

本门课程目的：帮助更多的学员入门WebRTC

腾讯课堂 零声学院 《WebRTC入门与提高》 <https://ke.qq.com/course/435382?quicklink=1>

咨询QQ群：782508536

WebRTC 入门几大坑

WebRTC入门与提高



看了那么多文章demo还是跑不起来

好不容易demo运行了画面确实黑的

为什么一到公网又黑屏？

HTTP的坑在哪里

终于看到画面了，但不懂原理啊

刚入门真要各种原理都深究吗？

为什么和Android通话不成功

以下是课程的部分文档

本门课程分为以下章节：

1. WebRTC入门
2. WebRTC开发环境搭建
3. Coturn穿透和转发服务器搭建
4. 音视频采集和播放
5. Nodejs实战
6. 手把手实现音视频一对一通话

7. 开源方案介绍

8. AppRTC开源方案搭建

1 WebRTC入门

本章目的：

1. 了解什么WebRTC
2. 掌握WebRTC通话原理
3. 学完该课程的收获

1.1 什么是WebRTC

WebRTC (Web Real-Time Communication) 是 Google于2010以6829万美元从 Global IP Solutions 公司购买，并于2011年将其开源，旨在建立一个互联网浏览器间的实时通信的平台，让 WebRTC技术成为 H5标准之一。我们看官网 (<https://webrtc.org>) 的介绍

WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose.

Our mission: To enable rich, high-quality RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allow them all to communicate via a common set of protocols.

The WebRTC initiative is a project supported by Google, Mozilla and Opera, amongst others. This page is maintained by the Google Chrome team.

New to WebRTC? Take a look at our [codelab](#).

Lots more resources for getting started are available from webrtc.org/start.

Supported Browsers & Platforms

Chrome



Firefox



Opera



Android

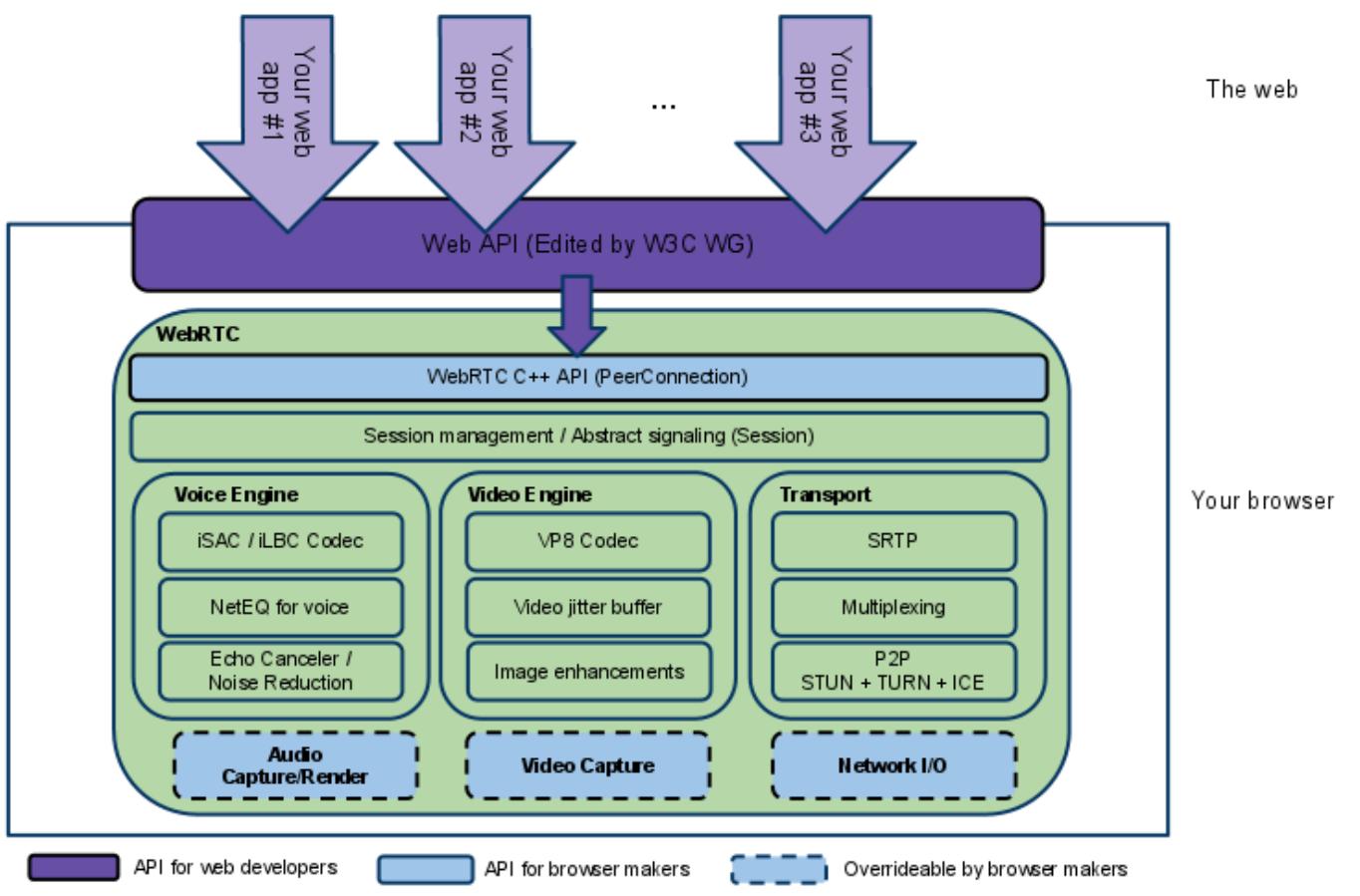


iOS



从官网上的描述我们可以知道，WebRTC是一个免费的开放项目，它通过简单的API为浏览器和移动应用程序提供实时通信 (RTC) 功能。

1.2 WebRTC框架



上图的框架对于不同的开发人员关注点不同：

- (1) 紫色部分是Web应用开发者API层
- (2) 蓝色实线部分是面向浏览器厂商的API层
- (3) 蓝色虚线部分浏览器厂商可以自定义实现

特别是图中的 `PeerConnection` 为 Web 开发人员提供了一个抽象，从复杂的内部结构中抽象出来。我们只需要关注 `PeerConnection` 这个对象即可以开发音视频通话应用内。

WebRTC架构组件介绍

Your Web App

Web开发者开发的程序，Web开发者可以基于集成WebRTC的浏览器提供的web API开发基于视频、音频的实时通信应用。

Web API

面向第三方开发者的WebRTC标准API（Javascript），使开发者能够容易地开发出类似于网络视频聊天的web应用，最新的标准化进程可以查看[这里](#)。

WebRTC Native C++ API

本地C++ API层，使浏览器厂商容易实现WebRTC标准的Web API，抽象地对数字信号过程进行处理。

Transport / Session

传输/会话层

会话层组件采用了libjingle库的部分组件实现，无须使用xmpp/jingle协议

VoiceEngine

音频引擎是包含一系列音频多媒体处理的框架。

PS：VoiceEngine是WebRTC极具价值的技术之一，是Google收购GIPS公司后开源的。在VoIP上，技术业界领先。

Opus：支持从6 kbit/s到510 kbit/s的恒定和可变比特率编码，帧大小从2.5 ms到60 ms，各种采样率从8 kHz (4 kHz 带宽) 到48 kHz (20 kHz带宽，可复制人类听觉系统的整个听力范围)。由IETF RFC 6176定义。

NetEQ模块是Webrtc语音引擎中的核心模块，一种动态抖动缓冲和错误隐藏算法，用于隐藏网络抖动和数据包丢失的负面影响。保持尽可能低的延迟，同时保持最高的语音质量。

VideoEngine

WebRTC视频处理引擎

VideoEngine是包含一系列视频处理的整体框架，从摄像头采集视频到视频信息网络传输再到视频显示整个完整过程的解决方案。

VP8 视频图像编解码器，是WebRTC视频引擎的默认的编解码器

VP8适合实时通信应用场景，因为它主要是针对低延时而设计的编解码器。

1.3 WebRTC发展前景

WebRTC虽然冠以“web”之名，但并不受限于传统互联网应用或浏览器的终端运行环境。实际上无论终端运行环境是**浏览器、桌面应用、移动设备（Android或iOS）还是IoT设备**，只要IP连接可到达且**符合WebRTC规范就可以互通**。这一点释放了大量智能终端（或运行在智能终端上的app）的实时通信能力，打开了许多对于实时交互性要求较高的应用场景的想象空间，譬如在线教育、视频会议、视频社交、远程协助、远程操控等等都是其合适的应用领域。

全球领先的技术研究和咨询公司Technavio最近发布了题为“全球网络实时通讯（WebRTC）市场，2017-2021”的报告。报告显示，2017-2021年期间，全球网络实时通信（WebRTC）市场将以**34.37%的年均复合增长率增长**，增长十分迅速。增长主要来自北美、欧洲及亚太地区。

1.4 国内方案厂商

声网、即构科技、环信、融云等公司都在基于WebRTC二次开发自己的音视频通话方案。

声网 <https://www.agora.io/cn/>

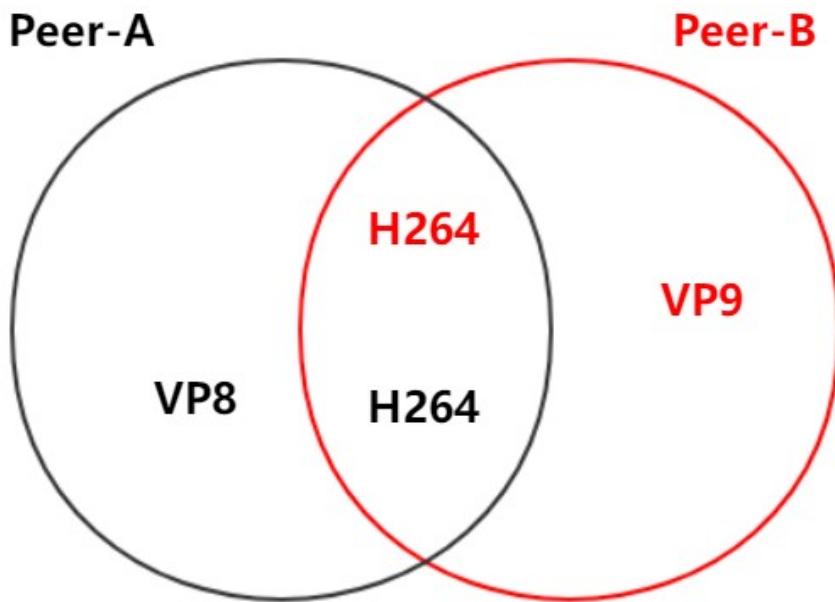
即构科技 <https://www.zego.im/>

1.5 WebRTC通话原理

首先思考的问题：两个不同网络环境的（具备摄像头/麦克风多媒体设备的）浏览器，要实现点对点的实时音视频对话，难点在哪里？

1. 媒体协商

彼此要了解对方支持的媒体格式



比如：Peer-A端可支持VP8、H264多种编码格式，而Peer-B端支持VP9、H264，要保证两端都正确的编解码，最简单的办法就是取它们的交集H264

注：有一个专门的协议，称为Session Description Protocol (SDP)，可用于描述上述这类信息，在WebRTC中，参与视频通讯的**双方必须先交换SDP信息**，这样双方才能知根知底，而交换SDP的过程，也称为“**媒体协商**”。

