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

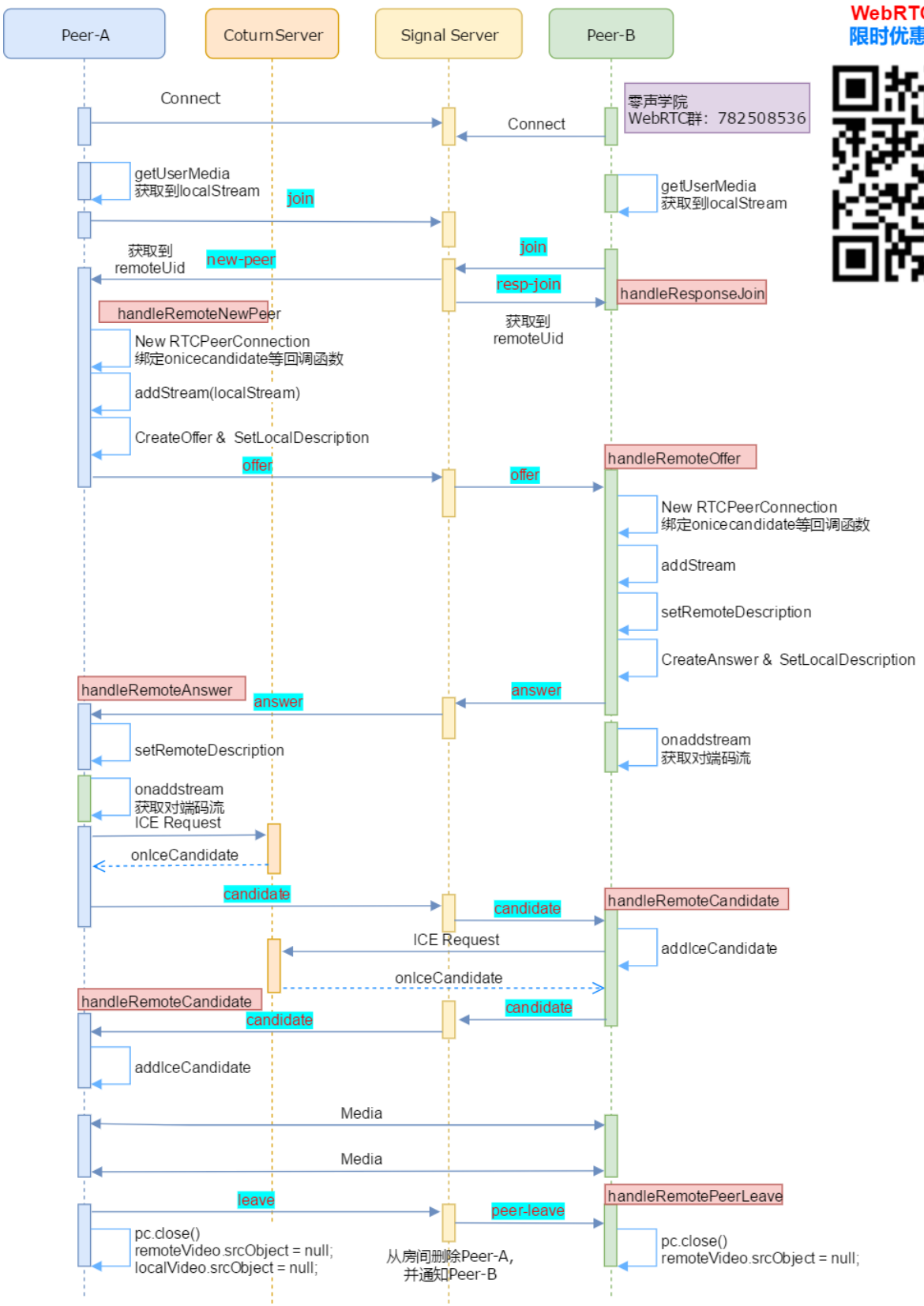
咨询QQ群：782508536

以下是课程的部分文档

本门课程完整内容分为以下章节：

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（含Web to Android）
7. 开源方案介绍

8. AppRTC开源方案搭建



1 WebRTC入门

本章目的：

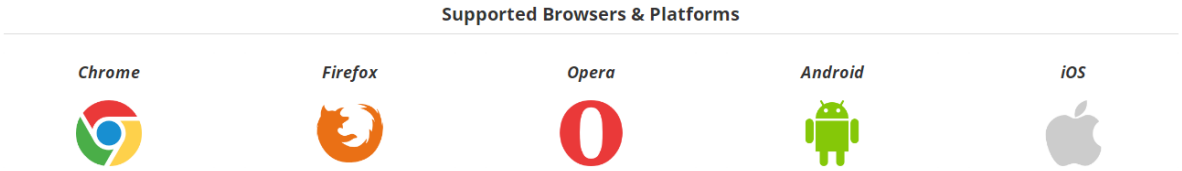
1. 了解什么WebRTC

2. 掌握WebRTC通话原理
3. 学完该课程的收获

1.1 什么是WebRTC

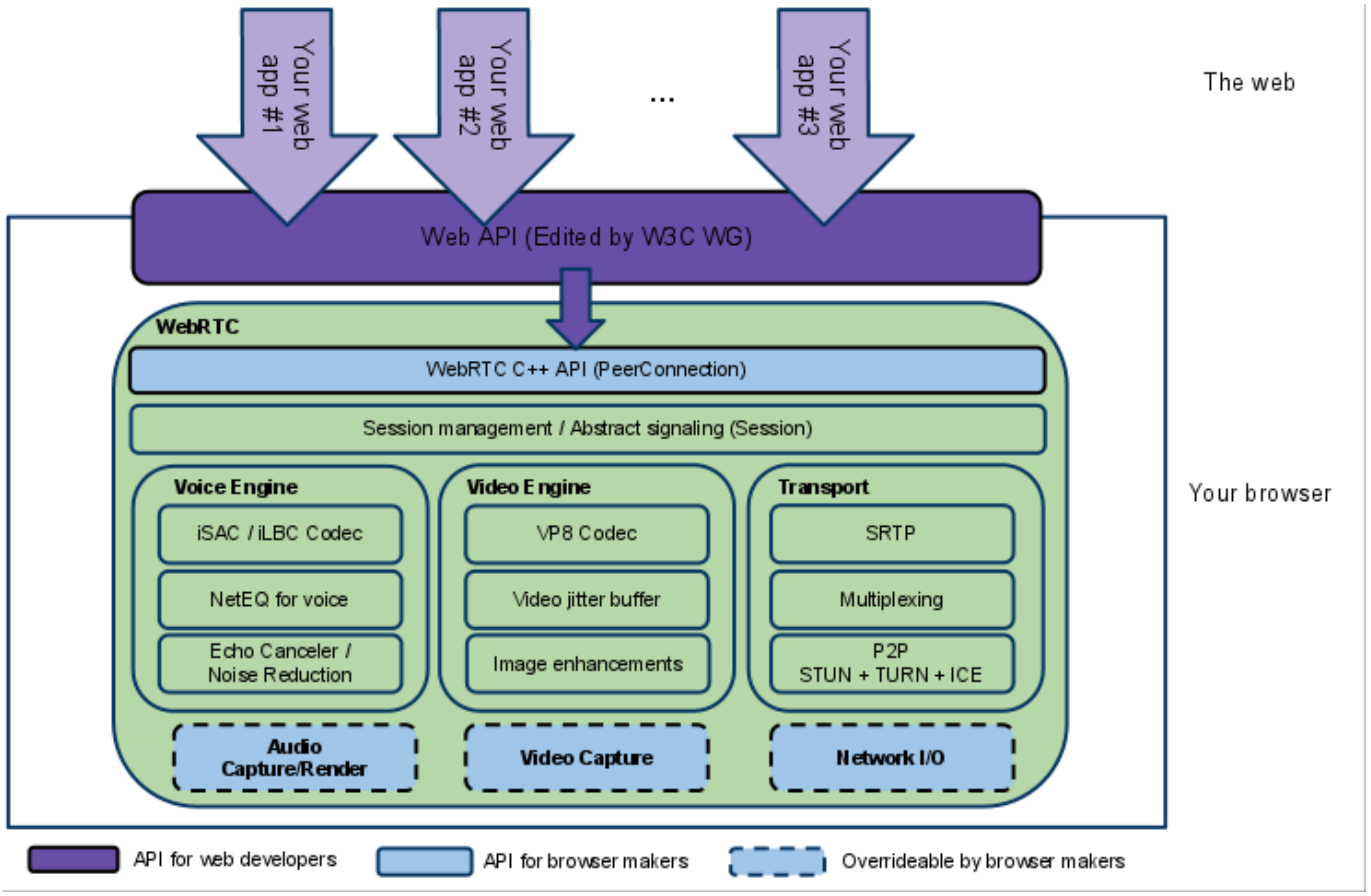
WebRTC（Web Real-Time Communication）是 Google于2010以6829万美元从 Global IP Solutions 公司购买，并于2011年将其开源，旨在建立一个互联网浏览器间的实时通信的平台，让 WebRTC技术成为 H5标准之一。我们看官网（https://webrtc.org）的介绍

