30 | 实战演练：通过WebRTC实现多人音视频实时互动直播系统

2019-09-21 李超

从0打造音视频直播系统 进入课程 >

讲述：李超
时长 19:04　大小 15.29M

关于通过 WebRTC 实现多人音视频实时互动的实战，其实我们在上一篇文章中已经向你做过详细介绍了，其中包括如何编译 Medooze 源码、如何将编译出的 Medooze SFU 进行布署，以及如何去使用等相关的内容。

那么今天我们再从另外一个角度来总结一下 Medooze 是如何实现多人音视频互动的。

下面我们就从以下三个方面向你做一下介绍：

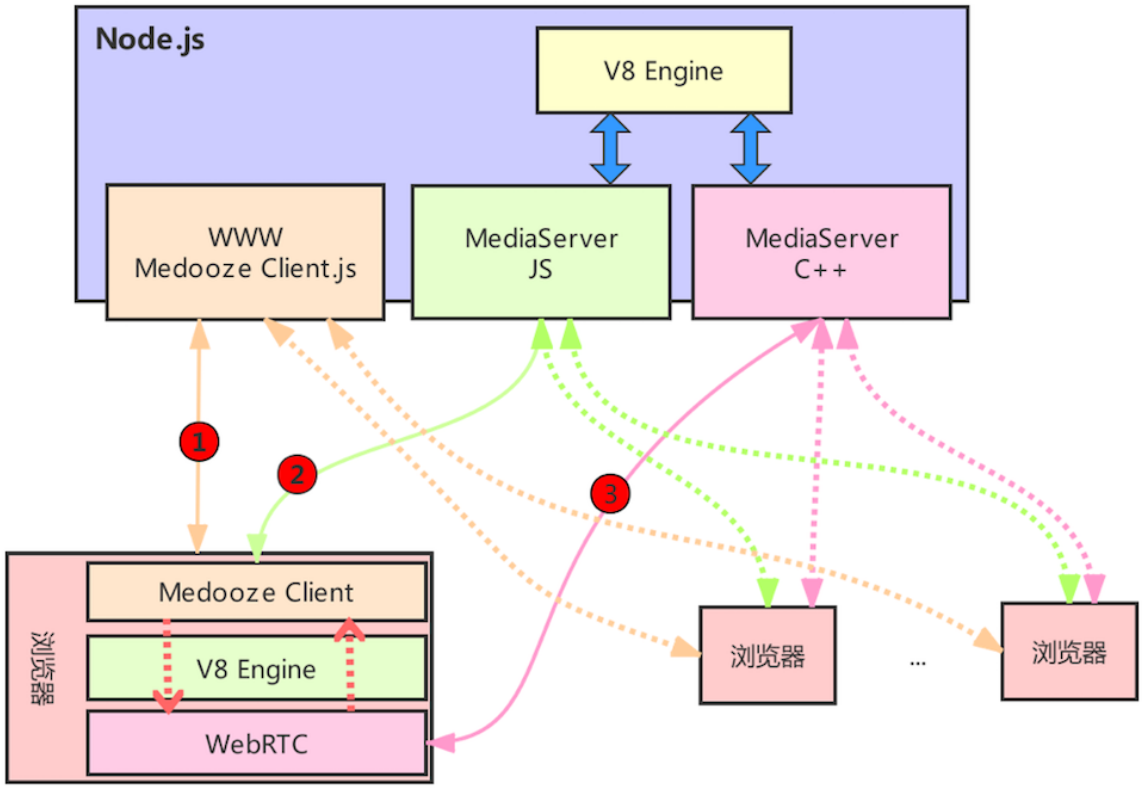
首先是多人音视频会议整体结构的讲解，这会让你从整体上了解利用 Medooze 搭建的流媒体服务器与 WebRTC 客户端是如何结合到一起运转的；

其次再对 WebRTC 客户端进行说明，你将知道无论是 Medooze 还是使用其他的流媒体服务器，对于客户端来讲它的处理流程基本是不变的；

最后是 Medooze 服务器的总结，让你了解 Medooze 各模块是如何协调工作的，实际上这部分知识我们在之前的文章中已经做过介绍了，但为了让你对它有更深刻的认识，这里我们还会从另外一个角度再重新剖析。