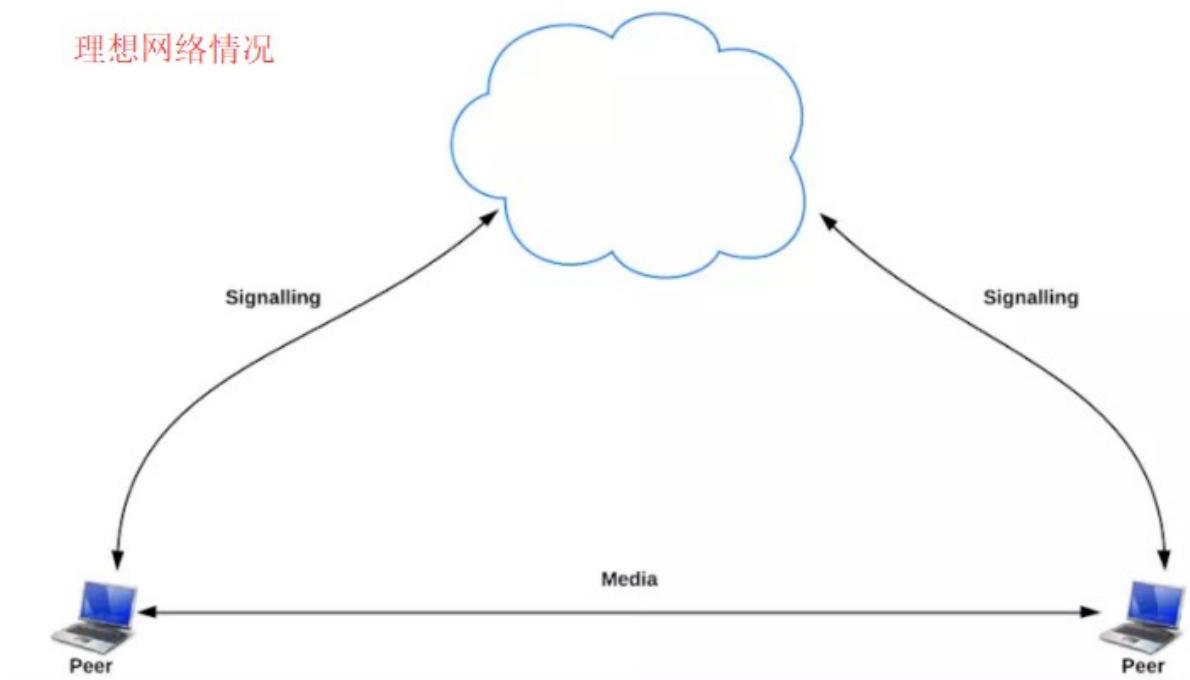
2. 网络协商

彼此要了解对方的网络情况，这样才有可能找到一条相互通讯的链路

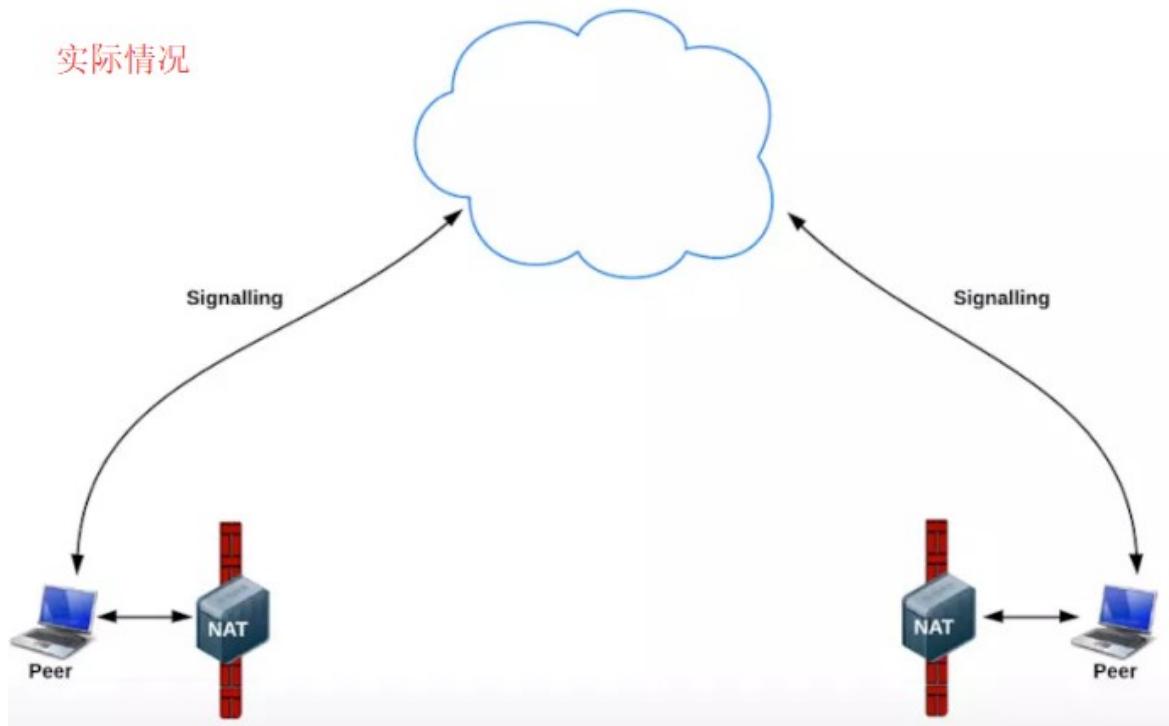
先说结论：(1)获取外网IP地址映射； (2) 通过信令服务器 (signal server) 交换“网络信息”

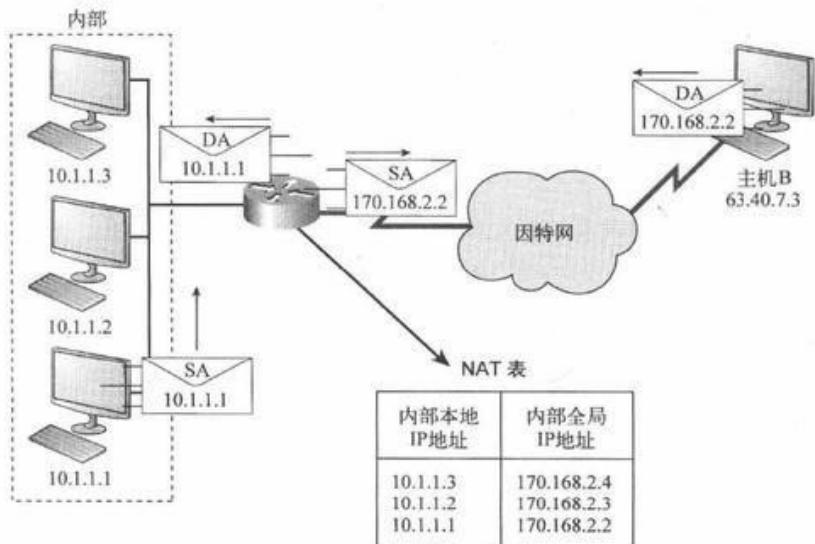
理想的网络情况是每个浏览器的电脑都是**私有公网IP**，可以直接进行点对点连接。

理想网络情况



实际情况是：我们的电脑和电脑之前或大或小都是在某个局域网中，**需要NAT (Network Address Translation, 网络地址转换)**，显示情况如下图：





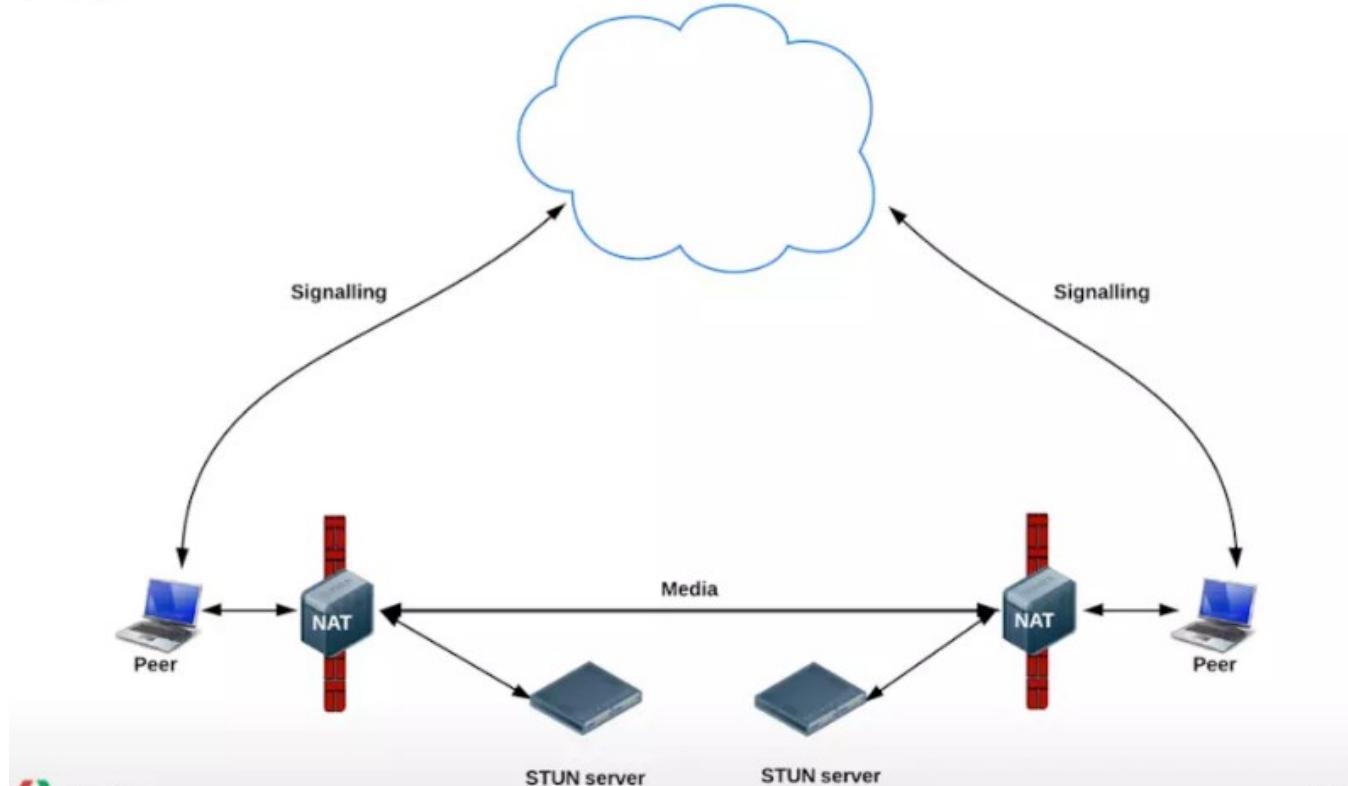
在解决WebRTC使用过程中的上述问题的时候，我们需要用到STUN和TURN。

STUN

STUN (Session Traversal Utilities for NAT, NAT会话穿越应用程序) 是一种网络协议，它允许位于NAT (或多重NAT) **后的客户端找出自己的公网地址**，查出自己位于哪种类型的NAT之后以及NAT为某一个**本地端口所绑定的Internet端端口**。这些信息被用来在两个同时处于NAT路由器之后的主机之间创建UDP通信。该协议由RFC 5389定义。

在遇到上述情况的时候，我们可以建立一个STUN服务器，这个服务器做什么用的呢？主要是给无法在公网环境下的视频通话设备分配公网IP用的。这样两台电脑就可以在公网IP中进行通话。

STUN



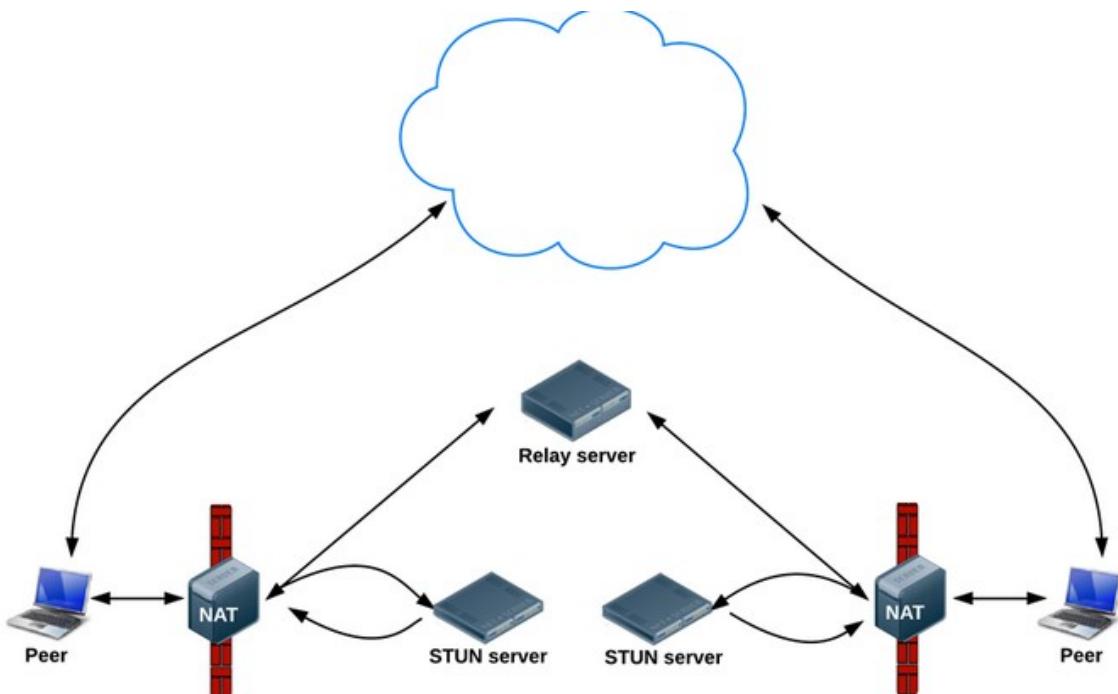
使用一句话说明STUN做的事情就是：告诉我你的公网IP地址+端口是什么。搭建STUN服务器很简单，媒体流传输是按照P2P的方式。

那么问题来了，STUN并不是每次都能成功的为需要NAT的通话设备分配IP地址的，P2P在传输媒体流时，使用的本地带宽，在多人视频通话的过程中，通话质量的好坏往往需要根据使用者本地的带宽确定。那么怎么办？TURN可以很好的解决这个问题。

TURN

TURN的全称为Traversal Using Relays around NAT，是STUN/RFC5389的一个拓展，主要添加了Relay功能。如果终端在NAT之后，那么在特定的情景下，有可能使得终端无法和其对等端（peer）进行直接的通信，这时就需要公网的服务器作为一个中继，对来往的数据进行转发。这个转发的协议就被定义为TURN。

在上图的基础上，再架设几台TURN服务器：



在STUN分配公网IP失败后，可以通过TURN服务器请求公网IP地址作为中继地址。**这种方式的带宽由服务器端承担**，在多人视频聊天的时候，本地带宽压力较小，并且，根据Google的说明，TURN协议可以使用在所有的环境中。（单向数据200kbps 一对一直通）

以上是WebRTC中经常用到的2个协议，STUN和TURN服务器我们使用coturn开源项目来搭建。

补充：ICE跟STUN和TURN不一样，ICE不是一种协议，而是一个框架（Framework），它整合了STUN和TURN。coturn开源项目集成了STUN和TURN的功能。

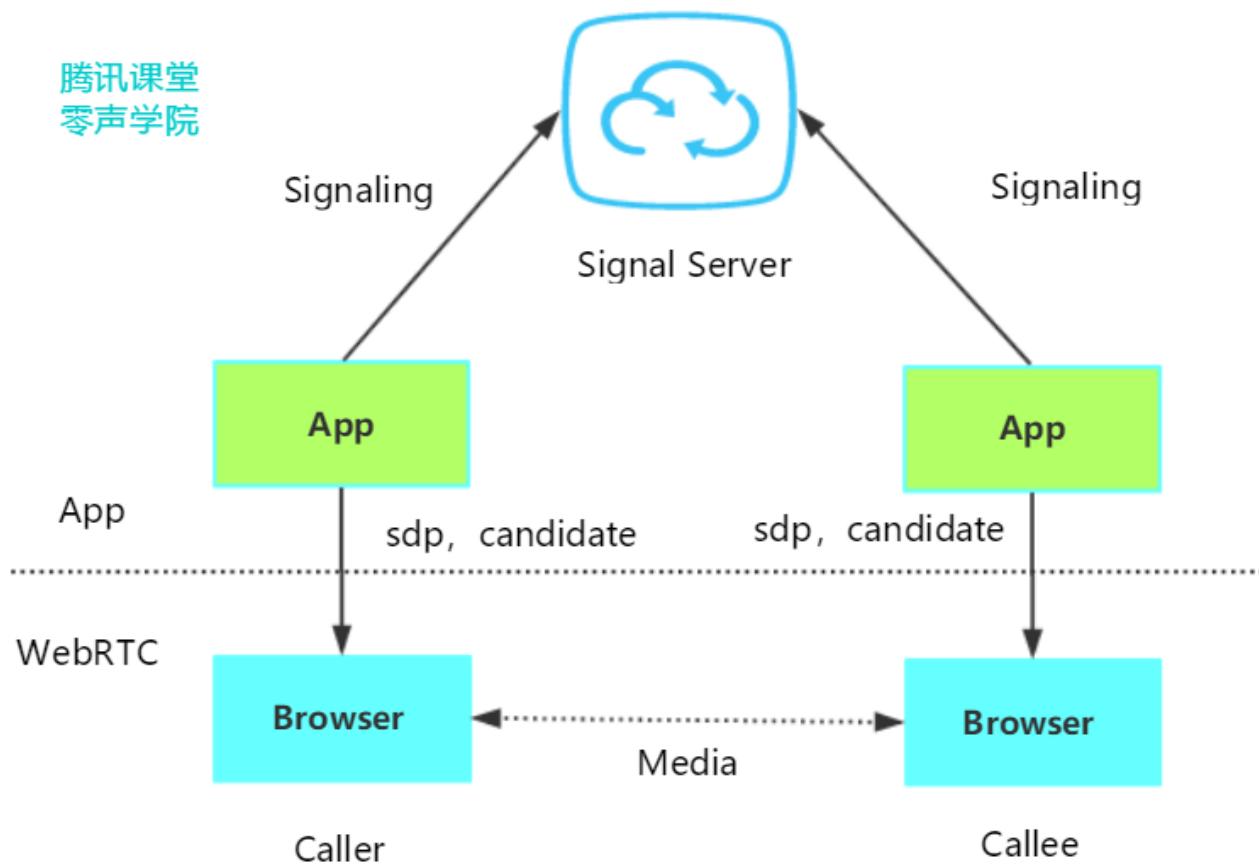
在WebRTC中用来描述网络信息的术语叫**candidate**。

媒体协商 sdp

网络协商 candidate

3. 媒体协商+网络协商数据的交换通道

从上面1/2点我们知道2个客户端协商媒体信息和网络信息，那怎么去交换？是不是需要一个中间商去做交换？所以我们需要一个信令服务器（Signal server）转发彼此的媒体信息和网络信息。



