

【网络电话】Juphoon Plugin for H5 开发集成指南



开发指导手册（H5）

宁波菊风系统软件有限公司

2025年01月

版权所有©宁波菊风系统软件有限公司 2024。保留一切权利。

版本	作者	日期	说明
v2501.0	徐广阔	2025.01	初版

1. 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 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 iOS 插件 专为 iOS 平台设计，适用于 iPhone、iPad、iTouch 等 Apple 公司移动终端设备。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。

2. 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取得联系：

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：



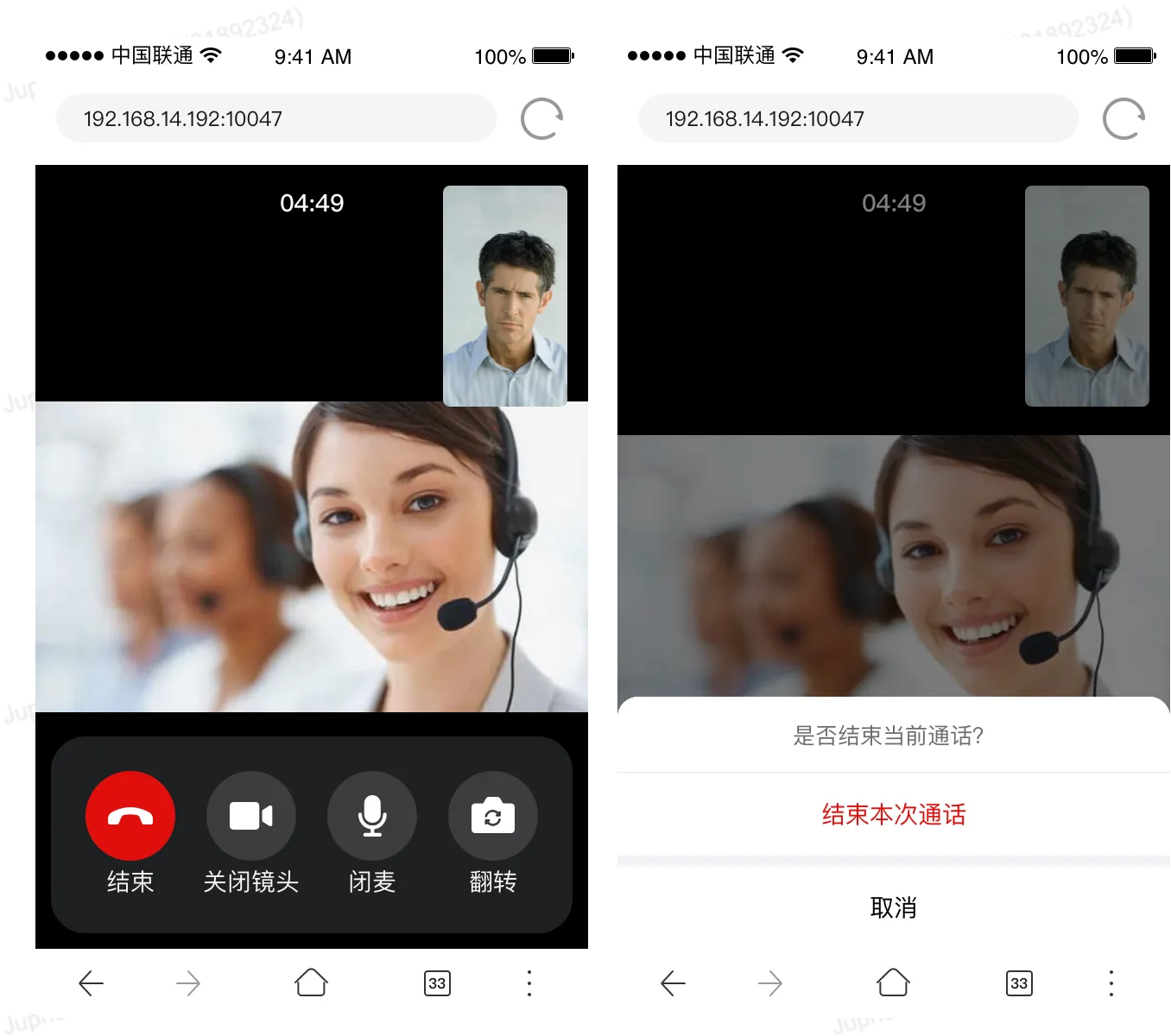
2.2 版权申明

“Juphoon RTC for H5 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

限公司的商标。

3. 效果图



4. 如何集成Plugin

4.1. 开发环境要求

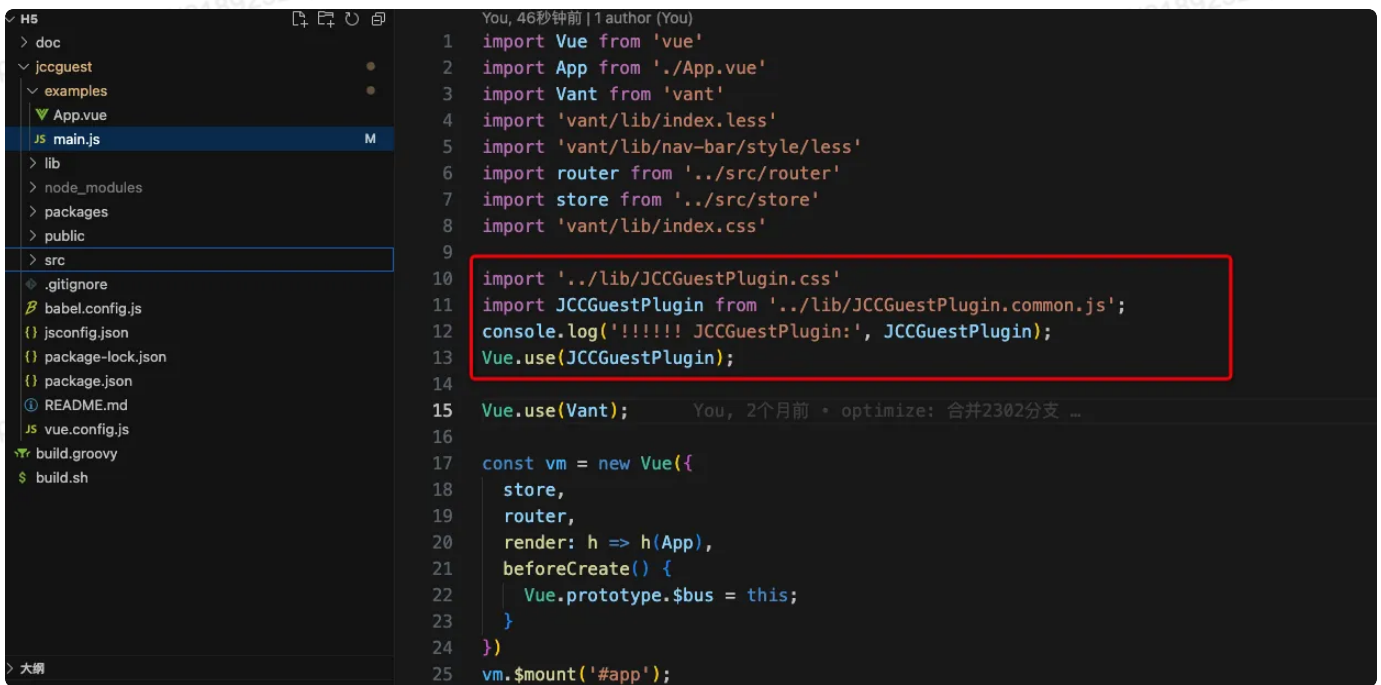
开发框架：Vue

4.2. 导入Plugin

JCCGuestPlugin.css

JCCGuestPlugin.js

将 JCCGuestPlugin.js 和 JCCGuestPlugin.css 一起导入到工程中，将 JCCGuestPlugin 注入到Vue 中。



```
1 import Vue from 'vue'
2 import App from './App.vue'
3 import Van from 'van'
4 import 'vant/lib/index.less'
5 import 'vant/lib/nav-bar/style/less'
6 import router from '../src/router'
7 import store from '../src/store'
8 import 'vant/lib/index.css'
9
10 import '../lib/JCCGuestPlugin.css'
11 import JCCGuestPlugin from '../lib/JCCGuestPlugin.common.js';
12 console.log('!!!!!! JCCGuestPlugin:', JCCGuestPlugin);
13 Vue.use(JCCGuestPlugin);
14
15 Vue.use(Vant);
16
17 const vm = new Vue({
18   store,
19   router,
20   render: h => h(App),
21   beforeCreate() {
22     Vue.prototype.$bus = this;
23   }
24 })
25 vm.$mount('#app');
```

使用到 JCCGuestPlugin.js 中的其他项，可以在使用的地方导出。

```

import { JCCCommonCustomConfig, JCCGuestCustomConfig } from '../lib/JCCGuestPlugin.js';
import { JCKRoomManager } from './JCKRoomManager';
import { JCGuestManager } from './JCGuestManager';
import { JCCallManager } from './JCCallManager';
import store from '@store'
import { LoginState, SceneMode } from './businessConst';
import pkg from '../package.json';

/** ----- JCManager ----- */

