音频

API 参考：enableAudioTrack

```
await session.enableAudioTrack()
```

通话数据统计 质量数据监控

更新时间:2024-08-30

获取质量数据

```
import { IRCRTCStateReport, IRCCandidatePairStat, IRCTrackStat } from '@rongcloud/plugin-rtc'
/**
* 在 session 实例上注册事件监听
*/
session.registerSessionListener({
/**
* 用于接收状态数据报告
* @param report 报告信息
* @param session 当前的 session 对象
*/
onRTCStateReport: (report: IRCRTCStateReport, session: RCCallSession) => {
/**
* 报告生成时间
*/
const timestamp: number = report.timestamp
/**
* 对等连接状态数据，其中包含反应当前与媒体服务器之间的 UDP 通道质量的信息
*/
const iceCandidatePair: IRCCandidatePairStat = report.iceCandidatePair
/**
* 所有的上行流的状态数据
*/
const senders: IRCTrackStat[] = report.senders
/**
* 订阅的下行流状态数据
*/
const receivers: IRCTrackStat[] = report.receivers
}
});
```

连接状态属性

```
interface IRCCandidatePairStat {
/**
* 本端 IP
*/
IP: string;
/**
* 本地 UDP 端口
*/
port: number;
/**
* 本地网络类型
*/
networkType: string | null;
/**
* 远端 IP
*/
remoteIP: string;
/**
* 远端 UDP 端口
*/
remotePort: number;
/**
* 协议
*/
protocol: string;
/**
* 发送总码率，单位 kbps
*/
bitrateSend: number;
/**
* 接收总码率，单位 kbps
*/
bitrateRecv: number;
/**
*  (Round-Trip-Time) 往返时延，单位 ms
*/
rtt: number | null;
/**
* 可用上行带宽，单位 bit
*/
availableOutgoingBitrate: number | null;
/**
* 可用下行带宽，在无下行资源时，其值为 `0`，单位: `bit`
*/
availableIncomingBitrate: number | null;
}
```

流状态属性

```
interface IRCTrackStat {
/**
* stat id
*/
id?: string;
/**
* 资源 Id
*/
trackId: string;
/**
* 资源类型
*/
kind: 'audio' | 'video';
/**
* 丢包率，有效值 `0` - `1`
*/
packetsLostRate: number | null;
/**
* 是否是远端资源
*/
remoteResource: boolean;
/**
* 音量
*/
audioLevel?: number | null;
/**
* 视频高度
*/
frameHeight?: number | null;
/**
* 视频宽度
*/
frameWidth?: number | null;
/**
* 视频帧率
*/
frameRate?: number | null;
/**
* 码率
*/
bitrate: number;
/**
* 网络抖动，单位 ms
* @description 下行数据中，同道流中只有一个 track 会有值，另一轨道数据值为 `0`
*/
jitter: number | null;
}
```

CallLib 3.X 升级到 5.X

更新时间:2024-08-30

CallLib SDK 5.X 是 Web 客户端 SDK 的最新版本。

相对于 3.X 版本，5.X 大量隐藏了 CallLib 业务的实现细节，封装性、易用性更好，且更稳定。我们建议 CallLib 3.X 客户尽早升级至新版 CallLib 5.X。

升级说明

CallLib 5.X 与 3.X 无法平滑升级，原因在于 3.X 过度开放了应该由 SDK 处理的内部消息，而 5.X 则将所有消息封装在了 SDK 内部，业务层不再需要关心；封装程度不同导致我们无法在 5.X 的基础上还原 3.X 的接口。

请根据 CallLib 5.X 客户端文档重新集成。您需要根据自身 API 使用情况与 API 差异，合理安排开发周期。

CallLib 5.X 升级到 CallPlus 安装依赖

更新时间:2024-08-30

为便于开发者从 CallLib 向 CallPlus 升级，融云自 CallPlus 2.1.4 版本开始，在 CallPlus 包中提供子包 `@rongcloud/plugin-call-plus/wrapper` 以用于开发者集成替换。

使用包管理器方式集成

```
# 移除旧版本 CallLib
npm rm @rongcloud/plugin-call

# 安装 CallPlus，自 2.1.4 版本后，包内将携带 wrapper 子包
npm i @rongcloud/plugin-call-plus
```

使用 CDN 方式集成

> 注意：使用 CDN 方式引入 CallPlus 与 CallPlusWrapper 包时，需确保两个包的版本号一致。

```
<script src="https://cdn.ronghub.com/RongIMLib-5.9.5.prod.js"></script>
<script src="https://cdn.ronghub.com/RCRTC-5.7.2.prod.js"></script>
<script src="https://cdn.ronghub.com/CallPlus-2.1.6.prod.js"></script>
<script src=" https://cdn.ronghub.com/CallPlusWrapper-2.1.6.prod.js"></script>
```

集成代码变动

初始化

旧方式:

```
import { installer as callInstaller, RCCallClient } from '@rongcloud/plugin-call';

const caller: RCCallClient = RongIMLib.installPlugin(callInstaller, {
rtcClient,
onSession: (session: RCCallSession) => {},
onSessionClose: (session: RCCallSession, summaryInfo?: IEndSummary) => {},
}
```