如上图，我们在基于WebRTC API开发应用（APP）时，可以将彼此的APP连接到信令服务器（Signal Server，一般搭建在公网，或者两端都可以访问到的局域网），借助信令服务器，就可以实现上面提到的SDP媒体信息及Candidate网络信息交换。

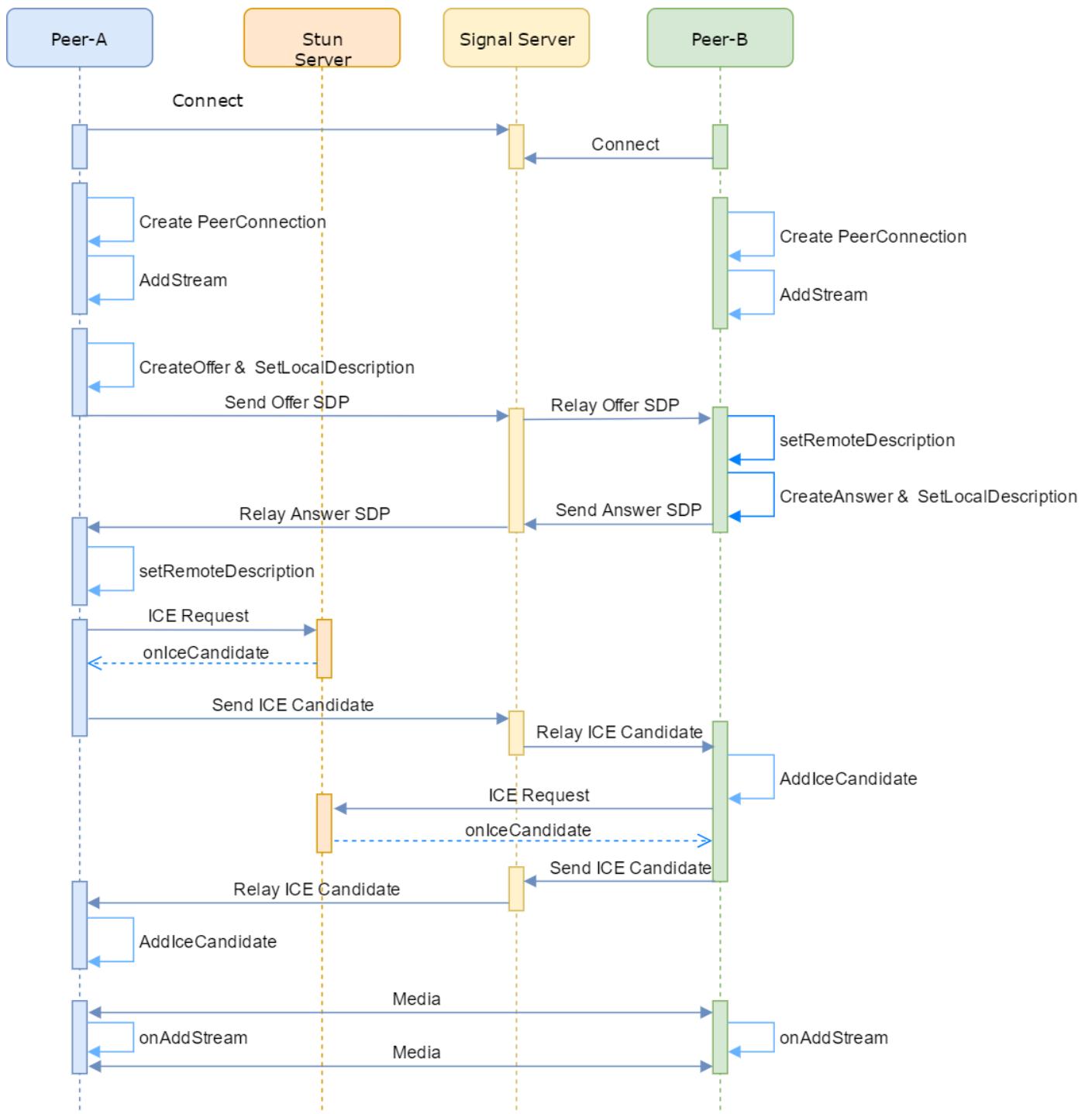
信令服务器不只是交互 媒体信息sdp和网络信息candidate，比如：

- (1) 房间管理
- (2) 人员进出房间

WebRTC APIs

1. **MediaStream** — MediaStream用来表示一个媒体数据流（通过getUserMedia接口获取），允许你访问输入设备，如麦克风和Web摄像机，该API允许从其中任意一个获取媒体流。
2. **RTCPeerConnection** — RTCPeerConnection对象允许用户在两个浏览器之间直接通讯，你可以通过网络将捕获的音频和视频流实时发送到另一个WebRTC端点。使用这些API，你可以在本地机器和远程对等点之间创建连接。它提供了连接到远程对等点、维护和监视连接以及在不再需要连接时关闭连接的方法。

4. 一对一通话



在一对通话场景中，每个 Peer 均创建有一个 PeerConnection 对象，由一方主动发 Offer SDP，另一方则应答 AnswerSDP，最后双方交换 ICE Candidate 从而完成通话链路的建立。但是在中国的网络环境中，据一些统计数据 显示，至少一半的网络是无法直接穿透打通，这种情况下只能借助 TURN 服务器 中转。

5. NAT知识补充

具体NAT打洞的知识在本课程不做进一步的讲解，这里提供些链接给大家做参考：

P2P技术详解(一): NAT详解——详细原理、P2P简介 <https://www.cnblogs.com/mlgjb/p/8243646.htm>

P2P技术详解(二): P2P中的NAT穿越(打洞)方案详解 <https://www.jianshu.com/p/9bfbcbee0abb>

P2P技术详解(三): P2P技术之STUN、TURN、ICE详解 <https://www.jianshu.com/p/258e7d8be2ba>

1.6 课程收获

1. WebRTC工作原理
2. WebRTC API的使用
3. WebRTC 一对一视频通话开发
4. AppRTC开源项目搭建方法
5. 常用的WebRTC开源方案

2 WebRTC开发环境

2.1 安装vscode

下载和安装vscode

vscode官网: <https://code.visualstudio.com/>

下载地

址: <https://vscode.cdn.azure.cn/stable/1b8e8302e405050205e69b59abb3559592bb9e60/VSCodiumUserSetup-x64-1.31.1.exe>

下载完后按引导安装即可

配置vscode

安装插件

- Prettier Code Formatter 使用 Prettier 来统一代码风格，当保存 HTML/CSS/JavaScript 文件时，它会自动调整代码格式。
- Live Server: 在本地开发环境中，实时重新加载(reload)页面。

第一个简单的HTML页面

HTML教程: <https://www.runoob.com/html/html-tutorial.html>

范例first_html.html

```
1 <html>
2 <body>
3 <h1>标题1</h1>
```

```
5
6 <p>第一个段落.</p>
7 <p>我的第一个HTML页面</p>
8 </body>
9 </html>
10
```

第一个js程序

JavaScript教程: <https://www.runoob.com/js/js-tutorial.html>

范例first_js.html

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>Body 中的 JavaScript</h2>
6
7 <p id="demo">一个段落。</p>
8
9 <button type="button" onclick="myFunction()">试一试</button>
10
11 <script>
12 function myFunction() {
13     document.getElementById("demo").innerHTML = "段落已被更改。";
14 }
15 </script>
16
17 </body>
18 </html>
19
```

2.2 安装 nodejs

源码安装nodejs

1. 下载nodejs

```
1 wget https://nodejs.org/dist/v10.16.0/node-v10.16.0-linux-x64.tar.xz
```

2. 解压文件

```
1 # 解压
2 tar -xvf node-v10.16.0-linux-x64.tar.xz
3 # 进入目录
4 cd node-v10.16.0-linux-x64/
5 # 查看当前的目录
6 pwd
```

3. 链接执行文件

```
1 # 确认一下nodejs下bin目录是否有node 和npm文件，如果有就可以执行软连接，比如
2 sudo ln -s /home/lqf/webrtc/nodejs/bin/npm /usr/local/bin/
3 sudo ln -s /home/lqf/webrtc/nodejs/bin/node /usr/local/bin/
4
5 # 看清楚，这个路径是你自己创建的路径，我的路径是/home/lqf/webrtc/nodejs
6
7 # 查看是否安装，安装正常则打印版本号
8 node -v
9 npm -v
```

第一个nodejs教程

nodejs教程：<https://www.runoob.com/nodejs/nodejs-tutorial.html>

在我们创建 Node.js 第一个 "Hello, World!" 应用前，让我们先了解下 Node.js 应用是由哪几部分组成的：

1. **引入 required 模块**：我们可以使用 `require` 指令来载入 Node.js 模块。
2. **创建服务器**：服务器可以监听客户端的请求，类似于 Apache、Nginx 等 HTTP 服务器。
3. **接收请求与响应请求** 服务器很容易创建，客户端可以使用浏览器或终端发送 HTTP 请求，服务器接收请求后返回响应数据。

创建 Node.js 应用

步骤一、引入 required 模块

我们使用 `require` 指令来载入 `http` 模块，并将实例化的 `HTTP` 赋值给变量 `http`，实例如下：

```
1 var http = require("http");
```

步骤二、创建服务器

接下来我们使用 `http.createServer()` 方法创建服务器，并使用 `listen` 方法绑定 8888 端口。函数通过 `request`, `response` 参数来接收和响应数据。

实例如下，在你项目的根目录下创建一个叫 server.js 的文件，并写入以下代码：

```
1 var http = require('http');
2
3 http.createServer(function (request, response) {
4
5     // 发送 HTTP 头部
6     // HTTP 状态值: 200 : OK
7     // 内容类型: text/plain
8     response.writeHead(200, {'Content-Type': 'text/plain'});
9
10    // 发送响应数据 "Hello World"
11    response.end('Hello World\n');
12 }).listen(8888);
13
14 // 终端打印如下信息
15 console.log('Server running at http://127.0.0.1:8888/');
```

以上代码我们完成了一个可以工作的 HTTP 服务器。

使用 **node** 命令执行以上的代码：

```
1 node server.js
2 Server running at http://127.0.0.1:8888/
```

接下来，打开浏览器访问 <http://127.0.0.1:8888/>，你会看到一个写着 "Hello World" 的网页。

分析Node.js 的 HTTP 服务器：

- 第一行请求 (require) Node.js 自带的 http 模块，并且把它赋值给 http 变量。
- 接下来我们调用 http 模块提供的函数：createServer。这个函数会返回一个对象，这个对象有一个叫做 listen 的方法，这个方法有一个数值参数，指定这个 HTTP 服务器监听的端口号。

3 coturn穿透和转发服务器

3.1 安装依赖

ubuntu系统

```
1 sudo apt-get install libssl-dev
2 sudo apt-get install libevent-dev
```

centos系统

```
1 sudo yum install openssl-devel  
2 sudo yum install libevent-devel
```

3.2 编译安装coturn

```
1 git clone https://github.com/coturn/coturn  
2 cd coturn  
3 ./configure  
4 make  
5 sudo make install
```

3.3 查看是否安装成功

```
1 # nohup是重定向命令，输出都将附加到当前目录的 nohup.out 文件中； 命令后加 &，后台执行起来后按  
2 #ctr+c，不会停止  
3 sudo nohup turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov &  
4 #然后查看相应的端口号3478是否存在进程  
5 sudo lsof -i:3478  
6
```

3.4 测试地址，请分别测试stun和turn

Coturn是集成了stun+turn协议。

测试网址：<https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/>

ICE options

IceTransports value: all relay
ICE Candidate Pool: 0 0

Time	Component Type	Foundation	Protocol Address	Port	Priority
0.068 39.815	1 relay	2630099496	udp 192.168.221.132	63749	2 32286 255 Done

Gather candidates

4. 音视频采集和播放

有三个案例：

- (1) 打开摄像头并将画面显示到页面；
- (2) 打开麦克风并在页面播放捕获的声音；
- (3) 同时打开摄像头和麦克风，并在页面显示画面和播放捕获的声音

4.1 打开摄像头

实战：打开摄像头并将画面显示到页面

效果展示



代码流程

1. 初始化button、video控件

2. 绑定“打开摄像头”响应事件onOpenCamera
3. 如果要打开摄像头则点击“打开摄像头”按钮，以触发onOpenCamera事件的调用
4. 当触发onOpenCamera调用时
 - a. 设置约束条件，即是getUserMedia函数的入参
 - b. getUserMedia有两种情况，一种是正常打开摄像头，使用handleSuccess处理；一种是打开摄像头失败，使用handleError处理
 - c. 当正常打开摄像头时，则将getUserMedia返回的stream对象赋值给video控件的srcObject即可将视频显示出来

示例代码

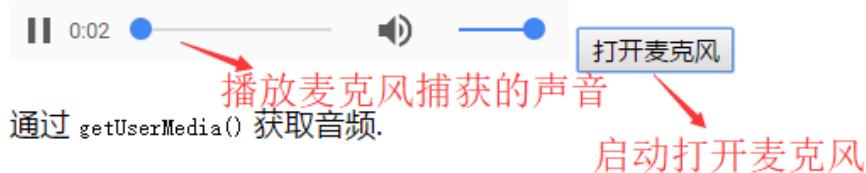
4.1 video.html

