会议 / 直播
小程序 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

> 提示
>
> 融云 RTCLib SDK 在小程序平台仅支持音视频会议，不支持音视频直播。

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

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：

- **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。

实时音视频服务端

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

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

> 前往融云服务端开发文档 · 集成必读 »

控制台

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

> 音视频服务必须要从控制台开通后方可使用。参见开通音视频服务。

实时音视频数据

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

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

率。

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

小程序合法域名

更新时间:2024-08-30

融云音视频服务暂仅支持微信小程序。小程序平台默认不允许小程序访问第三方域名。因此，为使小程序可正常用融云服务，您需要将融云服务相关域名加入小程序的合法域名列表。

您需要前往 微信公众平台 配置以下合法域名。

request 合法域名

```
https://cdn.ronghub.com
https://cometproxy-cn.ronghub.com（旧）
https://comet.rong-edge.com（新）
https://mini-cn.ronghub.com（旧）
https://minicomet-b.rong-edge.com（新）
https://mini-cn.ronghub.com（旧）
https://minicomet-b.rong-edge.com（新）
https://rtc-miniapp-wctgw-prod-bdcbj.ronghub.com
https://nav.cn.ronghub.com（旧）
https://nav2-cn.ronghub.com（旧）
https://nav.rong-edge.com（新）
https://nav-b.rong-edge.com（新）
https://logcollection.ronghub.com（旧）
https://collection.rong-edge.com（新）
```

socket 合法域名

```
wss://wsproxy.cn.ronghub.com（旧）
wss://wsap-cn.ronghub.com（旧）
wss://ws.rong-edge.com（新）
wss://ws-b.rong-edge.com（新）
```

uploadFile 合法域名

```
https://upload.qiniup.com
https://gz.bcebos.com
```

快速集成

更新时间:2024-08-30

本教程是为了让新手快速了解融云音视频微信小程序 RTCLib SDK（@rongcloud/plugin-wechat-rtc ）。

> 提示
>
> **融云 RTCLib SDK 在小程序平台仅支持音视频会议，不支持音视频直播。**

微信音视频小程序开发须知

单个房间内最多 16 人同时进行音频通话（因手机硬件配置通话人数会有上线浮动）。

- **音视频小程序 SDK 设计特点**：融云音视频小程序 SDK 基于微信的小程序框架提供的原生媒体组件封装了两个自定义组件。开发者仅需引入自定义组件、配合发布和订阅接口，即可完成小程序的基础音视频功能。
  - `rc-livepusher`：融云自定义的音视频推流组件，封装了微信的实时音视频录制原生组件 live-pusher 。
  - `rc-liveplayer`：融云自定义的音视频拉流组件，封装了微信的实时音视频播放原生组件 live-player 。
- **微信原生组件限制**：微信小程序的 `live-pusher`、`live-player` 是由客户端创建的原生组件（native-component）。在开始开发前，请确保您了解微信原生组件的功能及限制（详见微信文档原生组件的使用限制 ）。建议您在使用到原生组件时尽量在真机上进行调试，确保小程序音视频效果稳定可靠。
- **微信类目审核**：请确保您的小程序已通过支持 `live-pusher`、`live-player` 两个媒体组件的类目审核，且在微信小程序控制台开通了 `live-pusher` 和 `live-player` 的权限。
- **设备权限要求**：使用小程序录制音视频前，必须授于摄像头和麦克风权限。详见微信小程序用户信息授权文档：https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/authorize.html

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。**

- 开通音视频服务 ，请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 。

- 开通小程序服务。

Demo 项目

融云提供了微信小程序 Demo 项目。

https://downloads.rongcloud.cn/Wechat_RTCLib_Demo_v5.1.0.zip

步骤 1：导入 SDK

小程序 RTC SDK 强依赖小程序 IM SDK，使用 RTC SDK 前必须引入 IM SDK。

您可以使用 NPM 安装 RTC SDK 与其依赖的 IMLib SDK。

1. 安装 5.X 版本 IMLib。

```
npm install @rongcloud/engine@latest @rongcloud/imlib-next@latest -S
```

