

会议 / 直播

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。

提示

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

SDK 适用场景

CallKit、CallLib、RTCLib 是融云 RTC 服务提供的三款经典的客户端 SDK。其中 CallKit、CallLib 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）	CallKit、CallLib	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。

- **CallKit 与 CallLib** 是用于实现通话（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。CallKit 与 CallLib 均依赖 RTCLib，两者区别如下：
 - CallKit 提供了呼叫相关的通用 UI 扩展库。
 - CallLib 不含任何 UI 界面组件。
- **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 CallLib。
- 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) Windows(x86、x86-64)	---

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。

 提示

[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。

 提示

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](#)。

运行会议 Demo 项目

更新时间:2024-08-30

融云音视频会议 Demo 演示了融云产品[音视频会议](#)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、语音聊天室、娱乐直播、教学课堂、多人会议等场景需求。

- QuickDemo ([GitHub](#) · [Gitee](#))，本文以运行 QuickDemo 为例。

如果想要直接体验产品效果，欢迎[前往融云官网查看各场景的演示应用](#)。

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 Demo。

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。

- 开通音视频通话服务。开发环境下可免费开通，每个应用均可享有 10000 分钟免费体验时长，免费体验时长用完即止。生产环境下需要先预存费用才可开通。服务开通后最长 30 分钟生效。详见[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":"gxld6GHx3t1eDxof1qtxxYr0cjkbh1V@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

以下我们使用已获取的 Token，模拟 userId 为 1 的用户连接到融云服务器。

1. 克隆下载示例代码。

```
git clone https://github.com/rongcloud/web-quickdemo-rtc-meeting.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，和房间号即可进入体验。

房间号长度不能超过 64，可包含 A-Z、a-z、0-9、+、=、-、_。

运行直播 Demo 项目

更新时间:2024-08-30

融云低延迟直播 Demo 演示了融云产品[低延迟直播](#)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、语音聊天室、娱乐直播、教学课堂、多人会议等场景需求。

- QuickDemo ([GitHub](#) · [Gitee](#))，本文以运行 QuickDemo 为例。

如果想要直接体验产品效果，欢迎[前往融云官网查看各场景的演示应用](#)。

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 Demo。

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。

- 开通音视频通话服务，以及音视频直播服务。开发环境下可免费开通，每个应用均可享有 10000 分钟免费体验时长，免费体验时长用完即止。生产环境下需要先预存费用才可开通。服务开通后最长 30 分钟生效。详见[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":"gxld6GHx3t1eDxof1qtxxYr0cjkbhl1V@sgyu.cn.example.com;sgyu.cn.example.c
```

重复上一步，获取 **userId** 为 2 的用户的 Token。记录下该 Token，用于体验时使用。

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
- 已获取用于体验的两个 Token

以下我们使用已获取的两个 Token，分别模拟主播端和观众端的使用行为。

1. 克隆下载示例代码。

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

下载后可看到两个页面：`anchor.html` 为主播端页面，`audience.html` 为观众端页面。

2. 使用浏览器直接打开主播端页面 `anchor.html`。运行成功后，请按照提示点击，并输入以下值：
 - App Key
 - 主播 Token：userId 为 1 的用户的 Token
 - 房间 ID：房间号长度不能超过 64，可包含 `A-Z`、`a-z`、`0-9`、`+`、`=`、`-`、`_`
3. 主播端发布完资源后，页面将显示直播视频流地址 (`liveUrl`)。
4. 使用浏览器直接打开主播端页面 `audience.html`。运行成功后，请按照提示点击，并输入以下值：
 - App Key
 - 观众 Token：userId 为 2 的用户的 Token
 - 直播视频流地址：请从主播端的页面复制 `liveUrl` 地址，在观众端输入

安装 RTCLib SDK

更新时间:2024-08-30

实时音视频业务依赖即时通讯业务提供的信令通道。基于 RTCLib 开发应用，您必须安装两个库：

- 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。

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 --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
```

① 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。

安装 RTCLib

① 提示

- 使用 RTCLib 5.5.1 及以上版本 需要将 IM 升级到 5.5.0 及以上的版本。
- 使用 RTCLib 5.6.1 及以上版本 需要将 IM 升级到 5.6.0 及以上的版本。

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

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。请按步骤顺序完成初始化流程。

注意事项

RTCLib 是基于 IMLib 的插件机制所实现的插件。因此必须先初始化 IMLib，再初始化 RTCLib。

准备 App Key

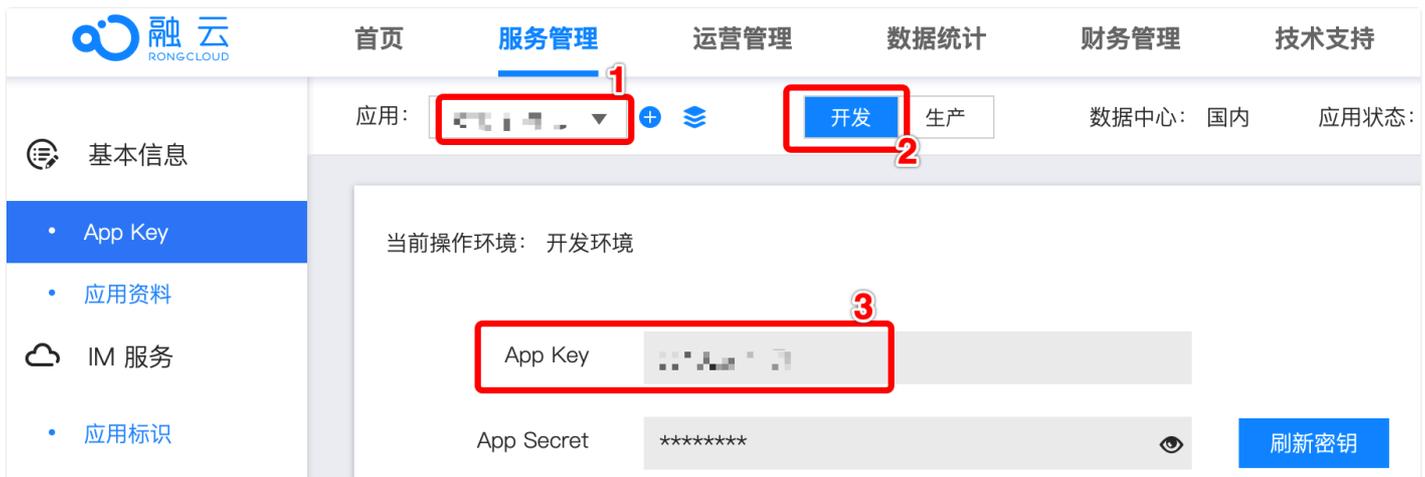
您必须拥有正确的 App Key，才能进行初始化。应用客户端只有在使用同一个 App Key 的情况下，才能实现用户间消息跨平台互通。

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：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 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](#)。

步骤 1: 初始化 IMLib

本步骤仅说明如何获取 IMLib 实例。关于 IMLib 的详细初始化过程，请查阅对应版本和平台的 IMLib 初始化文档。

初始化 5x 版本 IMLib

以下示例使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```
import * as RongIMLib from '@rongcloud/imlib-next'
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'
// 初始化 IM
RongIMLib.init({
  appkey: '<Your-Appkey>',
});

/**
 * 监听消息通知
 */
const Events = RongIMLib.Events;
RongIMLib.addEventListener(Events.MESSAGES, (event) => {
  console.log('received messages', event.messages);
});

/**
 * 监听 IM 连接状态变化
 */
RongIMLib.addEventListener(Events.CONNECTING, () => {
  console.log('onConnecting');
});
RongIMLib.addEventListener(Events.CONNECTED, () => {
  console.log('onConnected');
});
RongIMLib.addEventListener(Events.DISCONNECT, (status) => {
  console.log('连接中断，需要业务层进行重连处理 ->', status)
})
```

初始化 4x 版本 IMLib

以下示例使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```
import * as RongIMLib from '@rongcloud/imlib-v4-adapter'
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'
// 获取 IMLib 实例
const im = RongIMLib.init({ appkey: '<Your-Appkey>' })
// 监听 IMLib 连接状态变化
im.watch({
  status (status) {
  // IM 连接状态变更通知
  }
})
```

如需 IMLib v4 的 API 接口说明，请转至 [API 参考文档](#)：

- [init](#)
- [watch](#)

初始化 2x 版本 IMLib

以下示例使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```
import { RongIMClient, ConnectionStatus } from '@rongcloud/imlib-v2-adapter'  
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'  
// 初始化 IMLib  
RongIMClient.init('<Your-Appkey>')  
// 设置 IM 连接状态监听  
RongIMClient.setConnectionStatusListener({  
  onChange (status: ConnectionStatus) {  
    // 连接状态变更通知  
  }  
})
```

步骤 2: 初始化 RTCLib

以下示例使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

5x 版本 IMLib install RTCLib

如果使用 IMLib 5.X，调用 [installPlugin](#) 方法初始化 RTCLib。

```
// 初始化 RCRTCClient，初始化过程推荐放在建立连接之前  
const rtcClient = RongIMLib.installPlugin(installer, myIRCRTCInitOptions)
```

初始化 RTCLib 时，可按需传入 RTCLib 初始化参数，详见 [IRCRTCInitOptions](#)。以下仅为示例：

```

{
/**
 * 自定义 MediaServer Url，公有云用户无需关注
 * @description
 * 1. 仅当 `location.hostname` 为 `localhost` 时，`http` 协议地址有效，否则必须使用 `https` 协议地址
 * 2. 当该值有效时，将不再从 IMLib 导航数据中获取 mediaServer 地址
 */
mediaServer?: string,
/**
 * 输出日志等级，生产环境默认使用 WARN，开发环境默认使用 DEBUG
 * @description
 * * 4 - DEBUG
 * * 3 - INFO
 * * 2 - WARN
 * * 1 - ERROR
 */
logLevel?: LogLevel
/**
 * 与 MediaServer 的 http 请求超时时间，单位为毫秒，默认值为 `5000`，有效值 `5000-30000`。
 * 优先级：用户配置 > 导航配置 > 默认时间。
 */
timeout?: number,
/**
 * 房间 Ping 间隔时长，默认 `10000` ms，有效值 `3000`-`10000`
 */
pingGap?: number,
/**
 * 内置 CDN 观看地址直播拉流协议，默认为 RCInnerCDNPullKind.FLV
 */
pullInnerCDNProtocol?: RCInnerCDNPullKind
/**
 * 内置 CDN 观看地址是否使用 https，默认为 RCInnerCDNPullIsHttps.HTTPS
 */
pullInnerCDNUseHttps?: RCInnerCDNPullIsHttps
}

```

4x 版本 IMLib install RTCLib

如果使用 IMLib 4.X，需要调用 `install` 方法初始化 [RCRTCClient](#)。options 参数的详细说明参见 [IRCRTCInitOptions](#)

```

// 初始化 RCRTCClient，初始化过程推荐放在建立连接之前
const rtcClient = im.install(installer, { /*初始化参数请参考 IRCRTCInitOptions 参数说明*/ })

```

2x 版本 IMLib install RTCLib

如果使用 IMLib 2.X，需要调用 `install` 方法初始化 [RCRTCClient](#)。options 参数的详细说明参见 [IRCRTCInitOptions](#)

```

// 初始化 RCRTCClient，初始化过程推荐放在建立连接之前
const rtcClient = RongIMLib.RongIMClient.getInstance().install(installer, { /*初始化参数请参考 IRCRTCInitOptions 参数说明*/ })

```

步骤 3: IMLib 连接

5x 版本 IMLib 连接

```
RongIMLib.connect('<Your-Token>').then((user) => {
  console.log('connect success', user.data.userId);
})
.catch((error) => {
  console.log(`连接失败: ${error}`);
});
```

4x 版本 IMLib 连接

API 参考：[connect](#)

```
// 连接 im
im.connect({ token: '<Your-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);
});
```

2x 版本 IMLib 连接

```
// 连接 im
RongIMClient.connect('<Your-Token>', {
  onSuccess: function(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);
  }
});
```

Web 兼容性

Web 兼容性说明

更新时间:2024-08-30

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 Android 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+
	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

	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
	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

实现音视频会议

更新时间:2024-08-30

融云开发者账户是使用融云 SDK 产品的必要条件。在开始之前，请[前往融云官网注册开发者账户](#)。注册后，控制台将自动为你创建一个应用，默认为开发环境应用，使用国内数据中心。请获取该应用的 App Key，在本教程中使用。

首次使用融云音视频的用户，建议参考文档[运行示例项目](#)，以完成开发者账号注册、音视频服务开通等工作。

步骤 1：服务开通

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

具体步骤请参阅[开通音视频服务](#)。

提示

服务开通、关闭等设置完成后 30 分钟后生效。

步骤 2：SDK 导入

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

安装 IMLib

推荐使用 IMLib 5.X。您也可以使用 2.x 或 4.x 的 Adapter SDK。

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

安装 RTCLib

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

上述步骤仅简要说明安装方法，详见[安装 RTCLib SDK](#)。

步骤 3：初始化

您需要先初始化 IMLib，再初始化 RTCLib。

初始化 IMLib

RTCLib 是基于 IMLib 的插件机制所实现的插件。因此需要先初始化 IMLib。推荐使用 IMLib 5.X。

以下示例基于 IMLib 5.X。使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```
import * as RongIMLib from '@rongcloud/imlib-next'
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'
// 初始化 IM
RongIMLib.init({
  appkey: '<Your-Appkey>',
});

/**
 * 监听消息通知
 */
const Events = RongIMLib.Events;
RongIMLib.addEventListener(Events.MESSAGES, (event) => {
  console.log('received messages', event.messages);
});

/**
 * 监听 IM 连接状态变化
 */
RongIMLib.addEventListener(Events.CONNECTING, () => {
  console.log('onConnecting');
});
RongIMLib.addEventListener(Events.CONNECTED, () => {
  console.log('onConnected');
});
RongIMLib.addEventListener(Events.DISCONNECT, (status) => {
  console.log('连接中断，需要业务层进行重连处理 ->', status)
})
```

初始化 RTCLib

如果使用 IMLib 5.X，需要调用 [installPlugin](#) 方法初始化 [RCRTCClient](#)。options 参数的详细说明参见 [IRCRTCInitOptions](#)。

以下示例基于 IMLib 5.X。使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```

// 初始化 RCRTCClient，初始化过程需要在建立连接之前
const rtcClient = RongIMLib.installPlugin(installer, {

/**
 * 自定义 MediaServer Url，公有云用户无需关注
 * @description
 * 1. 仅当 `location.hostname` 为 `localhost` 时，`http` 协议地址有效，否则必须使用 `https` 协议地址
 * 2. 当该值有效时，将不再从 IMLib 导航数据中获取 mediaServer 地址
 */
mediaServer?: string,

/**
 * 输出日志等级，生产环境默认使用 WARN，开发环境默认使用 DEBUG
 * @description
 * * 4 - DEBUG
 * * 3 - INFO
 * * 2 - WARN
 * * 1 - ERROR
 */
logLevel?: LogLevel

/**
 * 与 MediaServer 的 http 请求超时时间，单位为毫秒，默认值为 `5000`，有效值 `5000-30000`。
 * 优先级：用户配置 > 导航配置 > 默认时间。
 */
timeout?: number,

/**
 * 房间 Ping 间隔时长，默认 `10000` ms，有效值 `3000`-`10000`
 */
pingGap?: number,

/**
 * 内置 CDN 观看地址直播拉流协议，默认为 RCInnerCDNPullKind.FLV
 */
pullInnerCDNProtocol?: RCInnerCDNPullKind

/**
 * 内置 CDN 观看地址是否使用 https，默认为 RCInnerCDNPullIsHttps.HTTPS
 */
pullInnerCDNUseHttps?: RCInnerCDNPullIsHttps
})

```

上述步骤仅简要说明初始化方法。详细说明请参阅[初始化](#)。

步骤 4：连接融云

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。应用客户端成功连接到融云服务器后，才能使用融云即时通讯 SDK 进行信令传输。

连接时融云服务端必须传入 Token 参数。Token 是与用户 ID 对应的身份验证令牌，是应用客户端用户在融云的唯一身份标识。融云服务端在收到客户端发起的连接请求后，会根据连接请求里携带的用户身份验证令牌（Token 参数），判断是否允许用户连接。

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向融云服务端申请取得 Token，具体方法可参考 [Server API 获取 Token](#)。

如果仅希望快速体验和测试，您可以直接从控制台获取一个用户 Token。具体操作请参见[运行示例项目](#)。

以下示例使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```
import * as RongIMLib from '@rongcloud/imlib-next'
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'

// 建立 IM 连接
RongIMLib.connect('<Your-Token>').then((user) => {
  console.log('connect success', user.data.userId);
})
.catch((error) => {
  console.log(`连接失败: ${`error`}`);
});
```

步骤 5: 加入房间

RTCLib 初始化后，可获取到 RTC 客户端实例。调用 [joinRTCRoomWithOptions](#) 方法可加入 RTC 房间。

原 [joinRTCRoom](#) 接口自 5.7.0 版本开始废弃，建议使用 [joinRTCRoomWithOptions](#) 替代。

以下示例中 `rtcClient` 表示 RTC 客户端实例。加入房间成功后，会返回 [RCRTCRoom](#) 房间实例、房间内其他用户 ID，以及房间内已发布的资源。

```
// 加入普通音视频房间，从 5.0.7 开始增加返回 `tracks` 与 `userIds`
// * userIds - 当前已加入房间的远端人员列表
// * tracks - 当前已发布至房间内的远端资源列表
const { code, room, userIds, tracks: remoteTracks } = await rtcClient.joinRTCRoomWithOptions('roomId')

// 若加入失败，则 room、userIds、tracks 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('join living room failed:', code)
  return
}
```

步骤 6: 获取房间数据

会议模式下，加入房间成功后可获取 [RCRTCRoom](#) 实例。RCRTCRoom 实例上提供以下获取资源与数据的方法：

方法	返回值类型	说明
<code>getRoomId()</code>	String	返回房间 ID。
<code>getSessionId()</code>	String	返回房间的 Session ID。重新加入房间时，该 ID 会重置。
<code>getLocalTracks()</code>	RCLocalTrack[]	返回已发布资源列表（从 RTCLib 5.0.5 开始支持）。
<code>getRemoteUserIds()</code>	string[]	返回房间中已存在的远端用户列表，不包含本端 <code>userId</code>
<code>getRemoteTracks()</code>	RCRemoteTrack[]	返回远端发布的所有资源列表（从 RTCLib 5.0.7 开始支持）。
<code>getRemoteTracksByUserId()</code>	RCRemoteTrack[]	返回指定用户在房间内发布的资源列表（从 RTCLib 5.0.7 开始支持）。

步骤 7: 获取资源

使用初始化时获取的 `RCRTCClient` 实例的 [createMicrophoneAndCameraTracks](#) 方法可同时捕获音视频流。

```

/**
 * @description tracks 是一个数组，当 `code !== RCRTCCode.SUCCESS` 时，tracks 长度为 0
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器摄像头、麦克风资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、\、- 的字符串，
 * @param options 音视频配置项，可参考上述 1、2 中的介绍
 */
const { code, tracks } = await rtcClient.createMicrophoneAndCameraTracks(tag: string = 'RongCloudRTC',
options?: { audio?: IMicphoneAudioProfile, video?: ICameraVideoProfile })

if (code === RCRTCCode.SUCCESS) {
// tracks 包含一个 RCMicphoneAudioTrack 实例和一个 RCCameraVideoTrack 实例
const [ audioTrack, videoTrack ] = tracks
}

```

使用 [play](#) 方法在本端播放采集的数据。通常情况下，尽量不要在本端播放本端采集的音频流，否则可能会引起回声问题。

```

// 通过 videoTrack.play 方法将 <video> 标签传递给 videoTrack 实例
videoTrack.play(videoNode)

// 播放音频时无需传参，尽量不要在本端播放本端采集的音频流，因为可能会引起回声问题
audioTrack.play()

// 播放音频设置音量，volume 为 0-100 的数字
audioTrack.play(null, {volume})

// 指定播放音频的输出设备，输出音频设备列表可通过 device.getSpeakers() 获取
// device 可从 @rongcloud/plugin-rtc 模块导出
audioTrack.play(null, {audioDeviceId})

```

步骤 8: 发布资源

使用 [RCRTCRoom](#) 房间实例的 [publish](#) 方法发布一个或多个音视频轨道 [RCLocalTrack](#) 数据。

```

const { code } = await room.publish([audioTrack, videoTrack])

// 若资源发布失败
if (code !== RCRTCCode.SUCCESS) {
console.log('资源发布失败:', code)
}

```

步骤 9: 资源订阅

调用 [subscribe](#) 方法订阅音视频资源。视频流支持订阅大小流机制。为了节省下行带宽，可选择订阅对方的视频小流。仅在其他用户发布资源时设置了发布大小流的情况下可订阅小流。若对方并未发布小流，接收的仍然是对方的大流数据。

```
const { code } = await room.subscribe([
// 音频不支持大小流
audioTrack,
{
track: videoTrack,
subTiny: true // 订阅小流
}
])
```

步骤 12: 退出房间

```
// room 为加入房间方法返回的实例对象
const { code } = await rtcClient.leaveRoom(room)
```

实现低延迟直播

更新时间:2024-08-30

融云开发者账户是使用融云 SDK 产品的必要条件。在开始之前，请先[前往融云官网注册开发者账户](#)。注册后，控制台将自动为您创建一个应用，默认为开发环境应用，使用国内数据中心。请获取该应用的 App Key，在本教程中使用。

首次使用融云音视频的用户，建议参考文档[运行示例项目](#)，以完成开发者账号注册、音视频服务开通等工作。

步骤 1：服务开通

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

具体步骤请参阅[开通音视频服务](#)。

提示

服务开通、关闭等设置完成后 30 分钟后生效。

步骤 2：SDK 导入

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

安装 IMLib

推荐使用 IMLib 5.X。您也可以使用 2.x 或 4.x 的 Adapter SDK。

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

安装 RTCLib

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

上述步骤仅简要说明安装方法，详见[安装 RTCLib SDK](#)。

步骤 3：初始化

您需要先初始化 IMLib，再初始化 RTCLib。

初始化 IMLib

RTCLib 是基于 IMLib 的插件机制所实现的插件。因此需要先初始化 IMLib。推荐使用 IMLib 5.X。

以下示例基于 IMLib 5.X。使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```
import * as RongIMLib from '@rongcloud/imlib-next'
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'
// 初始化 IM
RongIMLib.init({
  appkey: '<Your-Appkey>',
});

/**
 * 监听消息通知
 */
const Events = RongIMLib.Events;
RongIMLib.addEventListener(Events.MESSAGES, (event) => {
  console.log('received messages', event.messages);
});

/**
 * 监听 IM 连接状态变化
 */
RongIMLib.addEventListener(Events.CONNECTING, () => {
  console.log('onConnecting');
});
RongIMLib.addEventListener(Events.CONNECTED, () => {
  console.log('onConnected');
});
RongIMLib.addEventListener(Events.DISCONNECT, (status) => {
  console.log('连接中断，需要业务层进行重连处理 ->', status)
})
```

初始化 RTCLib

如果使用 IMLib 5.X，需要调用 [installPlugin](#) 方法初始化 [RCRTCClient](#)。options 参数的详细说明参见 [IRCRTCInitOptions](#)。

以下示例基于 IMLib 5.X。使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```

// 初始化 RCRTCClient，初始化过程需要在建立连接之前
const rtcClient = RongIMLib.installPlugin(installer, {

/**
 * 自定义 MediaServer Url，公有云用户无需关注
 * @description
 * 1. 仅当 `location.hostname` 为 `localhost` 时，`http` 协议地址有效，否则必须使用 `https` 协议地址
 * 2. 当该值有效时，将不再从 IMLib 导航数据中获取 mediaServer 地址
 */
mediaServer?: string,

