# WebRTC实战开发（web端）

# 课程安排

- WebRTC介绍
- WebRTC API的使用
  - enumerateDevices / getUserMedia / getDisplayMedia
- WebRTC媒体协商
  - RTCPeerConnection添加媒体轨道
  - 媒体协商的流程 createOffer / createAnswer / setLocalDescription / setRemoteDescription
  - 媒体协商的载体 - SDP协议
- WebRTC网络穿越
  - NAT与P2P穿越
  - webrtc网络穿越、STUN / TURN协议工作原理、coturn服务器搭建
  - ICE
  - RTCPeerConnection收集并交换候选者
- 信令服务器的设计
- 实战: 使用WebRTC API实现实时音视频通话

B站视频课程链接: https://www.bilibili.com/video/BV1Ru41177Gt/?vd_source=3ef74b23161ad2021a664761a9a8f634

# 什么是WebRTC

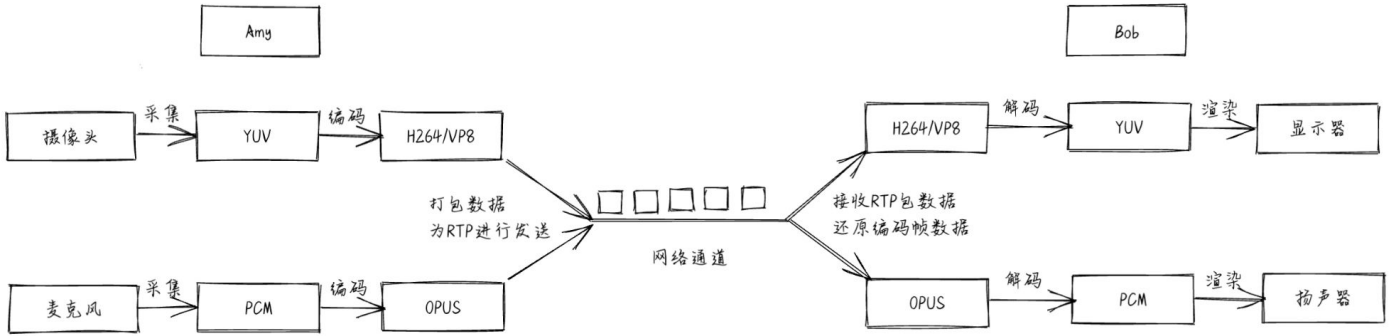
WebRTC (Web Real-Time Communication), 旨在建立一个浏览器间实时通信的平台

谷歌开源

跨平台（Android, IOS, Windows, Linux ...）

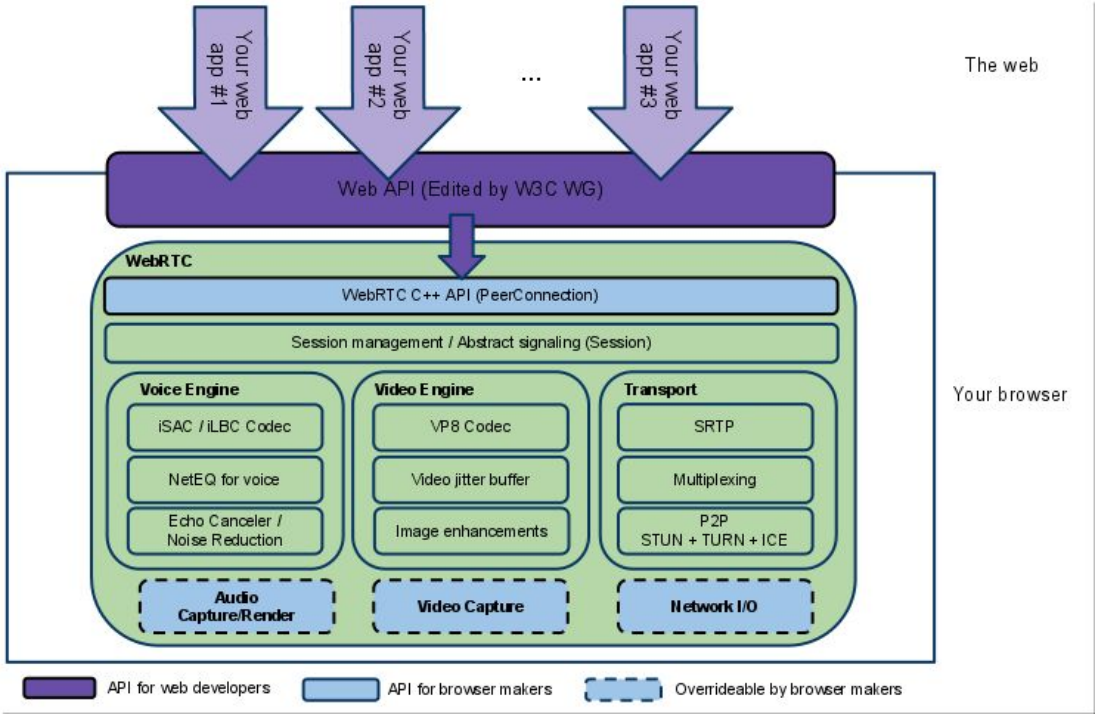
实时传输（提供强大的音视频引擎能力）

# RTC流程涉及的内容




采集:捕获摄像头、麦克风设备数据(yuv, pcm)
编码:将yuv, pcm格式数据编码 (h264/vp8, opus)
传输:将编码帧数据打包传输，需要应对弱网环境挑战(QOS)
解码:将编码后的数据解码成yuv/pcm数据
渲染:将解码后的数据展示到渲染窗口

# WebRTC架构




Your web app

Web API (Edited by W3C WG)
https://developer.mozilla.org/zh-CN/docs/Web/API/WebRTC_API