```

导入其他类

```

import { JCCallManager, JCCallStateChangeType } from '../lib/JCCGuestPlugin.js';
import store from '@store'
import { LoginState } from './businessConst'
import Vue from 'vue'

You, 8小时前 | 1 author (You)
export class JCCallManager {

    static instance = null;

```

导入其他类

5. 快速搭建Plugin

可以查看 JCCGuestManager.md(接口说明文档) 和 JCCGuestCallBack.md(接口回调说明文档)了解支持的接口和回调函数，便于快速实现插件的搭建。

5.1. 初始化销毁

在开始业务前，需要初始化对象，设置回调函数。使用静态 shared 方法获取对象。

```

1  /**
2   * 单例
3   */
4   static shared(): JCCCommonManager<T>
5
6   /**
7   * 添加回调
8   *
9   * @param callback 回调接口对象，用于接收 JCCCommonManager 相关通知
10  */
11  addCallback(callback: JCCCommonManager): void;
12
13  /**
14  * 删除回调
15  *
16  * @param callback 回调接口对象，用于接收 JCCCommonManager 相关通知
17  */
18  removeCallback(callback: JCCCommonManager): void;
19

```

示例代码

```

1  // 添加回调
2  JCCCallManager.shared().addCallback(this);
3
4  // 登录
5  JCCCallManager.shared().login({userName:userId, appKey: appKey, server:server});
6
7  ...
8
9  // 删除回调
10 JCCCallManager.shared().removeCallback(this);

```

5.2. 登录

登录菊风账号，调用 `login`，登录过程中进行初始化登录配置。

注意：登录成功以后需要关注session的创建销毁回调，如果收到 `onSessionDestroy` 回调表示session被销毁，需要重新登录。

```
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 onLogin 回调通知
5   * @param param 登录参数
6   * @returns 接口调用结果
7   * - true: 接口调用成功
8   * - false: 接口调用异常
9   * @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
10  * @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
11  */
12  public login(param: JCCClientLoginParam): boolean;
```

回调函数

```
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: JCCReasonCode): void;
8
9  /**
10   * 登录状态变化通知
11   *
12   * @param state    当前状态值
13   * @param oldState 之前状态值
14   */
15  onClientStateChanged(state: JCCClientState, oldState: JCCClientState): void;
16
17  /**
18   * session 创建结果回调
19   *
20   * @param result 创建结果
21   * - true: 成功
22   * - false: 失败
23   *
24   * @param reason 原因
25   */
26  onSessionCreate(result: boolean, reason: string): void;
27
28  /**
29   * session 销毁回调
30   * 须重新登录, 再进行业务
31   * @param reason 原因
32   */
33  onSessionDestroy(reason: string): void;
```

示例代码:

```
1 console.log('login user', user);
2 const userId = user.userName;
3 const appKey = user.appKey;
4 const server = user.server;
5 const ret = JCCallManager.shared().login({userName:userId, appKey: appKey
, server:server});
6
7 /**----- 回调处理 -----*/
8
9 // 登录结果回调
10 onLogin(result, reason) {
11     console.log('onLogin result: ', result, ' reason: ', reason);
12 }
13
14 // 录状态变化通知
15 onClientStateChanged(state, oldState) {
16     console.log('onClientStateChanged state: ', state);
17     // 处理被异常登出情况
18 }
```

5.3. 加入通话

使用 JCCGuestPlugin 组件，放入到应用层的通话页面中。

```

1 <template>
2   <div style="width: 100%;height: 100%;">
3     <JCCGuestPlugin ref="dualrecord"></JCCGuestPlugin>
4   </div>
5 </template>

```

```

7 <script>

```

```

9 export default {

```

```

1   data() {
2     return {
3
4     },
5
6   },
7
8   mounted() {
9     this.registerObserver();
10
11   },
12
13   methods: {

```

调用 join 方法，加入房间进行通话，在通话状态变为通话中时，进入到通话页面。

JavaScript |

```

1  /**
2   * 加入通话
3   * 结果通过 onJoin 回调通知
4   *
5   * @note 需要已经登录
6   * @note 该接口支持加入通话并且邀请其他用户加入，通过 {@link JCCallJoinParam.inviteCalleeUserId inviteCalleeUserId}
7   * 和 {@link JCCallJoinParam.inviteCalleeUserType inviteCalleeUserType} 设置
8   * 需要邀请的用户ID和用户类型
9   *
10  * 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。
11  * 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。
12  *
13  * @param joinParam 加入通话参数
14  * @param serialId 业务流水号，保证唯一，不填则自动填充
15  * @returns boolean
16  * true 表示调用成功
17  * false 表示调用失败
18  */
19 join(joinParam: JCCallJoinParam, serialId: string | undefined): boolean;

