

【网络电话】Juphoon SDK for WeChat 开发集成指南



开发指导手册（WeChat）

宁波菊风系统软件有限公司

2025年1月

版权所有©宁波菊风系统软件有限公司 2025。保留一切权利。

版本	作者	日期	说明
v2501.0	章克飞	2025.01	• 初版

一、系统概述

非常感谢您使用菊风系统软件的产品，我们将为您提供最好的服务。本手册可能包含技术上不准确的地方或排版错误。本手册的内容将做定期的更新，恕不另行通知；更新的内容将会在本手册的新版本中加入。我们随时会改进或更新本手册中描述的产品或程序。

1.1 系统介绍

菊风视频能力平台在实际的项目中定位为音视频能力的提供方，除此之外还包装了一些和音视频通讯强相关的业务。以银行项目为例可分为视频客服业务、视频会议业务、视频双录业务、AI双录业务、一对一通话及消息业务等。上述业务需要客户渠道类系统或者客户业务类系统集成我们 Juphoon RTC SDK 或者插件才能形成完整的业务，在整个完整的业务中我们提供基础的音视频通讯能力和一些对应业务上所需的特色能力。如视频客服业务的智能排队服务，视频会议业务的增强会控服务等。

菊风视频能力平台提供标准 Juphoon RTC SDK 用于给客户渠道类系统和客户业务类系统集成并通过 Juphoon RTC SDK 接入到视频能力平台进行音视频通讯。

菊风为开发者提供 JRTC SDK 功能开发包，涵盖了音视频引擎终端、服务器和业务模块，支持实现智能排队、全景录像、多人音视频等业务功能。

Juphoon RTC SDK支持 iOS、Android、鸿蒙Next、Windows、UOS、Kylin、微信小程序、H5 等操作系统平台。对于银行的其他公共平台或其他第三方平台，视频能力平台可提供标准第三方接口和其他平台进行对接。实现和银行环境的整体融入。

1.2 系统特性

菊风视频能力平台（Juphoon Video Capability Platform）提供高可用、高品质、超低延时的实时音视频通信服务，为远程银行、视频双录、视频会议、AI 双录、VoLTE 视频通话等泛金融场景化方案提供平台支撑。具有业界领先的实时音视频编码技术，以及抗啸叫降噪、ARS 码率自适应、SPo 视频甜点、智能路由等技术，应对网络质量非均衡性、网络异构性、多类型终端的接入的挑战，保证高音质、高画质。整个平台由宁波菊风系统软件有限公司独立研发，具有自主知识产权。

二、关于菊风软件

宁波菊风系统软件有限公司（简称“菊风”，英文简称“Juphoon”）成立于2005年，现有员工200余人，注册资金2050万元，总部位于宁波，在北京、广州、长沙设有区域中心（研发、销售和交付），在郑州和杭州设有交付中心，是一家提供实时音视频通信和RCS融合通信解决方案的供应商。宁波总部研发中心主要负责客户端SDK 和 APP、音视频引擎、服务器等产品的研发；云平台和服务器的运维、网管等支撑系统研发，现中心成员有180名。

菊风经过15年+音视频底层技术积累，为众多行业合作伙伴提供了超优音视频通信服务。凭借卓越的产品以及优质的服务，迄今为止，已有数十亿终端用户以及众多企业用户通过菊风云实现了音视频场景化沟通，涉及社交、教育、医疗、智能硬件、金融、电商等多个行业领域，为其提供了有针对性的行业化解决方案。

菊风为开发者提供的优而小的 SDK 极简接入，快速助其实现实时音视频通信能力。基于客户不同需求，菊风云提供灵活的部署模式——公有云，专有云，私有云，海外云以及混合云。对主流系统平台全覆

盖，支持 iOS、Android、鸿蒙Next、Windows、UOS、Kylin、微信小程序、H5 等。支持各移动设备（电脑、手机、平板）、VTM 机等多终端设备的适配。

2.1 技术支持

在您使用 Juphoon RTC SDK 的过程中，遇到任何困难，请与我们联系，我们将热忱为您提供帮助。

您可以通过如下方式与我们联系：

公司官网：<https://rtc.juphoon.com>

产品咨询：sales@juphoon.com

加急热线：13056832331

咨询电话：400-800-8708 / 0574-87901227

售前工程师微信二维码：



2.2 版权声明

“Juphoon RTC for WeChat SDK”是由宁波菊风系统软件有限公司开发，拥有自主知识产权（软著正式编号 2020SR0369466号）的系统平台，宁波菊风系统软件有限公司拥有与本产品所用技术相关的知识产权。这些知识产权包括但不限于一项或多项发明专利或者正在进行申请的专利（ZL202010288867.7、ZL201911393580.4）。

本产品发行所依照的许可协议限制其使用、复制分发和反编译。未经宁波菊风系统软件有限公司事先书面授权，不得以任何形式或借助任何手段复制本产品的任何部分。随本 SDK 一同发布的 Demo 演示程序源代码版权归宁波菊风系统软件有限公司。Juphoon 是宁波菊风系统软件有限公司的商标。

三、快速集成SDK

本文为您介绍 Wechat 端集成 SDK 的操作步骤，帮助您快速集成 SDK 并实现视频客服（访客端）的基本功能。

3.1 集成常见问题

见 [FAQ](#)

3.2 操作步骤

步骤一：获取 Juphoon_Rtc_SDK_for_Wechat

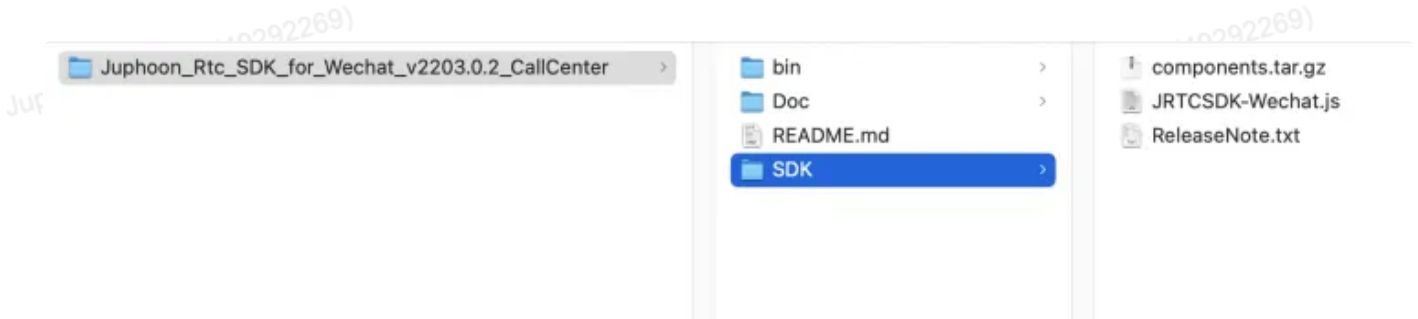
您可在 Juphoon 的产品官方网站下载到最新版的 Juphoon RTC SDK，访问[下载地址](#)，示例如下：



