# Webrtc

## 流媒体概念

所谓流媒体是指采用流式传输的方式在 Internet 播放的媒体格式。   
 流媒体又叫流式媒体，它是指商家用一个视频传送服务器把节目当成数据包发出，传送到网络上。用户通过解压设备对这些数据进行解压后，节目就会像发送前那样显示出来。

流媒体以流的方式在网络中传输音频、视频和多媒体文件的形式。流媒体文件格式是支持采用流式传输及播放的媒体格式。  
流式传输方式是将视频和音频等多媒体文件经过特殊的压缩方式分成一个个压缩包，  
由服务器向用户计算机连续、实时传送。在采用流式传输方式的系统中，用户不必像非流式播放那样等到整个文件全部下载完毕后才能看到当中的内容，而是只需要经过几秒钟或几十秒的启动延时即可在用户计算机上利用相应的播放器对压缩的视频或音频等流式媒体文件进行播放，剩余的部分将继续进行下载，直至播放完毕。

## 几种流媒体格式

### http flv

这两种也是可以播放视频的,只不过必须要整个视频都下载完才能播放.

### RTP RTCP

Real-time transport protocol

是用于internet针对多媒体数据的一种传输层协议,RTP和RTCP一起使用,后面的是控制协议.

他是建立在UDP协议之上的,基于udp的话那就是数据会丢失咯

服务端是以固定的速度在网上发送数据,客户端也是以一种固定的速率看视频,不能快进啥的,观看完了还得重新下载.

属于仅仅实现了视频流基本功能的东西.

### RTSP

Real-time stream protocol

rtsp就是在rtp那套上又加了一层功能, 可以使客户端和服务端有交互,进而也就是实现了快进和快退的功能.

而且可以支持udp,也可以支持tcp,(在rtp基础上增加了支持tcp的功能)

### RTMP

Real-time message protocol

视频直播协议 是基于tcp的,数据不会丢失

这个协议就是为了直播而出的, 所以那种多人直播,万人直播都是基于rtmp的.

RTMP 协议也要客户端和服务器通过"握手"来建立 RTMP Connection，然后在Connection上传输控制信息。RTMP 协议传输时会对数据格式化，而实际传输的时候为了更好地实现多路复用、分包和信息的公平性，发送端会把Message划分为带有 Message ID的Chunk，每个Chunk可能是一个单独的Message，也可能是Message的一部分，在接受端会根据Chunk中包含的data的长度，message id和message的长度把chunk还原成完整的Message，从而实现信息的收发.

而且rtmp的cdn做的比较好,一个主播向一个cdn服务器发流消息,cdn服务再带给其他的就可以支持万人直播了.

### HLS

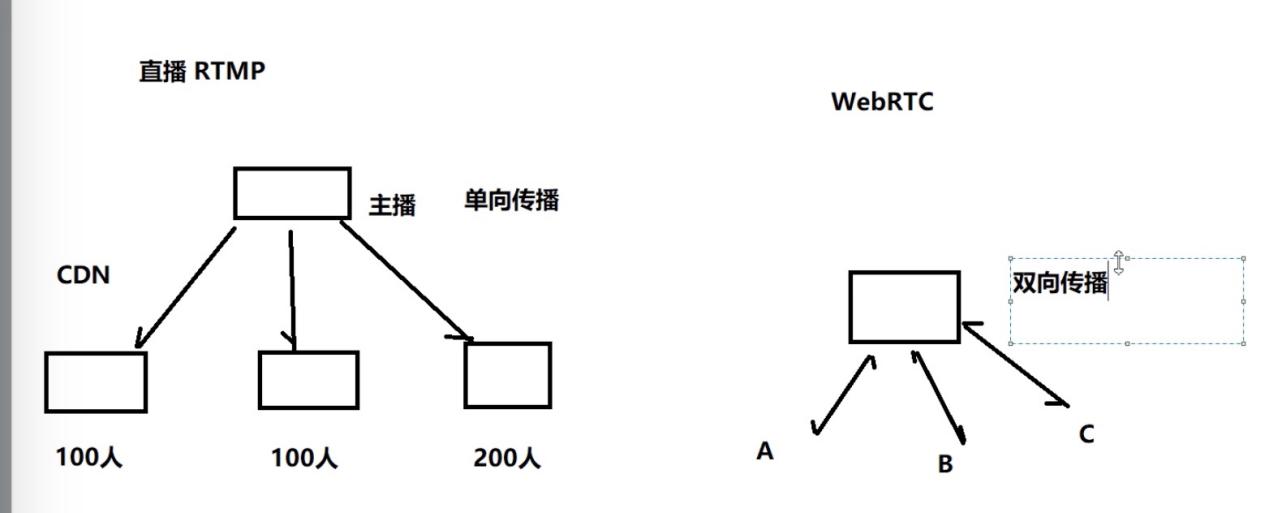
http living stream 可以从名字上看出来是基于http的.

