



Agora Web SDK

参考手册

1.2 版

support@agora.io

目录

简介	4
Agora CaaS	4
Agora Web SDK	4
用户须知	4
兼容性	4
浏览器支持	5
已知问题和局限性	5
入门指南	6
获取 SDK	6
获取 Vendor Key	6
程序开发和部署	7
运行示例程序	8
客户端	8
服务器	8
使用动态密钥（Dynamic Key）提高安全性	8
什么是动态密钥	8
如何使用动态密钥	10
动态密钥的安全性	12
Agora Web SDK - API 参考	13
AgoraRTC 接口类	13
针对实时通信创建客户端对象(createRtcClient)	13
针对在线直播创建客户端对象(createLiveClient)	14
针对跨平台互通创建客户端对象(createClient)	14
枚举系统设备(getDevices)	14
创建音视频对象(createStream)	14
Client 接口类	15
加入 AgoraRTC 频道(join)	16
离开 AgoraRTC 频道(leave)	16
上传本地音视频流(publish)	17
取消上传本地音视频流(unpublish)	17
订阅远程音视频流(subscribe)	17
取消订阅远程音视频流(unsubscribe)	18
本地音视频已上传回调事件(stream-published)	18
远程音视频流已添加回调事件(stream-added)	18
远程音视频流已删除回调事件(stream-removed)	19
远程音视频流已订阅回调事件(stream-subscribed)	19
对方用户已离开会议室回调事件(peer-leave)	19
错误代码	19
Stream 接口类	21

初始化音视频对象(init).....	21
获取音视频流 ID(getId)	21
获取音视频流属性(getAttributes)	21
获取视频 flag(hasVideo)	21
获取音频 flag(hasAudio)	21
启用视频轨道(enableVideo).....	21
禁用视频轨道(disableVideo)	22
启用音频轨道(enableAudio)	22
禁用音频轨道(disableAudio).....	22
设置视频属性(setVideoProfile)	22
播放音视频流(play)	22
停止音视频流(stop).....	23
关闭音视频流(close).....	23

简介

Agora CaaS

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。

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。在下文的[入门指南](#)一节中将为您阐述如何部署和使用 Agora Web SDK。

Agora Web SDK

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。

用户须知

兼容性

Agora Web SDK v1.2 版与 Agora Native SDK v1.2 和 v1.3 兼容，支持相应的音视频功能。Agora Native SDK 用于在 Windows、Mac、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。

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。

已知问题和局限性

1. Agora Web SDK 基于 WebRTC 技术运行，但以下浏览器和平台目前不支持 WebRTC：

- 浏览器：微软 IE 和苹果 Safari
- 平台：苹果 iOS

关于更多详细内容，参考上文[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 中不支持最大视频码率设置。
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 中未包含。

入门指南

本节主要阐述如何部署和使用 Agora Web SDK。

获取 SDK

请联系 sales@agora.io 获取最新版的 SDK。

SDK 内包含以下文件：

组件	描述
./doc	Agora Web SDK 文档： Agora_Web_SDK_Release_Notes_v1_2_CHS.pdf Agora_Web_SDK_Reference_Manual_v1_2_CHS.pdf
./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

在示例程序中启用音频或视频通话时，需要在启动窗口中输入您的 vendor key。在开发程序时，需要将 vendor key 嵌入您的代码中以便调用 SDK。Vendor Key 是唯一的。

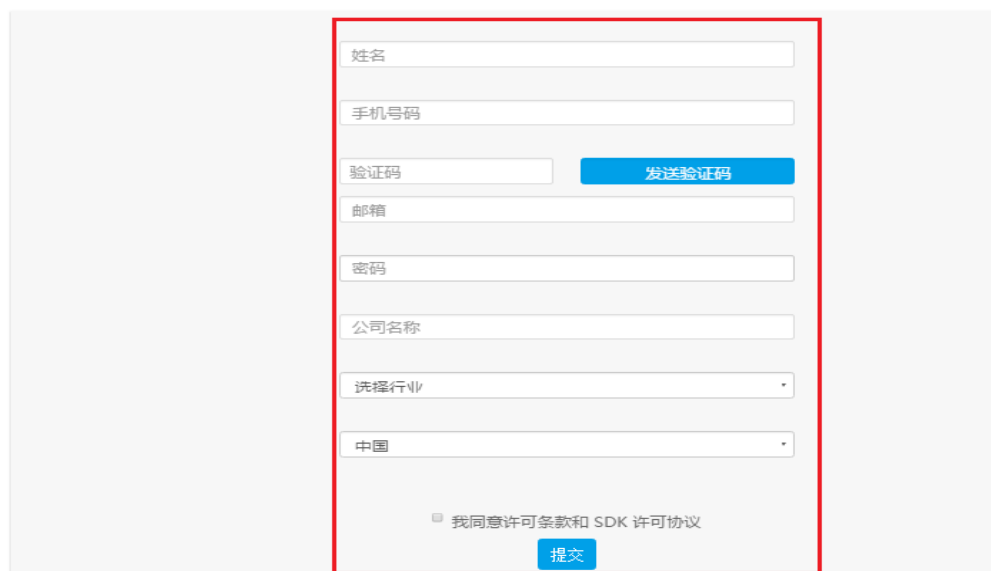
本章仅介绍如何单独使用 Vendor Key，您还可以在您的网页服务器代码内将其与 Sign Key 一起使用来生成一个安全性更高的 Dynamic Key（动态密钥）：