```

回调函数

```
1  /**
2   * 加入通话结果回调
3   *
4   * @param result 加入通话是否成功
5   * - true: 成功
6   * - false: 失败
7   * @param reason 加入失败原因, 当 result 为 false 时该值有效。失败原因参见: {@link JCCReasonCode}
8   */
9   onJoin(result: boolean, reason: JCCReasonCode): void;
```

示例代码


```
1  const joinParam = new JCCallJoinParam(); // 加入参数
2  // 加入的同时邀请其他成员
3  if (this.userId) {
4      joinParam.inviteCalleeUserId = this.userId; // 被邀请人userId
5      joinParam.inviteCalleeUserType = parseInt(this.userType); // 被邀请人用户类型
6      const multiStream = SettingManager.shared().getMultiStreamMode();
7      joinParam.multiStream = multiStream; // 分流
8      joinParam.video = this.isVideo; // 是否视频
9  }
10 joinParam.captureResolution = SettingManager.shared().getCaptureResolution(); // 采集分辨率
11 joinParam.captureFrameRate = SettingManager.shared().getCaptureFrameRate(); // 采集帧率
12
13 joinParam.routeId = SettingManager.shared().getRouteId(); // 线路Id
14 joinParam.extraInfo = SettingManager.shared().getExtraInfo(); // 额外参数
15 JCCallManager.shared().join(joinParam, this.serialId);
16
17 /**----- 回调处理 -----*/
18
19 onJoin(result, reason) {
20     console.log('!!! sample !!! call onJoin ', result);
21     if (result) { // 进入通话
22
23     } else {
24
25     }
26 }
27
```

5.4. 邀请

可以在通话中调用 `invite` 邀请其他成员加入通话。

```

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 onInviteResult 回调的 operatorId 参数
9   * - -1: 接口调用异常, 不会收到回调
10  */
11  invite(calleeUserId: string, inviteParam: JRTCCallExtraParam): boolean;

```

回调函数

```

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;
10
11  /**
12   * 对方接受邀请通知
13   *
14   * @param param 回调参数
15   */
16  onInviteAccepted(param: JCCallExtraParam): void;
17
18  /**
19   * 对方拒绝邀请通知
20   *
21   * @param param 其他参数
22   * @see JCCallExtraParam
23   */
24  onInviteRejected(param: JCCallExtraParam): void;

```

示例代码

```

1  const param = new JCCallExtraParam();
2  param.video = this.inviteParam.isVideo; // 是否视频邀请
3  param.calleeUserId = this.inviteParam.userId; // 被邀请人Id
4  param.calleeUserType = this.inviteParam.userType; // 被邀请人类型
5  param.callerDisplayName = this.inviteParam.displayName; // 昵称
6  JCCallManager.shared().invite(this.inviteParam.userId, param);
7
8  /**----- 回调处理 -----*/
9
10 // 邀请结果
11 onInviteResult: (result, reason) => {
12     console.log('onInviteResult result: reason:', result, reason);
13 },
14
15 // 邀请被接受
16 onInviteAccepted: param => {
17     console.log('onInviteAccepted param:', JSON.stringify(param));
18 },
19
20 // 邀请被拒绝
21 onInviteRejected: param => {
22     console.log('onInviteCanceled param:', JSON.stringify(param));
23 },

```

5.5. 取消邀请

在邀请其他成员，被邀请成员还未接受或者拒绝的时候可以调用 `cancelInvite` 取消邀请。

```

1  /**
2   * 取消邀请
3   *
4   * @returns boolean
5   * true 表示调用成功
6   * false 表示调用失败
7   */
8  cancelInvite(): boolean;

```

回调函数

```

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  JCCallManager.shared().cancelInvite();
2
3  /**----- 回调处理 -----*/
4
5  onCancelInviteResult: (result, errorCodeDetail) => {
6      console.log('onCancelInviteResult result:', result);
7  }
8

```

5.6. 接受邀请

当收到邀请时, 会收到 `onInviteReceived` 回调, 可以调用 `acceptInvite` 方法接受邀请。

```

1  /**
2   * 接受邀请
3   *
4   * @param video 是否需要视频
5   * @param isMultiStream 是否多流, 分流, 默认为 false
6   *
7   * @returns boolean
8   * true 表示调用成功
9   * false 表示调用失败
10  */
11  acceptInvite(video: boolean, isMultiStream: boolean): boolean;

```

回调函数

```

1  /**
2   * 收到邀请通知
3   *
4   * @param param 回调参数
5   */
6  onInviteReceived(param: JCCallExtraParam): void;
7
8  /**
9   * 接受邀请结果回调
10  *
11  * @param result 加入通话是否成功
12  *               - true: 成功
13  *               - false: 失败
14  * @param reason 接受邀请失败原因
15  */
16  onAcceptInviteResult(result: boolean, reason: string): void;

