

11 | 如何通过Node.js实现一套最简单的信令系统？

2019-08-08 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 20:54 大小 19.14M



通过前面几篇文章的讲解，我想现在你应该已经对 WebRTC 有了一个清楚的认知了。接下来的章节咱们就使用 WebRTC 逐步实现一套真实可用的 1 对 1 实时直播系统吧。

WebRTC 1.0 规范对 WebRTC 要实现的功能、API 等相关信息做了大量的约束，比如规范中定义了如何采集音视频数据、如何录制以及如何传输等。甚至更细的，还定义了都有哪些 API，以及这些 API 的作用是什么。但这些约束只针对于客户端，并没有对服务端做任何限制。

那 WebRTC 规范中为什么不对服务器也做约束呢？其实，这样做有以下三点好处。

第一点，可以集中精力将 WebRTC 库做好。 WebRTC 的愿景是使浏览器能够方便地处理音视频相关的应用，规范中不限制服务端的事儿，可以使它更聚焦。

第二点，让用户更好地对接业务。你也清楚，信令服务器一般都与公司的业务有着密切的关系，每家公司的业务都各有特色，让它们按照自己的业务去实现信令服务器会更符合它们的要求。

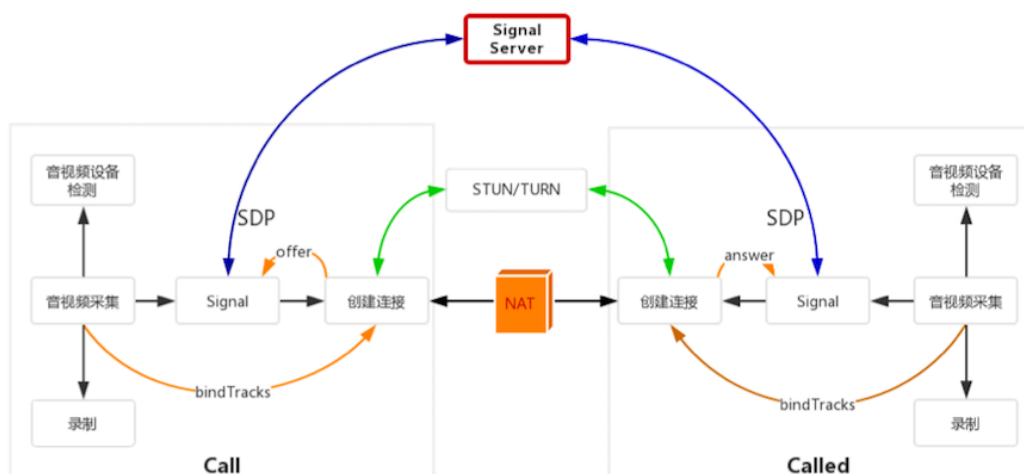
第三点，能得到更多公司的支持。WebRTC 扩展了浏览器的基础设施及能力，而不涉及到具体的业务或产品，这样会更容易得到像苹果、微软这种大公司的支持，否则这些大公司之间就会产生抗衡。

当然，这样做也带来了一些坏处，最明显的一个就是增加了学习 WebRTC 的成本，因为你在学习 WebRTC 的时候，必须**自己去实现信令服务器**，否则你就没办法让 WebRTC 运转起来，这确实增加了不少学习成本。

不过有了本专栏，你就不用再担心这个问题了。接下来，我就向你讲解一下，**如何实现一套最简单的 WebRTC 信令服务系统**。有了这套信令服务系统，WebRTC 就能运转起来，这样你就能真正地体验到 WebRTC 的强大之处了。

在 WebRTC 处理过程中的位置

在开始讲解 WebRTC 信令服务器之前，我们先来看一下本文在 WebRTC 处理过程中的位置。



WebRTC 处理过程图

通过上面这幅图，你可以清楚地知道本文所讲的主要内容就是红色方框中的**信令服务器**部分。

WebRTC 信令服务器的作用

你若想要实现 WebRTC 信令服务器，首先就要知道它在 WebRTC 1 对 1 通信中所起的作用。实际上它的功能是蛮简单的，就是**进行信令的交换，但作用却十分关键。在通信双方彼此连接、传输媒体数据之前，它们要通过信令服务器交换一些信息，如媒体协商。**

