

【网络电话】Juphoon Plugin 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、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 JRTC 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 JRTC SDK 用于给客户渠道类系统和客户业务类系统集成并通过 JRTC 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。

JRTC 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。Juphoon Plugin for WeChat 插件 专为微信小程序平台设计。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：



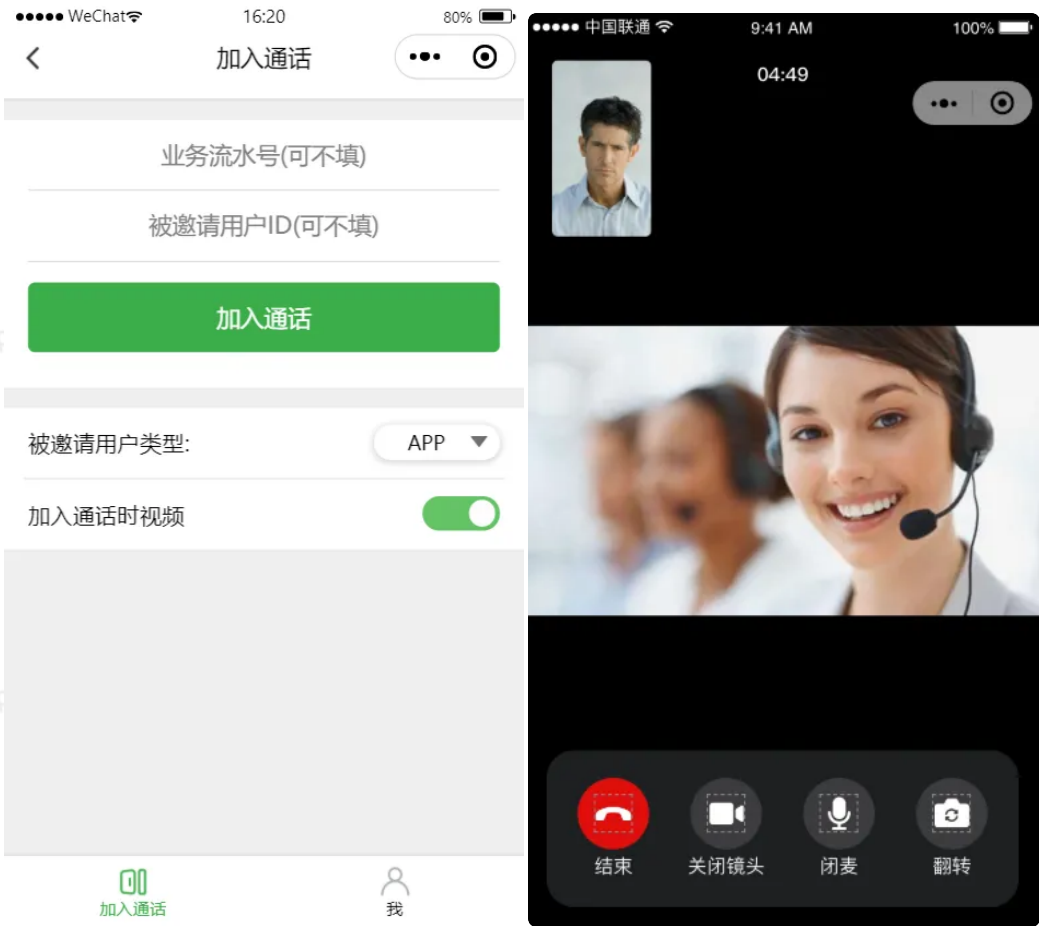
2.2 版权申明

“Juphoon RTC for WeChat Plugin”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。Juphoon 是宁波菊风系统软件有限公司商标。

三、效果图



四、快速搭建Plugin

4.1 插件概述

该插件是为小程序设计的视频通话插件，支持一对一视频通话和多方视频会议，支持发送消息。

Plugin 文件夹中包含样式文件和多个组件，style 文件夹、tools.wxs 文件、talk-view 组件、local-stream 组件、remote-stream 组件和 JRTCSDK-Wechat.js 库是必需的，网络电话还需要集成 call 组件，将它们一起放入项目 components 文件夹下即可。