```

示例代码

```

1  onInviteReceived: (param) => {
2      console.log('onInviteReceived param:', JSON.stringify(param));
3      const isVideo = false; // 是否视频接受邀请
4      const isMultiStream = false; // 是否分流
5      JCCallManager.shared().acceptInvite(isVideo, isMultiStream); // 接受邀
   请
6  },
7
8  /**----- 回调处理 -----*/
9
10 onAcceptInviteResult: (result, reason) => {
11     console.log('onAcceptInviteResult result: reason:', result, reason);
12 },

```

5.7. 拒绝邀请

当收到邀请时, 也可以调用 `rejectInvite` 方法拒绝邀请。

```

1  /**
2   * 拒绝邀请
3   *
4   * @returns boolean
5   * true 表示调用成功
6   * false 表示调用失败
7   */
8   rejectInvite(): boolean;

```

回调函数

```

1  /**
2   * 收到邀请通知
3   *
4   * @param param 回调参数
5   */
6   onInviteReceived(param: JCCallExtraParam): void;
7
8  /**
9   * 拒绝邀请结果回调
10  *
11  * @param result 加入通话是否成功
12  *               - true: 成功
13  *               - false: 失败
14  * @param reason 拒绝邀请失败原因
15  */
16  onRejectInviteResult(result: boolean, reason: string): void;

```

示例代码

```

1  onInviteReceived: (param) => {
2      console.log('onInviteReceived param:', JSON.stringify(param));
3      JCCallManager.shared().rejectInvite(); // 拒绝邀请
4  },
5
6  /**----- 回调处理 -----*/
7
8  onRejectInviteResult: (result, reason) => {
9      console.log('onRejectInviteResult result: reason:', result, reason);
10 },

```

5.8. 离开通话

插件中已经包含了离开通话的按钮，如果想要应用层控制成员离开，可以调用 **leave** 离开通话。通过 **onLeave** 回调结果，判断当前通话状态。

```
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 onLeave 回调
8   * - false: 接口调用异常
9   */
10 leave(): boolean;
```

示例代码

```
1  JCCallManager.shared().leave();
2
3  /**----- 回调处理 -----*/
4
5  onLeave(reason) {
6      console.log('!!! sample !!! call onLeave reason: ', reason);
7      // 返回上个页面
8  }
```

5.9. 结束通话

如果想结束整个通话，可以调用 **stop** 来结束通话。通过 **onLeave** 回调结果。

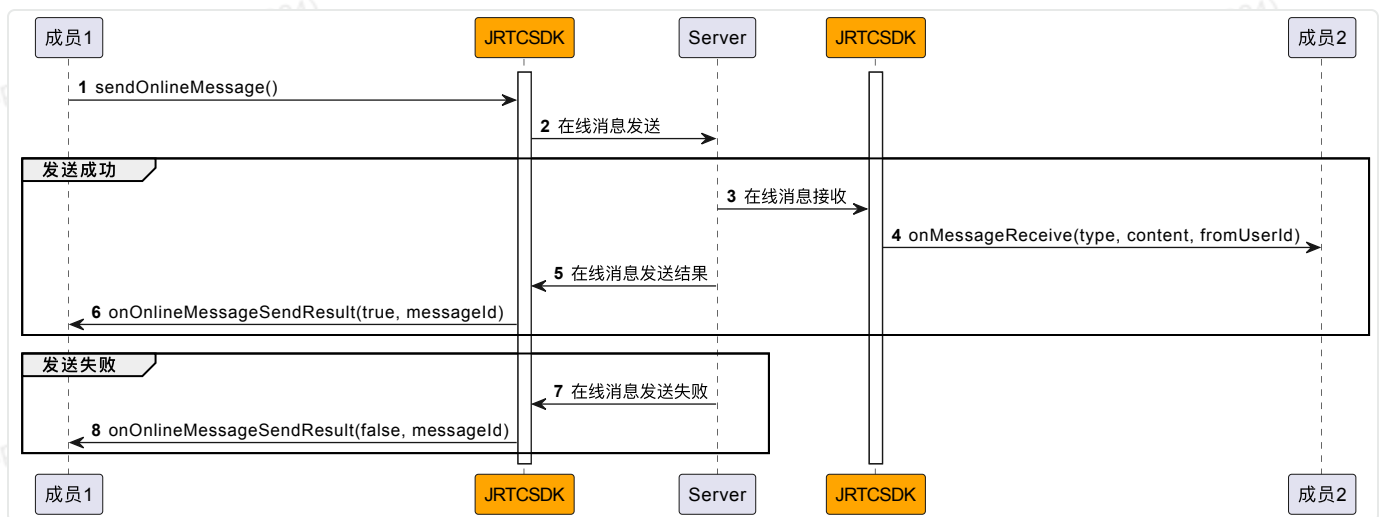
```

1  /**
2   * 结束通话
3   * @note 自己离开并踢出通话中的其他成员
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 onLeave 回调
8   * - false: 接口调用异常
9   */
10 stop(): boolean;