/**
 * 输出日志等级，生产环境默认使用 WARN，开发环境默认使用 DEBUG
 * @description
 * * 4 - DEBUG
 * * 3 - INFO
 * * 2 - WARN
 * * 1 - ERROR
 */
logLevel?: LogLevel

/**
 * 与 MediaServer 的 http 请求超时时间，单位为毫秒，默认值为 `5000`，有效值 `5000-30000`。
 * 优先级：用户配置 > 导航配置 > 默认时间。
 */
timeout?: number,

/**
 * 房间 Ping 间隔时长，默认 `10000` ms，有效值 `3000`-`10000`
 */
pingGap?: number,

/**
 * 内置 CDN 观看地址直播拉流协议，默认为 RCInnerCDNPullKind.FLV
 */
pullInnerCDNProtocol?: RCInnerCDNPullKind

/**
 * 内置 CDN 观看地址是否使用 https，默认为 RCInnerCDNPullIsHttps.HTTPS
 */
pullInnerCDNUseHttps?: RCInnerCDNPullIsHttps
})

```

上述步骤仅简要说明初始化方法。详细说明请参阅[初始化](#)。

步骤 4：连接融云

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。应用客户端成功连接到融云服务器后，才能使用融云即时通讯 SDK 进行信令传输。

连接时融云服务端必须传入 Token 参数。Token 是与用户 ID 对应的身份验证令牌，是应用客户端用户在融云的唯一身份标识。融云服务端在收到客户端发起的连接请求后，会根据连接请求里携带的用户身份验证令牌（Token 参数），判断是否允许用户连接。

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向融云服务端申请取得 Token，具体方法可参考 [Server API 获取 Token](#)。

如果仅希望快速体验和测试，您可以直接从控制台获取一个用户 Token。具体操作请参见[运行示例项目](#)。

以下示例使用 **Typescript** 进行编码，便于开发者更好的理解相关值的类型信息。

```

import * as RongIMLib from '@rongcloud/imlib-next'
import { installer, RCRTCCode } from '@rongcloud/plugin-rtc'

// 建立 IM 连接
RongIMLib.connect('<Your-Token>').then((user) => {
  console.log('connect success', user.data.userId);
})
.catch((error) => {
  console.log(`连接失败: ${`error`}`);
});

```

主播端

步骤 5.1: 加入房间

RTCLib 初始化后，可获取到 RTC 客户端实例。调用 [joinLivingRoom](#) 方法可加入房间。

以下示例中 `rtcClient` 表示 RTC 客户端实例。加入房间成功后，会返回主播角色用户的 [RCLivingRoom](#) 房间实例、房间内其他用户 ID，以及房间内已发布的资源。

```

// 获取 RCLivingType 枚举定义
import { RCLivingType } from '@rongcloud/plugin-rtc'

/**
 * 主播加入直播房间或观众上麦场景调用，观众上麦之前需先取消已订阅的直播间资源
 * 从 5.0.7 开始增加返回 `tracks` 与 `userIds`
 * userIds - 当前已加入房间的主播人员列表
 * tracks - 当前已发布至房间内的其他主播资源
 * 5.3.2 新增返回 PKRoomIds，为房间内已连麦的副房间 roomId 列表
 * @param roomId 房间 Id
 * @param livingType 直播类型
 * * 当 `livingType` 值为 `RCLivingType.AUDIO` 时表示开始音频直播
 * * 当 `livingType` 值为 `RCLivingType.VIDEO` 时表示开始音视频直播
 */
const { code, room, userIds, tracks: remoteTracks, PKRoomIds: string[] } = await
rtcClient.joinLivingRoom('roomId', RCLivingType.VIDEO)

// 若加入失败，则 room、userIds、tracks 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('join room failed:', code)
  return
}

// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
/**
 * 当本端被踢出房间
 * @description 被踢出房间可能是由于服务端超出一定时间未能收到 rtcPing 消息，所以认为己方离线。
 * 另一种可能是己方 rtcPing 失败次数超出上限，故而主动断线
 * @param byServer
 * 当值为 false 时，说明本端 rtcPing 超时
 * 当值为 true 时，说明本端收到被踢出房间通知
 * @param state 被踢出房间的原因
 */
onKickOff?(byServer: boolean, state?: RCKickReason): void
},
/**
 * 接收到房间信令时回调，用户可通过房间实例的 `sendMessage(name, content)` 接口发送信令
 * @param name 信令名
 */

```

```

* @param content 信令内容
* @param senderUserId 发送者 Id
* @param messageUId 消息唯一标识
*/
onMessageReceive (name: string, content: any, senderUserId: string, messageUId: string) {
},
/**
* 监听房间属性变更通知
* @param name
* @param content
*/
onRoomAttributeChange (name: string, content: string) {
},
/**
* 房间用户禁用/启用音频
* @param audioTrack RCRemoteAudioTrack 类实例
*/
onAudioMuteChange (audioTrack: RCRemoteAudioTrack) {
},
/**
* 房间用户禁用/启用视频
* @param videoTrack RCRemoteVideoTrack 类实例对象
*/
onVideoMuteChange (videoTrack: RCRemoteVideoTrack) {
},
/**
* 房间内用户发布资源
* @param tracks 新发布的音轨与视轨数据列表，包含新发布的 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
*/
async onTrackPublish (tracks: RCRemoteTrack[]) {
// 按业务需求选择需要订阅资源，通过 room.subscribe 接口进行订阅
const { code } = await room.subscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
* 房间用户取消发布资源
* @param tracks 被取消发布的音轨与视轨数据列表
* @description 当资源被取消发布时，SDK 内部会取消对相关资源的订阅，业务层仅需处理 UI 业务
*/
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
* 订阅的音视频流通道已建立，track 已可以进行播放
* @param track RCRemoteTrack 类实例
*/
onTrackReady (track: RCRemoteTrack) {
if (track.isAudioTrack()) {
// 音轨不需要传递播放控件
track.play()
} else {
// 视轨需要一个 video 标签才可进行播放
const element = document.createElement('video')
document.body.appendChild(element)
track.play(element)
}
},
/**
* 人员加入
* @param userIds 加入的人员 id 列表
*/
onUserJoin (userIds: string[]) {
},
/**
* 人员退出

```

```

* @param userIds
*/
onUserLeave (userIds: string[]) {
},
/**
* 房间内主播和观众切换身份 (@rongcloud/plugin-rtc@5.2.0 新增)
* @description @rongcloud/plugin-rtc@5.2.0 及其之后，
* 如业务层未传入 onSwitchRole 回调，“房间内观众升级为主播”通过 onUserJoin 通知，“主播降级为房间内的观众”通过
onUserLeave 通知；
* 业务层传入 onSwitchRole 时，“房间内主播和观众切换身份”通过 onSwitchRole 通知，onUserJoin 和 onUserLeave 仅通知
手动调用“加入或退出房间”的人员
* @param userId 用户 ID
* @param role 用户角色
* role 值为 RCRTCLiveRole.ANCHOR 时，代表房间内观众升级为主播
* role 值为 RCRTCLiveRole.AUDIENCE 时，代表主播降级为房间内的观众
*/
onSwitchRole (userId: string, role: RCRTCLiveRole) {
}
})

```

步骤 5.2: 捕获本地音视频资源

使用初始化时获取的 RCRTCClient 实例的 [createMicrophoneAndCameraTracks](#) 方法可同时捕获音视频流。

```

/**
* @description tracks 是一个数组，当 `code !== RCRTCCode.SUCCESS` 时，tracks 长度为 0
* @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器摄像头、麦克风资源，
* 也可传入其他包含 A-Z、a-z、0-9、+、=、- 的字符串，
* @param options 音视频配置项，可参考上述 1、2 中的介绍
*/
const { code, tracks } = await rtcClient.createMicrophoneAndCameraTracks(tag: string = 'RongCloudRTC',
options?: { audio?: IMicphoneAudioProfile, video?: ICameraVideoProfile })

if (code === RCRTCCode.SUCCESS) {
// tracks 包含一个 RCMicphoneAudioTrack 实例和一个 RCCameraVideoTrack 实例
const [ audioTrack, videoTrack ] = tracks
}

```

使用 [play](#) 方法在本端播放采集的数据。通常情况下，尽量不要在本端播放本端采集的音频流，否则可能会引起回声问题。

```

// 通过 videoTrack.play 方法将 <video> 标签传递给 videoTrack 实例
videoTrack.play(videoNode)

// 播放音频时无需传参，尽量不要在本端播放本端采集的音频流，因为可能会引起回声问题
audioTrack.play()

// 播放音频设置音量，volume 为 0-100 的数字
audioTrack.play(null, {volume})

// 指定播放音频的输出设备，输出音频设备列表可通过 device.getSpeakers() 获取
// device 可从 @rongcloud/plugin-rtc 模块导出
audioTrack.play(null, {audioDeviceId})

```

步骤 5.3: 发布资源

使用 [RCLivingRoom](#) 房间实例的 [publish](#) 方法发布一个或多个音视频轨道 [RCLocalTrack](#) 数据。

```
const { code } = await room.publish([audioTrack, videoTrack])

// 若资源发布失败
if (code !== RCRTCCode.SUCCESS) {
  console.log('资源发布失败:', code)
}
```

步骤 5.4: 资源订阅

使用 [RCLivingRoom](#) 房间实例的 [subscribe](#) 方法订阅视听资源。

```
const { code } = await room.subscribe([
  // 音频不支持大小流
  audioTrack,
  {
    track: videoTrack,
    subTiny: true // 订阅小流
  }
])
```

步骤 5.5: 退出房间

使用 [RCRTCClient](#) 实例的 [leaveRoom](#) 方法，传入 [RCLivingRoom](#) 房间实例退出该房间。

```
// room 为加入房间方法返回的实例对象
const { code } = await rtcClient.leaveRoom(room)
```

观众端

步骤 6.1: 加入房间

RTCLib 初始化后，可获取到 RTC 客户端实例。调用 [joinLivingRoomAsAudience](#) 方法可加入房间。

以下示例中 `rtcClient` 表示 RTC 客户端实例。加入房间成功后，会返回主播角色用户的 [RCAudienceLivingRoom](#) 房间实例、房间内其他用户 ID，以及房间内已发布的资源。

```
// 获取 RCLivingType 枚举定义
import { RCLivingType } from '@rongcloud/plugin-rtc'

/**
 * 观众加入直播房间调用
 * @param roomId 房间 Id
 * @param livingType 直播类型
 * * 当 `livingType` 值为 `RCLivingType.AUDIO` 是表示音频直播
 * * 当 `livingType` 值为 `RCLivingType.AUDIO_VIDEO` 是表示音视频直播
 */
// 从 5.2.3 开始，加入房间时可返回 RTCTracks、MCUTracks、CDNUrls、userIds
const { room: audienceRoom, RTCTracks, MCUTracks, CDNUrls, userIds, code } = await
rtcClient.joinLivingRoomAsAudience('roomId', RCLivingType.AUDIO_VIDEO)
```

```

// 若加入失败，则 room 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
console.log('join room as audience failed:', code)
}

// audienceRoom 为观众加入房间返回的 room
// 注册房间事件监听器，重复注册时，仅最后一次有效
audienceRoom.registerRoomEventListener({
/**
* 主播加入
* @param userIds 加入的主播 id 列表
*/
onAnchorJoin (userIds: string[]) {
},
/**
* 主播离开
* @param userIds 离开的主播 id 列表
*/
onAnchorLeave (userIds: string[]) {
},
/**
* 房间内直播合流资源发布
* @param track RCRemoteTrack 类实例
* * 房间类型以第一个加入房间用户设置的直播类型为准
* * 房间直播类型为 RCLivingType.AUDIO_VIDEO，tracks 包含 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
* * 直播类型为 RCLivingType.AUDIO，tracks 仅包含 RCRemoteAudioTrack 实例
* * 触发时机：主播发布资源后
*/
onTrackPublish (tracks: RCRemoteTrack[]) {
},
/**
* 房间内直播合流资源取消发布
* @param track RCRemoteTrack 类实例
* * 触发时机：全部主播退出房间（因资源为多个主播发布的合流资源，单个主播取消发布不会触发）
*/
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
* 房间内主播资源发布
* @param track RCRemoteTrack 类实例
* * 触发时机：主播发布资源后
*/
onAnchorTrackPublish (tracks: RCRemoteTrack[]){
},
/**
* 房间内主播资源取消发布
* @param track RCRemoteTrack 类实例
* * 触发时机：主播取消发布资源后
*/
onAnchorTrackUnpublish (tracks: RCRemoteTrack[]){
},
/**
* 订阅的音视频流通道已建立，track 已可以进行播放
* @param track RCRemoteTrack 类实例
*/
onTrackReady (track: RCRemoteTrack) {
// 订阅的音视频轨道已连接，可以根据业务需求选择性播放
if (track.isAudioTrack()) {
// 音频播放无需传递组件
track.play()
} else {
// 此处的 videoNode 为 <video> 标签元素实例
track.play(videoNode)
}
}
})

```

加入直播间后，直播间内可能已经存在主播发布的音视频轨数据，可以通过 `subscribe` 接口拉取这些声音或图像资源。

```
const { code } = await room.subscribe(remoteTracks)

if (code !== RCRTCCode.SUCCESS) {
  console.log(`资源订阅失败 -> code: ${code}`)
}

// 当资源订阅成功后，等待 onTrackReady 事件回调即可进行播放
```

步骤 6.2: 订阅资源

使用 [RCAudienceLivingRoom](#) 房间实例的 `subscribe` 方法订阅一个或多个音视频轨道 [RCLocalTrack](#) 数据。

```
const { code } = await audienceRoom.subscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
  console.log('资源订阅失败 ->', code)
}
```

步骤 6.3: 退出房间

使用 [RCRTCClient](#) 实例的 `leaveLivingRoomAsAudience` 方法，传入 [RCAudienceLivingRoom](#) 房间实例，退出该直播房间。

```
/**
 * @param audienceRoom 观众加入直播房间返回的 room
 */
const { code } = await rtcClient.leaveLivingRoomAsAudience(audienceRoom)
if (code !== RCRTCCode.SUCCESS) {
  console.log('join room as audience failed:', code)
}
```

设备管理

获取设备列表

更新时间:2024-08-30

导入 device 模块

```
import { device } from '@rongcloud/plugin-rtc'
```

获取麦克风设备列表

API 参考：[getMicrophones](#) [↗](#)

```
const microphones = await device.getMicrophones();
```

获取摄像头设备列表

API 参考：[getCameras](#) [↗](#)

```
const cameras = await device.getCameras();
```

获取扬声器设备列表

API 参考：[getSpeakers](#) [↗](#)

```
const speakers = await device.getSpeakers();
```

从指定设备获取音视频资源

如果已获取设备的 ID，例如 microphoneId、cameraId 等，可以使用以下方法从指定设备获取音视频资源。

从指定设备获取音频

获取音频 track 时可指定设备 ID。

API 参考：[createMicrophoneAudioTrack](#) [↗](#)

```
/**
 * @param tag 资源标识
 * @param IMicphoneAudioProfile.micphoneId 指定麦克风设备 Id
 */
const { code, track: audioTrack } = await rtcClient.createMicrophoneAudioTrack(tag: string = 'RongCloudRTC', options?: IMicphoneAudioProfile)
```

从指定设备获取视频

获取视频 track 时可指定设备 ID。

API 参考：[createCameraVideoTrack](#) [↗](#)

```
/**
 * @param tag 资源标识
 * @param ICameraVideoProfile.cameraId 指定摄像头设备 Id
 */
const { code, track: videoTrack } = await rtcClient.createCameraVideoTrack(tag: string = 'RongCloudRTC', options?: ICameraVideoProfile)
```

从指定设备同时获取音视频

同时获取音频 track 和视频 track 时可指定设备 ID。

API 参考：[createMicrophoneAndCameraTracks](#) [↗](#)

```
/**
 * @param tag 资源标识
 * @param IMicphoneAudioProfile.micphoneId 指定麦克风设备 Id
 * @param ICameraVideoProfile.cameraId 指定摄像头设备 Id
 */
const { code, tracks } = await rtcClient.createMicrophoneAndCameraTracks(tag: string = 'RongCloudRTC', options?: { audio?: IMicphoneAudioProfile, video?: ICameraVideoProfile })
```

基本操作

创建/加入房间

更新时间:2024-08-30

调用 `RCRTCClient` 实例的加入房间方法可以加入音视频房间。如果该房间之前不存在，则会在调用时自动创建并加入。以下示例中的 `rtcClient` 为 `RTCLib SDK` 初始化时获取的 `RCRTCClient` 实例。

提示

每个房间在创建之初，会由融云服务生成一个在用户全网唯一的 `SessionId`，可用于后台业务查询或与融云进行问题沟通。当房间内的所有人退出或被服务器判定掉线后，此 `Session` 结束。之后即便再用相同的 `RoomId` 创建房间，`SessionId` 也会更新为不同值。

会议模式下，使用 `RCRTCClient` 实例的方法 [joinRTCRoomWithOptions](#) 加入 RTC 会议房间。加入房间成功后，会返回 [RCRTCRoom](#) 房间实例、房间内其他用户 ID，以及房间内已发布的资源。

原 [joinRTCRoom](#) 接口自 5.7.0 版本开始废弃，建议使用 [joinRTCRoomWithOptions](#) 替代。

```
// 加入普通音视频房间，从 5.0.7 开始增加返回 `tracks` 与 `userIds`  
// * userIds - 当前已加入房间的远端人员列表  
// * tracks - 当前已发布至房间内的远端资源列表  
const { code, room, userIds, tracks: remoteTracks } = await rtcClient.joinRTCRoomWithOptions('roomId')  
  
// 若加入失败，则 room、userIds、tracks 值为 undefined  
if (code !== RCRTCCode.SUCCESS) {  
  console.log('join living room failed:', code)  
  return  
}
```

直播模式下，您需要根据用户当前角色使用不同处理方式。直播房间的媒体类型（音频/音视频）以第一个加入房间用户设置的为准。

- 主播角色：使用 `RCRTCClient` 实例的 [joinLivingRoom](#) 方法加入 RTC 直播房间。加入房间成功后可获取 [RCLivingRoom](#) 房间实例。

```

// 获取 RCLivingType 枚举定义
import { RCLivingType } from '@rongcloud/plugin-rtc'

/**
 * 主播加入直播房间或观众上麦场景调用，观众上麦之前需先取消已订阅的直播间资源
 * 从 5.0.7 开始增加返回 `tracks` 与 `userIds`
 * userIds - 当前已加入房间的主播人员列表
 * tracks - 当前已发布至房间内的其他主播资源
 * 5.3.2 新增返回 PKRoomIds，为房间内已连麦的副房间 roomId 列表
 * @param roomId 房间 Id
 * @param livingType 直播类型
 * * 当 `livingType` 值为 `RCLivingType.AUDIO` 时表示开始音频直播
 * * 当 `livingType` 值为 `RCLivingType.VIDEO` 时表示开始音视频直播
 */
const { code, room, userIds, tracks: remoteTracks, PKRoomIds: string[] } = await
rtcClient.joinLivingRoom('roomId', RCLivingType.VIDEO)

// 若加入失败，则 room、userIds、tracks 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('join room failed:', code)
  return
}

```

- 观众角色：使用 RCRTCClient 实例的 [joinLivingRoomAsAudience](#) 方法加入 RTC 直播房间。加入房间成功后可获取 [RCAudienceLivingRoom](#) 房间实例。

```

// 获取 RCLivingType 枚举定义
import { RCLivingType } from '@rongcloud/plugin-rtc'

/**
 * 观众加入直播房间调用
 * @param roomId 房间 Id
 * @param livingType 直播类型
 * * 当 `livingType` 值为 `RCLivingType.AUDIO` 是表示音频直播
 * * 当 `livingType` 值为 `RCLivingType.AUDIO_VIDEO` 是表示音视频直播
 */
// 从 5.2.3 开始，加入房间时可返回 RTCTracks、MCUTracks、CDNUrls、userIds
const { room: audienceRoom, RTCTracks, MCUTracks, CDNUrls, userIds, code } = await
rtcClient.joinLivingRoomAsAudience('roomId', RCLivingType.AUDIO_VIDEO)
// 若加入失败，则 room 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('join room as audience failed:', code)
}

```

获取房间数据

会议模式下，加入房间成功后可获取 [RCRTCRoom](#) 实例。直播模式下，主播角色用户加入房间成功后可获取 [RCLivingRoom](#) 房间实例。

[RCRTCRoom](#) 与 [RCLivingRoom](#) 实例上提供以下获取资源与数据的方法：

方法	返回值类型	说明
getRoomId()	String	返回房间 ID。
getSessionId()	String	返回房间的 Session ID。重新加入房间时，该 ID 会重置。
getLocalTracks()	RCLocalTrack[]	返回已发布资源列表（从 RTCLib 5.0.5 开始支持）。
getRemoteUserIds()	string[]	返回房间中已存在的远端用户列表，不包含本端 userId
getRemoteTracks()	RCRemoteTrack[]	返回远端发布的所有资源列表（从 RTCLib 5.0.7 开始支持）。
getRemoteTracksByUserId()	RCRemoteTrack[]	返回指定用户在房间内发布的资源列表（从 RTCLib 5.0.7 开始支持）。

在直播模式下，观众角色用户加入房间成功后返回 [RCAudienceLivingRoom](#) 房间实例。观众可以使用 RCAudienceLivingRoom 房间实例获取观众专用部分资源及数据。

方法	返回值类型	说明
getRoomId()	String	返回房间 ID。
getSessionId()	String	返回房间的 Session ID。重新加入房间时，该 ID 会重置。
getRemoteUserIds()	string[]	返回房间中已存在的主播角色用户列表，不包含本端 userId。
getRemoteTracks()	RCRemoteTrack[]	返回远端发布的所有资源列表（从 RTCLib 5.0.7 开始支持）。
getRemoteTracksByUserId()	RCRemoteTrack[]	返回指定用户在房间内发布的资源列表（从 RTCLib 5.0.7 开始支持）。在直播模式下，返回的列表中既包含分流也包含服务端合流。
getRemoteRTCTracks()	RCRemoteTrack[]	返回房间内远端主播角色用户发布的所有分流资源列表（从 RTCLib 5.2.3 开始支持）。
getRemoteMCUTracks()	RCRemoteTrack[]	返回房间内远端主播角色用户发布的所有合流资源列表（从 RTCLib 5.2.3 开始支持）。

离开房间

会议模式下的所有参会人员使用 `leaveRoom` 方法退出房间。直播模式下主播角色用户也使用相同方法退出房间。

```
const { code } = await rtcClient.leaveRoom(room)
```

在直播模式下，观众角色的用户必须使用 `leaveLivingRoomAsAudience` 方法退出房间。

```
/*
 * @param audienceRoom 观众加入直播房间返回的 room
 */
const { code } = await rtcClient.leaveLivingRoomAsAudience(audienceRoom)
if (code !== RCRTCCode.SUCCESS) {
  console.log('join room as audience failed:', code)
}
```

房间事件回调

更新时间:2024-08-30

成功获取房间实例后，可以使用 `registerRoomEventListener` 添加监听器，在房间内远端用户的状态及资源变化时接收通知。

会议模式与直播模式下的房间实例类型有差异：

- 会议模式：所有参会人员均需要获取 [RCRTCRoom](#) 实例。
- 直播模式：主播角色用户使用 [RCLivingRoom](#) 实例、观众角色用户使用 [RCAudienceLivingRoom](#) 实例。

会议模式下监听房间事件

获取 [RCRTCRoom](#) 实例后，调用 `registerRoomEventListener` 注册房间事件监听器 [IRoomEventListener](#)。重复注册时会导致后者覆盖前者，仅最后一次注册有效。可以使用 `registerRoomEventListener(null)` 取消注册。

以下示例展示了部分回调方法。完整列表请参见 [IRoomEventListener](#)。

