通话 CallPlus

Web 2.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。

实现首次通话

更新时间:2024-08-30

适用于 Web 的 CallPlus 可在您的应用程序中为用户之间的一对一和多人通信提供语音和视频通话能力。

CallPlus 支持 一对一通话 和 多人通话。按照下面的指南使用从头开始实现一对一通话和多人通话。

环境要求

- 浏览器页面地址必须为 **https** 协议地址，或使用 **localhost** 域名。
- WebRTC API 支持的浏览器兼容性可参考此文档 。

前置条件

- 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。

- 完成开通音视频服务 。您需要开通音视频通话服务。

Demo 项目

融云提供了一个 Web 端 CallPlus Demo 项目： https://github.com/rongcloud/web-quickdemo-callplus

快速上手

可以按下面的步骤快速集成音视频通话。

步骤 1 安装 SDK

CallPlus for Web SDK 是融云 `@rongcloud/imlib-next` 的一个插件。CallPlus SDK 提供音视频呼叫能力，通讯能力依赖融云 `@rongcloud/engine` 和 `@rongcloud/imlib-next`，音视频基础能力依赖融云 `@rongcloud/plugin-rtc`。

NPM

使用 NPM 需要安装四个 SDK 包

```
# npm
npm install @rongcloud/engine @rongcloud/imlib-next
npm install @rongcloud/plugin-rtc
npm install @rongcloud/plugin-call-plus
```

在 ES6 模块中导入 CallPlus SDK，如下所示：

```
import * as RongIMLib from '@rongcloud/imlib-next';
import { installer as rtcInstaller } from '@rongcloud/plugin-rtc';
import { installer as callPlusInstaller } from '@rongcloud/plugin-call-plus';
```

> 提示
>
> 如果您使用的是 TypeScript，请将 `--esModuleInterop` 选项设置为 `true` 以进行默认导入或使用 `import * as callPlus from '@rongcloud/plugin-call-plus'` 。

CDN

您可以使用 `<script>` 标签外联 Javascript 文件以引入 CallPlus 及其依赖的 SDK。

```
<!-- RongIMLib -->
<script src="https://cdn.ronghub.com/RongIMLib-5.9.5.prod.js"></script>
<!-- RTCLib v5 -->
<script src="https://cdn.ronghub.com/RCRTC-5.7.2.prod.js"></script>
<!-- CallPlus -->
<script src="https://cdn.ronghub.com/CallPlus-2.1.6.prod.js"></script>
```

步骤 2 请求访问媒体设备

CallPlus for Web SDK 需要访问麦克风和摄像头。请在集成页面的浏览器控制台中执行 `navigator.mediaDevices.getUserMedia({ audio: true, video: true })`，浏览器可能会提示用户授予麦克风和摄像头访问权限。

步骤 3 使用 App Key 初始化

CallPlus SDK 需要依赖 IMLib 作为信令通道。要在您的应用程序中集成和运行 CallPlus SDK，您需要先对 IM SDK 进行初始化。您需要按照以下步骤完成所有 SDK 初始化：

1. 使用您的在融云应用的 App Key 初始化 `RongIMLib` 实例。如果使用不同的 `APP_Key` 初始化实例，客户端应用程序中所有现有的与调用相关的数据将被清除。

2. 初始化 RTC SDK，此步骤可返回 RCRTCClient 对象。
3. 初始化 CallPlus SDK，此步骤可返回 RCCallPlusClient 对象。

> 提示
>
> 每个融云应用提供开发环境与生产环境，分别使用不同的 App Key，两个环境之间数据隔离。只要客户端应用使用同一个环境的 App Key，用户可以跨所有平台相互通信。

初始化（NPM）

```
import * as RongIMLib from '@rongcloud/imlib-next';
import { installer as rtcInstaller } from '@rongcloud/plugin-rtc';
import { installer as callPlusInstaller } from '@rongcloud/plugin-call-plus';

// im 初始化
RongIMLib.init({
appkey: '<Your-Appkey>',
});

// RTC 初始化
const rtcClient = RongIMLib.installPlugin(rtcInstaller, {});

// CallPlus 初始化
const callPlusClient = RongIMLib.installPlugin(callPlusInstaller, {
rtcClient
});
```

初始化（CDN）

```
// im 初始化
RongIMLib.init({
appkey: '<Your-Appkey>',
});

// RTC 初始化
const rtcClient = RongIMLib.installPlugin(RCRTC.installer);

// CallPlus 初始化
const callPlusClient = RongIMLib.installPlugin(CallPlus.installer, { rtcClient });
```

步骤 4 获取用户 Token

用户身份令牌（Token）与用户 ID 对应，是应用程序用户在融云的唯一身份标识。应用客户端在使用融云服务前必须与融云建立 IM 连接，连接时必须传入 Token。

在体验和调试阶段，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
```

> 融云的客户端 SDK 不提供获取 token 的 API。在实际业务运行过程中，调用融云 Server API `/user/getToken.json`，传入您的应用分配的用户标识（userId）申请 Token。详见 Server API 文档 注册用户。

步骤 5 连接融云服务器

要拨打和接听一对一对呼叫或开始多人呼叫，必须先通过 `connect` 方法连接融云服务器，传入用户身份令牌（Token），向融云服务器验证用户身份。

获取 Token 以后，可以调用 `RongIMLib.connect` 方法连接到融云服务器。

```
RongIMLib.connect('<Your-Token>').then(({code, data}) => {
if (code === RongIMLib.ErrorCode.SUCCESS) {
console.log('connect success', data?.userId);
return;
}
console.log(`connect error: ${`code`}`);
})
```

步骤 6 添加事件处理程序

CallPlus SDK 提供了 ICallPlusEventListener 事件处理程序，用于处理通话相关事件。例如收到通话呼入时可选择接听或挂断、收到通话媒体类型升级请求时可选择同意或拒绝、收到通话开始计时可在页面显示通话时长。

使用 `RCCallPlusClient` 对象的 setCallPlusEventListener 方法添加 `ICallPlusEventListener` 监听器。

```
// 注册 callPlus 业务层监听事件
callPlusClient.setCallPlusEventListener({
/**
* 呼入通知
* 收到呼入时，可选择接听或挂断通话
* @param session 通话实例
* @param extra 透传呼叫方发起呼叫时携带的附加信息
*/
async onReceivedCall(session: RCCallPlusSession, extra?: string) {
const callId = session.getCallId();
console.log(`呼入通知, callId: ${`callId`}`);
/**
* 接听电话
*/
const { code } = await callPlusClient.accept(callId);
console.log(`接听通话结果, code: ${`code`}`);
},
/**
* 收到某个人的音频或视频可播放
* 业务层可调 playMedia(userId, mediaType) 播放
* 请务必根据用户 ID 和媒体类型对流进行判断，如果是本地采集的音频流，请不要在本端播放，避免造成回声或啸叫问题。
*
* @param userId 用户 id
* @param mediaType 媒体类型
*/
onUserMediaAvailable(userId: string, mediaType: RCCallPlusMediaType) {
/**
* 播放
*/
callPlusClient.playMedia(userId, mediaType);
},
/**
* 人员状态变更
* @param callId 通话 id
* @param userId 状态变更人员 id
* @param state 人员变更后状态
* @param reason 变更原因
*/
onRemoteUserStateChanged(
// 增量
callId: string,
userId: string,
state: RCCallPlusSessionUserState,
reason?: RCCallPlusReason,
) {
console.log(`人员状态变更, callId: ${`callId`}, ${`userId`}状态: ${`RCCallPlusSessionUserState[state]`},
reason: ${`reason`}`);
},
/**
* 通话结束（群组通话时，客户端挂断不代表通话结束）
* @param session 通话实例
* @param reason 通话结束原因
*/
onCallEnded(session: RCCallPlusSession, reason: RCCallPlusReason) {
console.log(`通话结束, callId: ${`session.getCallId()`}, reason: ${`reason`}`);
}
});
```

> 提示
>
> 如果未注册 ICallPlusEventListener 事件处理程序，则用户无法接收 onReceivedCall 通话呼入事

> 件。因此，应在您注册完 callPlusInstaller 后立即执行以上代码，完成注册事件处理程序步骤。

步骤 7 添加视频媒体流播放元素

视频媒体播放元素是一个 HTMLVideoElement <video> ，用于显示视频流。要显示本端和远程视频媒体流，以下两个元素是必需添加在页面中的。id 为 local_video_element_id 的 video 标签用于播放本端视频，id 为 remote_video_element_id 的 video 标签用于播放远端视频。音频播放依赖 SDK 内部创建的 Audio 对象，业务层无需关注。

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

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

在页面准备好以上两个标签后，调 callPlusClient.setVideoView 可以设置要观看某些人的画面。如设置了要观看人员 A 的视频视图后，SDK 在获取到 A 的视频后，会在 ICallPlusEventListener 的 onUserMediaAvailable 事件通知业务层，此时业务层可调 callPlusClient.playMedia() 播放媒体资源。

