[WebRTC] 웹브라우저로 화상 채팅을 만들 수 있다고?

ehdrms2034 · 2020년 4월 24일

18

WEBRTC

글을 적게된 이유

항상 글 시작은 비슷한 레퍼토리다. 평소에 관심이 많아서..ㅋㅋ
이전에 코딩 과외 플랫폼을 제작한 적이있다.
그때 Web RTC 모듈을 이용해서 플랫폼을 제작했었는데,내부 구성 원리를 모른채 구현하니 뭔가 찝찝합이 남게 됐다.
그래서 이참에 **WEB RTC API를 직접사용하여 서버와 연결 해보자** 라고 생각을 하게 됐고, 포스팅을 하게 됐다.
그런데 막상 공부를 하게 되니, 공부해야할 양은 많고, 참고할만한 레퍼런스는 많지 않거나 영어라서 시간이 오래걸렸다. **(WEB RTC 튜토리얼을 분석하는데도 거의 일주일이란 시간을 다 쓴 것 같다.).**
그러다 보니 아 이걸 포스팅으로 남겨 놓지 않으면 손해겠구나라고 생각해서 글을 쓰게 됐다.

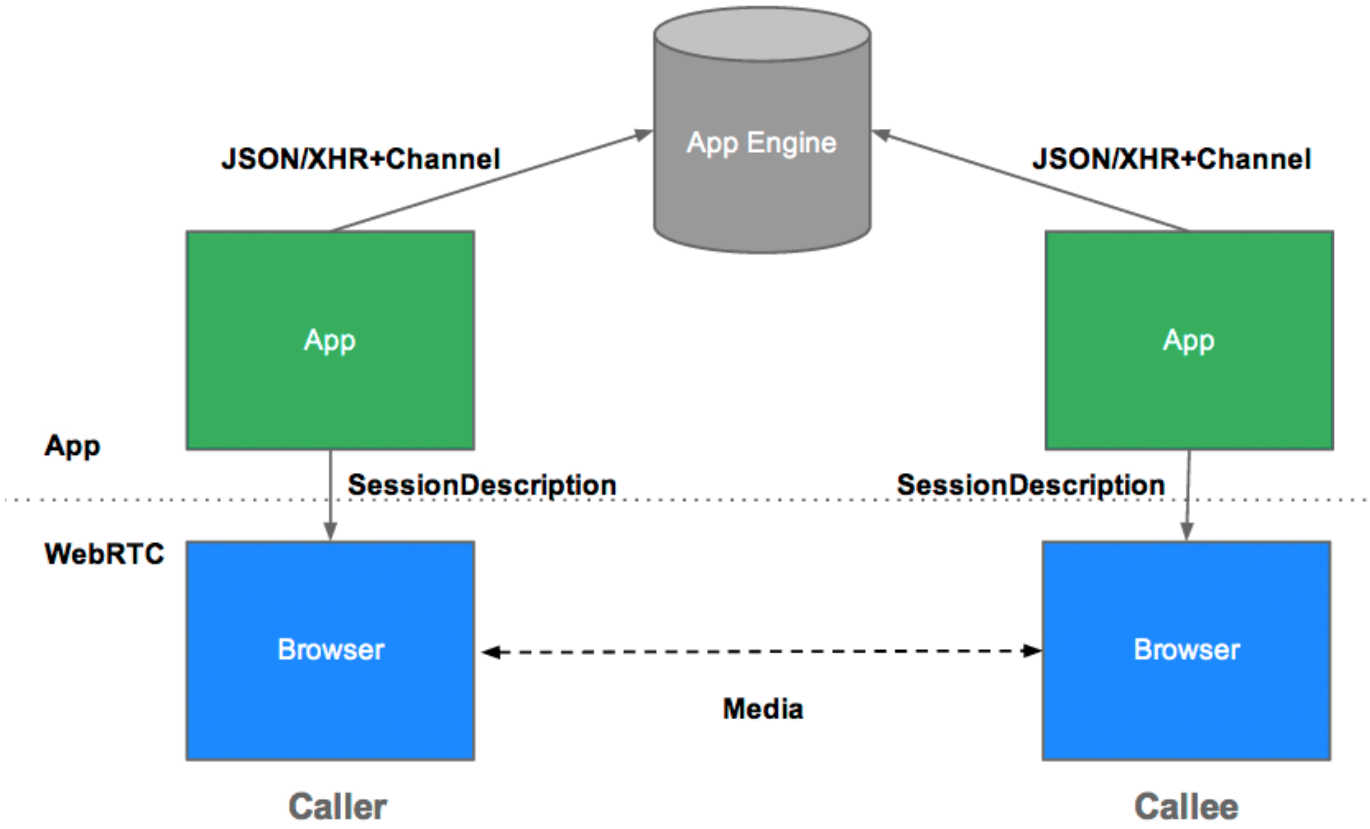
WEB RTC란?