上面的那几个都是基于传输层的,而这个是基于http的,苹果公司开发的基于http的流媒体协议

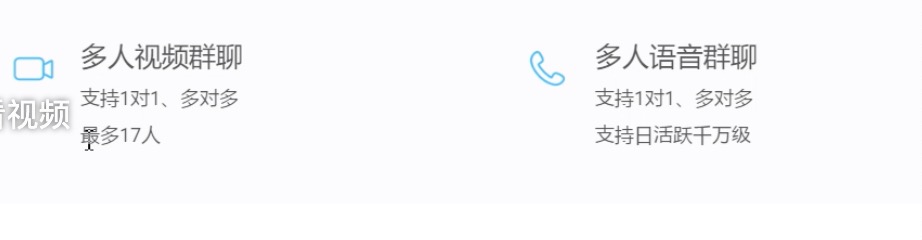
所以也就是ios用的比较多,专门为苹果手机和平板这种移动端准备的.

HLS 点播，基本上就是常见的分段HTTP点播，不同在于，它的分段非常小。  
相对于常见的流媒体直播协议，例如RTMP协议、RTSP 协议、MMS 协议等，HLS 直播最大的不同在于，直播客户端获取到的，并不是一个完  
整的数据流。  
HLS 协议在服务器端将直播数据流存储为连续的、很短时长的媒体文件(MPEG-TS格式)，而客户端则不断的下载并播放这些小文件，  
因为服务器端总是会将最新的直播数据生成新的小文件，这样客户端只要不停的按顺序播放从服务器获取到的文件，就实现了直播。由此可见，基本上可以认为，HLS 是以>>点播的技术方式来实现直播<<。由于数据通过 HTTP 协议传输，所以完全不用考虑防火墙或者代理的问题,而且分段文件的时长很短，客户端可以很快的选择和切换码率，以适应不同带宽条件下的播放。不过HLS的这种技术特点，决定了它的延迟一般总是会高于普通的流媒体直播协议。

### webrtc



可以看到webrtc是双向的, rtmp是单向的,万人直播基本都是rtmp,cdn支持的比较好.

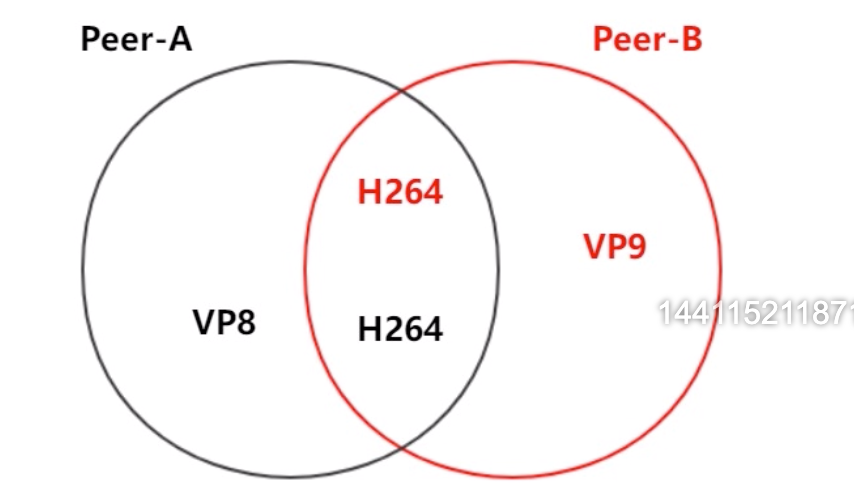


可以看到webrtc基本是并发比较小的.

H5的基本标准.