```
/**
* 设置要观看人员(包括自己和对方)的媒体播放器
* @param list 为一个列表，可传入多人的媒体播放器
* @param item.userId 用户 id
* @param item.videoElement 一个 video 元素，用于播放视频，需要挂载在业务层页面上
* @param item.isTiny 是否观看别人的小流，小窗口时可以配置为 true，可节省带宽，userId 为远端用户时有效
* @returns code 返回是否设置成功
*/
setVideoView(list: {userId: string, videoElement: HTMLVideoElement, isTiny: boolean}[]): { code:
RCCallPlusCode }
```

呼叫和接听前设置本端和远端的视频播放器。

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

> 本端的视频播放器需在呼叫和接听前设置。远端人员的视频播放器可以按需设置，既可以在通话前提前设置好，也可以在 ICallPlusEventListener 的 onRemoteUserStateChanged 事件中，收到远端人员状态为 RCCallPlusSessionUserState.ONCALL 确认远端接通时设置，如果需要看远端人员的小流，可以指定 isTiny 为 true。

进行一对一通话

CallPlus for Web SDK 提供了一对一通话类型 RCCallPlusType.SINGLE 和多人通话类型 RCCallPlusType.MULTI。此处以一对一通话为例，一对一呼叫只允许传入一个被叫用户 ID，被叫接听成功后才会建立通话。

步骤 8 发起呼叫

> 提示
>
> 从 CallPlus 2.0 开始，startCall 方法新增可选参数 pushConfig 和 extra。pushConfig 支持携带移动端推送配置 IRCCallPlusPushConfig，可用于自定义推送通知标题等属性 extra 支持携带透传的自定义数据。

现在可以使用 startCall 方法，传入要呼叫的人员 userId，拨打您的第一通音视频电话了。通话建立成功时，会生成一个 callId。

调用 startCall 成功后，被呼叫人将在步骤 4 中的 onReceivedCall 监听中收到您的呼叫，被呼叫人执行步骤 7 即可建立通话。

```
/**
* 发起呼叫
* @param userIds 被叫人员 userId 列表，单人呼叫仅需在数组中放置对方一人的 userId
* @param mediaType 通话媒体类型: 音频 or 音视频
* @param type 单人呼叫 or 多人呼叫
* @returns callId 呼叫成功后，产生的会话 id
*/
callPlusClient.startCall(['<userId>'], RCCallPlusType.SINGLE, RCCallPlusMediaType.AUDIO_VIDEO);
```

步骤 9 接听通话

被叫用户通过 ICallPlusEventListener 监听器的 onReceivedCall 方法接到来电通知后，可以使用 RCCallPlusClient 实例的 accept 方法开始接听。

```
/**
* 呼入通知
* 收到呼入时，可选择接听或挂断通话
* @param session 通话实例
* @param extra 透传呼叫方发起呼叫时携带的附加信息
*/
async onReceivedCall(session: RCCallPlusSession, extra?: string) {
/**
* 接听电话
*/
const callId = session.getCallId();
const { code } = await callPlusClient.accept(callId);
console.log(`接听通话结果, code: ${`code`}`);
}
```

当您接听电话成功后，CallPlus SDK 会自动建立媒体会话。

发起多人通话

CallPlus for Web SDK 提供了支持多人呼叫的通话类型 RCCallPlusType.MULTI。发起多人通话与一对一通话使用相同的方法，但通话类型必须指定为 RCCallPlusType.MULTI。多人通话一旦发起成功，融云即开始计时计费。

> 在一对一通话过程中可以邀请用户加入通话。一旦邀请成功，通话类型会自动转为多人通话。详见 多人通话。

步骤 10 拨打多人通话

使用 callPlusClient.startCall 发起多人通话时，除了传入远端被叫用户用户 ID 、媒体类型外，必须指定通话类型为 RCCallPlusType.GROUP。建议视频通话最多不超过 16 人，音频通话最多不超过 32 人。

```
/**
* 发起呼叫
* @param userIds 被叫人员 userId 列表，单人呼叫仅需在数组中放置对方一人的 userId
* @param mediaType 通话媒体类型: 音频 or 音视频
* @param type 单人呼叫 or 多人呼叫
* @returns callId 呼叫成功后，产生的会话 id
*/
callPlusClient.startCall(['<userId1>', '<userId2>'], RCCallPlusType.MULTI,
RCCallPlusMediaType.AUDIO_VIDEO);
```

多人通话的接听方法与一对一通话一致。

> 提示
>
> 有关在您的应用程序中构建语音和视频通话功能的详细指南，请参阅完整的 一对一通话和多人通话 文档。我们建议您先按照一对一通话实现完整的通话流程，再按照多人通话补充实现与一对一通话有差异的步骤。

与 IM 业务集成 与 IM 业务集成

更新时间:2024-08-30

您可以将 CallPlus 集成到基于 IMLib SDK 的应用程序中，为用户提供使用通话和聊天服务的无缝体验。

在会话页面插入通话结束消息

应用程序可以通话结束时返回聊天视图，并在会话页面中展示通话结束的消息。

为便于与 IMLib SDK 集成，从 2.1.1 版本开始，CallPlus SDK 可在通话结束时返回一个 IM 消息对象 IRCCallPlusReceivedSummaryMessage，其中包含消息内容对象 RCCallPlusSummaryMessageContent。应用程序可以通过调用 IMLib SDK 的接口直接将该消息内容对象插入到会话中。

获取通话结束消息

> **重要**
>
> CallPlus SDK 暂仅支持在一对一通话（通话类型为 `RCCallPlusType.SINGLE`）结束时返回用于展示通话记录的消息内容。暂不支持支持多人通话。

通话正常、异常结束，或在未接听时挂断，CallPlus SDK 会触发 ICallPlusEventListener 的 onReceivedCallPlusSummaryMessage 回调方法，可在该回调方法获取到通话结束的消息内容对象。

如果被叫时用户不在线，CallPlus SDK 会在下次成功连接 IM 后立即获取离线时产生的 `RCCallPlusSummaryMessageContent`。为了确保 CallPlus SDK 可在 IM 连接成功后立即获取到这些消息，请在 IM 连接之前调用 setCallPlusEventListener 注册监听器。

