

29 | 如何使用Medooze 实现多方视频会议？

2019-09-19 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 22:02 大小 15.15M



前面我们通过两篇文章详细介绍了 Medooze 的实现逻辑，我相信你现在已经对 Medooze 有了非常深刻的认知。通过对 Medooze 实现原理和架构的了解，我们可以知道 Medooze 支持的功能非常多，如录制回放、推流、SFU 等，其中最主要的功能是 SFU 的实现。

实际上，我们要实现的多方音视频会议就是由 SFU 来实现的，或者你可以认为 Medooz 实现的 SFU 就是一个最简单的音视频会议系统。Medooze 提供的 SFU Demo 的地址为：<https://github.com/medooze/sfu>。

由于 SFU 是一个 Node.js 项目，所以在环境部署、源码分析的时候，需要你有 JavaScript 基础，也需要掌握 NPM 和 Node 的常用命令，比如 `npm install` 等。接下来，我们主要从以下几方面向你介绍一下多方音视频会议（SFU）项目：

搭建多方音视频会议系统，让你感受一下多方通信；

分析多方音视频会议实现的架构；

分析 Medooze API 的具体使用；

分析多方音视频会议的入会流程。


多方音视频会议环境的搭建

多方音视频会议（SFU）项目是纯 Node.js 实现的，可以说在目前所有的操作系统上都可以运行。然而，由于该系统依赖 medooze-media-server 项目，而该项目仅支持 macOS X 和 Linux 两种操作系统，所以 SFU 项目也只能在 macOS X 和 Linux 这两种操作系统下运行了。

本文我们所搭建的这个 Demo 的实验环境是在 Ubuntu 18.04 系统上，所以接下来的实验步骤和实验结论都是基于 Ubuntu 18.04 得出的，如果你在其他操作系统下搭建可能结果会略有不同，但具体步骤都是一模一样的。


下面我们就来分析下这个环境搭建的具体步骤。

第一步，打开 Linux 控制终端，下载 SFU 代码，命令如下：

 复制代码


```
1 git clone https://github.com/medooze/sfu.git
```

第二步，安装 SFU 依赖的所有包：

 复制代码


```
1 cd sfu
2 npm install
```

第三步，在 `sfu` 目录下生成自签名证书文件：

 复制代码

```
1 openssl req -sha256 -days 3650 -newkey rsa:1024 -nodes -new -x509 -keyout server.key -o
```

第四步，在 `sfu` 目录下启动服务：


 复制代码

```
1 node index.js IP
```

上面命令中的 IP 是测试机器的 IP 地址，如果用内网测试，你可以将它设置为 127.0.0.1。如果你想用外网访问，则可以指定你的云主机的外网 IP 地址。另外需要注意的是，该服务启动后默认监听的端口是 8084，**所以你在访问该服务时要指定端口为 8084。**

通过上面的步骤，我们就将 Medooze 的 SFU 服务器搭建好了，下面我们来测试一下。

打开浏览器，在地址栏中输入以下地址就可以打开 Medooze SFU Demo 的测试界面了：

 复制代码

```
1 https://IP:8084/index.html
```

在这个地址中需要特别注意的是，**必须使用 HTTPS 协议，否则浏览器无法访问本地音视频设备。**此外，这个地址中的 IP 指的是 SFU 服务的主机的 IP 地址，可以是内网地址，也可以是外网地址。

当该地址执行后，会弹出登录界面，如下所示：

Medooze SFU

Please enter the room ID for the conference you want to create, your name and select the audio microphone you wish to use

Room Id

RANDOM

Name

Audio Device

默认 - Internal Microphone (Built-in) ▼



READY!

Medooze SFU 登录界面图

这是一个多人视频会议登录界面。关于登录界面，这里我们做一个简单的说明。

界面中的 Room Id 和 Name 可以随机生成，也可以自己指定。如果需要随机生成，点击后面的 RANDOM 按钮即可。

几个参会人若想要进入同一个房间，那么他们的 Room Id 必须相同。

在 Audio Device 列出了所有麦克风，可供参会人选择。

在 Audio Device 下面，有一个音量进度条，它可以测试麦克风是否正常工作。

一切准备就绪后，点击右下角的“READY!”按钮，就可以加入房间了。此时，在浏览器左下角显示了自己的本地视频。

由于这是基于浏览器的音视频应用，具有很好的跨平台性，因此你可选择 PC 浏览器、手机浏览器一起测试。

上面我们就将 Medooze 的音视频会议系统的 Demo 环境搭建好了，并且你可以进行多人的音视频互动了。接下来我们就分析一下这个音视频会议系统（SFU）Demo 是如何实现

