

【视频客服】Juphoon SDK for Web/H5 开发集成指南



开发指导手册（Web/H5）

宁波菊风系统软件有限公司

2022年1月

版权所有©宁波菊风系统软件有限公司 2021。保留一切权利。

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 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。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。

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 RTC SDK）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 RTC SDK 支持 iOS、Android、Windows、微信小程序、H5 等操作系统平台，可实现一次开发跨平台发布。满足App、Web、H5、微信小程序、PAD等不同移动设备的支持。

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Windows、Android、iOS、macOS、Web-OCX、H5-WebRTC、微信小程序等。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 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 Web/H5 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 Web/H5 端集成 SDK 的操作步骤，帮助您快速集成 SDK 并实现视频通话（访客侧）的基本功能。

前提条件

- 可以连接到互联网的 Windows 或 macOS 计算机。

- 计算机搭载 2.2 GHz Intel 第二代 i3/i5/i7 处理器或同等性能的其他处理器。
- 内置摄像头或外置 USB 摄像头。

集成说明

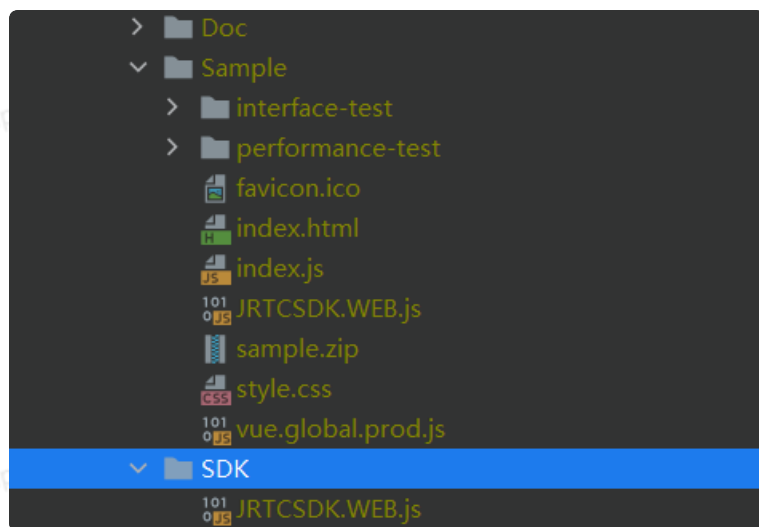
- 兼容性说明

下表列出目前 WebRTC SDK 对浏览器的支持情况：

此处为语雀内容卡片，点击链接查看：https://juphoon.yuque.com/video/products/loxs24/eofe0a?view=doc_embed

步骤一：获取 JC Web SDK 及 Sample 源码

SDK 包含以下文件，请确认后再进行开发。



步骤二：导入 JC Web SDK

将这些文件放入项目的开发路径下，确保项目可以引用到相关模块。SDK 可以通过html 标签引入。

```
1 import { JuphoonWebRTCGuest } from 'JRTCSDK.WEB.js';
2 /* or */
3 const JRTCSDK = require('JRTCSDK.WEB.js');
4 const JuphoonWebRTCGuest = JRTCSDK.JuphoonWebRTCGuest;
```

四、实现视频通话（访客侧）

本文档为您展示 实现视频通话（访客侧）的相关步骤，帮助您在远程银行和视频客服的场景下实现智能排队、屏幕共享、全景录像、访客管理的相关能力。

当您成功初始化 JRTC SDK 之后，您可以简单体验本产品的基本业务流程。

前提条件

请确认您已完成以下操作：

- 已获取 App Key。

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。

tip

请扫描下方二维码联系 Juphoon 市场售前工程师获取 AppKey。



- 集成 SDK (Java) 。

:::tip

如需集成指导，请扫描上方二维码联系 Juphoon 市场售前工程师获取相关咨询。

...

实现视频通话

本文主要展示如何初始化 Juphoon 的 Web SDK 以及登录视频平台。

1.初始化SDK

在使用业务接口前，需要对 JC Web SDK 进行初始化操作。

- 步骤一：设置环境

使用 `JuphoonWebRTCGuest.setConfig` 接口设置环境。

```
1 JuphoonWebRTCGuest.setConfig({address: 'https://xxxx:xxxx'});
```

