Agora Web SDK

参考手册

1.2 版

support@agora.io

 2016 年 3 月

目录

Agora Web SDK 简介

Agora 通信即服务（Agora Communications as a Service, CaaS）运用 Agora 全球网络（Agora Global Network）为客户提供基于互联网的音视频通信，在实时通信和移动对移动通信方面进行特别优化，确保满意的用户体验质量。Agora CaaS 致力于解决移动设备的用户体验问题、3G/4G/Wi-Fi 网络性能各异、全球网络瓶颈等一系列问题，为用户带来优质的通信体验。

Agora CaaS 包含用于移动设备的 Agora Native SDK，专为 iOS 和安卓智能手机研发，其通信基于 Agora Global Network，并针对不同移动设备平台进行优化。Agora Native SDK 也有 Windows 版本，并即将推出 Mac 版本。Native SDK 在程序生成时直接与应用程序建立连接。在下文的使用入门一节中将为您阐述如何使用 Agora Native SDK 组件为 Windows 平台搭建工程环境。

本文档所述的 **Agora Web SDK** 是 Agora CaaS 针对网页通信专门开发的一款 SDK，为任何支持标准 WebRTC（网络实时通信）的网页浏览器提供 Agora Global Network 通信支持，而无需额外下载或安装插件。

Agora Web SDK 是通过 HTML 网页加载的 JavaScript 库，同时它还提供一些简单易用的上层 JavaScript API 函数和方法，让用户在 Agora Global Network 上建立音视频通话。Agora Web SDK 库在网页浏览器中使用原语 WebRTC APIs 建立连接和进行音视频控制，同时也为开发者提供更基本的上层接口。

Agora Web SDK 可采用 JavaScript 编程实现以下功能：

- 加入或离开 Agora 会话（会话以频道名称为标识），可实现全球用户多方会话。
- 设置多种音视频参数以达到通话最佳效果。Agora SDK 的大部分操作均自动设置为默认值，因此即使未给参数赋值也可运行。
- 管理音视频流，设置静音和显示，控制页面 HTML 框架内的音视频显示区域。
- 同步支持多方音视频流，支持多方会话程序。

Agora Web SDK 提供 3 个 JavaScript 接口类实现以下功能：

- 单个 **Client** 对象用于建立和管理通话。
- 多个 **Stream** 对象用于管理不同的音视频流。
- 顶层 **AgoraRTC** 对象用于创建相应的 **Client** 和 **Stream** 对象。

Agora Web SDK 入门指南

本节主要阐述如何使用 Agora Web SDK，包括产品兼容性和相关文档介绍、支持的网页浏览器类型、产品组件、开发和部署流程、如何运行示例程序、动态密钥介绍，以及已知问题和局限性等。

兼容性及相关文档

Agora Web SDK v1.2 版与 Agora Native SDK v1.2 兼容，支持相应的音视频功能。Agora Native SDK 用于在 Windows、安卓系统和 iOS 系统上创建应用程序，而 Web SDK 则用于在支持 WebRTC 的浏览器上使用 browser-app。

如您需要同时开发 native 版和网页浏览器版的程序，请参考以下文档：

- Agora Native SDK for iOS 参考手册 - v1.2 或更高版本
- Agora Native SDK for Android 参考手册- v1.2 或更高版本
- Agora Native SDK for Windows 参考手册- v1.2 或更高版本

本参考手册以及各个平台的完整版 SDK 均可在以下网址下载：www.agora.com/developer.

网页浏览器支持

运行 Agora Web SDK 需要支持 WebRTC 的网页浏览器，下表中列出的浏览器均已经过测试和验证，可在不同操作平台上运行。**注**：Microsoft Internet Explorer 和 Apple Safari 不支持 WebRTC，因此也不支持 Agora Web SDK。限于 Apple 平台的局限性，Apple iOS 上目前没有支持 WebRTC 的浏览器，因此需要依赖 native 应用程序来实现嵌入式的实时通信。

自 Chrome v47 和 Opera v34 之后的新版浏览器都因安全性考虑而不允许在 HTTP domains 上使用 WebRTC。推荐开发者也跟 Google 一样，使用 HTTPS domains 运行 WebRTC 和 Agora Web SDK 程序。