Medooze 整体架构

文中接下来这张图清晰地展示了 Medooze 是如何实现多方通信的，你可以先参考下：



Medooze 多方通信整体结构图

从这张图你可以看到，它主要分成两大部分：**服务端**和**客户端**。下面我们就从这两个方面向你详细描述一下 Medooze 实现多方通信的过程。

首先，我们来看一下**服务端都提供了哪些功能**。实际上，服务端主要提供了三方面的功能：

1. 提供了 HTTP/HTTPS 的 WWW 服务。也就是说，我们将浏览器执行的客户端代码（JavaScript）放在 Node.js 的发布目录下，当用户想通过浏览器进行多方通信时，首先会向 Node.js 发起请求，将客户端 JavaScript 代码下载下来。然后，再与 Node.js 中的 WebSocket 服务连接。

2. 提供了 WebSocket 服务，主要用于信令通信。在我们这个系统中涉及到的信令并不多，有 join、update 以及 WebRTC 需要交换的 Offer/ Answer、Candidate 等几个简单的信令。对于这几个信令的具体作用，我们会在本文的后半部分（WebRTC 客户端）向你做详细介绍。
3. 提供了音视频流数据转发的能力。这块逻辑是使用 C++ 语言实现的，包括了 WebRTC 协议栈的实现、数据流的转发、最大带宽评估、防拥塞控制等功能。

接下来，我们就**从下载客户端代码开始，详述一下 WebRTC 客户端与 Medooze 交互的整个过程。**

当进行多方通信时，用户首先要向 Node.js 发起 HTTP/HTTPS 请求，然后从 Node.js 服务器上获取浏览器可以运行的 JavaScript 代码。浏览器获取到客户端代码后交由浏览器的 V8 引擎进行解析，然后调用 WebRTC 库相关的 API，从本地摄像头和麦克风采集音视频数据，以便后面将它们分享到 Medooze 服务器上，这就是我们上图中**用红圈标出的第 1 步。**

当然，在正式分享音视频数据之前，客户端的 JavaScript 还要与 Node.js 之间建立 WebSocket 连接，也就是上图中**用红圈标出的第 2 步。**在 Medooze 多方通信实现过程中，WebSocket Server 是由 Medooze 的 JavaScript 脚本来实现的，当浏览器中的客户端与 Node.js 中的 WebSocket 服务建立好连接后，客户端就会发送 **join** 信令到服务端。

服务端收到 join 信令后，会在它的管理模块中查找 join 要加入的房间是否存在，如果该房间已经存在，则将该用户加入到房间内；如果不存在，则在它的管理系统中创建一个新的房间，而后再将该用户加入到房间里。成功加入房间后，服务端会调用 media-server-node 中的 createEndpoint 接口，最终在 C++ 层打开一个 UDP 端口为传输音视频数据做准备。

当前面这些准备工作就绪后，客户端与服务端开始通过 WebSokcet 通道交换 SDP 信息，包括非常多的内容，比如双方使用的编解码器、WebRTC 协议栈、Candidate 等，除此之外，还包括 DTLS-ICE 相关的信息，通过这些信息客户端与服务器之间就可以建立起数据连接了。

接下来，我们再简要描述一下 **WebRTC 客户端与服务端之间是如何建立起 DTLS-ICE 连接的。**

在双方交换了各自的 SDP 信息后，客户端就拿到了连接服务器的用户名和密码，即 ice-ufrag 和 ice-passwd 两个字段。其中， ice-ufrag 是用户名，ice-passwd 是连接的密码。当客户端与服务端建立连接时，它要通过 STUN 协议向 Medooze server 发送 Request 消息，并带上 ice-ufrag 和 ice-passwd 这两个字段。 Medooze 服务端收到 STUN Request 消息后，将连接的用户名和密码与自己保存的用户名和密码做比较，如果一致，则说明该连接是有效的，这样客户端与服务端连接就建立起来了，并且后面所有从该客户端发到服务端的 UDP 数据包一律放行。

连接建立好后，为了保证数据安全，需要在客户端与服务器之间建立安全机制。这个安全机制就是我们在专栏第一模块分析“如何保障数据传输的安全”时所介绍的**证书和公钥的交换**。这一步执行完成之后，通信的双方就可以彼此传输音视频数据了。以上就是上图中**红圈标出的第 3 步**。