WebRTC C++ API

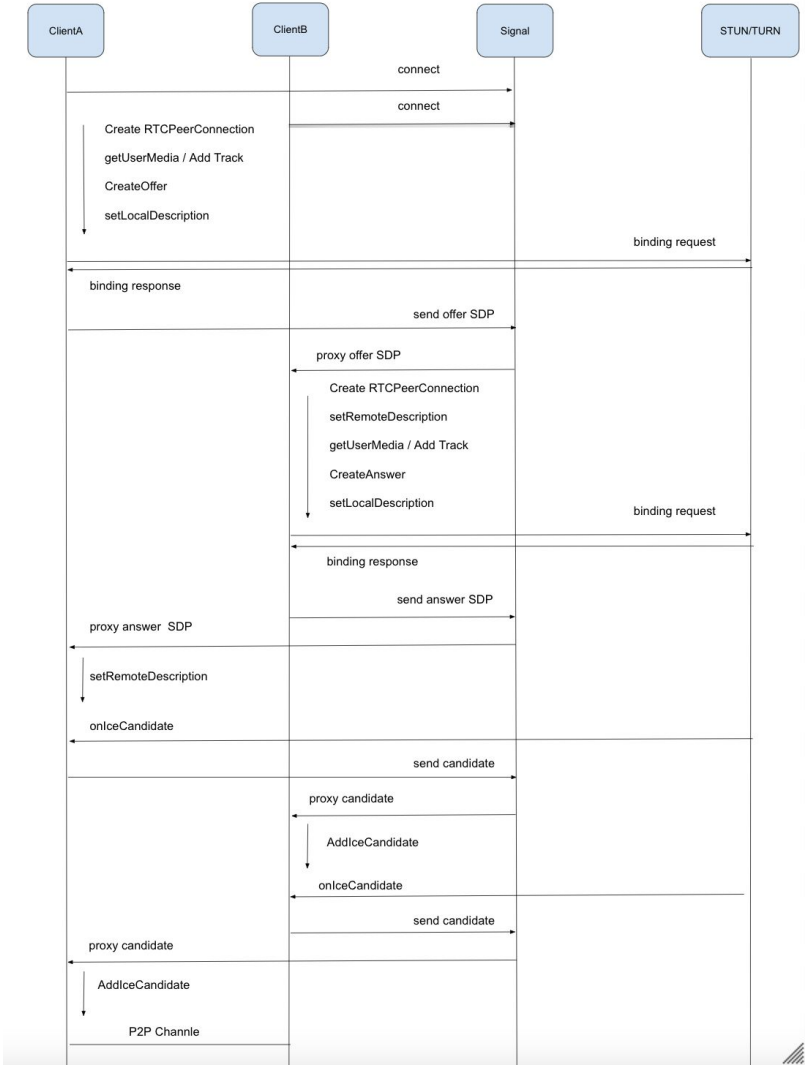
Session management
会话管理

Voice Engine
音频编解码，NetEQ，3A算法

Video Engine
视频编解码，JB，图像增强

Transport
SRTP, STUN+TURN+ICE

# WebRTC实时通话的流程图



# WebRTC API枚举音视频设备

var promise = navigator.mediaDevices.enumerateDevices();

功能说明:枚举音视频设备

返回说明:描述设备的MediaDeviceInfo数组

`MediaDeviceInfo.deviceId` Read only

Returns a string that is an identifier for the represented device that is persisted across sessions. It is un-guessable by other applications and unique to the origin of the calling application. It is reset when the user clears cookies (for Private Browsing, a different identifier is used that is not persisted across sessions).

`MediaDeviceInfo.groupId` Read only

Returns a string that is a group identifier. Two devices have the same group identifier if they belong to the same physical device — for example a monitor with both a built-in camera and a microphone.

`MediaDeviceInfo.kind` Read only

Returns an enumerated value that is either `"videoinput"`, `"audioinput"` or `"audiooutput"`.

`MediaDeviceInfo.label` Read only

Returns a string describing this device (for example "External USB Webcam").

https://developer.mozilla.org/zh-CN/docs/Web/API/MediaDevices/enumerateDevices
https://developer.mozilla.org/en-US/docs/Web/API/MediaDeviceInfo

# WebRTC API采集音视频设备数据

var promise = navigator.mediaDevices.getUserMedia(constaints);

功能说明:采集音视频设备数据

参数说明: Parameters

`constraints`

An object specifying the types of media to request, along with any requirements for each type.

The `constraints` parameter is an object with two members: `video` and `audio`, describing the media types requested. Either or both must be specified. If the browser cannot find all media tracks with the specified types that meet the constraints given, then the returned promise is rejected with `NotFoundError` `DOMException`.

For both `video` and `audio`, its value is either a boolean or an object. The default value is `false`.

- If `true` is specified for a media type, the resulting stream is *required* to have that type of track in it. If one cannot be included for any reason, the returned promise will reject.
- If `false` is specified for a media type, the resulting stream *must not* have that type of track, or the returned promise will reject. Because both `video` and `audio` default to `false`, if the `constraints` object contains neither property or if it's not present at all, the returned promise will always reject.
- If an object is specified for a media type, the object is read as a `MediaTrackConstraints` dictionary.

https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

# 浏览器音视频渲染

## <video>: 视频嵌入元素

**HTML `<video>` 元素** 用于在 HTML 或者 XHTML 文档中嵌入媒体播放器，用于支持文档内的视频播放。你也可以将 `<video>` 标签用于音频内容，但是 `<audio>` 元素可能在用户体验上更合适。

## 尝试一下

HTML Demo: <video> RESET

HTML CSS

```
1  <video controls width="250">
2      <source src="/media/cc0-videos/flower.webm"
   type="video/webm">
3
4      <source src="/media/cc0-videos/flower.mp4"
   type="video/mp4">
5
6      Download the
7      <a href="/media/cc0-
   videos/flower.webm">WEBM</a>
8      or
9      <a href="/media/cc0-
   videos/flower.mp4">MP4</a>
10     video.
11 </video>
12
```

OUTPUT

# WebRTC API采集桌面数据

var promise = navigator.mediaDevices.getDisplayMedia(constraints);

功能说明:采集桌面数据

参数说明:

Parameters

`options` Optional

An optional object specifying requirements for the returned `MediaStream`. The options for `getDisplayMedia()` work in the same as the constraints for the `MediaDevices.getUserMedia()` method, although in that case only `audio` and `video` can be specified. The list of possible option properties for `getDisplayMedia()` is as follows:

`video` Optional

A boolean or a `MediaTrackConstraints` instance; the default value is `true`. If this option is omitted or set to `true`, the stream will contain a video track A value of `true` indicates that the returned `MediaStream` will contain a video track. Since `getDisplayMedia()` requires a video track, if this option is set to `false` the promise will reject with a `TypeError`.

`audio` Optional

A boolean or a `MediaTrackConstraints` instance; the default value is `false`. A value of `true` indicates that the returned `MediaStream` will contain an audio track, if audio is supported and available for the display surface chosen by the user.

https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getDisplayMedia

# WebRTC API适配

WebRTC 标准推出之前，各大浏览器产商使用自己对 WebRTC 规范支持的 API

谷歌：webkitGetUserMedia

火狐：mozGetUserMedia

如果用户自行编写适配不同浏览器的应用层程序，如下所示

```
var getUserMedia = navigator.getUserMedia ||
                   navigator.webkitGetUserMedia ||
                   navigator.mozGetUserMedia;
```

兼容不同浏览器API的适配程序

```
<script src="https://webrtc.github.io/adapter/adapter-latest.js"></script>
```

# MediaStream

MediaStream 用于表示媒体内容，它包含了一系列音视频轨道（MediaStreamTrack）

构造方式：getUserMedia / getDisplayMedia / new MediaStream();

主要方法：

MediaStream.addTrack()