注：首次访问，请先注册后登录。

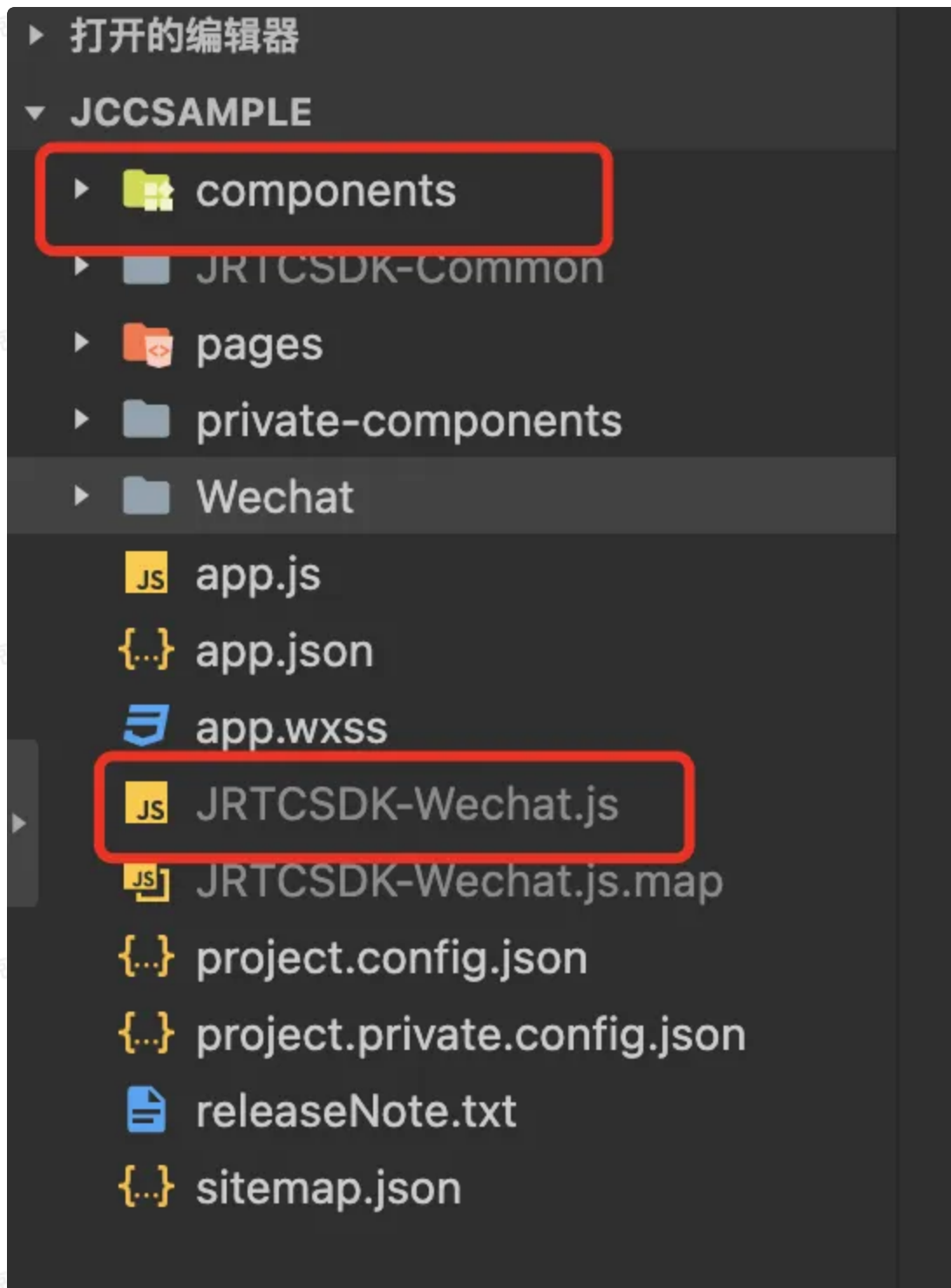
Juphoon_Rtc_SDK_for_Wechat_版本号_CallCenter包里面提供了所有支持开发语言 demo 程

序的编译程序、开发指南、demo 程序源码和 SDK 文件，其解压之后的目录结构如下所示：



步骤二：导入 SDK

1、拷贝 SDK 文件夹内的 JRTCSDK-Wechat.js 到您的工程目录中及将解压 components 出来的文件夹拷贝到程序上。如下图所求：



步骤三：导入工程需要使用到的相关模块

```
1  const {  
2    JRTCCClient,  
3    JRTCCClientInitParam,  
4    JRTCCClientLoginParam,  
5    ClientState,  
6    JRTCMediaDevice,  
7    RenderType,  
8    JRTCTracking,  
9    JRTCLog,  
10   JRTCLogLevel,  
11   JRTCCall,  
12   JRTCCallInitParam,  
13   JRTCCallJoinParam,  
14   JRTCCallExtraParam,  
15   PartRole,  
16   JRTCVersion,  
17 } = JRTCSDK;
```

步骤四：引用视频组件

在需展示视频画面的 json 文件中添加视频组件

```
<  →  pages > room > {...} index.json > ...  
{  
  "usingComponents": {  
    "local-stream": "../..components/local-stream",  
    "remote-stream": "../..components/remote-stream"  
  }  
}
```