服务器在收到客户端发来的音视频数据后，并不能直接使用，还需要使用 SRTP 协议将加密后的数据进行解密，然后才能做后面的逻辑处理。

在音视频数据传输的过程中，通信双方一直对传输的网络进行着监控。因为网络是变化的，可能刚刚网络质量还特别好，过一会儿就变得很差了。所以，无论是 Medooze 还是其他的流媒体服务器，对于网络质量的监控都是一项必备功能。**监控的方法也比较简单，就是通过 RTCP 协议每隔一段时间就上报一次数据**。上报的内容包括收了多少包、丢了多少包、延迟是多少等这些基本信息。有了这些基本信息后，在服务端就可以评估出目前网络的带宽是多少了。

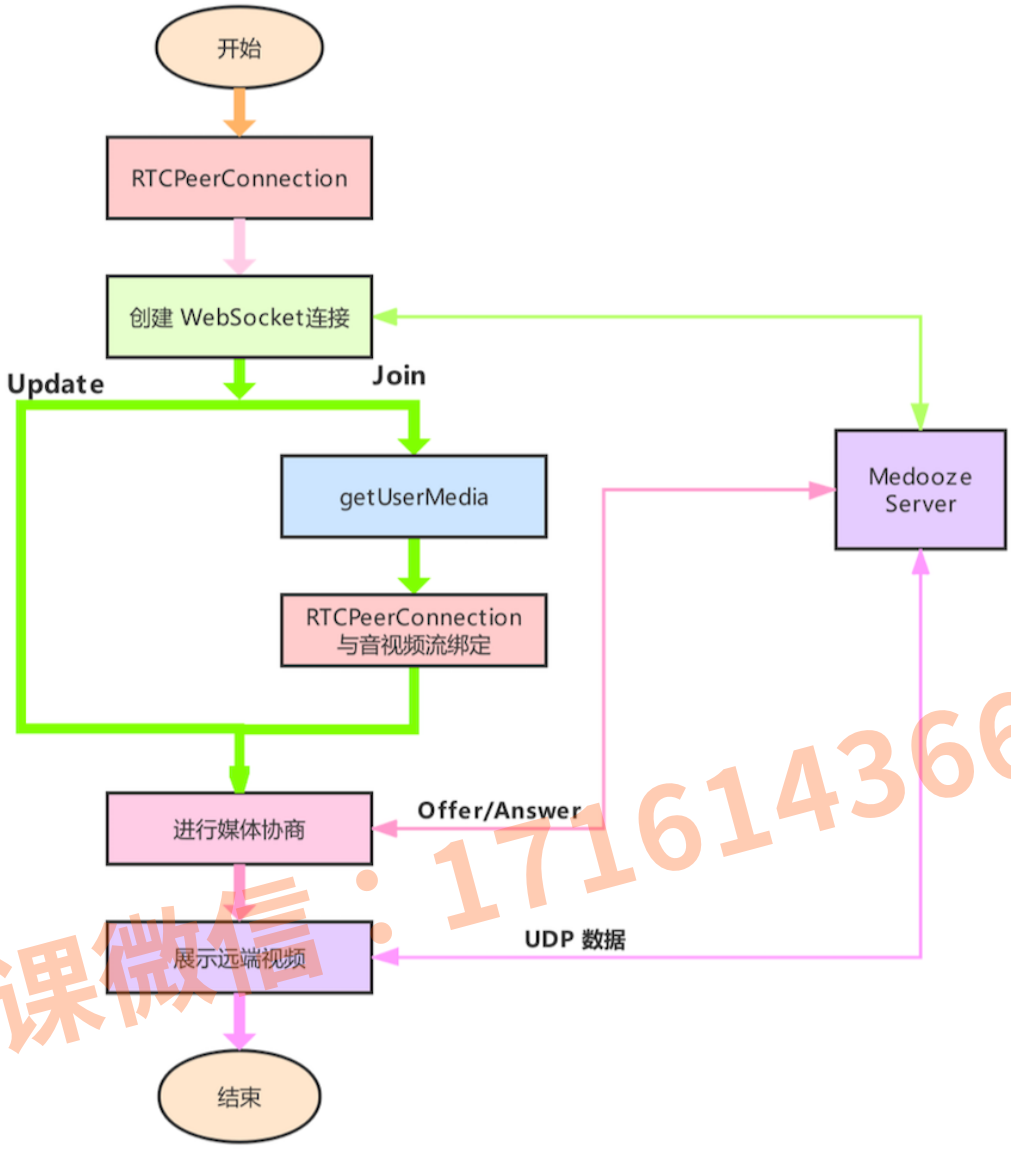
以上的描述是其中一个参会人加入到房间里的情况，而对于多人来说，每个加入到会议里的人，都要做上面相同的逻辑。