MediaStream.getTracks() / MediaStream.getAudioTracks() / MediaStream.getVideoTracks()

https://developer.mozilla.org/zh-CN/docs/Web/API/MediaStream

# MediaStreamTrack

MediaStreamTrack 表示一个具体的音视频轨道，例如音频轨道，视频轨道

重要属性：

| | |
|---|---|
| MediaStreamTrack.id | 轨道ID |
| MediaStreamTrack.kind | audio/video |
| MediaStreamTrack.label | 轨道标签 |

重要方法：

| | |
|---|---|
| MediaStreamTrack.getConstraints() | 获取轨道约束 |
| MediaStreamTrack.getSettings() | 获取轨道当前设置的属性 |

https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrack/getSettings

# RTCPeerConnection添加媒体轨道

概念：RTCPeerConnection 用于表示一个与远端的对等连接

构造方式：var pc = new RTCPeerConnection([configuration]);

媒体协商的主要方法：

AddTrack：将音视频轨道添加到 RTCPeerConnection，这些轨道将发送到对等端

createOffer / createAnswer：生成offer/answer SDP信息

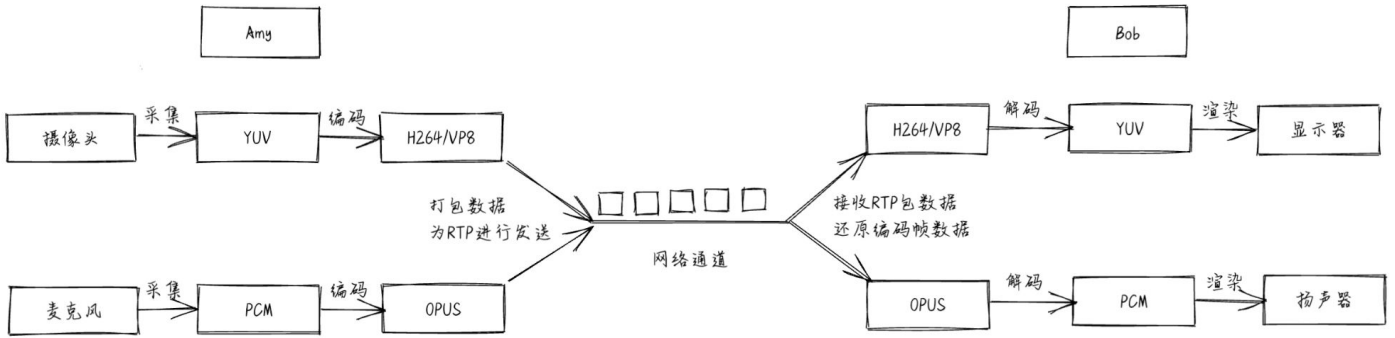
setLocalDescription/setRemoteDescription：将SDP描述信息设置为RTCPeerConnection的本地/远端描述

…... 其他方法后续补充

https://developer.mozilla.org/zh-CN/docs/Web/API/RTCPeerConnection

# 媒体协商

问题:什么是媒体协商？为什么要进行媒体协商？媒体协商包含了哪些信息？




Amy和Bob要进行音视频通话

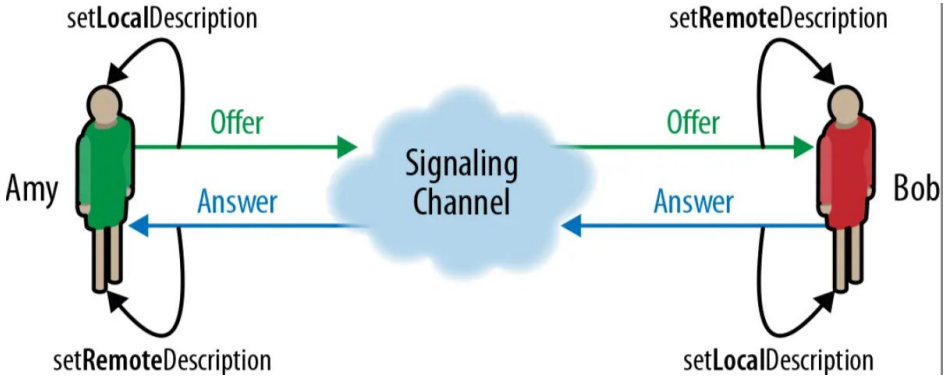
Amy采集摄像头/麦克风数据，进行编码，打包，然后通过网络传输发送给Bob

Bob接收媒体包数据后需要组帧，解码，渲染

双方需要协商使用的编解码器，编解码器的具体参数，通话包含了哪些媒体信息，媒体怎么区分等

# 媒体协商

媒体协商的流程




1. Amy RTCPeerConnection createOffer，将生成的 offerSdp 通过 RTCPeerConnection setLocalDescription 设置为本端描述
2. Amy 将 offerSdp 通过信令服务器发送给 Bob
3. Bob 收到 Amy 发送的 offerSdp，将其设置为 Bob RTCPeerConnection 的远端描述，即调用 setRemoteDescription
4. Bob 调用 RTCPeerConnection createAnswer 生成 answerSdp，然后将 answerSdp 通过 RTCPeerConnection setLocalDescription 设置为本端描述
5. Bob 将 answerSdp 通过信令服务器发送给 Amy
6. Amy 收到 Bob 发送的 answerSdp，通过 setRemoteDescription 将其设置为 Amy RTCPeerConnection 的远端描述

# 媒体协商

媒体协商的载体？怎么表述媒体协商的内容？

SDP协议：https://blog.csdn.net/weixin_38102771/article/details/121259974?spm=1001.2014.3001.5502

# WebRTC网络穿越

思考：两个WebRTC客户端怎么实现端到端通信？

场景1：客户端A和客户端B都处于公网

场景2：客户端A和客户端B有一方处于公网，另外一方处于内网（NAT之后）

场景3：客户端A和客户端B都处于内网

A：同一内部网络

B：不同内部网络

# NAT(Network Address Translator)

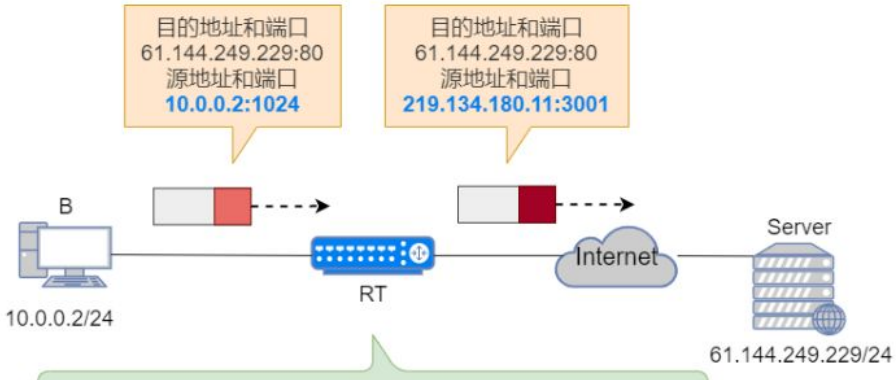
什么是NAT？

网络地址转换，负责将内网地址和公网地址的互相转换

为什么需要NAT？

IPv4地址不够

网络安全考虑




NAT表

| 内部地址 | 内部端口 | 外部地址 | 外部端口 |
|---|---|---|---|
| 10.0.0.1 | 1024 | 219.134.180.11 | 2001 |
| **10.0.0.2** | **1024** | **219.134.180.11** | **3001** |