举个例子，假设 A 与 B 要进行音视频通信，那么 A 要知道 B 已经上线了，同样，B 也要知道 A 在等着与它通信呢。也就是说，**只有双方都知道彼此存在，才能由一方向另一方发起音视频通信请求，并最终实现音视频通话。**比如我们在[《08 | 有话好商量，论媒体协商》](#)一文中讲的媒体信息协商的过程就是这样一个非常典型的案例，双方的 SDP 信息生成后，要通过信令服务器进行交换，从而达到媒体协商的目的。

那在 WebRTC 信令服务器上要实现哪些功能，才能实现上述结果呢？我想至少要实现下面两个功能：

1. **房间管理。**即每个用户都要加入到一个具体的房间里，比如两个用户 A 与 B 要进行通话，那么它们必须加入到同一个房间里。
2. **信令的交换。**即在同一个房间里的用户之间可以相互发送信令。

信令服务器的实现

了解了 WebRTC 信令服务器的作用，并且还知道了信令服务器要实现的功能，接下来我们就操练起来，看看如何实现信令服务器吧！我将从下面 5 个方面来向你逐步讲解如何实现一个信令服务器。

1. 为什么选择 Node.js？

要实现信令服务器，你可以使用 C/C++、Java 等语言一行一行从头开始编写代码，也可以以现有的、成熟的服务器为基础，做二次开发。具体使用哪种方式来实现，关键看你的服务器要实现什么功能，以及使用什么传输协议等信息来决策。

以我们要实现的信令服务器为例，因它只需要传输几个简单的信令，而这些信令既可以使用 TCP、HTTP/HTTPS 传输，也可以用 WebSocket/WSS 协议传输，所以根据它使用的传输协议，你就可以很容易地想到，通过 Web 服务器（如 Nginx、Node.js）来构建我们的信令服务器是最理想、最省时的、且是最优的方案。

你可以根据自己的喜好选择不同的 Web 服务器（如 Apache、Nginx 或 Node.js）来实现，而今天我们选择的是 Node.js，所以接下来我们将要讲解的是**如何使用 Node.js 来搭建信令服务器**。

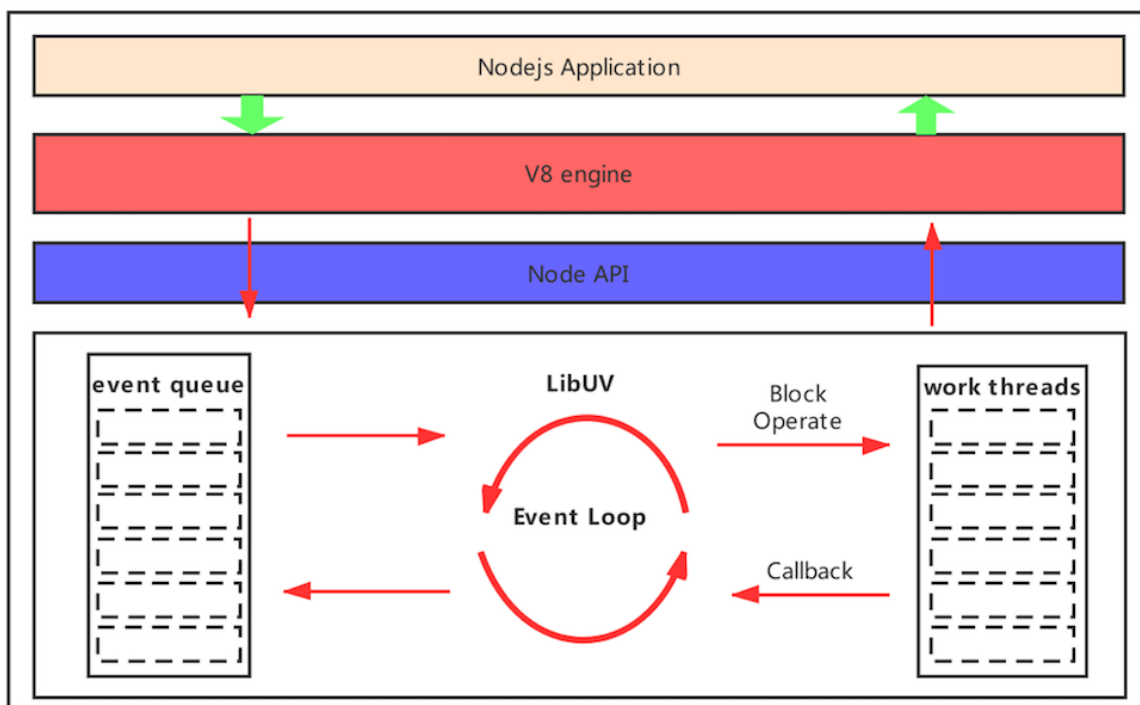
实际上，Apache、Nginx 和 Node.js 都是非常优秀、且成熟的 Web 服务器，其中 Nginx 可以说是性能最好的 Web 服务器了，但**从未来的发展角度来说，Node.js 则会更有优势**。

