

【多方视频】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

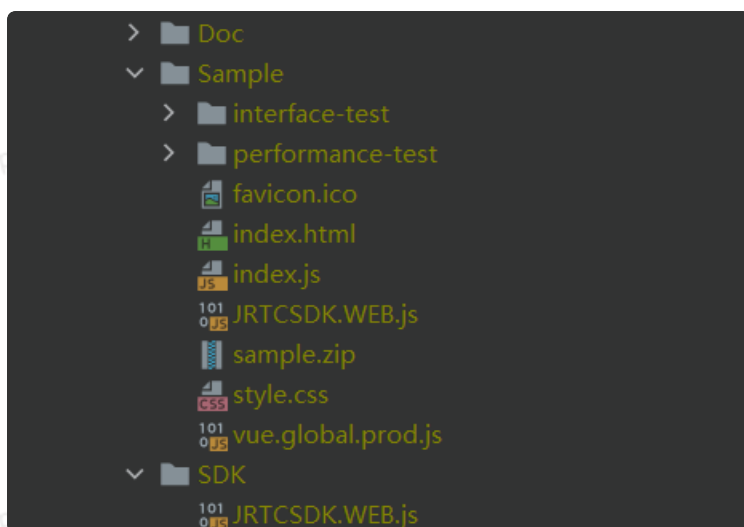
本文为您介绍 WebRTC 端集成 SDK 的操作步骤，帮助您快速集成 SDK 并实现视频双录的基本功能。

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。

SDK 集成

SDK文件说明



步骤一：下载 SDK

选择 `SDK` 目录下的文件 到您工程目录下的 `libs` 或其他文件夹中。

步骤二：在您的工程中引入 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 ES6 的 `import` 关键字或使用 `require` 引入。

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

title: 实现视频双录

实现视频通话

本文档为您展示通过 SDK 实现多方视频通话的相关步骤，帮助您在多人视频通话场景下实现创建会议、邀请新成员加入、结束/离开会议的相关能力。

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。

- 集成 SDK (WebRTC/H5) 。

功能实现

1.初始化SDK

1. 设置环境

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

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

2. 创建Client

通过 `JuphoonWebRTCConference.createConferenceClient` 接口创建 client 对象。

需要的参数如下所示：

AppKey

用户联系 Juphoon 市场售前工程师获取 AppKey

Token

连接 WebSocket 时用于进行验证的令牌，获取方式如下：

通过Authsever服务获取token, 获取方式见服务端 RESTful API, 需要设置校验类型为 `auth` .

```
1 JuphoonWebRTCConference.setConfig({websocketAuthType: 'auth'});
2 const client = JuphoonWebRTCConference.createConferenceClient(appKey, token);
```

2.登录

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. 加入会议

调用 [join](#) 接口加入/创建会议

通过 [ConferenceJoinParam](#) 可在加入会议前设置会议最大人数, 昵称, 参会密码, 是否开启录制, 在会议中的角色等。

```
1 client.join('confNumber', {
2   // todo 入会参数
3 })
4 .then(() => {
5   // 入会成功
6 })
7 .catch((reason) => {
8   // 入会失败
9 });
```

ConferenceJoinParam 参数说明

属性名	类型	描述
displayName	string	昵称
conferencePassword	string (数字/字母)	会议密码
enableRecord	boolean	是否开启服务端录制注：开启服务端录制后需要在收到 <code>ConferenceEventType.RECORD_DELIVERY_JOIN</code> 事件后才能开始录制
role	ConferenceRole	

4.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。

`RTCStream.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>
2 import { IRTCStreamRenderType } from 'JRTCSDK.WEB.js';
3 /* or */
4 const { IRTCStreamRenderType } = require('JRTCSDK.WEB.js');
5
6 client.startCameraVideo(IRTCStreamRenderType.CONTAIN)
7   // 将画面渲染到id为 local-video 的 dom元素中
8   .then(stream => stream.start('local-video'))
9   .then(() => {
10     // 打开摄像头成功
11   })
12   .catch((reason) => {
13     // 打开摄像头失败
14     console.error(reason);
15   });
```

5.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

6.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。

```
JavaScript | 复制代码
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。

```
JavaScript | 复制代码
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;
```

7. 离开会议

调用 `leave` 以离开会议。

```
1  client.leave();
2  .then(() => {
3    // 离开成功
4  })
5  .catch((reason) => {
6    // 离开失败
7  });
```