# NAT类型

- 完全锥型

  内网IP和端口的所有请求都映射到某个外网IP和端口，映射后的地址，其他主机可以通过该地址给内网主机发送数据

- 地址限制锥型

  内网IP和端口的所有请求都映射到某个外网IP和端口，映射后的地址，只有限定的IP地址才能给它发送数据转发给到内网地址，其他IP给它发送数据会被过滤

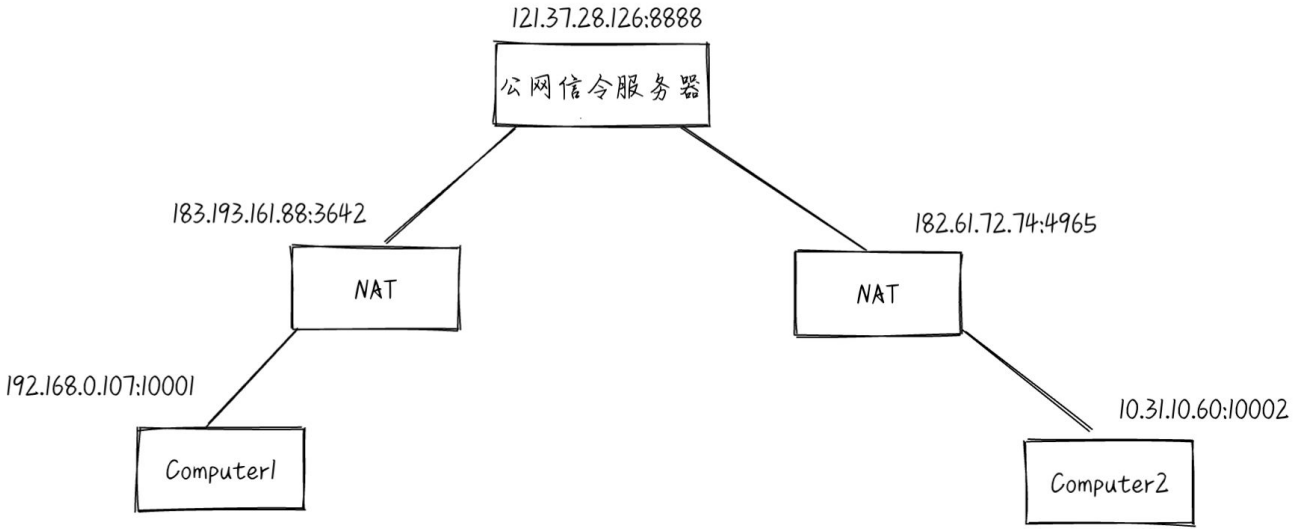
- 端口限制锥型

  在地址限制锥型的基础上增加了端口的限制，也就是映射后的地址只有限定的IP和端口才能给它发送数据转发给到内网地址，其它IP或者其他端口给它发送数据会被过滤

- 对称型

  内网IP和端口的不同目的地址的请求会映射到不同的地址，且它具有端口受限锥型的特性

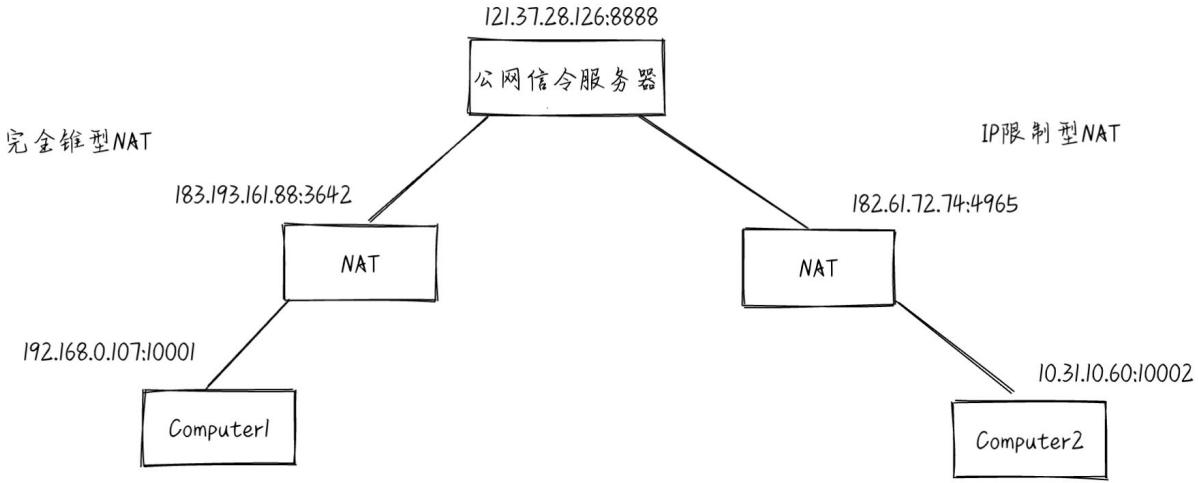
# P2P穿越




假设当前双方都是完全锥型NAT。
Computer1和Computer2都通过公网信令服务器交换彼此的NAT映射后的地址。
那么Computer1给Computer2 NAT映射后的地址发送数据是可以成功的，
Computer2给Computer1的NAT映射后的地址发送数据也是可以成功的