2. 安装 RTCLib。

```
npm install @rongcloud/plugin-wechat-rtc
```

RTCLib（@rongcloud/plugin-wechat-rtc）安装过程中，会将内部的 rcComponents 移动至 node_modules 下。rcComponents 文件夹中存放了推、拉流自定义组件 rc-livepusher 和 rc-liveplayer。

rcComponents 文件夹目录结构如下：

```
|- rcComponents/
|- rc-liveplayer.js
|- rc-liveplayer.json
|- rc-liveplayer.wxml
|- rc-liveplayer.wxss
|- rc-livepusher.js
|- rc-livepusher.json
|- rc-livepusher.wxml
|- rc-livepusher.wxss
```

3. 依赖包安装完毕后，在小程序开发者工具的菜单中，寻找 "工具 -> 构建 npm" 选项，对依赖的 npm 包进行小程序所需的二次编译。

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

```
const RongIMLib = require('@rongcloud/imlib-next');
const RongRTCLib = require('@rongcloud/plugin-wechat-rtc');
```

步骤 2：引入推、拉流自定义组件

开发者仅需引入融云基于微信的小程序框架原生媒体组件封装的自定义推拉流组件（rc-livepusher 与 rc-livepusher）、配合发布和订阅接口，即可完成小程序的基础音视频功能。

如需在您的小程序的某页面中使用自定义推、拉流组件，您需要在该页面同名的配置文件中增加以下配置：

```
{
"usingComponents": {
"rc-livepusher": "../../rcComponents/rc-livepusher",
"rc-liveplayer": "../../rcComponents/rc-liveplayer"
}
}
```

引入组件示例：假设 pages 目录下包含 main 目录，按照微信的开发框架要求，main 目录下会包含如下文件。

```
|- pages/
|- main/
|- main.js
|- main.json // 页面配置文件，请在该页面中增加配置
|- main.wxml
|- main.wxss
```

如果要在 main 页面（main.wxml）中使用自定义推拉流组件，您需要该页面的配置文件（main.json）中增加上面的配置。

> 提示
>
> Uniapp 开发者可参考 Uniapp 项目配置 文档。

步骤 3：初始化

RTCLib 强依赖 IMLib。在使用 RTCLib 的能力之前，**必须先调用 IMLib 的初始化接口**。

App Key 是使用 IMLib 进行即时通讯功能开发的必要条件，也是应用的唯一性标识。您必须拥有正确的 App Key，才能进行初始化。您可以登录 控制台，查看您已创建的各个应用的 App Key。

只有在 App Key 相同的情况下，不同用户之间才能互通数据。

初始化 IMLib

调用 IMLib 的 init 方法，初始化 IM 客户端。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。

初始化时需传入您的融云应用的 App Key。

```
import * as RongIMLib from '@rongcloud/imlib-next'
// 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

调用 IMLib 的 `installPlugin` 方法初始化 RTC 客户端。

```
// 初始化 RCRTCClient，初始化过程推荐放在建立连接之前
const rtcClient = RongIMLib.installPlugin(RongRTCLib.installer, { /*初始化参数请参考下方参数说明*/ })
```

- **RTCLib 初始化参数说明**

```
{
/**
* 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，通过 import { LogLevel } from '@rongcloud/plugin-rtc' 获取枚举值
* @description
* * 0 - DEBUG
* * 1 - INFO
* * 2 - WARN (default)
* * 3 - ERROR
*/
logLevel?: LogLevel
/**
* 覆盖默认的日志输出函数，便于业务层保存或上传日志
*/
logStdout?: (logLevel: LogLevel, content: string) => void
/**
* 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
}
```

步骤 4: 获取用户 Token

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

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

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

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

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

步骤 5: 建立 IM 连接

> 提示

> 必须成功 IM 连接后, 才可执行其他操作。

融云音视频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。应用客户端建立 IM 连接时必须传入 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。

取得临时测试 token 后，调用 `connect` 方法进行连接融云。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。

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

步骤 6：加入房间

```
/**
* 加入普通音视频房间
* @param roomId 房间 Id
* @returns data.room 当前加入的房间实例
* @returns data.userIds 当前房间内的其他用户 id
* @returns data.streams 当前房间内的其他用户发布的资源
*/
const { code, data } = await rtcClient.joinRTCRoom('roomId')