需替换为:

```
// 包名引用注意变更
import { installer as callPlusInstaller, RCCallPlusClient } from '@rongcloud/plugin-call-plus';
import { installer as wrapperInstaller, RCCallClient } from '@rongcloud/plugin-callplus/wrapper';

// 需要先初始化 CallPlus，RTCClient 初始化方式不变
const callPlusClient = RongIMLib.installPlugin(callPlusInstaller, { rtcClient });

// 初始化 CallPlusWrapper，以替换原 CallLib 的 CallClient
const caller = RongIMLib.installPlugin(wrapperInstaller, {
callPlusClient,
onSession: (session: RCCallSession) => {},
onSessionClose: (session: RCCallSession, summaryInfo?: IEndSummary) => {},
});
```

视频媒体标签

原 CallLib 在播放视频时，需要使用 `track.play()` 传入 `<video>` 标签元素；在 CallPlusWrapper 中，进行视频通话，需开发者通过调 setVideoView 设置房间人员与视频元素的绑定关系。

```
<video id="local_video_element_id"></video>
<video id="remote_video_element_id"></video>
```

```
callPlusClient.setVideoView([
{
userId: '<local-userId>',
videoElement: document.getElementById('local_video_element_id')
},
{
userId: '<remote-userId>',
videoElement: document.getElementById('remote_video_element_id')
},
]);
```

onTrackReady 通知

CallPlusWrapper 中的 `onTrackReady` 回调接口变更，不再对外回调 track 实例，因此业务层无法继续调用 `track.play()` 方法进行媒体播放，需要改为使用 CallPlus 客户端实例 `callPlusClient` 的 playMedia 方法播放。

> 注意，由于 CallPlusWrapper 需要尽量保持兼容 CallLib，因此，CallPlusWrapper 中的 `RCCallMediaType` 枚举值与 CallPlus 的 `RCCallPlusMediaType` 并不相同，需要注意在播放前进行类型转换。

```
{
onTrackReady(userId: string, mediaType: RCCallMediaType, session?: RCCallSession) {
// 枚举类型转换
const playMediaType = mediaType === RCCallMediaType.AUDIO
? RCCallPlusMediaType.AUDIO
: RCCallPlusMediaType.VIDEO;
// 使用 CallPlus 播放媒体
// 播放视频前，请确保已经调用 `setVideoView` 方法为其设置视频视图
callPlusClient.playMedia(userId, playMediaType);
}
}
```

质量数据

由于 CallPlus 中的质量数据接口变更，无法与既有 CallLib 兼容，因此 CallPlusWrapper 暂时不支持质量数据监听，`onRTCStateReport` 接口废弃；

后续版本中将在 CallPlusWrapper 中提供新的质量数据接口以满足业务对质量数据的需求。

其他

- `IRCCallInitOptions` 声明中部分字段废弃，详情请查阅 IRCCallInitOptions
- 废弃 `registerUserInfo` 方法，CallPlus Wrapper 中不支持注册用户信息和信息透传。
- `ISessionListener` 监听器中废弃包括质量数据 `onRTCStateReport` 在内的部分事件回调，详情请查阅 ISessionListener

更新日志 5.2.10

更新时间:2024-08-30

发布日期：2024/06/28

问题修复

- 修复了 npm 包内依赖声明错误导致开发者安装时可能会出现多个 `@rongcloud/engine` 版本，进而引发异常的问题。

5.2.9

发布日期：2024/06/05

功能优化

- 优化了 SDK 内部部分逻辑。

5.2.8

发布日期：2024/05/07

功能优化

- 优化了群呼叫邀请逻辑，增加通话中人员 ID 重复检查，以避免重复邀请导致的异常行为。

5.2.7

发布日期：2024/04/15

功能优化

- 当发起单呼时，若 `targetId` 与当前登录用户 ID 相同，中止呼叫并返回错误码 `RCCallErrorCode.PARAM_ERROR`。
- 当发起群呼时，若被呼叫用户列表包含当前登录用户 ID，中止呼叫并返回错误码 `RCCallErrorCode.PARAM_ERROR`。

5.2.6

发布日期：2024/03/29

问题修复

- 修复了 npm 包内 JS 文件存在高版本 ES 标准语法导致部分较低版本构建工具报错问题。
- 修复了 `disableAudioTrack` 与 `enableAudioTrack` 接口返回值错误的问题。

5.2.5

发布日期：2024/01/11

问题修复

- 修复群聊中被邀请人超时，再次邀请超时后，其他被邀请人收不到超时挂断通知问题。
- 修复本端超时，对端收到两次对端挂断通知问题。本端超时时，SDK 不再发送挂断消息。

5.2.4

发布日期：2023/11/30

功能优化

- 优化呼叫、挂断的推送信息配置（IPushConfig ），支持分移动端 OS 平台、推送厂商通道等配置推送行为。

5.2.3

发布日期：2023/10/12

- 修复 A 和 B 在通话中，A 邀请 C，C 超时未接听，B 的 `onHangup` 监听会收到 A 挂断的问题。
- 修复被邀请者接听后，获取远端人员，拿到的人员通话状态、是否是远端人员数据错误的问题。
- 修复 CallLib 正在邀请中时，当前已经在通话中的被呼叫者挂断原因为 cancel 的问题。