> **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.



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/

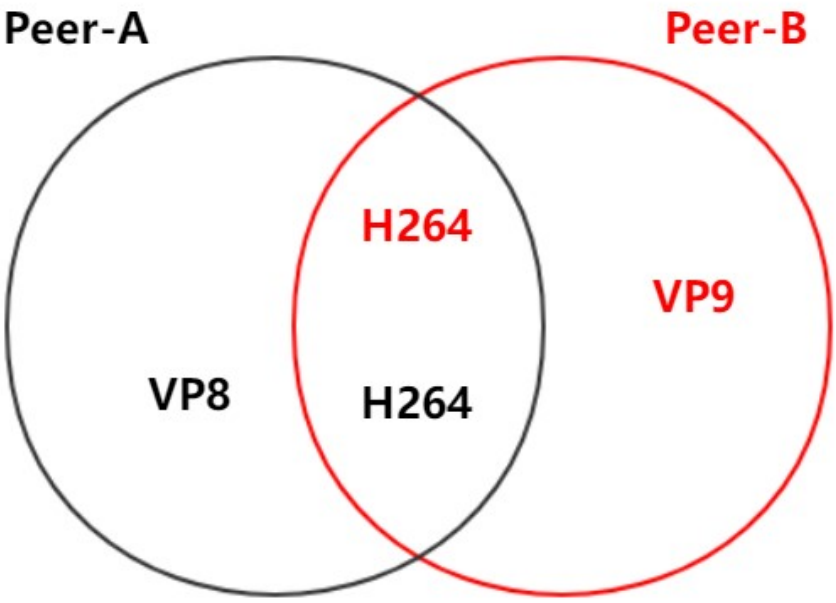
即构科技 https://www.zego.im/

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，要保证二端都正确的编解码，最简单的办法就是取它们的交集H264

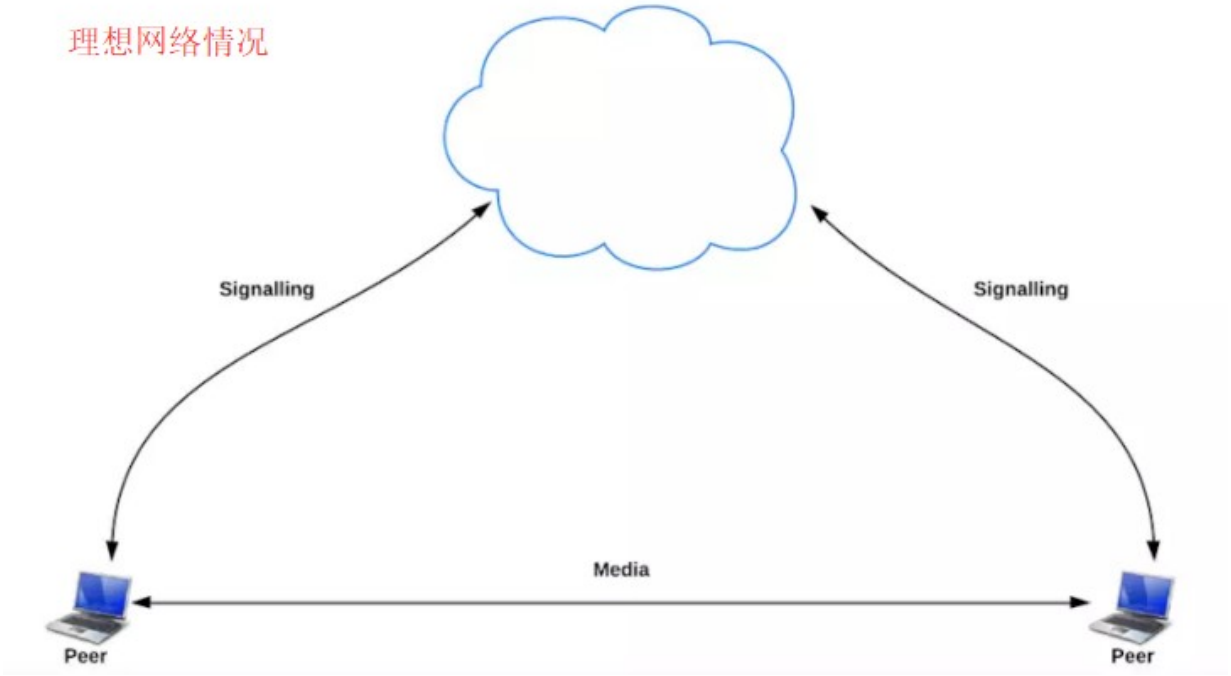
注：有一个专门的协议，称为Session Description Protocol (SDP)，可用于描述上述这类信息，在WebRTC中，参与视频通讯的双方必须先交换SDP信息，这样双方才能知根知底，而交换SDP的过程，也称为"媒体协商"。