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

const { room, userIds, streams: RCRemoteStreams } = data

// 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
* @param state 被踢出房间的原因
*/
onKickOff? (byServer: boolean, state?: RCKickReason): void
// 被踢出房间时，将不能继续发布资源、订阅资源，业务层可去掉远端资源的 UI 展示或重新加入房间
},
/**
* 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 stream RCRemoteStream 类实例
*/
onAudioMuteChange (stream: RCRemoteStream) {
},
/**
* 发布者禁用/启用视频
* @param stream RCRemoteStream 类实例对象
*/
onVideoMuteChange (stream: RCRemoteStream) {
},
/**
* 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.getRemoteStreamsByUserId('userId') 获取
* @param streams 新发布的资源列表，一组 RCRemoteStream 实例
*/
onStreamPublish (streams: RCRemoteStream[]) {
// 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(streams)
if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
* 房间用户取消发布资源
* @param streams 被取消发布的资源列表
* @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 展示
*/
onStreamUnpublish (streams: RCRemoteStream[]) {
},
/**
* 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[]) {
}
})
```

步骤 7：发布资源

页面如已引入自定义推流组件 `rc-livepusher`，可发布资源。

发布完资源之后，SDK 内部会把推流地址赋给微信的推流组件 live-pusher，房间内其他人可通过订阅看到发布的资源。

```
<rc-livepusher></rc-livepusher>
```

代码示例

```
// 发布不传任何参数时，默认会发一个 tag 为 RongCloud 的音视频资源
const { code } = await room.publishStream()
```

步骤 8：订阅资源

页面如已引入自定义拉流组件 rc-liveplayer，可订阅资源。

订阅完资源之后，SDK 内部会把拉流地址赋给微信的拉流组件 live-player，页面中就会展示订阅资源的画面。

```
<rc-liveplayer id="{{id}}"></rc-liveplayer>
```

属性 ID 为房间内其他人发布的资源 stream 的唯一标识，可通过 stream.getMsid() 获取。每个 stream 对应一个拉流组件，订阅多个 stream 时，需引入多个 rc-liveplayer 并传入 id，rc-liveplayer 的 id 属性为必需绑定项。

> 提示
>
> 注：订阅一组远端 stream 时，使用 wx:for 渲染 rc-liveplayer 列表，wx:key 需绑定唯一标识符，请使用 stream.getMsid() 的值作为 rc-liveplayer 的唯一标识。

```
// 一组远端资源
const remoteStreams = [{
stream,
msid: stream.getMsid()
}]
```

```
<view wx:for="{{remoteStreams}}" wx:key="msid">
<rc-liveplayer id="{{msid}}"></rc-liveplayer>
</view>
```

代码示例

```
// streams 为加入房间时，房间内其他人已发布的资源，或房间事件监听 `onStreamPublish` 收到的房间内其他人新发布的资源
const { code } = await room.subscribe(streams);
```

步骤 9：离开房间

离开 RTC 房间，退出后将不能再与其他成员进行音视频通话。

```
// room 为加房间返回的 data.room 实例
await rtcClient.leaveRoom(room);
```

基本操作

更新时间:2024-08-30

> 以下示例中的 rtcClient 实例是由 RTCLib 初始化时获取。详见 RTC 初始化。

加入房间

1. RTCLib 初始化后，可获取到 RTC 客户端实例。调用 joinRTCRoom 方法加入 RTC 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 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
* @param roomId 房间 Id
* @returns data.room 当前加入的房间实例
* @returns data.userIds 当前房间内的其他用户 id
* @returns data.streams 当前房间内的其他用户发布的资源
*/
const { code, data } = await rtcClient.joinRTCRoom('roomId')

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

const { room, userIds, streams: RCRemoteStreams } = data
```

2. 获取房间实例后，可以注册房间事件监听器。重复注册时，仅最后一次注册有效。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({
/**
* 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) {
// 被踢出房间时，将不能继续发布资源、订阅资源，业务层可去掉远端资源的 UI 展示或重新加入房间
},
```