## webrtc原理

### 媒体协商 sdp



VP8 VP9 H264这些都是编解码格式.

有一个专门的协议 session desripiton protocol SDP, 这个协议是专门用来描述支持的音视频信息的,所以也叫会话描述协议.

在webrtc当中,会话双方必须先交互sdp信息,才能知根知底,交换sdp的过程就叫做媒体协商.

### 网络协商 candidate

现实生活中我们的电脑都是在局域网中,对外交互需要NAT才可以.

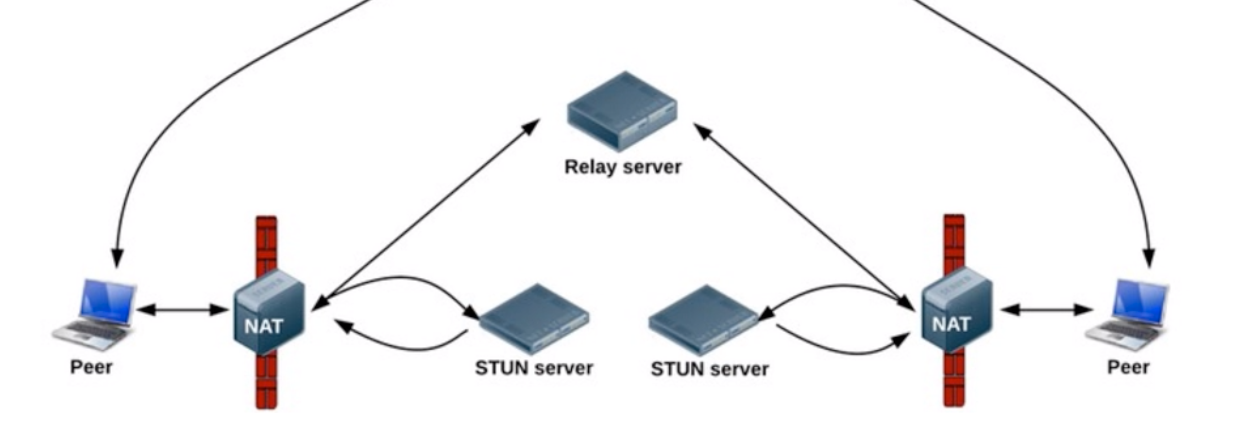
我们知道nat的机制是通过公网的ip+端口,映射到局域网的ip的.

这层映射关系我们可以说是维护在家里的路由器上的,而我们的客户端是不知道的,但是客户端是需要告诉我们的webrtc对端知道的,所以我们的客户端需要知道自己在外网上是用里什么nat,和自己的公网ip加端口.

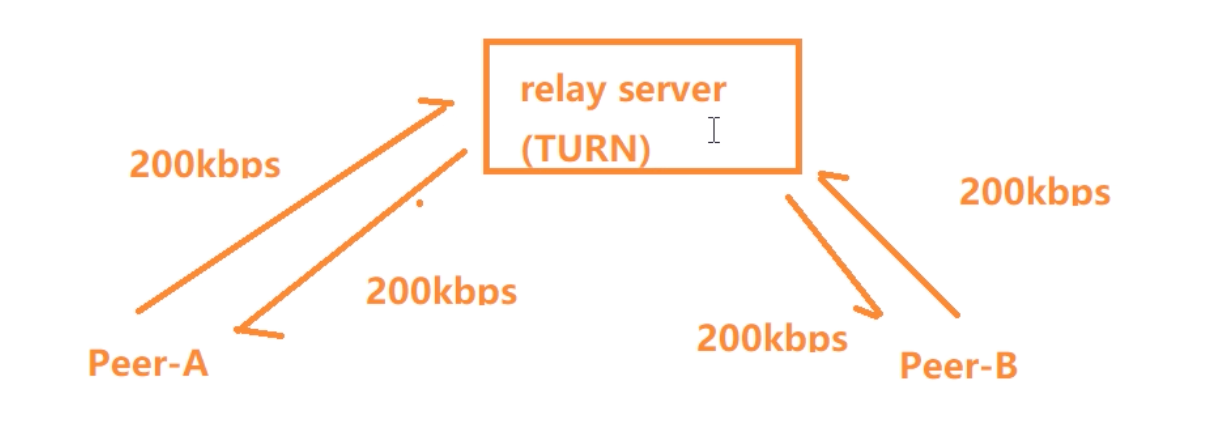
有个协议叫做STUN,他可以让客户端知道自己的nat之后在公网上的ip和端口,还有种服务器叫做STUN服务器,因为有很多种类型的nat,STUN服务器可以适配多种类型的nat,让客户端获取自己的公网信息.