- 步骤二：创建 Client

通过 `JuphoonWebRTCGuest.createGuestClient` 接口创建client对象。

`JuphoonWebRTCGuest.createGuestClient` 需要的参数如下所示：

AppKey

Token

连接 WebSocket 时用于进行验证的令牌，获取方式如下：

1. 通过

此处为语雀内容卡片，点击链接查看：

https://juphoon.yuque.com/videoproducts/hnvgf8/qiokmt?view=doc_embed

服务获取token, 获取方式见

此处为语雀内容卡片，点击链接查看：

https://juphoon.yuque.com/videoproducts/hnvgf8/ld946q?view=doc_embed&inner=Vc6yD

，需要设置校验类型为 `auth`。



JavaScript

复制代码

```
1 JuphoonWebRTCGuest.setConfig({webSocketAuthType: 'auth'});
```



JavaScript

复制代码

```
1 const client = JuphoonWebRTCGuest.createGuestClient(AppKey, token);
```

2. 登录

使用 Juphoon 视频的能力前，需要先登录到 Juphoon 视频平台。

在呼叫之前需要先进行登录，`login` 接口中可传入用户名(默认随机生成)密码等登录参数用于校验。

登录时可通过`accountName`指定用户名，未指定将自动生成随机的16位用户名。

```
1 client.login({
2   accountName: ''
3 })
4 .then(() => {
5   // 登录成功
6 })
7 .catch(reason => {
8   // 登录失败
9   alert(reason);
10 });
```

3.发起视频呼叫

创建成功后可以通过 `call` 发起呼叫，来呼叫自己要做的业务。

呼叫需指定业务号，业务号可从下一步骤中获得，在发起呼叫时可根据用户需求决定是否携带额外信息。

```
1 /**
2  * 发起呼叫
3  * @param number - 业务号
4  * @param callParam - 呼叫参数
5  */
6 call(number: string, callParam?: GuestCallParam): Promise<void>;
7
8 client.call(telNumber, {
9   extraParam: '',
10  displayName: 'nickname',
11  securityType: 0
12 })
13 .then(() => {
14   // 呼叫成功
15 })
16 .catch(() => {
17   // 呼叫失败
18 });
```


:::tip

:::

- 点对点呼叫

也可以通过 [oneToOneCall](#) 发起点对点呼叫

点对点呼叫可直接指定坐席的用户名进行呼叫。

```
1  /**
2   * 发起点对点呼叫
3   * @param number - 坐席用户名
4   * @param callParam - 呼叫参数
5   */
6  oneToOneCall(agentUri: string, callParam?: GuestCallParam): Promise<any>;
7
8  client.oneToOneCall(agentUri, {
9    extraParam: '',
10   displayName: 'nickname',
11   securityType: 0
12 })
13 .then(() => {
14 })
```

4 通话状态改变通知

当用户成功加入会议、坐席接通过话、用户自身状态改变、会议成员离开会议、通话结束，都会通过事件进行消息上报。

EventType	字面量	描述
GuestCallState.CALLING	calling	呼叫中
GuestCallState.WAITING	waiting	等待坐席接听
GuestCallState.TALKING	talking	通话中
GuestCallState.TERMED	termed	通话结束

```

1  import { GuestEventType, GuestCallState } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { GuestEventType, GuestCallState } = require('JRTCSDK.WEB.js');
4
5  client.addListener(GuestEventType.CALL_STATE_CHANGE, (ev) => {
6      const {callState, oldCallState} = ev.message.data;
7
8      switch (callState) {
9          case GuestCallState.CALLING:
10             break;
11          case GuestCallState.TERMED:
12             break;
13          case GuestCallState.TALKING:
14             break;
15          case GuestCallState.WAITING:
16             break;
17      }
18  });

```

5.创建本地视频画面

创建本地视频画面需要在加入会议后调用

`startCameraVideo` 接口会打开摄像头和麦克风并返回一个 `WebRTCStream` 对象, 可以通过该对象将画面渲染到页面上。渲染模式可指定两种渲染模式:

1. `IRTCStreamRenderType.CONTAIN`, 持原有尺寸比例。但部分内容可能被剪切。
2. `IRTCStreamRenderType.COVER`, 保持原有尺寸比例, 内容被缩放。

`WebRTCStream.start(domId, document)` 该方法会将本地/远端画面渲染到传入的 id 对应的 DOM 元素中。

```

1  <div id="local-video" style="width: 400px; height: 300px;"></div>

```

```

1  import { IRTCStreamRenderType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { IRTCStreamRenderType } = require('JRTCSDK.WEB.js');
4
5  client.startCameraVideo(IRTCStreamRenderType.CONTAIN)
6    // 将画面渲染到id为 local-video 的 dom元素中
7    .then(stream => stream.start('local-video'))
8    .then(() => {
9      // 打开摄像头成功
10    })
11    .catch((reason) => {
12      // 打开摄像头失败
13      console.error(reason);
14    });