最终通信的地址对 {183.193.161.88:3642, 182.61.72.74:4965}

# P2P穿越



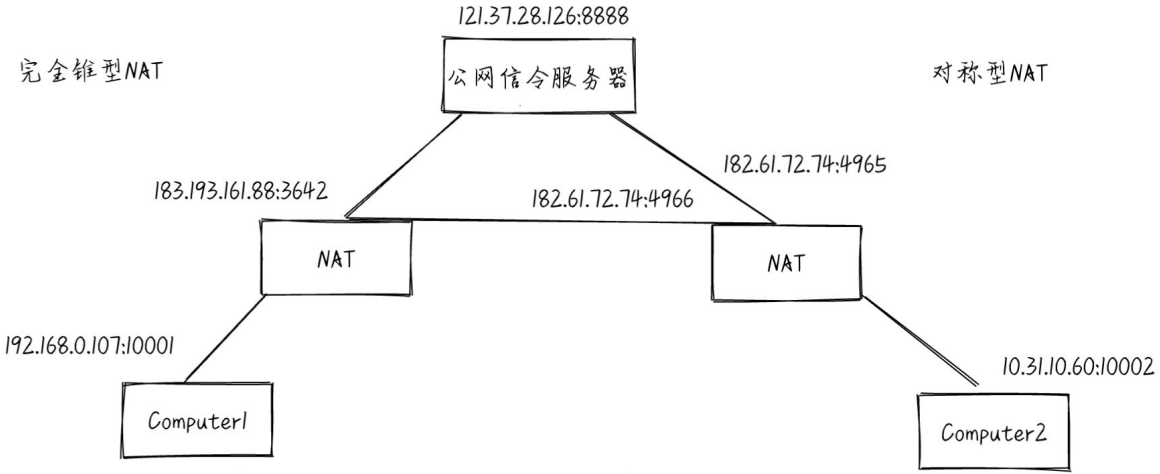

Computer1获取NAT映射后的地址，Computer2获取NAT映射后的地址，通过公网信令服务器互相交换

由于Computer1是完全锥型，Computer2可以通过NAT映射后的地址发送数据给Computer1
在Computer2给Computer1发送数据之前，Computer1是无法通过给Computer2的NAT地址发送数据给到Computer2的

最终通信的地址对是 {183.193.161.88:3642, 182.61.72.74:4965}

对应完全锥型与端口限制型的P2P穿越也是同理

# P2P穿越



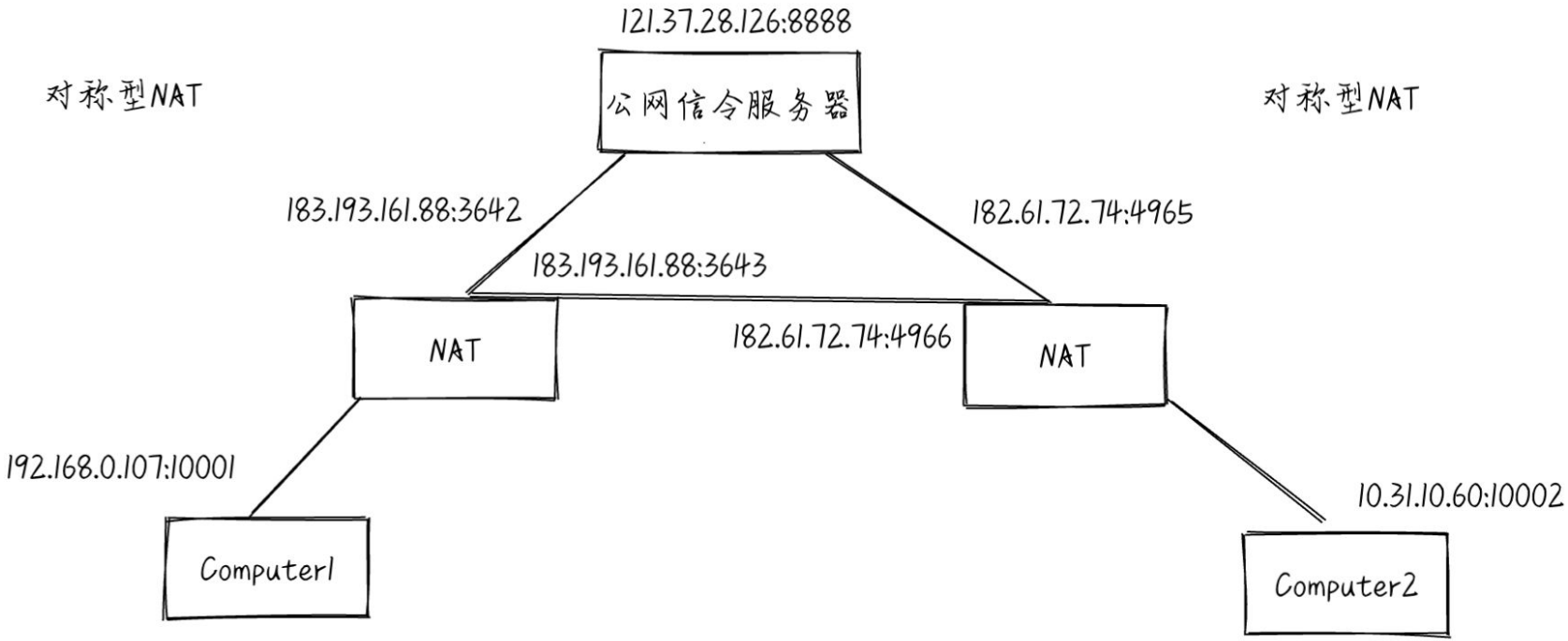

Computer1获取NAT映射后的地址，Computer2获取NAT映射后的地址，通过公网信令服务器互相交换

由于Computer1是完全锥型，Computer2可以通过NAT映射后的地址发送数据给Computer1
但是Computer1无法通过Computer2给的 182.61.72.74:4965 给Computer2通信，因为其具有端口限制的特性

Computer2发送数据给Computer1 NAT映射的地址，会映射成另外一个地址，与之前访问公网信令服务器获取的NAT映射地址不同

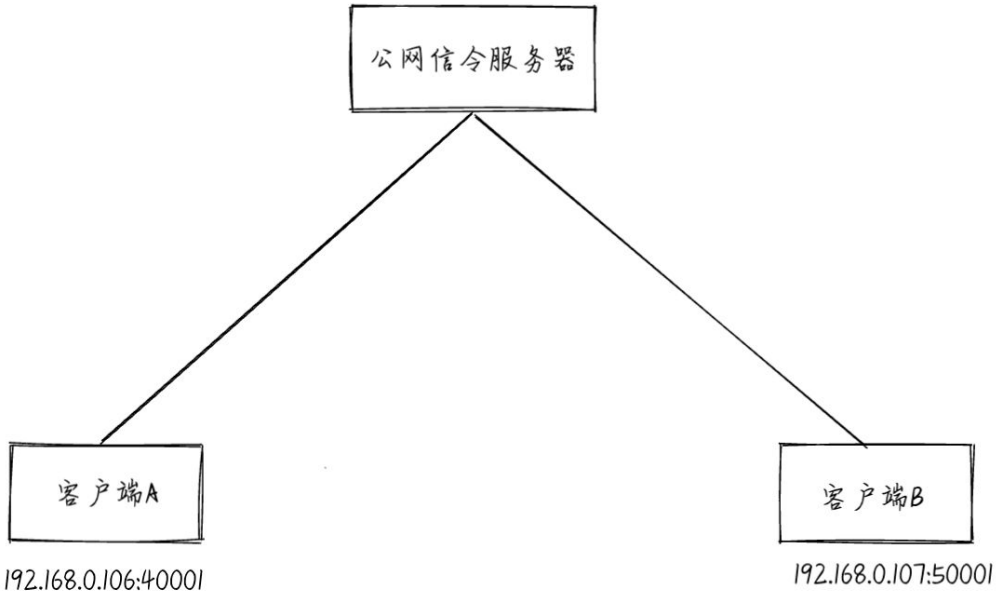
最后Computer1和Computer2通信的地址对是 {183.193.161.88:3642, 182.61.72.74:4966}