```
// callPlus 业务层监听事件增加 onReceivedCallPlusSummaryMessage
callPlusClient.setCallPlusEventListener({
/**
* 收到通话结束的消息记录，可用于在 IM 聊天界面插入通话结束消息
* 仅单聊可收到
* 触发时机：
* 1.单聊在线通话结束后
* 2.离线时收到单聊呼叫，通话结束后，重新连接 IM 在线时
* @param message 通话记录的消息体
* @param message.messageType 通话结束消息类型，为 RC:CallPlusMiss
* @param message.senderUserId 为通话发起者
* @param message.targetId 为单聊中对方的 userId
* @param message.messageDirection 如果本端是呼叫发起者，消息方向则为发送，反之为接收
* @param message.content IRCCallPlusSummaryMessageContent，内部包含通话的 callId、通话发起者、通话对方
userId、通话媒体类型、通话开始时间、通话结束时间、通话挂断原因
* @since 2.1.1
*/
onReceivedCallPlusSummaryMessage(message: IRCCallPlusReceivedSummaryMessage) {
console.log(message);
}
});
```

在 Electron IMLib V5 中插入通话结束消息

收到通话结束消息时，可以将 RCCallPlusSummaryMessageContent 通过 insertMessage 方法插入到本地会话中，并自行刷新 UI。

在 ICallPlusEventListener 的 onReceivedCallPlusSummaryMessage 回调方法中获取 IRCCallPlusReceivedSummaryMessage 对象中包含的 RCCallPlusSummaryMessageContent，在本地会话页面中插入一条通话结束消息。插入消息示例代码：

```
onReceivedCallPlusSummaryMessage(message: IRCCallPlusReceivedSummaryMessage) {
const { conversationType, targetId } = message;

const conversation = {
conversationType,
targetId,
};

RongIMLib.electronExtension.insertMessage(conversation, message).then(res => {
if (res.code === RongIMLib.ErrorCode.SUCCESS) {
// 消息插入成功，返回 IAReceivedMessage 类型的消息数据。
console.log('消息插入成功', res.data)
} else {
console.log('消息插入失败', res.code, res.msg)
}
});
}
```

一对一通话

更新时间:2024-08-30

本页介绍了一对一呼叫的主要功能，包括如何从您的应用程序拨打、接听、处理和结束呼叫。

关键类介绍

RCCallPlusClient

RCCallPlusClient ： CallPlus for Web 的核心类，用于管理客户端呼叫行为，例如发起、接听、挂断通话，操作音视频设备，管理通话记录等。

方法	说明
setCallPlusEventListener	注册应用层事件，包含“收到呼叫、通话结束、通话人员状态变更、通话类型变更”等事件
setStatusReportListener	注册音视频上下行丢包数据监听
setVideoView	设置要观看人员(包括自己和对方)的媒体播放器
removeVideoView	删除指定单个用户的媒体播放器，不观看指定单个用户的视频时，业务层可以直接移除其媒体播放器
playMedia	播放指定人员的音视频媒体，播放指定单个用户的视频前，请确保已经调用 `setVideoView` 方法为其设置视频视图
createVideoConfigBuilder	创建视频配置构建器
setVideoConfig	设置视频配置
createAudioConfigBuilder	创建音频配置构建器
setAudioConfig	设置音频配置
startCall	发起呼叫
joinMultiCall	直接加入进行中的群组通话
invite	通话中邀请他人
accept	接听通话
hangup	挂断通话
requestChangeMediaType	通话中发起媒体类型变更请求
cancelChangeMediaType	取消切换媒体类型变更请求
replyChangeMediaType	媒体类型变更应答
getCurrentCallSession	获取当前进行中的通话
getAvailableCallRecordsFromServer	从服务器获取未结束的通话记录
getRemoteTracks	获取远端资源列表
getCallRecords	分页查找当前用户的全部通话记录
deleteCallRecordsFromServer	根据 callId 集合批量删除通话记录

方法	说明
deleteAllCallRecordsFromServer	清空所有通话记录
startCamera	开启摄像头
stopCamera	关闭摄像头
switchCamera	切换摄像头
startMicrophone	开启麦克风
stopMicrophone	关闭麦克风
isMicrophoneEnable	麦克风是否开启
muteAllRemoteAudio	静音所有远端音频流
getVersion	获取 SDK 版本信息

RCCallPlusSession

RCCallPlusSession 对象代表一则通话的所有信息，包含获取通话实体信息的方法集合。

方法	说明
getCallId()	获取通话 ID。
getState()	获取通话实体的状态 RCCallPlusCallState。
getCallType()	获取通话类型 RCCallPlusType。
getMediaType()	获取通话媒体类型 RCCallPlusMediaType。
getUserList()	获取一组用户信息对象，每个对象中包含通话人员用户 ID 与用户状态 RCCallPlusSessionUserState。
getCreatorUserId()	获取通话发起者 userId
getInviterUserId()	获取通话邀请者 userId
getCreateTimestamp()	获取通话起始时间
getDuration()	获取通话时长

ICallPlusEventListener

ICallPlusEventListener 监听器提供来电事件、通话状态、通话记录等事件相关回调。

方法	说明
onReceivedCall	本地用户收到来电通知。
onRemoteUserInvited	本地用户收到收到远端人员被邀请加入通话通知，如 A 与 B 通话中，A 邀请 C，则 B 会收到此回调。`inviteeUserList` 表示受邀人用户 ID 列表。`inviterUserId` 表示邀请者用户 ID。`callId` 表示所属通话；群聊中，己方已不在通话中的情况下，也会收到服务侧的事件通知
onCallConnected	通话已建立，sdk 内部会发布音视频资源
onCallTypeChanged	本地用户收到通话类型变更通知。
onRemoteUserStateChanged	本地用户收到人员状态变更通知。
onCallEnded	本地用户收到通话结束（群组通话时，客户端挂断不代表通话结束）
onRemoteCameraStateChanged	本地用户收到远端摄像头开、关通知
onRemoteMicrophoneStateChanged	本地用户收到远端麦克风开、关通知

方法	说明
onReceivedCallLog	本地用户单呼或群呼结束后，收到服务器下发的通话日志
onReceivedChangeMediaTypeRequest	本地用户收到媒体类型变更请求（仅单聊）
onReceivedChangeMediaTypeResult	本地用户收到媒体类型变更结果（仅单聊）
onReceivedCallStartTime	本地用户收到收到通话开始计时通知
onReceivedCallFirstFrameTime	本地用户收到收到通话首帧到达通知
onUserMediaAvailable	本地用户收到指定单个用户的音频或视频可播放
onFetchRemoteMediaError	收到获取本人或远端媒体资源失败
onUserAudioLevelChanged	本地用户收到人员音量变化

呼叫流程

CallPlus SDK 中的一对一通话是指两人之间通话。一对一呼叫通过以下步骤建立：

1. 主叫方发起呼叫，SDK 内部为主叫用户创建在音视频资源，以在通话中使用。被叫方会收到 onReceivedCall 通知。
2. 当被叫方选择接听通话后，SDK 内部会为被叫用户创建音视频资源，以在通话中使用。
3. 接听成功后，主叫方和被叫方的音视频资源将会被推送至融云媒体服务器。
4. 融云信令服务器通知主叫方和被叫方可查看对方的音视频媒体资源。
5. 主叫方和被叫方可以调 setVideoView 指定要观看的用户视频画面，SDK 内部会自动获取音频资源。
6. 主叫方和被叫方可在 ICallPlusEventListener 监听器的 onUserMediaAvailable 事件中调 playMedia(useId,mediaType) 播放指定用户的音视频资源。**注意：请务必根据用户 ID 和媒体类型进行判断，如果是本地用户采集的音频流，请不要在本端播放，避免造成回声或啸叫问题。**

对于主叫用户来说， 调用 startCall 方法后，将返回一个解析为 `{ code: RCCallPlusCode, callId?: string }` 的 promise。对于被叫用户来说，将通过 `ICallPlusEventListener` 监听器的 `onReceivedCall` 方法收到 RCCallPlusSession 通话对象。对于主叫用户和被叫用户来说，都可以调用通话客户端 `RCallPlusClient` 对象的 getCurrentCallSession 方法获取当前进行中的通话对象。

前提条件

- App Key 已开通音视频通话服务。
- 已初始化 CallPlus SDK，并获取 `RCCallPlusClient` 对象。本文中的 `callPlusClient` 即为初始化时获取的 `RCCallPlusClient` 实例对象。

注册监听器

CallPlus for Web SDK 提供了 ICallPlusEventListener 事件处理程序，用于接收来自远端用户或服务端的事件，方便应用程序处理并做出反应。例如收到通话呼入时可选择接听或挂断、收到通话媒体类型升级请求时可选择同意或拒绝、收到通话开始计时可在页面显示通话时长。

获取 RCCallPlusClient 对象后，使用 `setCallPlusEventListener()` 注册监听器。