WebRTC(Web Real-Time Communications)란, 웹 어플리케이션(최근에는 android 및 ios도 지원) 및 사이트들이 별도의 소프트웨어 없이 음성, 영상 미디어 혹은 텍스트, 파일 같은

데이터를 브라우져끼리 주고 받을 수 있게 만든 기술이다. WebRTC로 구성된 프로그램들은 별도의 플러그인이나 소프트웨어 없이 p2p 화상회의 및 데이터 공유를 한다.- 출처 - 바로가기

한마디로 요약하자면, 웹 브라우저 상에서 **어떤 플러그인 없이 음성채팅은 물론이며 화상채팅, 데이터 교환 까지도 가능하게하는 기술**이라고 보면된다.

WEB RTC 통신 원리



Web RTC 기술은 P2P 통신에 최적화 되어있다.
Web RTC에 사용되는 기술은 여러가지가 있지만 크게 3가지의 클래스에 의해서 실시간 데이터 교환이 일어난다.

- MediaStream - 카메라/마이크 등 데이터 스트림 접근
- RTCPeerConnection - 암호화 및 대역폭 관리 및 오디오 또는 비디오 연결
- RTCDataChannel - 일반적인 데이터 P2P통신

이 3가지의 객체를 통해서 데이터 교환이 이뤄지며 RTCPeerConnection들이 적절하게 데이터를 교환할 수 있게 처리하는 과정을 **시그널링(Signaling)** 이라고 한다.

위그림은 시그널링 하는 과정을 나타낸 것인데,**PeerConnection**은 두 명의 유저가 스트림을 주고 받는 것이므로 연결을 요청한 콜러(caller)와 연결을 받는 콜리(callee)가 존재한다. 콜러와

콜리가 통신을 하기 위해서는 **중간 역할을 하는 서버**가 필요하고 서버를 통해서 **SessionDescription**을 서로 주고 받아야 한다.

간단한 용어 정리

Web RTC API를 들여다보면 참 익숙치 않은 메소드와 용어들이 자주 등장한다.
튜토리얼글을 싸지르기(?) 앞서, 혼자 분석하면서 어렵거나 이해하기 힘들었던 용어들을 간략하게 정리해보았다.

Stun Server , Turn Server

Web RTC는 P2P에 최적화 되어있다. 즉 Peer들간의 공인 네트워크 주소(ip)를 알아 데이터 교환을 해야하는데, 실제 개개인의 컴퓨터는 방화벽등 여러가지 보호장치들이 존재하고 있다. 그래서 Peer들간의 연결이 쉽지 않은데, 이렇게 서로간의 연결을 위한 정보를 공유하여 P2P 통신을 가능하게 해주는 것이 Stun/Turn Server이다.

(https://alnova2.tistory.com/1110 에 더 자세한 내용이 담겨있다.)

SDP (Session Description Protocol)

세션 기술 프로토콜(Session Description Protocol, SDP)은 스트리밍 미디어의 초기화 인수를 기술하기 위한 포맷이다. 이 규격은 IETF의 RFC 4566로 규정되어 있다.
실제로 WEB RTC는 SDP format 에 맞춰져 영상,음성 데이터를 교환하고 있다.