```

6. 创建远端视频画面

创建远端视频画面需要在加入会议后调用

`startVideo` 接口会打开摄像头并返回一个 `WebRTCStream` 对象, 可以通过该对象将画面渲染到页面上。渲染方式同上。

Autoplay问题

当我们获取到了远端的音视频数据并开始播放时, 有可能会受到浏览器的[自动播放策略](#)限制。

自动播放限制 是指如果以下任何一项未发生则媒体不允许播放:

1. 音频被静音或其音量设置为0
2. 用户已经与站点进行了交互 (通过单击, 按键等)
3. 如果该网站已被列入白名单; 如果浏览器确定用户经常与媒体互动, 则可能会自动发生这种情况, 也可能通过首选项或其他用户界面功能手动发生这种情况
4. 已通过自动播放功能策略向 `<iframe>` 授予自动播放权限

7. 挂断

访客可以在呼叫等待或者通话中通过调用 `term` 方法主动挂断. 挂断后会触发 `CALL_STATE_CHANGE` 事件.

```
1  import { GuestEventType, GuestCallState } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { GuestEventType, GuestCallState } = require('JRTCSDK.WEB.js');
4
5  client.term()
6  .then(() => {
7    // 挂断成功
8  })
9  .catch(reason => {
10    // 挂断失败
11    alert(reason);
12  });
13  client.addEventListener(GuestEventType.CALL_STATE_CHANGE, (ev) => {
14    switch (ev.message.state) {
15      case GuestCallState.TERMED:
16        // todo
17        break;
18    }
19  });
```

8.销毁本地和远端视频画面

挂断成功后会自动销毁本地/远端的视频画面。

9.登出

访客可以登出视频平台，与平台断开连接。可以调用 `logout` 登出。

```
1  client.logout();
```

10.销毁SDK

访客可以调用 [destroy](#) 销毁创建的 client 并释放资源。

```
1 client.destroy()
```

五、消息通道

5.1 随路参数

调用发起呼叫的接口 [call](#) 时携带额外信息。

呼叫时通过 [callParam.extraParam](#) 把随路参数传给坐席。

```
1 client.call('10086', {  
2   extraParam: '随路参数',  
3 })  
4 .then(() => {  
5   });
```

5.2 文字通道

给通话中成员发送消息需要调用发送文字消息的接口 [SendMessage](#)。

```

1  /**
2   * 频道中发送消息, 当 toUserId 不为 null 时, content 不能大于 4k
3   * 此接口调用成功后, 消息接收方会收到 {@link
ConferenceEventType.MESSAGE_RECEIVE} 事件, 通过此事件可以获取消息的文本类型和文
本内容
4   *
5   * @param type 消息类型
6   * @param content 消息内容
7   * @param toUserId 接收方成员的userid, 值为null发送给所有人
8   */
9  sendMessage(type: string, content: string | Record<string, unknown>,
toUserId?: string): Promise<any>;

```

```

1  client.sendMessage('TextMessage', 'hello', '[xxx]')
2  .then(() => {
3    // 发送成功
4  })
5  .catch(reason => {
6    // 发送失败
7    alert(reason);
8  });

```

client收到会议消息时会发出 `ConferenceEventType.MESSAGE_RECEIVE` 事件, 事件对象的 `message` 里有对应的消息信息。

key	类型	描述
type	string	消息类型
content	string	消息内容
fromUserId	string	消息发送者

```

1  import { ConferenceEventType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { ConferenceEventType } = require('JRTCSDK.WEB.js');
4
5  client.addListener(ConferenceEventType.MESSAGE_RECEIVE, (ev) => {
6      const {type, content, fromUserId} = ev.message;
7  });

```

5.3 在线消息

- 发送在线消息

在 client 登录后即可调用

参数说明：

```

1  /**
2   * 发送在线消息
3   *
4   * @param toUserId 接收者的uri
5   * @param content 消息内容，如传入json对象会通过{@link JSON}字符串化
6   */
7  sendOnlineMessage(toUserId: string, content: string | Record<string,
  unknown>) • Promise<void> •

```

```

1  /**
2   * 获取发送在线消息的结果(方案一)
3   */
4  client.sendOnlineMessage('toUserId', 'content')
5  .then(() => {
6      // 发送成功
7  })
8  .catch((reason) => {
9      // 发送失败
10     console.error(reason);
11 });

```

- 接收在线消息

属性名	类型	描述
fromUserId	string	发送者的uri
content	string	发送的消息内容

```

1  import { ConferenceEventType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { ConferenceEventType } = require('JRTCSDK.WEB.js');
4
5  client.addEventListener(ConferenceEventType.ONLINE_MESSAGE_RECEIVE, (ev)
6  => {
7      const {content, fromUserId} = ev.message;
8  });