Node.js 的最大优点是可以使用 JavaScript 语言开发服务器程序。这样使得大量的前端同学可以无缝转到服务器开发，甚至有可能前后端使用同一套代码实现。对于使用 JavaScript 语言实现全栈开发这一点来说，我想无论是对于个人还是对于企业都是极大的诱惑。更可贵的是 **Node.js 的生态链非常完整**，有各种各样的功能库，你可以根据自己的需要通过安装工具（如 NPM）快速将它们引入到你的项目中，这极大地提高了 JavaScript 研发同学的开发效率。

Node.js 的核心是 V8（JavaScript）引擎，Node.js 通过它解析 JavaScript 脚本来达到控制服务器的目的。对于 JavaScript 同学来说，Node.js 的出现是革命性的，它不仅让 JavaScript 同学成为了全栈开发工程师，而且还让 JavaScript 开发同学的幸福指数飙升，真正地感受到了 JavaScript 无所不能的能力。对于我这样的开发“老鸟”来说，10 年前还不敢想象通过 JavaScript 来写服务器程序呢，现在它却已成为现实！

当然，Node.js 不仅可以让你用 JavaScript 控制服务器，它还为你留了拓展接口，这些拓展接口甚至可以让你使用 C/C++ 为它编写模块，这样 Node.js 的功能就更加强大了。