Ice (Interactive Connectivity Establishment)

NAT환경에서 자신의 Public IP를 파악하고 상대방에게 데이터를 전송하기 위한 Peer간의 응답 프로토콜로 일반적으로 STUN/TURN을 이용해서 구축을 한다.
간단하게 설명하면, 한쪽이 Offer를 보내면 다른 한쪽이 Answer함으로써 피어간 연결이 설정된다
출처: https://wecomm.tistory.com/3 [미래를 여는 세상]

Web RTC를 이용하여 화상통화 구현하기

이제 화상통화와 관련된 간단한 튜토리얼을 작성할텐데, 소스코드양이 많기 때문에 주요 부분만 설명하고 넘어 갈터이니 **궁금한점 있다면 댓글 쓰윽 달아주길 바랍니다**

- rtc.html

```
<html>
<head>
    <meta charset="utf-8" />
    <title>WebRtc tutorial</title>
</head>

<body>
    <div>
        <video id="localVideo" autoplay width="480px"></video>
        <video id="remoteVideo" width="480px" autoplay></video>
    </div>

    <script src="/socket.io/socket.io.js"></script>
    <script src="./rtc.js"></script>
</body>

</html>
```

화상통화를 구현하기 위해서는 내 화면과 상대방의 화면을 나타낼 뷰를 제작하여 준다.
또한 시그널링과정에 필요한 socket.io cdn또한 설정하여 준다.

- rtc.js : **내 영상 정보 가져오기**

```
let localVideo = document.getElementById("localVideo");
let remoteVideo = document.getElementById("remoteVideo");
let localStream;

navigator.mediaDevices
  .getUserMedia({
    video: true,
    audio: false,
  })
  .then(gotStream)
  .catch((error) => console.error(error));

function gotStream(stream) {
  console.log("Adding local stream");
  localStream = stream;
  localVideo.srcObject = stream;
  sendMessage("got user media");
  if (isInitiator) {
    maybeStart();
  }
}
```

mediaDevice 객체의 getUserMedia Method를 통해서 사용자의 미디어 데이터를 스트림으로 받아올 수 있다. localStream과 localVideo에 출력할 영상을 본인 캠으로 지정한다.

- **sendMessage()**

```
function sendMessage(message){
  console.log('Client sending message: ',message);
```

```
  socket.emit('message',message);
}
```

시그널링 서버로 소켓정보를 전송하는 메소드이다. 후에 많이 언급됨. 시그널링 서버, 다른 Peer로의 데이터를 전송하는 method라고 보면 된다.

- **rtc.js : RTC Peer 연결하기**

```
function createPeerConnection() {
  try {
    pc = new RTCPeerConnection(null);
    pc.onicecandidate = handleIceCandidate;
    pc.onaddstream = handleRemoteStreamAdded;
    console.log("Created RTCPeerConnection");
  } catch (e) {
    alert("connot create RTCPeerConnection object");
    return;
  }
}

function handleIceCandidate(event) {
  console.log("iceCandidateEvent", event);
  if (event.candidate) {
    sendMessage({
      type: "candidate",
      label: event.candidate.sdpMLineIndex,
      id: event.candidate.sdpMid,
      candidate: event.candidate.candidate,
    });
  } else {
    console.log("end of candidates");
  }
}

function handleCreateOfferError(event) {
  console.log("createOffer() error: ", event);
}

function handleRemoteStreamAdded(event) {
  console.log("remote stream added");
  remoteStream = event.stream;
  remoteVideo.srcObject = remoteStream;
}
```

createPeerConnection을 통해 RTCPeerConnection에 대한 객체를 형성해주고 있다.
iceCandidate는 데이터 교환을 할 대상의 EndPoint 정보라고 보면 된다.
따라서 iceCandidate할 대상이 생긴다면 handleIceCandidate Method를 실행하게 된다.
이 부분은 signaling 서버로 넘겨줘 상대방 Peer가 내 Stream을 연결할 수 있도록 한다.
연결된 Peer는 handleRemoteStreamAdded Method를 통해서 remoteVideo 뷰에 띄우도록 한다.

```
function maybeStart() {
  console.log(">>MaybeStart() : ", isStarted, localStream, isChannelReady);
  if (!isStarted && typeof localStream !== "undefined" && isChannelReady) {
    console.log(">>>>> creating peer connection");
    createPeerConnection();
    pc.addStream(localStream);
    isStarted = true;
    console.log("isInitiator : ", isInitiator);
    if (isInitiator) {
      doCall();
    }
  }else{
    console.error('maybeStart not Started!');
  }
}
```

maybeStart method는 자신의 RTCPeerConnection을 초기화하고 상대방의 RTCPeerConnection과 연결하는 함수이다.