- 关于动态密钥的概念和使用方法（**推荐**），请参考章节[使用动态密钥（Dynamic Key）提高安全性](#)。
- 关于如何运行示例程序（针对 Vendor Key 和 Dynamic Key），请参考章节[运行示例程序](#)。

根据以下步骤获取 Vendor Key：

1. 访问 cn.agora.io，点击右上角的**注册**，如下图所示：

注册



姓名

手机号码

验证码 **发送验证码**

邮箱

密码

公司名称

选择行业

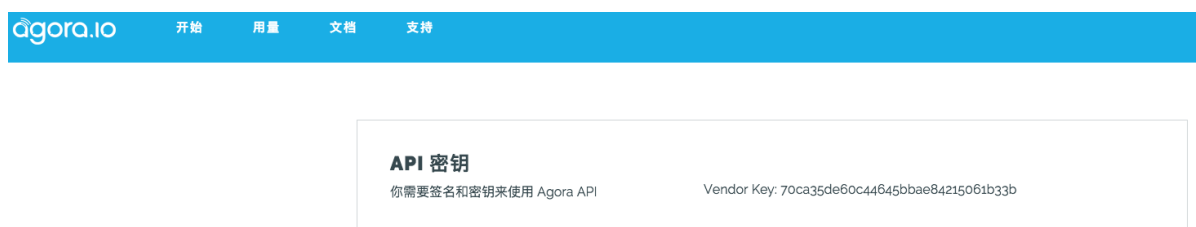
中国

☐ 我同意许可条款和 SDK 许可协议

提交

2. 按照页面提示完成注册。

在**注册完成**页面，点击**登录**，输入账户信息，登录之后，您将在页面 <http://dashboard.agora.io/> 看见自己的 Vendor Key（如下图所示）。同时，您的注册邮箱也会收到一封包含 Vendor Key 的系统邮件。



3. 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。

1. 加载 Agora JS 库 AgoraRTCSDK-1.2.5.js, 可从以下地址下载:

<https://rtc.sdk.agora.io/AgoraRTCSDK-1.2.5.js> 或者

<http://rtc.sdk.agora.io/AgoraRTCSDK-1.2.5.js>

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

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）。按照以下步骤运行网页示例程序：

1. 安装本地网页服务器，如 Apache, NginX 或 Node.js。
2. 将 `./client/` 下的文件部署到网页服务器上，并启动 http/https 服务。
3. 在网页服务器上用浏览器打开示例程序页面（请使用 [浏览器支持](#) 一节中所列出的浏览器，推荐使用 Google Chrome）。

如需使用**动态密钥**，请执行以下操作：

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

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。

注：更多信息，参考[使用动态密钥（Dynamic Key）提高安全性](#)。

本示例代码是 JavaScript 写成，需要标准的 Node.js 服务器：

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

使用动态密钥（Dynamic Key）提高安全性

每一家使用 Agora SDK 的企业都将获取一个唯一的代码，即“Vendor Key”，用于识别该企业。在 Agora Global Network 上的通信按照 vendor key 隔离，持有不同的 vendor key 的用户即使加入的频道名称相同也会隔离，不会互相干扰。

但是，Vendor Key 是一种静态密钥，如果有人非法获取了您的 Vendor Key，他将可以在 Agora 提供的 SDK 中使用您的 vendor key，如果知道频道名字，甚至有可能干扰正常的频道。为了提高程序的安全性，建议您在大规模的程序中使用动态密钥（**Dynamic Key**）方案。