```
// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
/**
 * 本端被踢出房间时触发
 * @description 被踢出房间可能是由于服务端超出一定时间未能收到 rtcPing 消息，所以认为己方离线。
 * 另一种可能是己方 rtcPing 失败次数超出上限，故而主动断线
 * @param byServer
 * 当值为 false 时，说明本端 rtcPing 超时
 * 当值为 true 时，说明本端收到被踢出房间通知
 */
onKickOff (byServer: boolean) {
// 当本地已获取资源后，需要调用 track.destroy() 销毁已获取的资源，track 为 RCMicphoneAudioTrack 或
RCCameraVideoTrack 类型实例
},
/**
 * 接收到房间信令时回调，用户可通过房间实例的 `sendMessage(name, content)` 接口发送信令
 * @param name 信令名
 * @param content 信令内容
 * @param senderUserId 发送者 Id
 * @param messageUId 消息唯一标识
 */
onMessageReceive (name: string, content: any, senderUserId: string, messageUId: string) {
},
/**
 * 监听房间属性变更通知
 * @param name
 * @param content
 */
onRoomAttributeChange (name: string, content: string) {
},
/**
 * 发布者禁用/启用音频
 * @param audioTrack RCRemoteAudioTrack 类实例
 */
}
```

```

onAudioMuteChange (audioTrack: RCRemoteAudioTrack) {
},
/**
 * 发布者禁用/启用视频
 * @param videoTrack RCRemoteVideoTrack 类实例对象
 */
onVideoMuteChange (videoTrack: RCRemoteVideoTrack) {
},
/**
 * 房间内其他用户新发布资源时触发
 * 如需获取加入房间之前房间内某个用户发布的资源列表，可使用 room.getRemoteTracksByUserId('userId') 获取
 * @param tracks 新发布的音轨与视轨数据列表，包含新发布的 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
 */
onTrackPublish (tracks: RCRemoteTrack[]) {
// 按业务需求选择需要订阅资源，通过 room.subscribe 接口进行订阅
const { code } = await room.subscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
 * 房间用户取消发布资源
 * @param tracks 被取消发布的音轨与视轨数据列表
 * @description 当资源被取消发布时，SDK 内部会取消对相关资源的订阅，业务层仅需处理 UI 业务
 */
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
onTrackReady (track: RCRemoteTrack) {
if (track.isAudioTrack()) {
// 音轨不需要传递播放控件
track.play()
} else {
// 视轨需要一个 video 标签才可进行播放
const element = document.createElement('video')
document.body.appendChild(element)
track.play(element)
}
},
/**
 * 人员加入
 * @param userIds 加入的人员 id 列表
 */
onUserJoin (userIds: string[]) {
},
/**
 * 人员退出
 * @param userIds
 */
onUserLeave (userIds: string[]) {
}
})

```

直播模式下主播用户监听房间事件

获取 [RCLivingRoom](#) 实例后，调用 [registerRoomEventListener](#) 注册房间事件监听器 [IRoomEventListener](#)。重复注册时会导致后者覆盖前者，仅最后一次注册有效。可以使用 [registerRoomEventListener\(null\)](#) 取消注册。

以下示例展示了部分回调方法。完整列表请参见 [IRoomEventListener](#)。

```
// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
/**
 * 当本端被踢出房间
 * @description 被踢出房间可能是由于服务端超出一定时间未能收到 rtcPing 消息，所以认为己方离线。
 * 另一种可能是己方 rtcPing 失败次数超出上限，故而主动断线
 * @param byServer
 * 当值为 false 时，说明本端 rtcPing 超时
 * 当值为 true 时，说明本端收到被踢出房间通知
 * @param state 被踢出房间的原因
 */
onKickOff? (byServer: boolean, state?: RCKickReason): void
},
/**
 * 接收到房间信令时回调，用户可通过房间实例的 `sendMessage(name, content)` 接口发送信令
 * @param name 信令名
 * @param content 信令内容
 * @param senderUserId 发送者 Id
 * @param messageUId 消息唯一标识
 */
onMessageReceive (name: string, content: any, senderUserId: string, messageUId: string) {
},
/**
 * 监听房间属性变更通知
 * @param name
 * @param content
 */
onRoomAttributeChange (name: string, content: string) {
},
/**
 * 房间用户禁用/启用音频
 * @param audioTrack RCRemoteAudioTrack 类实例
 */
onAudioMuteChange (audioTrack: RCRemoteAudioTrack) {
},
/**
 * 房间用户禁用/启用视频
 * @param videoTrack RCRemoteVideoTrack 类实例对象
 */
onVideoMuteChange (videoTrack: RCRemoteVideoTrack) {
},
/**
 * 房间内用户发布资源
 * @param tracks 新发布的音轨与视轨数据列表，包含新发布的 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
 */
async onTrackPublish (tracks: RCRemoteTrack[]) {
// 按业务需求选择需要订阅资源，通过 room.subscribe 接口进行订阅
const { code } = await room.subscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
 * 房间用户取消发布资源
 * @param tracks 被取消发布的音轨与视轨数据列表
 * @description 当资源被取消发布时，SDK 内部会取消对相关资源的订阅，业务层仅需处理 UI 业务
 */
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
}
```

```

*/
onTrackReady (track: RCRemoteTrack) {
if (track.isAudioTrack()) {
// 音轨不需要传递播放控件
track.play()
} else {
// 视轨需要一个 video 标签才可进行播放
const element = document.createElement('video')
document.body.appendChild(element)
track.play(element)
}
},
/**
* 人员加入
* @param userIds 加入的人员 id 列表
*/
onUserJoin (userIds: string[]) {
},
/**
* 人员退出
* @param userIds
*/
onUserLeave (userIds: string[]) {
},
/**
* 房间内主播和观众切换身份 (@rongcloud/plugin-rtc@5.2.0 新增)
* @description @rongcloud/plugin-rtc@5.2.0 及其之后，
* 如业务层未传入 onSwitchRole 回调，“房间内观众升级为主播”通过 onUserJoin 通知，“主播降级为房间内的观众”通过
onUserLeave 通知；
* 业务层传入 onSwitchRole 时，“房间内主播和观众切换身份”通过 onSwitchRole 通知，onUserJoin 和 onUserLeave 仅通知
手动调用“加入或退出房间”的人员
* @param userId 用户 ID
* @param role 用户角色
* role 值为 RCRTCLiveRole.ANCHOR 时，代表房间内观众升级为主播
* role 值为 RCRTCLiveRole.AUDIENCE 时，代表主播降级为房间内的观众
*/
onSwitchRole (userId: string, role: RCRTCLiveRole) {
}
})

```

直播模式下观众角色用户监听房间事件

获取 [RCAudienceLivingRoom](#) 实例后，调用 `registerRoomEventListener` 注册观众房间事件监听器 [IAudienceRoomEventListener](#)。重复注册时会导致后者覆盖前者，仅最后一次注册有效。可以使用 `registerRoomEventListener(null)` 取消注册。

以下示例展示了部分回调方法。完整列表请参见 [IAudienceRoomEventListener](#)。

```

// audienceRoom 为观众加入房间返回的 room
// 注册房间事件监听器，重复注册时，仅最后一次有效
audienceRoom.registerRoomEventListener({
/**
 * 主播加入
 * @param userIds 加入的主播 id 列表
 */
onAnchorJoin (userIds: string[]) {
},
/**
 * 主播离开
 * @param userIds 离开的主播 id 列表
 */
onAnchorLeave (userIds: string[]) {
},
/**
 * 房间内直播合流资源发布
 * @param track RCRemoteTrack 类实例
 * * 房间类型以第一个加入房间用户设置的直播类型为准
 * * 房间直播类型为 RCLivingType.AUDIO_VIDEO ，tracks 包含 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
 * * 直播类型为 RCLivingType.AUDIO ， tracks 仅包含 RCRemoteAudioTrack 实例
 * * 触发时机：主播发布资源后
 */
onTrackPublish (tracks: RCRemoteTrack[]) {
},
/**
 * 房间内直播合流资源取消发布
 * @param track RCRemoteTrack 类实例
 * * 触发时机：全部主播退出房间（因资源为多个主播发布的合流资源，单个主播取消发布不会触发）
 */
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
 * 房间内主播资源发布
 * @param track RCRemoteTrack 类实例
 * * 触发时机：主播发布资源后
 */
onAnchorTrackPublish (tracks: RCRemoteTrack[]){
},
/**
 * 房间内主播资源取消发布
 * @param track RCRemoteTrack 类实例
 * * 触发时机：主播取消发布资源后
 */
onAnchorTrackUnpublish (tracks: RCRemoteTrack[]){
},
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
onTrackReady (track: RCRemoteTrack) {
// 订阅的音视频轨道已连接，可以根据业务需求选择性播放
if (track.isAudioTrack()) {
// 音频播放无需传递组件
track.play()
} else {
// 此处的 videoNode 为 <video> 标签元素实例
track.play(videoNode)
}
}
})

```

自定义房间属性

更新时间:2024-08-30

提示

以下示例代码中的 `room` 指加入房间成功后获取到的实例。

房间属性变化通知

加入房间成功后，可以通过调用 `room.registerRoomEventListener()` 注册 `onRoomAttributeChange` 事件监听器。

当房间内成员对属性进行修改、删除，且选择发送通知时，注册的回调函数会被调用。

设置房间属性

提示

向房间内设置数据时不区分成员，若多人设置相同的 `key` 会相互覆盖。

API 参考：[setRoomAttribute](#)

```
// 设置房间属性，不发送通知
const { code } = await room.setRoomAttribute(key, value)

// 设置房间属性，并发送通知
const { code } = await room.setRoomAttribute(key, value, { name: '', content: '' })
```

获取房间属性

API 参考：[getRoomAttributes](#)

```
// 获取房间属性，支持同时获取多个属性
const { code, data } = await room.getRoomAttributes(['key_1', 'key2'])

// 当获取失败时，data 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.error('房间属性获取失败:', code)
  return
}

// 获取到的值以 `key-value` 键值对的方式挂载于 data 数据中
const data_1 = data['key_1']
const data_2 = data['key_2']
```

删除房间属性

API 参考：[deleteRoomAttributes](#) [↗](#)

```
// 删除房间属性，不发送通知
const { code } = await room.deleteRoomAttributes(['key_1', 'key_2'])

// 删除房间属性，并发送通知
const { code } = await room.deleteRoomAttributes(['key_1', 'key_2'], { name: '', content: '' })
```

本地用户流

更新时间:2024-08-30

本文描述了在会议模式下获取、发布默认音视频资源的流程。

获取音视频资源

您可以使用以下方法从本地摄像头、麦克风设备捕获默认音频 track、视频 track 资源。调用以下获取资源的 API，SDK 内部会把浏览器原生 API 获取的 mediaStream 对象转换为 RCLocalTrack 实例对象，用于后续集成。

提示

如果从指定设备获取音视频数据，通过 @rongcloud/plugin-rtc 的 device 模块获取设备 ID。详见设备管理文档。如果需要从本地文件、网络文件、或浏览器原生的 mediaStream 对象获取屏幕共享资源、获取自定义资源，请参考「视频管理」下的发布自定义视频流文档。

使用初始化时获取的 RCRTCClient 实例的 createMicrophoneAndCameraTracks 方法可同时捕获音视频流。

```
/**
 * @description tracks 是一个数组，当 `code !== RCRTCCode.SUCCESS` 时，tracks 长度为 0
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器摄像头、麦克风资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、- 的字符串，
 * @param options 音视频配置项，可参考上述 1、2 中的介绍
 */
const { code, tracks } = await rtcClient.createMicrophoneAndCameraTracks(tag: string = 'RongCloudRTC',
options?: { audio?: IMicphoneAudioProfile, video?: ICameraVideoProfile })

if (code === RCRTCCode.SUCCESS) {
// tracks 包含一个 RCMicphoneAudioTrack 实例和一个 RCCameraVideoTrack 实例
const [ audioTrack, videoTrack ] = tracks
}
```

使用初始化时获取的 RCRTCClient 实例的 createMicrophoneAudioTrack 由麦克风捕获音频流：

```
/**
 * @description 仅当 `code === RCRTCCode.SUCCESS` 时 audioTrack 有值，
 * audioTrack 为 RCMicphoneAudioTrack 类型实例
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器麦克风资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、- 的字符串，
 * @param IMicphoneAudioProfile 音频配置项，可选参数
 * @param IMicphoneAudioProfile.micphoneId 指定麦克风设备 Id
 * @param IMicphoneAudioProfile.sampleRate 指定音频采样率
 */
const { code, track: audioTrack } = await rtcClient.createMicrophoneAudioTrack(tag: string =
'RongCloudRTC', options?: IMicphoneAudioProfile)
```

使用初始化时获取的 `RCRTCClient` 实例的 `createCameraVideoTrack` 方法由摄像头捕获视频流。如需设置视频帧率、分辨率，可参考「视频管理」下的文档。

```
/**
 * @description 仅当 `code === RCRTCCode.SUCCESS` 时 videoTrack 有值
 * videoTrack 为 RCCameraVideoTrack 类型实例
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器摄像头资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、- 的字符串，
 * @param ICameraVideoProfile 视频配置项，可选参数
 * @param ICameraVideoProfile.cameraId 指定摄像头设备 Id
 * @param ICameraVideoProfile.frameRate 指定视频帧率，默认为 RCFramerate.FPS_15
 * @param ICameraVideoProfile.resolution 指定视频分辨率，默认为 RCRResolution.W640_H480
 */
const { code, track: videoTrack } = await rtcClient.createCameraVideoTrack(tag: string = 'RongCloudRTC',
options?: ICameraVideoProfile)
```

本地预览时，请使用 `play` 方法在本端播放采集的数据。通常情况下，尽量不要在本端播放本端采集的音频流，否则可能会引起回声问题。

```
// 通过 videoTrack.play 方法将 <video> 标签传递给 videoTrack 实例
videoTrack.play(videoNode)
// 播放音频时无需传参，尽量不要在本端播放本端采集的音频流，因为可能会引起回声问题
audioTrack.play()
```

发布资源

在会议模式下，使用 `RCRTCClient` 实例的方法 `joinRTCRoomWithOptions` 加入 RTC 会议房间。加入房间成功后，会返回 `RCRTCRoom` 房间实例、房间内其他用户 ID，以及房间内已发布的资源。

原 `joinRTCRoom` 接口自 5.7.0 版本开始废弃，建议使用 `joinRTCRoomWithOptions` 替代。

使用 `RCRTCRoom` 房间实例的 `publish` 方法发布一个或多个音视频轨道 `RCLocalTrack` 数据。

```
const { code } = await room.publish([audioTrack, videoTrack])

// 若资源发布失败
if (code !== RCRTCCode.SUCCESS) {
  console.log('资源发布失败:', code)
}
```

`publish` 方法的输入参数 `tracks` 接收 `RCLocalTrack` 或者 `IPublishAttrs` 数组。如果需要同时发布视频大小流，请使用 `IPublishAttrs` 数组，并在其中设置需要发布小流的视频轨道。

```
const { code } = await room.publish([
  {
    track: videoTrack,
    pubTiny: true // pubTiny 用于指定同时发布视频数据的同时，额外发布一个小流数据
  }
])
```

取消发布

使用 [RCRTCRoom](#) 房间实例的 [unpublish](#) 方法取消发布资源。

```
const { code } = await room.unpublish([audioTrack, videoTrack])

if (code !== RCRTCCode.SUCCESS) {
  console.log('取消发布失败:', code)
}

// 取消发布后，业务层若不再需要播放资源，可调 destroy 方法销毁资源
// audioTrack.destroy()
// videoTrack.destroy()
```

远端用户流

更新时间:2024-08-30

在会议模式下，参会用户进入音视频房间后，如果想看到别人的画面、听见别人的声音，需要订阅 (Subscribe) 其他人已发布的资源。

SDK 提供两套操作远端用户流的方法，适用于不同业务需求。

- 普通订阅、取消订阅方法：[subscribe](#) / [unsubscribe](#)
- (要求 SDK 版本 \geq 5.4.4) 支持任务队列、任务自动合并的订阅、取消订阅方法，针对频繁订阅或者取消远端用户流的操作进行了优化：[addSubscribeTask](#) / [addUnsubscribeTask](#) / [addUpdateSubscribeListTask](#)

订阅资源的时机

通过此方法订阅房间内其他用户的资源。订阅一般需要在两个地方进行处理：

- 加入房间成功后，会返回 [RCRTCRoom](#) 房间实例、房间内其他用户 ID，以及房间内已发布的资源。此时可进行订阅房间内已发布的资源。
- 加入房间后，通过房间实例注册房间事件监听器 [IRoomEventListener](#)。房间内 `onTrackPublish` 事件被触发时，即可按需订阅对方所发布的资源。

```

// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
/**
 * 房间内用户发布资源
 * @param tracks 新发布的音轨与视轨数据列表，包含新发布的 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
 */
async onTrackPublish (tracks: RCRemoteTrack[]) {
// 按业务需求选择需要订阅资源，通过 room.subscribe 接口进行订阅
const { code } = await room.subscribe(tracks)

if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
onTrackReady (track: RCRemoteTrack) {
// 订阅的音视频轨道已连接，可以根据业务需求选择性播放
if (track.isAudioTrack()) {
// 音频播放无需传递组件
track.play()
} else {
// 此处的 videoNode 为 <video> 标签元素实例
track.play(videoNode)
}
}
})

```

订阅资源

调用 [subscribe](#) 方法订阅视听轨资源。

视频流支持订阅大小流机制。为了节省下行带宽，可选择订阅对方的视频小流。仅在其他用户发布资源时设置了发布大小流的情况下可订阅小流。若对方并未发布小流，接收的仍然是对方的大流数据。

```

const { code } = await room.subscribe([
// 音频不支持大小流
audioTrack,
{
track: videoTrack,
subTiny: true // 订阅小流
}
])

```

取消订阅

调用 [unsubscribe](#) 方法取消订阅视听轨资源。

```
const { code } = await room.unsubscribe([audioTrack, videoTrack])
```

```
// 取消订阅完成后，业务层移除相关的 UI 信息即可
```

主播端 获取音视频资源

更新时间:2024-08-30

您可以使用以下方法从本地摄像头、麦克风设备捕获默认音频 track、视频 track 资源。调用以下获取资源的 API，SDK 内部会把浏览器原生 API 获取的 mediaStream 对象转换为 RCLocalTrack 实例对象，用于后续集成。

提示

如果从指定设备获取音视频数据，通过 `@rongcloud/plugin-rtc` 的 `device` 模块获取设备 ID。详见设备管理文档。如果需要从本地文件、网络文件、或浏览器原生的 `mediaStream` 对象获取屏幕共享资源、获取自定义资源，请参考「视频管理」下的发布自定义视频流文档。

使用 `createMicrophoneAndCameraTracks` 方法可同时捕获音视频流。

```
/**
 * @description tracks 是一个数组，当 `code !== RCRTCCode.SUCCESS` 时，tracks 长度为 0
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器摄像头、麦克风资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、- 的字符串，
 * @param options 音视频配置项，可参考上述 1、2 中的介绍
 */
const { code, tracks } = await rtcClient.createMicrophoneAndCameraTracks(tag: string = 'RongCloudRTC',
options?: { audio?: IMicphoneAudioProfile, video?: ICameraVideoProfile })

if (code === RCRTCCode.SUCCESS) {
// tracks 包含一个 RCMicphoneAudioTrack 实例和一个 RCCameraVideoTrack 实例
const [ audioTrack, videoTrack ] = tracks
}
```

使用 `createMicrophoneAudioTrack` 由麦克风捕获音频流：

```
/**
 * @description 仅当 `code === RCRTCCode.SUCCESS` 时 audioTrack 有值，
 * audioTrack 为 RCMicphoneAudioTrack 类型实例
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器麦克风资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、- 的字符串，
 * @param IMicphoneAudioProfile 音频配置项，可选参数
 * @param IMicphoneAudioProfile.micphoneId 指定麦克风设备 Id
 * @param IMicphoneAudioProfile.sampleRate 指定音频采样率
 */
const { code, track: audioTrack } = await rtcClient.createMicrophoneAudioTrack(tag: string =
'RongCloudRTC', options?: IMicphoneAudioProfile)
```

使用 `createCameraVideoTrack` 方法由摄像头捕获视频流。如需设置视频帧率、分辨率，可参考「视频管理」下的文档。

```

/**
 * @description 仅当 `code === RCRTCCode.SUCCESS` 时 videoTrack 有值
 * videoTrack 为 RCCameraVideoTrack 类型实例
 * @param tag 资源标识，不传时默认为 RongCloudRTC，代表浏览器摄像头资源，
 * 也可传入其他包含 A-Z、a-z、0-9、+、=、\、- 的字符串，
 * @param ICameraVideoProfile 视频配置项，可选参数
 * @param ICameraVideoProfile.cameraId 指定摄像头设备 Id
 * @param ICameraVideoProfile.frameRate 指定视频帧率，默认为 RCFrameRate.FPS_15
 * @param ICameraVideoProfile.resolution 指定视频分辨率，默认为 RCRResolution.W640_H480
 */
const { code, track: videoTrack } = await rtcClient.createCameraVideoTrack(tag: string = 'RongCloudRTC',
options?: ICameraVideoProfile)

```

本地预览时，请使用 [play](#) 方法在本端播放采集的数据。通常情况下，尽量不要在本端播放本端采集的音频流，否则可能会引起回声问题。

```

// 通过 videoTrack.play 方法将 <video> 标签传递给 videoTrack 实例
videoTrack.play(videoNode)
// 播放音频时无需传参，尽量不要在本端播放本端采集的音频流，因为可能会引起回声问题
audioTrack.play()

```

发布资源

使用 [publish](#) 方法发布一个或多个音视频轨道 [RCLocalTrack](#) 数据。

```

const { code } = await room.publish([audioTrack, videoTrack])

// 若资源发布失败
if (code !== RCRTCCode.SUCCESS) {
  console.log('资源发布失败:', code)
}

```

`publish` 方法的输入参数 `tracks` 接收 [RCLocalTrack](#) 或者 [IPublishAttrs](#) 数组。如果需要同时发布视频大小流，请使用 [IPublishAttrs](#) 数组，并在其中设置需要发布小流的视频轨道。

```

const { code } = await room.publish([
  {
    track: videoTrack,
    pubTiny: true // pubTiny 用于指定同时发布视频数据的同时，额外发布一个小流数据
  }
])

```

取消发布

使用 [unpublish](#) 方法取消发布资源。

```
const { code } = await room.unpublish([audioTrack, videoTrack])

if (code !== RCRTCCode.SUCCESS) {
  console.log('取消发布失败:', code)
}

// 取消发布后，业务层若不再需要播放资源，可调 destroy 方法销毁资源
// audioTrack.destroy()
// videoTrack.destroy()
```

订阅资源

1. 由房间注册的事件监听器中，监听到 onTrackPublish 事件被触发时，即可按需订阅对方所发布的资源。

API 参考：[registerRoomEventListener](#) [↗](#)