```
/**
* 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 stream RCRemoteStream 类实例
*/
onAudioMuteChange (stream: RCRemoteStream) {
},
/**
* 发布者禁用/启用视频
* @param stream RCRemoteStream 类实例对象
*/
onVideoMuteChange (stream: RCRemoteStream) {
},
/**
* 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.getRemoteStreamsByUserId('userId') 获取
* @param streams 新发布的资源列表，一组 RCRemoteStream 实例
*/
onStreamPublish (streams: RCRemoteStream[]) {
// 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(streams)
if (code !== RCRTCCode.SUCCESS) {
console.log('资源订阅失败 ->', code)
}
},
/**
* 房间用户取消发布资源
* @param streams 被取消发布的资源列表
* @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 展示
*/
onStreamUnpublish (streams: RCRemoteStream[]) {
},
/**
* 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[]) {
}
})
```

3. 加入房间后，可能会返回房间内已经存在其他参会者发布的资源。可以通过 subscribe 接口拉取这些资源。

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

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

获取房间数据

加入房间成功后可获取房间实例。可通过房间实例获取房间数据。

获取房间 ID

获取房间的 Room ID。getRoomId

```
// 获取房间 Id
const roomId: string = room.getRoomId()
```

获取 Session ID

获取房间的 Session ID。重新加入房间时，该 ID 会重置。getSessionId

```
// 获取房间的 sessionId，重新加入房间时，该 Id 会重置
const sessionId: string = room.getSessionId()
```

获取远端用户列表

获取房间中已存在的远端用户列表，不包含本端 User ID。getRemoteUserIds

```
// 获取房间中已存在的远端用户列表，不包含本端 userId
const userIds: string[] = room.getRemoteUserIds()
```

获取远端资源

获取远端发布的所有资源列表，或者通过指定 User ID 获取该用户在房间内发布的资源。

getRemoteStreams

getRemoteStreamsByUserId

```
// 获取远端发布的所有资源列表 streams
const remoteStreams: RCRemoteStream[] = room.getRemoteStreams()

// 根据 userId 获取此用户在房间内发布的资源列表
const remoteStreams: RCRemoteStream[] = room.getRemoteStreamsByUserId('some-user-id')
```

房间是否被销毁

isDestroyed

```
const isDestroyed = room.isDestroyed()
```

离开房间

leaveRoom

```
// room 为加房间返回的 data.room 实例
await rtcClient.leaveRoom(room)
```

自定义房间属性 房间数据存储

更新时间:2024-08-30

- 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。
- 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

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

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

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: '' })
```

设备管理

更新时间:2024-08-30

> 以下示例代码中的 room 指加入房间成功后获取到的实例。

操作房间内设备

指定默认摄像头

在集成使用 <rc-livepusher> 组件时，可以通过定义组件的 devicePosition 属性来指定使用的摄像头，默认为前置摄像头，其有效值通过 RCDevicePosition 枚举表示，RCDevicePosition.FRONT 与 RCDevicePosition.BACK;

开启摄像头预览

startPreview

```
/**
* 开启视频预览，仅在摄像头被打开的情况下有效
*/
room.startPreview()
```

关闭摄像头预览

stopPreview

```
/**
* 关闭视频预览，仅在摄像头被打开的情况下有效
*/
room.stopPreview()
```

打开摄像头

openCamera

```
/**
* 开启摄像头
* 未发布视频时，该方法仅打开摄像头硬件设备
* 发布视频后，打开摄像头硬件设备后，修改视频资源状态为可用，订阅端可以看到发布的视频
*/
const { code } = await room.openCamera()
```

切换摄像头

switchCamera

```
/**
* 开启视频预览，仅在摄像头被打开的情况下有效
*/
room.switchCamera()
```

关闭摄像头

closeCamera

```
/**
* 关闭摄像头
* 未发布视频时，该方法仅关闭摄像头硬件设备
* 发布视频后，关闭摄像头硬件设备后，修改视频资源状态为不可用，订阅端看到发布的视频为黑屏
*/
const { code } = await room.closeCamera()
```

打开麦克风

openMicphone

```
/**
* 开启麦克风
* 未发布音频时，该方法仅打开麦克风硬件设备
* 发布音频后，打开麦克风硬件设备后，修改音频资源状态为可用，订阅端可以听到发布的声音
*/
const { code } = await room.openMicphone()
```

关闭麦克风

closeMicphone

```
/**
* 关闭麦克风
* 未发布音频时，该方法仅关闭麦克风硬件设备
* 发布音频后，关闭麦克风硬件设备后，修改音频资源状态为不可用，订阅端听不到发布的声音
*/
const { code } = await room.closeMicphone()
```