什么是动态密钥

动态密钥方案是一种更为安全的用户身份验证方案，每次用户访问 Agora 的服务进入一个频道开始通话前，后台服务通过 Agora 提供的 Vendor Key 和 Sign Key^注，用基于 HMAC 的安全算法生成一个新的动态密钥发送给客户端，客户端调用

AgoraRTC.client.join() API 接口函数使用此动态密钥。Agora 的服务器通过密钥验证用户是否合法。

注：（术语解释）

- **Vendor Key:** 用于识别您的企业或产品的 ID。对于没有启用动态密钥功能的企业，可仅凭 Vendor Key 访问 Agora 的服务。
- **Sign Key:** 用于生成动态密钥的签名密钥（使用方法见下文表 1）。Sign key 存储于服务器端，对任何用户均不可见。

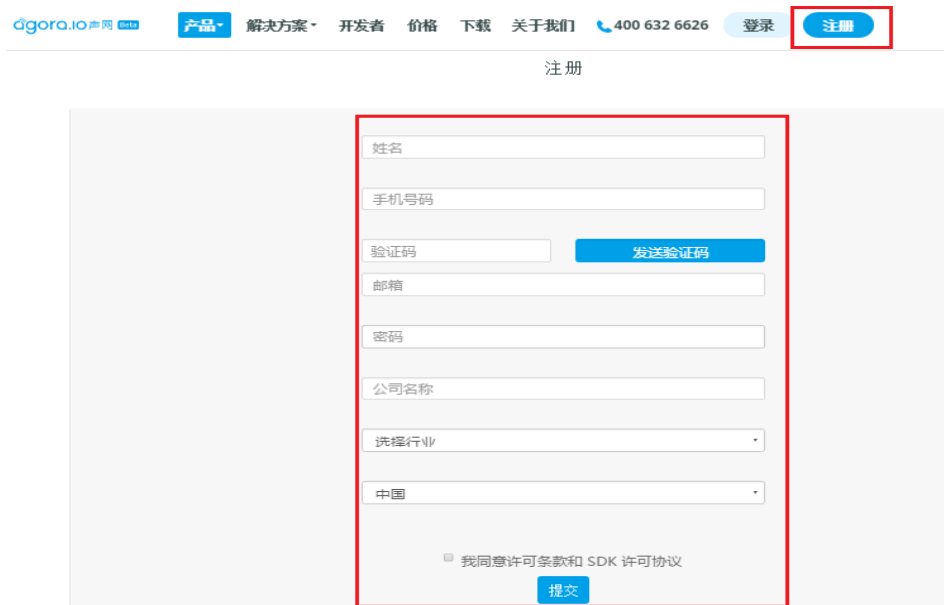
如何使用动态密钥

以下章节讲述如何使用动态密钥：

- [获取 Vendor Key](#)
- [获取 Sign Key](#)
- [集成动态密钥算法](#)
- [使用动态密钥](#)

获取 Vendor Key

1. 访问 cn.agora.io，点击右上角的**注册**，如下图所示：



2. 按照页面提示完成注册。

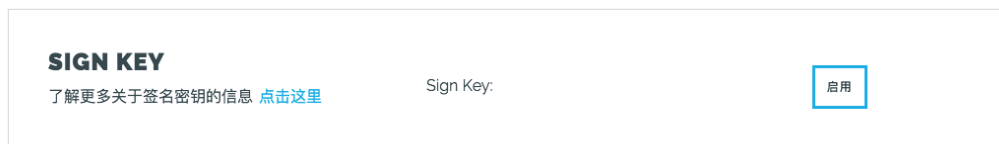
在**注册完成**页面，点击**登录**，输入账户信息，登录之后，您将在页面 <http://dashboard.agora.io/> 看见自己的 Vendor Key（如下图所示）。同时，您的注册邮箱也会收到一封包含 Vendor Key 的系统邮件。