4.2 引入组件

在项目的 WXML 文件中和 JSON 文件中按以下方式引入即可，引入组件时，可以将部分登录参数通过以下方式传给组件，通过该方法传递过后，登录时则无需传递已传递的登录参数。还需要传递回调函数给组件。注：引入绑定时，需全部小写，不能使用驼峰命名。

```
XML |
1 <call
2   id="call"
3   signal-server="http://192.168.19.4:19993"
4   call-server="http://192.168.19.4:9001"
5   app-key="05c97ce0d0a84d2592334093"
6   app-name="网络电话"
7   display-name=""
8   token=""
9   token-type="jrtc_access"
10  token-extra-param=""
11  bindonsessioncreate="onSessionCreate"
12  bindonlogin="onLogin"
13 ...>
14 </call>
```

```
JSON |
1 {
2   "usingComponents": {
3     "call": "/components/call/call"
4   }
5 }
```

回调函数按需传递，没有使用的无需传递。

4.3 插件通用属性

属性	类型	参数/属性值	必填	说明
signal-server	String		否	signal环境
call-server	String		否	呼叫环境
app-key	String		否	appKey
app-name	String		否	app名称
display-name	String		否	昵称

token	String		否	token
token-type	String		否	token类型
token-extra-param	String		否	token校验拓展参数
termType	Boolean	false	否	通话挂断方式 (true关闭通话, false离开通话)
videoHoldLastFrame	Boolean	false	否	保留最后一帧画面
closeCameraBackgroundndColor	String	#1C1C1E	否	关闭摄像头后背景颜色
newMessageToast	String	您收到了一条消息, 点击消息查看	否	收到新消息提示语
toastColor	String	#FFFFFF	否	消息提示文字颜色
toastBackgroundColor	String	#252525cc	否	消息提示遮罩背景色
closeCameraIconSrc	String		否	关闭摄像头图标样式
bindonlogin	Function	result、reason	否	登录的回调
bindonlogout	Function	reason	否	登出的回调
bindonsessioncreate	Function	result、reason	否	session创建的回调
bindonsessiondestroy	Function	reason	否	session销毁的回调
bindonclientstatechanged	Function	state、oldState	否	登录状态的回调

bindonmessagesendresult	Function	result、reason、operationId	否	发送通话内消息的回调
bindonmessageincallreceived	Function	content、contentType、fromUserId	否	收到通话内消息的回调
bindononlinemessagesendresult	Function	result、operationId	否	发送在线消息的回调
bindononlinemessagereceived	Function	userId、message	否	收到在线消息的回调
bindonjoin	Function	result、reason	否	加入的回调
bindonleave	Function	reason	否	离开的回调
bindoninvitereceived	Function	param	否	收到邀请的回调
bindoninvitecanceled	Function	param	否	邀请被取消的回调
bindonacceptinviteresult	Function	result、reason	否	接受邀请的回调
bindonrejectinviteresult	Function	result、reason	否	拒绝邀请的回调
bindoninviteresult	Function	result、reason	否	邀请结果的回调
bindoninviteaccepted	Function	param	否	邀请被接受的回调
bindoninviterejected	Function	param	否	邀请被拒绝的回调
bindonparticipantjoin	Function	participant	否	成员加入的回调
bindonparticipantleft	Function	participant	否	成员离开的回调

bindonerror	Function	event	否	错误事件的回调
-------------	----------	-------	---	---------

4.4 登录

在登录时需要传递几个参数，如下表。

参数名	参数含义	是否必填
signalServer	signal环境	是
callServer	呼叫环境	是
appKey	appKey	是
userId	账号	是
displayName	昵称	否
token	Token	否
tokenType	Token类型	否
tokenExtraParam	Token校验拓展参数	否

设置好参数后，即可调用 `login` 方法进行登录。登录时需要把这些参数作为一个对象传递给 `login` 方法，引入时已传递的参数则可以不用再传递。下面是一个示例。

```

1  const callComponent = this.selectComponent('#call');
2  const loginParam = {
3    signalServer: 'http://192.168.19.4:19993',
4    callServer: 'http://192.168.19.4:9001',
5    appKey: '05c97ce0d0a84d2592334093',
6    token: '',
7    tokenType: 'jrtc_access',
8    userId: '',
9    displayName: '',
10   tokenExtraParam: '',
11  };
12  callComponent.login(loginParam);

```

4.5 登出

登出无需设置参数，只需调用 `logout` 方法即可，下面是一个示例。

```
1 const callComponent = this.selectComponent('#call');
2 callComponent.logout();
```