```
// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
  /**
   * 房间内用户发布资源
   * @param tracks 新发布的音轨与视轨数据列表，包含新发布的 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
   */
  async onTrackPublish (tracks: RCRemoteTrack[]) {
    // 按业务需求选择需要订阅资源，通过 room.subscribe 接口进行订阅
    const { code } = await room.subscribe(tracks)

    if (code !== RCRTCCode.SUCCESS) {
      console.log('资源订阅失败 ->', code)
    }
  },
  /**
   * 订阅的音视频流通道已建立，track 已可以进行播放
   * @param track RCRemoteTrack 类实例
   */
  onTrackReady (track: RCRemoteTrack) {
    // 订阅的音视频轨道已连接，可以根据业务需求选择性播放
    if (track.isAudioTrack()) {
      // 音频播放无需传递组件
      track.play()
    } else {
      // 此处的 videoNode 为 <video> 标签元素实例
      track.play(videoNode)
    }
  }
})
```

2. 视频流支持订阅大小流机制。为了节省下行带宽，可选择订阅对方的视频小流。

提示

订阅小流仅在对方发布了小流的情况下有效，若对方并未发布小流，接收的仍然是对方的大流数据。

API 参考：[subscribe](#) [↗](#)

```
const { code } = await room.subscribe([
// 音频不支持大小流
audioTrack,
{
track: videoTrack,
subTiny: true // 订阅小流
}
])
```

取消订阅

使用 [unsubscribe](#) 取消订阅。取消完成后，业务层移除相关的 UI 信息即可。

```
const { code } = await room.unsubscribe([audioTrack, videoTrack])
// 取消订阅完成后，业务层移除相关的 UI 信息即可
```

合流布局

合流布局

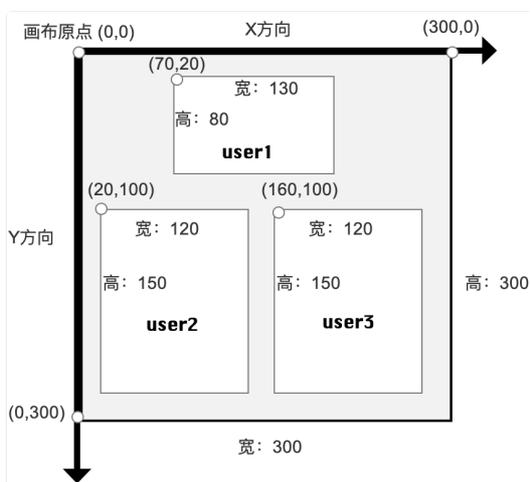
更新时间:2024-08-30

直播合流视频布局目前分为 3 种：1. 自定义布局；2. 悬浮布局；3. 自适应布局。下面分别介绍布局效果。

自定义布局

通过调用自定义布局接口可以设置合流视频整体尺寸，以及各个连麦者视图位置及大小。

合流布局以像素方式定义视频输出尺寸。如下图所示，整体视频尺寸为宽*高 = 300 * 300；以整体作为画布，画布的原点 (0,0) 在左上角，那么三个连麦用户窗口相对原点的位置、宽度、高度值分别如图所示（如需相应的设置代码，可在直播布局的接口文档中查看布局实例代码）：



悬浮布局

背景视频默认采用第一个加入房间的用户发布的视频 或合流布局接口指定的 `hostUserId` 发布的视频。显示区域为整个合流视频。合流视频大小需要调用接口设置 `mediaConfig`（默认值是 640 * 480）；

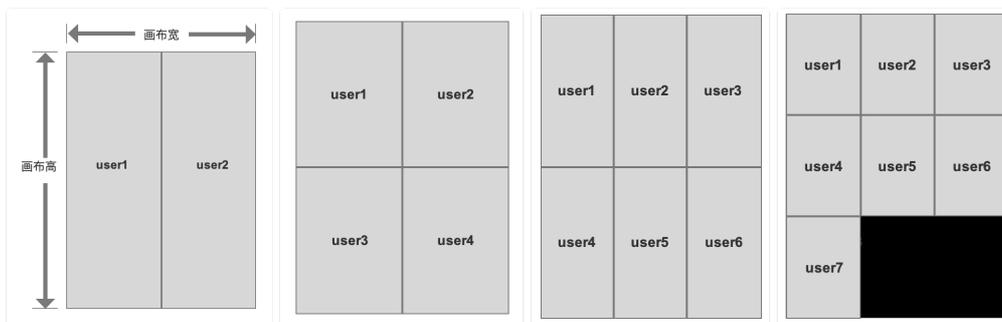
当连麦者依次加入时，按照下图显示的序列加载子视图；当有人离开时，系统会自动按照现有用户加入房间的次序重新布局视图。



自适应布局

视频的整体大小为默认值 $640 * 480$ 或通过合流接口自定义;当有多人加入房间后，直播系统会按照具体人数平分整体视频区域，使之每个子视图加载区域大小一致；录制系统会按照参会者进入房间的次序依次把参会者图像加载在示意图的相应序号上；

当有人离开时，系统会自动按照现有用户加入房间的次序重新布局视图



提示

以下示例代码中的 `room` 为主播加入直播房间成功后获取到的 `RCLivingRoom` 类型实例对象。

构建布局

App 可以调用 `RCMCUConfigBuilder` 中的接口配置 MCU 合流布局、视频水印、和 CDN 推流。

注意，通过 `RCMCUConfigBuilder` 完成配置后，需要调用 `flush()` 方法将配置同步到服务器方能生效。参考[使配置生效](#)。

获取 MCU 配置构建器实例

`RCMCUConfigBuilder` 是一个构建器类型，该类实例中包含了构建合流布局配置、水印配置 和 CDN 推流配置（详见[客户端旁路推流](#)）所需要的所有接口，支持链式调用。

API 参考：[getMCUConfigBuilder](#)

```
import { RCMCUConfigBuilder } from '@rongcloud/plugin-rtc'

// 获取构建器实例
const builder: RCMCUConfigBuilder = room.getMCUConfigBuilder()
```

修改布局模式

API 参考：

- [setMixLayoutMode](#)
- [MixLayoutMode](#)

```
import { MixLayoutMode } from '@rongcloud/plugin-rtc'
/**
 * 设置合流布局模式，当使用 `MixLayoutMode.CUSTOMIZE` 模式时，需自定义合流后的输出流布局结构
 * @param mode
 * * `MixLayoutMode.CUSTOMIZE`：自定义布局，需用户设置布局结构
 * * `MixLayoutMode.SUSPENSION`：悬浮布局（默认）
 * * `MixLayoutMode.ADAPTATION`：自适应布局
 */
builder.setMixLayoutMode(MixLayoutMode.SUSPENSION)
```

设置主视频流

API 参考：[setHostVideoTrack](#)

```
builder.setHostVideoTrack(videoTrack.getTrackId())
```

视频流配置

 提示

即合流后观众端可观看到的视频质量相关参数

API 参考：

- [setOutputVideoRenderMode](#)
- [setOutputVideoResolution](#)
- [setOutputVideoFPS](#)
- [setOutputVideoBitrate](#)

```
import { MixVideoRenderMode, RCResolution, RCFrameRate } from '@rongcloud/plugin-rtc'

builder
// 设置合流后的视频流渲染方式，默认值为 `MixVideoRenderMode.CROP`
// `MixVideoRenderMode.CROP`：当画布尺寸与视频分辨率比例不同时，裁剪视频内容
// `MixVideoRenderMode.WHOLE`：当画布尺寸与视频分辨率不同时，压缩视频尺寸以填充画布
.setOutputVideoRenderMode(MixVideoRenderMode.CROP)
// 设置视频分辨率
.setOutputVideoResolution(RCResolution.W640_H480)
// 设置视频帧率
.setOutputVideoFPS(RCFrameRate.FPS_15)
// 设置视频码率（不推荐主动设置，可能影响视频质量）
.setOutputVideoBitrate(bitrate)
```

视频小流配置

① 提示

即合流后观众端可观看到的视频小流质量相关参数

API 参考：

- [setOutputTinyVideoResolution](#) 
- [setOutputTinyVideoFPS](#) 
- [setOutputTinyVideoBitrate](#) 

```
builder
// 设置视频分辨率
.setOutputTinyVideoResolution(RCResolution.W176_H132)
// 设置视频帧率
.setOutputTinyVideoFPS(RCFrameRate.FPS_10)
// 设置视频码率（不推荐主动设置，可能影响视频质量）
.setOutputTinyVideoBitrate(bitrate)
```

音频流配置

① 提示

即合流后观众端接收到的音频质量相关参数

API 参考：[setOutputAudioBitrate](#) 

```
// 设置音频码率（不推荐主动设置，可能影响声音质量）
builder.setOutputAudioBitrate(bitrate)
```

视频流背景图

① 提示

向输出的视频中增加背景图、背景色等信息。

API 参考：

- [setOutputBackgroundPictureFillMode](#) 
- [setOutputBackgroundColor](#) 
- [addOutputBackgroundPicture](#) 
- [removeOutputBackgroundPicture](#) 
- [clearOutputBackgroundPicture](#) 

```
import { BackgroundPictureFillMode } from '@rongcloud/plugin-rtc'

builder
/**
 * 设置合流后的视频流中添加的背景图片的填充方式，默认为 `BackgroundPictureFillMode.CROP`
 * BackgroundPictureFillMode.CROP: 按比例裁剪
 * BackgroundPictureFillMode.WHOLE: 不裁剪，按比例压缩
 * @param fillMode
 */
.setOutputBackgroundPictureFillMode(BackgroundPictureFillMode.CROP)
/**
 * 设置合流后的视频流的背景色，默认为 `0x000000`
 * @param color 颜色参数，为 16 进制标识法，如 `0x000000`
 */
.setOutputBackgroundColor(color)
/**
 * 向合流后的视频流中增加背景图片
 * @param uri 图片资源的完整下载地址
 * @param x 相对于整体画布的起始位置 x 坐标（百分比），有效值 `0.0` - `1.0`
 * @param y 相对于整体画布的起始位置 y 坐标（百分比），有效值 `0.0` - `1.0`
 * @param w 相对于整体画布的宽（百分比），有效值 `0.0` - `1.0`
 * @param h 相对于整体画布的高（百分比），有效值 `0.0` - `1.0`
 */
.addOutputBackgroundPicture(uri, x, y, w, h)
/**
 * 移除对合流后的视频流中添加的指定背景图片
 * @param uri 需要移除的图片资源地址
 */
.removeOutputBackgroundPicture(uri)
/**
 * 清理视频流中添加的所有背景图片
 */
.clearOutputBackgroundPicture()
```

自定义布局

 提示

当设置布局模式为自定义布局（`MixLayoutMode.CUSTOMIZE`）时，需自行指定每个所需要渲染的视频流的位置与宽高

API 参考：

- [addCustomizeLayoutVideo](#) 
- [removeCustomizeLayoutVideo](#) 
- [clearCustomizeLayoutVideo](#) 

```
builder
/**
 * 在自定义布局中增加视频流配置
 * @param trackId 资源 Id
 * @param x - 在画布中的横向坐标点
 * @param y - 在画布中的纵向坐标点
 * @param width 视频分辨率宽度
 * @param height 视频分辨率高度
 */
.addCustomizeLayoutVideo (trackId, x, y, width, height)
/**
 * 在自定义布局配置中移除指定视频流
 * @param trackId 视频资源 Id
 */
.removeCustomizeLayoutVideo (trackId)
/**
 * 清空已配置的自定义视频流布局
 */
.clearCustomizeLayoutVideo()
```

自定义音频合流

提示

该功能依赖 RTCLib v5.3.7 及以上版本。

默认同一房间内的所有音频都会被合流至输出音频中。当需要对合流音频进行自定义时，可以通过如下接口实现。

API 参考：

- [setCustomizeInputAudio](#) 
- [addCustomizeInputAudio](#) 
- [removeCustomizeInputAudio](#) 
- [clearCustomizeInputAudio](#) 

```
builder
/**
 * 覆盖设置合流媒体中的音频流
 * @param trackIds 音频流 trackId 数组，当数组长度为 0 时，则合流媒体中将无音频输出
 * @returns
 */
.setCustomizeInputAudio([audioTrack.getTrackId(), audioTrack2.getTrackId()])
/**
 * 向已有的音频流合流配置中动态增加一道音频流
 * @param trackId 音频 trackId
 */
.addCustomizeInputAudio(audioTrack3.getTrackId())
/**
 * 从已有的音频流合流配置中动态删除一道音频流
 * @param trackId 音频对应的 trackId
 */
.removeCustomizeInputAudio(audioTrack.getTrackId())
/**
 * 清除音频流合流配置，恢复房间内的全音频流合流输出
 */
.clearCustomizeInputAudio()
```

为单道视频流添加图片水印

提示

该功能依赖 RTCLib v5.4.3 及以上版本。水印图片仅支持 PNG 格式。如果水印添加异常，请先检查并确保图片为 PNG 格式文件。

调用 [builder.addPictureWaterMark](#) 可以为合流中为单道视频流（子视图）添加图片水印。配置生效后，仅订阅合流的直播观众会看到合流子视图的视频携带水印。

- [addPictureWaterMark](#) 方法无法为客户端发布的视频流（上行视频流）添加水印。因此，订阅分流的用户（一般为其他主播）看到的视频不会携带水印。如果使用云端录制服务录制分流，也无法看到水印。如需为客户端上行视频流添加水印，详见[水印处理](#)。
- 客户端不支持在服务端输出的视频合流上添加整体水印。MCU 合流在服务端完成，调用服务端 API [/rtc/mcu/config](#) 可为单道视频流以及合流输出视频流添加水印，支持时间戳水印、文字水印或图片水印。详见服务端文档[直播合流](#)。客户端与服务端添加的水印相互独立。如果同时使用，则订阅合流的观众可能会看到水印叠加。

```

builder
/**
 * 给单道流添加水印
 * @param trackId 资源 Id
 * @param uri 水印图片的地址，需注意图片需要是 png 格式
 * @param x 相对于整体画布的起始位置 x 坐标（百分比），有效值 `0.0` - `1.0`
 * @param y 相对于整体画布的起始位置 y 坐标（百分比），有效值 `0.0` - `1.0`
 * @param width 相对于整体画布的宽（百分比），有效值 `0.0` - `1.0`
 * @param height 相对于整体画布的高（百分比），有效值 `0.0` - `1.0`
 * @description 注意，参数中 x + width 不得大于 1，y + height 不得大于 1，否则调用 flush() 时会提示 46020 错误
 */
.addPictureWaterMark(trackId, uri, x, y, width, height)
/**
 * 移除指定视频流中添加的指定水印图片
 * @param trackId 资源 Id
 * @param uri 水印图片的地址，需注意图片需要是 png 格式
 */
.removePictureWaterMark(trackId, uri)
/**
 * 清除所有水印
 */
.clearPictureWaterMark()

```

API 参考：

- [addPictureWaterMark](#) 
- [clearPictureWaterMark](#) 
- [removePictureWaterMark](#) 

重置默认配置

提示

该功能依赖 RTCLib v5.3.7 及以上版本。

```
builder.reset()
```

使配置生效

提示

- 在调用该方法前，所有接口调用只会对本地配置进行修改，不会产生实际效果。
- 调用 `flush` 接口前，需确保主播已在房间内发布了视频资源，否则会返回 `53005`。
- 调用 `flush` 接口后，默认会清空 `RCMCUConfigBuider` 实例存储在内存中的所有配置，因此用户修改配置时需重新构建。
- **v5.3.7** 及之后版本，可以通过 `flush(false)` 避免提交后清空存储在内存中的配置。

API 参考：[flush](#) 

```
// 上传配置给 media server 以修改最终合流效果  
const { code } = await builder.flush()
```

观众端 (加入房间)

更新时间:2024-08-30

在直播模式下，观众角色的用户可以在进入音视频房间后订阅远端主播角色用户已发布的资源。

获取房间内的资源

直播模式下主播角色用户发布的音视频流，会在服务端另行合并生成一道音频合流和一道视频合流。观众角色用户通常订阅“合流”（也可以直接订阅原始音视频流，简称“分流”）。

提示

观众订阅分流适合小众玩法灵活的直播业务场景。比如观众被分成多种角色，需要选择性观看或收听部分主播发布的资源；又或者不同观众的展示布局并不相同，甚至可以随时切换布局的情况。

加入音视频房间的观众角色用户可以通过以下方式获取资源：

- 加入房间的返回数据（要求 SDK \geq 5.2.3）。使用 `joinLivingRoomAsAudience` 加入房间后，SDK 会返回以下数据（房间内已有资源不会通过监听通知业务层）：
 - `RTCTracks`：房间内已有的主播资源
 - `MCUTracks`：房间内已有的合流资源
 - `CDNUri`：房间内置 CDN 信息
 - `userIds`：房间内主播列表

```
// 获取 RCLivingType 枚举定义
import { RCLivingType } from '@rongcloud/plugin-rtc'

/*
 * 观众加入直播房间调用
 * @param roomId 房间 Id
 * @param livingType 直播类型
 * * 当 `livingType` 值为 `RCLivingType.AUDIO` 是表示音频直播
 * * 当 `livingType` 值为 `RCLivingType.AUDIO_VIDEO` 是表示音视频直播
 */
// 从 5.2.3 开始，加入房间时可返回 RTCTracks、MCUTracks、CDNUri、userIds
const { room: audienceRoom, RTCTracks, MCUTracks, CDNUri, userIds, code } = await
rtcClient.joinLivingRoomAsAudience('roomId', RCLivingType.AUDIO_VIDEO)
// 若加入失败，则 room 值为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('join room as audience failed:', code)
}
```

- 观众角色的房间事件监听。观众角色的用户获取 [RCAudienceLivingRoom](#) 房间实例后，可以使用 `registerRoomEventListener` 注册房间事件监听器 [IAudienceRoomEventListener](#)。下表列出了 `IAudienceRoomEventListener` 与资源相关的部分方法：

方法	返回属性	说明
<code>onTrackPublish</code>	<code>RRemoteTrack[]</code>	(SDK \geq 5.1.0) 返回新发布的合流音轨与视轨数据列表，包含新发布的 <code>RRemoteAudioTrack</code> 与 <code>RRemoteVideoTrack</code> 实例。观众角色用户通常仅需订阅合流。
<code>onTrackUnpublish</code>	<code>RRemoteTrack[]</code>	(SDK \geq 5.1.0) 返回被取消发布的合流音轨与视轨数据列表。当房间内全部主播退出房间时，SDK 内部会取消对资源的订阅，业务层仅需处理 UI 业务。
<code>onAnchorTrackPublish</code>	<code>RRemoteTrack[]</code>	(SDK \geq 5.1.5) 返回主播新发布的音轨与视轨数据列表，包含新发布的 <code>RRemoteAudioTrack</code> 与 <code>RRemoteVideoTrack</code> 实例。
<code>onAnchorTrackUnpublish</code>	<code>RRemoteTrack[]</code>	(SDK \geq 5.1.5) 返回被主播取消发布的音轨与视轨数据列表。当资源被取消发布时，SDK 内部会取消对相关资源的订阅，业务层仅需处理 UI 业务。

```

// audienceRoom 为观众加入房间返回的 room
// 注册房间事件监听器，重复注册时，仅最后一次有效
audienceRoom.registerRoomEventListener({
/**
 * 主播加入
 * @param userIds 加入的主播 id 列表
 */
onAnchorJoin (userIds: string[]) {
},
/**
 * 主播离开
 * @param userIds 离开的主播 id 列表
 */
onAnchorLeave (userIds: string[]) {
},
/**
 * 房间内直播合流资源发布
 * @param track RCRemoteTrack 类实例
 * * 房间类型以第一个加入房间用户设置的直播类型为准
 * * 房间直播类型为 RCLivingType.AUDIO_VIDEO ，tracks 包含 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
 * * 直播类型为 RCLivingType.AUDIO ， tracks 仅包含 RCRemoteAudioTrack 实例
 * * 触发时机：主播发布资源后
 */
onTrackPublish (tracks: RCRemoteTrack[]) {
},
/**
 * 房间内直播合流资源取消发布
 * @param track RCRemoteTrack 类实例
 * * 触发时机：全部主播退出房间（因资源为多个主播发布的合流资源，单个主播取消发布不会触发）
 */
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
 * 房间内主播资源发布
 * @param track RCRemoteTrack 类实例
 * * 触发时机：主播发布资源后
 */
onAnchorTrackPublish (tracks: RCRemoteTrack[]){
},
/**
 * 房间内主播资源取消发布
 * @param track RCRemoteTrack 类实例
 * * 触发时机：主播取消发布资源后
 */
onAnchorTrackUnpublish (tracks: RCRemoteTrack[]){
},
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
onTrackReady (track: RCRemoteTrack) {
// 订阅的音视频轨道已连接，可以根据业务需求选择性播放
if (track.isAudioTrack()) {
// 音频播放无需传递组件
track.play()
} else {
// 此处的 videoNode 为 <video> 标签元素实例
track.play(videoNode)
}
}
})

```

- 通过 [RCAudienceLivingRoom](#) 房间实例获取。RCAudienceLivingRoom 房间实例上提供了以下主动获取资源的方法：

方法	返回属性	说明
<code>getRemoteMCUTracks()</code>	<code>RCRemoteTrack[]</code>	(SDK \geq 5.2.3) 返回房间内的合流音轨与视轨数据列表，包含新发布的 <code>RCRemoteAudioTrack</code> 与 <code>RCRemoteVideoTrack</code> 实例。观众角色用户通常仅需订阅合流。
<code>getRemoteRTCTracks()</code>	<code>RCRemoteTrack[]</code>	(SDK \geq 5.2.3) 返回主播发布的音轨与视轨数据列表，包含新发布的 <code>RCRemoteAudioTrack</code> 与 <code>RCRemoteVideoTrack</code> 实例。
<code>getCDNInfo()</code>	CDNInfo	(SDK \geq 5.2.3) 返回房间内 CDN 资源状态（是否开启、分辨率、帧率）。

订阅资源

观众加入房间并获取资源后，使用 [subscribe](#) 方法订阅资源：

```
const { code } = await audienceRoom.subscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
  console.log('资源订阅失败 ->', code)
}
```

仅在观众角色订阅分流，且远端主播发布资源时设置了发布大小流，则订阅对方视频流时，可选择订阅对方的视频小流，以节省下行带宽。订阅小流仅在对方发布了小流的情况下有效，若对方并未发布小流，接收的仍然是对方的大流数据。

```
const { code } = await audienceRoom.subscribe([
  // 音频不支持大小流
  audioTrack,
  {
    track: videoTrack,
    subTiny: true // 订阅小流
  }
])
```

取消订阅

使用 [unsubscribe](#) 取消订阅。

```
// 按业务需求选择取消订阅资源
const { code } = await audienceRoom.unsubscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
  console.log('资源取消订阅失败:', code)
}
```

观众端 (不加房间)

更新时间:2024-08-30

融云 RTC Web SDK 支持直播模式下的观众直接通过资源地址 (liveUrl) 订阅并观看直播 (观众不加房间订阅)。

提示

如果 SDK 版本 < 5.1.0，仅支持直接通过资源地址 liveUrl 订阅 (观众不加房间订阅)。直播的观众通过 liveUrl 的方式去订阅主播端，但 liveUrl 和 livingType 数据的分发需开发者自行实现。

与加入房间订阅观看直播方式的对比

- 如果 SDK < 5.1.0，仅支持直接通过资源地址 liveUrl 订阅并观看直播 (观众不加房间订阅)。liveUrl 和 livingType 数据的分发需开发者自行实现。
- 如果 SDK ≥ 5.1.0，支持观众加入 RTC 房间，并订阅房间内资源。观众角色的用户可以加入房间的返回值、房间事件监听器、房间实例的获取资源方法获取房间内的资源。

RCAudienceClient

如果观众需要通过资源地址 liveUrl 订阅主播端资源，需要先使用 RCRTCClient 实例的 getAudienceClient 方法获取 RCAudienceClient 实例。该类型实例仅会在初次获取时初始化一次。