```
// 注册 callPlus 业务层监听事件
callPlusClient.setCallPlusEventListener({
```

```
/**
* 呼入通知
* 收到呼入时，可选择接听或挂断通话
* @param session 通话实例
* @param extra 透传呼叫方发起呼叫时携带的附加信息
*/
async onReceivedCall(session: RCCallPlusSession, extra?: string) {
const callId = session.getCallId();
console.log(`呼入通知, callId: ${`callId`}`);
/**
* 接听电话
*/
const { code } = await callPlusClient.accept(callId);
console.log(`接听通话结果, code: ${`code`}`);
},
/**
* 群组通话收到远端人员被邀请加入通话通知，如 A 与 B 通话中，A 邀请 C，则 B 会收到此回调
* @param inviteeUserList - 受邀人用户 ID 列表
* @param inviterUserId - 邀请者用户 ID
* @param callId - 所属通话；群聊中，己方已不在通话中的情况下，也会收到服务侧的事件通知
*/
onRemoteUserInvited(inviteeUserList: string[], inviterUserId: string, callId: string) {
console.log(`收到远端人员被邀请加入通话通知, callId: ${`callId`}, inviteeUserList: ${`inviteeUserList`},
inviterUserId: ${`inviterUserId`}`);
},
/**
* 通话已建立，sdk 内部会发布音视频资源
*/
onCallConnected(session: RCCallPlusSession) {
console.log(`通话已建立, callId: ${`session.getCallId()`}`);
},
/**
* 通话类型变更
* @param type 单呼或群呼
* @param callId 通话 id
*/
onCallTypeChanged(type: RCCallPlusType, callId: string) {
console.log(`通话类型变更, callId: ${`callId`}, type: ${`type`}`);
},
/**
* 人员状态变更
* @param callId 通话 id
* @param userId 状态变更人员 id
* @param state 人员变更后状态
* @param reason 变更原因
*/
onRemoteUserStateChanged(
// 增量
callId: string,
userId: string,
state: RCCallPlusSessionUserState,
reason?: RCCallPlusReason,
) {
console.log(`人员状态变更, callId: ${`callId`}, ${`userId`}状态: ${`RCCallPlusSessionUserState[state]`},
reason: ${`reason`}`);
},
/**
* 通话结束（群组通话时，客户端挂断不代表通话结束）
* @param session 通话实例
* @param reason 通话结束原因
*/
onCallEnded(session: RCCallPlusSession, reason: RCCallPlusReason) {
console.log(`通话结束, callId: ${`session.getCallId()`}, reason: ${`reason`}`);
},
/**
```

```
* 远端摄像头开、关通知
* @param callId 通话 id
* @param userId 用户 id
* @param disabled 是否关闭
*/
onRemoteCameraStateChanged(callId: string, userId: string, disabled: boolean) {
console.log(`远端摄像头开、关通知, callId: ${`callId`}, userId: ${`userId`}, disabled: ${`disabled`}`);
},
/**
* 远端麦克风开、关通知
* @param callId 通话 id
* @param userId 用户 id
* @param disabled 是否关闭
*/
onRemoteMicrophoneStateChanged(callId: string, userId: string, disabled: boolean) {
console.log(`远端m麦克风开、关通知, callId: ${`callId`}, userId: ${`userId`}, disabled: ${`disabled`}`);
},
/**
* 单呼或群呼结束后，服务器下发的通话日志
* @param record 数据接口定义为：IRCCallPlusCallRecord，可通过 apiDoc 查看具体包含字段
*/
onReceivedCallRecord(record: IRCCallPlusCallRecord) {
console.log(`通话日志下发, record: ${`JSON.stringify(record)`}`);
},
/**
* 收到媒体类型变更请求（仅单聊）
* @param userId 请求发起人
* @param transactionId 事物 id，本次请求和应答的唯一标识
* @param mediaType 请求变更的媒体类型
*/
onReceivedChangeMediaTypeRequest(
userId: string, transactionId: string, mediaType: RCCallPlusMediaType,
) {
console.log(`收到媒体类型变更请求, userId: ${`userId`}, transactionId: ${`transactionId`}, mediaType:
${`mediaType`}`);
/**
* 应答是否同意变更媒体请求
*/
const isAgree = true;
callPlusClient.replyChangeMediaType(transactionId, isAgree);
},
/**
* 媒体类型变更结果（仅单聊）
* @param info.userId
* @param info.transactionId 事物 id，本次请求和应答的唯一标识
* @param info.mediaType
* @param info.code - 升级结果
* - 升级成功
* - 发起升级请求方取消
* - 收到升级请求方拒绝
* - 响应超时
*/
onReceivedChangeMediaTypeResult(info: {
userId: string,
transactionId: string,
mediaType: RCCallPlusMediaType,
code: RCCallPlusMediaTypeChangeResult
}) {
const { userId, transactionId, mediaType, code } = info;
console.log(`媒体类型变更结果, userId: ${`userId`}, transactionId: ${`transactionId`}, mediaType:
${`mediaType`}, code: ${`code`}`);
},
/**
* 收到通话开始计时
* @param info.callId 通话 id
* @param info.callType 单聊或群聊
```

```
* @param info.callStartTime 通话开始时间
*/
onReceivedCallStartTime(info: { callId: string, callType: RCCallPlusType, callStartTime: number }) {
console.log(`通话计时开始, startTime: ${`info.callStartTime`}`);
/**
* 起通话计时定时器
*/
let callDuration = Math.floor((+new Date() - info.callStartTime) / 1000);
setInterval(() => { callDuration++; }, 1000);
},
/**
* 收到首帧时间（仅单聊存在）
* 收到首帧开始计费
* @param callId 通话 id
* @param callFirstFrameTime 通话首帧到达时间
*/
onReceivedCallFirstFrameTime(callId: string, callFirstFrameTime: number) {
console.log(`收到首帧时间, callFirstFrameTime: ${`callFirstFrameTime`}`);
},
/**
* 收到指定单个用户的音频或视频可播放
* 业务层可调 playMedia(userId, mediaType) 播放
* @param userId 用户 id
* @param mediaType 媒体类型
*/
onUserMediaAvailable(userId: string, mediaType: RCCallPlusMediaType) {
/**
* 请务必根据用户 ID 和媒体类型对流进行判断，如果是本地采集的音频流，请不要在本端播放，避免造成回声或啸叫问题。
*/
callPlusClient.playMedia(userId, mediaType);
},
/**
* 获取远端媒体资源失败
* @param userId 远端人员 id
* @param code 失败原因
*/
onFetchRemoteMediaError(userId: string, code: RCCallPlusCode) {
console.log(`获取远端媒体资源失败, userId: ${`userId`}, code: ${`code`}`);
},
/**
* 人员音量变化
* @param userId 本端或远端人员 id
* @param audioLevel 音量值 (0-100)
*/
onUserAudioLevelChanged(userId: string, audioLevel: number) {
console.log(`人员音量变化, userId: ${`userId`}, audioLevel: ${`audioLevel`}`);
},
})
```

> 提示
>
> 如果未注册 ICallPlusEventListener 事件处理程序，则用户无法接收 onReceivedCall 通话呼入事件。建议在初始化 CallPlus SDK 后添加此处理程序。

添加媒体播放元素

视频媒体播放元素是一个 HTMLVideoElement <video> ，用于显示视频流。要显示本端和远程视频媒体流，以下两个元素是必需添加在页面中的。id 为 local_video_element_id 的 video 标签用于播放本端视频，id 为

remote_video_element_id 的 video 标签用于播放远端视频。音频播放依赖 SDK 内部创建的 Audio 对象，业务层无需关注。

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

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

在页面准备好以上两个标签后，调 callPlusClient.setVideoView 可以设置要观看某些人的画面。如设置了要观看人员 A 的视频视图后，SDK 在获取到 A 的视频后，会在 ICallPlusEventListener 的 onUserMediaAvailable 事件通知业务层，此时业务层可调 callPlusClient.playMedia() 播放媒体资源（请勿在本端播放本地用户采集的音频流，避免造成回声或啸叫问题）。