4.6 获取登录状态

可以使用登录状态的回调获取当前登录状态，它可以获取两个参数 `state`、`oldState`，`state` 表示当前的登录状态，`oldState` 表示之前的状态。下面是一个示例。

```
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  *   state = 1 未登录
7  *   state = 2 登录中
8  *   state = 3 已登录
9  *   state = 4 登出中
10 */
11 onClientStateChanged(e) {
12   const { state, oldState } = e.detail;
13   console.log('onClientStateChanged state:', state, oldState);
14 },
```

4.7 加入通话

在加入通话之前，需要设置一些参数，如下表。

参数名	参数含义	是否必填
serialId	业务流水号(String)	业务号呼叫时必填
inviteCalleeUserType	被邀请人类型(Number)	邀请用户时必填
inviteCalleeUserId	被邀请人id(String)	是

callerNumberForSip	呼叫主叫号(String)	被邀请用户类型为Sip时填
multiStream	是否合流(Boolean)	否
video	是否视频(Boolean)	是
extraInfo	随路参数(Object)	否
routeId	线路id(String)	否
token	呼叫token(String)	否

4.7.1 加入时邀请用户

在加入时邀请用户，需要选择所邀请用户的用户类型，当类型不为 Sip 时，需要输入用户ID；当类型为 Sip 时，需要输入呼叫主叫号。

在选择好呼叫用户类型和输入完用户ID或呼叫主叫号后，即可调用 `join` 方法进行加入，调用 `join` 方法需要传递一个参数 —— `joinParam`。`joinParam` 为加入参数，是一个对象。下面是个示例。

```

1  const callComponent = this.selectComponent('#call');
2  const joinParam = {
3    serialId: '', // 业务流水号
4    inviteCalleeUserId: '', // 被邀请人id
5    inviteCalleeUserType: 0, // 被邀请人类型
6    callerNumberForSip: '', // 呼叫主叫号
7    multiStream: false, // 是否合流
8    video: true, // 是否视频
9    extraInfo: {}, // 随路参数
10   routeId: '', // 线路id
11   token: '', // 呼叫token
12 };
13 callComponent.join(joinParam);

```

4.7.2 流水号加入

流水号类似于房间号，输入流水号加入成功后即可与房间中的其他人进行通话。它仅需一个参数流水号 —— `serialId`，如上表。参数设置完成后，即可调用 `join` 方法。

4.8 邀请

在邀请时，需要设置一些参数，如下表。

参数名	参数含义	是否必填
calleeUserId	被邀请用户ID	被邀请用户类型不为Sip时填
calleeUserType	被邀请用户类型	是
video	是否视频	是
extraInfo	随路参数	否
routeId	线路ID	否
callerNumberForSip	呼叫主叫号	被邀请用户类型为Sip时填

4.8.1 邀请用户

在通话内邀请用户时，需要选择所邀请用户的[用户类型](#)，当类型不为 Sip 时，需要输入用户ID；当类型为 Sip 时，需要输入呼叫主叫号。

在选择好呼叫用户类型和输入完用户ID或呼叫主叫号后，即可调用 [invite](#) 方法邀请用户，调用 [invite](#) 方法需要传递一个参数 —— [inviteParam](#)。inviteParam 为邀请参数，是一个[对象](#)。下面是个示例。

```
1  const callComponent = this.selectComponent('#call');
2  const inviteParam = {
3    calleeUserId: '',
4    calleeUserType: 0,
5    video: true,
6    extraInfo: {},
7    routeId: '',
8    callerNumberForSip: '',
9  };
10 callComponent.invite(inviteParam);
```

4.8.2 取消邀请

在发送邀请之后，被邀请人接受/拒绝之前，可以调用 [cancelInvite](#) 方法取消邀请。下面是一个示例。

```
1 const callComponent = this.selectComponent('#call');  
2 callComponent.cancelInvite();
```

4.8.3 接受邀请

4.8.3.1 视频接受邀请

当收到邀请时，可以调用 `acceptInvite` 方法接受邀请，调用时可以传入一个布尔值，当传入的为 `true` 时，则为视频接受邀请。调用成功后会进入到通话页面，下面是一个简单的示例。

```
1 const callComponent = this.selectComponent('#call');  
2 callComponent.acceptInvite(true);
```

