文 使用WebRTC搭建前端视频聊天室——信令篇 (/a/1190000000439103)

webrtc (/t/webrtc/blogs) node.js (/t/node.js/blogs) webim (/t/webim/blogs)

websocket (/t/websocket/blogs) javascript (/t/javascript/blogs)

天镶 (/u/lingyucoder) 2014年03月18日发布

博客原文地址 (http://skyinlayer.com/blog/2014/0…

建议看这篇之前先看一下使用WebRTC搭建前端视频聊天室——入门篇 (ht…

如果需要搭建实例的话可以参照SkyRTC-demo：github地址 (https://github.com/LingyuCoder/Sky…

其中使用了两个库：SkyRTC(github地址 (https://github.com/LingyuCoder/Sky…)和SkyRTC-client(github地址 (https://github.com/LingyuCoder/Sky…)

这两个库和demo都是我写的，如果有bug或是错误欢迎指出，我会尽力更正

前面的话

这篇文章讲述了WebRTC中所涉及的信令交换以及聊天室中的信令交换，主要内容来自 WebRTC in the real world: STUN, TURN and sig… ，我在这里提取出的一些信息，并添加了自己在开发时的一些想法。

WebRTC的服务器

WebRTC提供了浏览器到浏览器（点对点）之间的通信，但并不意味着WebRTC不需要服务器。暂且不说基于服务器的一些扩展业务，WebRTC至少有两件事必须要用到服务器：
1. 浏览器之间交换建立通信的元数据（信令）必须通过服务器
2. 为了穿越NAT和防火墙

为什么需要信令？

我们需要通过一系列的信令来建立浏览器之间的通信。而具体需要通过信令交换哪些内容呢？这里大概列了一下：
1. 用来控制通信开启或者关闭的连接控制消息
2. 发生错误时用来彼此告知的消息

3. 媒体流元数据，比如像解码器、解码器的配置、带宽、媒体类型等等
4. 用来建立安全连接的关键数据
5. 外界所看到的的网络上的数据，比如IP地址、端口等

在建立连接之前，浏览器之间显然没有办法传递数据。所以我们需要通过服务器的中转，在浏览器之间传递这些数据，然后建立浏览器之间的点对点连接。但是WebRTC API中并没有实现这些。

为什么WebRTC不去实现信令交换？

不去由WebRTC实现信令交换的原因很简单：WebRTC标准的制定者们希望能够最大限度地兼容已有的成熟技术。具体的连接建立方式由一种叫JSEP（JavaScript Session Establishment Protocol）的协议来规定，使用JSEP有两个好处：
1. 在JSEP中，需要交换的关键信息是多媒体会话描述（multimedia session description）。由于开发者在其所开发的应用程序中信令所使用的协议不同（SIP或是XMPP或是开发者自己定义的协议），WebRTC建立呼叫的思想建立在媒体流控制层面上，从而与上层信令传输相分离，防止相互之间的信令污染。只要上层信令为其提供了多媒体会话描述符这样的关键信息就可以建立连接，不管开发者用何种方式来传递。
2. JSEP的架构同时也避免了在浏览器上保存连接的状态，防止其像一个状态机一样工作。由于页面经常被频繁的刷新，如果连接的状态保存在浏览器中，每次刷新都会丢失。使用JSEP能使得状态被保存在服务器上

会话描述协议（Session Description Protocol）

JSEP将客户端之间传递的信令分为两种:offer信令和answer信令。他们主要内容的格式都遵循会话描述协议（Session Description Protocal，简称SDP）。一个SDP的信令的内容大致上如下：

```
v=0
o=- 7806956 075423448571 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio video data
a=msid-semantic: WMS 5UhOcZZB1uXtVbYAU5thB0SpkXbzk9FHo30g
m=audio 1 RTP/SAVPF 111 103 104 0 8 106 105 13 126
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:grnpQ0BSTSnBLroq
a=ice-pwd:N5i4DZKMM2L7FEYnhO8V7Kg5
a=ice-options:google-ice
a=fingerprint:sha-256 01:A3:18:0E:36:5E:EF:24:18:8C:8B:0C:9E:B0:84:F6:34:E9:42:E3:0F:43:64:ED:E
a=setup:actpass
a=mid:audio
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=recvonly
a=rtcp-mux
a=crypto:1 AES_CM_128_HMAC_SHA1_80 inline:qzcKu22ar1+lYah6o8ggzGcQ5obCttoOO2IzXwFV
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10
a=rtpmap:103 ISAC/16000
a=rtpmap:104 ISAC/32000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:106 CN/32000
a=rtpmap:105 CN/16000
```

这些都什么玩意？说实话我不知道，我这里放这么一大段出来，只是为了让文章内容显得很多...如果想深入了解的话，可以参考SDP for the WebRTC draft-nandakumar-rtcweb-… 自行进行解析

其实可以将其简化一下，它就是一个在点对点连接中描述自己的字符串，我们可以将其封装在JSON中进行传输，在PeerConnection建立后将其通过服务器中转后，将自己的SDP描述符和对方的SDP描述符交给PeerConnection就行了