步骤五：编译运行

以上步骤进行完后，编译工程，如果没有报错，恭喜您，您已经成功配置 SDK，可以进行下一步了。

四、实现视频通话（mpcall）

本文档为您展示通过 SDK 实现视频通话（mpcall）的相关步骤。

4.1 前提条件

请确认您已完成以下操作：

- 已获取AppKey

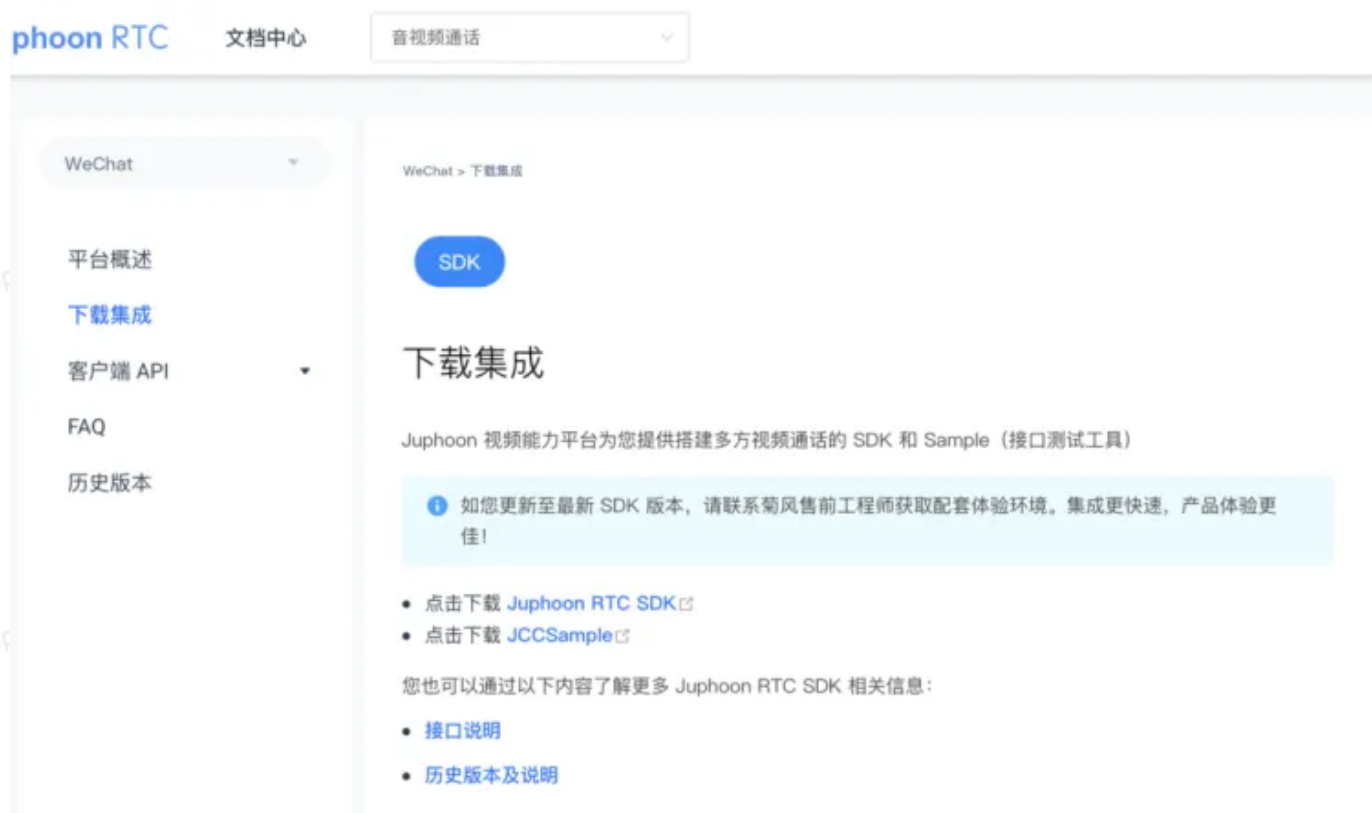
AppKey 作为同个环境的分域依据，同一个域的终端才能实现互通，AppKey 由 Juphoon 视频平台提供。

- 集成 SDK

4.2 快速跑通 Sample

1、在 Juphoon RTC SDK 文档中心，选择音视频通话选择项，在 SDK 下载页面下载体验 Sample 示例项目。

访问[下载地址](#)，示例如下：



2、下载完成后，解压出 JCCSample 。

3、使用微信开发者工具导入项目，如下图所示：

Q 输入项目名称

小程序项目

小程序

小游戏

代码片段

公众号网页项目

其他

导入项目

项目名称

JCCSample

目录

/Users/juphoon_lzz/Downloads/2203.0.2-2022082718014

该目录为非空目录，将保留原有文件创建项目

AppID

wx87578b325602ddb

[注册](#) 或使用 [测试号](#)

后端服务

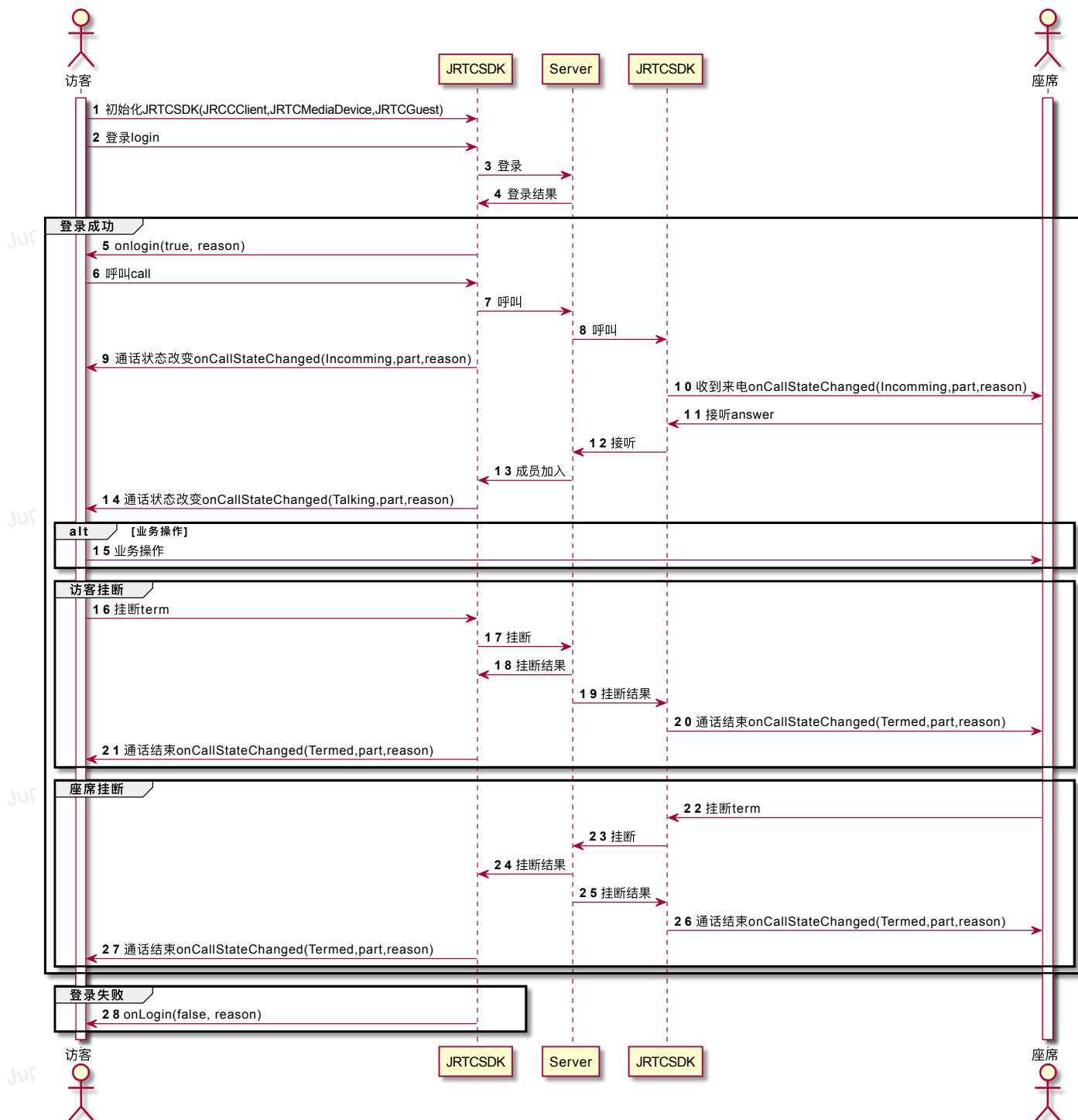
微信云开发

当前小程序 AppID 已开通云开发。[详情](#)

