视频会议接口文档

1. **WebRTC**
2. 引入js脚本

<script src="js/webrtc/adapter-latest.js"></script>

<script src="js/webrtc/userMedia.js"></script>

<script src="js/webrtc/displayMedia.js"></script>

<script src="js/webrtc/peerConnection.js"></script>

1. js脚本说明文档
2. adapter-latest.js 这是一个解决WebRTC接口对浏览器的兼容
3. userMedia.js 这是一个获取摄像头的接口

使用:

* 参数 options为输入输出设备的各种参数(必须为js的对象数据)

{

video: {

width: {

min: 200,

max: 800

},

height: 100,

frameRate: {

min: 1,

max: 50

},

facingMode: 'environment',

resizeMode: 'none',

deviceId: undefined,

…

},

audio: {

noiseSuppression: true,

echoCancellation: true,

…

},

}

【视频参数调整】

width 视频宽度

height 视频高度

aspectRatio 比例（一般可不设置，只设置宽高即可，可用来获取宽高比）

frameRate 帧率（可通过帧率设置码流，帧率越高，码流越大，视频越平滑）

facingMode 摄像头（PC会忽略，手机端可区分）

user 前置摄像头

environment 后置摄像头

left 前置左侧摄像头

right 前置右侧摄像头

resizeMode 采集画面是否裁剪

none 不裁剪

【音频参数调整】

volume 音量调整（范围 0-1.0， 0为静音，1为最大声）

sampleRate 采样率 （例 8000）

sampleSize 采样大小 （例 16位 2字节）

echoCancellation 回音消除 （true/false）

autoGainControl 自动增益 （在原有录音的基础上是否增加音量， true/false）

noiseSuppression 是否开启降噪功能 （true/false）

【其他参数】

latency 延迟大小 （ 延迟小，网络不好的情况下，会卡顿花屏等，好处在于可实时通信，建议200ms; 延迟大，网络不好的情况下，画面相对更平滑流畅，但即时性较差）

channelCount 单/双声道

deviceID 多个摄像头或音频输入输出设备时，可进行设备切换（例如切换前后置摄像头）

groupID 同一组设备

* 在 UserMedia这个构造函数中使用原型链的方式挂载几个函数

UM.prototype. getUserMediaSuccess = function(stream){

// stream 该流可以包括例如视频轨道（由诸如照相机，视频记录设备，屏幕共享服务等的硬件或虚拟视频源产生），音频轨道（类似地，由物理或虚拟音频源，如麦克风，A / D转换器等），以及可能的其他轨道类型

// 这里写获取到摄像头后处理的事件

};

UM.prototype. getUserMediaError = function(e){

// 这里写获取音视频失败的事件

// e 错误信息

};

UM.prototype. notSuportMdeia = function(){

// 这里写浏览器不支持获取音视频的事件

};

* 实例构造函数

userMedia = new UM(options);

获取摄像头: userMedia. getUserMedia();

获取设备信息: let devicesInfo = userMedia. gotDevices();

1. displayMedia.js 这是一个获取共享桌面的接口

使用:

* 在原型链上挂载方法

DM.prototype. getDisplayMediaSuccess = function(stream){

// 这里写获取到共享桌面的资源

// stream资源

};

DM.prototype. getDisplayMediaError= function(e){

// 这里写获取到共享桌面的资源

// e错误信息

};

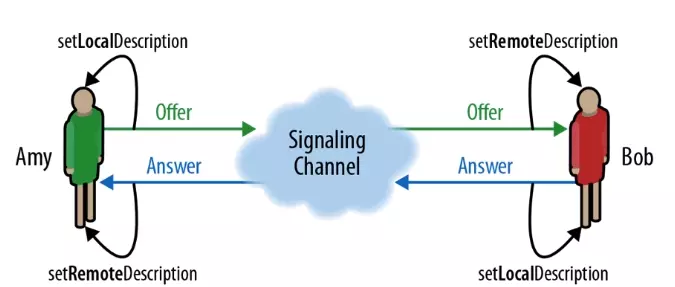
* 实例构造函数

displayMedia = new DM();

获取共享桌面 displayMedia. getDisplayMedia();

1. peerConnect.js 媒体能力协商过程

A创建offer形成一个SDP，通过云端信令channel传给B，在传之前要调一个setLocalDescription方法，触发一个非常重要的作用就是收集候选者；B端收到offer时，调用setRemoteDescription，将offer形成的SDP数据，放到自己远端的描述信息的槽里，然后回复一个Answer，Answer回复的是B本机的所有的媒体信息，网络信息，也会调用setLocalDescription方法，触发收集候选者；A收到Answer后存到自己的槽中；每个人都拿到两个SDP信息，在内部进行协商，取出交集，协商过程建立完成，进行下一步工作。



使用:

* 参数说明 myUserId 这个用户的唯一性标志(可以数字字母混合, 可以使用已经写好的获取随机字符串方法)。注意：得到随机字符串请前面拼接 ‘webrtc\_’, 如: 用户最终id为 webrtc\_abc123

roomId 房间号(大于0的数字, 可以使用已经写好的随机数字), 不能大于65535.