```
/**
* 设置要观看人员(包括自己和对方)的媒体播放器
* @param list 为一个列表，可传入多人的媒体播放器
* @param item.userId 用户 id
* @param item.videoElement 一个 video 元素，用于播放视频，需要挂载在业务层页面上
* @param item.isTiny 是否观看别人的小流，小窗口时可以配置为 true，可节省带宽，userId 为远端用户时有效
* @returns code 返回是否设置成功
*/
setVideoView(list: {userId: string, videoElement: HTMLVideoElement, isTiny: boolean}[]): { code:
RCCallPlusCode }
```

呼叫和接听前设置本端和远端的视频播放器。

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

> 本端的视频播放器需在呼叫和接听前设置。远端人员的视频播放器可以按需设置，既可以在通话前提前设置好，也可以在 ICallPlusEventListener 的 onRemoteUserStateChanged 事件中，收到远端人员状态为 RCCallPlusSessionUserState.ONCALL 确认远端接通时设置，如果需要看远端人员的小流，可以指定 isTiny 为 true。

发起呼叫

> 提示

> 从 CallPlus 2.0 开始，startCall 方法新增可选参数 pushConfig 和 extra。pushConfig 支持携带移动端推送配置 IRCCallPlusPushConfig，可用于自定义推送通知标题等属性 extra 支持携带透传的自定义数据。

通过在 startCall 方法中提供被叫方的用户 ID、通话类型、通话媒体类型来发起呼叫。通话发起成功时，会返回成一个 callId。

```
/**
* 发起呼叫
* @param userIds 被叫方人员 userId 列表，单人呼叫仅需在数组中放置对方一人的 userId
* @param type 单人呼叫 or 多人呼叫
* @param mediaType 通话媒体类型: 音频 or 音视频
* @returns callId 呼叫成功后，产生的会话 id
*/
callPlusClient.startCall(['<userId>'], RCCallPlusType.SINGLE, RCCallPlusMediaType.AUDIO_VIDEO);
```

接听电话

被叫方的客户端应用程序中必须已先注册 ICallPlusEventListener 才能收到来电事件 onReceivedCall 和通话中的其他事件。

使用 RCCallPlusClient 对象的 accept 或 hangup 方法接听或拒绝呼叫。如果呼叫被接听，CallPlus SDK 将自动建立媒体会话。

```
callPlusClient.setCallPlusEventListener({
/**
* 呼入通知
* 收到呼入时，可选择接听或挂断通话
* @param session 通话实例
* @param extra 透传呼叫方发起呼叫时携带的附加信息
*/
async onReceivedCall(session: RCCallPlusSession, extra?: string) {
/**
* 接听电话
*/
const callId = session.getCallId();
const { code } = await callPlusClient.accept(callId);
console.log(`接听通话结果, code: ${`code`}`);
}
});
```

CallPlus SDK 通过与融云服务器的 IM 连接接收呼叫事件，该连接由 RongIMLib.connect() 建立。如果意外网络中断导致断开连接，SDK 内部会自动重新连接。

处理当前通话

音频

在通话过程中，主叫方和被叫方的音频都可以通过 RCCallPlusClient 对象的 stopMicrophone 或 startMicrophone 方法静音或取消静音。如果一方变更麦克风状态，另一方会通过 ICallPlusEventListener 监听器的 onRemoteMicrophoneStateChanged 监听器接收事件回调。

```
/**
* 开启麦克风
* @description 通话过程中打开麦克风，如果扩散资源失败，即对方没收到本端打开麦克风的通知时，此次行为失败
*/
async startMicrophone(): Promise<{ code: RCCallPlusCode }>

/**
* 关闭麦克风
* @description 通话过程中关闭麦克风，如果扩散资源失败，即对方没收到本端关闭麦克风的通知时，此次行为失败
*/
async stopMicrophone(): Promise<{ code: RCCallPlusCode }>

/**
* 切换麦克风
* 切换成功后 `ICallPlusEventListener` 监听器的 `onUserMediaAvailable` 中会收到新切换的麦克风资源可播放，此时调用 `callPlusClient.playMedia()` 可播放新麦克风的声音
* @param microphoneId 浏览器中取得的麦克风设备 id，可调 getMicrophones 方法取得
* @description 通话前和通话中都可调用
* @since 2.1.1
*/
async switchMicrophone(microphoneId: string): Promise<{ code: number }> {}
```

视频

在通话过程中，主叫方和被叫方的视频都可以通过 RCCallPlusClient 对象的 startCamera 或 stopCamera 方法启用或禁用摄像头。如果一方变更摄像头状态，另一方会通过 ICallPlusEventListener 监听器的 onRemoteCameraStateChanged 接收事件回调。

```
/**
* 开启摄像头
* @description 通话过程中打开摄像头，如果扩散资源失败，即对方没收到本端打开摄像头的通知时，此次行为失败
*/
async startCamera(): Promise<{ code: RCCallPlusCode }>

/**
* 关闭摄像头
*/
async stopCamera(): Promise<{ code: RCCallPlusCode }>

/**
* 切换摄像头
* 切换成功后 `ICallPlusEventListener` 监听器的 `onUserMediaAvailable` 中会收到新切换的摄像头资源可播放，此时调用 `callPlusClient.playMedia()` 可显示新摄像头的画面
* @param cameraId 浏览器中取得的摄像头设备 id，可调 getCameras 方法取得
* @description 通话前和通话中都可调用
*/
async switchCamera(cameraId: string): Promise<{ code: number}> {}
```

还可以使用 switchCamera 切换摄像头，需指定设备 ID，切换成功后 ICallPlusEventListener 监听器的 onUserMediaAvailable 中会收到新切换的摄像头资源可播放，此时调用 callPlusClient.playMedia() 可显示新摄像头的画面

```
/**
* 切换摄像头
* @param cameraId 浏览器中取得的摄像头设备 id，可调 plugin-rtc 的 getCameras 方法取得，可参考：
https://doc.rongcloud.cn/meeting/Web/5.X/device#cameras
* @description 通话前和通话中都可调用
*/
async switchCamera(cameraId: string): Promise<{ code: RCCallPlusCode}>
```

变更媒体类型

> 仅一对一通话支持在通话中变更媒体类型。

CallPlus SDK 定义了音视频通话 RCCallPlusMediaType.AUDIO_VIDEO 和音频通话 RCCallPlusMediaType.AUDIO 媒体类型。在一对一通话过程中，主叫方和被叫方都可以通过 requestChangeMediaType 方法来请求切换媒体类型。