以上就是 Medooze 多方通信整体结构的描述，接下来我们再分别从客户端和服务端的角度详述它们都做了哪些事儿吧。

WebRTC 客户端

对 WebRTC 客户端来讲，无论是 1 对 1 通信还是多方通信，区别并不大。主要的逻辑，像采集音视频数据、创建 RTCPeerConnection、媒体协商等，都是一样的。只有在展示的时候稍有差别，在 1 对 1 通信中，只需要显示两个视频——本地视频和远端视频，而多方通信时则要展示多个视频。可以说这是 1 对 1 通信与多方通信在客户端的唯一差别了。

文中接下来这张图是我总结的 WebRTC 客户端在多方通信中的流程图，你可以参考下：



WebRTC 客户端流程图

从图中可以看出，WebRTC 客户端在多方通信中的基本处理逻辑是这样的：

首先创建 RTCPeerConnection 对象，用于与服务端传输音视频数据；

紧接着客户端与服务端建立 WebSocket 连接，建立好之后，双方就可以进行信令通信了；

当用户发送 join 消息给服务器，并成功加入到房间之后，客户端就可以调用 getUserMedia 进行音视频数据的采集了；

数据采集到之后，要与之前创建好的 RTCPeerConnection 进行绑定，然后才能通过 RTCPeerConnection 实例创建 Offer/Answer 消息，并与服务器端进行媒体协商；

媒体协商完成后，客户端就可以将音视频数据源源不断发送给 Medooze 服务器了；

当有其他 WebRTC 客户端进入到房间后，它们的音视频流也会通过 RTCPeerConnection 传送给早已加入到房间里的 WebRTC 终端，此时 WebRTC 终端会收到 onRemoteTrack 消息，然后创建 HTML5 的`<video>`标签将它们显示出来就好了。

除此之外，在上图中，你应该还注意到一个 **update 信令**，这个信令在多方通信中是**至关重要**的。下面我们就来讲解一下 update 信令的作用。

在进行多方通信时，第一个 WebRTC 客户端已经加入到房间里了，接着第二个用户开始加入。当第二个用户加入时，它可以获得第一个用户共享的音视频流，但对于第一个用户来说，它是否能获得第二个用户的音视频流呢？显然获取不到。获取不到的原因是什么呢？因为第一个用户与服务器之间进行媒体协商时，它还不知道有第二个用户，这样当第二个用户进来时，如果不与服务器重新进行媒体协商的话，它是不知道房间里已经有其他人共享了音视频流的。

