使用WebRTC搭建前端视频聊天室——入门篇 - 说学逗唱

- 推荐 **5** 推荐
- 收藏 **90** 收藏，**43.7k** 浏览

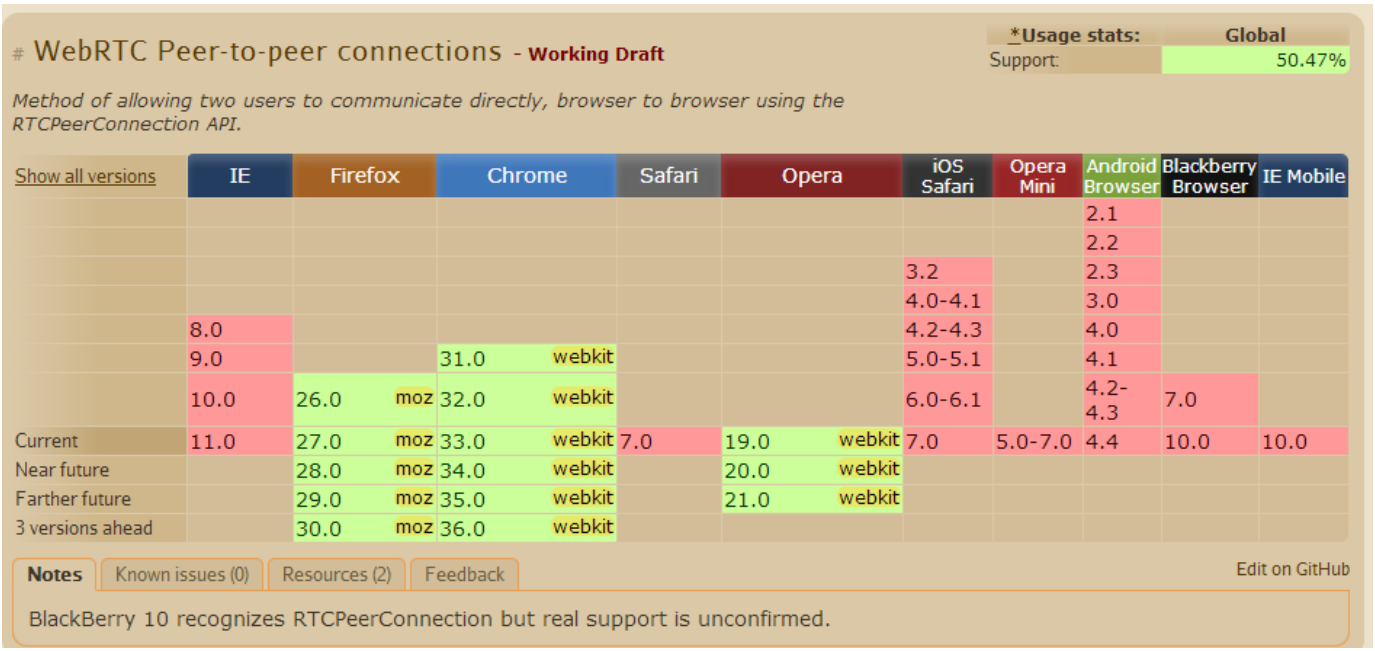
什么是WebRTC?

众所周知，浏览器本身不支持相互之间直接建立信道进行通信，都是通过服务器进行中转。比如现在有两个客户端，甲和乙，他们俩想要通信，首先需要甲和服务器、乙和服务器之间建立信道。甲给乙发送消息时，甲先将消息发送到服务器上，服务器对甲的消息进行中转，发送到乙处，反过来也是一样。这样甲与乙之间的一次消息要通过两段信道，通信的效率同时受制于这两段信道的带宽。同时这样的信道并不适合数据流的传输，如何建立浏览器之间的点对点传输，一直困扰着开发者。WebRTC应运而生

WebRTC是一个开源项目，旨在使得浏览器能为实时通信（RTC）提供简单的JavaScript接口。说的简单明了一点就是让浏览器提供JS的即时通信接口。这个接口所创立的信道并不是像WebSocket一样，打通一个浏览器与WebSocket服务器之间的通信，而是通过一系列的信令，建立一个浏览器与浏览器之间（peer-to-peer）的信道，这个信道可以发送任何数据，而不需要经过服务器。并且WebRTC通过实现MediaStream，通过浏览器调用设备的摄像头、话筒，使得浏览器之间可以传递音频和视频