获取房间内设备状态

摄像头是否开启

isCameraOpen

```
/**
* 摄像头是否开启
*/
const isCameraOpen = room.isCameraOpen()
```

麦克风是否开启

isMicphoneOpen

```
/**
* 麦克风是否开启
*/
const isMicphoneOpen = room.isMicphoneOpen()
```

本地用户流 发布资源

更新时间:2024-08-30

小程序只能发布一个流

请确保页面已经引入 rc-livepusher 组件，发布完资源之后，SDK 内部会把推流地址赋给微信的推流组件 live-pusher ，房间内其他人可通过订阅看到发布的资源。

publishStream

```
/**
* 发布资源
* @param tag 资源标识，可不传，默认为 RongCloudRTC
* @param mediaType 资源类型，代表发布音频、视频或音视频，可不传，默认为 RCMediaType.AUDIO_VIDEO
* @returns 发布资源状态码
*/
const { code } = await room.publishStream(tag, mediaType)
```

取消发布

unpublishStream

```
/**
* 取消发布
* @returns 取消发布的状态码
*/
const { code } = await room.unpublishStream()
```

远端用户流

更新时间:2024-08-30

> 以下示例中的 remoteStream 实例为加入房间时，房间内其他人已发布的资源，或房间事件监听 onStreamPublish 收到的房间内其他人新发布的资源。

订阅资源

请确保页面已经引入 rc-liveplayer 组件，订阅完资源之后，SDK 内部会把拉流地址赋给微信的拉流组件 live-player，页面中就会展示订阅资源的画面。

subscribe

```
/**
* 订阅资源
* @param subParams 订阅参数列表
* * ISubParams.stream 从房间数据或房间发布资源事件监听中拿到的 RCRemoteStream 对象
* * ISubParams.subTiny 是否订阅小流，可不传，默认为订阅小流，boolean 类型，true 代表订阅小流
* @returns 订阅的状态码
*/
const { code } = await room.subscribe (subParams: ISubParams[] | RCRemoteStream[])
```

取消订阅

unsubscribe

```
/**
* 取消订阅
* @param streams 一组远端 stream 对象
* @returns 取消订阅的状态码
*/
const { code } = await room.unsubscribe (streams: RCRemoteStream[])
```

查询远端流是否被订阅

isSubscribed

```
/**
* stream 是否被订阅
* @returns boolean 为 true 时代表为订阅，false 反之
*/
remoteStream.isSubscribed()
```

查询远端流是否被发布者禁用

- 获取音频资源是否被发布者禁用 isOwnerMuteAudio

```
/**
* 获取音频资源是否被发布者禁用
* @returns boolean 为 true 时代表被禁用，false 反之
*/
remoteStream.isOwnerMuteAudio()
```

- 获取视频资源是否被发布者禁用 isOwnerDisableVideo

```
/**
* 获取视频资源是否被发布者禁用
* @returns boolean 为 true 时代表被禁用，false 反之
*/
remoteStream.isOwnerDisableVideo()
```

查询远端流是否包含音频/视频

- 查询 stream 中是否有音频资源 hasAudio

```
/**
* stream 中是否有音频资源
* @returns boolean 为 true 时代表包含音频，false 反之
*/
remoteStream.hasAudio()
```

- 查询 stream 中是否有音频资源 hasVideo

```
/**
* stream 中是否有视频资源
* @returns boolean 为 true 时代表包含视频，false 反之
*/
remoteStream.hasVideo()
```

获取远端资源信息

- 获取远端流标识名称（Tag）。Tag 可用于区分流的类型，如 CDN 流、屏幕共享流等。创建自定义流时需要指定自定义的 tag。

> 提示
>
> 小程序端暂不支持发布自定义流、发布屏幕共享流，但支持观看其他平台发布的自定义流、屏幕共享流。

getTag

```
/**
* 获取 stream 的资源名称
* @returns 返回资源名称
*/
remoteStream.getTag()
```

- 获取远端资源的 ID getMsid

流 ID 的生成规则是 "userID' + '_' + 'tag'。

> 提示
>
> 流 ID 并不能作为区分流的唯一标识。因为同一个用户发布的默认音频和视频流的 ID 是相同的。如果需要获取一个流唯一标识，可以使用 ID + mediaType 拼接。

```
/**
* 获取 stream 的 Id，由“资源发布者的用户名_资源名”组成
* @returns 返回 stream Id
*/
remoteStream.getMsid()
```