的吧。

SFU 目录结构

首先我们来分析看一下 Medooze 的 SFU Demo 的目录结构，以及每个子目录中代码的作用，为我们后面的分析做好准备。

```
-rw-r--r--    1 root    root    5516 May 15 19:45 index.js
drwxr-xr-x    5 root    root    4096 May 15 16:19 www
drwxr-xr-x    2 root    root    4096 May 15 14:59 lib
-rw-r--r--    1 root    root     743 May 15 14:59 package.json
```

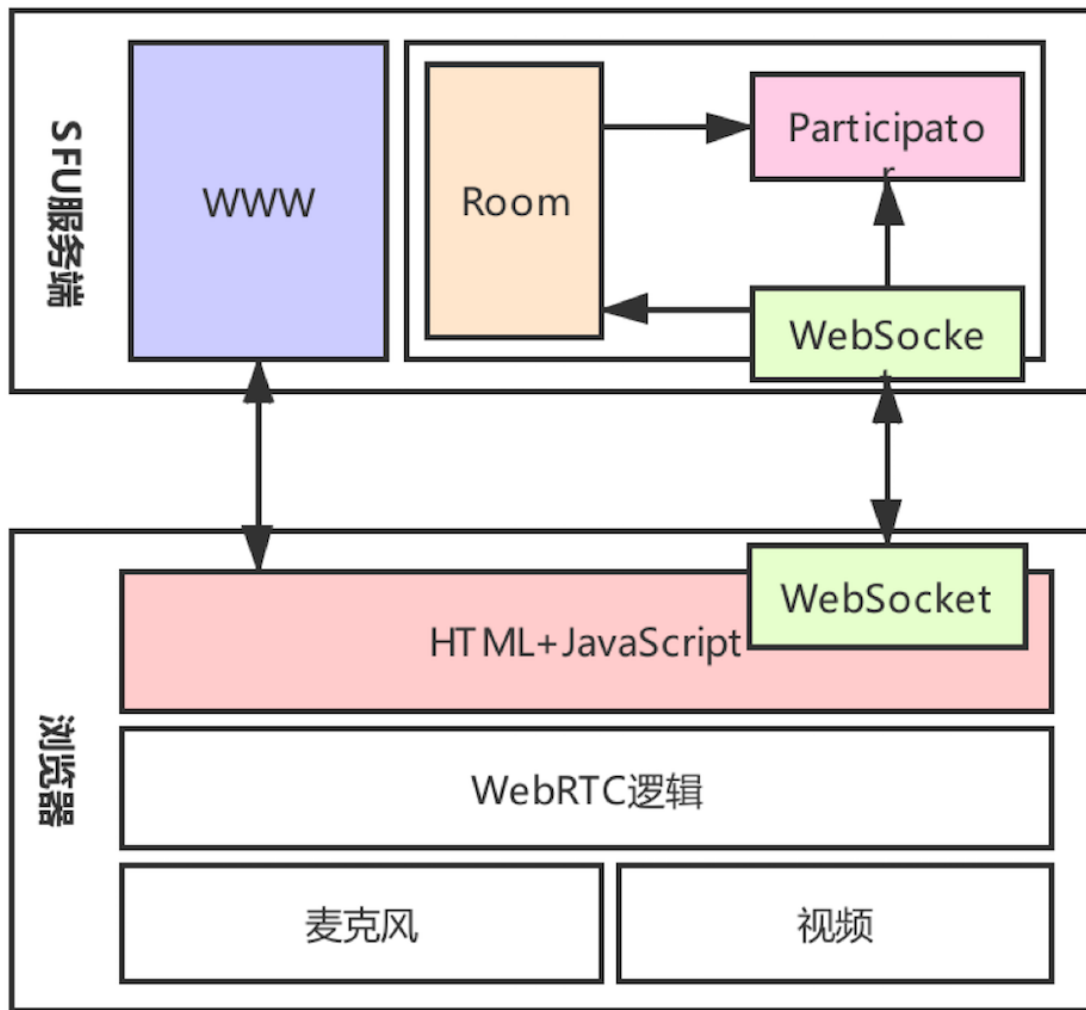
如上图所示，在 SFU Demo 目录下的文件非常少，我们来看一下每个文件或目录的具体作用吧！