WebRTC已经在我们的浏览器中

这么好的功能，各大浏览器厂商自然不会置之不理。现在WebRTC已经可以在较新版的Chrome、Opera和Firefox中使用了，著名的浏览器兼容性查询网站caniuse上给出了一份详尽的浏览器兼容情况



WebRTC Peer-to-peer connections - Working Draft

*Usage stats: Global
Support: 50.47%

Method of allowing two users to communicate directly, browser to browser using the RTCPeerConnection API.

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
								2.1		
								2.2		
						3.2		2.3		
						4.0-4.1		3.0		
	8.0					4.2-4.3		4.0		
	9.0		31.0 webkit			5.0-5.1		4.1		
	10.0	26.0 moz	32.0 webkit			6.0-6.1		4.2-4.3	7.0	
Current	11.0	27.0 moz	33.0 webkit	7.0	19.0 webkit	7.0	5.0-7.0	4.4	10.0	10.0
Near future		28.0 moz	34.0 webkit		20.0 webkit					
Farther future		29.0 moz	35.0 webkit		21.0 webkit					
3 versions ahead		30.0 moz	36.0 webkit							

Notes | Known issues (0) | Resources (2) | Feedback | Edit on GitHub

BlackBerry 10 recognizes RTCPeerConnection but real support is unconfirmed.

三个接口

WebRTC实现了三个API，分别是:

* MediaStream：通过MediaStream的API能够通过设备的摄像头及话筒获得视频、音频的同步流
* RTCPeerConnection：RTCPeerConnection是WebRTC用于构建点对点之间稳定、高效的流传输的组件
* RTCDataChannel：RTCDataChannel使得浏览器之间（点对点）建立一个高吞吐量、低延时的信道，用于传输任意数据

这里大致上介绍一下这三个API

MediaStream（getUserMedia）

MediaStream API为WebRTC提供了从设备的摄像头、话筒获取视频、音频流数据的功能

如何调用

同门可以通过调用navigator.getUserMedia()，这个方法接受三个参数：
1. 一个约束对象（constraints object），这个后面会单独讲
2. 一个调用成功的回调函数，如果调用成功，传递给它一个流对象
3. 一个调用失败的回调函数，如果调用失败，传递给它一个错误对象

浏览器兼容性

由于浏览器实现不同，他们经常会在实现标准版本之前，在方法前面加上前缀，所以一个兼容版本就像这样

```
var getUserMedia = (navigator.getUserMedia ||
                    navigator.webkitGetUserMedia ||
                    navigator.mozGetUserMedia ||
                    navigator.msGetUserMedia);
```

一个超级简单的例子

这里写一个超级简单的例子，用来展现getUserMedia的效果：

```
<!doctype html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>GetUserMedia实例</title>
</head>
<body>
    <video id="video" autoplay></video>
</body>
```

```
<script type="text/javascript">
    var getUserMedia = (navigator.getUserMedia ||
navigator.webkitGetUserMedia || navigator.mozGetUserMedia ||
navigator.msGetUserMedia);

    getUserMedia.call(navigator, {
        video: true,
        audio: true
    }, function(localMediaStream) {
        var video = document.getElementById('video');
        video.src = window.URL.createObjectURL(localMediaStream);
        video.onloadedmetadata = function(e) {
            console.log("Label: " + localMediaStream.label);
            console.log("AudioTracks" ,
localMediaStream.getAudioTracks());
            console.log("VideoTracks" ,
localMediaStream.getVideoTracks());
        };
    }, function(e) {
        console.log('Reeeejected!', e);
    });
</script>


</html>
```

将这段内容保存在一个HTML文件中，放在服务器上。用较新版本的Opera、Firefox、Chrome打开，在浏览器弹出询问是否允许访问摄像头和话筒，选同意，浏览器上就会出现摄像头所拍摄到的画面了