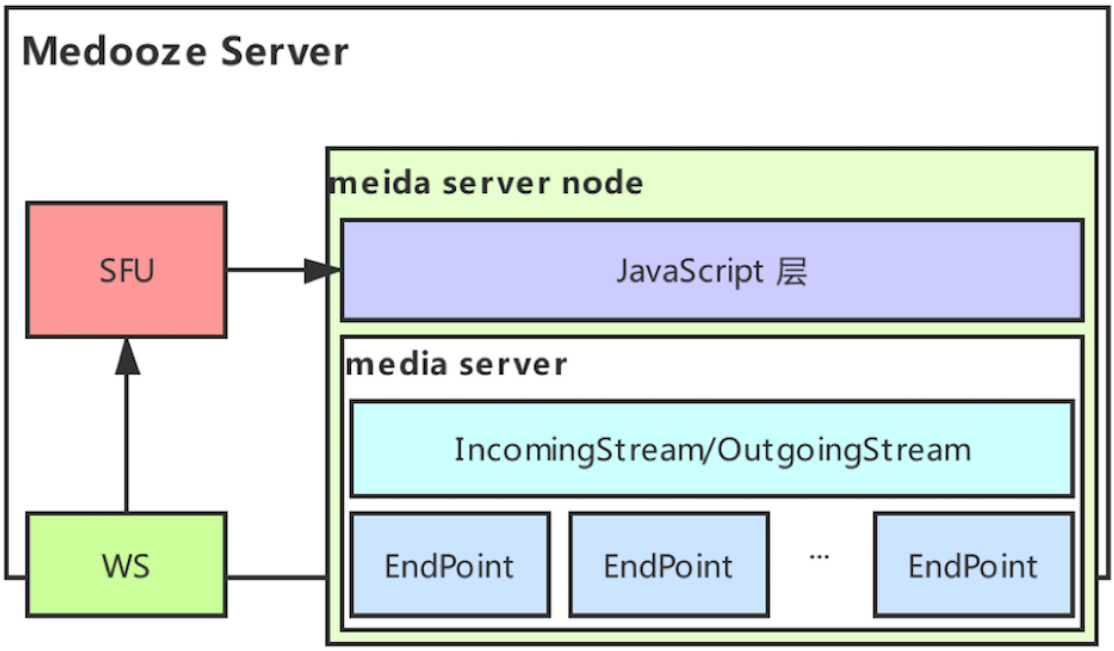
那如何解决这个问题呢？解决的办法就是**每当有新的用户进来之后，就通过 update 信令通知已经在房间内的所有用户，让它们重新与服务器进行媒体协商。**重新协商后，所有老用户就可以收到了新用户的视频流了。

以上就是客户端的基本处理逻辑。通过上面的讲解你可以了解到，无论是多人互动还是 1 对 1 通信，对于客户端来说基本逻辑都差不多。

WebRTC 客户端处理流程讲完后，接下来我们再来看看服务端。

服务端

文中接下来这张图是我总结的服务端架构图，你可以参考下：

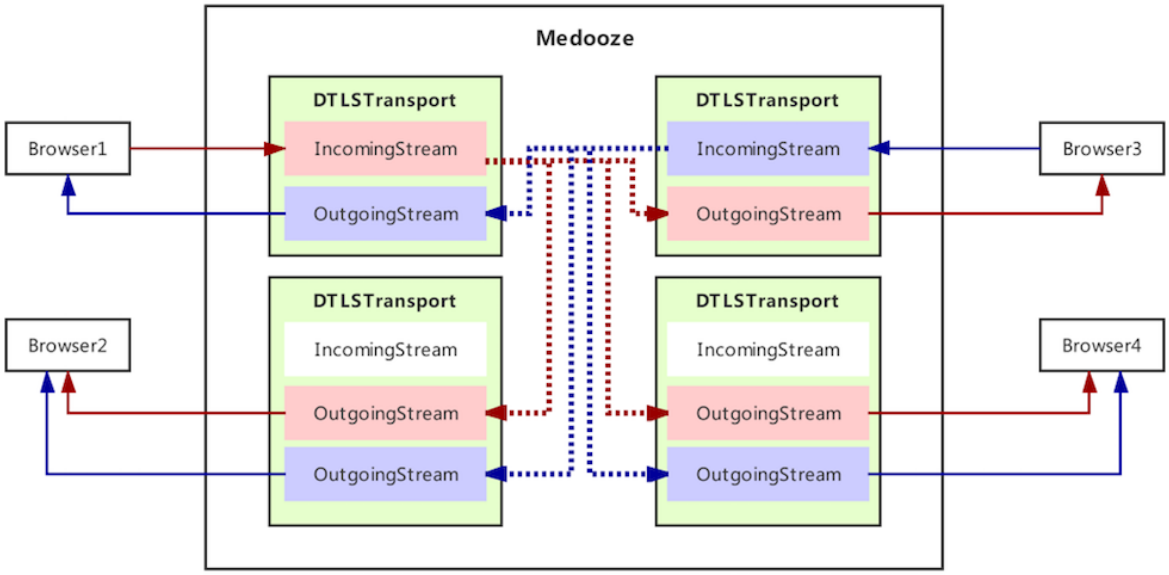


服务端架构图

如上图所示，从大的方面来讲，服务端由三部分组成，即 WebSocket 服务、SFU 逻辑处理以及媒体服务器。如果再细化的话，媒体服务器又是由 media-server-node 和 media-server 组成。