4、编译项目运行，如下图所求：



4.3 实现视频通话



4.3.1 初始化

AppKey 作为同个环境的分域依据，同一个域的终端才能实现互通，AppKey 由 Juphoon 视频平台提供。

在使用业务接口前，需对 Juphoon SDK 进行初始化操作。

类	模块	描述
---	----	----

JRTCClient	登录模块	负责视频平台的登录登出，只有登录到视频平台才可以使用视频相关的业务，AppKey 作为同个环境的分域依据，同一个域的终端才能实现互通。
JRTCMediaDevice	媒体模块	负责本地的媒体设备操作，视频画面渲染等功能
JRTCCall	mpcall 通话模块	可以通过流水号加入通话，邀请其他成员加入通话

示例代码：

```
1  const clientInitParam = new JRTCCClientInitParam();
2  clientInitParam.appName = appName;
3  clientInitParam.server = signalServer;
4  clientInitParam.appKey = appKey;
5  this.context.clientState = ClientState.IDLE;
6  this.initClientCallback();
7  this.context.client = JRTCCClient.create(this.context.clientCallback, clientInitParam);
8
9  this.initMediaDeviceCallback();
10 this.context.mediaDevice = JRTCMediaDevice.create(this.context.mediaDeviceCallback, {});
11
12 this.initCallCallback();
13 const initParam = new JRTCCallInitParam();
14 initParam.server = callServer;
15 this.context.call = JRTCCall.create(this.context.client, this.context.mediaDevice, initParam, this.context.callCallback);
16
17 /**
18  * 增加 JRTCCClientCallback 监听回调
19  */
20 initClientCallback() {
21   this.context.clientCallback = {
22     onLogin: (result, reason) => {},
23     onLogout: (reason) => {},
24     onClientStateChanged: (state, oldState) => {},
25     onSDKEvent: (event) => {},
26     onOnlineMessageReceived: (message, userId) => {},
27     onOnlineMessageSendResult: (result, operatorId) => {},
28   };
29 },
30
31 /**
32  * 增加 JRTCMediaDeviceCallback 监听回调
33  */
34 initMediaDeviceCallback() {
35   this.context.mediaDeviceCallback = {
36     onVolumeChanged: (volume, userId) => {},
37     onCameraUpdate: () => {},
38     onAudioInputUpdate: () => {},
39     onNetStateChanged: (self, netPoor) => {},
40   };
41 },
42
```



```

43  /**
44   * 增加 JRTCCallCallback 监听回调
45   */
46  initCallCallback() {
47      this.context.callCallback = {
48          onSessionCreate: (result, reason) => {},
49          onSessionDestroy: (reason) => {},
50          onMessageSendResult: (result, reason, operationId) => {},
51          onMessageReceived: (content, contentType, messageType, fromUserId) =>
52      {},
53          onJoin: (result, reason) => {},
54          onLeave: (reason) => {},
55          onCallPropertyChanged: (param) => {},
56          onInviteResult: (result, reason) => {},
57          onCancelInviteResult: (result, reason) => {},
58          onInviteAccepted: (param) => {},
59          onInviteRejected: (param) => {},
60          onInviteReceived: (param) => {},
61          onInviteCanceled: (param) => {},
62          onAcceptInviteResult: (result, reason) => {},
63          onRejectInviteResult: (result, reason) => {},
64          onParticipantJoin: (participant) => {},
65          onParticipantLeft: (participant, reason) => {},
66          onParticipantUpdate: (participant, changeParam) => {},
67      }
    },

```

4.3.2 登录

```

1  /**
2   * 登录 Juphoon RTC 平台，只有登录成功后才能进行平台上的各种业务
3   *
4   * 登录结果通过 {@link JRTCClientCallback.onLogin onLogin} 回调通知
5   *
6   * @param userId      用户ID
7   * @param password    密码，不能为空
8   * @param clientLoginParam 登录参数，一般不需要设置，不设置则按默认值
9   * @returns 接口调用结果
10  * - true: 接口调用成功
11  * - false: 接口调用异常
12  * @note 目前只支持免鉴权模式，服务器不校验账号密码，免鉴权模式下当账号不存在时会自动去
    创建该账号
13  * @note 用户名为英文数字和 '+' '-' '_' '.', 长度不要超过64字符，'- ' '_' '.' 不能作
    为第一个字符
14  */
15  public login(userId: string, password: string, clientLoginParam?: JRTCClientLoginParam): boolean;

```

[JRTCClientLoginParam](#) 参数介绍：

参数名	参数含义
token	token
tokenType	token校验类型
tokenExtraParam	token校验拓展参数

登录的结果将会通过 [JRTCClientCallback](#) 中的 [onLogin](#) 接口上报

```

1  /**
2   * 登录结果回调
3   *
4   * @param result true 表示登录成功, false 表示登录失败
5   * @param reason 登录失败原因, 当 result 为 false 时该值有效
6   */
7  onLogin(result: boolean, reason: ReasonCode): void;

```

示例代码：

```

1 let loginParam = new JRTCClientLoginParam();
2 loginParam.token = this.data.token;
3 loginParam.tokenType = this.data.tokenType;
4 loginParam.tokenExtraParam = this.data.tokenExtraParam;
5 this.context.client.setDisplayName(this.data.displayName);
6 this.context.client.login(param.userId, '123456', loginParam);
7
8 onLogin: (result, reason) => {
9     console.log('onLogin result: ', result, ' reason: ', reason);
10 },

```

4.3.3 等待会话创建

登录成功以后，SDK内部会自动创建会话，会话创建成功以后，才能开启 mpcall 通话。

```

1 /** session创建结果回调 */
2 onSessionCreate(result: boolean, reason: string): void;
3
4 /** session销毁回调 */
5 onSessionDestroy(reason: string): void;

```

示例代码：

```

1 onSessionCreate: (result, reason) => {
2     console.log('onSessionCreated result reason: ', result, reason);
3 },
4
5 onSessionDestroy: (reason) => {
6     console.log('onSessionDestroy reason: ', reason)
7 },