```
1  <!DOCTYPE html>
2  <!--
3  * 版权：腾讯课堂 零声学院 https://0voice.ke.qq.com/?tuin=137bb271 .
4  *
5  * 文件名：video.html
6  * 功能：获取视频并将其显示到页面
7  -->
8  <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
33      navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);
34    }
35    document.querySelector("#showVideo").addEventListener("click", onOpenCamera);
36  </script>
```

4.2 打开麦克风

实战：打开麦克风并在页面播放捕获的声音

效果展示



代码流程

1. 初始化button、audio控件
2. 绑定“打开麦克风”响应事件onOpenMicrophone
3. 如果要打开麦克风则点击“打开麦克风”按钮，以触发onOpenMicrophone事件的调用
4. 当触发onOpenCamera调用时
 - a. 设置约束条件，即是getUserMedia函数的入参
 - b. getUserMedia有两种情况，一种是正常打开麦克风，使用handleSuccess处理；一种是打开麦克风失败，使用handleError处理
 - c. 当正常打开麦克风时，则将getUserMedia返回的stream对象赋值给audio控件的srcObject即可将声音播放出来

示例代码

4.2 audio.html

```

1  <!DOCTYPE html>
2  <!--
3  * 版权：腾讯课堂 零声学院 https://voice.ke.qq.com/?tuin=137bb271 .
4  *
5  * 文件名：audio.html
6  * 功能：打开麦克风并在页面播放捕获的声音
7  -->
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
36     navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);
37
38     document.querySelector("#playAudio").addEventListener("click", onOpenMicrophone);
39
40 </script>
41
42 </html>

```

webrtc获取音视频设备

4.3 打开摄像头和麦克风

同时打开摄像头和麦克风，范例可以参考4.1，只是在约束条件中把

```

1 const constraints = {
2     audio: false, // 不打开麦克风
3     video: true
4 };

```

改为

```

1 const constraints = {
2     audio: true, // 打开麦克风
3     video: true
4 };

```

具体代码

4.3 video_audio.html

```
1 <!DOCTYPE html>
2 <!--
3 * 版权: 腾讯课堂 零声学院 https://0voice.ke.qq.com/?tuin=137bb271 .
4 *
5 * 文件名: video.html
6 * 功能: 获取音视频并将其显示到页面和播放声音
7 -->
8 <html>
9   <body>
10
11     <video id="local-video" autoplay playsinline></video>
12     <button id="showVideo">打开音视频</button>
13
14     <div id="errorMsg"></div>
15
16     <p>通过 <code>getUserMedia()</code> 获取音视频.</p>
17
18
19 <script>
20   // 设置约束条件, 同时打开音频流和视频流
21   const constraints = (window.constraints = {
22     audio: true,
23     video: true
24   });
25
26   // 处理打开摄像头+麦克风成功
27   function handleSuccess(stream) {
28     const video = document.querySelector("#local-video");
29     video.srcObject = stream;
30   }
31
32   // 处理打开摄像头+麦克风失败
33   function handleError(error) {
34     console.error("getUserMedia error: " + error);
35   }
36
37   async function onOpenAV(e) {
38
39     navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);
40   }
41
42   document
43     .querySelector("#showVideo")
44     .addEventListener("click", onOpenAV);
45
46 </script>
47   </body>
48 </html>
```

4.4 拓展讲解

1. getUserMedia API参考：<https://developer.mozilla.org/zh-CN/docs/Web/API/MediaDevices/getUserMedia>

2. !=和!==区别

!= 在表达式两边的数据类型不一致时,会隐式转换为相同数据类型,然后对值进行比较. 比如 1 和 "1" , 1 != "1" 为 false

!== 不会进行类型转换,在比较时除了对值进行比较以外,还比较两边的数据类型, 它是恒等运算符==的非形式., 1 != "1" 为true

3. video控件属性

<video>: The Video Embed element <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video>

HTML DOM Video 对象 <https://www.runoob.com/jsref/dom-obj-video.html>

5. Nodejs实战

对于我们WebRTC项目而言, nodejs主要是实现信令服务器的功能, 客户端和服务器端的交互我们选择websocket作为通信协议, 所以该章节的实战以websocket的使用为主。

web客户端的websocket和nodejs服务器端的websocket有一定的差别, 所以我们分开两部分进行讲解。

5.1 web客户端 websocket

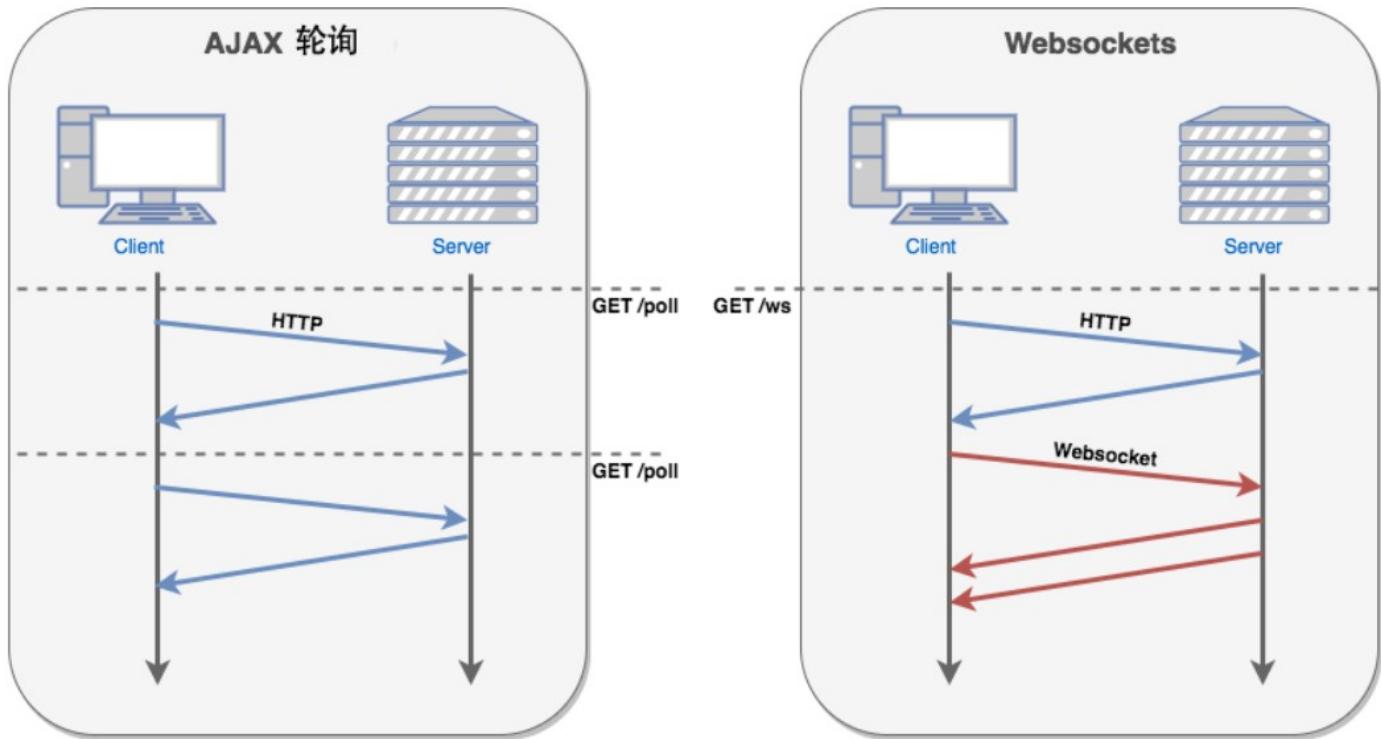
WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

WebSocket 使得客户端和服务器之间的数据交换变得更加简单, 允许服务端主动向客户端推送数据。在 WebSocket API 中, 浏览器和服务器只需要完成一次握手, 两者之间就直接可以创建持久性的连接, 并进行双向数据传输。

在 WebSocket API 中, 浏览器和服务器只需要做一个握手的动作, 然后, 浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。

现在, 很多网站为了实现推送技术, 所用的技术都是 Ajax 轮询。轮询是在特定的时间间隔 (如每1秒) , 由浏览器对服务器发出HTTP请求, 然后由服务器返回最新的数据给客户端的浏览器。这种传统的模式带来很明显的缺点, 即浏览器需要不断的向服务器发出请求, 然而HTTP请求可能包含较长的头部, 其中真正有效的数据可能只是很小的一部分, 显然这样会浪费很多的带宽等资源。

HTML5 定义的 WebSocket 协议, 能更好的节省服务器资源和带宽, 并且能够更实时地进行通讯。



浏览器通过 JavaScript 向服务器发出建立 WebSocket 连接的请求，连接建立以后，客户端和服务器端就可以通过 TCP 连接直接交换数据。

当你获取 Web Socket 连接后，你可以通过 **send()** 方法来向服务器发送数据，并通过 **onmessage** 事件来接收服务器返回的数据。

以下 API 用于创建 WebSocket 对象。

```
var Socket = new WebSocket(url, [protocol]);
```

以上代码中的第一个参数 **url**, 指定连接的 URL。第二个参数 **protocol** 是可选的，指定了可接受的子协议。

WebSocket 属性

以下是 WebSocket 对象的属性。假定我们使用了以上代码创建了 **Socket** 对象：

属性	描述
Socket.readyState	只读属性 readyState 表示连接状态，可以是以下值： <ul style="list-style-type: none"> • 0 - 表示连接尚未建立。 • 1 - 表示连接已建立，可以进行通信。 • 2 - 表示连接正在进行关闭。 • 3 - 表示连接已经关闭或者连接不能打开。
Socket.bufferedAmount	只读属性 bufferedAmount 已被 send() 放入正在队列中等待传输，但是还没有发出的 UTF-8 文本字节数。

WebSocket 事件

以下是 WebSocket 对象的相关事件。假定我们使用了以上代码创建了 Socket 对象：

事件	事件处理程序	描述
open	Socket.onopen	连接建立时触发
message	Socket.onmessage	客户端接收服务端数据时触发
error	Socket.onerror	通信发生错误时触发
close	Socket.onclose	连接关闭时触发

WebSocket 方法

以下是 WebSocket 对象的相关方法。假定我们使用了以上代码创建了 Socket 对象：

方法	描述
Socket.send()	使用连接发送数据
Socket.close()	关闭连接

为了建立一个 WebSocket 连接，客户端浏览器首先要向服务器发起一个 HTTP 请求，这个请求和通常的 HTTP 请求不同，包含了一些附加头信息，其中附加头信息"Upgrade: WebSocket"表明这是一个申请协议升级的 HTTP 请求，服务器端解析这些附加的头信息然后产生应答信息返回给客户端，客户端和服务端的 WebSocket 连接就建立起来了，双方就可以通过这个连接通道自由的传递信息，并且这个连接会持续存在直到客户端或者服务器端的某一方主动的关闭连接。

5.2 Nodejs服务器 websocket

Nodejs教程：<https://www.runoob.com/nodejs/nodejs-tutorial.html>

简单的说 Node.js 就是运行在服务端的 JavaScript。

服务器端使用websocket需要安装nodejs-websocket

```
1 cd 工程目录
2 # 此刻我们需要执行命令：
3 sudo npm init
4 #创建package.json文件，系统会提示相关配置，也可以使用命令：
5 sudo npm init -y
6 sudo npm install nodejs-websocket
```

官方参考：<https://www.npmjs.com/package/nodejs-websocket>

我们只要关注：

- (1) 如何创建websocket服务器，通过`createServer`和`listen`接口；
- (2) 如何判断有新的连接进来，`createServer`的回调函数判断；
- (3) 如何判断关闭事件，通过`on("close", callback)`事件的回调函数；
- (4) 如何判断接收到数据，通过`on("text", callback)`事件的回调函数；
- (5) 如何判断接收异常，通过`on("error", callback)`事件的回调函数；
- (6) 如何主动发送数据，调用`sendText`

参考代码

```
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)
```

5.3 websocket聊天室实战

效果展示+框架分析

效果展示

客服端

Websocket简易聊天

我是Darren老师

user1 进来啦

user2 进来啦

user1 说: 我是Darren老师

user2 说: 我是King 大叔

Websocket简易聊天

我是King 大叔

user2 进来啦

user1 说: 我是Darren老师

user2 说: 我是King 大叔

服务端

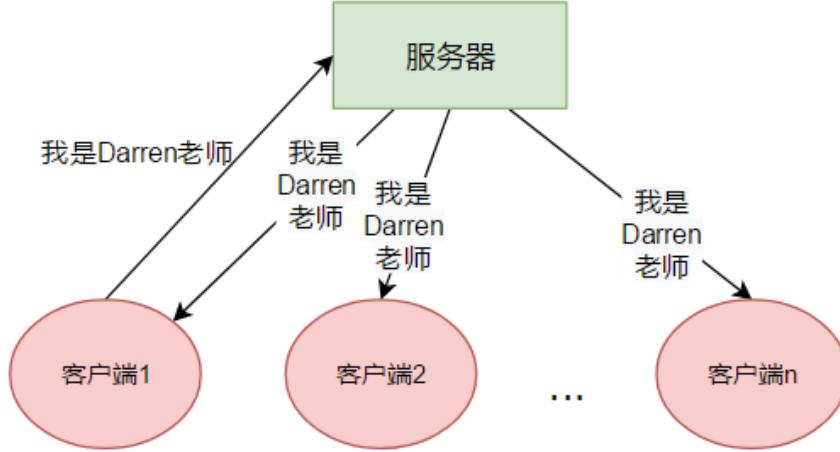
```
lqf@ubuntu:/mnt/hgfs/ubuntu
创建一个新的连接-----
创建一个新的连接-----
回复 我是Darren老师
回复 我是King 大叔
```

框架分析

消息类型分为三种：

1. enter: 新人进入 (蓝色字体显示)
2. message: 普通聊天信息 (黑色字体显示)
3. leave: 有人离开 (红色字体显示)

服务器在收到某个客户端的消息 (message+enter+leave) , 然后将其广播给所有的客户端 (包括发送者) 。



客户端代码

目录和文件名：05/5.3/client.html