```
// 获取 RCAudienceClient 观众端实例
const liveUrlAudienceClient = rtcClient.getAudienceClient()
```

RCAudienceClient 提供以下方法：

方法	说明
subscribe	订阅 liveUrl 资源。
unsubscribe	取消订阅主播资源。
registerReportListener	注册数据监控监听器 IRCRTCReportListener。
registerTrackEventListener	注册流事件监听 IRCRTCTrackEventListener，多次注册会导致后者覆盖前者，可以通过使用 registerTrackEventListener(null) 取消注册。

流数据监听

使用 registerTrackEventListener 注册 IRCRTCTrackEventListener。

```

// 注册流数据监听
liveUrlAudienceClient.registerTrackEventListener({
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
onTrackReady (track: RCRemoteTrack) {
// 订阅的音视频轨道已连接，可以根据业务需求选择性播放
if (track.isAudioTrack()) {
// 音频播放无需传递组件
track.play()
} else {
// 此处的 videoNode 为 <video> 标签元素实例
track.play(videoNode)
}
}
})

```

订阅资源

同一时间仅能订阅一个来自直播间的 liveUrl 资源，如需订阅其他直播间资源，请先取消订阅。

1. 获取 RCLivingType、RCMediaType 枚举定义。livingType 是主播建立直播房间时所选择的直播间类型，由开发者自行实现分发逻辑。

```

// 获取 RCLivingType, RCMediaType 枚举定义
import { RCLivingType, RCMediaType } from '@rongcloud/plugin-rtc'

```

2. 订阅直播间资源。订阅资源所需的 liveUrl 由主播发布资源后获取，由开发者自行实现分发逻辑。

```

/*
 * 订阅直播间资源
 * @param liveUrl 直播资源地址
 * @param livingType 直播间类型，其有效值为 `RCLivingType` 所定义的枚举值
 * @param mediaType 订阅资源类型，其有效值为 `RCMediaType` 所定义的枚举值
 * @param subTiny 当值为 `true` 时将订阅小流，否则订阅大流。默认值为 `false`
 */
const {
code,
// tracks 是一个数组，若 code !== RCRTCCode.SUCCESS，则 tracks 长度为 0
// tracks 最多包含一个 RCRemoteAudioTrack 实例和一个 RCRemoteVideoTrack 实例
tracks
} = await liveUrlAudienceClient.subscribe(liveUrl, RCLivingType.VIDEO, RCMediaType.AUDIO_VIDEO, false)

if (code !== RCRTCCode.SUCCESS) {
console.warn('订阅主播资源失败 ->', code)
}

```

取消订阅

使用 [unsubscribe](#) 取消订阅。

```
const { code } = await liveUrlAudienceClient.unsubscribe()
```

媒体播放与管理

更新时间:2024-08-30

不论是本端 `RCLocalTrack` 类，还是远端 `RCRemoteTrack` 类，都继承自 `RCTrack` 基类，因此对于本地和远端音视频轨道数据，资源管理接口是完全一致的。

提示

以下代码示例中的 `track` 表示为 `RCTrack` 基类实例，既表示远端流，也表示本地流。

播放音视频资源

使用 `videoTrack` 实例的 `play` 方法，传入 `<video>` 标签，可播放视频。

```
// 通过 videoTrack.play 方法将 <video> 标签传递给 videoTrack 实例
videoTrack.play(videoNode)
```

直接调用 `audioTrack` 实例的 `play` 方法即可播放音频，无需额外传入参数。如果是采集的音频流，请尽量不要在本端播放，因为可能会引起回声问题。

```
// 播放音频时无需传参，
audioTrack.play()
```

播放音频资源时，支持置播放音量，音量值接受 0-100 的整数。

```
audioTrack.play(null, {volume})
```

播放音频资源时，支持从指定播放音频设备输出。您需要先从 `@rongcloud/plugin-rtc` 模块导入 `device` 模块（详见[设备管理](#)）。通过 `device.getSpeakers()` 获取扬声器设备列表后，在 `play` 方法中传入。

```
audioTrack.play(null, {audioDeviceId})
```

禁用资源

调用本端发布的音轨或视轨的 `mute` 方法，可使房间内他人的客户端上不再播放该声音或视频。房间内的其他成员可通过[房间事件监听器](#)收到 `onAudioMuteChange` 与 `onVideoMuteChange` 事件回调。

调用他人发布的音轨或视轨的 `mute` 方法，本端暂停播放，对他人无影响。

```
track.mute()
```

启用资源

调用本端发布的音轨或视轨的 [unmute](#) 方法，将在房间内他人客户端上的恢复播放该资源。房间内的其他成员可通过 [房间事件监听器](#) 收到 `onAudioMuteChange` 与 `onVideoMuteChange` 事件回调。

调用他人发布的音轨或视轨的 [unmute](#) 方法，仅恢复本端播放。

```
track.unmute()
```

查询本端是否禁用资源

使用 [isLocalMuted](#) 查询资源在本地是否被禁用，即表示本端是否禁用了该资源。

```
const muted: boolean = track.isLocalMuted()
```

查询资源发布者是否禁用资源

使用 [isOwnerMuted](#) 查询资源在房间中是否被禁用，即资源发布者是否禁用了该资源。对于 `RCLocalTrack` 类实例，该值永远等于 `isLocalMuted()`。

```
const muted: boolean = track.isOwnerMuted()
```

查询资源是否已发布到房间中

对于本地资源（`RCLocalTrack` 类实例），可使用 [isPublished](#) 方法来确定资源是否已发布到房间中。

```
const published: boolean = localTrack.isPublished()
```

查询资源是否已被本端订阅

对于远端资源（`RCRemoteTrack` 类实例），可以通过 [isSubscribed](#) 方法来确定资源是否已被己方订阅。

```
const subscribed: boolean = remoteTrack.isSubscribed()
```

同房间连麦

更新时间:2024-08-30

同房间连麦是指让已加入直播房间中的观众切换为主播身份，并参与直播。

观众上麦（升级为主播房间）

使用 `RCRTCClient` 实例的 `upgradeToAnchorRoom` 方法，传入当前观众用户的 `RCAudienceLivingRoom` 实例，SDK 会返回主播角色的房间 `RCLivingRoom` 实例。

```
/*
 * @param audienceRoom 观众房间 room 实例，调用 rtcClient.joinLivingRoomAsAudience 或
 rtcClient.downgradeToAudienceRoom 方法返回的 room
 * @since @rongcloud/plugin-rtc@5.6.8 增加返回 userIds, tracks, CDNEnable，房间内已有的人员、资源、CDN 状态不再
 从 room 监听处抛出
 * @return userIds 返回加房间时，房间内的其他人
 * @return tracks 返回加房间时，房间内其他人发布的资源
 * @return CDNEnable 返回加房间时，房间内 CDN 开关是否开启
 */
const { room: anchorRoom, code, userIds, tracks, CDNEnable } = await
rtcClient.upgradeToAnchorRoom(audienceRoom)
// 若升级失败，anchorRoom 为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('upgrade failed:', code)
}
```

`RCAudienceLivingRoom` 实例升级为主播房间 `RCLivingRoom` 实例后，需要同步按需调用房间事件监听器或质量数据监听器。上麦成功后，用户身份与主播并无区别，可进行与主播同等的接口调用。详见[主播端](#)。

观众下麦（降级为观众房间）

1. 在降级前请释放摄像头资源。
2. 使用 `RCRTCClient` 实例的 `downgradeToAudienceRoom` 方法，传入当前主播用户的 `RCLivingRoom` 实例，SDK 会返回观众角色的房间 `RCAudienceLivingRoom` 实例。

```
/*
 * 使用场景为主播降级观众
 * @param anchorRoom 主播加入房间成功后返回的 room
 */
// 从 5.2.3 开始，加入房间时可返回 RTCTracks、MCUTracks、CDNUri、userIds
const { room: audienceRoom, RTCTracks, MCUTracks, CDNUri, userIds, code } = await
rtcClient.downgradeToAudienceRoom(anchorRoom)
// 若降级失败，会返回 code，room 为 undefined
if (code !== RCRTCCode.SUCCESS) {
  console.log('downgrade failed:', code)
}
```

从 5.2.3 版本开始，downgradeToAudienceRoom方法会返回以下房间内数据：

- **RTCTracks**：房间内已有的主播资源
- **MCUTracks**：房间内已有的合流资源
- **CDNUri**：房间内置 CDN 信息
- **userIds**：房间内主播列表

降级为观众房间后的方式详见[观众端](#)。

3. 如果 SDK 版本 < 5.2.3，降级成功后需同步注册观众房间事件监听器。

订阅 liveUrl 资源的观众如何参与连麦

如果观众不加入 RTC 房间，直接通过订阅 liveUrl 观看直播，参与连麦的流程如下：

1. 使用初始化时获取的 [RCRTCClient](#) 实例的 [joinLivingRoom](#) 方法加入 RTC 房间。加入成功后，该用户即具备主播身份。
2. 获取资源，并以主播角色发布资源、订阅资源。详见[主播端](#)
3. 如果连麦结束，使用 [RCRTCClient](#) 实例的 [leaveRoom](#) 方法下麦。

跨房间连麦

跨房间连麦

更新时间:2024-08-30

依赖版本: plugin-rtc@5.3.0, 对应 imlib 版本为 imlib-v2@2.11.0 或 imlib-next@5.1.0

前提: 跨房间连麦, 需要双方都加入各自的直播房间后进行。

跨房间连麦功能中有“主房间”和“副房间”两个概念。

- 主房间：本端加入的直播房间
- 副房间：在接受连麦邀请后加入的对方房间

主流程

获取跨房间连麦处理对象 roomPKHandler

```
// room 为主播加入各自直播间后, 得到的直播房间 room 实例  
  
// 获取跨房间连麦功能处理模块  
const { roomPKHandler } = room.getRoomPKHandler();
```

主播注册跨房间连麦事件监听

```
// 注册跨房间连麦相关事件监听  
roomPKHandler.registerRoomPKEventListener({  
  /**  
  * 收到连麦邀请, 此时可调 roomPKHandler.responseJoinOtherRoom 响应连麦  
  */  
  onRequestJoinOtherRoom (info: IPKInviteInfo) {},  
  /**  
  * 收到连麦邀请已被取消  
  */  
  onCancelRequestOtherRoom (info: IPKInviteInfo) {},  
  /**  
  * 收到被邀请方应答, 数据中包含对方是否同意请求  
  * 如果对方同意连麦, 此时可调 roomPKHandler.joinOtherRoom 加入对方房间  
  */  
  onResponseJoinOtherRoom: (info: IPKInviteAnswerInfo) => {  
    console.log(info.agree)  
  },  
  /**  
  * 收到连麦结束通知  
  * 连麦结束时, plugin-rtc 内部会退出副房间, 业务层只需做 UI 处理  
  */  
  onFinishOtherRoom: (info: IPKEndInfo) => {}  
})
```

发起连麦邀请

向指定房间的某个用户发送跨房间连麦请求

```
/**
 * 发起跨房间连麦请求
 * @param inviteeRoomId 被邀请者所处的房间 roomId
 * @param inviteeUserId 被邀请者 userId
 * @param options.autoMix 是否【在加入副房间后自动】将邀请者发布的资源，合并到被邀请者房间内的 MCU 流中。【退出副房间会自动取消合并】
 * @param options.extra 附加信息，可随邀请连麦消息携带给被邀请者
 */
const { code } = await roomPKHandler.requestJoinOtherRoom(inviteeRoomId: string, inviteeUserId: string,
options?: IReqRoomPKOptions)
if (code === RCRTCCode.SUCCESS) {
console.log('发送连麦邀请成功');
}
```

取消连麦邀请

被邀请者未响应连麦请求时，邀请方取消发出去的跨房间连麦请求。

```
/**
 * 取消跨房间连麦请求
 * @param inviteeRoomId 被邀请者所处的房间 roomId
 * @param inviteeUserId 被邀请者 userId
 * @param extra 附加信息，可随取消邀请连麦消息携带给被邀请者
 */
const { code } = await roomPKHandler.cancelRequestJoinOtherRoom(inviteeRoomId: string, inviteeUserId:
string, extra?: string)
if (code === RCRTCCode.SUCCESS) {
console.log('发送取消连麦成功');
}
```

响应连麦请求

被邀请方响应邀请方发出的跨房间连麦请求。

```
/**
 * 响应跨房间连麦请求
 * @param inviterRoomId 邀请者所处的房间 roomId
 * @param inviterUserId 邀请者 userId
 * @param agree 是否同意连麦
 * @param options.autoMix 是否【在加入副房间后自动】将被邀请者发布的资源，合并到邀请者房间内的 MCU 流中。【退出副房间会自动取消合并】
 * @param options.extra 附加信息，可随响应连麦消息携带给邀请者
 */
const { code } = await roomPKHandler.responseJoinOtherRoom (inviterRoomId: string, inviterUserId:
string, agree: boolean, options?: IReqRoomPKOptions)
if (code === RCRTCCode.SUCCESS) {
console.log('发送应答连麦成功');
}
```

加入副房间

在连麦邀请方与被邀请方建立连麦关系后，加入副房间。副房间内只能订阅资源，不能发布资源。

加入副房间后，SDK 内部会根据 autoMix 值，通知服务是否把本房间资源合并入副房间的合流资源中，所以指定 autoMix 为 true 后，还需要加入副房间。

```
/**
 * 加入副房间
 * @param roomId 副房间 Id, 支持大小写英文字母、数字、部分特殊符号 + = - _ 的组合方式 最长 64 个字符
 */
const res = await roomPKHandler.joinOtherRoom(roomId: string)
if (res.code === RCRTCCode.SUCCESS) {
  const { room: otherRoom, userIds, tracks } = res
  console.log('房间实例: ', otherRoom)
  console.log('房间内人员: ', userIds)
  console.log('房间内资源: ', tracks)
}
```

注册副房间事件监听

```
// 注册房间事件监听器，重复注册时，仅最后一次注册有效
otherRoom.registerRoomEventListener({
  /**
   * 当本端被踢出房间
   * @description 被踢出房间可能是由于服务端超出一定时间未能收到 rtcPing 消息，所以认为己方离线。
   * 另一种可能是己方 rtcPing 失败次数超出上限，故而主动断线
   * @param byServer
   * 当值为 false 时，说明本端 rtcPing 超时
   * 当值为 true 时，说明本端收到被踢出房间通知
   * @param state 被踢出房间的原因
   */
  onKickOff? (byServer: boolean, state?: RCKickReason): void
},
  /**
   * 接收到房间信令时回调，用户可通过房间实例的 `sendMessage(name, content)` 接口发送信令
   * @param name 信令名
   * @param content 信令内容
   * @param senderUserId 发送者 Id
   * @param messageUid 消息唯一标识
   */
  onMessageReceive (name: string, content: any, senderUserId: string, messageUid: string) {
  },
  /**
   * 监听房间属性变更通知
   * @param name
   * @param content
   */
  onRoomAttributeChange (name: string, content: string) {
  },
  /**
   * 房间用户禁用/启用音频
   * @param audioTrack RCRemoteAudioTrack 类实例
   */
  onAudioMuteChange (audioTrack: RCRemoteAudioTrack) {
  },
  /**
   * 房间用户禁用/启用视频
   * @param videoTrack RCRemoteVideoTrack 类实例对象
   */
  onVideoMuteChange (videoTrack: RCRemoteVideoTrack) {
```

```

onTrackUnpublish (videoTrack: RCRemoteVideoTrack) {
},
/**
 * 房间内用户发布资源
 * @param tracks 新发布的音轨与视轨数据列表，包含新发布的 RCRemoteAudioTrack 与 RCRemoteVideoTrack 实例
 */
async onTrackPublish (tracks: RCRemoteTrack[]) {
// 按业务需求选择需要订阅资源，通过 room.subscribe 接口进行订阅
const { code } = await room.subscribe(tracks)
if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
 * 房间用户取消发布资源
 * @param tracks 被取消发布的音轨与视轨数据列表
 * @description 当资源被取消发布时，SDK 内部会取消对相关资源的订阅，业务层仅需处理 UI 业务
 */
onTrackUnpublish (tracks: RCRemoteTrack[]) {
},
/**
 * 订阅的音视频流通道已建立，track 已可以进行播放
 * @param track RCRemoteTrack 类实例
 */
onTrackReady (track: RCRemoteTrack) {
if (track.isAudioTrack()) {
// 音轨不需要传递播放控件
track.play()
} else {
// 视轨需要一个 video 标签才可进行播放
const element = document.createElement('video')
document.body.appendChild(element)
track.play(element)
}
},
/**
 * 人员加入
 * @param userIds 加入的人员 id 列表
 */
onUserJoin (userIds: string[]) {
},
/**
 * 人员退出
 * @param userIds
 */
onUserLeave (userIds: string[]) {
},
/**
 * 房间内主播和观众切换身份
 * @param userId 用户 ID
 * @param role 用户角色
 * role 值为 RCRTCLiveRole.ANCHOR 时，代表房间内观众升级为主播
 * role 值为 RCRTCLiveRole.AUDIENCE 时，代表主播降级为房间内的观众
 */
onSwitchRole (userId: string, role: RCRTCLiveRole) {
}
})

```

订阅副房间资源

和主房间一致，副房间要订阅的远端资源有两个来源: joinOtherRoom 返回的 tracks 和房间监听事件 onTrackPublish 处收到的 tracks。订阅成功后，在房间监听事件 onTrackReady 处播放。

```
const { code } = await otherRoom.subscribe([
// 音频不支持大小流
audioTrack,
{
track: videoTrack,
subTiny: true // 订阅小流
}
])
```

取消订阅副房间资源

```
const { code } = await otherRoom.unsubscribe([audioTrack, videoTrack])
// 取消订阅完成后，业务层移除相关的 UI 信息即可
```

退出副房间

退出副房间，参数 `isQuitPK` 为 `false` 时，连麦关系依然存在，可重新加入副房间。

```
/**
 * 退出副房间
 * @param room 要退出的副房间 room 实例
 * @param isQuitPK 是否结束连麦
 */
const { code } = await roomPKHandler.leaveOtherRoom (room: RCLivingRoom, isQuitPK?: boolean)
```

房间是否为主房间

```
/**
 * @since version 5.3.2
 */
const isMainRoom = room.isMainRoom()
```

获取所有的连麦信息

```
/**
 * @since version 5.3.2
 */
const allPKInfo = roomPKHandler.getAllPKInfo()
console.log('所有连麦信息:', allPKInfo)
```

获取指定副房间的连麦信息

```
/**
 * 获取连麦信息
 * @param roomId 连麦房间的 roomId
 */
const PKInfo = roomPKHandler.getPKInfo(roomId)
console.log(`和 ${roomId} 的连麦信息:`, PKInfo)
```

获取已加入的副房间

```
const joinedPKRooms: { [roomId: string]: RCLivingRoom } = roomPKHandler.getJoinedPKRooms()
```

融云 CDN 服务配置

更新时间:2024-08-30

欢迎前往融云官网了解[直播 CDN 产品](#)。本文介绍如何开始配置融云 CDN 服务。

开通服务

融云 CDN 是付费增值服务，且需要开通后才能使用。您可以访问控制台的[融云 CDN](#)页面开通服务。



开通融云 CDN 服务后，可看到[域名配置](#)、[证书管理](#)、[防盗链配置](#)。

选择推拉流模式

融云 CDN 支持三种不同的推拉流地址模式：

- **开播自动推流**：在后台配置后，所有直播房间的直播流都会自动推流到 CDN，观众可在客户端加入房间时获取 CDN 信息，或者使用客户端 SDK 提供的监听房间内 CDN 资源的方法。如果您希望直接从融云服务端获取拉流地址，可从 App 服务端调用融云服务端 API，详见[获取融云 CDN 拉流地址](#)。
- **开播手动推流**：在直播主播开始推流后，您需要根据产品设计决定何时调接口让融云服务开始或停止推流到 CDN。观众可在客户端加入房间时获取 CDN 信息，或者使用客户端 SDK 提供的监听房间内 CDN 资源的方法。如果您希望直接从融云服务端获取拉流地址，可从 App 服务端调用融云服务端 API，详见[获取融云 CDN 拉流地址](#)。
- **自行生成推拉流地址**：融云服务端不负责生成推拉流地址，您需要在您的应用服务器自行生成 CDN 推拉流地址。



自行拼接推拉流地址

在自行生成推拉流地址模式下，可自行拼接生成 CDN 推拉流地址。需要拼接的字段包括推/拉流域名、自定义的 {AppName} 和自定义的 {StreamName}。您可以通过在 CDN 拉流地址中添加分辨率、码率参数 {with}、{height}、{fps}，拉取 CDN 转码流。

CDN 推流地址拼接规则

融云 CDN 仅支持 RTMP 协议的推流：

RTMP：rtmp://推流域名/{AppName}/{StreamName}

拼接推流地址后，可调用客户端 SDK 或服务端 API 的旁路推流接口，传入 CDN 推流地址进行推流。

CDN 拉流地址拼接规则

融云 CDN 支持使用 RTMP、FLV、HLS 协议进行拉流。

如需使用 CDN 流的原始分辨率、帧率：

- **RTMP**：rtmp://拉流域名/{AppName}/{StreamName}
- **FLV**：http(s)://拉流域名/{AppName}/{StreamName}.flv
- **HLS**：http(s)://拉流域名/{AppName}/{StreamName}.m3u8

如需拉取不同分辨率的 CDN 直播流（拉取非原始分辨率、码率的流会触发 CDN 转码服务，产生转码费用）：

- **RTMP**：rtmp://拉流域名/{AppName}/{StreamName}_{with}_{height}_{fps}
- **FLV**：http(s)://拉流域名/{AppName}/{StreamName}_{with}_{height}_{fps}.flv
- **HLS**：http(s)://拉流域名/{AppName}/{StreamName}_{with}_{height}_{fps}.m3u8

{with}、{height} 参数为要拉取的 CDN 转码流的分辨率宽高，参考下方枚举值。{fps} 为帧率参数，支持 10/15/24/30。

宽高限制列表：