4.8.3.2 语音接受邀请

当收到邀请时，可以调用 `acceptInvite` 方法接受邀请，调用时可以传入一个布尔值，当传入的为 `false` 时，则为语音接受邀请。调用成功后会进入到通话页面，下面是一个简单的示例。

```
1 const callComponent = this.selectComponent('#call');  
2 callComponent.acceptInvite(false);
```

4.8.4 拒绝邀请

当收到邀请时，也可以调用 `rejectInvite` 方法拒绝邀请，下面是一个简单的示例。

```
1 const callComponent = this.selectComponent('#call');  
2 callComponent.rejectInvite();
```

4.9 发送消息

4.9.1 发送通话内消息

发送通话内消息应发生在加入通话之后，否则会报错。在发送消息时，需要填写三个参数，如下表。

参数名	参数含义	是否必填
toUserId	接收用户账号	否
contentType	消息类型	是
content	消息内容	是

如果 toUserId 不填，则会发送给会场内所有人，如果填了，则是发送给所填的那个用户。在填好之后，即可调用 `sendMessageInCall` 方法发送通话内消息，调用时需传入一个对象参数 —— `sendMessageInCallInfo`，下面是一个示例。

```
JavaScript |
1  const callComponent = this.selectComponent('#call');
2  const sendMessageInCallInfo = {
3    contentType: '',
4    toUserId: '',
5    content: '',
6  };
7  callComponent.sendMessageInCall(sendMessageInCallInfo);
```

4.9.2 发送在线消息

登录成功后，即可发送在线消息。在发送在线消息时，需要填写两个参数，如下表。

参数名	参数含义	是否必填
message	消息内容	是
userId	接收用户账号	是

在填好接受用户账号和消息内容后，即可调用 `sendOnlineMessage` 方法发送在线消息，调用时需要传入一个对象参数 —— `sendOnlineMessageInfo`，下面是一个示例。

```

1  const callComponent = this.selectComponent('#call');
2  const sendOnlineMessageInfo = {
3    message: '',
4    userId: '',
5  };
6  callComponent.sendOnlineMessage(sendOnlineMessageInfo);

```

4.10 预检测功能

使用预检测功能需要引入 pre-check 组件，该组件可用于检测设备扬声器、麦克风和摄像头是否有用。

```

1  <view wx:if="{{isChecked}}">
2    <pre-check
3      id="pre-check"
4      bindischeckingchanged="isCheckedChanged">
5    </pre-check>
6  </view>

```

当需要控制 pre-check 组件的显示隐藏时，可以传入 ischeckingchanged 方法，并定义一个参数——isChecked。下面是一个示例。

```

1  data: {
2    isChecked: false;
3  },
4  isCheckedChanged() {
5    this.setData({
6      isChecked: !this.data.isChecked,
7    });
8  },

```

4.11 获取版本号信息

引入组件后，可调用 getVersion 方法获取当前组件版本信息，下面是一个示例。

```

1  const callComponent = this.selectComponent('#call')
2  const version = callComponent.getVersion()
3  console.log('version:', version);

```

4.12 设置用户角色

在进行通话之前，可以设置用户角色，如下表。

参数名	参数含义	是否必填
roleType	角色类型(Number)	否

可以在 data 中定义初始值，也可以绑定 `setCallRole` 事件来更改对应数据。下面是一个示例。

```

1  const callComponent = this.selectComponent('#call');
2  callComponent.setCallRole(roleType)

```

4.13 获取当前通话ID

在加入通话后，可调用 `getCallId` 方法获取当前通话的id，下面是一个示例。

```

1  const callComponent = this.selectComponent('#call');
2  const callId = callComponent.getCallId();
3  console.log('callId:', callId);