还有个服务器叫做TURN,是这样的,即使两个客户端都知道了对方的ip和端口,但是也可能链接不上,就需要一个服务器做为中继和转发,这个服务器就是TURN.

那么整体架构就是这个样子了



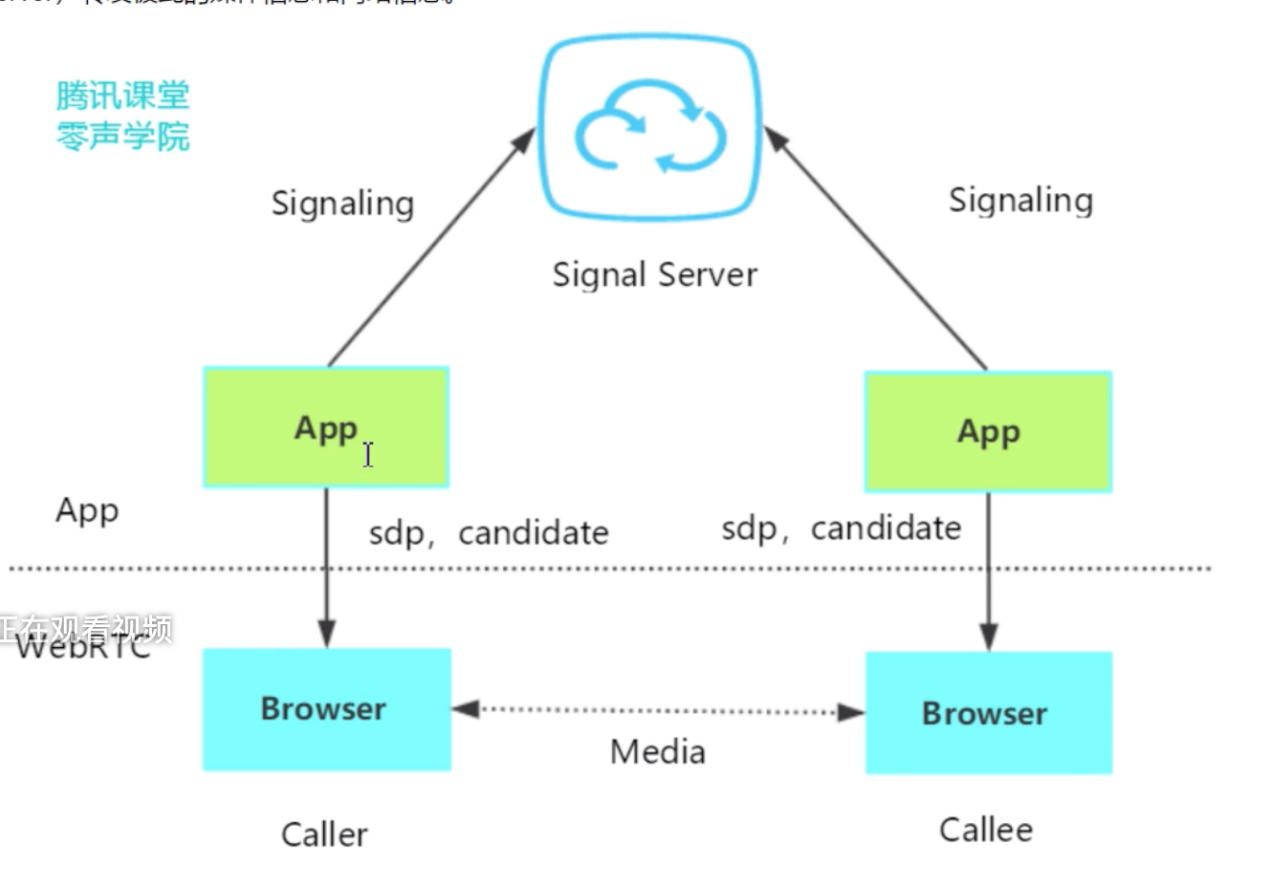
这样的话如果多人音视频通话的话中继服务器的压力就很大了.