	Chrome v42-46	**Chrome v47**	**Firefox v42**	**Opera v34**	**QQ v9.0**	**360 v8.5**	**Baidu v7.6**	**Sougou v6.0**
HTTP	Y	N	Y	N	Y	Y	Y	Y
HTTPS	Y	Y	Y	Y	Y	Y	Y	Y

注：所有以上列出的浏览器均已经过测试和验证。在此之前的版本有可能也支持，但并未经过 Agora.io 测试验证。

产品组件

Agora Web SDK 工具包中包含下列文件，可以很方便地在您的开发系统和网页服务器上打开并部署。

组件	描述
./doc	Agora Web SDK 文档: Agora_Web_SDK_Release_Notes_v1_EN.pdf Agora_Web_SDK_API_Reference_v1_EN.pdf

组件	描述
./client/	Web 版示例程序。
./server	Web 版服务器端示例代码和生成动态密钥所需库。

获取 Vendor Key

使用 API 访问 Agora Global Network 时必须用到 **Vendor key** 和 **sign key**，这两个秘钥可通过登录 Agora.io 注册或联系 support@agora.io 获取，注册后可在后台 Dashboard 页面可看到相应的秘钥。Vendor key 可以单独使用，也可与 sign key 一起使用，它们在网页服务器上共同生成一个新的 **dynamic key（动态密钥）**，以提高安全性。关于动态密钥的详细信息见下文相关章节，同时示例程序中也有生成动态密钥的演示。

如何获取 Vendor Key:

a) 登录 cn.agora.io 注册创建新用户，如下图所示：

图 1

b) 注册完成后，您将在注册完成页面 http://dashboard.agora.io/ 看到自己的 Vendor key，如下图所示，并在注册邮箱收到一封包含 Vendor key 的邮件。

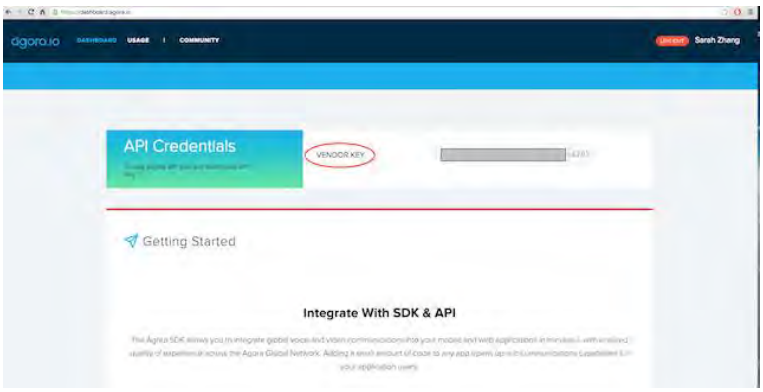

图 2

c) 获取 vendor key 后，即可使用对应的 Agora.io 服务功能。

程序开发和部署

运行 Agora Web SDK 的程序是标准的 JavaScript 程序，需加载 Agora JS 库并访问 SDK 提供的 JS 延伸库。部署 JavaScript 库时只需遵照通常的 JavaScript 变量声明提升机制（JavaScript hoisting procedure）执行代码即可。

注：1）加载 Agora JS 库 AgoraRTCSDK-1.2.0.js, 可从 https://rtcsdk.agora.io/AgoraRTCSDK-1.2.0.js 或者 http://rtcsdk.agora.io/AgoraRTCSDK-1.2.0.js 地址下载。

2）所需 JS 延伸库如下 (见软件包中 /client/js/文件夹):

jquery.js – version >= v2.0.0

socket,io.js – version >= 1.3.6

adapter.js – version >= 0.2.5

运行示例程序

客户端:

运行网页示例程序须先安装本地网页服务器，如 Apache, NginX 或 Node.js。将**./client/**下的文件部署到网页服务器上，并启动 http/https 服务。在网页服务器上用浏览器打开示例程序页面（应采用上文“浏览器支持”一节中所列出的浏览器，推荐使用 Google Chrome）。

默认的示例程序只需要静态的 **vendor key**（在示例程序页面上输入即可）。如需使用**动态密钥**，请执行以下操作：

- 在 index.html 中取消标注为“for dynamic key”的注释
- 将‘ip:port’替换为您的 dynamic key 生成服务器 URL（见下）
- 按下述方法配置并启动秘钥生成服务器