IncomingStream 和 OutgoingStream 是 media-server 的核心。就像之前的文章向你介绍的，你可以将 IncomingStream 和 OutgoingStream 看成是两个队列，其中 **IncomingStream 用于存储接收到的数据，而 OutgoingStream 用于存放要发送的数据。**

在一个会议中，每个参会人一般只有一个 **IncomingStream 实例**用于接收该参会人共享的音视频流。但却可以有多个 **OutgoingStream 实例**，这每个 OutgoingStream 实例表示你订阅的其他参会人共享的音视频流。如下图所示，Browser2 和 Browser4 都在两路 OutgoingStream 实例。



IncomingStream/OutgoingStream 示意图

上面这张图，我们在《27 | 让我们一起探索 Medooze 的具体实现吧（上）》一文中已经向你做过详细描述了，这里就不再赘述了。如果你记不清了，可以再去重温一下这篇文章。

通过 Medooze Server 架构图，你可以看到它有**两个网络接入点**：一个是信令接入点，也就是就是**WebSocket 服务**；另一个是音视频数据接入点，即 **Endpoint**。下面我们就对这两个接入点做一下介绍。

WebSocket 服务做的事件比较简单，就是用来收发信令。像上面“WebRTC 客户端”部分所介绍的那样，在 Medooze 的 SFU Demo 中，它定义的信令消息特别少，只有 join、update 以及 WebRTC 用于媒体协商的 Offer/Answer 等几个消息。

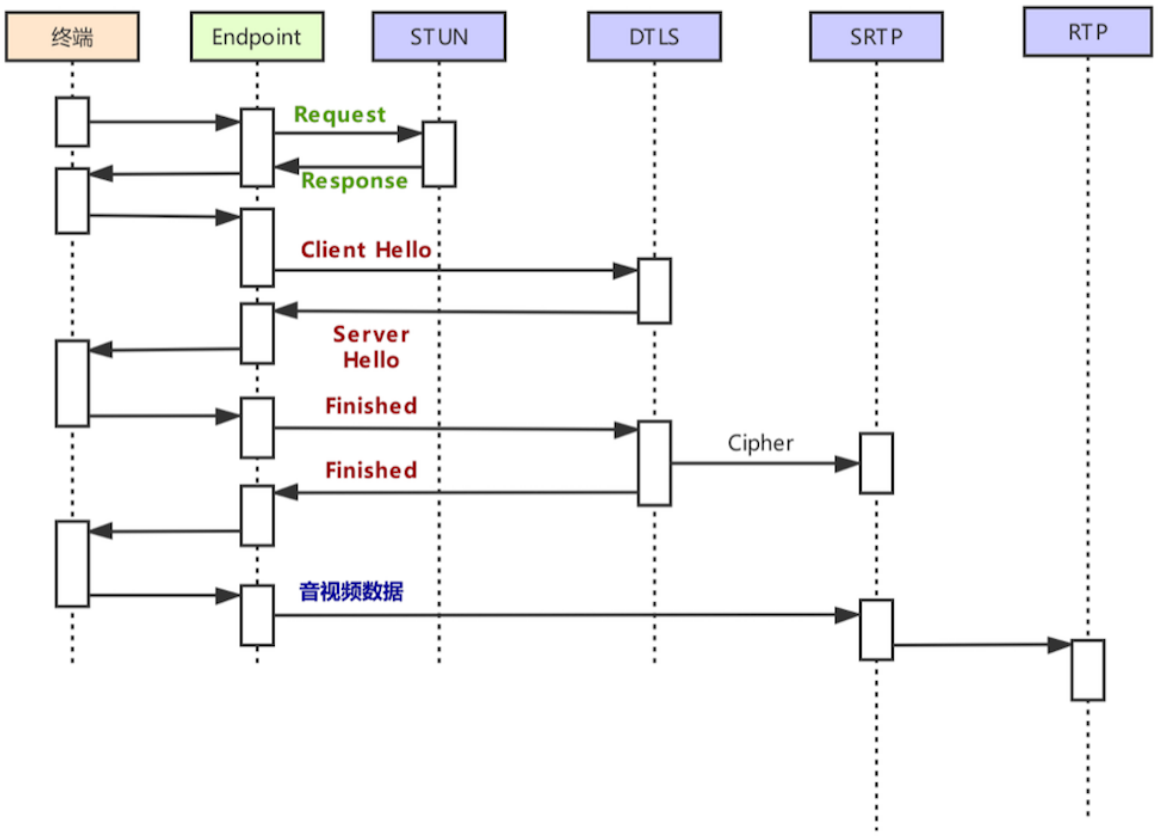
而每个 **Endpoint** 实际上就是一个 `<IP，port>` 对，它使用的是 UDP 协议。对 Medooze 服务端的每个房间，都会为之创建一个独立的 Endpoint，用来接收 / 发送音视频数据。Endpoint 接收的数据要复杂很多，除了音视频数据外，还有 STUN 协议消息、DTLS 协议消息、RTCP 协议消息等。所以如果你想将 Medooze 流媒体服务器理解透彻，就要花大量的时间先将上面这些协议搞清楚、弄明白。说实话，这可不是一朝一夕的事儿。