名称	说明
lib	实现了 Room 和 Participator 类，在这两个类中会调用 Medooze API。
www	包含了 WebRTC 客户端的实现，如客户端的 CSS、HTML、JavaScript 都放在这里。
index.js	Node.js 应用服务的实现，一些 SFU 服务器逻辑也放在这里。
package.json	这是 SFU 项目的配置文件，所有依赖的包都在这里描述。

了解了 SFU Demo 的目录结构及其作用后，我们再来看一下 SFU Demo 的架构是什么样子。

SFU 架构

下图就是 Medooze SFU Demo 的架构图，你可以参考下：



SFU 架构图

从图中你可以看出，SFU 架构包括了**浏览器客户端**和**SFU 服务端**两部分。而 SFU 服务端又包括两部分功能：WWW 服务、WebSocket 信令及业务管理。

接下来我们就来分析一下**服务端的处理逻辑**。

Node.js 中的 Web 服务器 (WWW) 是在 index.js 文件中实现的。当我们执行 `node index.js IP` 命令时，该命令启动了一个 HTTPS 服务，监听的端口是 8084。当 HTTPS 启动后，我们就可以从它的 WWW 服务获取到客户端代码了。需要说明的是，HTTPS 服务依赖 server.key 和 server.crt 文件，这两个文件需要我们提前生成好（生成证书的命令在上面 SFU 环境搭建一节已经向你做了说明）。

至于 WebSocket 服务及业务管理部分，逻辑会稍微复杂些。服务端启动 WebSocket 服务后，可以通过它来接收客户端发来的信令，同时也可以通过它向客户端发送通知消息；除

了对信令的处理外，SFU 的业务逻辑也是在这部分实现的，如 Room 和 Participator 实例的创建、参会人进入 / 离开房间等。

这里需要你注意的是，WebSocket 服务和业务管理的代码也是在 index.js 中实现的。换句话说，在服务端的 index.js 中，实现了 WWW 服务、WebSocket 服务和 SFU 业务管理三块逻辑，这样“大杂烩”的代码实现也只是在 Demo 中才能这么写了。另外，index.js 的实现依赖了 websocket 和 transaction-manager 两个包。

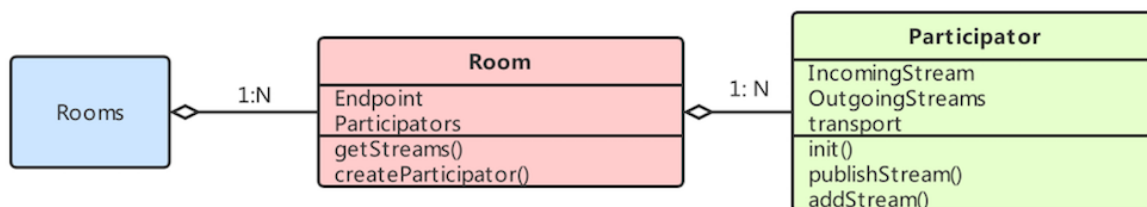
分析完服务端的处理逻辑后，接下来我们再来看看**客户端**。浏览器首先从 Medooze SFU 的 WWW 服务上获得 sfu.js 代码，在浏览器上运行 sfu.js 就可以与 Medooze SFU 建立连接，并进行多方音视频通信了。在 sfu.js 中，主要实现了房间的登录、音频设备选择、采集音视频数据、共享音视频数据等功能。实际上，客户端的实现逻辑就是使用咱们专栏第一模块所介绍的知识，如 RTCPeerConnection 的建立、获取音视频流、媒体协商等。

此外，Medooze SFU 还实现了几个信令，如 join 用于加入会议、update 用于通知客户端重新进行媒体协商等等，这几个信令还是非常简单的，不过却非常重要。尤其是在阅读 Medooze SFU 代码时，可以按照信令的流转来梳理 SFU 的代码逻辑，这样会大大提高你的工作效率。

以上就是 Medooze SFU 的基本架构，接下来我们看一下 SFU 的逻辑处理。

SFU 逻辑处理

SFU 的实现很简单。在服务端有两个重要的类，即**Room** 和 **Participator**。它们分别抽象了多人会议中的**房间**和**参会人**。为便于你理解，我画了一个简单类图，如下所示：