2. Node.js 的基本工作原理

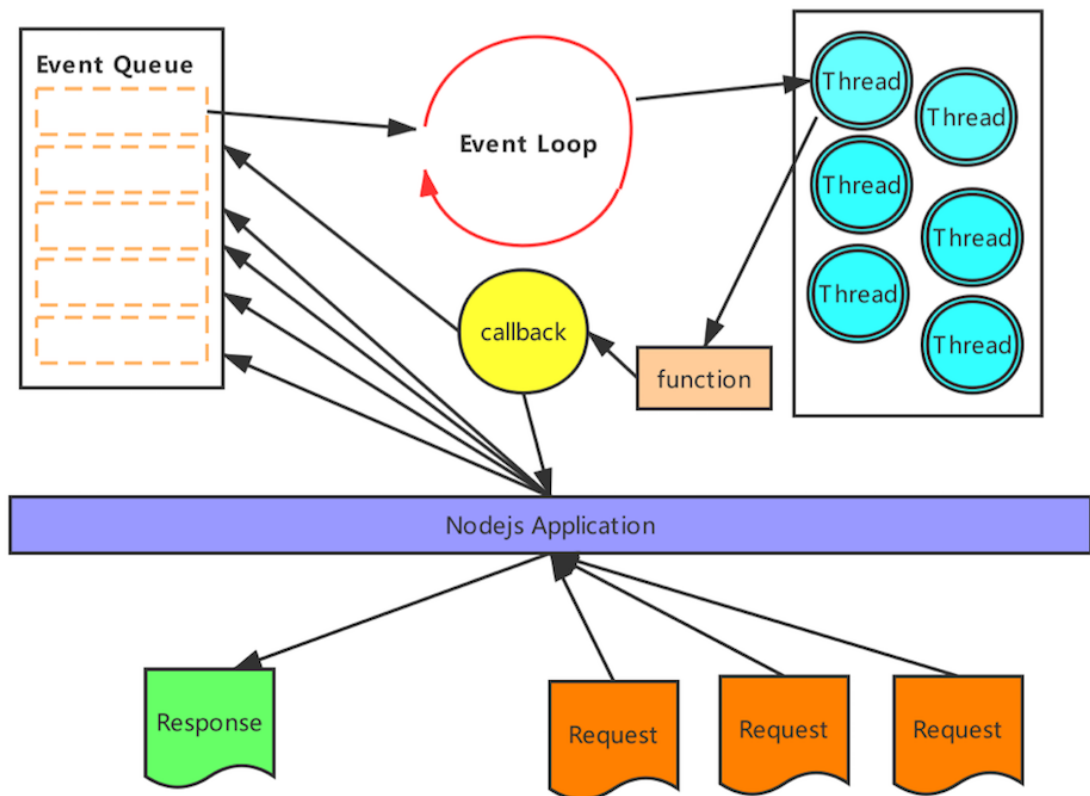


Node.js 工作原理图

Node.js 的工作原理如上图所示，其核心是 V8 引擎。通过该引擎，可以让 JavaScript 调用 C/C++ 方法或对象。反过来讲，通过它也可以让 C/C++ 访问 JavaScript 方法和变量。

Node.js 首先将 JavaScript 写好的应用程序交给 V8 引擎进行解析，V8 理解应用程序的语义后，再调用 Node.js 底层的 C/C++ API 将服务启动起来。所以 **Node.js 的强大就在于 JavaScript 与 C/C++ 可以相互调用，从而达到使其能力可以无限扩展的效果。**

我们以 Node.js 开发一个 HTTP 服务为例，Node.js 打开侦听的服务端口后，底层会调用 libuv 处理该端口的所有 HTTP 请求。其网络事件处理的过程就如下图所示：



Node.js 事件处理模型图

当有网络请求过来时，首先会被插入到一个事件处理队列中。libuv 会监控该事件队列，当发现有事件时，先对请求做判断，如果是简单的请求，就直接返回响应了；如果是复杂请求，则从线程池中取一个线程进行异步处理。

线程处理完后，有两种可能：一种是已经处理完成，则向用户发送响应；另一种情况是还需要进一步处理，则再生成一个事件插入到事件队列中等待处理。事件处理就这样循环往复下去，永不停歇。


3. 安装与使用 Node.js

了解了 Node.js 的基本原理后，接下来我们还是要脚踏实地来看看具体如何安装、使用 Node.js。

(1) 安装 Node.js


不同环境下安装 Node.js 的方法也不一样，不过都很简单。

在 Ubuntu 系统下执行：

 复制代码

```
1 apt install nodejs
```

或在 Mac 系统下执行：

 复制代码

```
1 brew install nodejs
```

通过以上步骤，Node.js 很快就安装好了（我这里安装的 Node.js 版本为：v8.10.0）。

（2）安装 NPM

除了安装 Node.js 之外，还要安装 NPM（Node Package Manager），也就是 Node.js 的包管理器，或叫包安装工具。它与 Ubuntu 下的 APT（Advanced Package Tool）命令或 Mac 系统下的 BREW 命令类似，是专门用来管理各种依赖库的。

以 Linux 为例，在 APT 没有出现之前，在 Linux 上安装软件是件特别麻烦的事儿，比如要安装一个编辑器，其基本步骤有如下：

先将这个工具（编辑器）的源码下载下来；

执行./configure 生成 Makefile 文件；

执行 make 命令对源码进行编译；

如果编译成功，执行 make install 将其安装到指定目录下；

如果编译过程中发现还需要其他库，则要对依赖库执行前面的 4 步，也就是先将依赖库安装好，然后再来安装该工具。


由这你可以看出，以前在 Linux 下安装个程序或工具是多么麻烦。

不过 Linux 有了 APT 工具后，一切都变得简单了。你只要执行 `apt install xxx` 一条命令就好了，它会帮你完成上面的一堆操作。

对于 Node.js 的安装包也是如此，**NPM 就是相当于 Linux 下的 APT 工具，它的出现大大提高了 JavaScript 开发人员的工作效率。**