```

4.3.4. 加入通话

加入对应业务流水号的通话。

```

1  /**
2   * 加入通话
3   *
4   * @note 需要已经登录
5   * @note 该接口支持加入通话并且邀请其他用户加入，通过 {@link JRTCCallJoinParam.inviteCalleeUserId inviteCalleeUserId}
6   * 和 {@link JRTCCallJoinParam.inviteCalleeUserType inviteCalleeUserType}
   设置需要邀请的用户ID和用户类型
7   *
8   * 该方法让用户加入通话，在同一个通话内的用户可以互相视频语音。<br>
9   * 如果用户已在通话中，必须退出当前通话，即处于空闲状态，才能进入其他通话，否则将直接返回 false，且不会收到回调通知。
10  *
11  * @param joinParam 加入通话参数
12  * @param serialId 业务流水号，保证唯一，不填则自动填充
13  */
14  public join(joinParam: JRTCCallJoinParam, serialId: string | undefined): boolean;

```

JRTCCallJoinParam 参数介绍：

参数名	参数含义
inviteCalleeUserId	被邀请人id。如果不为空，会在自身加入通话同时邀请
inviteCalleeUserType	被邀请人类型。默认APP用户。当被邀请者用户ID不为空时，该值有效
callerNumberForSip	呼叫主叫号
extraInfo	随路参数。当被邀请用户ID不为空时，该值有效
extraSerialId	自定义流水号
multiStream	是否合流
video	是否视频
routId	线路id

加入通话后，会收到加入通话结果的回调：

```
1  /**
2   * 加入通话结果回调
3   *
4   * 调用 {@link JRTCCall.join join} 接口成功后，会收到此回调。
5   *
6   * @param result 加入通话是否成功
7   * - true: 成功
8   * - false: 失败
9   * @param reason 加入失败原因，当 result 为 false 时该值有效。失败原因参见：{@link ReasonCode}
10  */
11  onJoin(result: boolean, reason: ReasonCode): void;
```

示例代码：

```
1  const joinParam = new JRTCCallJoinParam();
2  joinParam.inviteCalleeUserId = inviteCalleeUserId;
3  joinParam.inviteCalleeUserType = inviteCalleeUserType;
4  joinParam.video = video;
5  joinParam.extraInfo = extraInfo;
6  joinParam.callerNumberForSip = callerNumberForSip;
7  joinParam.multiStream = multiStream;
8  joinParam.routeId = routeId;
9  joinParam.token = token;
10 joinParam.extraSerialId = extraSerialId;
11 this.context.call.join(joinParam, serialId);
12
13 onJoin: (result, reason) => {
14   console.log('onJoin result reason:', result, reason);
15 };
```

4.3.5 邀请

```

1  /**
2   * 邀请其他成员加入通话
3   * @note 需要已经加入通话
4   *
5   * @param calleeUserId 被邀请者用户ID
6   * @param inviteParam 邀请参数
7   * @return
8   * - 操作id: 接口调用成功, 对应 {@link JRTCCallCallback.onInviteResult onInviteResult} 回调的 operatorId 参数
9   * - -1: 接口调用异常, 不会收到回调
10  */
11  public invite(calleeUserId: string, inviteParam: JRTCCallExtraParam);

```

JRTCCallExtraParam 参数介绍:

参数名	参数含义
calleeUserId	被邀请者用户ID
calleeUserType	被邀请者用户类型
video	是否视频通话
extraInfo	随路参数
routeld	sip外呼线路ID
callerNumberForSip	sip外呼主叫号码

邀请后会通过 JRTCCallCallback 里的 onInviteResult 接口进行上报。

```

1  /**
2   * 邀请结果回调
3   *
4   * @param result 加入通话是否成功
5   *             - true: 成功
6   *             - false: 失败
7   * @param reason 邀请失败原因
8   */
9  onInviteResult(result: boolean, reason: string): void;

```

示例代码:

```

1  const param = new JRTCCallExtraParam();
2  param.calleeUserId = calleeUserId;
3  param.calleeUserType = calleeUserType;
4  param.video = video;
5  param.extraInfo = extraInfo;
6  param.routeId = routeId;
7  param.callerNumberForSip = callerNumberForSip;
8  this.context.call.invite(calleeUserId, param);
9
10 onInviteResult: (result, reason) => {
11     console.log('onInviteResult result reason', result, reason);
12 };

```

4.3.5.1 邀请被接受

对方接受邀请后会通过 `JRTCCallCallback` 里的 `onInviteAccepted` 接口进行上报。

```

1  /**
2   * 对方接受邀请通知
3   *
4   * @param param 回调参数
5   */
6  onInviteAccepted(param: JRTCCallExtraParam): void;

```

示例代码：

```

1  onAcceptInviteResult: (result, reason) => {
2      console.log('onAcceptInviteResult result reason', result, reason);
3  },

```

4.3.5.2 邀请被拒绝

对方拒绝邀请后会通过 `JRTCCallCallback` 里的 `onInviteRejected` 接口进行上报。

```

1  /**
2   * 对方拒绝邀请通知
3   *
4   * @param param 其他参数
5   * @see JRTCCallExtraParam
6   */
7  onInviteRejected(param: JRTCCallExtraParam): void;

```

示例代码：

```

1  onInviteRejected: (param) => {
2    console.log('onInviteRejected param', param);
3  },

```

4.3.6 取消邀请

在被邀请人未回应邀请前，可以取消当前的邀请。

```

1  /**
2   * 取消邀请
3   *
4   * @return
5   * - 操作id: 接口调用成功，对应 {@link JRTCCallCallback.onCancelInviteResult} 回调的 operatorId 参数
6   * - -1: 接口调用异常，不会收到回调
7   */
8  public cancelInvite(): boolean;

```

取消邀请结果会通过 [JRTCCallCallback](#) 里的 [onCancelInviteResult](#) 接口进行上报。


```

1  /**
2   * 取消邀请结果回调
3   *
4   * @param result 加入通话是否成功
5   *             - true: 成功
6   *             - false: 失败
7   * @param reason 取消邀请失败原因
8   */
9  onCancelInviteResult(result: boolean, reason: string): void;

```

示例代码：

```

1  this.context.call.cancelInvite();
2
3  onCancelInviteResult: (result, reason) => {
4      console.log('onCancelInviteResult result reason', result, reason);
5  };

```

4.3.7 收到邀请

收到邀请时会通过 [JRTCCallCallback](#) 里的 [onInviteReceived](#) 接口进行上报。

```

1  /**
2   * 收到邀请通知
3   *
4   * @param param 回调参数
5   */
6  onInviteReceived(param: JRTCCallExtraParam): void;

```

4.3.7.1 接受邀请

```

1  /**
2   * 接受邀请
3   * @param isVideo 是否视频模式
4   * @returns
5   */
6  public acceptInvite(isVideo: boolean): boolean;