信令与RTCPeerConnection建立

在前一篇文章中介绍过，WebRTC使用RTCPeerConnection来在浏览器之间传递流数据，在建立RTCPeerConnection实例之后，想要使用其建立一个点对点的信道，我们需要做两件事：
1. 确定本机上的媒体流的特性，比如分辨率、编解码能力啥的（SDP描述符）
2. 连接两端的主机的网络地址（ICE Candidate）

需要注意的是，由于连接两端的主机都可能在内网或是在防火墙之后，我们需要一种对所有联网的计算机都通用的定位方式。这其中就涉及NAT/防火墙穿越技术，以及WebRTC用来达到这个目的所ICE框架。这一部分在上一篇文章中有介绍，这里不再赘述。

通过offer和answer交换SDP描述符

大致上在两个用户（甲和乙）之间建立点对点连接流程应该是这个样子（这里不考虑错误的情况，RTCPeerConnection简称PC）：
1. 甲和乙各自建立一个PC实例
2. 甲通过PC所提供的 `createOffer()` 方法建立一个包含甲的SDP描述符的offer信令

3. 甲通过PC所提供的 `setLocalDescription()` 方法，将甲的SDP描述符交给甲的PC实例
4. 甲将offer信令通过服务器发送给乙
5. 乙将甲的offer信令中所包含的的SDP描述符提取出来，通过PC所提供的 `setRemoteDescription()` 方法交给乙的PC实例
6. 乙通过PC所提供的 `createAnswer()` 方法建立一个包含乙的SDP描述符answer信令
7. 乙通过PC所提供的 `setLocalDescription()` 方法，将乙的SDP描述符交给乙的PC实例
8. 乙将answer信令通过服务器发送给甲
9. 甲接收到乙的answer信令后，将其中乙的SDP描述符提取出来，调用 `setRemoteDescripttion()` 方法交给甲自己的PC实例

通过在这一系列的信令交换之后，甲和乙所创建的PC实例都包含甲和乙的SDP描述符了，完成了两件事的第一件。我们还需要完成第二件事——获取连接两端主机的网络地址

通过ICE框架建立NAT/防火墙穿越的连接

这个网络地址应该是能从外界直接访问，WebRTC使用ICE框架来获得这个地址。RTCPeerConnection在创立的时候可以将ICE服务器的地址传递进去，如：

```
var iceServer = {
    "iceServers": [{
        "url": "stun:stun.l.google.com:19302"
    }]
};
var pc = new RTCPeerConnection(iceServer);
```

当然这个地址也需要交换，还是以甲乙两位为例，交换的流程如下（RTCPeerConnection简称PC）：
1. 甲、乙各创建配置了ICE服务器的PC实例，并为其添加 `onicecandidate` 事件回调
2. 当网络候选可用时，将会调用 `onicecandidate` 函数
3. 在回调函数内部，甲或乙将网络候选的消息封装在ICE Candidate信令中，通过服务器中转，传递给对方
4. 甲或乙接收到对方通过服务器中转所发送过来ICE Candidate信令时，将其解析并获得网络候选，将其通过PC实例的 `addIceCandidate()` 方法加入到PC实例中

这样连接就创立完成了，可以向RTCPeerConnection中通过 `addStream()` 加入流来传输媒体流数据。将流加入到RTCPeerConnection实例中后，对方就可以通过 `onaddstream` 所绑定的回调函数监听到了。调用 `addStream()` 可以在连接完成之前，在连接建立之后，对方一样能监听到媒体流

聊天室中的信令

上面是两个用户之间的信令交换流程，但我们需要建立一个多用户在线视频聊天的聊天室。所以需要进行一些扩展，来达到这个要求

用户操作

首先需要确定一个用户在聊天室中的操作大致流程：
1. 打开页面连接到服务器上
2. 进入聊天室

3. 与其他所有已在聊天室的用户建立点对点的连接，并输出在页面上
4. 若有聊天室内的其他用户离开，应得到通知，关闭与其的连接并移除其在页面中的输出
5. 若又有其他用户加入，应得到通知，建立于新加入用户的连接，并输出在页面上
6. 离开页面，关闭所有连接

从上面可以看出来，除了点对点连接的建立，还需要服务器至少做如下几件事：
1. 新用户加入房间时，发送新用户的信息给房间内的其他用户
2. 新用户加入房间时，发送房间内的其他用户信息给新加入房间的用户
3. 用户离开房间时，发送离开用户的信息给房间内的其他用户

实现思路

以使用WebSocket为例，上面用户操作的流程可以进行以下修改：
1. 浏览器与服务器建立WebSocket连接
2. 发送一个加入聊天室的信令（join），信令中需要包含用户所进入的聊天室名称
3. 服务器根据用户所加入的房间，发送一个其他用户信令（peers），信令中包含聊天室中其他用户的信息，浏览器根据信息来逐个构建与其他用户的点对点连接
4. 若有用户离开，服务器发送一个用户离开信令（remove_peer），信令中包含离开的用户的信息，浏览器根据信息关闭与离开用户的信息，并作相应的清除操作
5. 若有新用户加入，服务器发送一个用户加入信令（new_peer），信令中包含新加入的用户的信息，浏览器根据信息来建立与这个新用户的点对点连接
6. 用户离开页面，关闭WebSocket连接