下面我们就来看一下如何安装 NPM 工具，实际上，NPM 的安装就像前面安装 Node.js 一样简单。

在 Ubuntu 下执行：

 复制代码

```
1 apt install npm
```

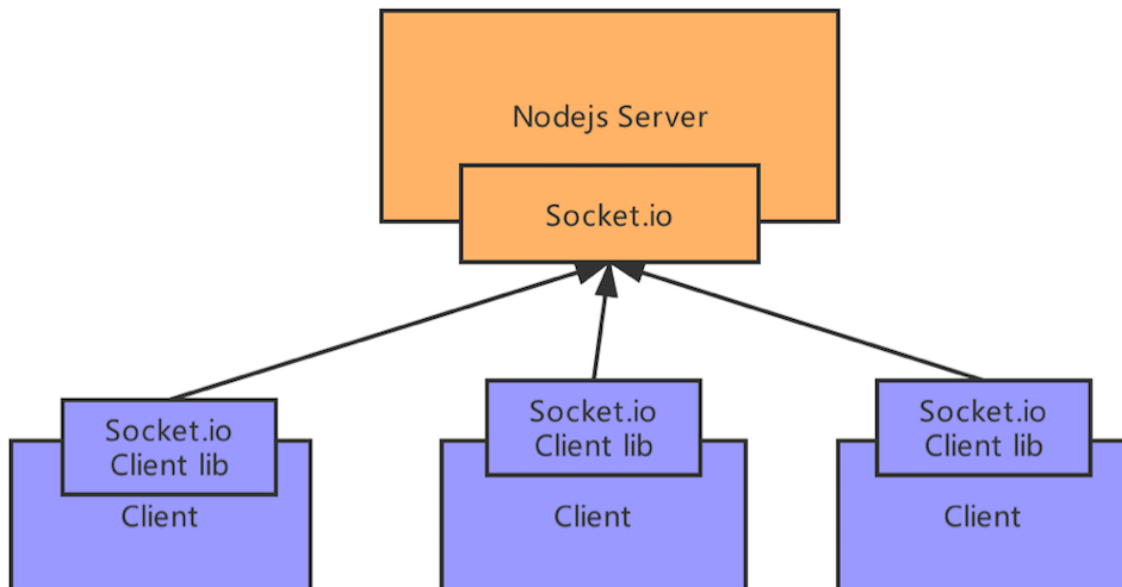
或在 Mac 下执行：

 复制代码

```
1 brew install npm
```

4. Socket.io 的使用

除了 Node.js 外，我们最终还要借助 Socket.io 来实现 WebRTC 信令服务器。Socket.io 特别适合用来开发 WebRTC 的信令服务器，通过它来构建信令服务器大大简化了信令服务器的实现复杂度，这主要是因为它内置了**房间**的概念。



Socket.io 与 Node.js 关系图

上图是 Socket.io 与 Node.js 配合使用的逻辑关系图，其逻辑非常简单。Socket.io 分为服务端和客户端两部分。服务端由 Node.js 加载后侦听某个服务端口，客户端要想与服务端相连，首先要加载 Socket.io 的客户端库，然后调用 `io.connect()`；即可与服务端连接上。

这里需要特别强调的是 Socket.io 消息的发送与接收。Socket.io 有很多种发送消息的方式，其中最常见的有下面几种，也是你必须要掌握的。

给本次连接发消息

复制代码


```
1 socket.emit()
```

给某个房间内所有人发消息

复制代码


```
1 io.in(room).emit()
```

除本连接外，给某个房间内所有人发消息

 复制代码

```
1 socket.to(room).emit()
```


除本连接外，给所有人发消息

 复制代码

```
1 socket.broadcast.emit()
```


你也可以看看下面的例子，其中 S 表示服务器，C 表示客户端，它们是发送消息与接收消息的比对。

发送 command 命令

 复制代码


```
1 S: socket.emit('cmd');
2 C: socket.on('cmd',function(){...});
```

发送了一个 command 命令，带 data 数据

 复制代码

```
1 S: socket.emit('action', data);
2 C: socket.on('action',function(data){...});
```

发送了 command 命令，还有两个数据

 复制代码

```
1 S: socket.emit(action,arg1,arg2);
2 C: socket.on('action',function(arg1,arg2){...});
```

有了以上这些知识，你就可以实现信令数据通讯了。

5. 实现信令服务器

接下来我们来看一下，如何通过 Node.js 下的 Socket.io 来构建一个服务器。


首先是客户端代码，也就是在浏览器里执行的代码。以下是 index.html 代码：

 复制代码

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>WebRTC client</title>
5   </head>
6   <body>
7     <script src='/socket.io/socket.io.js'></script>
8     <script src='js/client.js'></script>
9   </body>
10 </html>
```

该代码十分简单，就是在 body 里引入了两段 JavaScript 代码。其中，socket.io.js 是用来与服务端建立 Socket 连接的；client.js 的作用是做一些业务逻辑，并最终通过 Socket 与服务端通讯。

下面是 client.js 的代码：

 复制代码