- 获取远端资源的媒体类型 getMediaType

```
/**
* 获取 stream 里面的资源类型
* @returns 返回媒体类型 RCMediaType
* * RCMediaType.AUDIO_ONLY 为 0，代表仅是音频
* * RCMediaType.VIDEO_ONLY 为 1，代表仅是视频
* * RCMediaType.AUDIO_VIDEO 为 2，代表是音视频
*/
remoteStream.getMediaType()
```

- 获取远端资源的发布者 Id getUserId

```
/**
* 获取 stream 的发布者
* @returns 资源发布者用户 Id
*/
remoteStream.getUserId()
```

流静音

更新时间:2024-08-30

> 以下示例中的 remoteStream 实例为加入房间时，房间内其他人已发布的资源，或房间事件监听 onStreamPublish 收到的房间内其他人新发布的资源。

静音远端音频

执行此方法之后，仅决定本人是否听远端发布者的声音，不影响远端音频的状态

mute

```
/**
* 本端静音
*/
remoteStream.mute()
```

取消静音远端音频

执行之后，可以继续听远端发布者的声音

unmute

```
/**
* 本端取消静音
*/
remoteStream.unmute()
```

切换音频播放设备

更新时间:2024-08-30

> 以下示例中的 remoteStream 实例为加入房间时，房间内其他人已发布的资源，或房间事件监听 onStreamPublish 收到的房间内其他人新发布的资源。

设置远端音频输出设备

> 该方法仅 iOS 设备稳定支持，部分 Android 设备无法有效进行设备切换。

setAudioOutputDevice

```
/**
* 设置音频输出设备
* @param mode 包含扬声器或听筒，默认为扬声器
* * RCAudioOutputDevice.SPEAKER 扬声器
* * RCAudioOutputDevice.EAR 听筒
*/
remoteStream.setAudioOutputDevice (mode: RCAudioOutputDevice)
```

推拉流自定义组件

更新时间:2024-08-30

由于微信开发者工具模拟器并不支持 rtmp 协议，所以推拉流组件在模拟器中无法正常显示，建议在真机中查看推拉流组件的效果。

引入推拉流组件配置属性需要的枚举值:

```
import {
ORIENTATION,
ASPECT,
LOCALMIRROR,
AUDIOREVERBTYPE,
BEAUTYSTYLE,
FILTER,
OBJECTFIT,
REFERRERPOLICY,
RCAudioOutputDevice,
RCAudioVolumeType
} from '@rongcloud/plugin-wechat-rtc'
```

提示

以下属性和自定义事件，除拉流组件的 id 属性，其他均为可选配置项，可按业务需求添加。

推流组件

配置属性

```
<rc-livepusher style="{{style}}"></rc-livepusher>
```

参数	类型	说明
style	String	组件样式，默认样式为 width:100%;height:100px，可以传入配置样式修改
minBitrate	Number	最小码率，默认为 200
maxBitrate	Number	最大码率，默认为 1000
audioVolumeType	RCAudioVolumeType	音量类型，包括：自动、媒体音量、通话音量，默认为 RCAudioVolumeType.AUTO，即自动
beauty	Number	美颜，取值范围为 0-9，默认为 0，代表关闭
whiteness	Number	美白，取值范围为 0-9，默认为 0，代表关闭

参数	类型	说明
autoFocus	Boolean	是否自动聚焦，默认为 true
orientation	ORIENTATION	画面方向，默认为 ORIENTATION.VERTICAL，即竖直
aspect	ASPECT	宽高比，默认为 ASPECT.NINEDIVIDESIXTEEN，即 9/16
waitingImage	String	进入后台时推流的等待画面
zoom	Boolean	调整焦距，默认为 false
remoteMirror	Boolean	设置推流画面是否镜像，产生的效果在 live-player 反应到，默认为 false
localMirror	LOCALMIRROR	控制本地预览画面是否镜像，默认为 LOCALMIRROR.AUTO，即前置摄像头镜像，后置摄像头不镜像
audioReverbType	AUDIOREVERBTYPE	音频混响类型，默认为 AUDIOREVERBTYPE.CLOSE，代表关闭
enableAgc	Boolean	是否开启音频自动增益，默认为 false
enableAns	Boolean	是否开启音频噪声抑制，默认为 false
videoWidth	Number	上推的视频流的分辨率宽度，默认为 360
videoHeight	Number	上推的视频流的分辨率高度，默认为 640
beautyStyle	BEAUTYSTYLE	美颜类型，默认为 BEAUTYSTYLE.SMOOTH，代表光滑
filter	FILTER	色彩滤镜，默认为 FILTER.STANDARD，即标准
devicePosition	RCDevicePosition	指定摄像头，默认为前置摄像头（>= v5.2.0）