# P2P穿越



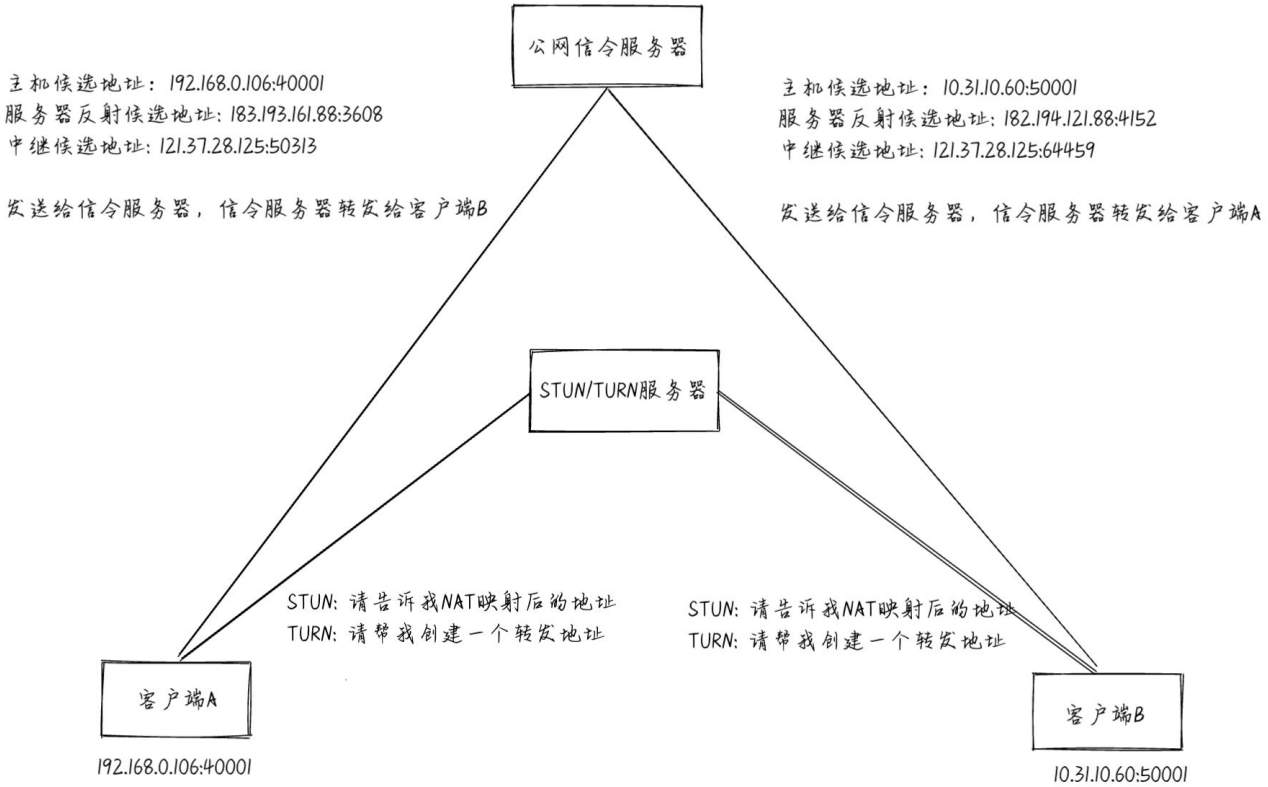

对称型NAT之间无法完成P2P穿越

# WebRTC网络穿越




客户端A和客户端B处于同个内网
只需要由客户端A和客户端B收集本地网卡的地址
然后将地址转发给到对端就可以了

# WebRTC网络穿越



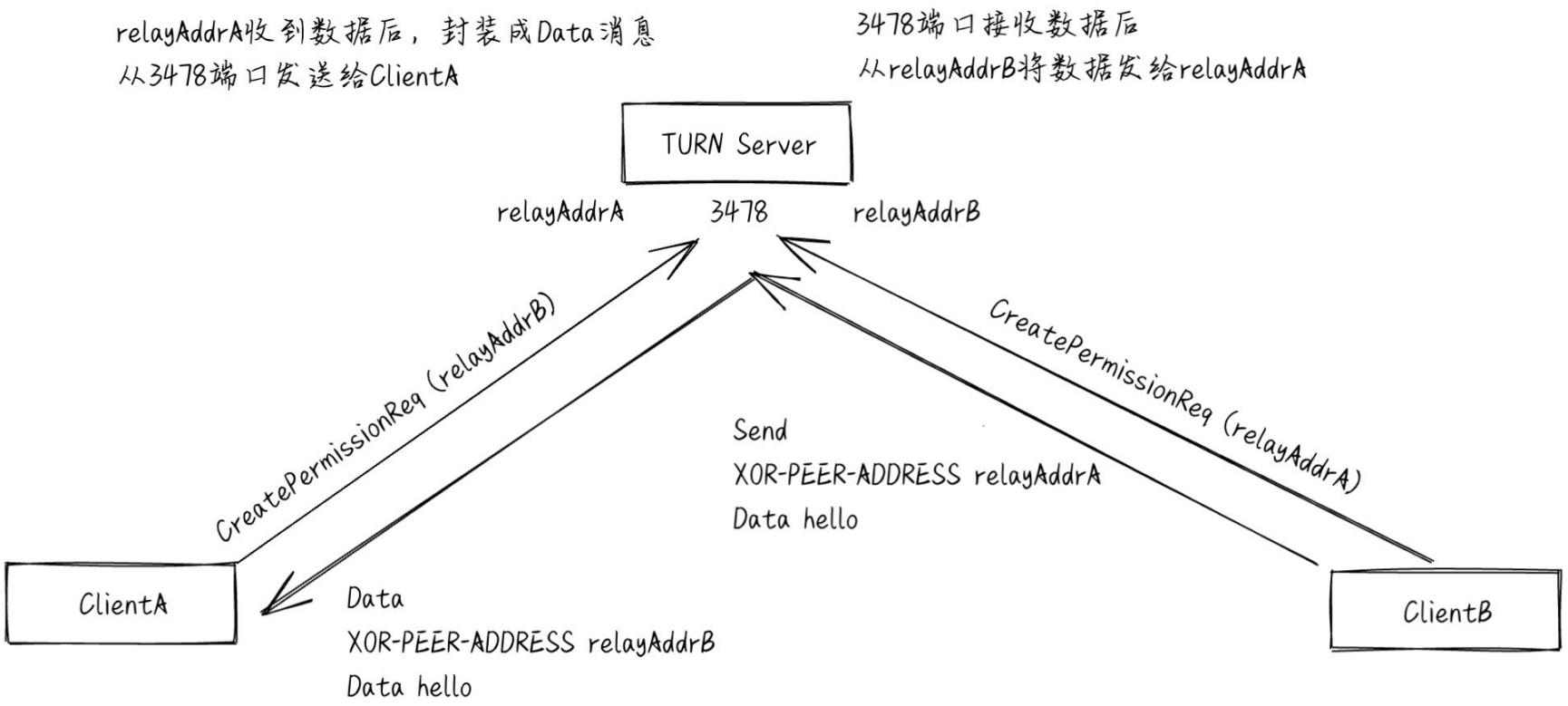
# STUN协议

https://blog.csdn.net/weixin_38102771/article/details/124069029?spm=1001.2014.3001.5502