```

5.10. 发送消息

5.10.1. 在线消息



只要登录到 Juphoon RTC 平台就可以通过 `sendOnlineMessage` 实现在线消息的收发，消息内容不能大于4K。


```

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 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;

```

在线消息发送结果通过 onOnlineMessageSendResult 回调通知。

```

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 sendOnlineMessage 的返回值
8   */
9  onOnlineMessageSendResult(result: boolean, operatorId: number): void;

```

在线消息接收者会收到 onOnlineMessageReceived 回调通知。

```

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 // 给成员 agent1 发送在线消息
2 JCCallManager.shared().sendOnlineMessage("content", "agent1");
3
4 /**----- 回调处理 -----*/
5
6 // 在线消息发送结果回调
7 onOnlineMessageSendResult(result, operatorId) {
8
9 }
10
11 // 收到消息
12 onMessageReceived(content, fromUserId) {
13     // 收到消息，消息内容为 content，消息来自 fromUserId
14 }

```

5.11. 登出

调用 `logout` 登出账号。

```

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 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 logout(): boolean;

```

回调函数

```

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

```

```

JavaScript |
1  JCCallManager.shared().logout();
2
3  /**----- 回调处理 -----*/
4
5  onLogout: (reason) => {
6      console.log('onLogout reason: ', reason);
7      // 账号登出, 销毁处理
8  },

```

6. Plugin进阶应用

6.1. 设置定制化参数

```

JavaScript |
1  /**
2   * 设置定制配置参数
3   *
4   * @param customConfig 定制化配置参数
5   */
6  setCustomConfig(config: JCCCommonCustomConfig<T>);

```

定制化配置参数

```

JavaScript |
1  // 通用定制化配置
2  class JCCCommonCustomConfig {
3      talkingConfig: JCCTalkingConfig;
4  }
5
6  // 通话参数
7  class JCCTalkingConfig {
8      toolBarBackground: string; // 工具栏背景色
9      localVideoFullScreen: boolean; // 本端全屏显示, 默认true
10     remoteVideoFullScreen: boolean; // 远端全屏显示, 默认false
11     videoHoldLastFrame: boolean; // 视频保持最后一帧, 默认false
12 }

```

示例代码

```
1  const commonCustomConfig = new JCCCommonCustomConfig();
2  commonCustomConfig.talkingConfig.localVideoFullScreen = false;
3  ...
4  JCCCallManager.shared().setCustomConfig(commonCustomConfig);
```

7. 参数说明

7.1. 登录参数 JCCClientLoginParam

```
1  class JCCClientLoginParam {
2
3      userName = ''; // 用户名
4      displayName = ''; // 昵称
5      appKey = ''; // 用户从 Juphoon RTC 平台上申请的 AppKey 字符串
6      server = ''; // 服务地址
7
8      terminalType = ''; // 获取终端登录类型
9      deviceId = ''; // 获取设备ID
10     token = ''; // 校验token
11     tokenType = ''; // token类型
12     tokenExtraParam = ''; // token额外参数
13 }
```

7.2. 通话加入参数 JCCCallJoinParam

```
1
2 // 网络电话call通话加入参数
3 class JCCallJoinParam {
4
5     // 被邀请人userId
6     inviteCalleeUserId = '';
7
8     // 被邀请人用户类型
9     inviteCalleeUserType = -1; // JCCallUserType
10
11     // 邀请主叫号，用于SIP呼叫
12     callerNumberForSip = '';
13
14     // 是否视频
15     video = true;
16
17     // 是否合流
18     multiStream = false;
19
20     // 随路参数，对象形式传值
21     extraInfo = null;
22
23     // 线路Id
24     routeId = '';
25
26     // 外部自定义流水号
27     extraSerialId = '';
28 }
29
```

7.3. 通话额外参数 JCCallExtraParam

```
1 // 网络电话call通话额外参数，用于邀请，被邀请
2 class JCCallExtraParam {
3
4     // 是否视频通话
5     video = false;
6
7     // 业务流水号
8     serialId = "";
9
10    // 邀请人userId
11    callerUserId = "";
12
13    // 邀请人昵称
14    callerDisplayName = "";
15
16    // 呼叫主叫号，用于SIP呼叫
17    callerNumberForSip = "";
18
19    // 被邀请人userId
20    calleeUserId = "";
21
22    // 被邀请人昵称
23    calleeDisplayName = "";
24
25    // 用户类型
26    calleeUserType = -1; // JCCallUserType
27
28    // 随路参数，对象形式传值
29    extraInfo = null;
30
31    // routeId参数 拨号线路选择
32    routeId = "";
33 }
```