```
1 var isInitiator;
2
3 room = prompt('Enter room name:'); // 弹出一个输入窗口
4
5 const socket = io.connect(); // 与服务端建立 socket 连接
6
7 if (room !== '') { // 如果房间不空，则发送 "create or join" 消息
8   console.log('Joining room ' + room);
9   socket.emit('create or join', room);
10 }
11
12 socket.on('full', (room) => { // 如果从服务端收到 "full" 消息
```

```


13 console.log('Room ' + room + ' is full');
14 });
15
16 socket.on('empty', (room) => { // 如果从服务端收到 "empty" 消息
17   isInitiator = true;
18   console.log('Room ' + room + ' is empty');
19 });
20
21 socket.on('join', (room) => { // 如果从服务端收到 "join" 消息
22   console.log('Making request to join room ' + room);
23   console.log('You are the initiator!');
24 });
25
26 socket.on('log', (array) => {
27   console.log.apply(console, array);
28 });

```

在该代码中，首先弹出一个输入框，要求用户写入要加入的房间；然后，通过 `io.connect()` 建立与服务端的连接；最后再根据 `socket` 返回的消息做不同的处理，比如收到房间满或空的消息等。

以上是客户端（也就是在浏览器）中执行的代码。下面我们来看一下服务端的处理逻辑。

服务器端代码，`server.js` 是这样的：

 复制代码

```

1 const static = require('node-static');
2 const http = require('http');
3 const file = new(static.Server)();
4 const app = http.createServer(function (req, res) {
5   file.serve(req, res);
6 }).listen(2013);
7
8 const io = require('socket.io').listen(app); // 侦听 2013
9
10 io.sockets.on('connection', (socket) => {
11
12   // convenience function to log server messages to the client
13   function log(){
14     const array = ['>>> Message from server: '];
15     for (var i = 0; i < arguments.length; i++) {
16       array.push(arguments[i]);
17     }
18     socket.emit('log', array);
19   }

```

```

20
21 socket.on('message', (message) => { // 收到 message 时，进行广播
22     log('Got message:', message);
23     // for a real app, would be room only (not broadcast)
24     socket.broadcast.emit('message', message); // 在真实的应用中，应该只在房间内广播
25 });
26
27 socket.on('create or join', (room) => { // 收到 “create or join” 消息
28
29     var clientsInRoom = io.sockets.adapter.rooms[room];
30     var numClients = clientsInRoom ? Object.keys(clientsInRoom.sockets).length : 0; //
31
32     log('Room ' + room + ' has ' + numClients + ' client(s)');
33     log('Request to create or join room ' + room);
34
35     if (numClients === 0){ // 如果房间里没人
36         socket.join(room);
37         socket.emit('created', room); // 发送 "created" 消息
38     } else if (numClients === 1) { // 如果房间里有一人
39         io.sockets.in(room).emit('join', room);
40         socket.join(room);
41         socket.emit('joined', room); // 发送 “joined”消息
42     } else { // max two clients
43         socket.emit('full', room); // 发送 "full" 消息
44     }
45     socket.emit('emit(): client ' + socket.id +
46         ' joined room ' + room);
47     socket.broadcast.emit('broadcast(): client ' + socket.id +
48         ' joined room ' + room);
49
50 });
51
52 });

```


该段代码中，在服务端引入了 node-static 库，使服务器具有发布静态文件的功能。服务器具有此功能后，当客户端（浏览器）向服务端发起请求时，服务器通过该模块获得客户端（浏览器）运行的代码，也就是上面我们讲到的 index.html 和 client.js，下发给客户端（浏览器）。

服务端侦听 2013 这个端口，对不同的消息做相应的处理：

服务器收到 message 消息时，它会直接进行广播，这样所有连接到该服务器的客户端都会收到广播的消息。

服务端收到 “create or join” 消息时，它会对房间里的人数进行统计，如果房间里没有人，则发送 “created” 消息；如果房间里有一人，发送 “join” 消息和 “joined” 消息；如果超过两个人，则发送 “full” 消息。


要运行该程序，需要使用 NPM 安装 socket.io 和 [node-static](#)，安装方法如下：

 复制代码