# TURN协议

https://blog.csdn.net/weixin_38102771/article/details/124530900?spm=1001.2014.3001.5502

# WebRTC使用TURN



# 使用coturn搭建STUN/TURN服务器

sudo apt-get update

sudo apt-get install coturn

修改 /etc/turnserver.conf 文件的配置，如下

listening-port=3478

external-ip=公网云服务器外网IP

lt-cred-mech

user=username1:password1

realm=mycompany.org

启动：turnserver -c /etc/turnserver.conf

https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/

# ICE

ICE: 交互式连接建立 (Interactive Connectivity Establishment)

1. 收集候选地址
2. 交换候选项
3. STUN连接检查
4. 选定地址并启动媒体
5. KeepAlive

# ICE

收集候选地址即收集本端或许可用于接收媒体以建立起对等连接的IP地址和端口

候选地址分类

1. 主机候选者 (host)：网卡的实际地址
2. 服务器反射候选者 (srflx)：STUN服务器回复的STUN binding success response中的反射地址，即最外层NAT的地址
3. 中继候选者 (relay)：请求turn服务器的中继地址
4. 对端反射候选者 (prflx)：从对端发送的STUN binding request中获取的传输地址，本端无法主动收集到该类型的候选地址

# ICE candidate

a=candidate:{foundation} {component-id} {protocol} {priority} {ip} {port} typ {type} generation {generation} ufrag {username} network_id {id} network_cost {cost}


foundation: 相同候选者地址的区分（same type，same ip，from same STUN/TRUN server，same protocol）

component: rtp/rtcp, 分别对应的component-id为1、2

protocol: udp/tcp

priority: 优先级（1, 2^31-1），priority=(2^24)*(type preference) + (2^8) * (local preference) + (2^0) * (256 - component ID)，priority (host > srflx > relay)

type: host/srflx/relay

generation: 代数

network_id: 网卡id

cost: 网络代价

# ICE

交换候选者即通过信令服务器将本端潜在的用于接收媒体的地址发送给对端，对端拿到地址之后可以进行连接检查以找到可以连通的地址。

收到地址候选项后，客户端会开始连通性检查，即检查对端发送过来的候选地址是否可以连通。检查的方式是发送STUN binding request消息，对端收到之后回复STUN binding reponse。

ICE有两种角色，一种是控制方，另一种是受控制方，通常发起offer的一方为控制方，answer的一方为受控制方，选择最终地址对的操作是由控制方决定的。

选定最终地址对后，通话过程中双方仍然要不断发送STUN binding request并响应STUN binding response，以保证连接的有效性。

# RTCPeerConnection收集并交换候选者

事件：icecandidate

发生时机：RTCPeerConnection调用setLocalDescription之后

处理方式：rtcPeerConnection.onicecandidate = (event) => {

```
    if (event.candidate) {
        // 将本地候选者发送给对端
    } else {
        // 表示在本地协商中没有更多的候选者了
    }
}
```

https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/icecandidate_event

# RTCPeerConnection收集并交换候选者

接收到对端候选者后的处理方式 addIceCandidate

语法：addIceCandidate(candidate)

参数：candidate 类型是 RTCIceCandidateInit，包含 candidate, sdpMid, sdpMLineIndex, usernameFragment 成员

https://developer.mozilla.org/en-US/docs/Web/API/RTCIceCandidate/RTCIceCandidate

# RTCPeerConnection ontrack

事件: ontrack

发生时机: 收到对端的媒体轨道

处理方式: 将MediaStreamTrack添加到MediaStream，将MediaStream赋值给<video>展示出对端的媒体画面

```
var remoteMs = new MediaStream();
rtcPeerConnection.ontrack = (event) => {
        remoteMs.addTrack(event.track);
        videoElement.srcObject = remoteMs;
}
```

# RTCPeerConnection构造参数说明

构造方式：var pc = new RTCPeerConnection([configuration]);

```
interface RTCConfiguration {
    bundlePolicy?: RTCBundlePolicy;
    certificates?: RTCCertificate[];
    iceCandidatePoolSize?: number;
    iceServers?: RTCIceServer[];
    iceTransportPolicy?: RTCIceTransportPolicy;
    rtcpMuxPolicy?: RTCRtcpMuxPolicy;
}
```

https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/RTCPeerConnection

# RTCPeerConnection构造参数说明

iceServers?: RTCIceServer[];

作用:ICE服务器配置选项，填STUN/TURN服务地址，它是一个数组，每个元素是一个RTCIceServer对象

JS

```
const configuration = {
  iceServers: [
    {
      urls: "stun:stun.services.mozilla.com",
      username: "louis@mozilla.com",
      credential: "webrtcdemo",
    },
    {
      urls: ["stun:stun.example.com", "stun:stun-1.example.com"],
    },
  ],
};

const pc = new RTCPeerConnection(configuration);
```

# RTCPeerConnection构造参数说明

iceTransportPolicy?: RTCIceTransportPolicy;

作用:指定ICE传输策略，可选值有 all，relay，默认是 all

# RTCPeerConnection构造参数说明

rtcpMuxPolicy?: RTCRtcpMuxPolicy;

作用：收集ICE候选者时指定RTCP复用的策略，可选值有 negotiate 和 require，默认值是 require

negotiate: 同时收集RTP和RTCP的候选者，如果对端能支持在RTP candidate上复用RTCP则复用，如果对端不支持，则将RTP和RTCP分开

require: 只收集RTP的候选者，如果对端不支持RTCP复用RTP候选者，则媒体协商失败

# RTCPeerConnection构造参数说明

bundlePolicy?: RTCBundlePolicy;