注意，HTML文件要放在服务器上，否则会得到一个NavigatorUserMediaError的错误，显示PermissionDeniedError，最简单方法就是cd到HTML文件所在目录下，然后`python -m SimpleHTTPServer`（装了python的话），然后在浏览器中输入`http://localhost:8000/{文件名称}.html`

这里使用`getUserMedia`获得流之后，需要将其输出，一般是绑定到`video`标签上输出，需要使用`window.URL.createObjectURL(localMediaStream)`来创造能在`video`中使用`src`属性播放的Blob URL，注意在`video`上加入`autoplay`属性，否则只能捕获到一张图片

流创建完毕后可以通过`label`属性来获得其唯一的标识，还可以通过`getAudioTracks()`和`getVideoTracks()`方法来获得流的追踪对象数组（如果没有开启某种流，它的追踪对象数组将是一个空数组）

约束对象（Constraints）

约束对象可以被设置在`getUserMedia()`和RTCPeerConnection的`addStream`方法中，这个约束对象

是WebRTC用来指定接受什么样的流的，其中可以定义如下属性：
* video: 是否接受视频流
* audio：是否接受音频流
* MinWidth: 视频流的最小宽度
* MaxWidth：视频流的最大宽度
* MinHeight：视频流的最小高度
* MaxHiehgt：视频流的最大高度
* MinAspectRatio：视频流的最小宽高比
* MaxAspectRatio：视频流的最大宽高比
* MinFramerate：视频流的最小帧速率
* MaxFramerate：视频流的最大帧速率

RTCPeerConnection

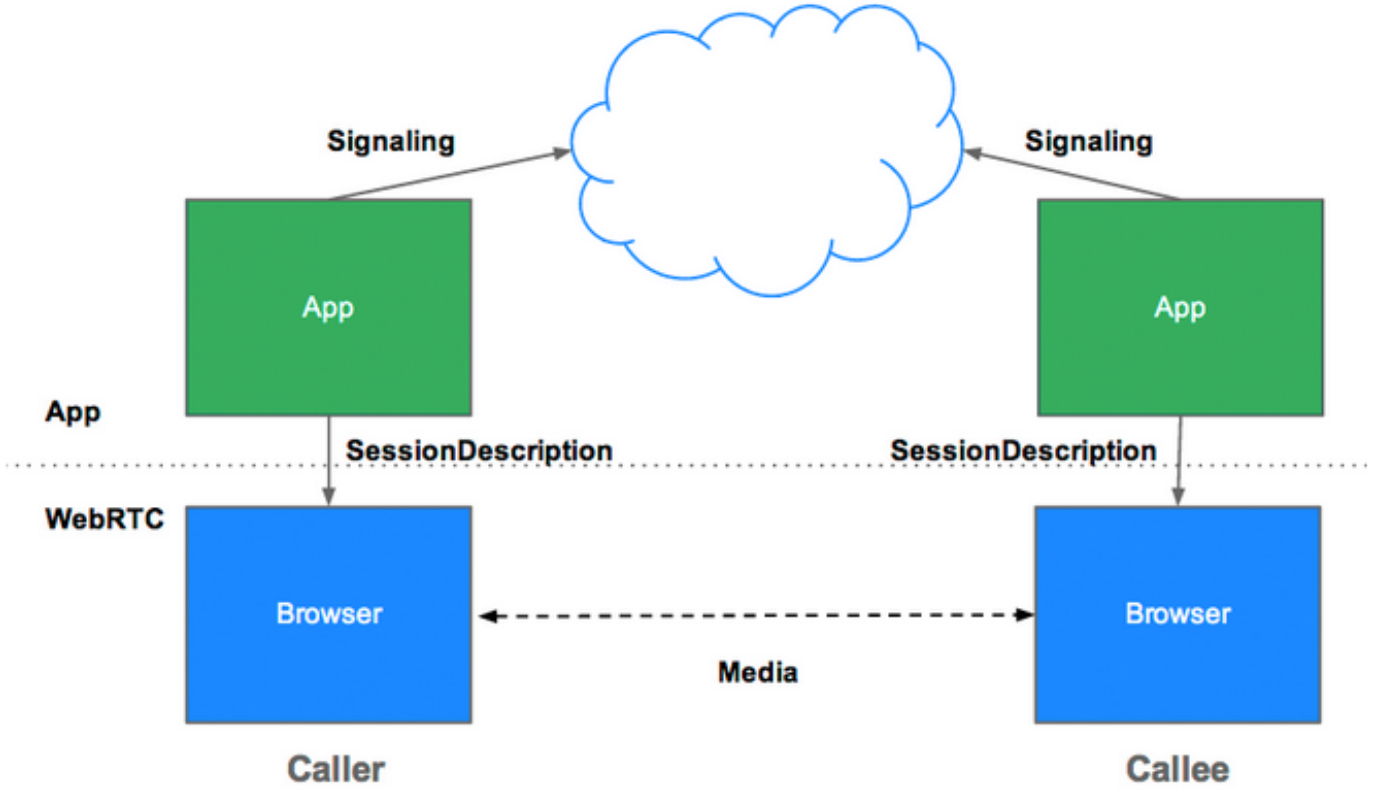
WebRTC使用RTCPeerConnection来在浏览器之间传递流数据，这个流数据通道是点对点的，不需要经过服务器进行中转。但是这并不意味着我们能抛弃服务器，我们仍然需要它来为我们传递信令（signaling）来建立这个信道。WebRTC没有定义用于建立信道的信令的协议：信令并不是RTCPeerConnection API的一部分