增加自定义事件

推流自定义事件的抛出值可参考 微信小程序 live-pusher 中的说明 。

参数	类型	说明
bindstatechange	Function	状态变化事件
bindnetstatus	Function	网络状态通知
binderror	Function	渲染错误事件
bindbgmstart	Function	背景音开始播放时触发
bindbgmprogress	Function	背景音进度变化时触发
bindbgmcomplete	Function	背景音播放完成时触发
bindaudiovolumenotify	Function	麦克风采集的音量大小通知

拉流组件

每个 stream 对应一个拉流组件，订阅多个 stream 时，需引入多个拉流组件并传入 id。id 为房间内其他人发布的资源 stream 的唯一标识，可通过 stream.getMsid() 获取。**id 为必须配置项。**

配置属性

```
<rc-liveplayer id="{{id}}"></rc-liveplayer>
```

参数	类型	说明
id	String	播放组件 id
style	String	组件样式，默认样式为 width:100%;height:100px，可以传入配置样式修改
muted	Boolean	是否静音，默认为 false
objectFit	OBJECTFIT	填充模式，视频短边被填充为黑色或裁剪视频长边，默认为：视频短边被填充为黑色
orientation	ORIENTATION	画面方向，竖直或水平，默认为竖直
autoPauseIfNavigate	Boolean	跳转到本小程序的其他页面时，是否自动暂停本页面的音视频播放，默认为 true
autoPauseIfOpenNative	Boolean	跳转到其它微信原生页面时，是否自动暂停本页面的音视频播放，默认为 true
pictureInPictureMode	string/Array	设置小窗模式: push, pop，空字符串或通过数组形式设置多种模式（如: ["push", "pop"]）;[]: 取消小窗、push: 路由 push 时触发小窗、pop : 路由 pop 时触发小窗
referrerPolicy	REFERRERPOLICY	发送访问来源策略，默认为不发送

增加自定义事件

拉流自定义事件的抛出值可参考 微信小程序 live-player 中的说明 。

参数	类型	说明
bindstatechange	Function	播放状态变化事件
bindfullscreenchange	Function	全屏变化事件
bindnetstatus	Function	网络状态通知
bindaudiovolumenotify	Function	播放音量大小通知
bindenterpictureinpicture	Function	播放器进入小窗
bindleavepictureinpicture	Function	播放器退出小窗

Uniapp 项目配置

更新时间:2024-08-30

> 本篇用于说明在 Uniapp 开发框架下如何使用配置自定义组件 rc-liveplayer 和 rc-livepusher，非 Uniapp 用户可略过。

安装依赖

```
# 安装 IMLib
npm i @rongcloud/engine@latest @rongcloud/imlib-next@latest -S

# 安装 RTCLib
npm i @rongcloud/plugin-wechat-rtc@latest -S
```

安装 RTCLib 会在项目根目录下生成 rcComponents 目录，该目录下包含了 rc-liveplayer 和 rc-livepusher 组件。

项目配置

1. 修改 rcComponents 目录名，将目录重命名为 wxcomponents。
2. 修改 Uniapp 项目根目录下的 pages.json 配置文件，在需要引入组件的 page 页面中增加配置。

```
{
"pages": [
{
"path": "pages/index/index",
"style": {
"navigationBarTitleText": "uni-app",
"mp-weixin": {
"usingComponents": {
"rc-livepusher": "/wxcomponents/rc-livepusher",
"rc-liveplayer": "/wxcomponents/rc-liveplayer",
}
}
}
}
],
}
```