```

六、智能排队

6.1 排队人数

坐席接听前排队人数变化会通过 `GuestEventType.CALL_QUEUE_COUNT` 事件进行上报。


```

1  import { GuestEventType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { GuestEventType } = require('JRTCSDK.WEB.js');
4
5  // 添加对客户端实例 `GuestEventType.CALL_QUEUE_COUNT` 事件的监听回调，获取排队
   人数信息
6  client.addEventListener(GuestEventType.CALL_QUEUE_COUNT, (ev) => {
7      const {count, time} = ev.message;
8  });

```

6.2 加急

访客呼叫时调用申请加急的接口优先进行通话：

```

1  /**
2   * 排队加急
3   */
4  client.requestUrgent()
5  .then(() => {
6      alert('加急成功');
7  })
8  .catch(() => {
9      alert('加急失败');
10 });

```

6.3 获取业务列表

client 调用 `listAllGroups` 接口请求业务列表，该接口无需登录即可使用。

业务列表项说明如下：

Name	Type	Description
callNumber	string	业务号
group	string	组号
memo	string	业务名称



JavaScript | 复制代码

```

1  client.listAllGroups()
2  .then((groupList) => {
3      // 获取成功
4  })
5  .catch(reason => {
6      // 获取失败
7      alert(reason);
8  });

```

6.4 获取所有通话成员

通过 `participantMap` 属性获取所有参会者成。

七、音频管理

本文将介绍访客用于音频中的相关功能。

7.1 控制音频流上传

通过 `enableUploadAudioStream` / `disableUploadAudioStream` 开启关闭发送本地视频流。

```
1  /**
2   * 发送本地音频流.
3   */
4  enableUploadAudioStream(): Promise<void>;
5  /**
6   * 关闭本地音频流.
7   */
8  disableUploadAudioStream(): Promise<void>;
```

八、视频管理

本文将介绍访客用于视频中的相关功能。

8.1 控制视频流上传

通过 [enableUploadVideoStream](#) / [disableUploadVideoStream](#) 开启关闭发送本地视频流。

```
1  /**
2   * 发送本地视频流
3   */
4  enableUploadVideoStream(): Promise<any>;
5
6  /**
7   * 发送本地视频流
8   */
9  disableUploadVideoStream(): Promise<any>;
```

8.2 切换摄像头

调用 [switchCamera](#) 接口可以根据传入的参数切换使用的摄像头。

[IMediaDevicesConstraints](#) 支持以 [facingMode](#) 的形式指定摄像头。

StreamCameraConstraints 参数说明:

参数名	类型	必填	描述
facingMode	'user' 'environment'	否	user: 前置摄像头environment: 后置摄像头

示例如下:

```
1 // 指定后置摄像头
2 client.switchCamera({facingMode: 'environment'})
3 .then(() => {
4     // 切换成功
5 })
6 .catch((reason) => {
7     // 切换失败
8 });
```

8.3 设置请求分辨率

通过 `requestVideo` 在会议中修改对端画面的分辨率，这个参数结合访客 SVC 来使用可以实现通话中切换设置的分辨率。

```

1  /**
2   * 请求用户的视频流
3   * 当 pictureSize 为 PictureSizeNone 标识关闭请求
4   *
5   * @param participant 用户对象
6   * @param pictureSize 视频请求尺寸类型
7   * @see ConferencePictureSize
8   */
9  requestVideo(participant: Participant, pictureSize: PictureSize):
    Promise<void>;
10

```

在会议中修改本地端画面的分辨率，可以实现通话中切换设置的分辨率。

```

1  import { PictureSize } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { PictureSize } = require('JRTCSDK.WEB.js');
4
5  client.setRequestVideo(PictureSize.MIN)
6  .catch(reason => {
7    alert('设置请求分辨率失败\n' + JSON.stringify(reason));
8  });

```

SVC设置说明

根据实际订阅需求和网络状况动态调整视频发送分辨率是JSM会议的特性之一，SVC可用于设置会议视频的每一层编码分辨率。该参数在会议创建时设置，且全局统一。

具体使用详见文档：<https://juphoon.yuque.com/dptof9/uudqv0/bttcx0>

可在发起呼叫前，通过呼叫参数 `GuestCallParam` 的 `svcResolution` 属性进行设置。

8.4 视频渲染管理

远端和本地媒体流都是一个 [WebRTCStream](#) 对象, 可以通过 [startCameraVideo](#) / [startVideo](#) 接口获取本地/远端的媒体流对象:

```
1  /**
2   * 开启发送/接收本地音频流
3   */
4   enableAudio(): Promise<void>;
5
6   /**
7   * 关闭发送/接收本地音频流
8   */
9   disableAudio(): Promise<void>;
10
11  /**
12   * 开启发送/接收本地视频流
13   */
14   enableVideo(): Promise<void>;
15
16  /**
17   * 关闭发送/接收本地视频流
18   */
19   disableVideo(): Promise<void>;
```

九、屏幕共享

目前只能够接收屏幕共享, 不支持发送屏幕共享.