7.4. 通话属性变化参数 JCCRoomPropChangeParam

```
1
2 // 通话房间属性变化参数
3 export class JCCRoomPropChangeParam {
4
5     /**
6      * 上传音频状态是否变化
7      * - true: 变化
8      * - false: 没变化
9      */
10    uploadAudio = false;
11
12    /**
13     * 上传视频状态是否变化
14     * - true: 变化
15     * - false: 没变化
16     */
17    uploadVideo = false;
18
19    /**
20     * 远程录制状态是否变化
21     * - true: 变化
22     * - false: 没变化
23     */
24    remoteRecordState = false;
25
26    /**
27     * 屏幕共享状态是否变化
28     * - true: 变化
29     * - false: 没变化
30     */
31    screenShare = false;
32 }
```

7.5. 通话成员 JCCRoomParticipant

```
1 // 通话成员
2 class JCCRoomParticipant {
3     netStatus = 0;
4     role = -1; // 角色
5     audio = false; // 是否打开音频
6     video = false; // 是否打开视频
7     type = 0; // JCCRoomParticipantType
8     userId = ''; // 用户Id
9     userUri = string; // 用户Uri, 主要用于服务器交互使用, 同步移动端
10    displayName = string; // 昵称
11    virtual = false; // 是否虚拟成员
12    streamUrl = ''; // 推拉流地址
13    changeParam = {}; // JCCRoomParticipantChangeParam
14 }
```

7.6. 房间成员属性变化参数

JCCRoomParticipantChangeParam


```
1 // 房间成员属性变化参数
2 class JCCRoomParticipantChangeParam {
3
4     /**
5      * 上传音频是否变化
6      * - true: 变化
7      * - false: 没变化
8      */
9     audio = false;
10
11     /**
12      * 上传视频是否变化
13      * - true: 变化
14      * - false: 没变化
15      */
16     video = false;
17
18     /**
19      * 视频请求大小是否变化
20      * - true: 变化
21      * - false: 没变化
22      */
23     videoSize = false;
24
25     /**
26      * 成员类型是否变化
27      * - true: 变化
28      * - false: 没变化
29      */
30     type = false;
31
32     /**
33      * 成员网络状态是否变化
34      * - true: 变化
35      * - false: 没变化
36      */
37     netStatus = false;
38 }
```

8. 枚举值说明

终端类型 JCCCallUserType

```
1 // 终端类型
2 export const JCCallUserType = {
3     /** 普通APP用户 */
4     APP: 0,
5     /** SIP用户 */
6     SIP: 1,
7     /** H5 */
8     H5: 2,
9     /** WeChat */
10    WECHAT: 3,
11 }
```

登录状态 JCCClientState

```
1  /**
2   * 登录状态枚举
3   */
4  export const JCCClientState = {
5   /**
6    * 未初始化
7    */
8    NOT_INIT : 0,
9
10   /**
11    * 未登录
12    */
13    IDLE : ClientState.IDLE,
14
15   /**
16    * 登录中
17    */
18    LOGINING : ClientState.LOGINING,
19
20   /**
21    * 登录成功
22    */
23    LOGINED : ClientState.LOGINED,
24
25   /**
26    * 登出中
27    */
28    LOGOUTING : ClientState.LOGOUTING,
29  }
```

通话结束的原因 JCCCallTermReason

```
1  /**
2   * 通话结束的原因
3   */
4  export const JCCCallTermReason = {
5   /**
6    * 无错误
7    */
8   NONE : 0,
9   /**
10    * 本端结束通话
11    */
12   QUIT : CallTermReason.QUIT,
13   /**
14    * 对端结束通话
15    */
16   OVER : CallTermReason.OVER,
17   /**
18    * 网络异常导致本端结束通话
19    */
20   OFFLINE : CallTermReason.OFFLINE,
21   /**
22    * 呼叫来电取消
23    */
24   INCOMING_CANCEL : CallTermReason.INCOMING_CANCEL,
25   /**
26    * 呼叫来电超时
27    */
28   INCOMING_TIMEOUT : CallTermReason.INCOMING_TIMEOUT,
29   /**
30    * 心跳超时
31    */
32   TIME_OUT : CallTermReason.TIME_OUT,
33   /**
34    * 会场人员上限, licence限制
35    */
36   JOIN_LICENCE_LIMIT : CallTermReason.JOIN_LICENCE_LIMIT,
37   /**
38    * 拒绝呼叫
39    */
40   REJECT : CallTermReason.REJECT,
41   /**
42    * 房间异常 (服务端下发)
43    */
44   ROOM_ERROR : CallTermReason.ROOM_ERROR,
45   /**
```

```

46     * 呼叫失败
47     */
48     CALL_FAIL : CallTermReason.CALL_FAIL,
49     /**
50     * 参数错误
51     */
52     INVALID_PARAM : CallTermReason.INVALID_PARAM,
53     /**
54     * 方法调用失败
55     */
56     CALL_FUNCTION_ERROR : CallTermReason.CALL_FUNCTION_ERROR,
57     /**
58     * 用户未登录
59     */
60     NOT_LOGIN : CallTermReason.NOT_LOGIN,
61     /**
62     * 其他原因
63     */
64     OTHER : CallTermReason.OTHER,
65 }