* 属性说明 pc (对象, 下标为连接的userId) 所有连接的peerConnection信息

dc (对象, 下标为连接的userId) 所有连接的dataChannel信息

* 在原型链上挂载方法

PC.prototype. beforePC = function(userId){

// userId为连接的用户标志

// 这里写实例化RTCPeerConnection 之前该做的事

// 一般为创建一个该连接用户的video标签, 用于放后面该用户的音视频资源

};

PC.prototype. iceCandidateEvent = function(candidate){

// 这里写监听到远程连接的事件, 为发送websocket给对方

// 可以使用下面给出的, 也可以编写自己的ws逻辑

// ws.send({

// action: 'webrtc',

// event: 'candidate',

// mine:{

// id: this.myUserId

// },

// webrtc:{

// id: userId

// },

// data: {

// sdpMLineIndex:e.candidate.sdpMLineIndex,

// sdpMid:e.candidate.sdpMid,

// candidate: e.candidate.candidate

// }

// });

};

PC.prototype.onTrackEvent = function(userId, stream){

// userId为连接的用户标志, stream为连接的用户的音视频资源

// 把资源放到对应的标签上

// if(document.getElementById(userId)){

// document.getElementById(userId).srcObject = null;

// document.getElementById(userId).srcObject = stream;

// }

};

PC.prototype. createOfferError = function(e){

// e错误信息

// 这里是创建offer失败事件

// offer不能创建, 代表双方不能通信

};

PC.prototype. setLocalOfferEvent = function(userId, offerDesc){

// offerDesc 创建offer的描述

// 这里是自己创建offer后, 把自己的offer发送给对方, ws事件

//send offer sdp;

// ws.send({

// action: 'webrtc',

// event: 'offer',

// mine:{

// id: this.myUserId

// },

// webrtc:{

// id: userId

// },

// data: offerDesc

// });

};

PC.prototype. setLocalOfferError = function(e){

// e 错误信息

// 这里是设置本地offer失败的事件

};

PC.prototype.createAnswerError = function(e){

// e 错误信息

// 创建answer失败事件

};

PC. setLocalAnswerEvent = function(userId, answerDesc){

// answerDesc answer回应的描述信息

// ws.send({

// action: 'webrtc',

// event: 'answer',

// mine:{

// id: this.myUserId

// },

// webrtc:{

// id: userId

// },

// data: answerDesc

// });

};

PC.setLocalAnswerError = function(e){

// e错误信息

// 设置本地answer描述符错误事件

};

PC.prototype. setRemoteDescriptionSuccess() = function(userId){

// 设置远程offer/answer描述成功

console.log(userId);

};

PC.prototype. setRemoteDescriptionError = function(e){

// e 错误信息

// 设置远程offer/answer描述失败

console.log(e);

};

PC.prototype. onDataChannelOpenEvent = function(userId){

// 监听dataChannel的open状态触发的事件

};

PC.prototype. onDataChannelCloseEvent = function(userId){

// 监听 dataChannel的 close状态触发的事件

};

PC.prototype. onDataChannelErrorEvent = function(userId){

// 监听 dataChannel的error事件

};

PC.prototype. onDataChannelMessageEvent = function(message){

// dataChannel接收信息事件(JOSN.parse解析后的数据);’

};

PC.prototype. onDataChannelFileEvent = function(arrayBuffer){

// dataChannel接收文件arrayBuffer

};

PC.prototype. sendFileErrorEvent = function(e){

// e 错误信息

// 发送文件错误事件

};

PC.prototype. sendFileStartEvent = function(file){

// file 发送的文件信息

// 发送文件开始事件, 可以在这里创建进度条标签,用于文件流传送的进度

};

PC.prototype. sendFileLoadEvent = function(result, offset){

// result 文件流的每一块

// offset 已经发送文件的大小

// 发送文件流事件, 可以在这里处理自己要发送的文件流, 改变进度条

};

PC.prototype. sendFileEndEvent = function(file, offset){

// file 发送的文件信息

// offset 已经发送的文件大小(这里就是文件的实际大小 file.size)