可以通过会议属性变化事件 [ConferenceEventType.CONFERENCE_PROPERTY_CHANGE](#) 监听是否有成员开启了屏幕共享, 屏幕共享开启后接受的座席画面会自动变为屏幕共享. `screenUserId` 表示发起屏幕共享成员的 `userId`, `screenRenderId` 表示共享的屏幕视频流的标识, 用以传入 [startScreenVideo](#) 获取屏幕共享的媒体流对象.

```

1  import { ConferenceEventType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { ConferenceEventType } = require('JRTCSDK.WEB.js');
4
5  let screenStream;
6  client.addListener(ConferenceEventType.CONFERENCE_PROPERTY_CHANGE,
7    (ev) => { //CONFERENCE_PROPERTY_CHANGE: 是否开启屏幕共享的时候
8      // changeParam表示某个属性变化了, 并不表示该属性的实际值.
9      const {participant, changeParam} = ev.message.data;
10
11      if (changeParam.screenShare) {
12        // 判断是否正在屏幕共享
13        if (client.screenUserId && client.screenRenderId) {
14          // 获取屏幕共享的视频流.
15          client.startScreenVideo(client.screenRenderId)
16            .then((stream) => screenStream = stream)
17            .then(() => screenStream.start('screen-stream', document));
18        } else {
19          screenStream?.stop();
20          screenStream = undefined;
21        }
22      }
23    });

```

十、加密传输

为了建立安全通道, 就是通信双方建立连接通道后, 在这个通道中传递的信息不可被第三方窃取, 或者即使窃取后, 也不能识别信息内容。但是仅通信双方建立安全通道是不够的, 还需要进行合法性确认。

通常终端的合法性是基于用户名密码实现的, 即服务器通过客户端提交的用户名和密码来进行鉴权, 确认此用户的合法性。

在建立安全通道以及身份合法性验证后, 后续大量的信息交互需要高效率的加密和解密, 通常我们会使用对称加密方式进行处理。

国密加密

国密SM4分组密码算法是我国自主设计的分组对称密码算法，用于实现数据的加密/解密运算，以保证数据和信息的机密性。

要保证一个对称密码算法的安全性的基本条件是其具备足够的密钥长度，SM4算法与AES算法具有相同的密钥长度分组长度128比特，因此在安全性上高于3DES算法。

在呼叫时设置加密方式，需传入 `securityType` 参数：

```
1  import { ConferenceSecurityType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { ConferenceSecurityType } = require('JRTCSDK.WEB.js');
4
5  client.call(number, {securityType: ConferenceSecurityType.DISABLE});
```

十一、第三方管理

本文将介绍通话过程中第三方管理的功能：访客无邀请和转接的接口，但是能收到第三方加入离开的事件。

成员状态变化

通话中成员加入，状态变化或离开都会触发对应的事件。

成员加入会触发 `ConferenceEventType.CONFERENCE_PARTICIPANT_JOIN` 事件。


```

1  import { ConferenceEventType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { ConferenceEventType } = require('JRTCSDK.WEB.js');
4
5  client.addEventListener(ConferenceEventType.CONFERENCE_PARTICIPANT_JOIN,
    (ev) => {
6      const {participant} = ev.message;
7  });

```

成员离开时会触发 `CONFERENCE_PARTICIPANT_LEFT` 事件，该事件会携带 `participant` 属性表示离开的成员对象。

```

1  import { ConferenceEventType } from 'JRTCSDK.WEB.js';
2  /* or */
3  const { ConferenceEventType } = require('JRTCSDK.WEB.js');
4
5  client.addEventListener(ConferenceEventType.CONFERENCE_PARTICIPANT_LEFT,
    (ev) => {
6      const {participant} = ev.message;
7  });

```

`participant` 对象有下列方法可供判断成员类型

```

1  /**
2   * 该成员是否为坐席
3   */
4  isAgent(): boolean;
5
6  /**
7   * 该成员是否为访客
8   */
9  isGuest(): boolean;
10

```