```
1 npm install socket.io
2 npm install node-static
```

启动服务器并测试

通过上面的步骤，你就使用 “Socket.io + Node.js” 实现了一个信令服务器。现在你还可以通过下面的命令将服务启动起来了：

 复制代码

```
1 node server.js
```

如果你是在本机上搭建的服务，则可以在浏览器中输入 “localhost:2013”，然后在浏览器中新建一个 tab，在里边再次输入 “localhost:2013”。这时，你就可以通过浏览器的控制台去看看发生了什么吧！

最后，再说一个快捷键小技巧吧，在 Chrome 下，你可以使用 Command-Option-J 或 Ctrl-Shift-J 的 DevTools 快速访问控制台。

小结

在本文中，我们介绍了 Node.js 的工作原理、Node.js 的安装与部署，以及如何使用 “Socket.io + Node.js” 实现 WebRTC 信令消息服务器。Socket.io 由于有房间的概念，所以与 WebRTC 非常匹配，因此，用它开发 WebRTC 信令服务器就会大大地减少工作量。

另外，本文中的例子虽说很简单，但在后面的文章中我会以这个例子为基础，在其上面不断增加一些功能，这样最终你就会看到一个完整的 Demo 程序。所以你现在还是要学习好这每一个知识点，打好基础，积跬步才能至千里。

思考时间

文中所讲的 JavaScript 代码需要运行在两个不同的 V8 引擎上，你知道它们的对应关系吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

 极客时间

从 0 打造音视频直播系统

手把手教你打造实时互动音视频直播系统

李超

新东方音视频直播技术专家
前沪江音视频架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | WebRTC NAT穿越原理

下一篇 12 | RTCPeerConnection：音视频实时通讯的核心

精选留言 (6)



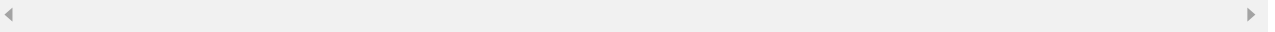
写留言



Ethan
2019-08-09

客户端一定要引入socket.io吗？可以直接用 websocket api吗

作者回复: 可以，用websocket你要自己写一个房间服务器

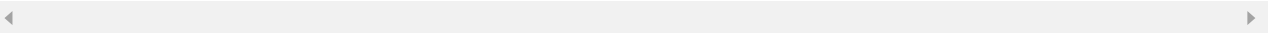


Beast-Of-Prey

2019-08-09

老师 `http://file/socket.io/?EIO=3&transport=polling&t=MnpuVBE net::ERR_NAME_NOT_RESOLVED` 这个错误是什么原因导致的？我百度了 说是浏览器设置了代理，但是我检查我的浏览器，没有进行设置啊。

作者回复: `Http://file/.` 这个开头？你是本地文件访问的？

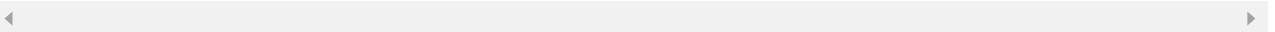


Beast-Of-Prey

2019-08-08

老师 我按步骤安装了 socket.io 但是 我本地 html 加载 socket.io.js文件的时候 提示2 个错误，1、文件未发现ERR_FILE_NOT_FOUND，2、io 未定义 io is not defined。

作者回复: 是库没有加载对，改为这样试试 `<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.3/socket.io.js"></script>`



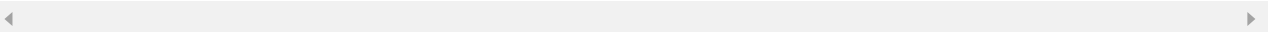
许童童

2019-08-08

两个不同的 V8 引擎上，你知道它们的对应关系吗
一个是nodejs服务端的V8，一个是浏览器中客户端的V8。

展开 ▾

作者回复: 没错！



Beast-Of-Prey



2019-08-08

读了好几遍

展开 ▾

作者回复: 赞！



彭刚

2019-08-08

期待更新 每篇文章都能看好几次 都是精华

展开 ▾

作者回复: 谢谢！