2. 网络协商

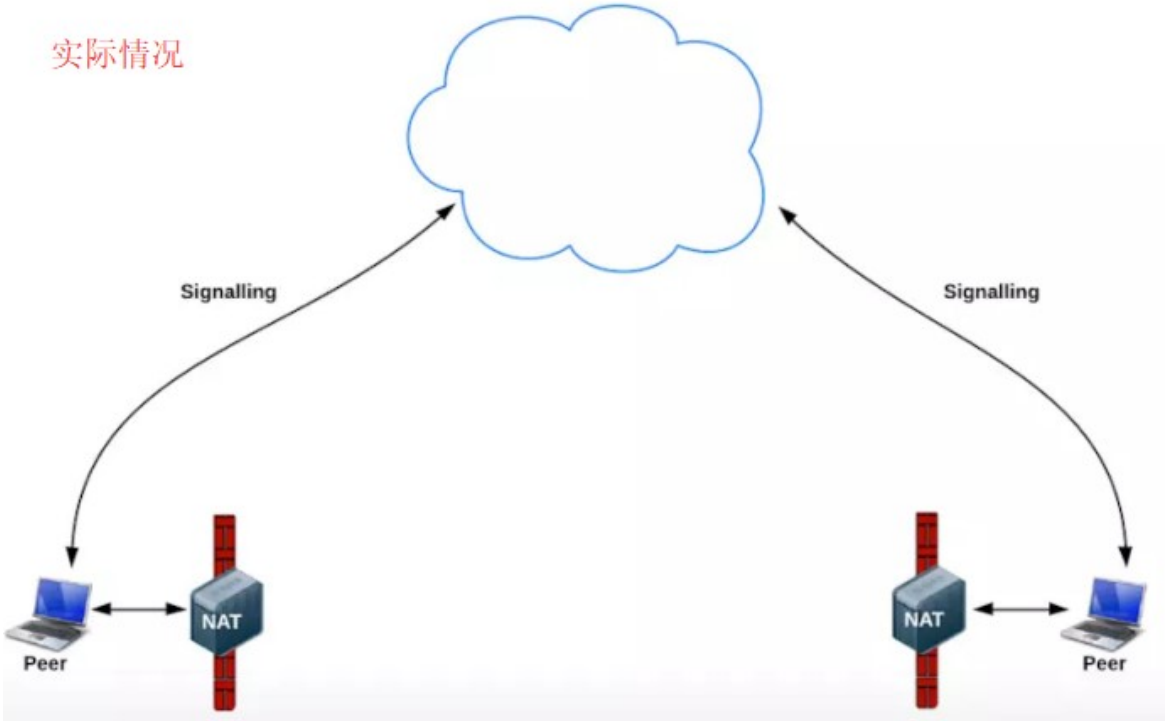
彼此要了解对方的网络情况，这样才有可能找到一条相互通讯的链路

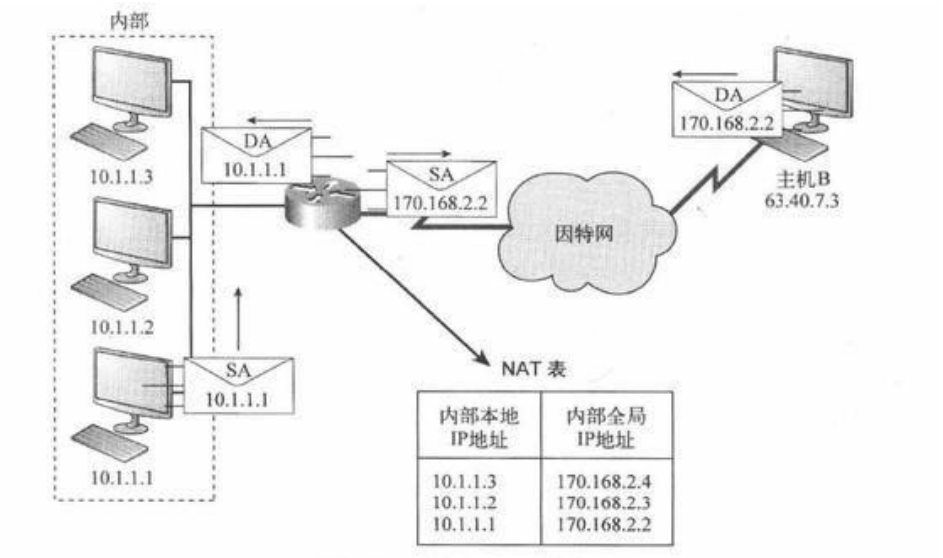
先说结论：(1)获取外网IP地址映射； （2）通过信令服务器（signal server）交换"网络信息"