另外，为了使服务器在处理网络 IO 时更高效，Medooze 使用异步 IO 的方式来管理 EndPoint。也就是使用的 poll API 对 Endpoint(socket) 做处理。关于这一点我们在《28 | 让我们一起探索 Medooze 的具体实现吧（下）》一文中已经做了全面的介绍，如果记不清了，你可以再重新回顾一下那篇文章。

当然 Medooze 使用 poll 来处理异步 IO 事件性能还是低了点，其实更好的方式是使用 epoll API，这个 API 在 Linux 下是最高效的；或者可以使用开源的异步 IO 事件处理库，如 libevent/libuv 等，因为它们底层也是使用的 epoll。

接下来，我们再来看一下**服务端通过 Endpoint 从客户端收到数据包后的处理逻辑是怎样的。**

像前面我们介绍过的，从 Endpoint 收到的数据包包括 STUN 消息、DTLS 消息、RTCP/SRTCP 消息、音视数据等多种类型。所以 Medooze 后面的处理逻辑非常复杂，涉及到很多细节，这需要你一点一点去理解。但大体的脉落还是比较清晰的，如下图所示：



数据包时序图

上图就是 Medooze 处理各种数据包的时序图。客户端首先发 STUN Request 包，服务端收到该请求包后，对客户端的身份做验证，检查该客户端是否是一个合法用户，之后给客户端回 Response 消息。

如果客户端是合法用户，则发送 DTLS 消息与服务端进行 DTLS “握手”。在“握手”过程中要交换双方的安全证书和公钥等相关信息，也就是时序图中的 Client Hello、Server

Hello、Finished 等消息。

证书交换完成后，服务端和客户端都获得了协商好的加密算法等信息，此时就可以传输音视频数据包了。服务端在收到数据包后还要调用 SRTP 库进行解密，也就是将 SRTP 消息解密为 RTP 消息，最终获取到用户的 RTP 音视频消息后，就可以进行后面的操作了。

以上就是 Medooze 流媒体服务器大致的处理逻辑。

小结

本文我们从 Medooze 的整体结构、WebRTC 客户端处理流程以及 Medooze 流媒体服务器三个方面向你详细介绍了 WebRTC 是如何实现多人实时互动直播的。这篇文章是一个总结性的文章，其主要目的是带你梳理一下我们前面所讲的知识点，通过本文你会对 Medooze 实现多方通信有一个全局的理解。

另外，Medooze 实现多方通信涉及到很多细节，需要你花大量的时间去阅读代码才可以将它们弄明白、搞清楚，而我们专栏中的几篇文章就是给你讲清楚 Medooze 在实现多方通信时的主脉络是什么，以便让你更加容易地去理解 Medooze 的代码实现。

在阅读 Medooze 源码时，建议你先按照上一篇文章中的步骤将 Medooze Demo 环境搭建出来，然后再以 Medooze 分析的几篇文章为主线对 Medooze 的源码进行阅读和实验，这样你很快就可以将 Medooze 的细节搞清楚了。

至此，我们专栏的第二模块“WebRTC 多人音视频实时通话”就讲解完了。

思考时间

今天我留给你的思考题是：Medooze 为什么要使用 STUN/DTLS 等协议？可以不用吗？

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。

上一篇　29 | 如何使用Medooze 实现多方视频会议?

精选留言 (1)

写留言

Derek
2019-09-21

实现p2p和媒体数据加密传输不是都得用吗?

展开

作者回复: 不是，没说到关键点