8. 结束会议

如果想结束会议，可以调用 `stop` 接口，此时所有成员都将被退出。

```
1 client.stop()
2 .then(() => {
3     // 结束会议成功
4 })
5 .catch((reason) => {
6     // 结束会议失败
7 });
```

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();
```

音频管理

本文将介绍视频双录中的音频管理。

开启关闭远端音频(扬声器)

通过 [enableAudioOutput](#) / [disableAudioOutput](#) 开启关闭远端音频(扬声器)。

```
1  /**
2   * 订阅会议音频流，关闭订阅将无法听到会议中其他成员的声音
3   */
4  enableAudioOutput(): Promise<void>;
5
6  /**
7   * 取消订阅会议音频流
8   */
9  disableAudioOutput(): Promise<void>;
```

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>;
```

视频管理

本文将介绍视频双录中的视频管理相关功能。

控制视频流上传

通过 `enableUploadVideoStream` / `disableUploadVideoStream` 开启关闭发送本地视频流。

JavaScript | 复制代码

```
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>;
```

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
deviceId	string	否	代表设备的唯一id，可通过 <code>JuphoonWebRTCConference.getMediaDevices()</code> 获取。
facingMode	'user' 'environment'	否	user: 前置摄像头environment: 后置摄像头

示例如下：

```
1 client.switchCamera({deviceId: device.deviceId})
2 .then(() => {
3     // 切换成功
4 })
5 .catch((reason) => {
6     // 切换失败
7 });
8 // or
9 // 指定后置摄像头
10 client.switchCamera({facingMode: 'environment'})
11 .then(() => {
12     // 切换成功
13 })
14 .catch((reason) => {
15     // 切换失败
16 });
```

屏幕共享

目前只能接收屏幕共享, 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 `WebRTCStream`.

```

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

```

音视频录制

在开展在线理财、开户、面签等业务时，应国家监管要求，必须提供录音录像服务，形成交易记录的视频，存档备查。

Juphoon RTC SDK 在音视频通话过程中，支持全程进行实时的服务器录制或本地录制，录制的场景画面覆盖坐席画面和所有终端类型的访客画面，按需可支持多摄像头、多屏幕合成录制，满足用户记录业务办理全过程录制的需求。

服务器音视频录制

远程录制功能需要在加入或开始会议的时候将参数 `enableRecord` 设为 `true`。之后会收到 `ConferenceEventType.RECORD_DELIVERY_JOIN` 事件后才能调用开始录制接口。

使用 `startRemoteRecord` 方法开始录制。传入的参数如下。

参数名	类型	必填	描述
<code>recordResolution</code>	MediaRemoteRecordPictureSize	否	录制分辨率
<code>frameRate</code>	number	否	录制帧率
<code>mergeMode</code>	VideoLayoutMergeMode	否	媒体录制视频合并模式
<code>videoWidth</code>	number	否	视频宽度
<code>videoHeight</code>	number	否	视频高度
<code>bitrate</code>	number	否	码率
<code>storage</code>	MediaRemoteRecordStorageOptions	是	存储参数
<code>fileName</code>	string	否	自定义文件名 (Juphoon 文件服务)
<code>remoteFileName</code>	string	否	自定义文件名(第三方)

MediaRemoteRecordStorageOptions

参数名	类型	必填	描述
<code>protocol</code>	string	是	存储协议

```
1
2 client.startRemoteRecord({
3   storage: {protocol: 's3'}
4 })
5 .then(() => {
6   // 开启录制成功
7 })
8 .catch((reason) => {
9   // 开启录制失败
10  });
```

频道管理

本文将介绍视频双录中频道管理的相关功能。

查询频道

如需查询频道相关信息，例如频道名称、是否存在、成员名、成员数，可以调用 `query` 接口进行查询操作。

```
1 query(channelId: string): Promise<any>;
2 this.client.query(channelId)
3 .then(info => {
4   // 查询频道号
5   const number = info.number;
6   // 频道人数
7   const clientCount = info.queryInfo.clientCount;
8   // 成员列表
9   const members = info.queryInfo.members;
10  })
11 .catch(reason => {
12   // 查询失败
13  });
```

在线消息

- 发送在线消息

在 client 登录后即可调用

参数说明：

JavaScript | 复制代码

```
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>;
```

JavaScript | 复制代码

```
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.addListener(ConferenceEventType.ONLINE_MESSAGE_RECEIVE, (ev)
   => {
6    const {content, fromUserId} = ev.message;
7  });
```