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。



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：



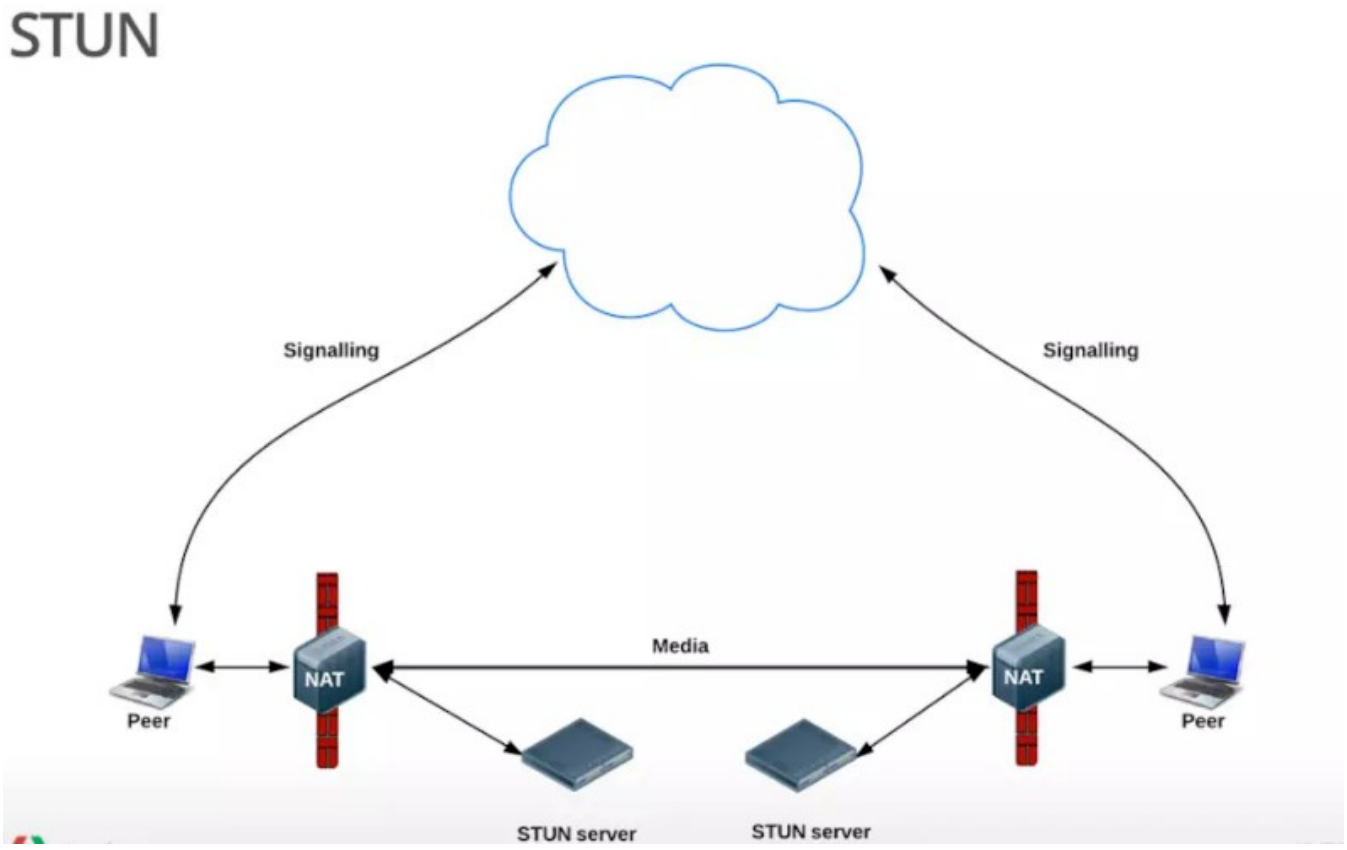


在解决WebRTC使用过程中的上述问题的时候，我们需要用到STUN和TURN。

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。



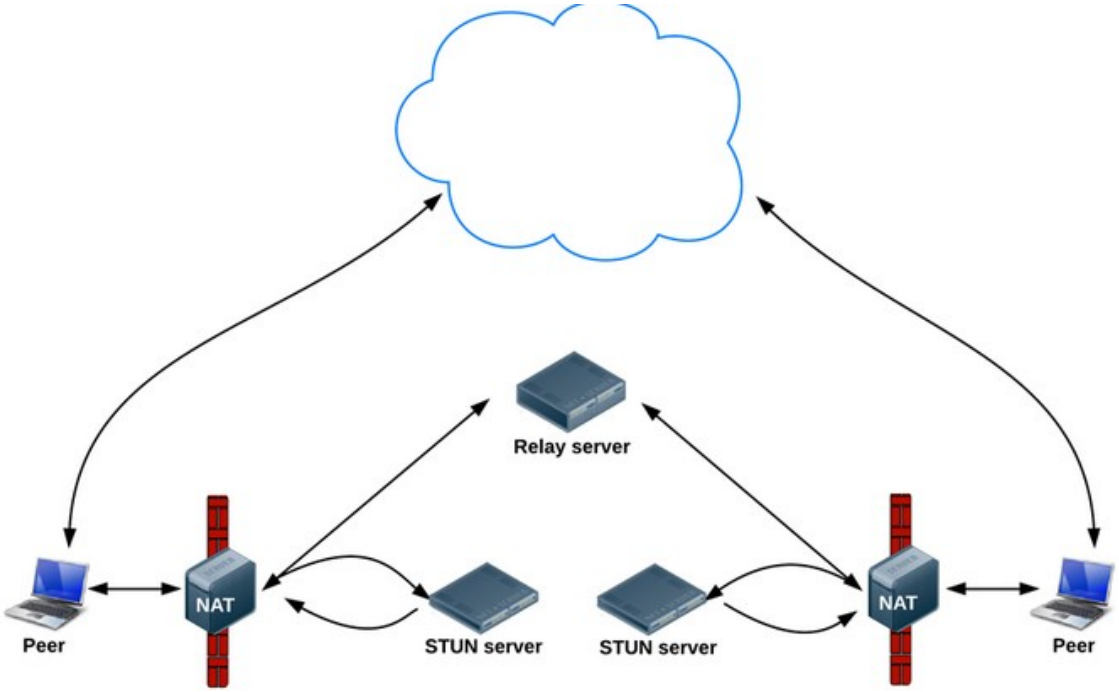
使用一句话说明STUN做的事情就是：告诉我你的公网IP地址+端口是什么。搭建STUN服务器很简单，媒体流传输是按照P2P的方式。