3. 获取 Vendor Key 后，即可使用对应的 Agora.io 服务功能。

获取 Sign Key

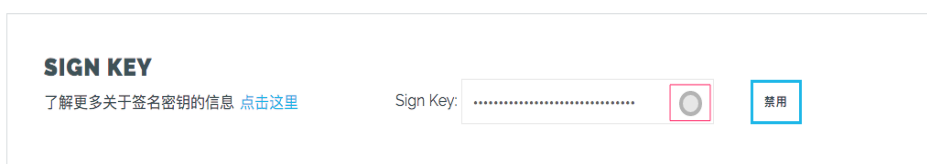
1. 在 dashboard.agora.io 网页上点击启用。



2. 点击下图红色标记处的按钮，Sign key 就会完整显示出来。

再次点击该按钮 Sign Key 再次回到隐藏状态。

当心！：请将 Sign Key 存在服务器端，且对任何客户端均不可见。



3. 如果由于某些原因需要更新 Sign Key，请联系 support@agora.io。

集成动态密钥算法

开发者将生成动态密钥的算法集成到自己企业的信令服务中。Agora 提供了示例代码，涵盖 C++, Java, Python, node.js 等语言，可以直接将代码应用在您的程序中。

请登录以下任意网址获取示例代码：

- Dashboard 的 Sign Key 页面：<https://dashboard.agora.io/signkey>
- Github：<https://github.com/AgoraLab/AgoraDynamicKey/tree/develop>

使用动态密钥

每一次客户端加入频道前（如发起会话、接收到会话邀请）：

1. 客户端请求企业自己的服务器的信令服务授权。
2. 服务器端基于 Sign Key、Vendor Key、频道名称、当前时间戳，客户端用户 id，有效期时间戳等信息通过 Agora 提供的算法生成动态密钥，返回给授权的客户端应用程序。
3. 客户端应用程序调用 `AgoraRTC.client.join()`，第一个参数要求为动态密钥。

4. Agora 的服务器接收到动态密钥信息，验证该通话是来自于合法用户，并允许访问 Agora Global Network。

注：采用了动态密钥方案后，在用户进入频道时，Dynamic Key 将替换原来的 Vendor Key。

动态密钥的安全性

采用动态密钥方案的企业，在用户加入频道时都需要提供动态密钥，确保用户每次通话都经过信令服务器的验证。动态密钥采用 HMAC/SHA1 签名方案，提高了系统及通话的安全性。

动态密钥结构

下表中所有字段从前往后拼接，共 103 字节。

字段	类型	长度	说明
版本号	字符串	3	动态密钥算法的版本信息
签名	字符串	40	签名的 hex 编码。将 Sign Key 以及下列字段作为输入，通过 HMAC 计算和 hex 编码而成的 40 字节字符串： <ul style="list-style-type: none">✓ 服务类型：ASCII 可见字符串，由 Agora 提供。详见章节 服务类型。✓ Vendor key: 32 位 vendor key 字符串。✓ Timestamp: 动态密钥生成时的时间戳。✓ 随机数: 32 位整数 hex 编码。随每个请求重新生成。✓ Channel: 频道名称，用户自定义，不超过 64 字节。✓ 用户 id : 客户端自定义的用户 id✓ 服务过期时间 : 用户在频道内可以通话截止的时间戳
Vendor Key	字符串	32	Vendor Key
授权时间戳	数字	10	动态密钥生成时的时间戳，自 1970.1.1 开始到当前时间的秒数。授权 5 分钟内可以访问 Agora 服务。
随机数	整数	8	32 位整数 hex 编码。随每个请求重新生成。
服务过期时间戳	数字	10	用户使用 Agora 服务终止的时间戳，在此时间之后，将不能继续使用 Agora 服务（比如进行的通话会被强制终

			止)；如果对终止时间没有限制，设置为 0。
--	--	--	-----------------------

表 1

服务类型

服务	值	说明
通话服务	ACS	Agora 提供的音视频通信服务，例如当使用 AgoraRtc.client.join() API 时，如果需要传入动态密钥，需要使用这个类型来生成动态密钥。

签名算法：HMAC/SHA1

动态密钥采用业界标准化的 HMAC/SHA1 加密方案，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。具体加密方案可参看以下网页：

http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

如需更多技术支持，请联系 Agora.io。

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。

AgoraRTC 接口类

针对实时通信创建客户端对象(createRtcClient)

createRtcClient()

该方法创建和返回针对实时通信用户场景的客户端对象，包含音视频聊天和会议。该方法在每次会话里仅调用一次。

注：

推荐小群组聊天和会议（包括一对一）场景使用此方法，因为该方法专门针对此种场景做了优化。

示例代码：

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

针对在线直播创建客户端对象(createLiveClient)

createLiveClient()

该方法创建和返回针对在线直播（一对多）用户场景的客户端对象。该方法在每次会话里仅调用一次。

示例代码：

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

针对跨平台互通创建客户端对象(createClient)

createClient()