```

1 <html>
2
3 <body>
4     <h1>WebSocket简易聊天</h1>
5     <div id="app">
6         <input id="sendMsg" type="text" />
7         <button id="submitBtn">发送</button>
8     </div>
9 </body>
10 <script type="text/javascript">
11     //在页面显示聊天内容
12     function showMessage(str, type) {
13         var div = document.createElement("div");
14         div.innerHTML = str;
15         if (type == "enter") {
16             div.style.color = "blue";
17         } else if (type == "leave") {
18             div.style.color = "red";
19         }
20         document.body.appendChild(div);
21     }
22
23     //新建一个websocket
24     var websocket = new WebSocket("ws://192.168.221.132:8010");
25     //打开websocket连接
26     websocket.onopen = function () {
27         console.log("已经连上服务器----");
28         document.getElementById("submitBtn").onclick = function () {
29             var txt = document.getElementById("sendMsg").value;
30             if (txt) {
31                 //向服务器发送数据

```

```

32         websocket.send(txt);
33     }
34   };
35 };
36 //关闭连接
37 websocket.onclose = function () {
38   console.log("websocket close");
39 };
40 //接收服务器返回的数据
41 websocket.onmessage = function (e) {
42   var mes = JSON.parse(e.data); // json格式
43   showMessage(mes.data, mes.type);
44 };
45 </script>
46
47 </html>

```

服务器端代码

目录和文件名：05/5.3/server.js

```

1 var ws = require("nodejs-websocket")
2 var port = 8010;
3 var user = 0;
4
5 // 创建一个连接
6 var server = ws.createServer(function (conn) {
7   console.log("创建一个新的连接-----");
8   user++;
9   conn.nickname="user" + user;
10  conn.fd="user" + user;
11  var mes = {};
12  mes.type = "enter";
13  mes.data = conn.nickname + " 进来啦"
14  broadcast(JSON.stringify(mes)); // 广播
15
16 //向客户端推送消息
17 conn.on("text", function (str) {
18   console.log("回复 "+str)
19   mes.type = "message";
20   mes.data = conn.nickname + " 说:      " + str;
21   broadcast(JSON.stringify(mes));
22 });
23
24 //监听关闭连接操作
25 conn.on("close", function (code, reason) {
26   console.log("关闭连接");
27   mes.type = "leave";
28   mes.data = conn.nickname+" 离开了"

```

```

29         broadcast(JSON.stringify(mes));
30     });
31
32     //错误处理
33     conn.on("error", function (err) {
34         console.log("监听到错误");
35         console.log(err);
36     });
37 }).listen(port);
38
39 function broadcast(str){
40     server.connections.forEach(function(connection){
41         connection.sendText(str);
42     })
43 }

```

5.4 Map实战

因为信令服务器使用map管理房间，所以我们先做个小练习。

主要涉及put/get/remove/size等操作。

目录和文件名：05/5.4/map.js

```

1  /** ----- ZeroRTCMMap ----- */
2  var ZeroRTCMMap = function () {
3      this._entrys = new Array();
4      // 插入
5      this.put = function (key, value) {
6          if (key == null || key == undefined) {
7              return;
8          }
9          var index = this._getIndex(key);
10         if (index == -1) {
11             var entry = new Object();
12             entry.key = key;
13             entry.value = value;
14             this._entrys[this._entrys.length] = entry;
15         } else {
16             this._entrys[index].value = value;
17         }
18     };
19     // 根据key获取value
20     this.get = function (key) {
21         var index = this._getIndex(key);
22         return (index != -1) ? this._entrys[index].value : null;
23     };
24     // 移除key-value
25     this.remove = function (key) {
26         var index = this._getIndex(key);

```

```
27     if (index != -1) {
28         this._entrys.splice(index, 1);
29     }
30 };
31 // 清空map
32 this.clear = function () {
33     this._entrys.length = 0;
34 };
35 // 判断是否包含key
36 this.contains = function (key) {
37     var index = this._getIndex(key);
38     return (index != -1) ? true : false;
39 };
40 // map内key-value的数量
41 this.size = function () {
42     return this._entrys.length;
43 };
44 // 获取所有的key
45 this.getEntrys = function () {
46     return this._entrys;
47 };
48 // 内部函数
49 this._getIndex = function (key) {
50     if (key == null || key == undefined) {
51         return -1;
52     }
53     var _length = this._entrys.length;
54     for (var i = 0; i < _length; i++) {
55         var entry = this._entrys[i];
56         if (entry == null || entry == undefined) {
57             continue;
58         }
59         if (entry.key === key) { // equal
60             return i;
61         }
62     }
63     return -1;
64 };
65 }
66
67 function Client(uid, conn, roomId) {
68     this.uid = uid; // 用户所属的id
69     this.conn = conn; // uid对应的websocket连接
70     this.roomId = roomId;
71     console.log('uid:' + uid + ', conn:' + conn + ', roomId: ' + roomId);
72 }
73
74 var roomMap = new ZeroRTCMap();
75
76 // Math.random() 返回介于 0 (包含) ~ 1 (不包含) 之间的一个随机数;
77 // toString(36)代表36进制, 其他一些也可以, 比如toString(2)、toString(8), 代表输出为二进制和八进制。最高支持几进制
78 // substr(2) 舍去0/1位置的字符
```

```
79 console.log('\n\n-----Math.random() -----');
80 var randmo = Math.random();
81 console.log('Math.random() = ' + randmo);
82 console.log('Math.random().toString(10) = ' + randmo.toString(10));
83 console.log('Math.random().toString(36) = ' + randmo.toString(36));
84 console.log('Math.random().toString(36).substr(0) = ' + randmo.toString(36).substr(0));
85 console.log('Math.random().toString(36).substr(1) = ' + randmo.toString(36).substr(1));
86 console.log('Math.random().toString(36).substr(2) = ' + randmo.toString(36).substr(2));
87
88 console.log('\n\n-----create client -----');
89 var roomId = 100;
90 var uid1 = Math.random().toString(36).substr(2);
91 var conn1 = 100;
92 var client1 = new Client(uid1, conn1, roomId);
93
94 var uid2 = Math.random().toString(36).substr(2);
95 var conn2 = 101;
96 var client2 = new Client(uid2, conn2, roomId);
97
98 // 插入put
99 console.log('\n\n-----put-----');
100 console.log('roomMap put client1');
101 roomMap.put(uid1, client1);
102 console.log('roomMap put client2');
103 roomMap.put(uid2, client2);
104 console.log('roomMap size:' + roomMap.size());
105
106 // 获取get
107 console.log('\n\n-----get-----');
108 var client = null;
109 var uid = uid1;
110 client = roomMap.get(uid);
111 if(client != null) {
112     console.log('get client->' + 'uid:' + client.uid + ', conn:' + client.conn + ', roomId: ' +
+ client.roomId);
113 } else {
114     console.log("can't find the client of " + uid);
115 }
116
117 uid = '123345';
118 client = roomMap.get(uid);
119 if(client != null) {
120     console.log('get client->' + 'uid:' + client.uid + ', conn:' + client.conn + ', roomId: ' +
+ client.roomId);
121 } else {
122     console.log("can't find the client of " + uid);
123 }
124
125 console.log('\n\n-----traverse-----');
126 // 遍历map
127 var clients = roomMap.getEntries();
128 for (var i in clients) {
129     let uid = clients[i].key;
```

```

130 let client = roomMap.get(uid);
131   console.log('get client->' + 'uid:' + client.uid + ', conn:' + client.conn + ', roomId: '
+ client.roomId);
132 }
133
134 console.log('\n\n-----remove-----');
135 console.log('roomMap remove uid1');
136 roomMap.remove(uid1);
137 console.log('roomMap size:' + roomMap.size());
138
139 console.log('\n\n-----clear-----');
140 console.log('roomMap clear all');
141 roomMap.clear();
142 console.log('roomMap size:' + roomMap.size());

```

6 手把手实现音视频一对一直播

1. 语法补充 =>

=>是es6语法中的arrow function

06/6.1 arrow.html

```

1 <html>
2   <head>
3     <title>arrow</title>
4   </head>
5   <body>
6     <script>
7       console.log("普通函数方式");
8       var arr1 = [1, 2, 3, 4, 5];
9       arr1.forEach(function(e) {
10         console.log(e);
11       });
12
13       console.log("箭头函数方式");
14       var arr2 = [1, 2, 3, 4, 5];
15       arr2.forEach((e) => {
16         console.log(e);
17       });
18     </script>
19   </body>
20 </html>
21

```

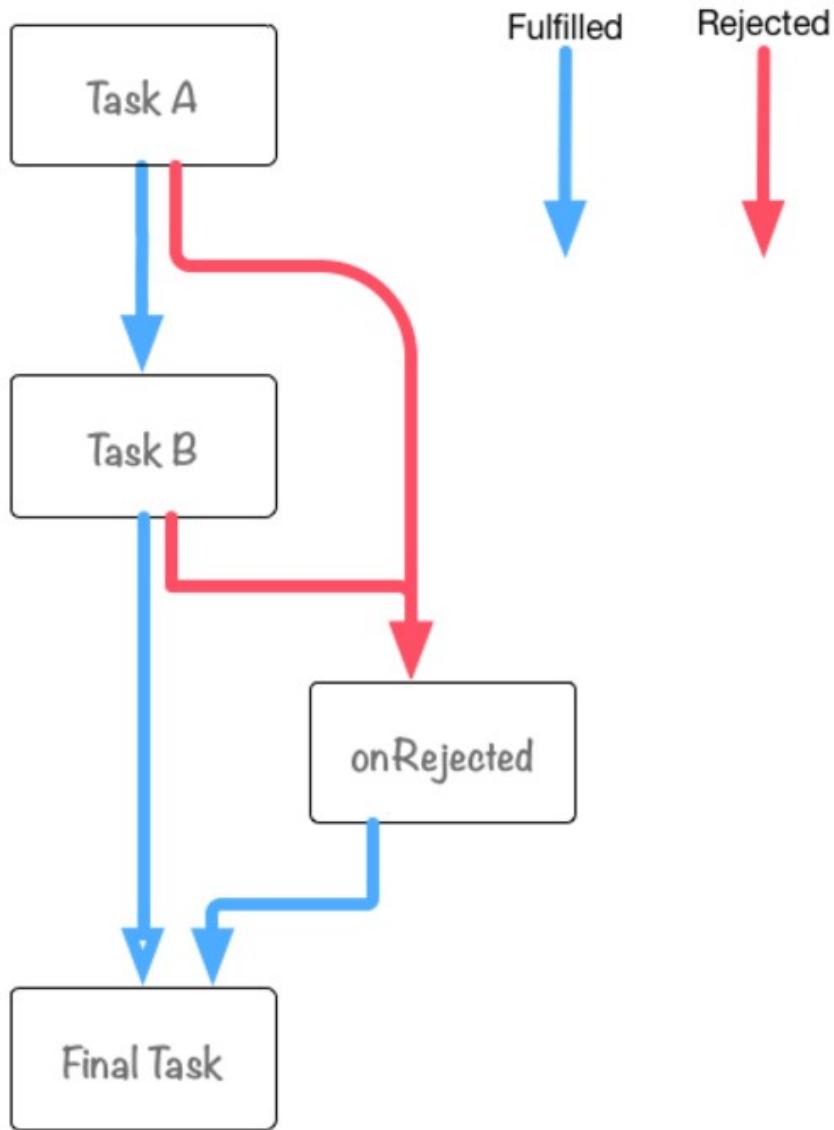
2. 语法补充promise

promise的then是异步执行，但链路的then/catch是顺序执行，我们直接看范例

代码：06/6.1 promise.html

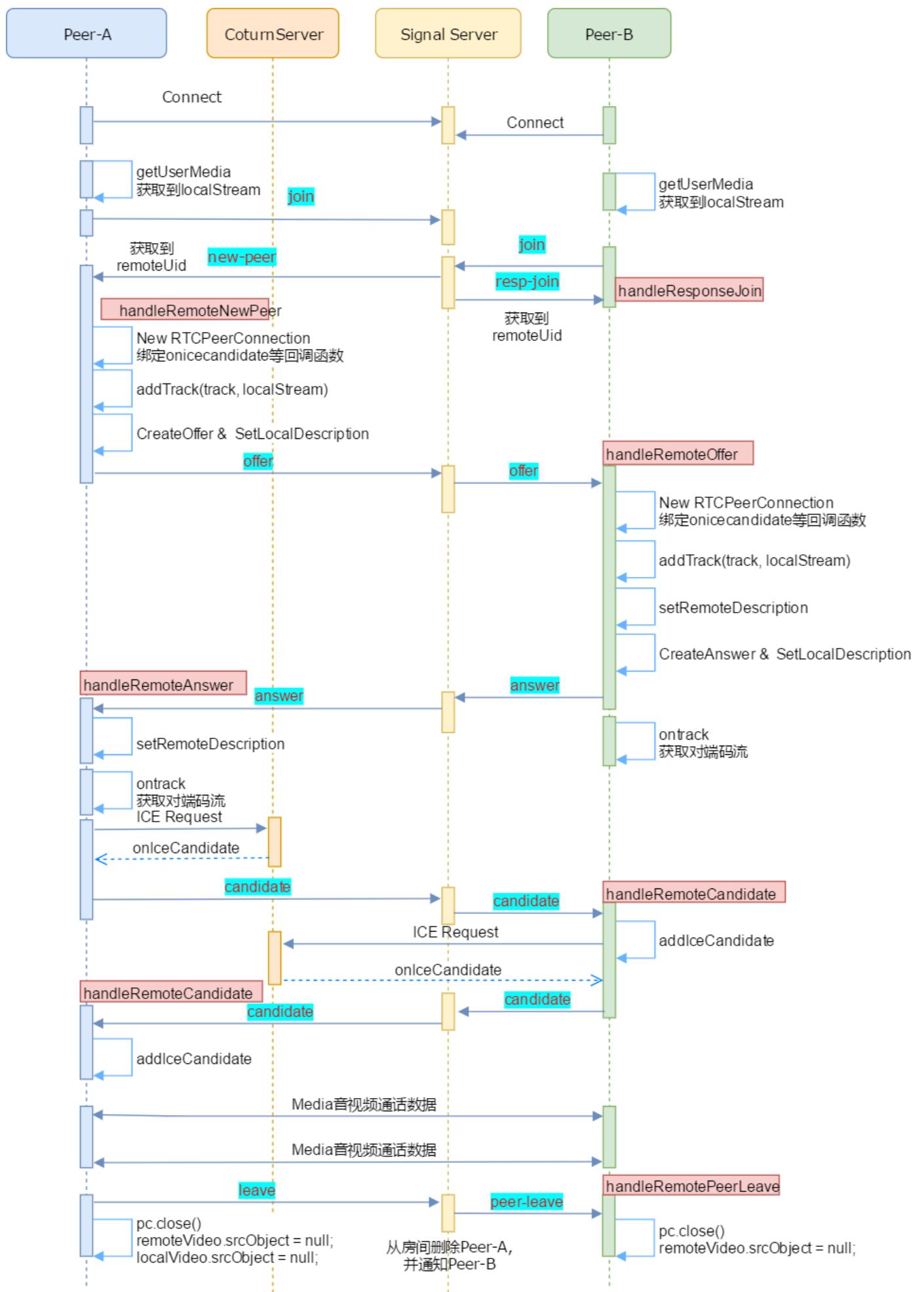
```
1 <html>
2   <head>
3     <title>arrow</title>
4   </head>
5   <body>
6     <script>
7       function taskA() {
8         console.log("Task A");
9       }
10      function taskB() {
11        console.log("Task B");
12        throw new Error("taskB掉坑里了");
13      }
14      function onRejected(error) {
15        console.log("onRejected catch Error: A or B", error);
16      }
17      function finalTask() {
18        console.log("Final Task");
19      }
20
21      var promise = Promise.resolve();
22      promise
23        .then(taskA)
24        .then(taskB)
25        .catch(onRejected)
26        .then(finalTask);
27     </script>
28   </body>
29 </html>
30
```

代码流程



6.1 一对一口话原理

对于我们WebRTC应用开发人员而言，主要是关注[RTCPeerConnection](#)类，我们以（1）信令设计；（2）媒体协商；（3）加入Stream/Track；（4）网络协商 四大块继续讲解通话原理



6.1.1 信令协议设计

采用json封装格式

1. join 加入房间
2. resp-join 当join房间后发现房间已经存在另一个人时则返回另一个人的uid；如果只有自己则不返回
3. leave 离开房间，服务器收到leave信令则检查同一房间是否有其他人，如果有其他人则通知他有人离开
4. new-peer 服务器通知客户端有新人加入，收到new-peer则发起连接请求
5. peer-leave 服务器通知客户端有人离开
6. offer 转发offer sdp
7. answer 转发answer sdp
8. candidate 转发candidate sdp

join