```
[
"176*144", "180*180", "256*144", "240*180", "320*180", "240*240", "320*240",
"360*360", "480*360", "640*360", "480*480", "640*480", "720*480", "848*480",
"960*720", "1280*720", "1920*1080", "144*176", "144*256", "180*240", "180*320",
"240*320", "360*480", "360*640", "480*640", "480*720", "480*848", "720*960",
"720*1280", "1080*1920"
]
```

例如，拉取 flv 协议的 CDN 流，并指定宽x高为 720*920，帧率为 15，则拼接后的地址如下：

http(s)://yourdomain/appkey/roomid_720_960_15.flv

推荐使用融云 CDN 播放器进行播放。

提示

- 如在控制台启用了防盗链，则您拼接的推拉流地址中还必须携带防盗链信息。防盗链地址的具体拼接规则请参见下文「防盗链配置」。
- 2022年6月前开通融云 CDN 服务的客户，如获取到的 HLS 拉流地址无法正常播放，请提交工单咨询。

域名配置

在域名配置标签下配置并保存了推流域名和拉流域名后，会看到完整的配置界面。

The screenshot shows the 'CDN 服务配置' (CDN Service Configuration) page. At the top, it indicates '融云 CDN 服务状态: 已开通' (Ronyun CDN Service Status: Enabled) with a '关闭服务' (Close Service) button. Below this, there's a '推拉流模式' (Push/Pull Mode) dropdown set to '自行生成推拉流地址' (Generate push/pull stream addresses myself) and a '保存修改' (Save Changes) button. The '域名配置' (Domain Configuration) tab is active, with sub-tabs for '证书管理' (Certificate Management) and '防盗链' (Anti-leech). Under '域名配置', there are two rows for domain settings. The first row is for the '推流域名' (Push Stream Domain), and the second is for the '拉流域名' (Pull Stream Domain). Each row has a text input field, a status indicator '已生效' (Effective), a 'cname 域名' (CNAME Domain) field, and a '复制' (Copy) button. A red note below states: '注意: CNAME 域名不能直接访问, 您需要在域名服务提供商处完成 CNAME 配置。配置生效后, 即可使用 CDN 链路进行直播拉流' (Note: CNAME domains cannot be accessed directly; you need to complete CNAME configuration at the domain service provider. After configuration is effective, you can use the CDN link for live streaming). Below the note, it says: '在 20:00 至次日 2:00 进行的配置不会立即生效, 会在 2 点后依次配置生效。在此时间段外进行配置会在 30 分钟后生效' (Configurations made between 20:00 and 02:00 the next day will not take effect immediately, but will take effect sequentially after 2:00. Configurations made outside this time period will take effect 30 minutes later). There is a '禁用' (Disable) button. The 'HTTPS 设置' (HTTPS Settings) section includes a提示: 'FLV/HLS 协议的直播流默认使用 HTTP, 配置 SSL 证书后可以使用 HTTPS。如果您没有在融云平台上上传过证书请先上传证书' (提示: FLV/HLS protocol live streams default to HTTP, and can use HTTPS after configuring an SSL certificate. If you haven't uploaded a certificate on the Ronyun platform, please upload it first) and a '新增证书' (Add Certificate) button. At the bottom, there are two '绑定证书' (Bind Certificate) buttons, one for '推流' (Push) and one for '拉流' (Pull), each with a '请选择' (Please select) dropdown menu.

配置推拉流域名

推拉流域名都需要填写二级域名，请确保此域名没有在别的 App Key 下配置过。

提示

- 域名是成对的，推流和拉流有绑定关系，新增和修改时需要一并修改。
- 您需要确保一级域名已经备案过。如果因域名没有备案或涉及非法业务等原因导致推流或者拉流域名不可用，由此产生的任何后果由您自身承担。

推流与拉流域名配置生效后，系统会自动给该域名分配一个 CNAME。使用您的推流或拉流域名进行直播之前，需要您公司开发或服务运维人员在域名解析配置中添加对应的 CNAME 记录，让您的域名指向 CNAME。具体如何配置可以咨询域名供应商。

HTTPS 设置

HTTPS 设置是可选项。FLV/HLS 协议的直播流默认使用 HTTP，配置 SSL 证书后可以使用 HTTPS。

如果您没有在融云平台上传过证书，请点击新增证书，将证书上传。下图展示了提交证书的界面。

证书管理 / 新增 返回

提示:

- 1、不支持新增 MD5 加密签名算法证书。由于此类证书安全性较低，且部分主流浏览器已限制使用。
- 2、超过 44 KB 的 crt 文件在 IE11 浏览器上使用存在高风险，请确保您上传的每个 crt 文件大小都小于 44 KB。

*证书名称:

请输入证书内容

*证书内容:

请输入私钥内容

*私钥内容:

请输入备注

备注:

防盗链

防盗链配置是可选项。配置后会提升推拉流域名的安全。

开启后一定要配置推流和拉流地址有效期时间和加密 URL 的密文 KEY。

域名配置 证书管理 **防盗链**

防盗链配置

推流: KEY: 时长: 秒

拉流: KEY: 时长: 秒

在 20:00 至次日 2:00 进行的配置不会立即生效, 会在 2 点后依次配置生效, 在此时间段外进行配置会在 30 分钟后生效

防盗链开启后，如果观众端遇到弱网（融云 SDK 内部帮您在网络恢复后重新订阅成功）会比没有防盗链多出一些恢复订阅时间。

自行生成防盗链推拉流地址

如果您选择的推拉流模式为自行生成推拉流地址，在您的服务端生成推拉流地址时在地址中加入防盗链相关信息。

相比普通推拉流地址，生成防盗链推拉流地址还需要用到以下防盗链参数：

防盗链参数	描述	补充说明
wsTime	生成推拉流地址时的 UNIX 时间。防盗链地址中直接携带该参数。	格式为 16 进制 UNIX 时间。防盗链地址中直接携带该参数。
KEY	MD5 计算方式的密钥，在控制台配置，仅用于计算 wsSecret。	可以自定义。
wsSecret	播放 URL 中的加密参数。防盗链地址中直接携带该参数。	值是通过将 KEY，URI，wsTime 依次拼接的字符串进行 MD5 加密算法得出，即 $wsSecret = MD5(wsTime + URI + KEY)$ 。

防盗链的推拉流地址生成示例：

假设原始 url（已包含推/拉流域名 + AppName + StreamName）为：<http://cdn.rongcloud.cn/myappname/stream.flv>

1. 获取在控制台防盗链配置填入的 KEY，假设 KEY 为：rongcloud。

- 获取 wsTime，即生成推拉流地址时的 UNIX 时间，假设为 1601026312，转换成 16 进制 5f6db908。在计算 wsSecret 时需要用到。防盗链 URL 中也会直接携带该参数。
- 计算 wsSecret。防盗链 URL 中会直接携带该参数。
 - 获取计算 wsSecret 需要用到的 URI 参数值。需要用到您自定义的 AppName 与 StreamName。拼接规则为 `/{AppName}/{StreamName}`。从本例的原始 URL 可得出 URI 为 `/app/stream.flv`。
 - 依次拼接 wsTime + URI + KEY，获取签名字符串。在本例中该拼接字符串为：`5f6db908/app/stream.flvrongcloud`
 - 对签名字符串计算 md5hash，得到 wsSecret。即，`wsSecret = md5sum("5f6db908/app/stream.flvrongcloud") = 79aead3bd7b5db4adefb93a010298b5`
- 使用上述步骤中得到的 wsSecret 与 wsTime 参数，拼接成防盗链 URL：

<http://cdn.rongcloud.cn/app/stream.flv?wsSecret=79aead3bd7b5db4adefb93a010298b5&wsTime=5f6db908> 

当用户发起带时间戳防盗链的 url 请求后，CDN 服务器会对 url 内容进行校验，判断时间有效性及 MD5 值，两个值其中有一个不符合要求，返回 403。

生成防盗链代码示例（golang）

```
package main

import (
    "crypto/md5"
    "encoding/hex"
    "fmt"
    "strconv"
    "strings"
    "time"
)

func main() {
    secret, t := GenWsSecret("/live/abc", "key")
    fmt.Println(secret)
    fmt.Println(t)
}

func GenWsSecret(uri, key string) (string, string) {
    uri = fmt.Sprintf("%s", strings.TrimLeft(uri, "/"))

    t := strconv.FormatInt(time.Now().Unix(), 16)

    ori := fmt.Sprintf("%s%s%s", t, uri, key)

    h := md5.New()
    h.Write([]byte(ori))

    return hex.EncodeToString(h.Sum(nil)), t
}
```

提示

- 在 20:00 至次日 2:00 进行的配置不会立即生效，会在 2 点后依次配置生效。在此时间段外进行配置会在 30 分钟后生效。

2. 新配置生效后，原有的配置即刻作废。建议开发者避开业务高峰时段，并给还在使用老配置的房间用户发送提醒通知，重新订阅新地址。

融云 CDN 插件

依赖包说明

更新时间:2024-08-30

Web 内置 CDN 功能对 RTCLib 及依赖包有版本限制：

- **RTCLib**：要求使用 @rongcloud/plugin-rtc@5.1.10 或更高版本。
- **IMLib**：具体要求如下：
 - 如 IMLib 为 IMLib 5.X，所有版本均支持 Web 内置 CDN 功能。
 - 如 IMLib 为 Adapter SDK (imlib-v4-adapter 或 imlib-v4-adapter)，均支持 Web 内置 CDN 功能。
 - 如仍使用旧版 IMLib v2/v4，请注意存在以下最低版本限制（建议替换升级为对应的 Adapter SDK）：

SDK	包名	最低版本
IMLib v2	@rongcloud/imlib-v2	@2.9.9
IMLib v4	@rongcloud/imlib-v4	@4.4.9

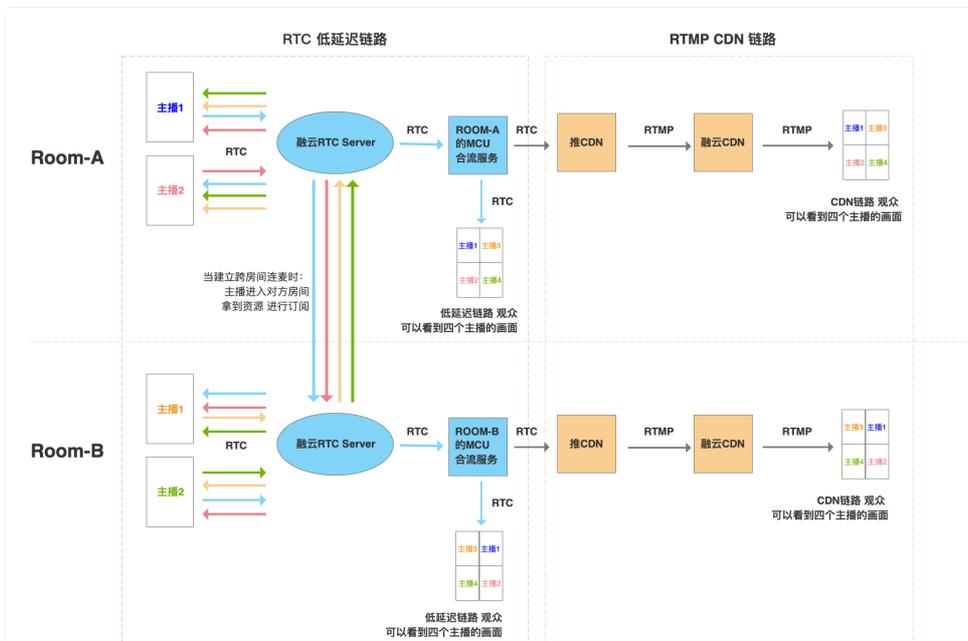
场景描述

当您使用 RTC SDK 做直播应用时，可以选择用 CDN 分发直播媒体流。在控制台开启 [融云 CDN](#) 服务并配置域名等信息后，观众端 APP 可以调接口来选择订阅 CDN 链路或者 RTC 低延迟链路的直播流。

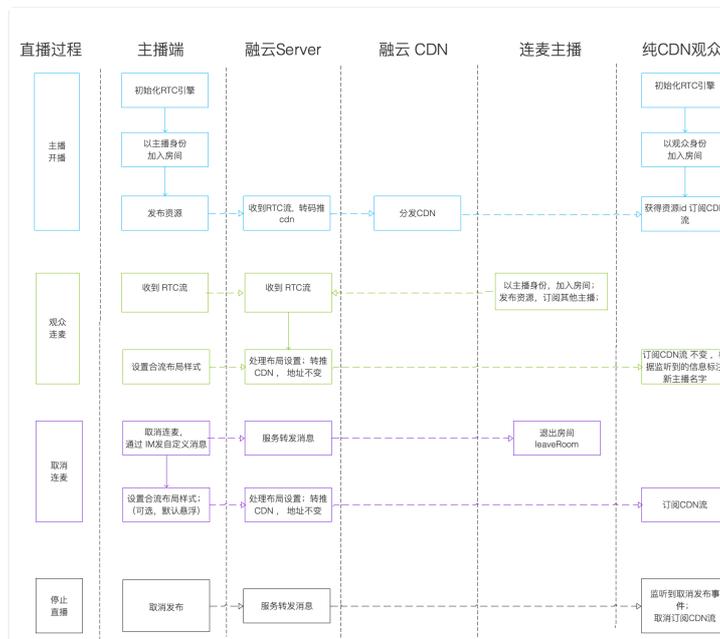
使用 CDN 拉流时观众端默认按 CDN 服务输出的视频格式拉流，也可以调接口设置指定的拉流分辨率、码率、帧率（说明：会触发 CDN 转码服务）。

需要说明的是主播端还是用 RTC 协议发布音视频资源，资源由融云 Server 来接收并转协议到 RTMP 并推给 CDN。此外观众端 SDK 可以感知到主播是否推流，服务端合流情况，无需 App Server 做过多的任务管理。

服务架构图：



主要业务流程图：



服务开通

融云 CDN 是付费增值服务，且需要开通后才能使用。详见[融云 CDN 服务配置](#)。

主播

提示

以下示例代码中的 `room` 指加入房间成功后获取到的房间实例。

开启/停用内置 CDN

如果您在控制台配置融云 CDN 为手动模式，可调用 `enableInnerCDN` 开启、停用向 CDN 推流。

提示

房间内需有资源才可操作内置 CDN

API 参考：[enableInnerCDN](#)

```
/**
 * 开启/停用推 CDN
 * @params enable true 为开启、false 为停用
 */
const { code } = await room.enableInnerCDN(enable: boolean)
```

主播设置 CDN 资源分辨率和帧率

具体可参考[合流布局](#)。

API 参考：[setOutputVideoResolution](#)

API 参考：[setOutputVideoFPS](#)

```
import { RCMCUConfigBuilder } from '@rongcloud/plugin-rtc'

// 获取构建器实例
const builder: RCMCUConfigBuilder = (room as RCLivingRoom).getMCUConfigBuilder()

// 设置视频分辨率
builder.setOutputVideoResolution(RCResolution.W640_H480)
// 设置视频帧率
builder.setOutputVideoFPS(RCFrameRate.FPS_15)

// 上传配置给 media server 以修改最终合流效果
const { code } = await builder.flush()
```

主播端之间 CDN 状态同步

1. 主播调 `joinLivingRoom` 加入房间时，如果房间的内置 CDN 被其他主播手动开启或停用，`joinLivingRoom` 方法会返回房间内的 CDN 开启状态，`CDNEnable` 为 `true` 时代表开启，为 `false` 时代表停用。

```
const { code, CDNEnable } = await rtcClient.joinLivingRoom('roomId', RCLivingType.VIDEO)
```

2. 主播加入房间后，可通过在 [registerRoomEventListener](#) 中增加注册 `onCDNEnableChange`，监听房间内 CDN 开启状态的变化。在房间内其他主播调 `enableInnerCDN` 方法开启或停用内置 CDN 时可收到通知。详细步骤可参考[房间事件监听器](#)。

```
// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
  /**
   * 房间内 CDN 状态被改变时触发
   * @param enable true 为开启、false 为停用
   */
  onCDNEnableChange (enable: boolean) {
    console.log('onCDNEnableChange', enable);
  }
})
```

观众

① 提示

以下示例代码中的 `room` 指加入房间成功后获取到的房间实例。

设置 CDN 观看地址参数

如需修改 CDN 观看地址相关参数，可在初始化 RTCLib 时传入以下参数：

- `pullInnerCDNProtocol`：设置观看地址直播拉流协议
- `pullInnerCDNUseHttps`：设置观看地址是否使用 https

具体可参考 [初始化](#) 以及 [IRCRTCInitOptions](#)。

获取房间内 CDN 观看地址

API 参考：[getCDNPlayUrl](#)

```
/**
 * 获取 CDN 资源对应的直播观看地址
 * @params resolution 分辨率，不传时，获取到的资源分辨率为主播端 MCU 资源的宽高
 * @params fps 帧率，不传时，获取到的资源帧率为主播端 MCU 资源的帧率
 * @returns CDNPlayUrl 为一个协议为 rtmp、flv 或 hls 的直播观看地址
 */
const { code, CDNPlayUrl } = await (room as RCAudienceLivingRoom).getCDNPlayUrl(resolution?:
RCResolution, fps?: RCFrameRate)
```

监听房间内 CDN 资源变动

观众加入房间后，可通过在 [registerRoomEventListener](#) 中增加注册以下事件，监听主播端 CDN 资源变动。

```
// 注册房间事件监听器，重复注册时，仅最后一次注册有效
room.registerRoomEventListener({
/**
 * 房间内主播把发布的资源推至 CDN 时触发
 * CDNInfo 包含分辨率和帧率
 */
onCDNInfoEnable (CDNInfo: {resolution: RCResolution, fps: RCFrameRate}) {
console.log('onCDNInfoEnable', CDNInfo);
},
/**
 * 主播停止推 CDN 时触发
 */
onCDNInfoDisable () {
console.log('onCDNInfoDisable');
}
/**
 * 主播改变推 CDN 的分辨率或帧率时触发
 * CDNInfo 包含分辨率和帧率
 */
onCDNInfoChange (CDNInfo: {resolution: RCResolution, fps: RCFrameRate}) {
console.log('onCDNInfoChange', CDNInfo);
}
})
```

播放 CDN 直播资源

- 推荐本地下载 [VLC](#) 播放器进行播放。
- 如需在页面内播放，可引入第三方插件。例如[哔哩哔哩 flv.js](#)、[百度 cyberplayer](#) 等。

如何排查播放问题

- 如果 CDN 观看地址不可播放，可先排查观看地址是否在防盗链配置的有效期内。如果超过有效期，可通过获取新 CDN 观看地址解决。
- 如果在使用第三方插件播放 CDN 资源时遇到问题，可先使用 VLC 播放，排除资源本身不可用的问题。

第三方插件参考资源：

如果遇到第三方插件的使用问题，您可以参考第三方插件的集成文档。以下仅列出部分资源：

- **cyberplayer 兼容性**：<https://cloud.baidu.com/doc/MCT/s/zjwvz4w4z>
- **cyberplayer demo 地址**：<http://cyberplayer.bcelive.com/demo/new/index.html>

直播协议支持对比

下表列出了 VLC 播放器、第三方插件、浏览器等对各直播协议的支持情况：

协议	VLC	手机浏览器、PC safari	哔哩哔哩 flv.js	百度 cyberplayer
hls	支持	支持		支持
flv	支持		支持 (比 cyberplayer 体积小)	支持
rtmp	支持			

客户端旁路推流

更新时间:2024-08-30

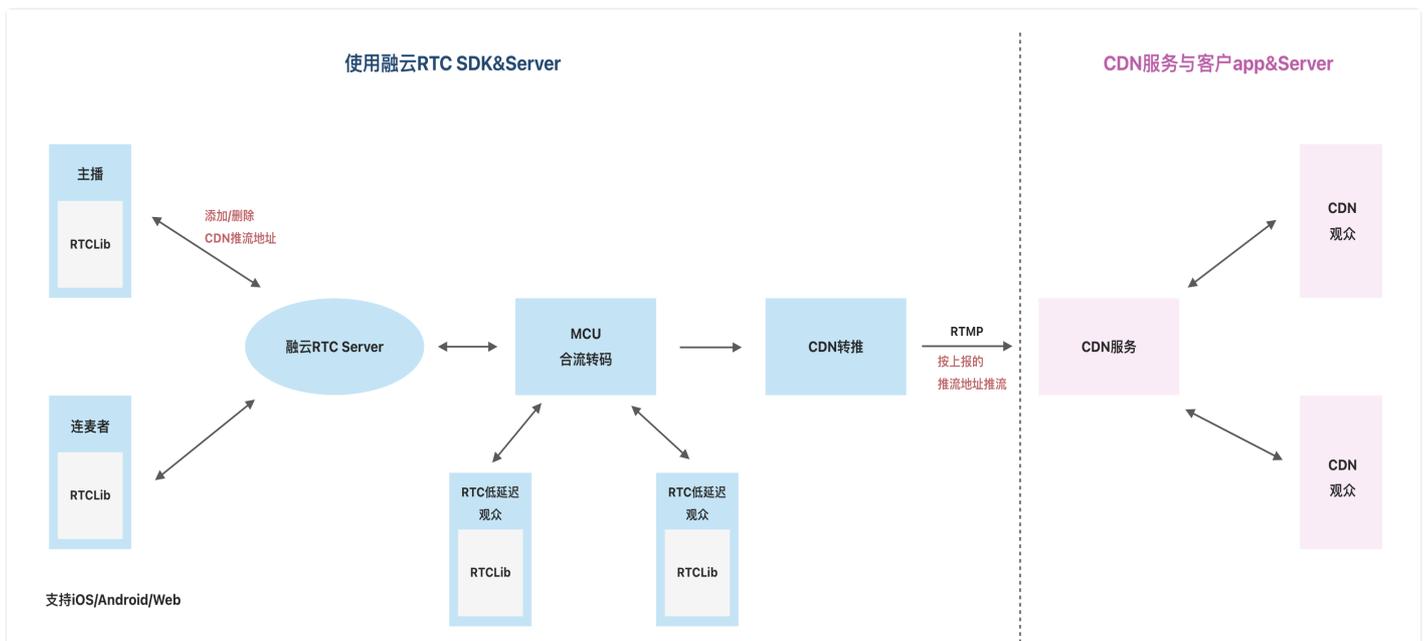
旁路推流：将融云实时音视频流转为 rtmp 流媒体协议，推流到 CDN 服务。

旁路推流支持两种控制方式，本文仅介绍方案一：

- 方案一：使用客户端接口进行控制，使用 `addPublishStreamUrl` 接口配置 CDN 地址。
- 方案二：使用服务端 API 的 `/rtc/mcu/config` 接口进行控制。本文不做介绍，具体请参见服务端文档[旁路推流](#)。

业务链路

融云直播支持将实时音视频流转推到 CDN 服务，业务链路如下图所示：



配置直播 CDN 推流地址

当主播加入直播间之后，就可设置 CDN 推流地址。

设置 CDN 地址有以下几点要求：

1. 必须开通音视频服务和直播服务。
2. 房间模式必须为直播模式。
3. 设置的 CDN 地址不能为空。
4. 最多设置 5 个 CDN 地址。

RCCMCUConfigBuiler

RCMCUConfigBuilder 是一个构建器类型，该类实例中包含了构建一个有效布局配置和 CDN 推流配置所需要的所有接口，构建接口均为同步接口且支持链式调用。

RCMCUConfigBuilder 实例对象内部会在每一步构建接口调用时修改内存中存储的配置信息。当构建完成后，需要调用 flush() 接口上传配置到服务器以使配置生效。

提示

以下示例代码中的 room 为主播加入直播房间成功后获取到的 RCLivingRoom 类型实例对象。

API 参考：[getMCUConfigBuilder](#)

```
import { RCMCUConfigBuilder } from '@rongcloud/plugin-rtc'

// 获取构建器实例
const builder: RCMCUConfigBuilder = room.getMCUConfigBuilder()
```

增加推流地址

提示

最多可以添加 5 个推流地址。当推流地址超出 5 个时，当次调用传入的地址将全部无效。

API 参考：[addPublishStreamUrls](#)

```
builder.addPublishStreamUrls(['url_1', 'url_2'])
```

移除推流地址

API 参考：[removePublishStreamUrls](#)

```
builder.removePublishStreamUrls(['url_1', 'url_2'])
```

配置生效

提示

在调用该方法前，所有接口调用只会对本地配置进行修改，不会产生实际效果。

API 参考：[flush](#)

```
// 上传配置给 media server 以修改最终合流效果
const { code } = await builder.flush()
```

CDN 推流回调

融云在低延迟直播旁路推流到融云 CDN 或第三方 CDN 时，支持将推流的状态变化实时通知您的服务器。具体请参见音视频服务端文档：

- [CDN 推流回调](#)

3A 音频处理

软 3A 音频处理

更新时间:2024-08-30

自 5.6.19 版本开始，RTCLib 提供内置针对麦克风采集音频的软 3A 处理方案，以优化部分场景下的音频采集质量。

目前该软 3A 方案仅对 createMicrophoneAndCameraTracks 与 createMicrophoneAudioTrack 两个接口获取到的 RCLocalAudioTrack 数据可用。

代码示例

```
const { code, track } = await rtcClient.createMicrophoneAudioTrack();

if (code !== RCRTCCode.SUCCESS) {
  console.error(`麦克风音频获取失败 -> ${code}`);
  return;
}

// 对采集的音频应用软 3A 处理
const res = await (track as RCMicphoneAudioTrack).apply3ANoiseWorklet();
if (res.code !== RCRTCCode.SUCCESS) {
  console.warn(`软 3A 处理失败 -> ${res.code}`);
}

// 发布资源
// ...