问题修复

5.2.2

发布日期：2023/07/11

问题修复

- 修复初始化时日志打印等级控制失效问题，废弃 `logLevel` 与 `logOutput` 配置，新增 `logOutputLevel` 配置；
- 修复对 iOS 设备发起呼叫后立即挂断，iOS 设备端偶现无法正常结束问题；
- 修复当本地系统时间与服务器时间偏差较大时，登录后无法收到 1 分钟内所产生的离线通话邀请问题；

5.2.1

发布日期：2023/05/11

功能优化

- 废弃依赖包 `@rongcloud/plugin-call-engine`，该包在 5.2.0 版本开始不再需要安装
- 内部代码结构优化

5.1.3

发布日期：2023/04/20

新增功能

- vivo 推送呼叫、挂断增加 `category` 字段

5.1.2

发布日期：2023/03/17

新增功能

- 华为推送呼叫、挂断增加 `category` 字段
- 增加呼叫、挂断推送信息配置接口 `setPushConfig`

问题修复

- 修复使用 5.7.0 以上版本的 im sdk 时，控制台不显示 call sdk 日志打印问题

5.1.1

发布日期：2023/01/11

问题修复

- 修复: userId 带下划线时呼叫报 40033 问题
- 修复: 群聊先邀请 A、A 呼叫超时后，再先后邀请 B 和 A，B 呼叫超时挂断时，A 也会挂断的问题
- 修复: 群聊中，通话在其他端处理后，当前通话邀请其他人时，本端报错问题

5.0.7

发布日期：2022/01/13

问题修复

- 增加：发起呼叫和邀请时可以携带扩展信息。
- 增加：收到呼叫时可接收扩展信息。
- 增加：发起呼叫和邀请时可自定义app端收到通知的标题和内容。

5.0.5

发布日期：2021/12/09

问题修复

- 修复：当多端同时在线，均未接听通话的情况时，Web 端未正确处理通话结束原因。
- 修复：当多端同时在线，且其中一个登录用户正处在通话中时，如果再收到通话邀请，未在通话的另一端无法正确处理通话

结束原因。

- 修复：在单聊通话中继续邀请他人加入通话时，报错不准确的问题（注意，仅群组通话允许在通话过程中继续邀请他人加入）。修复后，在单聊通话过程中继续邀请他人加入通话时，抛出错误 53311 。

5.0.4

发布日期：2021/11/26

问题修复

- 修复：当群组通话中再邀请新成员，新成员超时未接听后未自动结束通话。

5.0.3

发布日期：2021/11/04

问题修复

- 修复：当群组通话中包含 Web 端与 iOS 端时，如果 Web 邀请了新成员加入通话，iOS 端无法正常渲染新加入通话者的视频。
- 修复：当 Web 端在群组通话中时，如果 Web 端继续发起邀请，已在通话中的用户无法收到通知。

5.0.2

发布日期：2021/10/28

问题修复

- 修复发起呼叫时对事件监听的验证。
- 修复群聊中，无法监听对方挂断问题。
- 修复群聊通话中继续邀请，iOS 端接听后视频视图无法渲染问题。

5.0.1

发布日期：2021/10/22

重构说明

- 使用 Typescript 重构底层实现，提升了 SDK 的健壮性。
- 重新设计所有功能接口 API，提升了 SDK 的易用性。
- 支持通过 npm 等模块管理器安装。
- 增加详尽的注释信息，通过 IDE 编码提示可直接查阅相关接口、类型定义，使集成过程更简单。
- 增加详尽的内存数据维护，提供便捷的状态查询接口，能够有效的减少业务层的编码量及出错概率。
- 增加详尽的异常信息提示，有效提升了问题排查效率。

新增功能

- 通话中更换音频输入设备

状态码

更新时间:2024-08-30

状态码	说明
10000	成功
53200	存在未结束的通话
53201	发送 IM 消息失败
53202	被对方加入黑名单
53301	获得本地音频流失败。可根据控制台输出的 RTCLib 状态码查询具体原因。
53302	获得本地视频流失败。可根据控制台输出的 RTCLib 状态码查询具体原因。
53303	获得本地音视频流失败。可根据控制台输出的 RTCLib 状态码查询具体原因。
53304	加入房间失败。可根据控制台输出的 RTCLib 状态码查询具体原因。
53305	发布音频失败。 可根据控制台输出的 RTCLib 状态码查询具体原因。
53306	发布视频失败。可根据控制台输出的 RTCLib 状态码查询具体原因。
53307	发布音视频失败。可根据控制台输出的 RTCLib 状态码查询具体原因。
53308	查询房间用户信息失败
53309	禁用启用视频时，房间内缺少视频流
53310	取消发布视频失败。可根据控制台输出的 RTCLib 状态码查询具体原因
53311	当前不是群组通话