服务器：

如需使用更为安全的**动态密钥，**可尝试以下代码。在实际投入使用时，开发者应将该方法应用到自己的服务器端程序上，并使用服务器上的现有的编程语言重新编程。本示例代码是 JavaScript 写成，需要标准的 Node.js 服务器。

- 在本地服务器或云主机上安装标准的 Node.js 服务器
- 在../server/nodejs/ 下运行‘npm install’
- 在../server/nodejs/DemoServer.js 下填写 VENDOR_KEY 和 SIGN_KEY 值
- 用‘node DemoServer.js’打开服务器

注：如果没有 vendor key 和 sign key，请联系 support@agora.io 获取。

使用动态密钥提高安全性

每一家使用 Agora SDK 的机构都将获取一个唯一的代码，即“vendor key”，用于识别该机构。在 Agora Global Network 上使用同一 vendor key 的用户之间的通信是完全独立的，不与使用其他 vendor key 的机构相交。所有统计数据、信息管理、账单资料等通过 vendor key 识别并确保独立保存。如果在同一家机构中须运行不同的应用程序并要保证它们之间相互隔离，则应使用不同的 vendor key；如果各个应用程序之间需要进行通信，则应采用同一个 vendor key。

在客户端调用 **init()**方法时将用到 Vendor key，用以在 Agora Global Network 上识别您的机构。然后在此机构内再调用 **join()**方法，用唯一的频道名称创建一对一的对话或多人的会议。使用 vendor key 的操作简单明了，有助于初级阶段的程序开发。

但是，如果有人非法获取了您的 vendor key, 他将可以在他的电脑程序上执行同样的操作，可以加入您的机构中的通话而费用将由您支付。为了防止这一侵权行为，并确保您的程序更为安全，您应当考虑在大规模的程序中使用动态密钥（**dynamic keys**）。

动态密钥是一个更为安全的方法，它在调用 Client init()时采用一个额外的动态密钥，而非 vendor key。动态密钥在每一次客户端调用 init()时，由程序的服务器端生成。在服务器端，频道名称（即会议室名称）会被加密形成动态密钥，加密由 vendor key、Agora.io 提供的额外的 Sign Key（任何客户均不可见）、时间戳以及一个随机数组成。这一动态密钥被传输至客户端程序中，在 joinChannel()调用时使用。当 Agora Global Network 接收到动态密钥信息，则可以识别该通话是来自于您的程序，因为只有您的客户端代码和 Agora.io 知道这个唯一的 Sign Key.

如果使用 **动态密钥**，调用 init()时会将频道名称加密至动态密钥中，在之后调用 join()时务必指定同一个频道名称。

注：在创建新频道时，Web SDK 与原生 SDK 的处理稍有不同，不同处在于前者动态密钥是传入 join 中。这样可以使 **vendor key** 和 **sign key** 都只保存在服务器上而无需传至客户端浏览器，因此秘钥对客户机是不可见的，进一步提高了安全性。

又注：示例程序仅用于诠释动态密钥的概念。在实际应用中，应确保将该方法应用在安全的服务器代码上，而不是在网络上公开秘钥生成服务。

如您需要在您的程序中采用动态密钥方案，请联系 support@agora.io ，以便获取 Sign Key 及其他相关信息。Agora.io 可提供加密操作的客户端范例代码，分别由 JavaScript for Node.js（包含在 Agora Web SDK 的示例代码中）、Java 和 C 语言写成。加密方法采用业界标准化的 HMAC/SHA1 加密方案（见 https://en.wikipedia.org/wiki/Hash-based_message_authentication_code），在 Node.js, PHP, Python, Ruby 等绝大多数通用的服务器端开发平台上均可获得所需库。

已知问题和局限性

1. Agora Web SDK 基于 WebRTC 技术运行，但 Microsoft Internet Explore 和 Apple Safari 等浏览器，以及 Apple iOS 等操作平台并不支持 WebRTC。关于支持的平台见上文“浏览器支持”一节。
2. 如果在客户端安装了高清摄像头，则 Agora Web SDK 支持最大为 1080p 分辨率的视频流，但取决于不同的摄像头设备，最大分辨率也会受到影响。
3. Mozilla Firefox 中不支持最大视频流量设置。
4. 在 Agora Native SDK 中提供的一些功能，如质量提示、测试服务、通话服务评分、录音和日志记录等，在 Agora Web SDK v1.2 中暂未包含。