```
/**
* 通话中发起媒体类型变更请求
* @param mediaType 媒体类型
* @returns transactionId 客户端和服务端交互的事务 id，在取消或应答媒体类型切换时传入
*/
async requestChangeMediaType(mediaType: RCCallPlusMediaType): Promise<{ code: RCCallPlusCode,
transactionId?: string }>
```

发起请求变更媒体类型的一方，还可以通过 cancelChangeMediaType 方法来取消切换通话媒体类型的请求。

```
/**
* 取消切换媒体类型变更请求
* @param transactionId 客户端和服务端交互的事务 id，为发起媒体类型变更请求返回的 transactionId
*/
async cancelChangeMediaType(transactionId: string): Promise<{ code: RCCallPlusCode }>
```

远端用户会通过 ICallPlusEventListener 监听器的 onReceivedChangeMediaTypeRequest 方法收到媒体类型变更请求。收到请求后，通过 RCCallPlusClient 对象的 replyChangeMediaType 方法选择是否同意变更通话媒体类型。

```
/**
* 媒体类型变更应答
* @param transactionId 客户端和服务端交互的事务 id，为 onReceivedChangeToVideo 监听收到 transactionId 参数值
*/
async replyChangeMediaType(transactionId: string, isAgree: boolean): Promise<{ code: RCCallPlusCode }>
```

媒体类型变更最终的结果会通过 ICallPlusEventListener 监听器的 onReceivedChangeMediaTypeResult 方法通知请求方和确认方。

```
/**
* 媒体类型变更结果（仅单聊）
* @param info.userId
* @param info.transactionId 事物 id，本次请求和应答的唯一标识
* @param info.mediaType 最终的媒体类型
* @param info.code - 升级结果
* - 请求方取消媒体类型变更
* - 应答方拒绝媒体类型切换
* - 服务仲裁允许媒体类型切换
*/
onReceivedChangeMediaTypeResult(info: {
userId: string,
transactionId: string,
mediaType: RCCallPlusMediaType,
code: RCCallPlusMediaTypeChangeResult
}): void
```

结束通话

> 提示
>
> 从 CallPlus 2.0 开始，hangup 方法新增可选参数 pushConfig，支持携带移动端推送配置 IRCCallPlusPushConfig，可以用于自定义推送通知标题等属性。

挂断与拒绝接听动作均使用 RCCallPlusClient 对象的 hangup 方法，主叫用户或被叫用户都可以使用 hangup() 结束正在进行的通话。如果一方结束正在进行的通话，另一方会通过 ICallPlusEventListener.onCallEnded() 方法收到事件回调。

```
/**
* 挂断通话
* @param callId 为可选参数，不传时，挂断正在进行的主 session，传入时可指定挂断主通话或暂未建立连接的通话
* 暂未建立连接的通话：指已在通话中时，收到呼叫但暂未接听的通话
*/
async hangup(callId?: string): Promise<{ code: RCCallPlusCode }>
```

> 如果通话为一对一类型，一方挂断则双方同时挂断，融云将停止对通话计时计费。

在通话中接听来电

您可以在通话过程中接听来电。因为一次只能有一个活动呼叫，所以在接听新的来电时，SDK 内部会挂断正在进行中的通话。

> 提示
>
> CallPlus SDK 不支持在通话过程中保持和恢复通话。

要接收来电，请使用 RCCallPlusClient 对象的 accept 方法，来电将变为当前通话。

```
/**
* 接听通话
* @param callId 通话 id
*/
async accept(callId: string): Promise<{ code: RCCallPlusCode }>
```

获取通话记录

通话结束后获取通话记录

通话结束（或拒接来电）后，ICallPlusEventListener.onReceivedCallRecord() 中会立马下发刚结束的通话记录。

```
/**
* 单呼或群呼结束后，服务器下发的通话日志
* @param record 数据接口定义为：IRCCallPlusCallRecord，可通过 apiDoc 查看具体包含字段
*/
onReceivedCallRecord(record: IRCCallPlusCallRecord): void
```

检索服务端通话记录

可以调 RCCallPlusClient 对象的 getCallRecords 方法主动从融云服务端获取用户的通话历史记录。

从 2.1.0 版本开始新增 order 参数，支持倒序查询通话记录。例如，分页获取最近的通话记录。首次可以将 syncTime 设置为 -1。如果需要继续查询，可以将返回的 syncTime 作为下次查询的 syncTime 值。

```
/**
* 分页查找当前用户的全部通话记录
* @param startTimestamp 起始时间戳，首次可传 -1
* @param count 每次获取的数量，最大 100
* @param order 正序或倒序获取，默认为正序。0 为正序，向开始时间戳之后查询。1 为倒序，向开始时间戳之前查询。
* @returns result.hasMore 是否有更多通话记录
* @returns result.list 通话记录列表，类型为 IRCCallPlusCallRecord[]，可通过 apiDoc 查看具体包含字段
* @returns result.syncTime hasMore 为 true 时，继续获取通话记录需要传入该时间戳
*/
async getCallRecords(startTimestamp: number, count: number, order: 0|1 = 0): Promise<{ code:
RCCallPlusCode, result?: IRCCallPlusGetCallRecordRes }>
```

管理音视频配置

CallPlus SDK 使用本地用户浏览器默认的摄像头与麦克风设备。默认视频分辨率为 W640_H480，帧率为 FPS_15。可以通过 RCVideoConfigBuilder 和 RCAudioConfigBuilder 对象构建新的视频、音频配置项，再通过 RCCallPlusClient 的 setVideoConfig 和 setAudioConfig 使配置项生效。

设置本地视频采集配置

构建视频配置项

需先创建一个视频配置构建器 RCVideoConfigBuilder，使用 RCVideoConfigBuilder 可以配置视频分辨率、帧率、摄像头 id、视频传输码率。

```
import { RCResolution, RCFrameRate } from '@rongcloud/plugin-rtc';
/**
* createVideoConfigBuilder 创建视频配置构建器
* @returns RCVideoConfigBuilder 用于构造视频采集时的配置项，可以设置摄像头设备 id、帧率、分辨率等，
* 设置完需调 build 方法，最终在 setVideoConfig 后生效
*/
const videoConfigBuilder = callPlusClient.createVideoConfigBuilder();

// 配置分辨率
videoConfigBuilder.setVideoResolution(RCResolution.W640_H360)
// 配置帧率
.setFrameRate(RCFrameRate.FPS_24)
/**
* 配置摄像头Id
* cameraId 取值可参考
https://doc.rongcloud.cn/rtc/Web/5.X/device#%E8%8E%B7%E5%8F%96%E6%91%84%E5%83%8F%E5%A4%B4%E8%AE%BE%E5%A4
5%88%97%E8%A1%A8
*/
.setDefaultCameraId(cameraId)
/**
* 设置最小、最大码率
* 单位 kbps
* 不设置时，取动态码率计算结果
* @param min 最小码率
* @param max 最大码率
*/
.setBitrate(min, max);

// 构建视频配置
const videoConfig = videoConfigBuilder.build();
```

设置视频配置项

```
/**
* 设置视频配置
* 下次采集时生效，如切换摄像头、重新建立通话
*/
callPlusClient.setVideoConfig(videoConfig);
```

修改本地音频采集配置

需先创建一个音频配置构建器 RCAudioConfigBuilder，使用 RCAudioConfigBuilder 可以配置麦克风 id、音频采样率、音频传输码率。

构建音频配置项