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 一对一通话）

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。

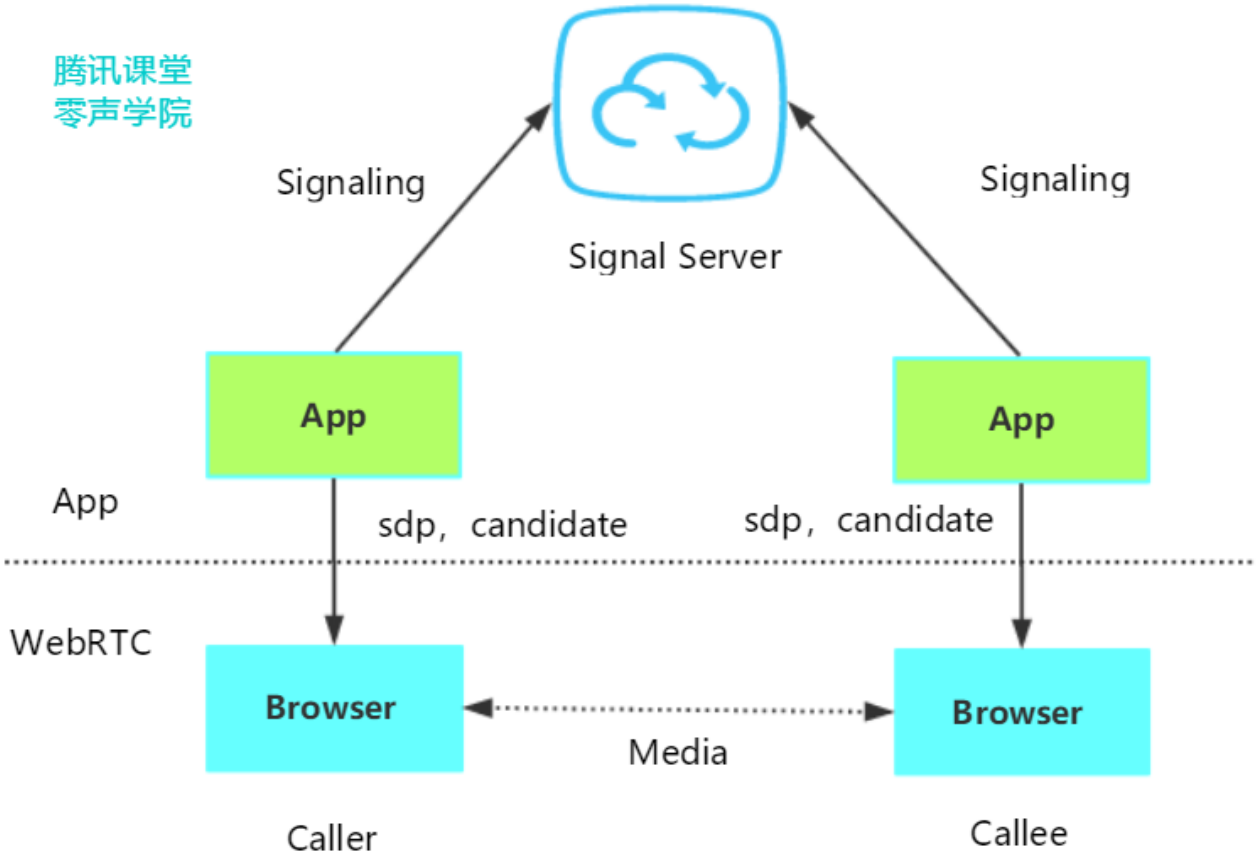
在WebRTC中用来描述 网络信息的术语叫candidate。

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点我们知道了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：

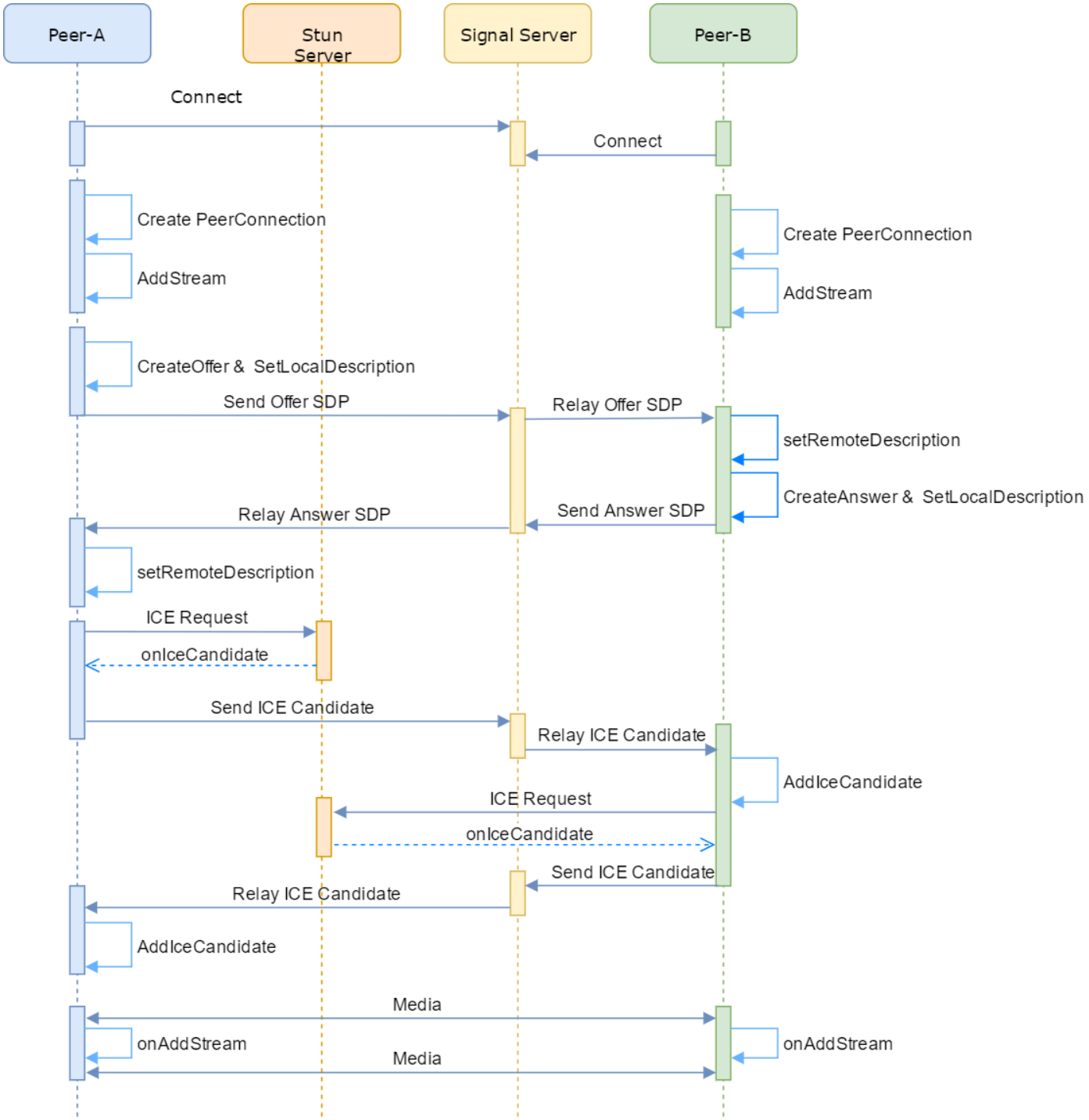
（1）房间管理

（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



在一对一通话场景中，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1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

详解P2P技术中的NAT穿透原理 https://www.jianshu.com/p/f71707892eb2

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/VSCodeUserSetup-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

```
<html>
<body>

<h1>标题1</h1>
```

```
<p>第一个段落.</p>
<p>我的第一个HTML页面</p>
</body>
</html>
```