可以看出,中继的方式是很耗费带宽的,单向的音频流大约是200kb每秒的带宽.

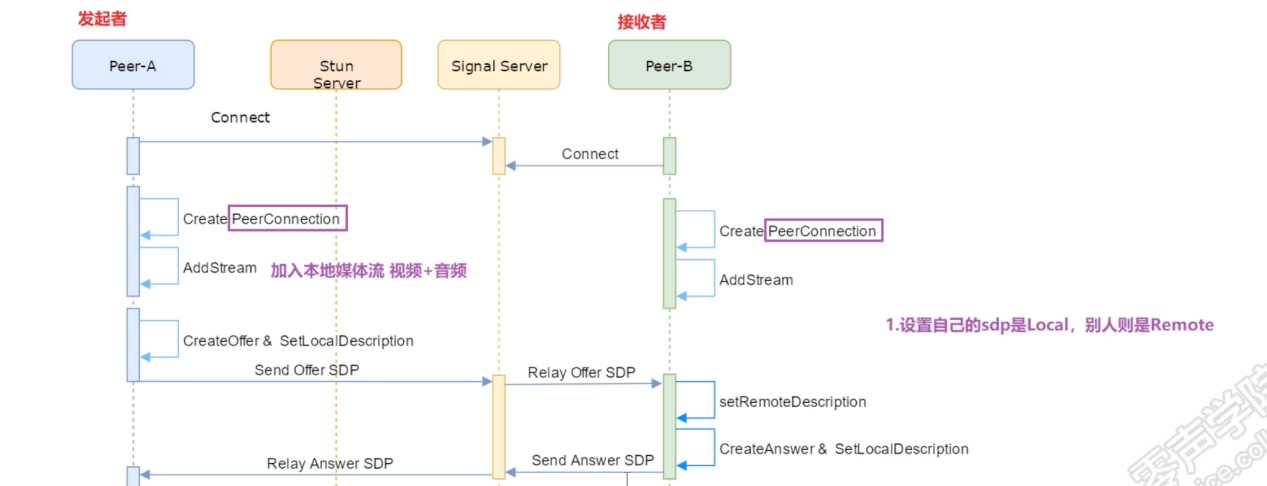
### 信令服务器 singal server

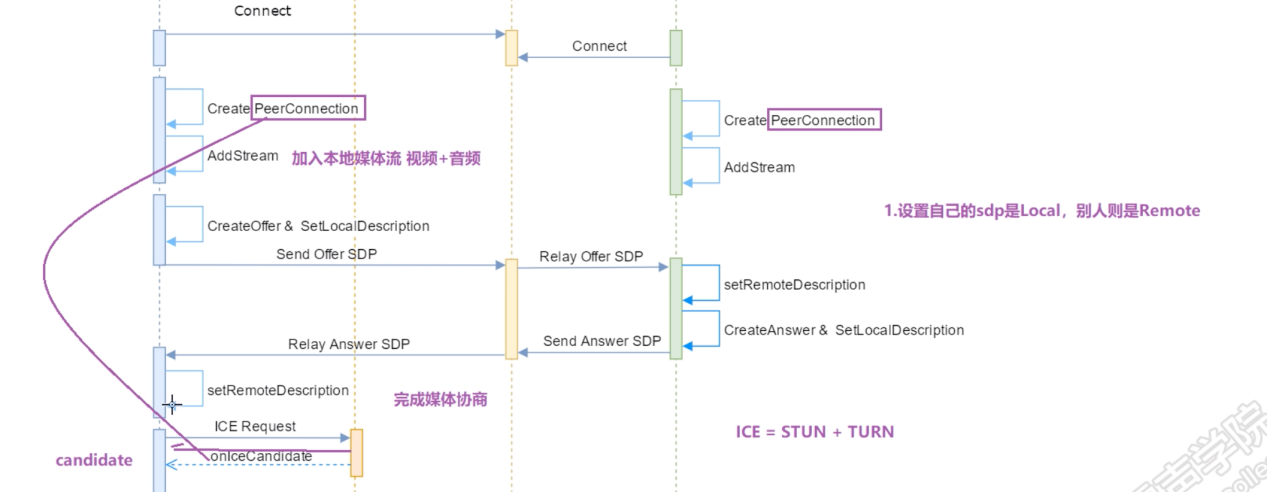
由于上面有两种信息需要交换,所以需要一个信令服务器.