```
/**
* 创建音频配置构建器
* @returns RCAudioConfigBuilder 用于构造音频采集时的配置项，可以设置麦克风设备 id
* 设置完需调 build 方法，最终在 setAudioConfig 后生效
*/
const audioConfigBuilder = callPlusClient.createAudioConfigBuilder();

/**
* 指定默认麦克风
* microphoneId 取值可参考
https://doc.rongcloud.cn/rtc/Web/5.X/device#%E8%8E%B7%E5%8F%96%E9%BA%A6%E5%85%8B%E9%A3%8E%E8%AE%BE%E5%A4
5%88%97%E8%A1%A8
*/
audioConfigBuilder.setDefaultMicrophoneId(microphoneId);
/**
* 设置最小、最大码率
* 单位 kbps
* 不设置时，取动态码率计算结果
* @param min number 最小码率
* @param max number 最大码率
*/
.setBitrate(min, max);
/**
* 设置音频采样率
* @param sampleRate number 音频采样率
*/
.setSampleRate(sampleRate);

// 构建音频配置
const audioConfig = audioConfigBuilder.build();
```

设置音频配置项

```
/**
* 设置音频配置
* 下次采集时生效，如重新建立通话
*/
callPlusClient.setAudioConfig(audioConfig);
```

管理通话质量

用户可以收到有关通话质量变化的通知，以便他们可以检查自己的网络连接，以避免在通话过程中出现任何中断。

设置 IStatusReportListener

要检测通话质量的变化，需要设置 IStatusReportListener。调用 RCCallPlusClient 对象的 setStatusReportListener() 可设置通话质量监听事件。

```
/**
* 注册音视频上下行丢包数据监听
* 每秒上报一次
* @param listener 类型为 IStatusReportListener，详细定义可参考 apiDoc
* @param listener.onSendPacketLoss 发送丢包率信息回调
* @param listener.onReceivePacketLoss 接收丢包率信息回调
*/
setStatusReportListener(listener: IStatusReportListener)
```

IStatusReportListener 说明

方法	说明
onSendPacketLoss(packetLostRate: number, delay: number): void	packetLostRate 丢包率，0-100，delay 发送端的网络延迟，单位毫秒，每秒上报一次
onReceivePacketLoss(data: { [userId: string]: RCCallPlusPacketLossStats }): void	data[userId].packetLostRate 丢包率:取值范围是 0-100，data[userId].bitRate 码率大小，单位是 kbps

多人通话

更新时间:2024-08-30

CallPlus SDK 提供了支持多人呼叫的通话类型 `RCCallPlusType.GROUP`。本文介绍了多人通话的主要功能，包括如何发起多人通话、单人通话转为多人通话等。

> 提示
>
> 因为 CallPlus for Web 多人通话与一对一通话流程相似，本文仅介绍多人通话与一对一通话有差异的使用方法。我们建议您先按照一对一通话实现完整的通话流程。

前提条件

- App Key 已开通音视频通话服务。
- 已初始化 CallPlus SDK，并获取 RCCallPlusClient 对象。本文中的 `callPlusClient` 即为初始化时获取的 `RCCallPlusClient` 对象。

发起多人通话

> 提示
>
> 从 CallPlus 2.0 开始，startCall 方法新增可选参数 `pushConfig` 和 `extra`。`pushConfig` 支持携带移动端推送配置 IRCCallPlusPushConfig，可用于自定义推送通知标题等属性 `extra` 支持携带透传的自定义数据。

如需直接发起多人通话，请在 startCall 方法中传入多个远端用户 ID，并设置通话类型为 `RCCallPlusType.GROUP`。媒体类型可为视频或音频。您可以创建最多 16 名视频参与者的多人通话，或最多 32 名音频参与者的多人通话。该方法会通过 Promise 返回 RCCallPlusCode 和 `callId`。被叫用户会通过 ICallPlusEventListener 的 `onReceivedCall(session: RCCallPlusSession)` 方法收到来电通知。

```
const { code, callId } = await callPlusClient.startCall(['<userId1>', '<userId2>'],
RCCallPlusType.MULTI, RCCallPlusMediaType.AUDIO_VIDEO);
```

> 提示
>
> 多人通话一旦发起成功，融云即开始计时计费。

接听多人通话

> 提示
>
> 被叫用户必须已设置 ICallPlusEventListener 监听器，并实现 onReceivedCall(session: RCCallPlusSession) 方法。被叫用户通过与融云的 IM 连接接收来电通知。

被叫用户收到来电通知后，可以选择接听或拒绝来电。应用程序可以通过 RCCallPlusSession 的 getCallId() 方法获取通话 ID，可以通过 getCallType() 方法查询通话类型是否为多人通话。

- 接听来电，使用 `RCCallPlusClient` 对象的 `accept(callId)` 方法，或者 `joinCall(callId)` 方法。
- 挂断来电，使用 `RCCallPlusClient` 对象的 `hangup(callId)` 方法。

接听方法、加入通话方法与挂断方法在 SDK 内部为异步执行，API 会通过 Promise 返回调用结果 RCCallPlusCode，为 0 表示成功。

邀请他人加入通话

您可以向某些用户发送邀请来让用户进入多人通话。被邀请者可以接受或拒绝您的邀请。

> 提示
>
> CallPlus 暂不支持取消邀请。

邀请用户

> 提示
>
> 从 CallPlus 2.0 开始，invite 方法新增可选参数 pushConfig 和 extra。pushConfig 支持携带移动端推送配置 IRCCallPlusPushConfig，可用于自定义推送通知标题等属性 extra 支持携带透传的自定义数据。

首先，本地用户需要已进入一个多人通话或一对一通话。要邀请其他用户，使用 invite 方法，并传入用户 ID。本地用户可通过返回的 Promise 获取调用结果。Promise 中的 busyUsers 参数表示忙线的用户列表。

```
/**
* 通话中邀请他人
* 单人通话中邀请他人会引发通话类型变更
* @param userIds 被邀请人列表
* @returns busyUsers 返回忙线人员列表
*/
async invite(userIds: string[]): Promise<{ code: RCCallPlusCode, busyUsers?: RCCallPlusCallUser[] }>
```

成功发送邀请后，受邀用户的设备上将通过 ICallPlusEventListener 监听器的 onReceivedCall(RCCallPlusSession callSession) 方法收到来电通知，通知他们您已邀请他们加入通话。

获取邀请详情

在通话中发起邀请后，被邀请者、正在通话中的用户、已退出多人通话的用户都将通过 ICallPlusEventListener 监听器的 onRemoteUserInvited 方法收到通知，其中携带通话 ID、发起邀请者用户 ID 与所有受邀用户的 ID。

```
/**
* 群组通话收到远端人员被邀请加入通话通知，如 A 与 B 通话中，A 邀请 C，则 B 会收到此回调
* @param inviteeUserList - 受邀人用户 ID 列表
* @param inviterUserId - 邀请者用户 ID
* @param callId - 所属通话；群聊中，己方已不在通话中的情况下，也会收到服务侧的事件通知
*/
onRemoteUserInvited(inviteeUserList: string[], inviterUserId: string, callId: string):void
```

接受或拒绝邀请

当被邀请的用户接受或拒绝邀请时，邀请他们的用户将会收到通知。您可以添加 UI 更新，让邀请者知道他们邀请的用户是接受还是拒绝邀请。

- 被邀请者接受邀请，可以使用 `RCCallPlusClient` 对象的 `accept(callId)` 方法接听来电，或者 `joinCall(callId)` 方法加入多人通话。
- 被邀请者拒绝邀请，可以使用 您的 `RCCallPlusClient` 对象的 `hangup(callId)` 方法挂断来电。

接听方法与挂断方法在 SDK 内部为异步执行，调用结果会通过 Promise 返回 RCCallPlusCode，为 0 表示成功。

被邀请者进入通话后，邀请者和被邀请者将能够查看彼此的媒体流。

如果原通话是一对一通话，受邀请者加入后，通话类型将变更为 RCCallPlusType.GROUP，通话参与者（包括已退出的用户）都会通过 onCallTypeChanged 方法收到通知。

获取未结束的多人通话

用户可能拒绝一个多人通话邀请，或退出一个多人通话，只要该多人通话未结束，用户就可以再次加入。

首先，使用 getAvailableCallRecordsFromServer 方法向服务端查询本地用户所有曾经参与过（或曾经被邀请过）、且仍在进行中的多人通话。