```
1 var jsonMsg = {  
2   'cmd': 'join',  
3   'roomId': roomId,  
4   'uid': localUserId,  
5 };
```

resp-join

```
1 jsonMsg = {  
2   'cmd': 'resp-join',  
3   'remoteUid': remoteUid  
4};
```

leave

```
1 var jsonMsg = {  
2   'cmd': 'leave',  
3   'roomId': roomId,  
4   'uid': localUserId,  
5 };
```

new-peer

```
1 var jsonMsg = {  
2   'cmd': 'new-peer',
```

```
3     'remoteUid': uid  
4 };
```

peer-leave

```
1 var jsonMsg = {  
2     'cmd': 'peer-leave',  
3     'remoteUid': uid  
4 };
```

offer

```
1 var jsonMsg = {  
2     'cmd': 'offer',  
3     'roomId': roomId,  
4     'uid': localUserId,  
5     'remoteUid': remoteUserId,  
6     'msg': JSON.stringify(sessionDescription)  
7 };
```

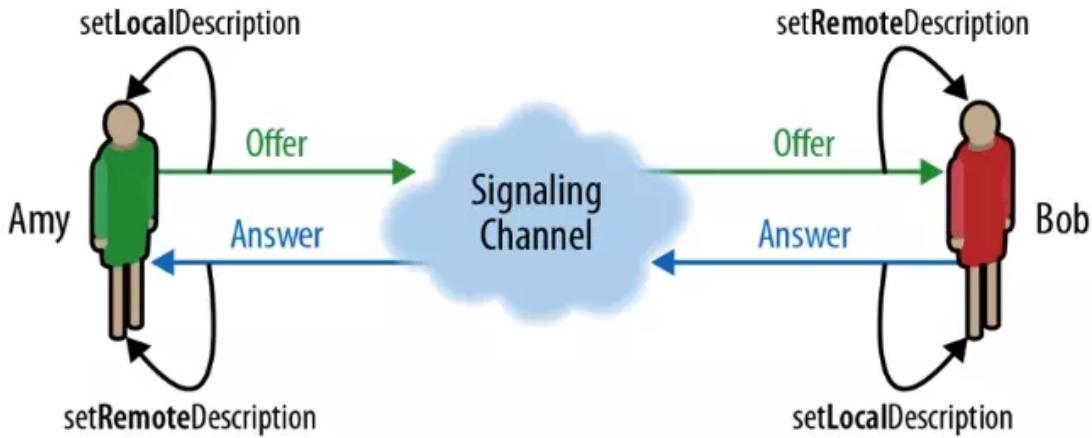
answer

```
1 var jsonMsg = {  
2     'cmd': 'answer',  
3     'roomId': roomId,  
4     'uid': localUserId,  
5     'remoteUid': remoteUserId,  
6     'msg': JSON.stringify(sessionDescription)  
7 };
```

candidate

```
1 var jsonMsg = {  
2     'cmd': 'candidate',  
3     'roomId': roomId,  
4     'uid': localUserId,  
5     'remoteUid': remoteUserId,  
6     'msg': JSON.stringify(candidateJson)  
7 };
```

6.1.2 媒体协商



- `createOffer`

基本格式

```
aPromise = myPeerConnection.createOffer([options]);
```

- [options]

```
1 var options = {  
2     offerToReceiveAudio: true,           // 告诉另一端, 你是否想接收音频, 默认true  
3     offerToReceiveVideo: true,          // 告诉另一端, 你是否想接收视频, 默认true  
4     iceRestart: false,                 // 是否在活跃状态重启ICE网络协商  
5 };
```

<https://webrtc.github.io/samples/src/content/peerconnection/restart-ice/>

iceRestart: 只有在处于活跃的时候, `iceRestart=false`才有作用。

- `createAnswer`

基本格式

```
aPromise = RTCPeerConnection .createAnswer ([ options ]) ; 目前createAnswer的options是无效的。
```

- `setLocalDescription`

基本格式

```
aPromise = RTCPeerConnection .setLocalDescription (sessionDescription) ;
```

- `setRemoteDescription`

基本格式

```
aPromise = pc.setRemoteDescription (sessionDescription) ;
```

6.1.3 加入Stream/Track

- addTrack

基本格式

```
rtpSender = rtcPeerConnection.addTrack(track, stream...);
```

track: 添加到RTCPeerConnection中的媒体轨 (音频track/视频track)

stream: getUserMedia中拿到的流, 指定track所在的stream

6.1.4 网络协商

addIceCandidate

基本格式

```
aPromise = pc.addIceCandidate(候选人);
```

candidate

属性	说明
candidate	候选者描述信息
sdpMid	与候选者相关的媒体流的识别标签
sdpMLineIndex	在SDP中 m= 的索引值
usernameFragment	包括了远端的唯一标识

```
var candidateJson = {
  'label': event.candidate.sdpMLineIndex,
  'id': event.candidate.sdpMid,
  'candidate': event.candidate.candidate
};
```

注意Android和Web端的不同。

6.2 RTCPeerConnection补充

6.2.1 构造函数

语法

```
pc = new RTCPeerConnection([configuration]);
```

configuration 可选

- **bundlePolicy** 一般用max-bundle

balanced: 音频与视频轨使用各自的传输通道

max-compat: 每个轨使用自己的传输通道

max-bundle: 都绑定到同一个传输通道

- **iceTransportPolicy** 一般用all

指定ICE的传输策略

relay: 只使用中继候选者

all: 可以使用任何类型的候选者

- **iceServers**

其由RTCIceServer组成，每个RTCIceServer都是一个ICE代理的服务器

属性	含义
credential	凭据，只有TURN服务使用
credentialType	凭据类型，可以password或oauth
urls	用于连接服中的ur数组
username	用户名，只有TURN服务使用

- **rtcpMuxPolicy** 一般用require

rtcp的复用策略，该选项在收集ICE候选者时使用

选项	说明
negotiate	收集RTCP与RTP复用的ICE候选者，如果RTCP能复用就与RTP复用，如果不能复用，就将他们单独使用
require	只能收集RTCP与RTP复用的ICE候选者，如果RTCP不能复用，则失败

6.2.2 重要事件

- onicecandidate 收到候选者时触发的事件
- ontrack 获取远端流
- onconnectionstatechange PeerConnection的连接状态，参考：<https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/connectionState>

```
1 pc.onconnectionstatechange = function(event) {
2   switch(pc.connectionState) {
3     case "connected":
4       // The connection has become fully connected
5       break;
6     case "disconnected":
7     case "failed":
```

```
8     // One or more transports has terminated unexpectedly or in an error
9     break;
10    case "closed":
11        // The connection has been closed
12        break;
13    }
14 }
```

- `oniceconnectionstatechange` ice连接事件 具体参考：<https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/iceConnectionState>

6.3 手把手实现WebRTC音视频通话

开发步骤

1. 客户端显示界面
2. 打开摄像头并显示到页面
3. websocket连接
4. join、new-peer、resp-join信令实现
5. leave、peer-leave信令实现
6. offer、answer、candidate信令实现
7. 综合调试和完善

6.3.1 客户端显示界面

步骤：创建html页面

主要是input、button、video控件的布局。

6.3.2 打开摄像头并显示到页面

需要通过

6.3.3 websocket连接

6.3.4 join、new-peer、resp-join信令实现

思路：（1）点击加入按钮；（2）响应加入按钮事件；（3）将join发送给服务器；（4）服务器根据当前房间的人数做处理，如果房间已经有人则通知房间里面的人有新人加入（new-peer），并通知自己房间里面是什么人（resp-join）。

6.3.5 leave、peer-leave信令实现

思路：（1）点击离开按钮；（2）响应离开按钮事件；（3）将leave发送给服务器；（4）服务器处理leave，将发送者删除并通知房间（peer-leave）的其他人；（5）房间的其他人在客户

端响应peer-leave事件。

6.3.6 offer、answer、candidate信令实现

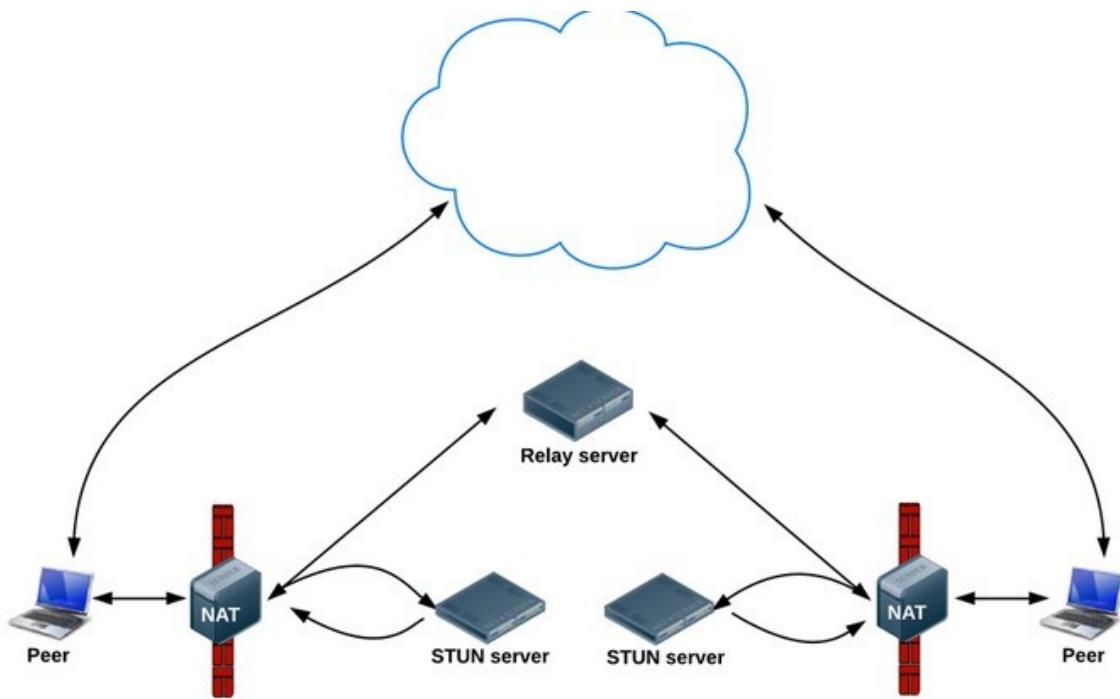
思路：

- (1) 收到new-peer (handleRemoteNewPeer处理)，作为发起者创建RTCPeerConnection，绑定事件响应函数，加入本地流；
- (2) 创建offer sdp，设置本地sdp，并将offer sdp发送到服务器；
- (3) 服务器收到offer sdp 转发给指定的remoteClient；
- (4) 接收者收到offer，也创建RTCPeerConnection，绑定事件响应函数，加入本地流；
- (5) 接收者设置远程sdp，并创建answer sdp，然后设置本地sdp并将answer sdp发送到服务器；
- (6) 服务器收到answer sdp 转发给指定的remoteClient；
- (7) 发起者收到answer sdp，则设置远程sdp；
- (8) 发起者和接收者都收到ontrack回调事件，获取到对方码流的对象句柄；
- (9) 发起者和接收者都开始请求打洞，通过onIceCandidate获取到打洞信息 (candidate) 并发送给对方
- (10) 如果P2P能成功则进行P2P通话，如果P2P不成功则进行中继转发通话。

6.3.7 综合调试和完善

思路：

- (1) 点击离开时，要将RTCPeerConnection关闭 (close)；
- (2) 点击离开时，要将本地摄像头和麦克风关闭；
- (3) 检测到客户端退出时，服务器再次检测该客户端是否已经退出房间。
- (4) RTCPeerConnection时传入ICE server的参数，以便当在公网环境下可以进行正常通话。



启动coturn