```

4.14 回调函数

配置如下：

```
1 <call
2   id="call"
3   bindonsessioncreate="onSessionCreate"
4   bindonsessiondestroy="onSessionDestroy"
5   bindonlogin="onLogin"
6   bindonlogout="onLogout"
7   bindonclientstatechanged="onClientStateChanged"
8   bindonmessagesendresult="onMessageSendResult"
9   bindonmessageincallreceived="onMessageInCallReceived"
10  bindononlinemessagesendresult="onOnlineMessageSendResult"
11  bindononlinemessagereceived="onOnlineMessageReceived"
12  bindonjoin="onJoin"
13  bindonleave="onLeave"
14  bindonreceivedinvite="onReceivedInvite"
15  bindoninvitecanceled="onInviteCanceled"
16  bindonacceptinviteresult="onAcceptInviteResult"
17  bindonrejectinviteresult="onRejectInviteResult"
18  bindoninviteresult="onInviteResult"
19  bindoninviteaccepted="onInviteAccepted"
20  bindoninviterejected="onInviteRejected"
21  bindonparticipantjoin="onParticipantJoin"
22  bindonparticipantleft="onParticipantLeft"
23  bindonerror="onError">
24 </call>
```

4.14.1 session创建的回调

示例代码：

```
1  /**
2   * session 创建结果回调
3   *
4   * @param result 创建结果
5   * - true: 成功
6   * - false: 失败
7   *
8   * @param reason 原因
9   */
10 onSessionCreate(e) {
11     const { result, reason } = e.detail;
12 },
```

4.14.2 session销毁的回调

示例代码:

```
1  /**
2   * session 销毁回调
3   *
4   * @param reason 原因
5   */
6  onSessionDestroy(e) {
7     const { reason } = e.detail;
8     console.log('onSessionDestroy', reason);
9 },
```

注: session 销毁后需要重新登录。

4.14.3 登录的回调

示例代码:

```
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(e) {
8      const { result, reason } = e.detail;
9      console.log('onLogin result: ', result, ' reason: ', reason);
10 }
```

4.14.4 登出的回调

示例代码:

```
1  /**
2   * 登出回调
3   *
4   * @param reason 登出原因
5   */
6  onLogout(e) {
7      const { reason } = e.detail;
8      console.log('onLogout reason: ', reason);
9  }
```

4.14.5 发送在线消息的回调

示例代码:

```

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 sendOnlineMessage} 的返回值
8   */
9  onOnlineMessageSendResult(e) {
10     const { result, operatorId } = e.detail;
11     this.handleEvent('openTextToast', '发送' + (result ? '成功' : '失败') + 'operatorId:' + operatorId);
12 },

```

4.14.6 收到在线消息的回调

示例代码:

```

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(e) {
8     const { userId, message } = e.detail;
9     this.handleEvent('openTextToast', '收到' + userId + '发来的在线消息: ' + message);
10 },

```

4.14.7 发送通话内消息的回调

示例代码:

```

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 JRTCBase.sendMessage sendMessage} 的返回值
9   */
10 onMessageSendResult(e) {
11     const { result, reason, operatorId } = e.detail;
12     this.handleEvent('openTextToast', '发送' + (result ? '成功' : '失败') + '
    operatorId:' + operatorId + 'reason:' + reason);
13 },

```

4.14.8 收到通话内消息的回调

示例代码:

```

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 onMessageInCallReceived(e) {
12     const { content, contentType, messageType, fromUserId } = e.detail;
13     this.handleEvent('openTextToast', '收到' + fromUserId + '发送的消息: ' + c
    ontent + '消息类型:' + contentType + '消息归属类型' + messageType);
14 },

```

4.14.9 加入通话的回调

示例代码:


```

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(e) {
12      const { result, reason } = e.detail;
13      console.log('onJoin result reason: ', result, reason);
14  }

```

4.14.10 离开通话的回调

示例代码：

```

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(e) {
9      const { reason } = e.detail;
10     console.log('onLeave reason: ', reason);
11 }

```

4.14.11 收到邀请的回调

示例代码：

```
1  /**
2   * 收到邀请通知
3   *
4   * @param param 回调参数
5   *   param:{
6   *     calleeUserId 被邀请者id
7   *     callerDisplayName 邀请者昵称
8   *     callerUserId 邀请者id
9   *     routeId 拨号线路选择
10  *     serialId 业务流水号
11  *     video 是否视频通话
12  *   }
13  */
14  onReceivedInvite(e) {
15    const { param } = e.detail;
16    console.log('onReceivedInvite param:', param);
17  }
```

4.14.12 收到取消邀请的回调

示例代码:

```
1  /**
2   * 对方取消邀请通知
3   *
4   * @param param 回调参数
5   */
6  onInviteCanceled(e) {
7    const { param } = e.detail;
8    console.log('onInviteCanceled param: ', param);
9  }
```