Agora Web SDK - API 参考

Agora Web SDK 库包含以下三种接口类：

AgoraRTC	使用 AgoraRTC 对象创建客户端和音视频流对象。
Client	提供 AgoraRTC 核心功能的 web 客户端对象。
Stream	通话中的本地或远程音视频流。

1. AgoraRTC 接口方法

createClient()

创建并返回客户端对象。单次通话中仅需调用一次。

示例代码：

```
var client = AgoraRTC.createClient();
```

getDevices(callback)

This method enumerates platform camera/microphone devices.枚举系统中摄像头/麦克风设备。

参数名称	类型	描述
callback	函数	用以获取设备信息的回调函数。 例：callback(devices): devices[0].deviceId: 设备 ID 号码 devices[0].label:设备名称 devices[0].kind: ‘audioinput’（音频输入）, ‘videoinput’（视频输入）

示例代码：

```
AgoraRTC.getDevices (function(devices) {
  var dev_count = devices.length;
  var id = devices[0].deviceId;
});
```

createStream(spec)

创建并返回音视频流对象。

参数名称	类型	描述
spec	对象	该对象包含以下属性： streamID: 音视频流 ID，通常设置为 uid，可通过 `Client join` 回调方法获取。 audio: (flag) True/False，指定该音视频流是否包含音频资源。

参数名称	类型	描述
		video: (flag) True/False，指定该音视频流是否包含视频资源。 screen: (flag) True/False，指定该音视频流是否包含屏幕共享功能；在当前版本中应设置为否（False）。 attributes: （非必选项）包含以下属性： - resolution: 分辨率，可设置为 {‘sif’, ‘vga’, ‘hd720p’} 中任一项 - minFrameRate: 最小视频帧率 - maxFrameRate: 最大视频帧率 (可选) cameraId: 通过 getDevices 方法获取的摄像头设备 ID。 (可选) microphoneId: 通过 getDevices 方法获取的麦克风设备 ID。

示例代码：

```
var client = AgoraRTC.createClient({streamID: uid, audio:true, video:true, screen:false});
```

2. Client 接口方法和回调事件

方法

`init(key, onSuccess, onFailure)`

初始化客户端对象。

参数名称	类型	描述
key	字符串	Agora 提供的厂商秘钥。
onSuccess	函数	（非必选项）方法调用成功时执行的回调函数。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

```
client.init(vendorKey, function() {
    log(“client initialized”);
    //join channel
    ……
}, function(err) {
    log(“client init failed ”, err);
    //error handling
});
```

`join(token, uid, channel, onSuccess, onFailure)`

该方法让用户加入 AgoraRTC 频道。

参数名称	类型	描述
token	字符串	今后的升级版本中将会用到的动态秘钥；在当前版本中可设置为**未定义**。
uid	字符串	（非必选项）指定用户 ID。32 位无符号整数。建议设置范围：1 到(2^32-1)，并保证唯一性。如果不指定（即设为 0），服务器会自动分配一个，并在 onSuccess 回调方法中返回。
channel	字符串	标识通话频道的字符串。
onSuccess	函数	（非必选项）方法调用成功时执行的回调函数，返回值为代表用户身份的 uid。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
client.join(undefined, '1024abc', function(uid) {
  log("client " + uid + "" joined channel");
  //create local stream
  ……
}, function(err) {
  log("client join failed ", err);
  //error handling
});
```

leave(onSuccess, onFailure)

该方法让用户离开 AgoraRTC 频道。

参数名称	类型	描述
onSuccess	函数	（非必选项）方法调用成功时执行的回调函数。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
client.leave(function() {
  log("client leaves channel");
  ……
}, function(err) {
  log("client leave failed ", err);
  //error handling
});
```

publish(stream, onFailure)

将本地音视频流上传至服务器。

参数名称	类型	描述
stream	对象	本地音视频流对象。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
client.publish(stream, function(err) {
  log.(“stream published”);
  ……
})
```

unpublish(stream, onFailure)

取消上传本地音视频流

参数名称	类型	描述
stream	对象	本地音视频流对象。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
client.unpublish(stream, function(err) {
  log.(“stream unpublished”);
  ……
})
```

subscribe(stream, onFailure)