SFU 类图

从图中可以看到，index.js 中定义了一个**全局对象 Rooms**，保存多个房间（Room）实例。

Room 类是对房间的抽象：

Endpoint 属性是 Medooze Endpoint 类型；

Participators 属性是一个 Map 类型，保存本房间中所有 Participator 实例；

该类还有两个重要方法，createParticipator 用于创建参会人实例，getStreams 获取所有参会人共享的媒体流。

Participator 类是对参会人的抽象：

IncomingStreams 属性是 Map 类型，保存自己共享的媒体流，元素属性类型是 Medooze::IncomingStream；

OutgoingStreams 属性是 Map 类型，保存自己观看的媒体流，元素属性类型是 Medooze::OutgoingStream；

transport 属性是 Medooze::Transport 类型，表示 DTLS 传输；

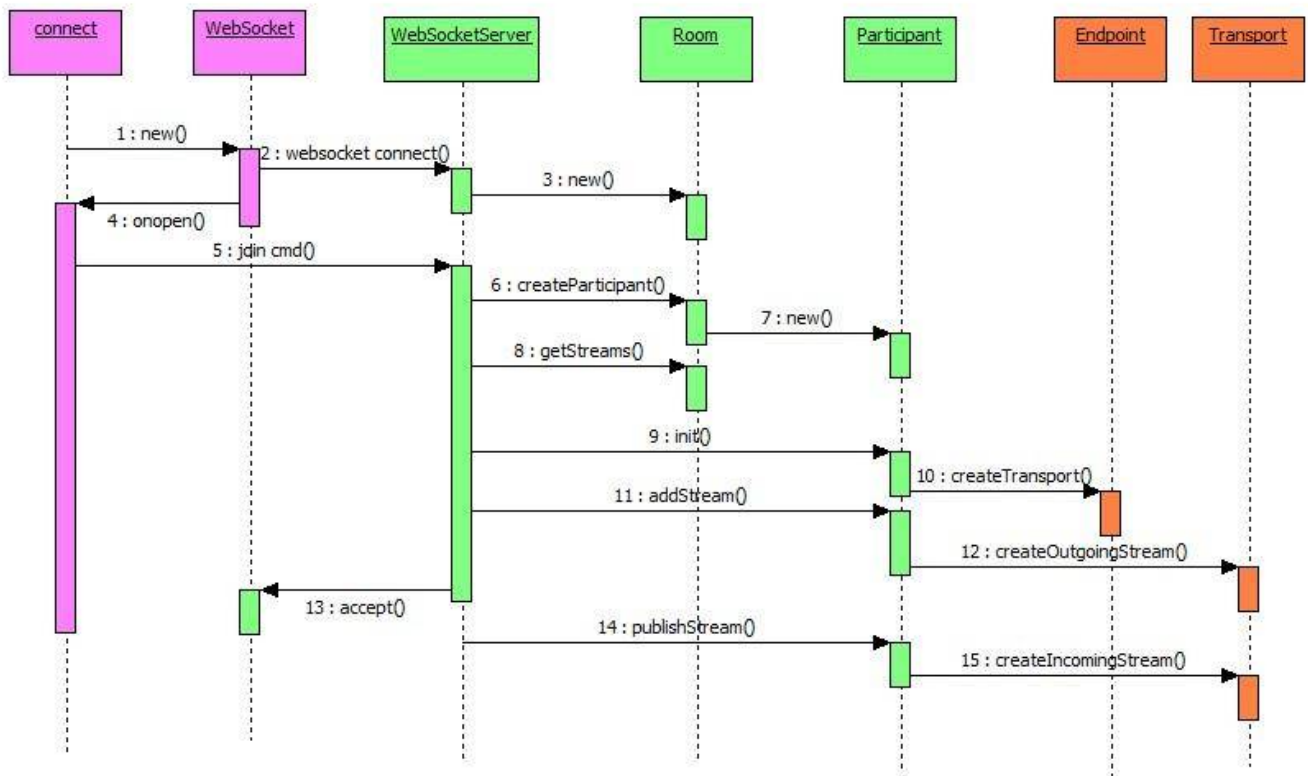
init() 方法，初始化 transport 实例等；

publishStream() 方法用于发布自己共享的流，创建 IncomingStream；

addStream() 方法用于观看其他参会人共享的流，创建 OutgoingStream 实例，并且 attachTo 到共享人的 IncomingStream 中。