// 需要注意：3A 处理需要在资源发布之前进行，否则无法生效
```

WASM 模块

软 3A 处理依赖内置的 WASM 模块进行，该模块会在业务层调用 apply3ANoiseWorklet 函数时从远程 CDN 服务下载；

- WASM_URL: <https://cdn.ronghub.com/plugin-rtc/wasm/5.0.0-alpha.2/AudioProcessing.wasm>
- SCRIPT_URL: <https://cdn.ronghub.com/plugin-rtc/wasm/5.0.0-alpha.2/process-worklet.js>

建议开发者将以上两个文件下载至自己的工程内，并与您的其他资源一同管理，避免在某些特殊情况下远程资源加载失败造成异常。

将资源下载到本地后，在 RTCLib 初始化时，可通过配置将地址传入 SDK 内部，以替换默认下载地址；

```
// 初始化 RTCLib 时配置
{
// 3A 配置
AAWorklet: {
wasmUrl: './xxxxx/AudioProcessing.wasm',
scriptUrl: './xxxxx/process-worklet.js'
}
})
```

大小流

更新时间:2024-08-30

大小流模式是指在发布资源时上传一大一小两道视频流。

SDK 默认打开发布大小流功能，即每个用户在发布视频资源时自动发布大小两个视频流。小流的分辨率默认跟随大流。

提示

在多人音视频通话过程中，大小流模式可有效减少下行带宽占用。订阅方可按需订阅小流。

小视频流与大视频流的分辨率对应关系如下:

大流分辨率	小流分辨率	比例
176X144	176X144	11:9
180X180	180X180	1:1
256X144	256X144	16:9
240X180	240X180	4:3
320X180	256X144	16:9
240X240	180X180	1:1
320X240	240X180	4:3
360X360	180X180	1:1
480X360	240X180	4:3
640X360	256X144	16:9
480X480	180X180	1:1
640X480	240X180	4:3
720X480	240X180	3:2
848X480	256X144	9:5
960X720	240X180	4:3
1280X720	256X144	16:9
1920X1080	256X144	16:9

提示

音频资源不存在大小流概念，因此以下接口对于音频资源没有任何效果。以下示例代码中的 `room` 指加入房间成功后获取到的实例。

发布方开关大小流

开启功能后，发布资源时会发布大小两道流。

API 参考：[publish](#)

```
const { code } = await room.publish([
  {
    track: videoTrack,
    pubTiny: true // 指定同时发布小流资源
  }
])
```

设置小流属性

开启了大小流功能后，小流会自动使用默认属性。您也可以通过 `pubTiny` 为小流视频设置指定的分辨率和码率，示例如下：

API 参考：[publish](#)

```
// 通过 import { RCResolution, RCFrameRate } from '@rongcloud/plugin-rtc' 获取相关枚举值
const { code } = await room.publish([
  {
    track: videoTrack,
    // 通过 pubTiny 指定小流分辨率与帧率
    pubTiny: { frameRate: RCFrameRate.FPS_15, resolution: RCResolution.W176_H144 }
  }
])
```

订阅方切换大小流流

API 参考：[subscribe](#)

```
// 该接口支持同时订阅多道资源，且支持订阅指定视频资源的大流或小流，默认订阅大流
const { code } = await room.subscribe([
  // 订阅 videoTrack_1 的大流
  videoTrack_1,
  // 订阅 videoTrack_2 的小流
  { track: videoTrack_2, subTiny: true }
])
```

对于已经订阅的资源，重新通过 `room.subscribe()` 指定订阅目标资源的小流或大流即可。

分辨率/码率/帧率设置

分辨率 & 帧率

更新时间:2024-08-30

提示

SDK 从 5.6.6 版本开始支持采集和发布竖版视频资源。

Web RTCLib 5.0 中分别定义了 [RCFrameRate](#) 和 [RCResolution](#) 两个枚举型数据，用于在采集视频资源时指定融云推荐的视频流帧率与分辨率配置。

```
import { RCFrameRate, RCResolution } from '@rongcloud/plugin-rtc'
```

按指定配置从摄像头中采集视频流

API 参考：[createCameraVideoTrack](#)

```
const { code, track: videoTrack } = await rtcClient.createCameraVideoTrack('RongCloudRTC', {  
  // 默认帧率为 15  
  frameRate: RCFrameRate.FPS_15,  
  // 默认分辨率为 640 * 480  
  resolution: RCResolution.W640_H480  
})
```

按指定配置获取屏幕共享视频流

API 参考：[createScreenVideoTrack](#)

```
const { code, track: videoTrack } = await rtcClient.createScreenVideoTrack('screenshare', {  
  // 默认帧率为 15  
  frameRate: RCFrameRate.FPS_15,  
  // 默认分辨率为 1280 * 720  
  resolution: RCResolution.W1280_H720  
})
```

上行码率

提示

SDK 内部已实现了动态的码率计算，非必要情况下，或当开发者无法准确评估业务所需要的带宽时，不建议主动调用修改。浏览器 WebRTC 是动态码率，SDK 设置的码率是推荐码率。最终码率及画质效果，由当前运行的网络带宽、网络质量及计算机性能所决定。

为视频设置上行码率

API 参考：[videoTrack.setBitrate](#)

```
const { code, track: videoTrack } = await rtcClient.createCameraVideoTrack('RongCloudRTC');

/**
 * 为本地流设定上行码率，仅视频流有效，音频默认 15 kbps，不支持修改
 * @description 当 `max` 或 `min` 值为 `0` 时，取动态码率计算结果
 * @param max 最大码率
 * @param min 最小码率
 * @param start 起始码率
 */
const {max, min, start} = { max: 1000, min: 200, start: 700};
videoTrack.setBitrate(max, min, start);
```

码率设置会在下一次调用 `room.publish()`、`room.unpublish()`、`room.subscribe()`、`room.unsubscribe()` 后生效。

发布自定义流

捕获自定义媒体流

更新时间:2024-08-30

除了从本地摄像头、麦克风设备捕获默认音视频资源外，SDK 还支持获取本地/网络文件流，以及转化浏览器的 MediaStream 实例为 SDK 的音视频资源。

自定义转换 mediaStream

使用 [createLocalTracks](#) 可以将浏览器原生 API 获取的 mediaStream 对象转换为 RCLocalTrack 实例对象，满足 App 自行转换的需求。

```
/**
 * 根据 MediaStream 实例对象创建 RCLocalTrack 实例
 * @param tag 轨道标识，包含 A-Z、a-z、0-9、+、=、- 的字符串。trackId 中将包含该 tag。
 * @param stream MediaStream 实例
 * @param options 可用于指定 `withoutVideo` 与 `withoutAudio` 以剔除视轨与音轨
 */
const { code, tracks } = await rtcClient.createLocalTracks(tag: string, stream: MediaStream, options?: ICreateLocalTrackOptions)
```

捕获本地或网络文件流

从 RTCLib v5.0.5 开始，新增了 [createLocalFileTracks](#) 接口以方便用户实现自定义媒体流的发布。

```
/**
 * 根据本地或网络媒体文件资源创建 `RCLocalFileTrack` 实例
 * @param tag 资源标识
 * @param file 网络文件地址，或通过 <input type='file'> 获取到的 File 实例
 * @param options 可用于指定 `withoutVideo` 与 `withoutAudio` 以剔除视轨或音轨
 */
createLocalFileTracks (tag: string, file: string | File, options?: ICreateLocalTrackOptions): Promise<{
  code: RCRTCCode, tracks: RCLocalFileTrack[] }>;
```

从网络媒体文件获取音视频流

提示

由于浏览器同源策略限制，网络媒体资源要求与调用 [createLocalFileTracks](#) API 的页面同源，或者允许跨域访问。

```
const { code, tracks } = await rtcClient.createLocalFileTracks(  
  'tag', // track 标识  
  'https://abc.com/a.mp4', // 网络资源 url 地址  
  {  
    withoutAudio: false, // 当值为 true 时将不产生音轨，默认为 false  
    withoutVideo: false, // 当值为 true 时将不产生视轨，默认为 false  
  }  
)
```

从硬盘存储的媒体文件获取音视频流

使用 [createLocalFileTracks](#) 从硬盘存储的媒体文件获取音视频流。

```
// 此处假定已通过 <input type="file"> 成功获取 File 类型实例变量 file  
const { code, tracks } = await rtcClient.createLocalFileTracks('tag', file)
```

发布自定义媒体流

使用房间实例的 [publish](#) 方法发布媒体流。

```
const { code } = await room.publish(tracks)
```

水印处理

更新时间:2024-08-30

融云支持在视频流上添加水印。添加水印有多种控制方式，本文仅介绍方案一：

- 方案一：基于 canvas 实现添加水印，并发布带水印的视频流。客户端发布的视频流即带有图片水印，因此订阅分流或合流的直播观众均会看到带水印的视频流。
- 方案二：使用客户端 SDK 提供构建 MCU 配置的方法 `addPictureWaterMark`，为合流中为单道视频流（子视图）添加图片水印。仅订阅合流的直播观众可看到水印。详见[合流布局](#)。
- 方案三：使用服务端 API 的 `/rtc/mcu/config` 接口，在服务端处理，添加时间戳水印、文字水印或图片水印。这种方式支持为单人视频流或合流视频添加水印，但只有订阅合流的观众可看到带水印的视频流。本文不介绍服务端的处理方案，如有需要，请参见服务端文档[直播合流](#)。

方案一、二适用于实现 App 客户端用户自主添加个性化水印，例如每个主播可自行设置贴纸、挂件等；方案三更适用于由 App 添加统一风格的水印。

客户端与服务端添加的水印相互独立。如果同时使用，则订阅合流的观众可能会看到水印叠加。

前提条件

添加水印功能主要使用到了 `canvas.captureStream` 捕获 canvas 中绘制出的流对象。兼容性情况可参考 `canvas.captureStream` [兼容性情况](#)。

实现流程

1. 采集音视频资源。
2. 创建 video 标签播放视频流，用于将视频绘制到 canvas 画布中。
3. 创建 image 实例，加载水印图片。
4. 创建 canvas 标签，将视频和水印绘制到 canvas 画布中。
5. 使用 `canvas.captureStream` 从画布中采集视频轨道，转为融云 RTC 视频轨道，通过 RTC SDK 的 `publish` 接口发布到房间中。

步骤一：采集音视频资源

以获取摄像头、麦克风的资源为例，使用浏览器原生 API `MediaDevices.getUserMedia()` 获取 `mediaStream`。

```
const mediaStream = await navigator.mediaDevices.getUserMedia({
  audio: true,
  video: {
    width: 640,
    height: 480
  }
})
```

如需为屏幕共享需要添加水印，您也可以从屏幕分享源采集获得资源。如果播放文件，可以从文件获取资源。

步骤二：播放音视频资源

创建播放视频的 video 元素。元素宽高需与采集的视频资源的分辨率宽高一致，避免拉伸。本端不需要播放自己的声音，因此示例中已将音轨作静音处理。

```
// 使用 video 标签播放摄像头视频；不播放自己的声音，避免回声
<video muted id="videoEl" style="display:none; width:640px; height:480px"></video>
```

播放资源。以下示例使用了 [srcObject](#)：

```
/**
 * 播放音、视频
 */
const videoEl = document.getElementById('videoEl');
videoEl.srcObject = mediaStream;
```

步骤三：加载水印图片

创建 image 实例，加载水印图片。注意，加载线上图片时，需允许跨域访问。参见 [允许图片和 canvas 跨源使用](#)。

```
const loadImage = (imageUrl) => {
  return new Promise((resolve) => {
    const image = new Image();
    image.src = imageUrl;
    image.onload = () => resolve(image);
  });
}

const image = await loadImage('/assets/image/watermark.jpg')
```

步骤四：将视频和水印绘制到 canvas 画布中

准备 canvas 画布元素，元素大小也需与采集的视频分辨率一致。

```
// 绘制视频和水印
<canvas id="canvasEl" style="width:640px; height:480px"></canvas>
```

在 canvas 画布上绘制视频和水印。

```
const canvasEl = document.getElementById('canvasEl');
const loop = () => {
  const ctx = canvasEl.getContext('2d');
  // 将视频绘制到画布中
  ctx.drawImage(videoEl, 0, 0, 640, 480);
  // 将水印图片绘制到画布中，可以控制水印的位置和大小
  ctx.drawImage(image, 0, 0, image.width, image.height);
}
```

循环绘制。setInterval 和 requestAnimationFrame 都可以进行循环绘制。

```
setInterval(loop, 1000/30);
```

提示：原始视频渲染和解码不占 JS 线程，但循环绘制 canvas 流程跟其他 JS 执行代码处于同一线程，会因渲染进程阻塞而造成卡顿。

- [setInterval](#)：存在执行间隔不精确的问题。delay 参数只是指定了把动画代码添加到浏览器 UI 线程队列中以等待执行的时间，如果队列前面已经加入了其他任务，动画就必须等前面的任务执行完才可以执行。
- [requestAnimationFrame](#)：限制必须停留在绘制页面。可保证执行步伐与系统的绘制频率一致，即保证回调函数在屏幕每一次的绘制间隔中只被执行一次，这样就不会引起丢帧现象，也不会导致动画出现卡顿的问题。但是 [requestAnimationFrame](#) 运行在后台标签页时，会被暂停调用以提升性能和电池寿命，使用 [requestAnimationFrame](#) 时，需保持停留在绘制页面。

步骤五：发布带水印的视频流

从 canvas 元素中抓取带水印的视频流。参见 [captureStream](#)。

```
const canvasStream = canvasEl.captureStream();
```

将带水印的视频 track 和之前获取的音频 track 转为融云 [publish](#) 接口接受的入参类型。这一步需要调用融云 SDK 的 [createLocalVideoTrack](#) 与 [createLocalAudioTrack](#) 方法。

```
/**
 * 转化带水印的视频 track 为 融云 publish 接口参数类型
 */
const captureVideoTrack = canvasStream.getVideoTracks()[0];
const audioTrack = mediaStream.getAudioTracks()[0];
const { track: newVideoTrack } = await rtcClient.createLocalVideoTrack('RongCloudRTC',
captureVideoTrack)
const { track: newAudioTrack } = await rtcClient.createLocalAudioTrack('RongCloudRTC', audioTrack);
```

在房间中发布音频和带水印的视频。注意，您需要在加入房间时获取 [RCLivingRoom](#) 实例。若发布的资源 tag 及媒体类型重复，后者将覆盖前者进行发布。

```
/**  
 * 发布音频和带水印的视频  
 * crtRoom 为加房间返回的 room 对象  
 */  
crtRoom.publish([newVideoTrack, newAudioTrack]);
```

屏幕共享

更新时间:2024-08-30

该功能要求 **Chrome** 浏览器版本在 72 以上

获取屏幕共享流

API 参考：

- 获取屏幕共享视频流：[createScreenVideoTrack](#)
- 获取屏幕共享视频流并尝试获取音频（Mac OSX 系统无法获取到音频）[createScreenWithAudioTracks](#)

```
const { code, track: screenTrack } = await rtcClient.createScreenVideoTrack()

if (code !== RCRTCCode.SUCCESS) {
  console.warn('捕获屏幕共享流失败 ->', code)
  return
}
```

注意，**Electron** 中获取屏幕共享流时，必须指定 `chromeMediaSourceId`，其值通过 **Electron** 官方 API [desktopCapturer](#) 工具获取。

```
const { code, track } = await rtcClient.createScreenVideoTrack('screenshare', {
  chromeMediaSourceId: '<chromeMediaSrouceId>'
})
```

发布屏幕共享流

API 参考：[publish](#)

```
import { RCLocalTrack } from '@rongcloud/plugin-rtc';

const { code } = await room.publish([ screenTrack ])

if (code !== RCRTCCode.SUCCESS) {
  console.warn('发布屏幕共享流失败 ->', code)
}

// 监听屏幕共享流结束事件
screenTrack.on(RCLocalTrack.EVENT_LOCAL_TRACK_END, (track: RCLocalTrack) => {
  // 屏幕共享流已结束通知
})
```

结束屏幕共享

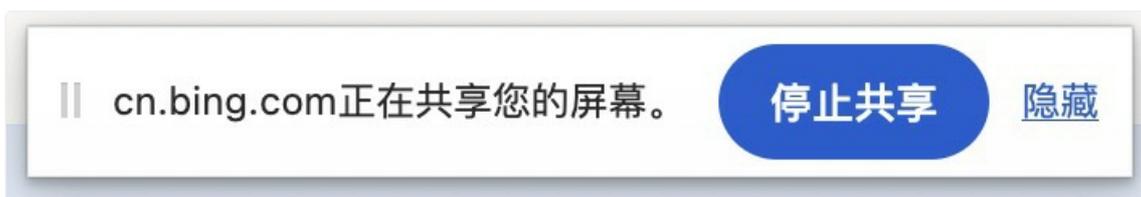
通过 `destroy` 方法结束屏幕共享

可以通过 `screenTrack.destroy()` 方法销毁屏幕共享流，同时会触发 `RCLocalTrack.EVENT_LOCAL_TRACK_END` 事件。

销毁后，SDK 内部处理取消发布流程，开发者无需手动调用 `unpublish` 来取消流发布。

通过浏览器默认 UI 按钮结束屏幕共享

在获取屏幕共享流时，浏览器会弹出内置的屏幕共享流结束按钮，不同的浏览器按钮样式不同，参考下图。



当用户点击后，`track` 实例会触发 `RCLocalTrack.EVENT_LOCAL_TRACK_END` 事件。

- 当 RTCLib 版本 $< 5.6.20$ 时，开发者需要手动调用 `unpublish` 来取消流发布。
- 当 RTCLib 版本 $\geq 5.6.20$ 时，SDK 内部处理取消发布流程，开发者无需手动调用 `unpublish` 来取消流发布。

```
// 监听屏幕共享流结束事件，RCLocalTrack 可以通过 import { RCLocalTrack } from '@rongcloud/plugin-rtc' 获取
screenTrack.on(RCLocalTrack.EVENT_LOCAL_TRACK_END, (track: RCLocalTrack) => {
  // 仅当 SDK 版本 < 5.6.20 时需要调用
  room.unpublish([ track ])
})
```

通话数据统计

更新时间:2024-08-30

提示

以下示例代码中的 `room` 指加入房间成功后获取到的房间实例。

SDK 通过 [IRCRTCReportListener](#) 提供详细的通话状态数据，上层依据此数据可进行提示，状态判断等处理。

监听通话质量数据

在直播模式下、会议模式下均支持监听通话质量数据。

1. 获取房间实例。

- 会议模式：[RCRTCRoom](#)
- 直播模式：主播角色用户使用 [RCLivingRoom](#) 实例、观众角色用户使用 [RCAudienceLivingRoom](#) 实例。在直播模式下，如果观众不加入房间直接订阅 `liveUrl`，也可以在获取 [RCAudienceClient](#) 实例后使用 `registerReportListener` 注册 [IRCRTCReportListener](#) 监听。

2. 使用 `registerReportListener` 注册监听器。

```

import { IRCRTCStateReport, IRCCandidatePairStat, IRCTrackStat, ILiveAudioState } from
 '@rongcloud/plugin-rtc'

// 注册房间质量数据监听器
room.registerReportListener({
  /**
   * 用于接收状态数据报告
   * @param report
   */
  onStateReport (report: IRCRTCStateReport) {
    /**
     * 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
  },
  /**
   * 上报直播房间内音频合流的音源信息
   * 仅在直播房间内观众端有效
   * ILiveAudioState 中包含 userId、trackId、audioLevel
   * @since @rongcloud/plugin-rtc@5.6.3
   */
  onReportLiveAudioStates (liveAudioStates: ILiveAudioState[]) {
    console.log('liveAudioStates', liveAudioStates);
  }
})

```

[IRCRTCReportListener](#) 提供两个方法：

- [onStateReport](#)：（适用于会议和直播场景）上报音视频统计数据，例如音视频流相关用户数据、音视频流质量数据、网络状态与质量数据等。
- [onReportLiveAudioStates](#)：（仅限直播场景，要求 SDK 版本 $\geq 5.6.3$ ）上报合流的各主播音源属性 [ILiveAudioState](#)，其中包含每个主播的音量数据。

通信质量数据

SDK 会通过 [IRCRTCReportListener](#) 的 [onStateReport](#) 方法上报音视频质量统计数据，例如音视频流相关用户数据、音视频流质量数据、网络状态与质量数据等。

- [IRCRTCStateReport](#)：RTCPeerConnection 的详细状态数据
- [IRCCandidatePairStat](#)：连接状态数据
- [IRCTrackStat](#)：流状态数据

直播合流中各音源音量

提示

- 5.6.3 版本增加了 `onReportLiveAudioStates` 回调方法，仅限直播场景下使用。
- 5.6.3 之前的版本，当观众订阅音频合流时只能获取合流整体的音量，无法单独获取每个主播的音量。

从 5.6.3 开始，直播场景下观众订阅音频合流，SDK 会通过 [IRCRTCReportListener](#) 的 `onReportLiveAudioStates` 方法实时上报音频合流的声音状态。例如当前音频合流由 A B C 三个主播音频流产生，此时 A 在发声，SDK 会通过该方法回调 `ILiveAudioState` 类型的数组，数组中包含 A 主播的音量 `audioLevel` 取值是 0~100。

```
export interface ILiveAudioState {  
  /**  
   * 用户 id  
   */  
  userId: string,  
  /**  
   * 资源 trackId  
   */  
  trackId: string,  
  /**  
   * 音量 0-100  
   */  
  audioLevel: number  
}
```

上报音量

当业务层需要获知当前房间内的人员说话状态时，可以通过该接口音量数据判断处理。

1. 获取房间实例。
 - 会议模式：[RCRTCRoom](#)
 - 直播模式：主播角色用户使用 [RCLivingRoom](#) 实例、观众角色用户使用 [RCAudienceLivingRoom](#) 实例。
2. 构建 [IAudioLevelChangeListener](#) 实例，在房间实例的 `onAudioLevelChange` 方法中传入。

```
const handler = (audioLevelReportList: {track: RCLocalAudioTrack | RCRemoteAudioTrack, audioLevel:  
number}[]) => {}  
// 注册后，SDK 会以每秒一次的频率进行回调  
room.onAudioLevelChange(handler)
```

信令管理

更新时间:2024-08-30

您可以根据自己的业务需求创建自定义信令消息。以下 RTC 房间实例提供发送、接收自定义信令功能：

- 会议模式下 [RCRTCRoom](#) 实例
- 直播模式：主播角色用户的 [RCLivingRoom](#) 实例

接收信令

加入房间成功后，可以通过调用 `room.registerRoomEventListener()` 注册 `onMessageReceive` 事件监听器。

当房间内有人发送信令时，注册的监听函数会被调用。

发送信令

使用房间实例上提供的 [sendMessage](#) 方法发送信令。

```
const { code } = await room.sendMessage(name, content)
```

参数	类型	必填	说明
name	String	是	信令名
content	any	是	信令内容

自定义加密

更新时间:2024-08-30

RTCLib 从 5.7.0 版本开始支持自定义加密，用户可以自定义加密算法，实现对音频帧和视频帧的加密与解密。

自定义加密机

RTCLib 内部使用 H.264 视频编解码和 Opus 音频编解码，开发者可以按需实现自定义的加解密算法，并在 RTCLib 初始化时传递给 RTCLib。

```
import { installPlugin } from '@rongcloud/imlib-next';
import { IRCEncryptorHooks, installer } from '@rongcloud/plugin-rtc';