从服务器端接收远程音视频流。

参数名称	类型	描述
stream	对象	远程音视频流对象。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
client.subscribe(stream, function(err) {
  log.(“stream unpublished”);
  ……
})
```

unsubscribe(stream, onFailure)

取消接收远程音视频流。

参数名称	类型	描述

参数名称	类型	描述
stream	对象	远程音视频流对象。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
client.unsubscribe(stream, function(err) {
  log.(“stream unpublished”);
  ……
})
```

回调事件

stream-published

通知应用程序本地音视频流已上传。

示例代码：

```
client.on(‘stream-published’, function(evt) {
  log.(“local stream published”);
  ……
})
```

stream-added

通知应用程序远程音视频流已添加。

示例代码：

```
client.on(‘stream-added’, function(evt) {
  var stream = evt.stream;
  log(“new stream added ”, stream.getId());
  //subscribe the stream
  ……
})
```

stream-removed

通知应用程序已删除远程音视频流，意即对方调用了 `unpublish stream`。

示例代码：

```
client.on(‘stream-removed, function(evt) {
  var stream = evt.stream;
  log.(“remote stream was removed”, stream.getId());
  ……
})
```

stream-subscribed

通知应用程序已取消接收远程音视频流。

示例代码：

```
client.on('stream-subscribed', function(evt) {
    var stream = evt.stream;
    log("new stream subscribed ", stream.getId());
    //play the stream
    ……
})
```

peer-leave

通知应用程序对方用户已离开会议室，意即对方调用了 `client.leave()`。

示例代码：

```
client.on('peer-leave', function(evt) {
    var uid = evt.uid;
    log("remote user left ", uid);
    ……
        })
```

错误代码

onFailure 回调函数返回以下错误代码：

错误代码	描述
10	用户秘钥无效。
11	操作无效。
12	本地音视频流无效。
13	远程音视频流无效。
100	Socket 连接错误。
101	无法连接 web 服务器。
102	通话连接丢失。
1000	服务不可用。
1001	加入频道失败。
1002	发布音视频流失败。
1003	取消发布音视频流失败。
1004	接收音视频流失败。
1005	取消接收音视频流失败。

3. Stream 接口方法

`init(onSuccess, onFailure)`

初始化音视频流对象。

参数名称	类型	描述
onSuccess	函数	（非必选项）方法调用成功时执行的回调函数。
onFailure	函数	（非必选项）方法调用失败时执行的回调函数。

示例代码：

```
stream.init(function() {
   log(“local stream initialized”);
   // publish the stream
   ……
}, function(err) {
   log(“local stream init failed ”, err);
   //error handling
        });
```

`getId()`

获取音视频流 ID。

`getAttributes()`

获取音视频流属性。

`hasVideo()`

获取视频 flag.

`hasAudio()`

获取音频 flag.

`enableVideo()`

启用视频轨道。在创建视频流时将 video flag 设置为 True 即可使用该方法，其功能相当于重启视频。

`disableVideo()`

禁用视频轨道。在创建视频流时将 video flag 设置为 True 即可使用该方法，其功能相当于暂停视频。

`enableAudio()`

启用音频轨道。其功能相当于重启音频。

disableAudio()

禁用音频轨道。其功能相当于暂停音频。

setVideoProfile(profile)

设置视频属性，为非必选项，且须在 stream.init()之前调用。

参数名称	类型	描述
profile	字符串	视频属性，可设置为以下几种形式： ‘120p_1’ : 160x120, 15fps, 80kbps ‘240p_1’ : 320x240, 15fps, 200kbps ‘480p_1’ : 640x480, 15fps, 500kbps （默认设置） ‘480p_2’ : 640x480, 30fps, 1Mbps ‘720p_1’ : 1280x720, 15fps, 1Mbps ‘720p_2’ : 1280x720, 30fps, 2Mbps ‘1080p_1’ : 1920x1080, 15fps, 1.5Mbps ‘1080p_2’ : 1920x1080, 30fps, 3Mbps

示例代码：

```
stream.setVideoProfile(‘480p_1’);
```

play(elementID)

播放视频流或音频流。

参数名称	类型	描述
elementID	字符串	html 元素 ID。
assetsURL	字符串	(非必选项) 资源文件的 URL 地址；默认保存在/assets/文件夹下。

示例代码：

```
stream.play(‘div_id’, ‘/res’); // stream will be played in div_id element, resource files under /res/assets/
```

stop()

停止音视频流，同时清除 elementID 下的 display DOM 树。

close()

关闭视频流或音频流。调用该方法则取消摄像头和麦克风的访问权限。