```

接受邀请结果会通过 [JRTCCallCallback](#) 里的 [onAcceptInviteResult](#) 接口进行上报。

```

1  /**
2   * 接受邀请结果回调
3   *
4   * @param result 加入通话是否成功
5   *               - true: 成功
6   *               - false: 失败
7   * @param reason 接受邀请失败原因
8   */
9  onAcceptInviteResult(result: boolean, reason: string): void;

```

4.3.7.2 拒绝邀请

```

1  /**
2   * 拒绝邀请
3   * @returns
4   */
5  public rejectInvite(): boolean;

```

拒绝邀请结果会通过 [JRTCCallCallback](#) 里的 [onRejectInviteResult](#) 接口进行上报。

```

1  /**
2   * 拒绝邀请结果回调
3   *
4   * @param result 加入通话是否成功
5   *               - true: 成功
6   *               - false: 失败
7   * @param reason 拒绝邀请失败原因
8   */
9  onRejectInviteResult(result: boolean, reason: string): void;

```

4.3.8 邀请被取消

在收到邀请还没做出操作后邀请被取消，会收到 `onInviteCanceled` 邀请被取消的回调。

```

1  /**
2   * 对方取消邀请通知
3   *
4   * @param param 回调参数
5   */
6  onInviteCanceled(param: JRTCCallExtraParam): void;

```

示例代码：

```

1  onInviteCanceled: (param) => {
2    console.log('onInviteCanceled param', param);
3  },

```

4.3.9 视频渲染

如果是视频通话，需要渲染视频画面。

4.3.9.1 创建本地视频画面

初始化成功后，可以创建本地的视频画面，创建本地视频画面的时机没有具体要求，在加入房间前后调用皆可。

1. 通过 `startCameraVideo` 方法，获取 `JRTCMediaDeviceVideoCanvas` 本地的视频对象。

2. 在界面画布上渲染本地视频对象画面。

```
JavaScript |
1  /**
2   * 开始本端视频渲染
3   *
4   * 获取本端视频预览对象 JRTCMediaDeviceVideoCanvas, 通过此对象能获得视图用于UI显示
5   * @param renderType 视频渲染模式
6   * @param viewId 本端视频视图组件 (local-stream) id
7   * @param pageTarget 页面节点
8   * @param enableMic 是否打开麦克风, 默认打开
9   * @returns Promise
10  */
11  public async startCameraVideo(renderType: RenderType, viewId: string, page
    Target: WechatMiniprogram.Component.TrivialInstance, enableMic?: boolean):
    Promise<JRTCMediaDeviceVideoCanvas>;
```

示例代码:

```
JavaScript |
1  startCameraVideo(that) {
2      console.debug("startCameraVideo");
3      let viewId = 'local-stream';
4
5      // 或者传入viewId和document
6      that.context.mediaDevice.startCameraVideo(this.data.renderType, viewId,
    this, true).then((canvas) => {
7          that.context.localCanvas = canvas;
8      }).catch((e) => {
9          that.openTextToast('本地视频渲染失败');
10     });
11 },
```

4.3.9.2 新成员加入

当新成员加入房间后, 其他成员会收到 `onParticipantJoin` 成员加入的回调:

```

1  /**
2   * 新成员加入回调
3   *
4   * @param participant 成员对象
5   */
6  onParticipantJoin(participant: JRTCRoomParticipant) : void;

```

4.3.9.3 成员离开

当成员离开房间后，其他成员会收到 `onParticipantLeft` 成员离开的回调：

```

1  /**
2   * 成员离开回调
3   *
4   * @param participant 成员对象
5   * @param reason      成员离开原因
6   */
7  onParticipantLeft(participant: JRTCRoomParticipant, reason: ReasonCode) : void;

```

4.3.9.4 创建远端视频画面

当加入房间后，除了本地的视频画面，还有房间内其他成员的视频画面，如果房间内其他成员有视频流上传，本端可以获取到其他成员的视频流并进行渲染。

由于小程序渲染是合流模式，所以当成员视频状态变化，比如该成员开始上传视频，此时会将该成员视频与其他成员视频合流成一个视频；该成员结束上传视频，则会将该成员停止渲染只渲染其他成员视频。当所有成员都结束上传视频后，则会停止渲染视频。

4.3.9.4.1 订阅/取消订阅视频

```

1  /**
2   * 订阅房间中其他用户的视频流
3   *
4   * @param participant JRTCRoomParticipant 成员对象
5   * @param videoSize 视频请求的尺寸, 该参数目前不生效, 请使用 #setRemoteMergeVi
   deoProperty 来设置远端视频分辨率
6   * @returns Promise
7   */
8   public requestVideo(participant: JRTCRoomParticipant, videoSize: JRTCVideoS
   ize): Promise<string | {streamUrl: string, mediaStream: MediaStream}>;

```

4.3.9.4.2 渲染/停止渲染视频

```

1  /**
2   * 开始远端视频渲染
3   *
4   * @param streamUrl 远端视频拉流地址
5   * @param renderType 视频渲染模式
6   * @param viewId 视频视图组件 (remote-stream) id
7   * @param pageTarget 页面节点
8   * @returns
9   * - JRTCMediaDeviceVideoCanvas 对象: 开始远端视频渲染成功
10  * - undefined: 开始远端视频渲染失败
11  */
12  public startVideo(streamUrl: string, renderType: RenderType, viewId: strin
   g, pageTarget: WechatMiniprogram.Component.TrivialInstance): JRTCMediaDevi
   ceVideoCanvas | undefined;
13
14  /**
15   * 停止视频渲染
16   *
17   * @param canvas JRTCMediaDeviceVideoCanvas 对象, 由 {@link startVideo} 或
   {@link startCameraVideo} 接口返回
18   */
19  public stopVideo(canvas: JRTCMediaDeviceVideoCanvas): void;

```

示例代码：

```

1 this.context.call.requestVideo(participant, { width: 100, height: 100 }).then((result) => {
2     let viewId = 'remote-stream';
3     this.context.remoteCanvas = this.context.mediaDevice.startVideo(result, this.data.renderType, viewId, this);
4     if (!this.context.remoteCanvas) {
5         that.openTextToast('视频流请求失败');
6     };
7 });

```

4.3.10 结束离开

成员可以自己离开通话或者结束通话。

```

1 /**
2  * 离开通话
3  * @note 仅自己离开
4  * @note 需要已经加入通话
5  *
6  * @return 接口调用结果
7  * - true: 接口调用成功, 非空闲状态下, 会收到 {@link JRTCCallCallback.onLeave onLeave} 回调
8  * - false: 接口调用异常
9  */
10 public leave(): boolean;
11
12 /**
13  * 结束通话
14  * @note 自己离开并踢出通话中的其他成员
15  * @note 需要已经加入通话
16  *
17  * @return 接口调用结果
18  * - true: 接口调用成功, 非空闲状态下, 会收到 {@link JRTCCallCallback.onLeave onLeave} 回调
19  * - false: 接口调用异常
20  */
21 public stop();