```

1 # nohup是重定向命令，输出都将附加到当前目录的 nohup.out 文件中； 命令后加 & ,后台执行起来后按
2 #ctr+c,不会停止
3 sudo nohup turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov &
4 # 前台启动
5 sudo turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov
6 #然后查看相应的端口号3478是否存在进程
7 sudo lsof -i:3478

```

设置configuration，先设置为relay中继模式，只有relay中继模式可用的时候，才能部署到公网去（部署到公网后也先测试relay）。

```

1 var defaultConfiguration = {
2     bundlePolicy: "max-bundle",
3     rtcpMuxPolicy: "require",
4     iceTransportPolicy:"relay", //relay
5     // 修改ice数组测试效果，需要进行封装
6     iceServers: [
7         {
8             "urls": [
9                 "turn:192.168.221.134:3478?transport=udp",
10                "turn:192.168.221.134:3478?transport=tcp"           // 可以插入多个进行备选
11            ],
12            "username": "lqf",
13            "credential": "123456"
14        },
15        {
16            "urls": [
17                "stun:192.168.221.134:3478"

```

```

18         ]
19     }
20   ]
21 };
22 pc = new RTCPeerConnection(defaultConfiguration);

```

relay中继网络状况

03:27:05 PM	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
%ifutil								
03:27:06 PM	ens33	510.00	515.00	409.28	410.74	0.00	0.00	0.00
0.34								
03:27:06 PM	lo	502.00	502.00	399.99	399.99	0.00	0.00	0.00
0.00								

局域网P2P

03:28:41 PM	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
%ifutil								
03:28:42 PM	ens33	0.99	0.99	0.06	0.61	0.00	0.00	0.00
0.00								
03:28:42 PM	lo	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00								

r

6.4 部署到公网

公网防火墙问题，比如 coturn涉及到的3478端口是否开放

启动coturn

```
sudo nohup turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov &
```

```

1 # nohup是重定向命令，输出都将附加到当前目录的 nohup.out 文件中； 命令后加 &，后台执行起来后按
2 # ctr+c，不会停止
3 sudo nohup turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov &
4 # 前台启动
5 sudo turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov
6 #然后查看相应的端口号3478是否存在进程
7 sudo lsof -i:3478

```

编译和启动nginx

```
1 sudo apt-get update
2 #安装依赖: gcc、g++依赖库
3 sudo apt-get install build-essential libtool
4 #安装 pcre依赖库 (http://www.pcre.org/)
5 sudo apt-get install libpcre3 libpcre3-dev
6 #安装 zlib依赖库 (http://www.zlib.net)
7 sudo apt-get install zlib1g-dev
8 #安装ssl依赖库
9 sudo apt-get install openssl
10
11
12 #下载nginx 1.15.8版本
13 wget http://nginx.org/download/nginx-1.15.8.tar.gz
14 tar xvzf nginx-1.15.8.tar.gz
15 cd nginx-1.15.8/
16
17
18 # 配置, 一定要支持https
19 ./configure --with-http_ssl_module
20
21 # 编译
22 make
23
24 #安装
25 sudo make install
26
```

默认安装目录: [/usr/local/nginx](#)

启动: sudo /usr/local/nginx/sbin/nginx

停止: sudo /usr/local/nginx/sbin/nginx -s stop

重新加载配置文件: sudo /usr/local/nginx/sbin/nginx -s reload

产生证书

```
1 mkdir -p ~/cert
2 cd ~/cert
3 # CA私钥
4 openssl genrsa -out key.pem 2048
5 # 自签名证书
6 openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

配置web服务器

(1) 配置自己的证书

```
ssl_certificate /home/lqf/cert/cert.pem; // 注意证书所在的路径  
ssl_certificate_key /home/lqf/cert/key.pem;
```

(2) 配置主机域名或者主机IP

```
server_name 192.168.221.134;
```

(3) web页面所在目录

```
root /mnt/hgfs/ubuntu/ubuntu/module/webrtc/src/06/6.4/client;
```

完整配置文件：/usr/local/nginx/conf/conf.d/webrtc-https.conf

```
1 server {  
2     listen 443 ssl;  
3     ssl_certificate /home/lqf/cert/cert.pem;  
4     ssl_certificate_key /home/lqf/cert/key.pem;  
5     charset utf-8;  
6     # ip地址或者域名  
7     server_name 192.168.221.134;  
8     location / {  
9         add_header 'Access-Control-Allow-Origin' '*';  
10        add_header 'Access-Control-Allow-Credentials' 'true';  
11        add_header 'Access-Control-Allow-Methods' '*';  
12        add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Content-Type,  
Accept';  
13        # web页面所在目录  
14        root /mnt/hgfs/ubuntu/ubuntu/module/webrtc/src/06/6.4/client;  
15        index index.php index.html index.htm;  
16    }  
17 }
```

编辑nginx.conf文件，在末尾}之前添加包含文件

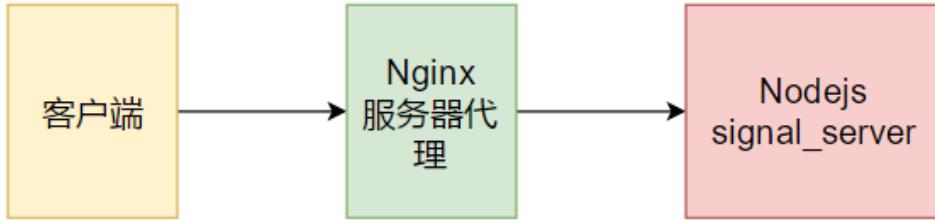
```
1     include /usr/local/nginx/conf/conf.d/*.conf;  
2 }
```

配置websocket代理

ws 不安全的连接 类似http

wss是安全的连接，类似https

https不能访问ws，本身是安全的访问，不能降级做不安全的访问。



ws协议和wss协议两个均是WebSocket协议的SCHEM,两者一个是非安全的,一个是安全的。也是统一的资源标志符。就好比HTTP协议和HTTPS协议的差别。

Nginx主要是提供wss连接的支持, https必须调用wss的连接。

完整配置文件: /usr/local/nginx/conf/conf.d/webrtc-websocket-proxy.conf

```

1 map $http_upgrade $connection_upgrade {
2     default upgrade;
3     '' close;
4 }
5 upstream websocket {
6     server 192.168.221.134:8099;
7 }
8
9 server {
10     listen 8098 ssl;
11     #ssl on;
12     ssl_certificate /home/lqf/cert/cert.pem;
13     ssl_certificate_key /home/lqf/cert/key.pem;
14
15     server_name 192.168.221.134;
16     location /ws {
17         proxy_pass http://websocket;
18         proxy_http_version 1.1;
19         proxy_set_header Upgrade $http_upgrade;
20         proxy_set_header Connection $connection_upgrade;
21     }
22 }

```

wss://192.168.221.134:8098/ws 端口是跟着IP后面

信令服务器后台执行

```

1 sudo nohup node ./signal_server.js &

```

解决websocket自动断开

我们在通话时，出现60秒后客户端自动断开的问题，是因为经过nginx代理时，如果websocket长时间没有收发消息则该websocket将会被断开。我们这里可以修改收发消息的时间间隔。

proxy_connect_timeout :后端服务器连接的超时时间_发起握手等候响应超时时间

proxy_read_timeout:连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间）

proxy_send_timeout :后端服务器数据回传时间_就是在规定时间之内后端服务器必须传完所有的数据

nginx使用proxy模块时，默认的读取超时时间是60s。

完整配置文件：/usr/local/nginx/conf/conf.d/webrtc-websocket-proxy.conf

```
1 map $http_upgrade $connection_upgrade {
2     default upgrade;
3     '' close;
4 }
5 upstream websocket {
6     server 192.168.221.134:8099;
7 }
8
9 server {
10    listen 8098 ssl;
11    ssl_certificate /home/lqf/cert/cert.pem;
12    ssl_certificate_key /home/lqf/cert/key.pem;
13    server_name 192.168.221.134;
14    location /ws {
15        proxy_pass http://websocket;
16        proxy_http_version 1.1;
17        proxy_connect_timeout 4s; #配置点1
18        proxy_read_timeout 6000s; #配置点2，如果没效，可以考虑这个时间配置长一点
19        proxy_send_timeout 6000s; #配置点3
20        proxy_set_header Upgrade $http_upgrade;
21        proxy_set_header Connection $connection_upgrade;
22    }
23 }
24 }
```

客户端 - 服务器 信令：心跳包

keeplive 间隔5秒发送一次给信令服务器，说明客户端一直处于活跃的状态。

6.5 Web和Android实现通话

本章主要内容

1. 获取权限和引入库 (WebRTC、websocket)
2. 信令处理
3. Android WebRTC框架分析

4. Android实战-走读代码

6.5.1 获取权限和引入库

涉及到

1. camera权限
2. audio访问权限
3. 网络访问权限

使用Android studio 3.2 开发

1 Android权限管理

申请静态权限

AndroidManifest.xml文件配置

```
1 <uses-permission android:name="android.permission.CAMERA" />
2 <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
3 <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
4 <uses-permission android:name="android.permission.RECORD_AUDIO" />
5 <uses-permission android:name="android.permission.INTERNET" />
6 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

动态申请权限

```
1 void requestPermissions(
2     @NonNull Activity host, @NonNull String rationale,
3     int requestCode, @Size(min = 1) @NonNull String... perms)
```

申请范例

```
1 String[] perms = {Manifest.permission.CAMERA, Manifest.permission.RECORD_AUDIO};
2     if (!EasyPermissions.hasPermissions(this, perms)) {
3         EasyPermissions.requestPermissions(this, "Need permissions for camera &
4             microphone", 0, perms);
5     }
```

2 引入库

```
1 // WebRTC库
2 implementation 'org.webrtc:google-webrtc:1.0.+'
3 // websocket库
4 implementation "org.java-websocket:Java-WebSocket:1.4.0"
5 // 处理权限库
6 implementation 'pub.devrel:easypermissions:1.1.3'
```

6.5.2 信令处理





和js代码一致，我们重点关注代码的基本流程。

通过 RTCSignalClient类

配置websocket地址 (RTCSignalClient类) (一定要根据自己的IP地址) :

```
1 private static final String WS_URL = "ws://192.168.2.112:8099";
```

主动调用函数

1. joinRoom
2. leaveRoom
3. sendOffer
4. sendAnswer
5. sendCandidate

回调函数

```
1 public interface OnSignalEventListener {  
2     void onConnected();
```

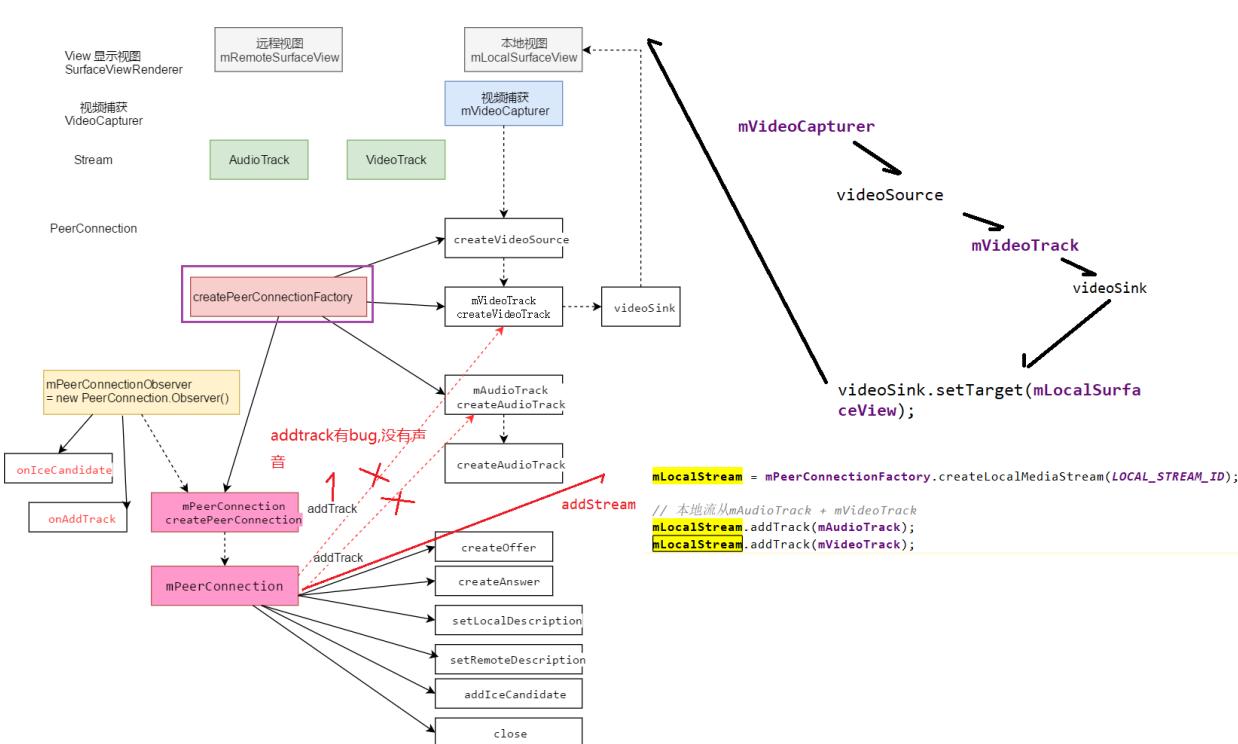
```
3     void onConnecting();
4     void onDisconnected();
5     void onClosse();
6     void onRemoteNewPeer(JSONObject message); // 新人加入
7     void onResponseJoin(JSONObject message); // 加入回应
8     void onRemotePeerLeave(JSONObject message);
9     void onRemoteOffer(JSONObject message);
10    void onRemoteAnswer(JSONObject message);
11    void onRemoteCandidate(JSONObject message);
12 }
```

6.5.3 Android WebRTC框架分析

配置coturn地址（CallActivity类）：

```
1 private static MyIceServer[] iceServers = {
2     new MyIceServer("stun:192.168.2.112:3478"),
3     new MyIceServer("turn:192.168.2.112:3478?transport=udp",
4         "lqf",
5         "123456"),
6     new MyIceServer("turn:192.168.2.112:3478?transport=tcp",
7         "lqf",
8         "123456")
9 };
```

Android端需要使用addstream的方式添加audiotrack 和videotrack，否则会出现web端听不到Android端的的声音。



web端和Android端的candidate格式是有一定的区别。

(1) 发送传输

Android