服务器实现

由于用户可以只是建立连接，可能还没有进入具体房间，所以首先我们需要一个容器来保存所有用户的连接，同时监听用户是否与服务器建立了WebSocket的连接：

```
var server = new WebSocketServer();
var sockets = [];

server.on('connection', function(socket){
    socket.on('close', function(){
        var i = sockets.indexOf(socket);
        sockets.splice(i, 1);
        //关闭连接后的其他操作
    });
    sockets.push(socket);
    //连接建立后的其他操作
});
```

由于有房间的划分，所以我们需要在服务器上建立一个容器，用来保存房间内的用户信息。显然对象较为合适，键为房间名称，值为用户信息列表。

同时我们需要监听上面所说的用户加入房间的信令（join），新用户加入之后需要向新用户发送房间内其他用户信息（peers）和向房间内其他用户发送新用户信息（new_peer），以及用户离开时向其他用户发送离开用户的信息（remove_peer）：

于是乎代码大致就变成这样：

```
                        "socketId": socket.id
                    }
                }));
            }
        }
        //从room中删除socket
        if (room) {
            i = this.rooms[room].indexOf(socket);
            this.rooms[room].splice(i, 1);
            if (this.rooms[room].length === 0) {
                delete this.rooms[room];
            }
        }
        //关闭连接后的其他操作
    });
    //根据前台页面传递过来的信令进行解析，确定应该如何处理
    socket.on('message', function(data) {
        var json = JSON.parse(data);
        if (json.eventName) {
            if (json.eventName === "join") {
                joinRoom(data, socket);
            }
        }
    });
    //将连接保存
    sockets.push(socket);
    //连接建立后的其他操作
});
```

最后再加上点对点的信令转发就行了，一份完整的代码可参照我写的SkyRTC项目源码 (https://github.com/LingyuCod…

参考资料

WebRTC in the real world: STUN, TURN and sig…

SDP for the WebRTC draft-nandakumar-rtcweb-…

2014年03月18日发布 (/a/1190000000439103)

你可能感兴趣的文章

使用WebRTC搭建前端视频聊天室——入门篇 (/a/1190000000436544) 87 收藏，42.1k 浏览

使用WebRTC搭建前端视频聊天室——点对点通信篇 (/a/1190000000733774) 25 收藏，5.5k 浏览

WebRTC 工作流程 (/a/1190000000608413) 1 收藏，1.9k 浏览

讨论区

谢谢了。。。不错。。

(/c/1050000000448461) **fzzf** (/u/fzzf) · 2014年03月28日

你好，请问支持多人在一个房间吗

(/c/1050000000601090) **JasonOldWoo** (/u/jasonoldwoo) · 2014年07月10日

回复 JasonOldWoo (/u/jasonoldwoo) :
支持多人的。

(/c/1050000000601139) **天镶** (/u/lingyucoder) · 2014年07月10日

通过offer和answer交换SDP描述符，其中有利用到服务器模块的中转，这里的的服务器模块描述符交换，是我们自己写吗？

(/c/1050000000720105) **neverMei** (/u/neverMei) · 2014年10月13日

回复 neverMei (/u/neverMei) :
不用自己写的，createOffer会生成

(/c/1050000000720332) **天镶** (/u/lingyucoder) · 2014年10月13日

回复 天镶 (/u/lingyucoder) :
谢谢了

(/c/1050000000720502) **neverMei** (/u/neverMei) · 2014年10月13日

回复 天镶 (/u/lingyucoder) :
我这边想实现一个Android手机P2P的视频音频交流，android手机使用的是libjingle XMPP协议，有没有以XMPP协议实现的信令服务器，这个信令服务器需不需要自己来实现？

(/c/1050000000720779) **neverMei** (/u/neverMei) · 2014年10月13日

回复 neverMei (/u/neverMei) :
我记得openfire是有jingle插件的

(/c/1050000000722814) **天镶** (/u/lingyucoder) · 2014年10月14日

看过你引用国外的一篇关于信令交换文章，明白了，signaling mechanisms是要自己实现的，我再去看看openfire

(/c/1050000000723369) **neverMei** (/u/neverMei) · 2014年10月15日

我写信令交换是onicecandidate一直没出发为什么？

(/c/1050000004317737) **kinglisky** (/u/kinglisky) · 1月16日

请先 登录 后评论

本文隶属于专栏

说学逗唱 (/blog/skyinlayer)

天镶 (/u/lingyucoder)
作者

关注专栏

系列文章

使用WebRTC搭建前端视频聊天室——点对点通信篇 (/a/1190000000733774) 25 收藏， 5.5k 浏览

使用WebRTC搭建前端视频聊天室——数据通道篇 (/a/1190000000733779) 25 收藏， 9.3k 浏览

分享扩散：

•••