```

离开或者结束通话会收到 `onLeave` 回调

```

1  /**
2   * 离开通话结果回调
3   *
4   * 调用 {@link JRTCCall.leave leave} 接口成功后，会收到此回调。
5   *
6   * @param reason 离开原因，参见: {@link ReasonCode}
7   */
8  onLeave(reason: ReasonCode): void;

```

示例代码：

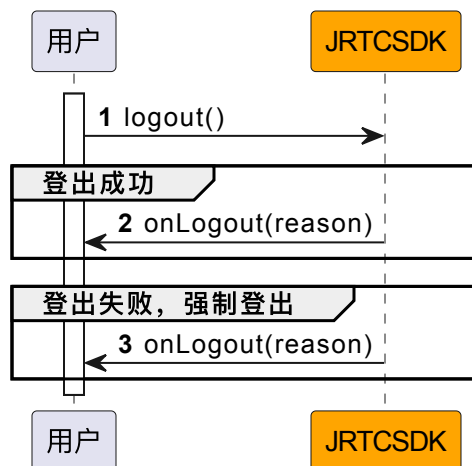
```

1  // 离开
2  this.context.call.leave();
3
4  // 结束
5  this.context.call.stop();
6
7  onLeave: (reason) => {
8    console.log('onLeave', reason);
9  };

```

4.3.11 登出

离开房间后，可以做登出操作，登出接口调用流程如下所示：



成员可以通过调用 `logout` 方法登出视频平台，与平台断开一切连接。


```

1  /**
2   * 登出 Juphoon RTC 平台，登出后不能进行平台上的各种业务
3   *
4   * 登出结果通过 {@link JRTCClientCallback.onLogout onLogout} 回调通知
5   * @returns 接口调用结果
6   * - true: 接口调用成功
7   * - false: 接口调用异常
8   */
9  public logout(): boolean;

```

登出结果通过 [JCClientCallback](#) 中的 [onLogout](#) 接口进行上报。

```

1  /**
2   * 登出回调
3   *
4   * @param reason 登出原因
5   */
6  onLogout(reason: ReasonCode): void;

```

示例代码：

```

1  this.context.client.logout();
2
3  onLogout: (reason) => {
4    console.log('onlogout:', reason);
5  };

```

4.3.12 登录状态

可以使用 [onClientStateChanged](#) 回调获取登录状态：

```

1  /**
2   * 登录状态变化通知
3   *
4   * @param state    当前状态值
5   * @param oldState 之前状态值
6   */
7  onClientStateChanged(state: ClientState, oldState: ClientState): void;

```

示例代码：

```

1  onClientStateChanged: (state, oldState) => {
2    console.log('onClientStateChanged', state, oldState);
3  };

```

4.3.13 销毁

每个模块都有对应的销毁接口。如不需再使用 SDK 的相关功能，可以强制释放 SDK 的资源。

```

1  JRTCClient.destroy();
2  this.context.client = null;
3
4  JRTCMediaDevice.destroy();
5  this.context.mediaDevice = null;
6
7  JRTCCall.destroy();
8  this.context.call = null;

```

五、消息通道

5.1 通话内消息

5.1.1 发送通话内消息

加入通话后，即可调用 `sendMessage` 方法发送通话内消息。

```

1  /**
2   * 发送消息
3   * @param contentType 消息类型
4   * @param content 消息内容
5   * @param toUserId 用户ID, 如果为空则发送给房间内所有人
6   * @returns 接口调用结果
7   * - 操作id: 接口调用成功, 对应 {@link JRTCCallCallback.onMessageSendResult} 回调的 operationId 参数
8   * - -1: 接口调用异常, 不会收到回调
9   */
10 sendMessage(contentType: string, content: string, toUserId?: string): number;

```

调用后会触发 `onMessageSendResult` 发送信息结果回调。

```

1  /**
2   * 消息发送结果回调
3   *
4   * @param result 发送结果是否成功
5   *               - true: 发送成功
6   *               - false: 发送失败
7   * @param reason 发送失败原因
8   * @param operatorId 操作id, 对应 {@link JRTCCallBase.sendMessage} 的返回值
9   */
10 onMessageSendResult(result: boolean, reason: string, operationId: number): void;

```

示例代码:

```

1  this.context.call.sendMessage(contentType, content, toUserId);
2
3  onMessageSendResult: (result, reason, operationId) => {
4    console.log('onMessageSendResult result, reason, operationId:', result, reason, operationId);
5  };

```

5.1.2 收到通话内消息

当其他成员给你发送通话内消息后，会触发 `onMessageReceived` 收到通话内消息的回调。

```
1  /**
2   * 收到消息回调
3   * @param content 消息内容
4   * @param contentType 消息内容类型
5   * @param messageType 消息归属类型
6   * - {@link MessageType.TYPE_UNKNOWN TYPE_UNKNOWN} : 未知
7   * - {@link MessageType.TYPE_1T01 TYPE_1T01} : 一对一消息
8   * - {@link MessageType.TYPE_GROUP TYPE_GROUP} : 群发消息(发送给通话中所有成员)
9   * @param fromUserId 发送方的用户ID
10  */
11  onMessageReceived(content: string, contentType: string, messageType: MessageType, fromUserId: string): void;
```

示例代码：

```
1  onMessageReceived: (content, contentType, messageType, fromUserId) => {
2    console.log('onMessageReceived content, messageType, contentType, fromUserId:', content, contentType, messageType, fromUserId);
3  };
```

5.2 在线消息

5.2.1 发送在线消息

登录成功后，即可调用 `sendOnlineMessage` 发送在线消息。

```

1  /**
2   * 发送在线消息
3   *
4   * @note 消息大小不超过4k
5   * @param message 消息内容
6   * @param userId 对端的用户名
7   * @returns 接口调用结果
8   * - 操作id: 接口调用成功, 对应 {@link JRTCClientCallback.onOnlineMessageSend
  Result onOnlineMessageSendResult} 回调的 operatorId 参数
9   * - -1: 接口调用异常, 不会收到回调
10  */
11  public sendOnlineMessage(message: string, userId: string): number;

```

调用后会触发 [onOnlineMessageSendResult](#) 发送在线消息结果的回调。

```

1  /**
2   * 在线消息发送结果回调
3   *
4   * @param result      发送结果是否成功
5   *                    - true: 发送成功
6   *                    - false: 发送失败
7   * @param operatorId 操作id, 对应 {@link JRTCClient.sendOnlineMessage sendOn
  lineMessage} 的返回值
8   */
9  onOnlineMessageSendResult(result: boolean, operatorId: number): void;