第一个js程序

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

范例first_js.html

```
<!DOCTYPE html>
<html>
<body>

<h2>Body 中的 JavaScript</h2>

<p id="demo">一个段落。</p>

<button type="button" onclick="myFunction()">试一试</button>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "段落已被更改。";
}
</script>

</body>
</html>
```

2.2 安装 nodejs

源码安装nodejs

1. 下载nodejs

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

2. 解压文件

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

3. 链接执行文件

```
# 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
sudo ln -s /home/lqf/webrtc/nodejs/bin/npm /usr/local/bin/
sudo ln -s /home/lqf/webrtc/nodejs/bin/node /usr/local/bin/

# 看清楚，这个路径是你自己创建的路径，我的路径是/home/dds/webrtc/nodejs

# 查看是否安装，安装正常则打印版本号
node -v
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:

```
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：

```
var http = require('http');

http.createServer(function (request, response) {

    // 发送 HTTP 头部
    // HTTP 状态值: 200 : OK
    // 内容类型: text/plain
    response.writeHead(200, {'Content-Type': 'text/plain'});

    // 发送响应数据 "Hello World"
    response.end('Hello World\n');
}).listen(8888);

// 终端打印如下信息
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：

```
node server.js
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

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

centos系统

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

3.2 编译安装coturn

```
git clone https://github.com/coturn/coturn
cd coturn
./configure
make
sudo make install
```

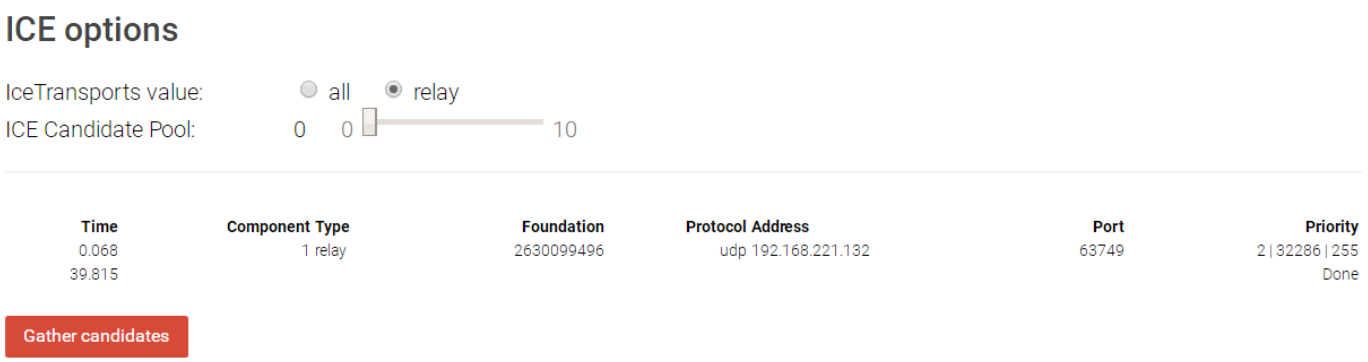
3.3 查看是否安装成功

```
# 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 ctr+c,不会停止
sudo nohup turnserver -L 0.0.0.0 -a -u lqf:123456 -v -f -r nort.gov &
#然后查看相应的端口号3478是否存在进程
sudo lsof -i:3478

```

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/



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

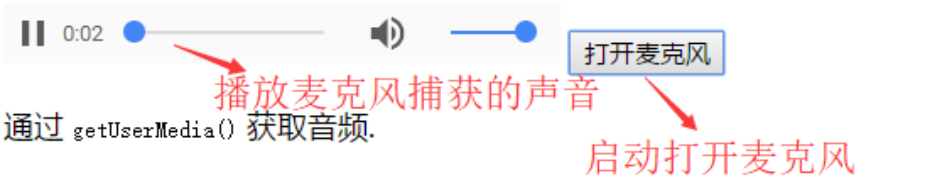
```
<!DOCTYPE html>
```

```
<!--
 *  版权：腾讯课堂 零声学院 https://0voice.ke.qq.com/?tuin=137bb271 .
 *
 *  文件名：video.html
 *  功能：获取视频并将其显示到页面
-->
<html >
    <body >
        <video id="local-video" autoplay playsinline></video>
        <button id="showVideo" >打开摄像头</button>
        <p>通过getUserMedia()获取视频</p>
    </body>
    <script >
        const constraints = {
            audio: false,
            video: true
        };

        // 处理打开摄像头成功
        function handleSuccess(stream) {
            const video = document.querySelector("#local-video");
            video.srcObject = stream;
        }

        // 异常处理
        function handleError(error) {
            console.error("getUserMedia error: " + error);
        }

        function onOpenCamera(e) {

 navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);
        }
        document.querySelector("#showVideo").addEventListener("click", onOpenCamera);
    </script>
</html>
```

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

```
<!DOCTYPE html>
<!--
 *  版权：腾讯课堂 零声学院 https://0voice.ke.qq.com/?tuin=137bb271 .
 *
 *  文件名：audio.html
 *  功能：打开麦克风并在页面播放捕获的声音
-->
<html >
<body>
    <audio id="local-audio" autoplay controls>播放麦克风捕获的声音</audio>
    <button  id="playAudio">打开麦克风</button>

    <p>通过getUserMedia()获取音频</p>
</body>

<script>
    // 约束条件
    const constraints = {
        audio: true,
        video: false
    };

    // 处理打开麦克风成功
    function handleSuccess(stream) {
        const audio = document.querySelector("#local-audio");
        audio.srcObject = stream;
    }

    // 异常处理
    function handleError(error) {
        console.error("getUserMedia error: " + error);
    }

```

```
    function onOpenMicrophone(e) {

 navigator.mediaDevices.getUserMedia(constraints).then(handleSuccess).catch(handleError);
    }
    document.querySelector("#playAudio").addEventListener("click", onOpenMicrophone);
</script>

</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

```
const constraints = {
        audio: false, // 不打开麦克风
        video: true
      };
```

改为

```
const constraints =  {
        audio: true, // 打开麦克风
        video: true
      };
```

具体代码

4.3 video_audio.html

```
<!DOCTYPE html>
<!--
 *  版权：腾讯课堂 零声学院 https://0voice.ke.qq.com/?tuin=137bb271 .
 *
 *  文件名：video.html
 *  功能：获取音视频并将其显示到页面和播放声音
-->
<html>
  <body>

      <video id="local-video" autoplay playsinline></video>
      <button id="showVideo">打开音视频</button>
```