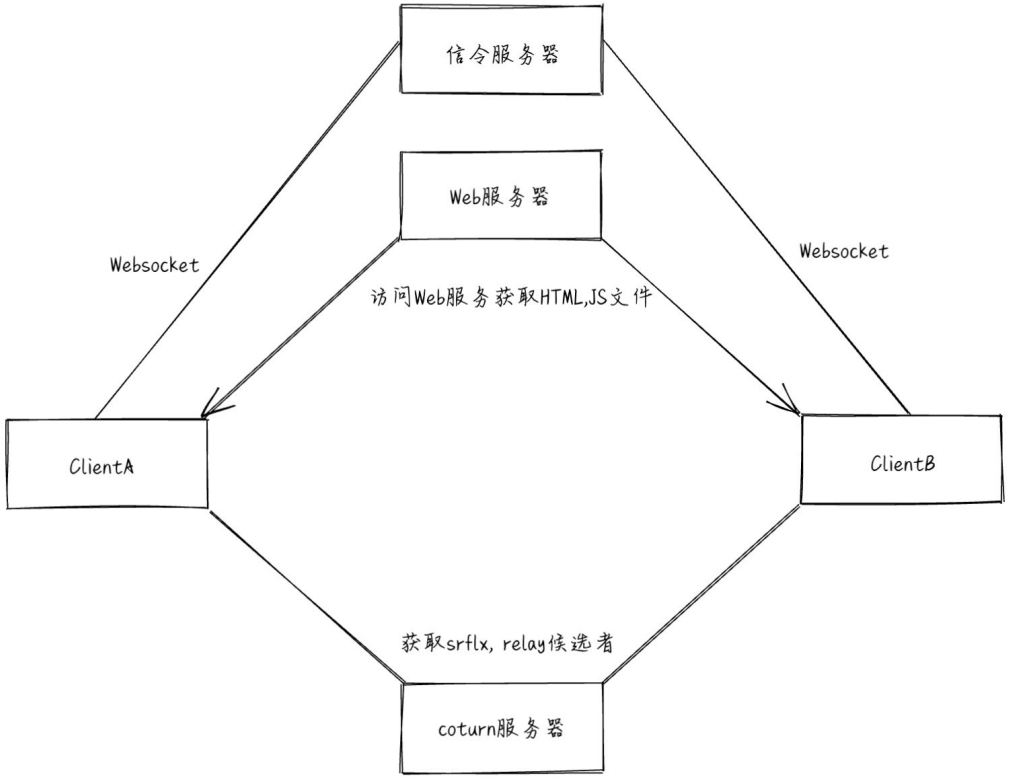
作用:指定当对端不支持 bundle 策略时如何处理候选项的协商，默认值是 balanced，如果双方都是支持 bundle 策略的，则最终所有的轨道都在同一个通道进行传输

balanced: 所有音频轨用一个传输通道，所有视频轨道用另一个传输通道

max-compat: 每一个轨道都使用自己的传输通道

max-bundle: 所有的轨道都绑定到同一个传输通道

# 服务器设计




Web服务器的作用：提供HTML，JS文件

信令服务器的作用

1. 管理用户
2. 转发SDP，candidate等信息或者自定义的业务层信息

对于服务器端的实现，可以选择任何一种你熟悉的语言框架实现，例如golang, nodejs, java等

# 服务器设计 - Web服务器设计

注意事项

1. 由于浏览器的限制，navigator.mediaDevices在https协议下可以正常使用，而在http协议则只允许localhost/127.0.0.1地址访问，其他http协议地址下调用会出现mediaDevices undefined的问题，注意非本地运行时需要使用https

```
/** Available only in secure contexts. */
readonly mediaDevices: MediaDevices;
readonly mediaSession: MediaSession;
```

2. 使用了https，建立websocket连接时也需要访问带tls的地址，否则会被浏览器限制

# 服务器设计 - Web服务器设计

nodejs搭建一个https的静态资源服务器 (ubuntu)

1. 安装nodejs, npm

   sudo apt-get update, sudo apt-get install nodejs npm

2. 编写代码文件，如右图示例
3. 在项目目录下，执行 npm install express serve-index
4. 运行服务器，node webrtc_demo_server.js

```
const https = require('https');
const fs = require('fs');
const express = require('express');
const serveIndex = require('serve-index');

const publicPath = __dirname + '/public';
const options = {
    key  : fs.readFileSync('./certificate/server.key'),
    cert : fs.readFileSync('./certificate/server.crt')
};
const PORT = 4443;

const app = express();
app.use(express.static(publicPath));
app.use('/', serveIndex(publicPath, { icons: true }));

const httpsServer = https.createServer(options, app);
httpsServer.listen(PORT, '0.0.0.0');
```

浏览器访问 https://ip:4443，选择指定 html 文件运行即可

使用openssl生成自签名证书

openssl genrsa -out server.key 2048

openssl req -new -key server.key -out server.csr

openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

# 服务器设计 - 信令服务器设计

客户端与信令服务器使用 websocket 连接，在 websocket 上自定义消息协议格式，客户端加入信令时连接 wss://ip:port?peerId=xxx，xxx 为用户的 peerId，以此作为客户端的标识

当客户端加入连接上信令服务器，服务器给该客户端发送 messageId='CURRENT_PEERS' 的消息，消息格式如下

```
{
        messageId: 'CURRENT_PEERS',
        messageData: {
                peerList: ['pa', 'pc']
        }
}
```

同时服务器给其他所有客户端发送 messageId='PEER_JOIN' 的消息，表示有新客户端加入信令，消息格式如下

```
{
        messageId: 'PEER_JOIN'
        messageData: {
                peerId: 'pb'
        }
}
```

# 服务器设计 - 信令服务器设计

如果有客户端断开信令服务器，服务器给其他所有客户端发送 messageId=‘PEER_LEAVE’ 的消息，消息格式如下

```
{
        messageId: ‘PEER_LEAVE’
        messageData: {
                peerId: ‘pb’
        }
}
```

当客户端A点击客户端B的ID，即发起实时通话，客户端A发送如下消息格式的消息给服务器

```
{
        messageId: ‘PROXY’,
        type: ‘start_call’,
        fromPeerId: ‘pa’,
        toPeerId: ‘pb’
}
```

服务器收到 messageId=’PROXY’ 类型的消息，就查找目的 toPeerId，将消息转发给 toPeerId

# 服务器设计 - 信令服务器设计

客户端B收到 messageId='PROXY', type=''start_call'' 的消息后，开启实时通话，并回复信令服务器以下消息

```
{
        messageId: 'PROXY',
        type: 'receive_call',
        fromPeerId: 'pb',
        toPeerId: 'pa'
}
```

服务器会将该消息转发给客户端A，之后开始媒体协商，候选者交换，消息格式如下

```
{
        messageId: 'PROXY',
        type: 'sdp',
        fromPeerId: 'xx',
        toPeerId: 'yy',
        messageData: {
                sdp: 'sdp内容'
        }
}
```

```
{
        messageId: 'PROXY'
        type: 'candidate',
        fromPeerId: 'xx',
        toPeerId: 'yy',
                messageData: {
                candidate: 'candidate内容'
        }
}
```

# 实战:使用WebRTC实现实时音视频通话

git clone https://gitee.com/mengbieting_0/webrtc_js_demo_code.git