// 发送文件结束事件, 可以在这里展示发送或接收到的文件

};

PC.prototype. cleanOneUserEvent = function(userId){

// 用户关闭事件

};

PC.prototype.getLocalInfo = function(report, userId){

// report 自己发送给用户的信息

};

PC.prototype. remoteInfo = function(report, userId){

// report接收每个

};

peerConnect = new PC(myUserId, roomId);

**交换信令四部曲**—收到相应的websocket事件时做的事

* 交换信令四部曲-1

收到ws otherJoin事件

b进入房间, a该做的事 发送offer

peerConnect.otherJoin(userId, stream);

* 交换信令四部曲-2

收到ws offer事件

b 收到 a 的offer信令, b 回应answer

peerConnect.offer(userId, stream, offerDesc);

* 交换信令四部曲-3

收到ws answer事件

peerConnect.answer(userId, answerDesc);

a 收到 b 的 answer

* 交换信令四部曲-4

收到ws candidate事件

a 和 b 交换candidate信息

peerConnect.candidate(userId, sdpMLineIndex, candidate);

* 创建 RTCPeerConnection, 每有一个用户进来必须创建一个pc与之对应, 无返回值

peerConnect.**createPeerConnectiont(userId, stream)**;

* 停止资源分发, 在停止音视频或共享桌面时, 用此接口把资源释放掉, 无返回值

peerConnect.**stopTrack(stream)**;

* 获取跟我连接远程音视频资源, 在进行video样式修改时此方法会使用到, (此接口快废弃, 不建议使用, 推荐使用getReceovers), 返回媒体资源 stream, 可以直接用在 video.srcObject = stream;

peerConnect. **getRemoteStreams()**;

* 获取远程媒体资源

peerConnect.**getReceivers(userId, callback)**;

userId 那个用户的媒体资源

callback(stream) 回调函数, 参数 stream媒体资源(video.srcObject = stream)

* 获取本地媒体资源

peerConnect.**getSenders(userId, callback);**

userId 我发送给哪个用户的我的媒体资源

callback(stream) 回调函数, 参数 stream媒体资源

* 替换本地媒体资源(把我的音视频换成桌面共享时使用)

peerConnect.**replaceTrack(stream)**;

stream要换成的媒体资源

* 获取媒体资源的音频状态(true开启/false关闭)

peerConnect.**isMuteAudio(stream)**;

stream 当前的媒体资源

返回值 true/false

* 获取媒体资源的视频状态(true开启/false关闭)  
  peerConnect.**isMuteVideo(stream)**;

stream当前的媒体资源

返回值 true/false

* 是否静音(对房间内所有人有效)

peerConnect.**muteAudio(stream, mute)**;

stream 当前的媒体资源

mute true不静音/false静音(默认false)

返回值 是否操作成功(true操作成功/false操作失败)

* 是否可见媒体资源(对房间内所有人有效)

peerConnect.**muteVideo(stream, mute)**;

stream 当前媒体资源

mute true可见/false不可见

返回值 是否操作成功(true操作成功/false操作失败)

* dataChannel发送数据(可以把dataChannel理解为websocket)

peerConnect.**sendChannelData(data, receiveId)**;

data发送的数据, 如果不是发送的文件, data要进行JSON. stringify(data)进行序列化

received 接收人id, 默认为空是发送给所有人

* dataChannel发送文件

peerConnect.**sendFile(file, received, chunkSize)**;

file 文件信息, 通过<input type=’file’ />拿到的

received 接收人id, 默认为空发送给所有人

chunkSize 每次读取/发送文件块的大小, 默认10240b

* 用户退出

peerConnect.**cleanOneUser(userId)**;

userId 用户id

把用户在RTCPeerConnection和dataChannel中的信息删除掉

* 获取视频媒体资源信息

peerConnect.**getStreamInfo()**;

1. **webSocket**
2. 引入js脚本

<script src="js/webrtc/webSocketEvent.js"></script>

使用

在原型链上挂载方法—websocket

这些挂载的方法名是主持人/管理者/用户发送的ws事件的方法名

WS.prototype.open = function(e){

// e ws.onopen信息

// 此时应该告诉后台你的个人信息

this.send({

action: 'open',

webrtc: {

id: myUserId

},

info: {

id: myUserId,

name: userName,

}

});

};

WS.prototype.actionOpen = function(data){

// ws连接是否成功

// e.state == 1代表成功, 0代表失败

};

WS.prototype.joined = function(data){

// 我加入房间给我返回的信息

};