4.14.13 接受邀请的回调

示例代码:

```
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(e) {
10     const { result, reason } = e.detail;
11     console.log('onAcceptInviteResult result, reason: ', result, reason);
12 }
```

4.14.14 拒绝邀请的回调

示例代码：

```
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(e) {
10     const { result, reason } = e.detail;
11     console.log('onRejectInviteResult result, reason: ', result, reason);
12 }
```

4.14.15 邀请的回调

示例代码：

```
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(e) {
10     const { result, reason } = e.detail;
11     console.log('result, reason', result, reason);
12 }
```

4.14.16 取消邀请的回调

示例代码:

```
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  onInviteResult(e) {
10     const { result, reason } = e.detail;
11     console.log('result, reason', result, reason);
12 }
```

4.14.17 邀请被接受的回调

示例代码:

```
1  /**
2   * 对方接受邀请通知
3   *
4   * @param param 回调参数
5   */
6  onInviteAccepted(e) {
7      const { param } = e.detail;
8      console.log('onInviteAccepted param:', param);
9  }
```

4.14.18 邀请被拒绝的回调

示例代码：

```
1  /**
2   * 对方拒绝邀请通知
3   *
4   * @param param 其他参数
5   * @see JRTCCallExtraParam
6   */
7  onInviteRejected(e) {
8      const { param } = e.detail;
9      console.log('onInviteRejected param:', param);
10 }
```

4.14.19 成员加入的回调

示例代码：

```

1  /**
2   * 新成员加入回调
3   *
4   * @param participant 成员对象
5   *   participant: {
6   *     userId 成员的账号
7   *     userUri 成员的地址
8   *     virtual 是否为虚拟用户
9   *   }
10  */
11  onParticipantJoin(e) {
12    const { participant } = e.detail;
13    console.log('onParticipantJoin participant: ', participant);
14  }

```

4.14.20 成员离开的回调

示例代码:

```

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(e) {
8    const { participant, reason } = e.detail;
9    console.log('onParticipantLeft participant reason: ', participant, reason);
10  }

```

4.14.21 发生错误的回调

示例代码:

```

1  /**
2   * 错误事件回调
3   *
4   * @param event 事件对象
5   *   event:{
6   *       errorCode    错误码
7   *       errorCodeDetail  错误描述
8   *   }
9   */
10 onError(e) {
11     const { event } = e.detail
12     console.log('event:', event);
13 },

```

4.15 参数

4.15.1 reason

参数值	含义
NONE = 0	正常
OTHER	其他原因
INVALID_PARAM	参数错误
CLI_STATE_CANNOT_LOGIN	当前状态无法再次登录
CLI_TIMEOUT	登录超时
CLI_NETWORK	登录网络异常
ROOM_TIMEROUT	接入房间网络超时
ROOM_NERWORK	接入房间网络异常
ROOM_KICKED	被踢出房间
ROOM_OFFLINE	接入房间掉线
ROOM_QUIT	主动离开房间
ROOM_OVER	房间关闭

ROOM_FULL	房间满员
ROOM_INVALID_PASSWORD	房间密码无效
REASON_CONF_NUMBER_NOT_FOUND	该房间号的房间不存在
ROOM_APP_CONCURRENCY_FULL	服务器房间成员总数上限（移动端房间人数）
ROOM_ALL_CONCURRENCY_FULL	服务器房间成员总数上限（总房间人数）
ROOM_INTERNAL_ERROR	房间异常（服务端下发）
ROOM_JOIN_LICENCE_LIMIT	会场人员上限，licence限制
ROOM_AGENT_RELEASE	录制cd被释放

4.15.2 roleType（用户角色）

roleType	说明
0	主座席
1	座席
2	主访客
3	访客

4.15.3 userType（用户类型）

userType	说明
0	APP
1	Sip
2	H5
3	WeChat

4.15.4 clientState（登录状态）

参数值	含义
1	未登录
2	登录中
3	已登录
4	登出中

4.15.5 inviteParam (邀请参数)

参数值	含义
calleeUserId	被邀请者id
callerDisplayName	邀请者昵称
callerUserId	邀请者id
routeld	拨号线路选择
serialId	业务流水号
video	是否视频通话

4.15.6 participant (成员信息)

参数值	含义
userId	成员的账号
userUri	成员的地址
virtual	是否为虚拟用户