通话数据统计

更新时间:2024-08-30

> 以下示例代码中的 room 指加入房间成功后获取到的实例。

registerReportListener

```
import { ILivePusherNetStatus, ILivePlayerNetStatus, IResourceNetStatus } from '@rongcloud/plugin-wechat-rtc'
room.registerReportListener((report: IResourceNetStatus) => {
/**
* 小程序发布流的网络数据
*/
const senders: ILivePusherNetStatus = report.senders
/**
* 小程序所有订阅流的网络数据
* msid 为订阅资源的唯一标识
*/
const receivers: { [msid: string]: ILivePlayerNetStatus } = report.receivers
})
```

```
/**
* 小程序发布流的网络数据接口定义
*/
export interface ILivePusherNetStatus {
/**
* 当前视频编/码器输出的比特率，单位 kbps
*/
videoBitrate: number
/**
* 当前音频编/码器输出的比特率，单位 kbps
*/
audioBitrate: number
/**
* 当前视频帧率
*/
videoFPS: number
/**
* 当前视频 GOP,也就是每两个关键帧(I帧)间隔时长，单位 s
*/
videoGOP: number
/**
* 当前的发送/接收速度
*/
netSpeed: number
/**
* 网络抖动情况，抖动越大，网络越不稳定
*/
netJitter: number
/**
* 网络质量
* 0：未定义 1：最好 2：好 3：一般 4：差 5：很差 6：不可用
*/
netQualityLevel: number
/**
* 视频画面的宽度
*/
videoWidth: number
/**
* 视频画面的高度
*/
videoHeight: number
/**
* 主播端堆积的视频帧数
*/
videoCache: number
/**
* 主播端堆积的音频帧数
*/
audioCache: number
}
```

```
/**
* 小程序订阅流的网络数据接口定义
*/
export interface ILivePlayerNetStatus extends ILivePusherNetStatus {
/**
* 解码器中缓存的视频帧数（Android 端硬解码时存在）
*/
vDecCacheSize: number
/**
* 缓冲的总视频帧数，该数值越大，播放延迟越高
*/
vSumCacheSize: number
/**
* 音画同步错位时间（播放），单位 ms，此数值越小，音画同步越好
*/
avPlayInterval: number
/**
* 音画同步错位时间（网络），单位 ms，此数值越小，音画同步越好
*/
avRecvInterval: number
/**
* 音频缓冲时长阈值，缓冲超过该阈值后，播放器会开始调控延时
*/
audioCacheThreshold: number
}
```

音量上报

更新时间:2024-08-30

> 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。

发布资源的音量上报

onLocalAudioLevelChange

```
/**
* 注册本端麦克风采集的音量通知
* 回调函数接收一个数字类型的音量值
*/
room.onLocalAudioLevelChange((volume: number) => {
// 打印音量
console.log(volume)
})
```

订阅资源的音量上报

onRemoteAudioLevelChange

```
/**
* 注册远端订阅资源的音量通知
* 回调函数接收两个参数
* * msid: 代表订阅资源的唯一标识
* * volume: 订阅资源对应的音量值
*/
room.onRemoteAudioLevelChange((msid: string, volume: number) => {
// 打印资源 id 和音量
console.log(id, volume)
})
```

信令管理

更新时间:2024-08-30

> 以下示例代码中的 room 指加入房间成功后获取到的实例。

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

sendMessage

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

参数说明

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

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。

更新日志 v5.2.0

更新时间:2024-08-30

发布日期：2024/03/07

- 重构远端资源管理模块，修复远端资源状态变更时，`RCRemoteStream` 实例的 `isOwnerMuteAudio` 与 `isOwnerDisableVideo` 取值错误。
- 废弃 `RCRemoteStream` 类的 `registerStreamEventListener` 方法，流状态变更时统一通过房间 `onAudioMuteChange` 与 `onVideoMuteChange` 回调通知。
- 支持指定前后置摄像头进行本地流采集，支持摄像头切换功能。

v5.1.1

发布日期：2024/01/23

- 修复加入房间后，初始化 RemoteStream 资源时状态取值错误
- 修复内部消费队列错误导致的资源状态通知处理错误

v5.1.0

发布日期：2023/05/23

- 修复对新版本 IM SDK 的兼容性问题
- 优化内部日志模块
- 优化内部代码结构，以提升 SDK 稳定性

v5.0.1

发布日期：2022/09/22

- 修改 IMLib 最低依赖版本为 5.4.5

v5.0.0

发布日期：2022/07/11

- 首次发布融云音视频微信小程序 RTCLib SDK（@rongcloud/plugin-wechat-rtc ）。