```
/**
* 从服务器获取未结束的通话记录
* @returns records 一组未结束的通话记录
*/
async getAvailableCallRecordsFromServer(): Promise<{ code: RCCallPlusCode, records?:
IRCCallPlusCallRecord[] }>
```

该方法返回一个 Promise 其中包含调用结果码与 IRCCallPlusCallRecord 对象列表。获取未结束通话的记录后，通过 IRCCallPlusCallRecord 中 getCallId() 方法获取通话 ID，使用 joinCall 方法传入通话 ID 可加入正在进行中的多人通话。

配置推送属性

更新时间:2024-08-30

CallPlus 2.0 支持对呼叫生成的信令消息的推送行为进行个性化配置。例如：

- 自定义推送标题
- 自定义通知栏图标
- 其他 APNs 或 Android 推送通道支持的个性化配置

请在发起通话前和邀请通话前提供 IRCCallPlusPushConfig 配置。

推送属性说明

`IRCCallPlusPushConfig` 提供以下参数：

参数	类型	说明
disablePushTitle	boolean	是否屏蔽通知标题，此属性只针目标用户为 iOS 平台时有效，Android 第三方推送平台的通知标题为必填项，所以暂不支持。
pushTitle	String	推送标题，此处指定的推送标题优先级最高。如不设置，则使用 CallPlus 服务端默认标题。
templateId	String	推送模板 ID，设置后根据目标用户通过移动端 IM SDK 中的 `setPushLauguageCode` 设置的语言环境，匹配模板中设置的语言内容进行推送，未匹配成功时使用默认内容进行推送。模板内容在“控制台-自定义推送文案”中进行设置，具体操作请参见 配置和使用自定义多语言推送模板 。
iOSConfig	IiOSPushConfig	iOS 平台相关配置。支持 `threadId`、`apnsCollapseId`、`richMediaUri`，用法详见 API 文档。
androidConfig	IAndroidPushConfig	Android 平台相关配置。支持针对小米、华为、荣耀、OPPO、vivo、魅族、FCM 推送渠道配置消息分类、渠道 ID、通知栏图片等。用法详见 API 文档。

更新日志

更新时间:2024-08-30

更新日志按时间顺序列出了 CallPlus 的所有新功能、变更、和已修复的问题。格式基于 Keep a Changelog 。

变动类型：

- 新增(`Added`)：新添加的功能。
- 变更(`Changed`)：对现有功能的变更。
- 废弃(`Deprecated`)：已经不建议使用，即将移除的功能。
- 移除(`Removed`)：已经移除的功能。
- 修复(`Fixed`)：对 bug 的修复。
- 安全改进(`Security`)：对安全性的改进。

2.1.6 - 2024-06-28

新增

- 新增了 `setAudioOutputDeviceId` 接口以指定媒体声音播放设备。
- `@rongcloud/plugin-call-plus/wrapper` 包中新增了 `onSendPacketLoss`、`onReceivePacketLoss` 事件通知，以便于业务层接口通话质量数据。

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。

2.1.5 - 2024-06-11

修复

- 修复 npm 包内资源缺失导致安装 `@rongcloud/plugin-call-plus` 包后项目编译失败问题（受影响版本 v2.1.2 - v2.1.4）。

2.1.4 - 2024-06-05

新增

- 新增了 `@rongcloud/plugin-call-plus/wrapper` 包，以帮助 CallLib 集成用户快速迁移至 CallPlus。

修复

- 修复了通话中用户列表数据可能存在异常和重复，进而导致的当对端挂断时本端无法挂断的问题。

2.1.3 - 2024-05-08

修复

- 修复了通话中接听其他呼入通话后，新接听通话中使用已挂断通话的 mediaType 进行取流问题。
- 修复了主动呼叫后，对方未接听的情况下取消并挂断，再次呼叫失败（错误码 80002）问题。

2.1.2 - 2024-03-29

修复

- 修复了多人通话中有人员挂断后被重新邀请，发出邀请者一端内存中记录的人员信息重复而导致的异常。
- 内部代码优化，并修复内部日志不规范导致的异常错误。
- 修复一个偶现的 SDK 内部异常错误。

2.1.1 - 2024-01-31

新增

- 为便于与 IM 业务集成，从 2.1.1 版本开始，CallPlus SDK 可在通话结束时返回一个 IM 消息对象。应用程序可以通过调用 IM SDK 的相关接口直接将通话记录消息插入到会话中。

2.1.0 - 2024-01-04

新增

- getCallRecords 方法新增 `order` 参数，支持倒序查询通话记录，可用于查询用户最近的通话记录。

变更

- getCallRecords 方法返回数据中的 `maxTime` 字段改为 `syncTime`。

修复

- 修复通话中程序被杀死或意外崩溃后，60 秒内无法正常发起呼叫的问题。

2.0.0 - 2023-11-30

新增

- startCall 和 invite 方法新增可选参数 `pushConfig` 和 `extra`。`pushConfig` 支持携带移动端推送配置 IRCCallPlusPushConfig，可用于自定义推送通知标题等属性 `extra` 支持携带透传的自定义数据。支持在发起呼叫时和邀请通话配置自定义通知标题等属性，支持携带自定义数据。
- hangup 方法新增可选参数 `pushConfig`，支持携带移动端推送配置 IRCCallPlusPushConfig，可以用于自定义推送通知标题等属性。

1.1.1 - 2023-11-02

变更

- 由 SDK 内部拦截本地播放已方音频的行为，避免造成回声、啸叫等问题。

1.1.0 - 2023-09-25

新增

- 自动上报 SDK 版本号。

1.0.0 - 2023-09-15

新增

- 首次发布 CallPlus for Web SDK。

状态码 RCCallPlusCode

更新时间:2024-08-30

callPlus 中定义的接口返回错误码内容如下：

属性	值	说明
SUCCESS	0	执行成功
PARAM_ERROR	80001	参数错误
SESSION_EXIST	80002	存在未结束的 session
NOT_IN_CALL	80003	未加入通话
MEDIATYPE_INVALID	80004	媒体类型被禁止
NOT_VIDEO_CALL	80005	当前不是视频通话
RTC_SERVICE_UNAVAILABLE	80006	开通的音视频服务没有及时生效或音视频服务已关闭
USER_LIST_INVAILD	80007	调用函数时，传入的用户列表为空错误
CALL_ID_INVALID	80008	调用函数时，传入的通话 Id 为空错误
TRANSACTION_ID_INVALID	80009	事务 Id 为空错误
USER_ID_INVALID	80010	调用函数时，传入的用户 Id 为空错误
SINGLE_CALL_NOT_SUPPORT_MULTI_PERSON	80011	发起单人呼叫时，人员列表中只能有一个人
CAMERA_CLOSED	80100	摄像头未打开
MICROPHONE_CLOSED	80101	麦克风未打开
MEDIA_RESOURCE_INVALIED	80102	无媒体资源
NOT_INSTALL_RTC_PLUGIN	80103	未 install RTC 插件
PLAY_MEDIA_FAILED	80104	播放异常
VIDEO_VIEW_NOT_SET	80105	播放视频需要 video 元素