```
      <div id="errorMsg"></div>

      <p>通过 <code>getUserMedia()</code> 获取音视频.</p>


    <script>
      // 设置约束条件，同时打开音频流和视频流
      const constraints = (window.constraints = {
        audio: true,
        video: true
      });

      // 处理打开摄像头+麦克风成功
      function handleSuccess(stream) {
        const video = document.querySelector("#local-video");
        video.srcObject = stream;
      }

      // 处理打开摄像头+麦克风失败
      function handleError(error) {
        console.error("getUserMedia error: " + error);
      }

      async function onOpenAV(e) {

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

      document
        .querySelector("#showVideo")
        .addEventListener("click", onOpenAV);
    </script>
  </body>
</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

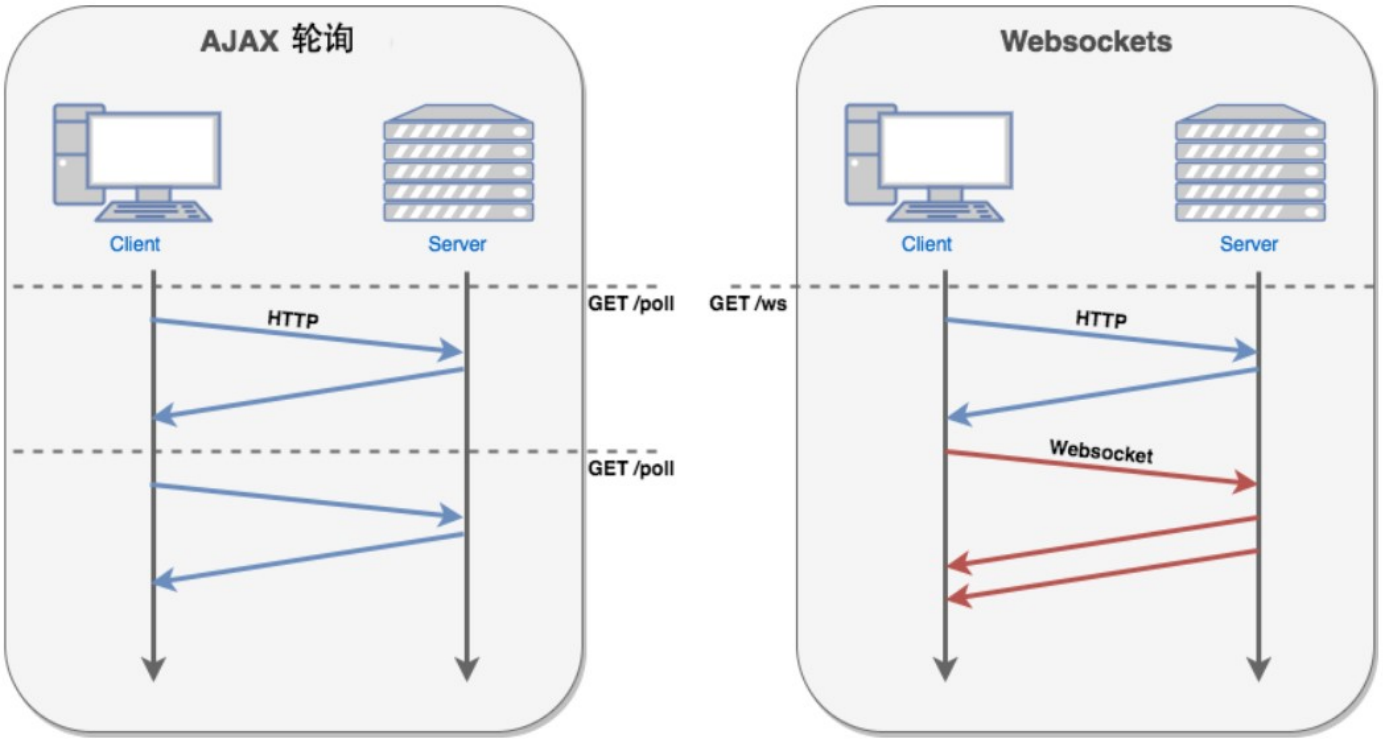
WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

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： • 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，客户端和服务器端的 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

```
cd 工程目录
# 此刻我们需要执行命令：
sudo npm init
#创建package.json文件，系统会提示相关配置，也可以使用命令：
sudo npm init -y
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", callkback)事件的回调函数；

（5）如何判断接收异常，通过on("error", callkback)事件的回调函数；

（6）如何主动发送数据，调用sendText

参考代码

```
var ws = require("nodejs-websocket")

// Scream server example: "hi" -> "HI!!!"
```

```
var server = ws.createServer(function (conn) {
    console.log("New connection")
    conn.on("text", function (str) {    // 收到数据的响应
        console.log("Received "+str)
        conn.sendText(str.toUpperCase()+"!!!") // 发送
    })
    conn.on("close", function (code, reason) { // 关闭时的响应
        console.log("Connection closed")
    })
    conn.on("error", function (err) {    // 出错
        console.log("error:" + err);
    });
}).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

服务端

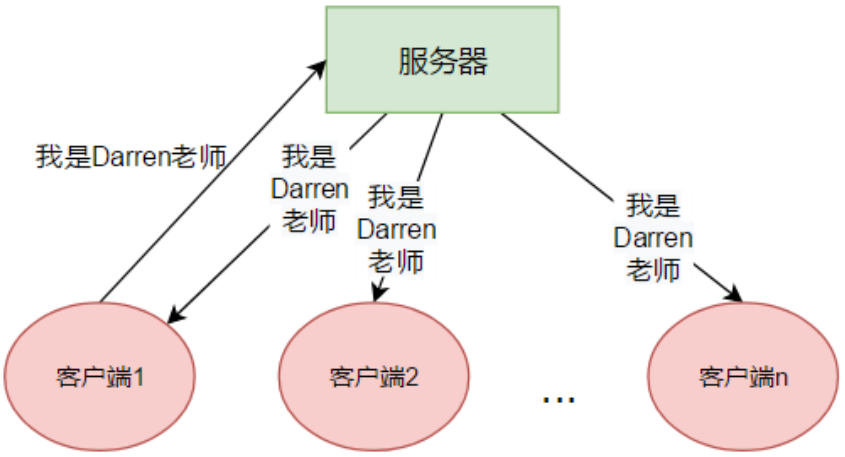
```
lqf@ubuntu:/mnt/hgfs/ubunt
创建一个新的连接--------
创建一个新的连接--------
回复 我是Darren老师
回复 我是King 大叔
```