```

示例代码：

```

1  this.context.client.sendOnlineMessage(message, userId);
2
3  onOnlineMessageSendResult: (result, operatorId) => {
4    console.log('onOnlineMessageSendResult result operatorId', result, operat
  orId);
5  },

```

5.2.2 收到在线消息

当其他成员给你发送在线消息后，会触发 [onOnlineMessageReceived](#) 收到通话内消息的回调。

```

1  /**
2   * 收到在线消息回调
3   *
4   * @param message 消息内容
5   * @param userId 对方用户ID
6   */
7  onOnlineMessageReceived(message: string, userId: string): void;

```

示例代码：

```

1  onOnlineMessageReceived: (message, userId) => {
2    console.log('onOnlineMessageReceived message userId', message, userId);
3  };

```

六、通话操作

6.1 通话属性变化

通话属性变化通过实现 `JRTCCallCallback` 中的 `onCallPropertyChanged` 接口上报。

```

1  /**
2   * 通话属性变化回调
3   *
4   * 当通话的属性发生变化时，会收到此回调，例如通话中有成员发起屏幕共享、录制状态发生变化
   * 等。
5   *
6   * @param changeParam 变化标识集合
7   */
8  onCallPropertyChanged(changeParam: JRTCRoomPropChangeParam): void;

```

示例代码：

```

1 onCallPropertyChanged: (changeParam) => {
2   const shareUserId = that.context.call.getShareUserId();
3   if (shareUserId !== undefined) {
4     that.setData({ shareUserId: shareUserId });
5   };
6   if (changeParam.screenShare) {
7     that.setData({
8       screenShare: !this.data.screenShare
9     }, () => {
10      if (that.data.screenShare) {
11        that.openTextToast(that.data.shareUserId + '发起屏幕共享');
12      } else {
13        that.openTextToast(that.data.shareUserId + '的屏幕共享已结束');
14      };
15    });
16  };
17 },

```

6.2 成员属性变化

成员属性变化通过实现 `JRTCCallCallback` 中的 `onParticipantUpdate` 接口上报。

```

1 /**
2  * 成员属性更新回调
3  *
4  * 当房间中有成员的属性发生变化时，房间中的其他成员会收到此回调，例如音频上传状态、视频上传状态、网络状态等发生变化。
5  *
6  * @param participant JRTCRoomParticipant 成员对象
7  * @param changeParam {@link JRTCRoomParticipantChangeParam} 更新标识类对象
8  */
9 onParticipantUpdate(participant: JRTCRoomParticipant, changeParam: JRTCRoomParticipantChangeParam): void;

```

示例代码：

JavaScript

```

1 onParticipantUpdate: (participant, changeParam) => {
2   console.log('onParticipantUpdate changeParam', participant, changeParam);
3 },

```

6.3 获取通话内所有成员

JavaScript

```

1 /**
2  * 获取所有成员列表
3  */
4 public getParticipants(): Array<JRTCRoomParticipant>;

```

示例代码：

JavaScript

```

1 let participants = this.context.call.getParticipants();

```

6.4 获取发起屏幕共享者的用户ID

JavaScript

```

1 /**
2  * 获取发起屏幕共享者的用户ID，无屏幕共享时为 undefined
3  *
4  * 可用来判断当前通话中是否有成员发起屏幕共享。
5  * @returns 发起屏幕共享者的用户ID
6  */
7 public getShareUserId(): string | undefined;

```

示例代码：

JavaScript

```

1 const shareUserId = this.context.call.getShareUserId();

```

七、设备管理

7.1 视频设备管理

在视频场景中，您可能需要根据实际的情况选择视频的采集设备，以及相关的采集参数。

7.1.1 切换摄像头

切换摄像头，内部会根据当前摄像头类型来进行切换。

```
1  /**
2   * 切换摄像头
3   *
4   * @note 内部会根据当前摄像头类型来进行切换
5   *
6   * - 调用此方法时要保证摄像头已打开，否则将直接返回 false
7   * - 设备拥有两个以上摄像头，否则将直接返回 false
8   *
9   * @returns
10  * - true: 切换摄像头成功
11  * - false: 切换摄像头失败
12  *
13  */
14  public switchCamera(): boolean;
```

示例代码：

```
1  this.context.mediaDevice.switchCamera();
```

7.1.2 开关摄像头

```

1  /**
2   * 开启/关闭发送本地视频流
3   *
4   * - 调用该方法可开启或关闭发送本地视频流。开启后，房间成员将可以看见本端视频画面；关闭
      后，房间成员将看不见本端视频画面
5   * - 房间中调用此方法不影响接收远端视频
6   * - 初始化 JCRoom 时，默认发送本地视频流。若要加入房间时，让房间内其他成员看见本端视
      频画面，建议在调用 {@link join} 加入房间前设置
7   * - 该方法在房间内和房间外均可调用，且在离开房间后该设置仍然有效。也就是说这一次设置了
      关闭发送本地视频流，那么在下一次加入房间时默认会关闭发送本地视频流
8   * - 会议中也可调用此方法开启或关闭发送本地视频流，服务器会更新状态并同步给其他房间成
      员，即房间中所有成员都会收到 {@link JRTCCallCallback.onParticipantUpdate} 回调
9   *
10  * @param enable 是否发送本地视频流
11  * - true: 开启，即发送本地视频流
12  * - false: 关闭，即不发送本地视频流
13  * @returns 接口调用结果
14  * - true: 接口调用成功
15  * - 在调用此方法时，用户不在房间中，不会收到回调
16  * - 在调用此方法时，用户在房间中，会收到 {@link JRTCCallCallback.onCallPropertyChanged} 回调
17  * - false: 接口调用异常
18  */
19  public override enableUploadVideoStream(enable: boolean): boolean;

```

示例代码：

```

1  // 开启摄像头
2  this.context.call.enableUploadVideoStream(true)
3
4  // 关闭摄像头
5  this.context.call.enableUploadVideoStream(false)

```

7.2 音频设备管理

7.2.1 开关麦克风

```
1  /**
2   * 开启/关闭音频输入（麦克风）
3   *
4   * @note
5   * 调用该接口需要先开启本端视频预览{@link startCameraVideo}
6   *
7   * @param enable
8   * - true 开启音频输入（麦克风）
9   * - false 关闭音频输入（麦克风）
10  *
11  * @returns 调用结果
12  * - true 调用成功
13  * - false 调用失败
14  *
15  * @internal
16  */
17  async enableLocalAudio(enable: boolean): Promise<boolean>
```

示例代码：

```
1  // 开启麦克风
2  this.context.mediaDevice.enableLocalAudio(true)
3
4  // 关闭麦克风
5  this.context.mediaDevice.enableLocalAudio(false)
```