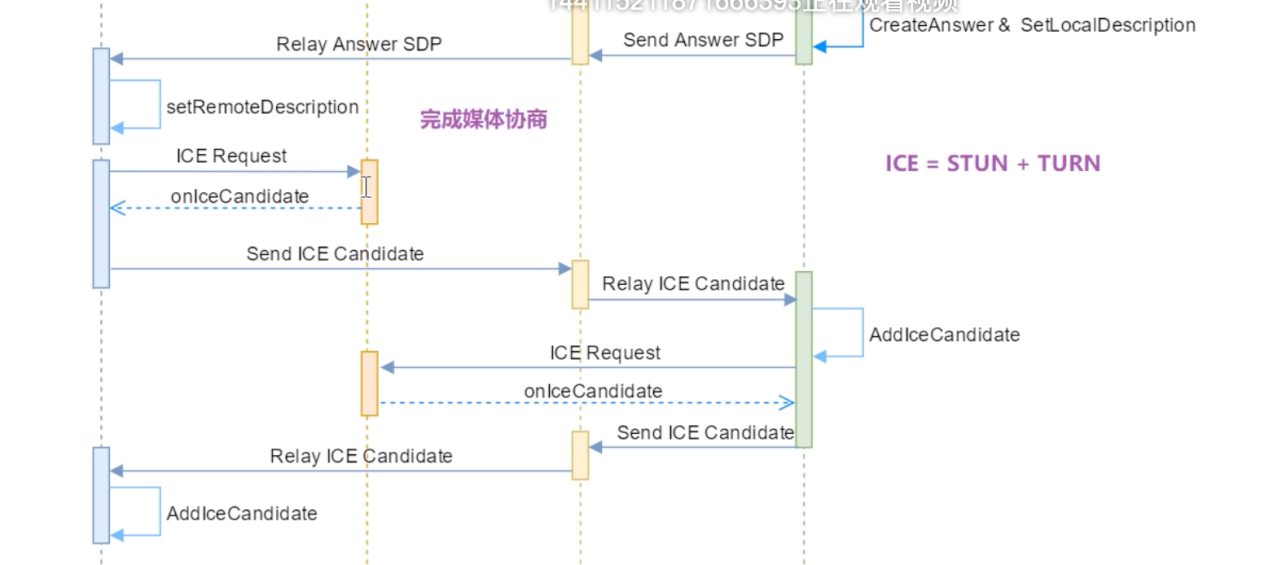


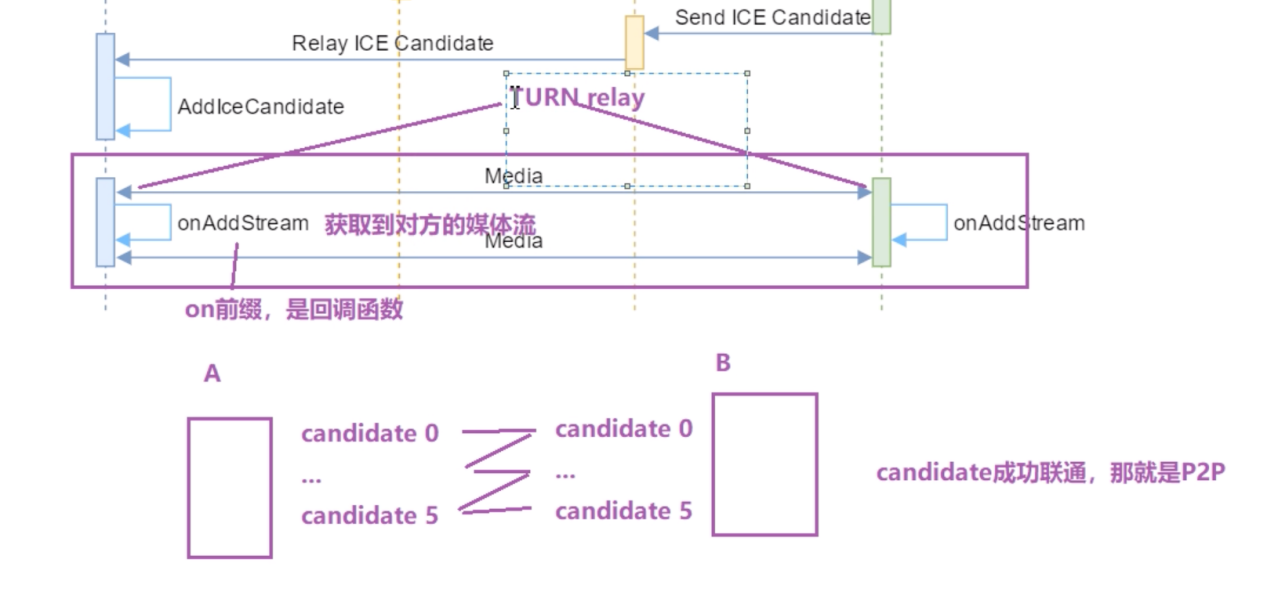
当然除了sdp和candidate还可以交换一些房间管理,也就是业务方面的信息管理,就跟classroom服务似的.