실제로 연결이 됐다면 doCall함수를 실행시켜 데이터를 주고 받는다.

```
function doCall() {
  console.log("Sending offer to peer");
  pc.createOffer(setLocalAndSendMessage, handleCreateOfferError);
}

function doAnswer() {
  console.log("Sending answer to peer");
  pc.createAnswer().then(
    setLocalAndSendMessage,
    onCreateSessionDescriptionError
  );
}

function setLocalAndSendMessage(sessionDescription) {
  pc.setLocalDescription(sessionDescription);
  sendMessage(sessionDescription);
}
```

doCall과 doAnswer를 통해서 Description을 교환하고 이 과정을 통해서 내 화상 정보가 상대방에게, 상대방의 화상정보가 내 뷰에 출력할 수 있게 되는 것이다.

```
let pcConfig = {
    'iceServers': [{
        'urls': 'stun:stun.l.google.com:19302'
      }]
}

socket.on('message', (message)=>{
  console.log('Client received message :',message);
  if(message === 'got user media'){
    maybeStart();
```

```
    }else if(message.type === 'offer'){
      if(!isInitiator && !isStarted){
        maybeStart();
      }
      pc.setRemoteDescription(new RTCSessionDescription(message));
      doAnswer();
    }else if(message.type ==='answer' && isStarted){
      pc.setRemoteDescription(new RTCSessionDescription(message));
    }else if(message.type ==='candidate' &&isStarted){
      const candidate = new RTCIceCandidate({
        sdpMLineIndex : message.label,
        candidate:message.candidate
      });

      pc.addIceCandidate(candidate);
    }
  })
```

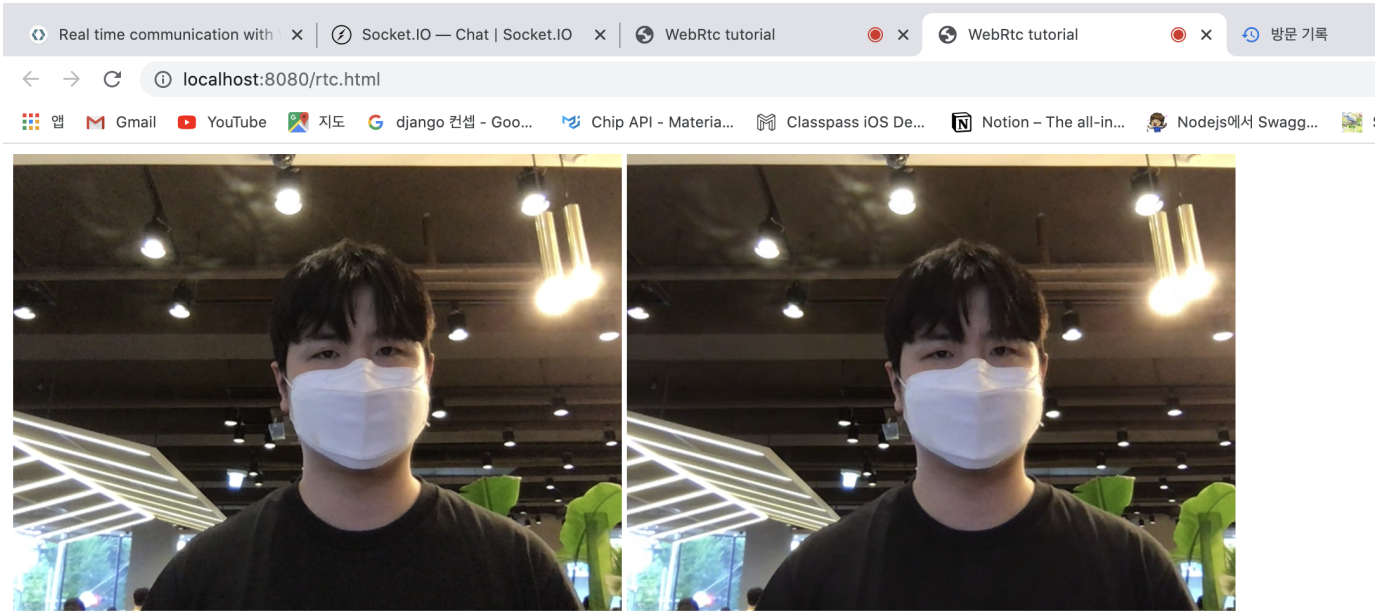
위는 소켓통신에 대한 부분을 정의해서 데이터 교환을 올바르게 할 수 있게 해준다.

```
const http = require('http');
const os = require('os');
const socketIO = require('socket.io');
const nodeStatic = require('node-static');

let fileServer = new(nodeStatic.Server)();
let app = http.createServer((req,res)=>{
    fileServer.serve(req,res);
}).listen(8080);

let io = socketIO.listen(app);
io.sockets.on('connection',socket=>{
    function log() {
        let array = ['Message from server:'];
        array.push.apply(array,arguments);
        socket.emit('log',array);
    }

    socket.on('message',message=>{
        log('Client said : ' ,message);
        socket.broadcast.emit('message',message);
    });

    socket.on('create or join',room=>{
        let clientsInRoom = io.sockets.adapter.rooms[room];
        let numClients = clientsInRoom ? Object.keys(clientsInRoom.sockets).length : 0;
        log('Room ' + room + ' now has ' + numClients + ' client(s)');

        if(numClients === 0){
            console.log('create room!');
            socket.join(room);
            log('Client ID ' + socket.id + ' created room ' + room);
            socket.emit('created',room,socket.id);
        }
        else if(numClients===1){
            console.log('join room!');
```

```
            log('Client Id' + socket.id + 'joined room' + room);
            io.sockets.in(room).emit('join',room);
            socket.join(room);
            socket.emit('joined',room,socket.id);
            io.sockets.in(room).emit('ready');
        }else{
            socket.emit('full',room);
        }
    });


});
```