WS.protype.otherJoin = function(data){

// 有其他人加人房间, 系统给我的事件

// 处理房间内用户

// 创建pc, 参见peerConnect.js里的 otherJoin方法

};

WS.protype.offer = function(data){

// 接收offer事件, 见peerConnect,js里的offer方法

}

WS.protype.answer = function(data){

// 接收answer事件, 参见peerConnect.js里的answer方法

};

WS.prototype.candidate = function(data){

// 接收 candidate事件. 参见peerConnect.js里的 candidate方法

};

WS.prototype.muteAudio = function(data){

// 全体静音事件

// 接收到此事件, 只需把自己调为静音即可,

// 参见 peerConnect.js里的muteAudio

};

WS.prototype.userAudio = function(data){

// 单个用户静音事件

// 同 muteAudio

};

WS.prototype.setAdmin = function(data){

// 主持人设置一个管理者协同主持人进行管理

};

WS.prototype.cancelAdmin = function(data){

// 取消管理员

};

WS.prototype.roomLayout = function(data){

// 主持人进行布局同步到与会者

};

WS.prototype.maxVideo = function(data){

// 主持人显示谁为最大视频

};

WS.prototype.editName = function(data){

// 主持人修改与会者的名字

};

WS.prototype.removeUser = function(data){

// 主持人移除一个参会者

};

WS.prototype.closeRoom = function(data){

// 主持人关闭会议室

ws.close(); // 主动断开websocket连接

};

WS.prototype.out = function(data){

// 用户退出事件

};

WS.prototyoe.wsDefaultEvent = function(data){

// ws接收到除上面的事件外, 可以自定义事件, 但是得跟后台 ws服务提供者商量好

};

实例化 构造函数

ws = new WS(obj);

参数 obj 对象url为websocket的地址, 存在就替换默认的

属性 url websocket的地址

方法

* 客户端主动结束ws

ws.**close()**;

* 发送ws消息, data在发送前自动进行JSON.stringify序列化处理, 使用者无需再进行序列化操作

ws.**send(data = {})**

*发送的各种事件*

* 全体静音

ws.send({

action: 'webrtc',

event: 'muteAudio',

mine: {

id: myUserId

},

room: {

id: roomId

},

auth: isAllMute // true静音/false取消静音

});

* 结束会议

ws.send({

action: 'webrtc',

event: 'closeRoom',

mine: {

id: myUserId

},

room: {

id: roomId

},

});

* 设置会议室管理者

ws.send({

action: 'webrtc',

event: 'setAdmin',

mine: {

id: myUserId

},

room: {

id: roomId,

},

admin: {

id: adminId

}

});

* 取消会议室管理者

ws.send({

action: 'webrtc',

event: 'cancelAdmin',

mine: {

id: myUserId

},

room: {

id: roomId,

},

admin: {

id: adminId

}

});

* 显示最大视频

ws.send({

action: 'webrtc',

event: 'maxVideo',

mine: {

id: myUserId

},

room: {

id: roomId,

},

maxVideoId: maxVideoId

});

* 移除一个参会者

ws.send({

action: 'webrtc',

event: 'removeUser',

mine: {

id: myUserId

},

room: {

id: roomId,

},

webrtc: {

id: removeUserId

}

});

* 修改参会者姓名

ws.send({

action: 'webrtc',

event: 'editName',

mine: {

id: myUserId

},

room: {

id: roomId

},

webrtc: {

id: otherUserId,

name: val

},

});

* 用户静音

ws.send({

action: 'webrtc',

event: 'userAudio',

mine: {

id: myUserId

},

room:{

id: roomId

},

webrtc: {

id: otherUserId// 用户id

},

auth: isAudio // true静音/false取消静音

});

* 加入会议室

ws.send({

action: 'webrtc',

event: 'joinRoom',

mine: {

id: myUserId

},

room: {

id: roomId, // 会议室id

pwd: psw, // 会议室密码

},

});

* 发送offer

ws.send({

action: 'webrtc',

event: 'offer',

mine:{

id: myUserId

},

webrtc:{

id: otherUserId // 用户id

},

data: offerDesc // offer描述

});

* 发送candidate

ws.send({

action: 'webrtc',

event: 'candidate',

mine:{

id: myUserId

},

webrtc:{

id: otherUserId

},

data: {

sdpMLineIndex: candidate.sdpMLineIndex,

sdpMid: candidate.sdpMid,

candidate: candidate.candidate

}

});

* 发送answer

ws.send({

action: 'webrtc',

event: 'answer',

mine:{

id: myUserId

},

webrtc:{

id: otherUserId

},

data: answerDesc // answer描述

});