入会流程

这里我们可以结合客户端和服务器的逻辑，分析一下参会人入会的大致流程，如下图所示：



SFU 时序图

图中深粉部分是客户端实现，在 `www/js/sfu.js` 中；绿色部分是服务器实现，在 `index.js` 中；橘色是在 `medooze` 中实现。那具体的流程是怎样的呢？

首先，浏览器从 Node.js 服务器获取到 `www/index.html` 页面及引用的 JavaScript 脚本，然后执行 `connect` 方法。在 `connect` 方法中使用 `WebSocket` 与 SFU 服务 (`index.js`) 建立连接。

服务端收到 `WebSocket` 连接请求后，从 URL 中解析到 `roomid` 参数。然后，根据该参数判断房间是否在之前已经创建过。如果该房间已创建，则什么都不做；如果没创建，则尝试创建 `Room` 实例。

对于客户端来说，`WebSocket` 连接建立成功后，客户端会发送 `join` 信令到服务端。而服务端收到 `join` 信令后，创建 `Participant`（参与人）实例，并且调用它的 `init` 方法进行初始化。

之后，`Participant` 获取 `Room` 中所有已共享的媒体流，并调用它的 `addStream` 方法，这样它就可以订阅 `Room` 中所有的媒体流了。订阅成功后，媒体流的数据就被源源不断地发送给 `Participant` 对应的终端了。

上面的逻辑完成之后，服务端会通知客户端 join 已经成功。此时，客户端也可以共享它的音视频流给服务器了。

此外，在客户端将自己的媒体流推送给服务端之前，服务器会调用 `publishStream` 方法在服务端创建 `IncomingStream` 实例，这样当客户端的数据到达服务器端后，就可以将数据交给 `IncomingStream` 暂存起来，以备后面分发给其他终端。

通过上面的步骤，新加入的客户端就可以看到会议中其他人分享的音视频流了，同时还可以将自己的音视频推送到服务器了。但此时，其他参会人还看不到新加入用户的音视频流，因为还有关键的一步没有完成。

这关键的一步是，当新参会人一切准备就绪后，服务端要发送广播消息（`update`）通知所有已经在会中的其他人，有新用户进来了，需要重新进行媒体协商。**只有重新媒体协商之后，其他参会人才能看到新入会人的音视频流。**

以上步骤就是一个新参会者加入会议的大致过程，同时也介绍了这个过程中其他已经在会中的参与人需要如何处理。

小结

本文我们首先介绍了通过 Medooze SFU 如何搭建最简单的音视频会议系统，该系统搭建好后就可以进行多人音视频互动的“实验”了。

然后，我们对 Medooze 的 SFU 项目的逻辑做了详细介绍，分析了 Medooze SFU 的基本架构、逻辑处理、入会流程等。相信通过本文的学习，你对 Medooze 的使用和实现原理都会有更深刻的认识。

当然，如果你要想在自己的产品中更好地应用 Medooze，还需要掌握更多的细节，只有这样才能做到心中有数。比如：如何能够承担百万用户级的负载？如何为不同的地区、不同的运营商的用户提供优质的服务？这些都是你需要进一步研究和学习的。

思考时间

今天留给你的思考题：一个多人存在的会议中，又有一个人加入进来，其他参会人是如何看到后加入人的视频流的呢？你清楚其中的原理吗？

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。



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 28 | 让我们一起探索Medooze的具体实现吧（下）

下一篇 30 | 实战演练：通过WebRTC实现多人音视频实时互动直播系统

精选留言 (2)

写留言



accessory

2019-09-21

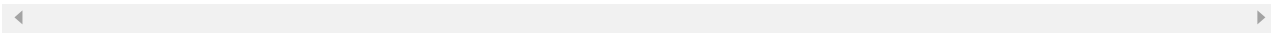
/home/toney/source/sfu/lib/Room.js:59

```
this.activeSpeakerDetector = MediaServer.createActiveSpeakerDetector();  
^
```

TypeError: MediaServer.createActiveSpeakerDetector is not a function...

展开

作者回复: 进入到 sfu，重新执行npm install 看看会报错吗



1



frank

2019-09-19

不懂nodejs，大致看了下，应该是participant.on('stream' 这边，但是stream怎么触发就不知道了

作者回复: 再仔细看看代码哈，这个知识点非常关键！