위는 signaling 서버에 대한 구현으로 room이 없다면 생성하고, room이 이미 존재한다면 room에 참가하여 내 뷰를 상대방에게 중개해주는 그런 역할을 한다.

그런다음 서버를 실행시키면,,



실시간으로 중개되는 서버를 볼수가 있다. (난 뿌듯했다.)

코드 전체 보기

코드의양이 생각보다 많기 때문에 (합쳐서 300~400줄?) , 포스팅 된 글은 참고하면서 소스를 분석하는 방법을 추천한다.
코드는 https://github.com/ehdrms2034/WebRtcTutorial 에서 본문의 소스를 볼 수 있다.

600g (Kim Dong Geun)

수동적인 과신과 행운이 아닌, 능동적인 노력과 치열함

다음 포스트

[WebRtc] RTCPeerConnection.addStream is not defined 관련 문제 해결 방법

이전 포스트

Spring boot + React로 간단한 채팅방 제작하기

14개의 댓글

댓글을 작성하세요

댓글 작성

velopert

2020년 4월 24일

유익한 포스트 감사합니다!!

1개의 답글

calmglow

2020년 4월 29일

좋은 글 감사합니다. - RemoteMonster

⊞ 1개의 답글

sungyong
2020년 5월 9일

잘 읽었습니다. 저도 최근 webRTC 프로젝트 하면서, 많은 시행착오를 했는데도 여전히 어렵더군요...

⊞ 1개의 답글

imbla2
2020년 5월 20일

WEB RTC와 turn server면 무엇이든 가능할것만 같지만... 막상 프로젝트가 시작되면 시행착오가 많을 것 같네요... HLS와 WEB RTC 비교중인데 덕분에 WEB RTC에 대해 잘 배웠습니다.
감사합니다.

⊞ 1개의 답글

drilldog
2020년 5월 29일

컨트롤러는 어떻게 작성해야할까요?

⊞ 1개의 답글

tlatldms
2020년 6월 3일

헐 뭐야 나 졸작이 webRTC로 과외플랫폼인데;;;;;; 개깜놀 당신이 왜 여기서나와?

⊞ 1개의 답글

monomonokor
2020년 6월 10일

안녕 하세요 ~ 포스트 감사 합니다.
unity 개발자인데 백엔드는 왕초보 입니다.

혹시 실행 부분을 모듈로 뺄려면 어떻게 해야 하는지 몰라서 질문 드려요..^^;;
아래 처럼 수정 해 봤는데 안되네요...ㅎㅎ;;
서버에서 에러는 안나는데 브라우저 접속 하면 404 입니다.

실행 부분은 요기 이고

```
var chattingVoiceServer = require('./chattingVoice/server').start(3100);
```

포스팅 수정은 요기 입니다.

```
let app = http.createServer((req,res)=>{
fileServer.serve(req,res);
});
let io = socketIO(app);

.....

function start(port) {
app.listen(port, function() {
console.log( new Date() + ' / Chatting Voice Server Running : ' , port);
});
io.listen(server);
}
module.exports = {
start : start,
}
```

⊞ **답글 달기**