信令

既然没有定义具体的信令的协议，我们就可以选择任意方式（AJAX、WebSocket），采用任意的协议（SIP、XMPP）来传递信令，建立信道，比如我写的demo，就是用的node的ws模块，在WebSocket上传递信令

需要信令来交换的信息有三种：
* session的信息：用来初始化通信还有报错
* 网络配置：比如IP地址和端口啥的
* 媒体适配：发送方和接收方的浏览器能够接受什么样的编码器和分辨率

这些信息的交换应该在点对点的流传输之前就全部完成，一个大致的架构图如下：



JSEP architecture

通过服务器建立信道

这里再次重申，就算WebRTC提供浏览器之间的点对点信道进行数据传输，但是建立这个信道，必须有服务器的参与。WebRTC需要服务器对其进行四方面的功能支持：

1. 用户发现以及通信
2. 信令传输
3. NAT/防火墙穿越
4. 如果点对点通信建立失败，可以作为中转服务器

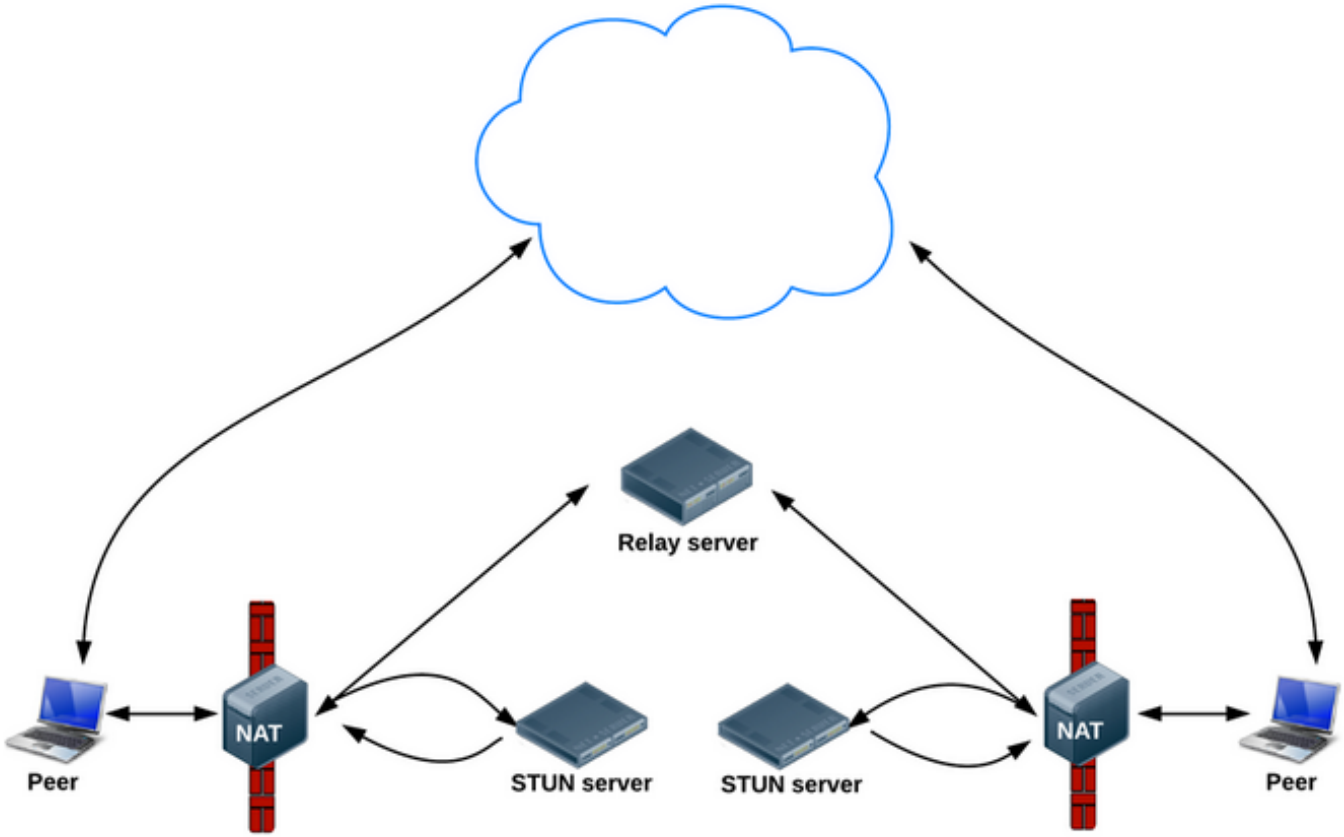
NAT/防火墙穿越技术

建立点对点信道的一个常见问题，就是NAT穿越技术。在处于使用了NAT设备的私有TCP/IP网络中的主机之间需要建立连接时需要使用NAT穿越技术。以往在VoIP领域经常会遇到这个问题。目前已经有很多NAT穿越技术，但没有一项是完美的，因为NAT的行为是非标准化的。这些技术中大多使用了一个公共服务器，这个服务使用了一个从全球任何地方都能访问得到的IP地址。在RTCPeeConnection中，使用ICE框架来保证RTCPeerConnection能实现NAT穿越

ICE，全名叫交互式连接建立（Interactive Connectivity Establishment）,一种综合性的NAT穿越技术，它是一种框架，可以整合各种NAT穿越技术如STUN、TURN（Traversal Using Relay NAT 中继NAT实现的穿透）。ICE会先使用STUN，尝试建立一个基于UDP的连接，如果失败了，就会去TCP（先尝试HTTP，然后尝试HTTPS），如果依旧失败ICE就会使用一个中继的TURN服务器。

我们可以使用Google的STUN服务器：`stun:stun.l.google.com:19302`，于是乎，一个整合了ICE

框架的架构应该长这个样子



浏览器兼容

还是前缀不同的问题，采用和上面类似的方法：

```
var PeerConnection = (window.PeerConnection ||
                      window.webkitPeerConnection00 ||
                      window.webkitRTCPeerConnection ||
                      window.mozRTCPeerConnection);
```