### 一对一时序图









图中可以看到,获取到sdp,通过信令服务发送到对端,通过stun服务获取到自己的candidate,再通过信令服务发送到对端,链接弄好后,media数据走turn的中继relay.

### Websocket

H5提供的一种基于单个tcp链接的全双工通讯的协议.

互相都可以主动发来发去,完成一次握手后就可以创建持久性的链接.

websocket是一个协议,js node go都有自己的封装方式互相之间都是可以互相直接通信的.

无非就是一个websocket的对象呗,参数就是ip和port.

里面有很多回调, 什么

Onclose onCon onMessage 和send什么的

server还可以foreach获取到自己所有的con

1 var ws = require("nodejs‐websocket")

2

3 // Scream server example: "hi" ‐> "HI!!!"

4 var server = ws.createServer(function (conn) {

5 console.log("New connection")

6 conn.on("text", function (str) { // 收到数据的响应

7 console.log("Received "+str)

8 conn.sendText(str.toUpperCase()+"!!!") // 发送

9 })

10 conn.on("close", function (code, reason) { // 关闭时的响应

11 console.log("Connection closed")

12 })

13 conn.on("error", function (err) { // 出错

14 console.log("error:" + err);

15 });

16 }).listen(8001)

‘通过这玩意就可以搞一个类似的im系统了.

## 实战

### 打开摄像头

<html >

9 <body >

10 <video id="local‐video" autoplay playsinline></video>

11 <button id="showVideo" >打开摄像头</button>

12 <p>通过getUserMedia()获取视频</p>

13 </body>

14 <script >

15 const constraints = {

16 audio: false,

17 video: true

18 };

19

20 // 处理打开摄像头成功

21 function handleSuccess(stream) {

22 const video = document.querySelector("#local‐video");

23 video.srcObject = stream;

24 }

25

26 // 异常处理

27 function handleError(error) {

28 console.error("getUserMedia error: " + error);

29 }

30

31 function onOpenCamera(e) {

32

navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);

33 }

34 document.querySelector("#showVideo").addEventListener("click", onOpenCamera);

35 </script>

### 打开麦克风

8 <html >

9 <body>

10 <audio id="local‐audio" autoplay controls>播放麦克风捕获的声音</audio>

11 <button id="playAudio">打开麦克风</button>

12

13 <p>通过getUserMedia()获取音频</p>

14 </body>

15

16 <script>17 // 约束条件

18 const constraints = {

19 audio: true,

20 video: false

21 };

22

23 // 处理打开麦克风成功

24 function handleSuccess(stream) {

25 const audio = document.querySelector("#local‐audio");

26 audio.srcObject = stream;

27 }

28

29 // 异常处理

30 function handleError(error) {

31 console.error("getUserMedia error: " + error);

32 }

33

34 function onOpenMicrophone(e) {

35

navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);

36 }

37 document.querySelector("#playAudio").addEventListener("click", onOpenMicrophone);

38 </script>

39

40 </html>

### 打开摄像头与麦克风

const constraints = (window.constraints = {

22 audio: true,

23 video: true

24 });

只要把这个约束设置成全都是true就可以了.