该方法创建和返回针对跨平台互通的客户端对象。例如，它可以实现手机和网页用户的互通。本方法将在后续的发布版本中弃用。

示例代码：

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

枚举系统设备(getDevices)

getDevices (callback)

该方法枚举系统中摄像头/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;  
});
```

创建音视频对象(createStream)

createStream (spec)

该方法创建并返回音视频流对象。

参数名称	类型	描述
------	----	----

参数名称	类型	描述
spec	对象	<p>该对象包含以下属性：</p> <p><u>streamID</u>: 音视频流 ID，通常设置为 uid，可通过 Client join 回调方法获取。</p> <p><u>audio</u>: (flag) True/False，指定该音视频流是否包含音频资源。</p> <p><u>video</u>: (flag) True/False，指定该音视频流是否包含视频资源。</p> <p><u>screen</u>: (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）。</p> <p><u>attributes</u>: （非必选项）包含以下属性：</p> <ul style="list-style-type: none"> - 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 <p>(可选) <u>cameraId</u>: 通过 getDevices 方法获取的摄像头设备 ID。</p> <p>(可选) <u>microphoneId</u>: 通过 getDevices 方法获取的麦克风设备 ID。</p>

示例代码：

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

Client 接口类

方法

初始化客户端对象 (init)

init(key, onSuccess, onFailure)

该方法初始化客户端对象。

参数名称	类型	描述
key	字符串	<p>为以下任一密钥：</p> <ul style="list-style-type: none"> ● Vendor Key: 注册时由 Agora.io 提供的。 ● Dynamic Key: 根据 Vendor Key 和 Sign Key，通过 Agora.io 提供的安全算法生成。 <p>Vendor Key 和 Sign Key 均可通过 dashboard.agora.io 获取。</p>
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
});

```

加入 AgoraRTC 频道(join)

join(channel, uid, onSuccess, onFailure)

该方法让用户加入 AgoraRTC 频道。

参数名称	类型	描述
channel	字符串	标识通话频道的字符串。
uid	字符串	指定用户的 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。
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
});

```

离开 AgoraRTC 频道(leave)

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
});

```

上传本地音视频流(publish)

publish(stream, onFailure)

该方法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");
    .....
});

```

取消上传本地音视频流(unpublish)

unpublish(stream, onFailure)

该方法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");
    .....
});

```

订阅远程音视频流(subscribe)

subscribe(stream, onFailure)

该方法从服务器端接收远程音视频流。

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.subscribe(stream, function(err) {
  log("stream unpublished");
  .....
})
```

取消订阅远程音视频流(**unsubscribe**)

unsubscribe(stream, onFailure)

该方法取消接收远程音视频流。

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.unsubscribe(stream, function(err) {
  log("stream unpublished");
  .....
})
```

回调事件

本地音视频已上传回调事件(**stream-published**)

该回调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");
  .....
})
```

远程音视频流已添加回调事件(**stream-added**)

该回调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
```

```
.....  
})
```

远程音视频流已删除回调事件(stream-removed)

该回调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());  
  .....  
})
```

远程音视频流已订阅回调事件(stream-subscribed)

该方法通知应用程序已接收远程音视频流。

示例代码：

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

对方用户已离开会议室回调事件(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	用户密钥无效。
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。

Stream 接口类

初始化音视频对象(init)

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
});
```

获取音视频流 ID(getId)

getId()

获取音视频流 ID。

获取音视频流属性(getAttributes)

getAttributes()

获取音视频流属性。

获取视频 flag(hasVideo)

hasVideo()

获取视频 flag。

获取音频 flag(hasAudio)

hasAudio()

获取音频 flag。

启用视频轨道(enableVideo)

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。

禁用视频轨道(disableVideo)

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。

启用音频轨道(enableAudio)

enableAudio()

启用音频轨道。其功能相当于重启音频。

禁用音频轨道(disableAudio)

disableAudio()

禁用音频轨道。其功能相当于暂停音频。

设置视频属性(setVideoProfile)

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');
```

播放音视频流(play)

play(elementID, assetsURL)

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/

停止音视频流(stop)

stop()

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

关闭音视频流(close)

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。

Agora CaaS, Agora Global Network, Agora Native SDK和Agora Web SDK为Agora.io的注册商标。Agora.io经营业务中也使用Agora Lab这一名称。本文中提及的其他产品或公司名称均为其各自公司的注册商标。

©2016 Agora.io. 版权所有。