// 定义加解密钩子
const encryptor: IRCEncryptorHooks = {
  encode(chunk) {
    if (chunk instanceof RTCEncodedVideoFrame) {
      // 自定义 H.264 视频加密算法
      return encodeH264(chunk)
    } else {
      // 自定义 Opus 音频加密算法
      return encodeOpus(chunk)
    }
  },
  decode(chunk) {
    if (chunk instanceof RTCEncodedVideoFrame) {
      // 自定义 H.264 视频解密算法
      return decodeH264(chunk)
    } else {
      // 自定义 Opus 音频解密算法
      return decodeOpus(chunk)
    }
  },
};

// 初始化 RTCLib
const rtcClient = installPlugin(installer, { encryptor })
```

启用加密机

从 5.7.0 开始，RTCLib 新增了 `joinRTCRoomWithOptions` 方法用于加入房间。通过指定 `encrypt` 参数为 `true`，即可启动自定义加密机，对房间内的上行音频帧和视频帧进行加密，对房间内的下行音频帧和视频帧进行解密。

```
const { code, room } = await rtcClient.joinRTCRoomWithOptions('<Room-Id>', {
  // 启用加密机
  encrypt: true
})
```

原 `joinRTCRoom` 方法自 5.7.0 开始废弃，且不支持自定义加解密功能。

3.X 升级到 5.X

更新时间:2024-08-30

RTCLib SDK 5.X 是 Web 客户端 SDK 的最新版本。相对于 3.X 版本，5.X 版本功能更丰富，更稳定，并在之前版本上修复了大量问题，我们建议融云客户尽早升级至新版 RTCLib SDK。

升级概述

RTCLib 5x SDK 与旧版 SDK 不兼容，无法平滑升级。请根据 RTCLib 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。

关于 Adapter 的说明

融云同时为已集成 RTCLib 3.X 版本的客户提供了[基于 RCRTCAdapter 库替换升级选项](#)。

注意：

- Adapter 库仅用于后续维护，仅提供问题修复，不会在旧版 SDK 基础上增加新功能。
- 我们推荐使用融云旧版 SDK 客户使用新版本 5.X SDK。

更新日志

5.7.2

更新时间:2024-08-30

发布日期：2024/06/28

- 允许基于 Chromium 的浏览器在配置 Insecure origins treated as secure 白名单的情况下使用 http 协议进行 RTC 业务。
- 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。

5.7.1

发布日期：2024/06/05

功能优化

- 优化了 SDK 内部部分逻辑。

5.7.0

发布日期：2024/05/07

- 新增了支持房间音视频流数据的自定义加解密功能，使用说明参考：[自定义加密](#)。
- 新增了 `joinRTCRoomWithOptions` 接口，现有接口 `joinRTCRoom` 声明废弃。
- 优化了关键接口参数校验流程，避免参数类型错误导致的接口调用异常。
- 修复了 ice 连接中断后可能启动多个定时器进行重试，进而导致的频繁调用 `/exchange` 接口问题。
- 修复了 IM 连接中断后，会立即被踢出房间问题；受影响 SDK 版本包括 5.6.16 至 5.6.21。
- 修复了被踢出房间后，SDK 内部房间心跳未及时终止问题，留存较多无效日志。

5.6.21

发布日期：2024/04/01

- 修复 5.6.20 版本 SDK 引入的加入房间失败（错误码：53009）问题

5.6.20

发布日期：2024/03/29

- 优化屏幕共享流结束流程，屏幕共享流结束后，不再需要业务层调用 `unpublish` 接口
- 修复 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
- 兼容 Safari 17 浏览器下 `RTCPeerConnection` 质量数据统计报告
- 修复质量数据报告中无 `rtt` 数据问题
- 修复部分日志打印错误问题

5.6.19

发布日期：2024/01/30

- 修复 mediaServer 数据中心漂移问题
- 重构 3A 功能模块。RTCLib 提供内置的针对麦克风采集音频的软 3A 处理方案，以优化部分场景下的音频采集质量。

5.6.18

发布日期：2023/12/28

- 修复房间内观众获取带下划线的 userId 错误的问题。
- 修复播放接口业务传参序列化日志记录时报错的问题。

5.6.17

发布日期：2023/10/08

- 修复 iOS 14.x 上第二次进行媒体交互不成功的问题。

5.6.16

发布日期：2023/09/13

功能优化

- 房间事件监听器 ([IRoomEventListener](#)) 的 [onKickOff](#) 方法返回的踢出房间原因枚举 [RCKickReason](#) 增加了三个枚举值：
 - [IM_DISCONNECTED](#) : 3 (IM 断开连接)
 - [IM_LOGOUT](#) : 4 (IM 主动断开连接)
 - [OTHER_CLIENT_IM_CONNECTED](#) : 5 (用户在其他设备连接 IM，被挤下线)

5.6.15

发布日期：2023/07/20

功能优化

- 质量数据的上行数据 [IRCTrackStat](#) 增加了往返时延 ([rtt](#)) 字段。

5.6.14

发布日期：2023/07/04

- 修复 Chromium 86 版本 (360 极速浏览器等) 中，媒体 readyState 状态异常导致的播放问题。

5.6.13

发布日期：2023/06/08

- 修复偶现的业务层监听事件被多次触发的问题

5.6.12

发布日志：2023/05/31

- `RCRemoteAudioTrack` 远端音频流增加 `getVolume()` 方法以获取播放时设定的音量值
- 修复播放音频时，`volume` 音量参数失效问题
- 修复修改未发布的本地资源 `muted` 状态无结果返回问题

5.6.11

发布日期：2023/05/11

功能优化

- 优化内部代码结构，以降低代码复杂度，提升功能稳定性
- 优化对 `Proxy` 类型数据的支持，以提升 Vue 3 用户的开发体验

问题修复

- 修复 SDP 协议中 `maxaveragebitrate` 带宽赋值错误
- 修复修改本地流状态，接口始终返回 `RCRTCCode.SUCCESS` 问题
- 修复 Chrome 112 及以上版本浏览器中质量数据统计失效问题

5.6.9

发布日期：2023/02/13

问题修复

- 浏览器 `version` 取不到值时，webRTC 使用 `unified-plan` 协议，具体表现为：支持在 chrome 浏览器模拟移动设备中调试音视频功能
- `Promise.any` 兼容低版本浏览器
- 修复上报合流音源报错问题
- 修复合流加水印参数少校验边界问题

5.6.8

发布日期：2023/01/06

新增功能

1. 通过 `createMicrophoneAudioTrack` 与 `createMicrophoneAndCameraTracks` 接口获取音频时，增加 `noiseSuppression`、`echoCancellation`、`autoGainControl` 配置，以控制硬 3A 开启或关闭。
2. 升级主播 `upgradeToAnchorRoom` 返回房间内其他人和资源、cdn 开关状态：`userIds`、`tracks`、`CDNEnable`，房间内已有的人员、资源、CDN 状态不再从 `room` 监听处抛出

5.6.7

发布日期：2022/12/27

功能优化

- 优化 SDK 回调业务层代码时，业务层代码报错的日志信息，保留业务错误堆栈以便于业务层排查问题。

5.6.6

发布日期：2022/12/20

新增功能

- 支持采集与发布竖版视频资源，订阅方无需特别处理。

5.6.5

问题修复

1. 修复低版本浏览器下（使用 `plan-b` 协议）中发布屏幕共享视频异常问题

5.6.4

发布日期：2022/11/30

问题修复

1. 优化内部音视频质量数据缓存机制，并修复音量上报通知与实际听感延迟问题
2. 修复发布资源时，IM 连接中断引发异常，导致后续发布接口无响应问题

5.6.3

发布日期：2022/11/22

新增功能

- 直播场景增加音频合流的音源识别功能，`room.registerReportListener` 增加 `onReportLiveAudioStates` 监听，接收直播房间内音频合流的音源信息，包含音源的所属人、id 标识、音量值。

问题修复

- 优化网络异常情况下，资源中断的恢复处理

- 退出房间后，不再处理 signal 下发的通知
- 优化队列阻塞时，重复拉房间数据问题

5.6.2

发布日期：2022/11/10

问题修复

- 修复取消订阅后，还有停留在最后一帧的画面的问题

5.6.1

发布日期：2022/11/08

使用 RTCLib 5.6.1 及以上版本，需要将 IM 升级到 5.6.0 及以上的版本。

新增功能

- WebRTC 状态数据 (R1 ~ R4) 上报，由 WebSocket 更换为 http3
- 发布资源时增加信令重试并添加 `joinRTCRoom(roomId, [, signalRetryTime])` 连接信令服务器的超时时间
 - 描述：一般信令失败后直接返回给用户失败状态码，设置之后 `signalRetryTime`。在用户指定时间内如果信令交互失败，则会进行重试。

内部重构

- 增加拉取模式，信令推拉模式结合，增强房间信令健壮性。
- 离线踢出房间 `offlineKickTime` 增加限制，默认60秒，最小15秒

内部优化

- 北极星 http 获取导航信息增加缓存
- 媒体服务 Http 请求增加 gzip 压缩，以减少传输内容增加网络响应速度。

缺陷修复

- 获取标签页带有音视频的时候，会创建自定义视频流失败。
- 修复 Chrome 107 `Cannot read properties of undefined (reading 'timestamp')`

5.5.6

发布日期：2022/11/01

功能优化

- CDN 自动模式下，合并《资源扩散》和《CDN 扩散》这两条信令。

5.5.5

发布日期：2022/10/17

问题修复

- 修复 5.5.1 资源禁用时报错问题，导致资源禁用失败的问题。

5.5.2

发布日期：2022/10/11

内部重构

- IM 与 RTC 实现信令结构解耦

功能优化

- 优化北极星上报，轮询方式优化，对质量数据增加缓存 LRU 算法
- 日志优化
 - 新增 MediaService 日志，上报超时设置
 - 人员加入、退出日志级别从 debug 调整为 info 级别

5.5.1

发布日期：2022/09/20

使用 5.5.1 需要将 IM 升级到 5.5.X 以上的版本

内部重构

- IM 与 RTC 实现信令结构解耦

问题修复

- 优化北极星日志上报策略，减少定时器触发任务，增加主进程稳定性。
- 日志优化
 - 人员加入、退出日志级别从 debug 调整为 info 级别
 - 增加房间内消息被过滤时的日志

5.4.7

发布日期：2022/09/20

新增功能

- 增加服务稳定性，为 MediaServer 添加服务探针

问题修复

- 修复大小流一起发送时大流模糊的问题，修改了大小流 Addtrack 顺序

功能优化

- 分辨率与码率各端对齐

5.4.6

发布日期：2022/08/23

问题修复

- 观众不加房间，多次订阅页面报错，导致订阅失败
- 会议场景：多次订阅资源，视频黑屏
- 修复用户问题：频繁调用发布资源、取消资源发布，页面控制台报错。

5.4.5

发布日期：2022/08/17

问题修复

- 资源发布、取消发布 SSRC资源不变更，导致发布资源不变更，远端订阅失败
- 修复音频降噪处理调试模块内存溢出的问题，以及资源销毁后的内存回收。

5.4.4

发布日期：2022/08/09

问题修复

- 修复 track.play 可选参数报错的异常

5.4.3

发布日期：2022/08/02

问题修复

- 修复屏幕分享时mandatory.chromeMediaSourceId属性不存在引发的报错
- 拆分出 RCMediaStreamCapture 对媒体资源优化

功能优化

- 音频采集时添加默认参数
- 音视频码率设置 `RCLocalVideoTrack::setBitrate` `RCLocalAudioTrack::setBitrate`
- 录屏时增加获取音频流 `createScreenWithAudioTracks`
- 日志优化: Exchange 接口增加 reqId

5.4.2

发布日期：2022/07/12

问题修复

- 日志模块细化，增加日志 Tag 方便日志分类查询。

功能优化

- 内部重构，建立事件队列，防止事件错乱。

5.4.1

发布日期：2022/06/14

功能优化

- 退出房间时，SDK 内部先静音本地发布资源，避免在退出房间的过程中（尤其流程耗时较长时）对方一直能听到发布端的声音。
- 房间保活机制中 RTC ping 超时时间不再使用固定的 60 秒，改为使用服务下发的超时时间配置。
- 优化动态码率计算策略。在 SDK 无法准确获取视频分辨率时，指定一个较大的上行码率，以优化部分浏览器（如 iOS 中的 Safari）中出现的视频模糊、丢帧问题。
- 北极星上报音视频上下行数据由一秒一次改为两秒一次。

5.3.13

发布日期：2022/11/28

新增功能（仅限 5.3.13 版本）

1. 通过 `createMicrophoneAudioTrack` 与 `createMicrophoneAndCameraTracks` 接口获取音频时，增加 `noiseSuppression`、`echoCancellation`、`autoGainControl` 配置，以控制硬 3A 开启或关闭。

5.3.12

发布日期：2022/05/30

功能优化

- 优化动态码率计算策略，所有无法准确获取视频分辨率的视频流统一按 1920*1080 来计算上行码率，以优化部分浏览器（如 Safari）中出现的视频模糊、丢帧问题。

5.3.11

发布日期：2022/05/27

问题修复

- 修复偶现的信令丢失导致的房间状态通知问题
- 优化动态码率计算策略，解决无法准确获取视频分辨率信息的情况下的视频模糊、丢帧问题
- 修复偶现 onKickoff 重复通知问题

5.4.0

发布日期：2022/05/09

新增功能

- 发布资源支持使用多个 PeerConnection 实例

5.3.10

发布日期：2022/05/09

问题修复

- 修复 IM 连接中断情况下，当服务端房间超时销毁后，客户端恢复连接重新加入房间时偶发加入失败问题

5.3.8

发布日期：2022/04/18

问题修复

- 修复主播结束跨房间连麦并退出副房间后，副房间内合流中仍有该主播音视频流的偶现问题。

5.3.7

发布日期：2022/04/16

问题修复

- 优化日志打印，便于问题排查
- 过滤 MCU 合流配置中的重复配置项

新增功能

- 支持在 MCU 合流配置中指定需要合流的音频资源列表。参考 [RCMCUConfigBuilder](#)

5.3.6

发布日期：2022/04/07

问题修复

- 修复跨房间连麦场景中，加入、退出副房间，主房间时偶现 `setRemoteDescription failed` 错误，导致主房间向副房间推流异常
- 修复跨房间连麦场景中，退出副房间时错误抛出 `the getRoomPKHandler is disabled in PK room` 异常日志

5.3.4

发布日期：2022/04/06

问题修复

- 修复跨房间连麦中，`onResponseJoinOtherRoom` 回调数据中无 `extra` 字段问题

5.3.3

发布日期：2022/03/31

问题修复

- 修复音视频会议场景下客户端通过调 `leaveRoom` 离开房间后，服务端不会立即停止录制房间内画面的问题。

5.3.2

发布日期：2022/02/24

新增功能

- `joinLivingRoom` 加直播房间增加返回已连麦的副房间 `roomIds` 列表: `PKRoomIds`
- `joinRTCRoom`、`joinLivingRoom` 支持业务层设置身份标识
- `RCLivingRoom` 增加 `isMainRoom` 方法，判断当前房间是否为主房间
- `RCLivingPKHandler` 模块增加 `getAllPKInfo` 方法，获取所有连麦信息

问题修复

- 主房间获取人员列表、房间内人员变动过滤掉副房间人员

5.3.1

发布日期：2022/01/28

问题修复

- 发起连麦后，超过默认 30s 对方无应答之后，重新邀请时报“邀请连麦中”的问题
- 合流布局设置副房间资源校验未通过问题

5.3.0

发布日期：2022/01/07

新增功能

- 增加跨房间连麦功能

5.2.3

发布日期：2021/12/30

新增功能

- 观众房间增加 `getRemoteRTCTracks`、`getRemoteMCUTracks`、`getCDNInfo` api，可获取房间内 RTC 资源、MCU 资源、CDN 信息
- `joinLivingRoomAsAudience`、`downgradeToAudienceRoom` 返回值增加 `userIds`、`RTCTracks`、`MCUTracks`、`CDNUri`

问题修复

- 修复观众加房间在未注册房间事件监听时，拿不到房间内资源的问题

5.2.2

发布日期：2021/11/25

新增功能

- 获取视频时，增加设置 `faceMode` 属性
- 订阅资源时返回订阅失败列表

问题修复

- 修复 `peerConnection` 底层音量值极小时，`onStateReport` 和 `onAudioLevelChange` 中 `audioLevel` 为 0 的问题
- 修复 chrome 85 上 `onAudioLevelChange` 中 `audioLevel` 为 `undefined` 的问题

5.2.1

发布日期：2021/11/05

问题修复

- 修复：用户断网重连成功后，无法监听到断网期间加入的用户。
- 修复：用户订阅的视频流在断网重连后出现黑屏的问题。
- 修复：在极少数情况下，参会者断网重连后可能看不到对方视频。问题举例：A 参会者断网重连期间，A 已订阅的会议成员 B 退出重新加入房间，A 重连后再订阅 B，无法看到 B 的视频。

- 修复：观众不加入房间进行订阅时，重复订阅会失败

5.2.0

发布日期：2021/10/22

新增功能

- 增加指定音频播放输出设备
- 房间监听器增加主播和观众切换身份通知: onSwitchRole

5.1.10

发布日期：2021/09/24

新增功能

- 增加内置 CDN 功能

5.1.9

发布日期：2021/09/10

问题修复

- 修复偶现计算出的码率特别大的问题
- 修复同一个房间两个人同时取消发布，订阅的资源在房间中不存在的问题
- 增加本人退出房间，收到信令不再处理

5.1.8

发布日期：2021/09/02

问题修复

- 收到用户退出通知，需等取消订阅完成再更新内存数据
- 订阅不存在的资源时，不再继续执行，增加提醒
- 离开房间时，清理内存中远端资源的 audio 标签
- 销毁本端 track 时，清理内存中本端资源的 audio 标签

新增功能

- 增加 h5 页面 video 标签在页面内播放属性
- 播放音频时，增加设置音频音量
- 本地资源销毁时，取消发布此资源

5.1.5

发布日期：2021/08/06

问题修复

- 修复质量数据定时器可能存在的内存泄漏问题

新增功能

- 增加 im 重连成功后，peerConnection 意外关闭的通知事件

5.1.4

发布日期：2021/07/22

问题修复

- 修复无音量上报时，直接退出房间清除定时器报错的问题

5.1.3

发布日期：2021/07/15

新增功能

- 增加了观众端支持分流订阅功能

问题修复

- 修复了 adapter 和 plugin-rtc 中分辨率枚举值不一致时的转化问题

5.1.2

发布日期：2021/07/01

新增功能

- 增加了公有云 SDK 连接私有云服务校验，校验不通过返回错误码: 53025。状态码请参见状态码表。
- 增加了个人发布资源个数限制，个人发布总 track 数量限制（包含小流视频 track）10 个，超出限制发布接口返回错误码: 53026。状态码请参见状态码表。

5.1.1

发布日期：2021/06/11

问题修复

- 修复 peerConnection stats 数据计算时，部分浏览器中存在内存对象未定义的错误

5.1.0

发布日期：2021/06/10

新增功能

- 兼容 UnifiedPlan 协议，支持移动端 H5 及非 Chrome 浏览器
- 增加观众加房间功能，以便于观众接收主播资源变更通知

5.0.9

新增功能

- 支持 Electron 环境下通过 `createScreenVideoTrack` 获取屏幕共享流

5.0.8

发布日期：2021/06/03

问题修复：

- 修复了兼容因老版本 MediaServer 产生的 SDP 行末存在非法空格导致 SDP 协商失败的问题
- 修复了 `unpublish` 未经发布的资源导致产生非法消息结构，进而引起 iOS 端崩溃问题

5.0.7

发布日期：2021/05/21

新增功能：

- 支持 Electron 桌面端应用使用自定义协议加载页面
- 新增 `room.getRemoteTracks()` 接口以获取所有房间内的远端资源
- `joinRTCRoom` 接口新增 `userIds` 与 `tracks` 数据返回当前房间内的人员列表和资源列表
- `joinLivingRoom` 接口新增 `userIds` 与 `tracks` 数据返回当前房间内的人员列表和资源列表

问题修复：

- 修复北极星数据未上报问题

5.0.6

发布日期：2021/05/13

问题修复：

- 支持 Electron 环境下的 file 协议页面，以修复 Electron 环境下的异常

功能优化

- 优化 IM 连接中断期间的事件补偿，优化相关处理逻辑

5.0.5

发布日期：2021/05/07

新增功能：

- RCRTCClient 类新增 `createLocalFileTracks` 接口以降低用户实现自定义文件流的复杂度
- RCRTCClient 类新增 `getCurrentId` 接口以获取当前用户 Id
- 房间中新增 `getLocalTracks` 接口以获取所有本端已发布的资源列表
- 新增 `IRCTrackStat`、`IRCRTCStateReport`、`IRCCandidatePairStat` 类型导出，以供 TS 开发者引用

问题修复：

- 修复低延迟直播观众端订阅失败时返回的状态码为 `RCRTCCode.SUCCESS` 的问题

5.0.3

发布日期：2021/04/30

问题修复

- 修复资源发布失败后，`RCLocalTrack.isPublished()` 值错误
- 修复连续多次调用 `publish`、`unpublish`、`subscribe`、`unsubscribe` 时可能引起的状态错误

5.0.2

发布日期：2021/04/23

问题修复

- 修复 `subscribe`、`unsubscribe`、`publish`、`unpublish` 接口连续调用引起的 `RTCPeerConnection` 状态异常报错
- 优化使用打包工具进行编译构建时的包体大小

5.0.1

发布日期：2021/04/15

问题修复

- 修复房间用户退出时未清理房间内的响应缓存资源问题
- 修复在部分低版本打包构建工具中可能打包失败问题

5.0.0

发布日期：2021/04/09

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、yarn 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
- 支持同时发布、取消发布、订阅、取消订阅多道流，极大地缩减了业务层处理复杂订阅关系的响应时间
- 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

- 支持直播合流布局中增加背景图、背景色
- 发布小流资源时，不再需要业务层关注小流数据的采集跟状态维护
- 支持单独取消同一 tag 流中的音轨或视轨

状态码

更新时间:2024-08-30

状态码	说明
10000	成功
50000	IM 服务未连接
50001	参数错误
50002	加入房间错误，重复加入 RTC 房间内
50003	当前不在房间内
50004	MediaServer 未开启
50006	RTC Token 无效
53001	底层信令调用错误
53003	创建 Offer 失败
53004	网络请求失败
53005	MCU 地址不可为空
53007	直播订阅失败，当前存在已订阅资源
53008	房间已被销毁，需重新加入房间获取 Room 实例
53009	没有可用的音视频服务器地址，一般是没有开通音视频服务导致的
53010	获取用户媒体资源流失败

状态码	说明
53011	获取屏幕共享流失败
53012	权限问题导致获取媒体流被拒绝
53013	创建自定义流失败
53014	无效的 TAG 定义
53015	IM 连接无效，无法识别当前登录的用户身份
53016	创建文件流失败
53017	无效的 File 实例
53018	setRemoteDescription failed
53019	浏览器不支持此方法
53020	媒体流无法播放，可能是远端流未订阅或本地流已销毁
53021	视频流播放时需传参 HTMLVideoElement 作为显示组件
53022	媒体流播放失败
53023	观众加入直播房间信令错误
53024	直播房间切换身份错误
53025	公有云 SDK 包不允许使用私有云环境
53026	单个用户发布资源超过限制（MediaServer 限制最多 10 个 track）
53027	房间内无主播推 CDN
53028	加入 RTC 房间 joinType 为 1 时，当前有其他端在房间时的应答码

状态码	说明
53029	设置音频输出设备时，无权限使用请求的设备
53030	方法在 PK 房间上不可用
53031	资源没有全部发成功
53032	electron 中 mac 系统暂不支持屏幕共享采集声音
53033	获取媒体资源时，无系统权限
53034	发布时无有效资源，如 track 已被销毁