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

```
<html>

<body>
    <h1>Websocket简易聊天</h1>
    <div id="app">
        <input id="sendMsg" type="text" />
        <button id="submitBtn">发送</button>
    </div>
</body>
<script type="text/javascript">
```

```
    //在页面显示聊天内容
    function showMessage(str, type) {
        var div = document.createElement("div");
        div.innerHTML = str;
        if (type == "enter") {
            div.style.color = "blue";
        } else if (type == "leave") {
            div.style.color = "red";
        }
        document.body.appendChild(div);
    }

    //新建一个websocket
    var websocket = new WebSocket("ws://192.168.221.132:8010");
    //打开websocket连接
    websocket.onopen = function () {
        console.log("已经连上服务器----");
        document.getElementById("submitBtn").onclick = function () {
            var txt = document.getElementById("sendMsg").value;
            if (txt) {
                //向服务器发送数据
                websocket.send(txt);
            }
        };
    };
    //关闭连接
    websocket.onclose = function () {
        console.log("websocket close");
    };
    //接收服务器返回的数据
    websocket.onmessage = function (e) {
        var mes = JSON.parse(e.data);   // json格式
        showMessage(mes.data, mes.type);
    };
</script>

</html>
```

服务器端代码

目录和文件名：05/5.3/server.js

```
var ws = require("nodejs-websocket")
var port = 8010;
var user = 0;

// 创建一个连接
var server = ws.createServer(function (conn) {
    console.log("创建一个新的连接--------");
    user++;
```

```
    conn.nickname="user" + user;
    conn.fd="user" + user;
    var mes = {};
    mes.type = "enter";
    mes.data = conn.nickname + " 进来啦"
    broadcast(JSON.stringify(mes)); // 广播

    //向客户端推送消息
    conn.on("text", function (str) {
        console.log("回复 "+str)
        mes.type = "message";
        mes.data = conn.nickname + " 说:    " + str;
        broadcast(JSON.stringify(mes));
    });

    //监听关闭连接操作
    conn.on("close", function (code, reason) {
        console.log("关闭连接");
        mes.type = "leave";
        mes.data = conn.nickname+" 离开了"
        broadcast(JSON.stringify(mes));
    });

    //错误处理
    conn.on("error", function (err) {
        console.log("监听到错误");
        console.log(err);
    });
}).listen(port);

function broadcast(str){
    server.connections.forEach(function(connection){
        connection.sendText(str);
    })
}
```

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

```
/** ----- ZeroRTCMap ----- */
var ZeroRTCMap = function () {
    this._entrys = new Array();
    // 插入
    this.put = function (key, value) {
        if (key == null || key == undefined) {
            return;
```

```
        }
        var index = this._getIndex(key);
        if (index == -1) {
            var entry = new Object();
            entry.key = key;
            entry.value = value;
            this._entrys[this._entrys.length] = entry;
        } else {
            this._entrys[index].value = value;
        }
    };
    // 根据key获取value
    this.get = function (key) {
        var index = this._getIndex(key);
        return (index != -1) ? this._entrys[index].value : null;
    };
    // 移除key-value
    this.remove = function (key) {
        var index = this._getIndex(key);
        if (index != -1) {
            this._entrys.splice(index, 1);
        }
    };
    // 清空map
    this.clear = function () {
        this._entrys.length = 0;
    };
    // 判断是否包含key
    this.contains = function (key) {
        var index = this._getIndex(key);
        return (index != -1) ? true : false;
    };
    // map内key-value的数量
    this.size = function () {
        return this._entrys.length;
    };
    // 获取所有的key
    this.getEntrys = function () {
        return this._entrys;
    };
    // 内部函数
    this._getIndex = function (key) {
        if (key == null || key == undefined) {
            return -1;
        }
        var _length = this._entrys.length;
        for (var i = 0; i < _length; i++) {
            var entry = this._entrys[i];
            if (entry == null || entry == undefined) {
                continue;
            }
            if (entry.key === key) {// equal
                return i;
            }
```

```
        }
        return -1;
    };
}

function Client(uid, conn, roomId) {
    this.uid = uid;     // 用户所属的id
    this.conn = conn;   // uid对应的websocket连接
    this.roomId = roomId;
    console.log('uid:' + uid +', conn:' + conn + ', roomId: ' + roomId);
}

var roomMap = new ZeroRTCMap();

// Math.random() 返回介于 0（包含） ~ 1（不包含） 之间的一个随机数：
// 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
// substr(2) 舍去0/1位置的字符
console.log('\n\n----------Math.random() ----------');
var randmo = Math.random();
console.log('Math.random() = ' + randmo);
console.log('Math.random().toString(10) = ' + randmo.toString(10));
console.log('Math.random().toString(36) = ' + randmo.toString(36));
console.log('Math.random().toString(36).substr(0) = ' + randmo.toString(36).substr(0));
console.log('Math.random().toString(36).substr(1) = ' + randmo.toString(36).substr(1));
console.log('Math.random().toString(36).substr(2) = ' + randmo.toString(36).substr(2));

console.log('\n\n----------create client ----------');
var roomId = 100;
var uid1 = Math.random().toString(36).substr(2);
var conn1 = 100;
var client1 = new Client(uid1, conn1, roomId);

var uid2 = Math.random().toString(36).substr(2);
var conn2 = 101;
var client2 = new Client(uid2, conn2, roomId);

// 插入put
console.log('\n\n--------------put--------------');
console.log('roomMap put client1');
roomMap.put(uid1, client1);
console.log('roomMap put client2');
roomMap.put(uid2, client2);
console.log('roomMap size:' + roomMap.size());

// 获取get
console.log('\n\n--------------get--------------');
var client = null;
var uid = uid1;
client = roomMap.get(uid);
if(client != null) {
    console.log('get client->' + 'uid:' +  client.uid +', conn:' + client.conn + ', roomId: ' + client.roomId);
} else {
```

```
    console.log("can't find the client of " + uid);
}

uid = '123345';
client = roomMap.get(uid);
if(client != null) {
    console.log('get client->' + 'uid:' +  client.uid +', conn:' + client.conn + ', roomId: '
+ client.roomId);
} else {
    console.log("can't find the client of " + uid);
}

console.log('\n\n--------------traverse--------------');
// 遍历map
var clients = roomMap.getEntrys();
for (var i in clients) {
    let uid = clients[i].key;
    let client = roomMap.get(uid);
    console.log('get client->' + 'uid:' +  client.uid +', conn:' + client.conn + ', roomId: '
+ client.roomId);
}

console.log('\n\n--------------remove--------------');
console.log('roomMap remove uid1');
roomMap.remove(uid1);
console.log('roomMap size:' + roomMap.size());

console.log('\n\n--------------clear--------------');
console.log('roomMap clear all');
roomMap.clear();
console.log('roomMap size:' + roomMap.size());
```