```

原因枚举 JCCReasonCode

```
1  /**
2   * 原因枚举
3   */
4  export const JCCReasonCode = {
5   /**
6    * 正常
7    */
8   NONE : 0,
9   /**
10    * 其他原因
11    */
12   OTHER : ReasonCode.OTHER,
13   /**
14    * 参数错误
15    */
16   INVALID_PARAM : ReasonCode.INVALID_PARAM,
17   /**
18    * 当前状态无法再次登录
19    */
20   CLI_STATE_CANNOT_LOGIN : ReasonCode.CLI_STATE_CANNOT_LOGIN,
21   /**
22    * 登录超时
23    */
24   CLI_TIMEOUT : ReasonCode.CLI_TIMEOUT,
25   /**
26    * 登录网络异常
27    */
28   CLI_NETWORK : ReasonCode.CLIENT_NETWORK_ERROR,
29   /**
30    * 接入房间网络超时
31    */
32   ROOM_TIMEOUT : ReasonCode.ROOM_TIMEOUT,
33   /**
34    * 接入房间网络异常
35    */
36   ROOM_NETWORK : ReasonCode.ROOM_NETWORK,
37   /**
38    * 被踢出房间
39    */
40   ROOM_KICKED : ReasonCode.ROOM_KICKED,
41   /**
42    * 接入房间掉线
43    */
44   ROOM_OFFLINE : ReasonCode.ROOM_OFFLINE,
45   /**
```

```

46      * 主动离开房间
47      */
48      ROOM_QUIT : ReasonCode.ROOM_QUIT,
49      /**
50      * 房间关闭
51      */
52      ROOM_OVER : ReasonCode.ROOM_OVER,
53      /**
54      * 房间成员已满
55      */
56      ROOM_FULL : ReasonCode.ROOM_FULL,
57      /**
58      * 房间密码无效
59      */
60      ROOM_INVALID_PASSWORD : ReasonCode.ROOM_INVALID_PASSWORD,
61      /**
62      * 该房间号的房间不存在
63      */
64      REASON_CONF_NUMBER_NOT_FOUND : ReasonCode.REASON_CONF_NUMBER_NOT_FOUND
65      ,
66      /**
67      * 服务器房间成员总数上限（移动端房间人数）
68      */
69      ROOM_APP_CONCURRENCY_FUL : ReasonCode.ROOM_APP_CONCURRENCY_FUL,
70      /**
71      * 服务器房间成员总数上限（总房间人数）
72      */
73      ROOM_ALL_CONCURRENCY_FUL : ReasonCode.ROOM_ALL_CONCURRENCY_FUL,
74      /**
75      * 房间异常（服务端下发）
76      */
77      ROOM_INTERNAL_ERROR : ReasonCode.ROOM_INTERNAL_ERROR,
78      /**
79      * 会场人员上限，licence限制
80      */
81      ROOM_JOIN_LICENCE_LIMIT : ReasonCode.ROOM_JOIN_LICENCE_LIMIT,
82      // 录制cd被释放
83      ROOM_AGENT_RELEASE : ReasonCode.ROOM_AGENT_RELEASE,
84  }

```

视频编码 JCCVideoEncodeType

```
▼ JCCVideoEncodeType JavaScript |
1  /**
2   * 视频编码
3   */
4  export const JCCVideoEncodeType = {
5    /**
6     * H264
7     */
8    H264 : 0,
9    /**
10     * AV1
11     */
12    AV1 : 1
13  }
```

音频编码 JCCAUDIOEncodeType

```
▼ JCCAUDIOEncodeType JavaScript |
1  /**
2   * 音频编码
3   */
4  export const JCCAUDIOEncodeType = {
5    /**
6     * OPUS
7     */
8    OPUS : 0,
9    /**
10     * PCMA
11     */
12    PCMA : 1,
13    /**
14     * PCMU
15     */
16    PCMU : 2
17  }
```