```
json.put( name: "id", candidate.sdpMid);
json.put( name: "label", candidate.sdpMLineIndex);
json.put( name: "candidate", candidate.sdp);
```

web

```
var candidateJson = {
  'label': event.candidate.sdpMLineIndex,
  'id': event.candidate.sdpMid,
  'candidate': event.candidate.candidate
};
```

(2) 接收处理

Android 端

```
return new IceCandidate(
    json.getString( name: "id"), json.getInt( name: "label"), json.getString( name: "candidate"));
```

接口: `IceCandidate(String sdpMid, int sdpMLineIndex, String sdp)`

Web端

```
var candidateMsg = {
  'sdpMLineIndex': jsonMsg.label,
  'sdpMid': jsonMsg.id,
  'candidate': jsonMsg.candidate
};
```

web端和Android端的sdp有区别。

6.5.4 Android实战-走读代码

权限 在 manifests文件中添加权限

库 在module的gradle中添加依赖库

收发指令

实现Activity的切换

编写signal类使用websocket收发指令

创建PeerConnection

音视频数据采集

创建PeerConnection

媒体协商

协商媒体能力

网络协商

candidate连通检测

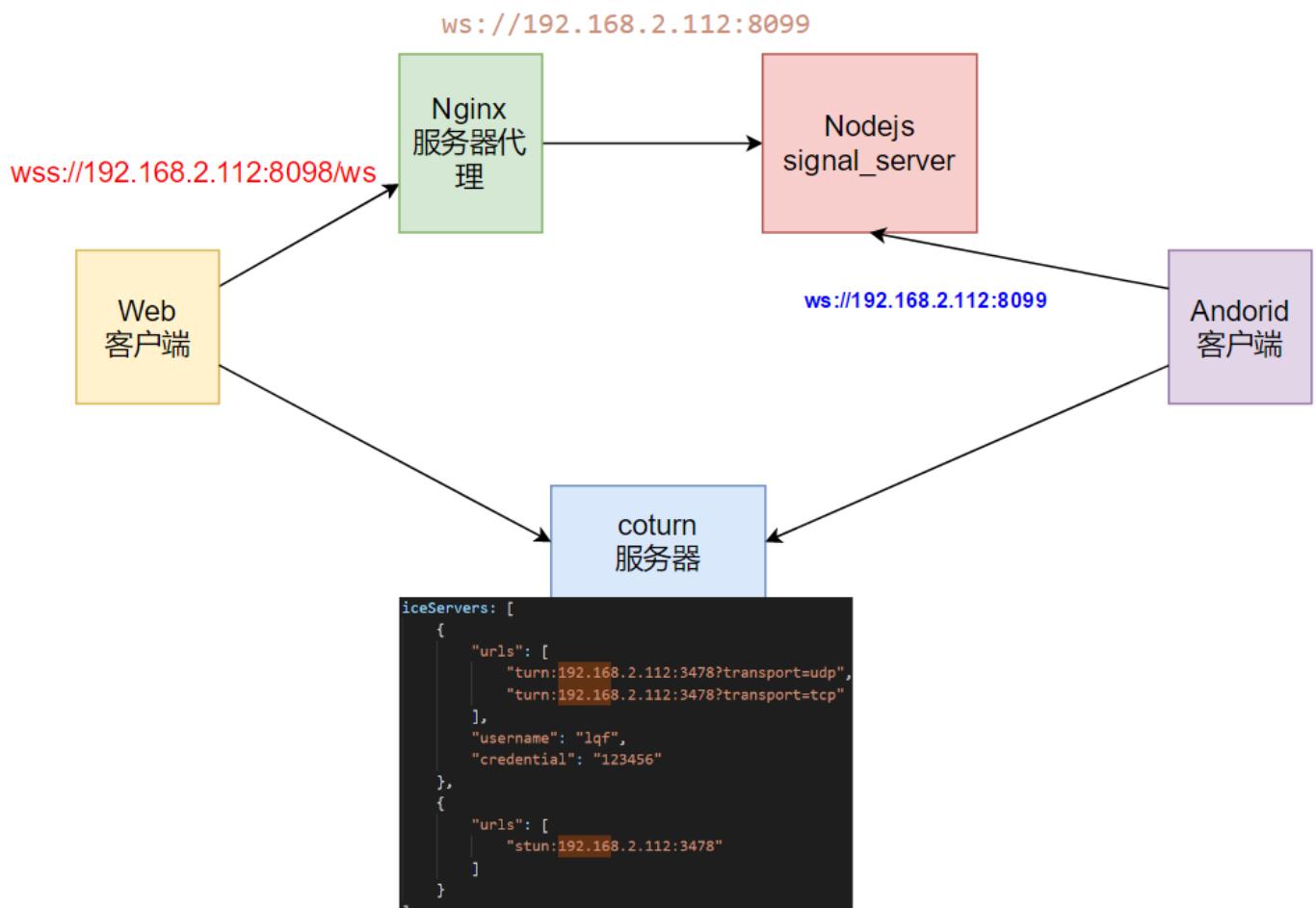
视频渲染

6.5.5 Web和Android通话总结

Web客户端、Android客户端、Nginx服务器一定要按照自己的IP去设置相关的连接，比如websocket和coturn地址。

要启动的服务器：

- (1) nginx
- (2) 信令服务器 signal_server
- (3) 打洞服务器 coturn (stun+turn)



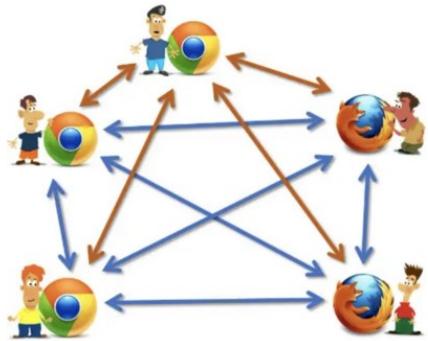
7. 开源方案简介

一、Mesh架构

即：每个端都与其它端互连。以上图最左侧为例，5个浏览器，**一一建立p2p连接**，每个浏览器与其它4个建立连接，总共需要10个连接。如果每条连接占用1m带宽，则每个端上行需要4m，下行带宽也要4m，总共带宽消耗20m。而且除了带宽问题，每个浏览器上还要有音视频“**编码/解码**”，cpu使用率也是问题，一般这种架构只能支持**4-6人左右**，不过优点也很明显，没有中心节点，实现很简单。

Mesh

Connections:	4 10
Uplink:	4 mbps
Downlink:	4 mbps
Total:	20 mbps

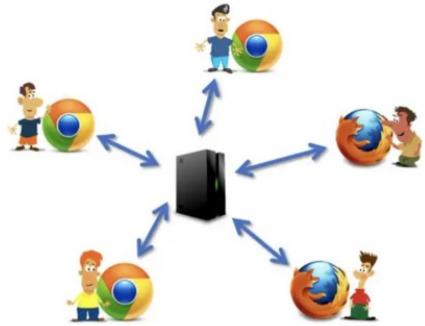


二、MCU (MultiPoint Control Unit)

这是一种传统的中心化架构(上图中间部分)，每个浏览器仅与中心的**MCU服务器连接**，MCU服务器负责所有的视频编码、转码、解码、混合等复杂逻辑，每个浏览器只要1个连接，整个应用仅消耗5个连接，带宽占用(包括上行、下行)共10m，**浏览器端的压力要小很多**，可以支持更多的人同时音视频通讯，比较适合多人视频会议。但是MCU服务器的压力较大，需要较高的配置。

MCU

Connections:	1 5
Uplink:	1 mbps
Downlink:	1 mbps
Total:	10 mbps



三、SFU(Selective Forwarding Unit)

上图右侧部分，乍一看，跟MCU好象没什么区别，但是思路不同，仍然有中心节点服务器，但是中心节点只负责转发，不做太重的处理，所以服务器的压力会低很多，配置也不象MUC要求那么高。但是每个端需要建立一个连接用于上传自己的视频，同时还要有N-1个连接用于下载其它参与方的视频信息。所以总连接数为 5×5 ，消耗的带宽也是最大的，如果每个连接1M带宽，总共需要25M带宽，它的典型场景是1对N的视频互动。

Mesh

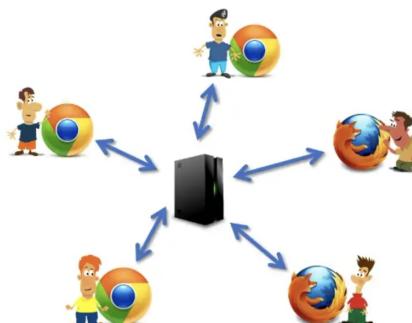
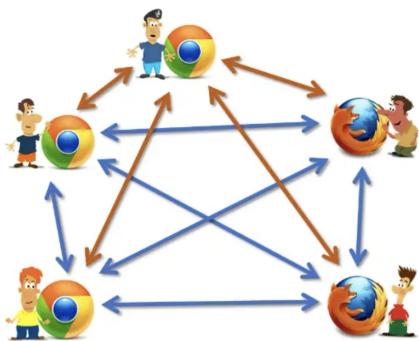
Connections:	4 10
Uplink:	4 mbps
Downlink:	4 mbps
Total:	20 mbps

MCU

Connections:	1 5
Uplink:	1 mbps
Downlink:	1 mbps
Total:	10 mbps

SFU

Connections:	5 25
Uplink:	1 mbps
Downlink:	4 mbps
Total:	25 mbps



如果数据走MCU或者SFU，那就没有P2P可研。

Name	SFU	MCU	Recording	Document	License	Comments
Kurento	✓	✓	✓	✓	Apache v2.0	elasticRTC
Janus	✓	✓	✓	✓	GPLv3	Meetecho Slack
Jitsi	✓		✓	✓	Apache v2.0	
Licode	✓	✓	✓	✓	MIT	Intel CS
Intel CS for webrtc	✓	✓	✓	✓	Free	Licode iqiyi

Janus工程



相关结果约116,000个

虽然它的描述中没有提到“meidia server”，但Janus可以很容易地将其设置为SFU。其最显着的特征之一是其插件架构，可以增强服务的核心功能。有一个演示页面，显示了一些有趣的Janus用例，例如SIP Gateway，屏幕共享等。

官网地址：<https://janus.conf.meetecho.com>

Github地址：<https://github.com/carlhuda/janus>

Janus架构以及基本开发 <https://blog.csdn.net/sonysuqin/article/details/84988120>

Kurento工程



相关结果约49,800个

这是最通用的解决方案之一。它不仅仅是一个媒体服务器，而是构建了一个工具包。Kurento的主要优点是通过引入媒体工作流(meidia workflow)的概念实现了多功能性，它允许在代码中定义媒体流以何种方式传输以及传到哪里。这就允许WebRTC开发者将非常有趣的功能进行集成，例如计算机视觉（例如识别QR码，面部检测），实时媒体修正和与RTP (VoIP) 服务的互操作。Kurento还可以在单个实例中配置成SFU或MCU（或者同时使用）。

官网地址：<http://www.kurento.org/>

Github地址：<https://github.com/Kurento>

Kurento架构 <https://www.jianshu.com/p/8c31479e6083>

Licode工程

相关结果约82,800个

Licode是基于webRTC技术之上的开源项目，通过更便捷（easy, fast and scalable）的接口你可以快速搭建出基于webRTC技术的网络视频会议系统，或者与此类似的系统。你可以通过Try it!对Licode有个更为直观的认识。

官网地址：<http://www.kurento.org/>

Github地址：<https://github.com/lynckia/licode>

Licode精选文章：<https://www.openwebrtc.cn/?cat=6>

Jitsi工程



相关结果约68,400个 (java语言)

Jitsi不仅仅是一个WebRTC媒体服务器，而是围绕者webrtc构建了一整个平台。 Jitsi系列产品包括Jitsi Videobridge（媒体中继，SFU），Jitsi Meet（会议web客户端），Jicofo（Jitsi Conference Focus），Jigasi（Jitsi Gateway to SIP）和Jitsi SIP Phone。 Jitsi平台最吸引人的特性是它包含了在数小时内启动和运行的通信平台的所有功能。它还使用Jingle（XMPP）和功能齐全的Web interface实现了自己的信令。遗憾的是，它没有一个稳固易用的媒体录制功能实现。

官网地址：<https://jitsi.org/>

AppRTC工程

相关结果约134,000个 (唯一没有提供SFU/MCU)

AppRTC是谷歌官网提供的参考方案。[apprtc.debug.js](https://apprtc.appspot.com/)

测试地址：<https://appr.tc/>

github：<https://github.com/webrtc/apprtc>

IOS：<https://github.com/ISBX/apprtc-ios>

Android：<https://github.com/njovy/AppRTCDemo>

8 AppRTC开源方案搭建

见独立的《AppRTC搭建》文档。

9 课程展望

1. 信令完善
 - a. 房间满提示
 - b. 心跳机制
 - c. 多人通话
2. 媒体控制
 - a. 码率控制
 - b. 帧率控制
 - c. 分辨率控制
3. 质量统计
 - a. 丢包率检测
 - b. P2P成功率统计
 - c. 码率统计
4. SDP详解
 - a. 优先编码器
 - b. 控制码率

VIP课程课程 即时通讯项目 涵盖上述知识点。

有问题一定要找darren沟通，或者在群里多问问，大家不要闷头去学。

祝大家能够从课程学到更多的有用知识。

参考

WebRTC是什么 https://blog.csdn.net/ZDK_csdn/article/details/89012853

WebRTC架构简介 <https://blog.csdn.net/fishmai/article/details/69681595>

展望2018： WebRTC技术现状、应用开发与前景 <https://cloud.tencent.com/developer/news/44934>

webrtc 快速搭建 视频通话 视频会议 https://blog.csdn.net/u011077027/article/details/86225524#2_nodenpm_20

Licode—基于webrtc的SFU/MCU实现 <https://zhuanlan.zhihu.com/p/40462946>

网页端实时音视频服务架构与实践 <https://zhuanlan.zhihu.com/p/31442854>

https://github.com/ddssingsong/webrtc_server/blob/master/public/dist/js/SkyRTC.js

webrtc笔记(1): 基于coturn项目的stun/turn服务器搭建 <https://cloud.tencent.com/developer/article/1460709>

菜鸟教程websocket <https://www.runoob.com/html/html5-websocket.html>

RTCPeerConnection <https://www.jianshu.com/p/3f56f6455ac8>

WebRTC 1.0: Real-time Communication Between Browsers <https://w3c.github.io/webrtc-pc/>