创建和使用

```
//使用Google的stun服务器
var iceServer = {
    "iceServers": [{
        "url": "stun:stun.l.google.com:19302"
    }]
};
//兼容浏览器的getUserMedia写法
var getUserMedia = (navigator.getUserMedia ||
                    navigator.webkitGetUserMedia ||
                    navigator.mozGetUserMedia ||
                    navigator.msGetUserMedia);
//兼容浏览器的PeerConnection写法
```

```
var PeerConnection = (window.PeerConnection ||
                    window.webkitPeerConnection00 ||
                    window.webkitRTCPeerConnection ||
                    window.mozRTCPeerConnection);
//与后台服务器的WebSocket连接
var socket = __createWebSocketChannel();
//创建PeerConnection实例
var pc = new PeerConnection(iceServer);
//发送ICE候选到其他客户端
pc.onicecandidate = function(event){
    socket.send(JSON.stringify({
        "event": "__ice_candidate",
        "data": {
            "candidate": event.candidate
        }
    }));
};
//如果检测到媒体流连接到本地，将其绑定到一个video标签上输出
pc.onaddstream = function(event){
    someVideoElement.src = URL.createObjectURL(event.stream);
};
//获取本地的媒体流，并绑定到一个video标签上输出，并且发送这个媒体流给其他客户端
getUserMedia.call(navigator, {
    "audio": true,
    "video": true
}, function(stream){
    //发送offer和answer的函数，发送本地session描述
    var sendOfferFn = function(desc){
            pc.setLocalDescription(desc);
            socket.send(JSON.stringify({
                "event": "__offer",
                "data": {
                    "sdp": desc
                }
            }));
        },
        sendAnswerFn = function(desc){
            pc.setLocalDescription(desc);
            socket.send(JSON.stringify({
                "event": "__answer",
                "data": {
                    "sdp": desc
                }
            }));
```

```
        };
    //绑定本地媒体流到video标签用于输出
    myselfVideoElement.src = URL.createObjectURL(stream);
    //向PeerConnection中加入需要发送的流
    pc.addStream(stream);
    //如果是发送方则发送一个offer信令，否则发送一个answer信令
    if(isCaller){
        pc.createOffer(sendOfferFn);
    } else {
        pc.createAnswer(sendAnswerFn);
    }
}, function(error){
    //处理媒体流创建失败错误
});
//处理到来的信令
socket.onmessage = function(event){
    var json = JSON.parse(event.data);
    //如果是一个ICE的候选，则将其加入到PeerConnection中，否则设定对方的session描述为传递过来的描述
    if( json.event === "__ice_candidate" ){
        pc.addIceCandidate(new RTCIceCandidate(json.data.candidate));
    } else {
         pc.setRemoteDescription(new RTCSessionDescription(json.data.sdp));
    }
};
```

实例

由于涉及较为复杂灵活的信令传输，故这里不做简短的实例，可以直接移步到最后

RTCDataChannel

既然能建立点对点的信道来传递实时的视频、音频数据流，为什么不能用这个信道传一点其他数据呢？RTCDataChannel API就是用来干这个的，基于它我们可以在浏览器之间传输任意数据。DataChannel是建立在PeerConnection上的，不能单独使用

使用DataChannel

我们可以使用`channel = pc.createDataCHannel("someLabel");`来在PeerConnection的实例上创建Data Channel，并给与它一个标签

DataChannel使用方式几乎和WebSocket一样，有几个事件：

* onopen
* onclose
* onmessage
* onerror

同时它有几个状态，可以通过`readyState`获取：
* connecting: 浏览器之间正在试图建立channel
* open：建立成功，可以使用`send`方法发送数据了
* closing：浏览器正在关闭channel
* closed：channel已经被关闭了

两个暴露的方法:
* close(): 用于关闭channel
* send()：用于通过channel向对方发送数据

通过Data Channel发送文件大致思路

JavaScript已经提供了File API从`input[type='file']`的元素中提取文件，并通过FileReader来将文件的转换成DataURL，这也意味着我们可以将DataURL分成多个碎片来通过Channel来进行文件传输

一个综合的Demo

SkyRTC-demo，这是我写的一个Demo。建立一个视频聊天室，并能够广播文件，当然也支持单对单文件传输，写得还很粗糙，后期会继续完善

使用方式

1. 下载解压并cd到目录下
2. 运行`npm install`安装依赖的库（express, ws, node-uuid）
3. 运行`node server.js`，访问`localhost:3000`，允许摄像头访问
4. 打开另一台电脑，在浏览器（Chrome和Opera，还未兼容Firefox）打开`{server所在IP}:3000`，允许摄像头和话筒访问
5. 广播文件：在左下角选定一个文件，点击“发送文件”按钮
6. 广播信息：左下角input框输入信息，点击发送
7. 可能会出错，注意F12对话框，一般F5能解决

功能

视频音频聊天（连接了摄像头和话筒，至少要有摄像头），广播文件（可单独传播，提供API，广播就是基于单独传播实现的，可同时传播多个，小文件还好说，大文件坐等内存吃光），广播聊天信息