初探WebRTC - Joe - SegmentFault

- 推荐 **2** 推荐
- 收藏 **13** 收藏，**1.4k** 浏览

简介

> WebRTC is a new front in the long war for an open and unencumbered web.

这句话是JavaScript之父*Brendan Eich*对于WebRTC的评价，大致意思是『WebRTC是争取开放和无阻碍Web的漫长战争中一条新战线』。

那么WebRTC到底是一种什么样的技术呢？WebRTC是一个免费的开放项目，提供了几个简单的API让浏览器、手机平台还有其他设备通过一个通用的协议进行实时通信，帮助开发者开发出丰富而且高质量的跨平台实时通信应用。

本文的目的就是通过简洁的介绍和引导让开发者了解WebRTC的工作流程并能够快速上手开发WebRTC应用。

API

- **MediaStream(getUserMedia)**
  MediaStream表示一段多媒体流，获取多媒体流的一个简单方式就是通过 **getUserMedia**函数，该函数可以调用设备的摄像头和麦克风，并拿到这些硬件生成的多媒体流。这些多媒体流可以输出到`video`标签或者一个**RTCPeerConnection**。

  getUserMedia接受3个参数：

  1. 一个限制对象，用于指定接受的流
  2. 获取stream成功后的回调函数，被调用时能获取到对应的流
  3. 获取stream失败后的回调函数，被调用时能获取到一个错误对象

  具体用法如下所示：

```
var getUserMedia = navigator.getUserMedia ||
                   navigator.webkitGetUserMedia ||
                   navigator.mozGetUserMedia;
var constraints = {video: true};
getUserMedia(constraints, function (stream) {
    var video = document.querySelector("video");
    //URL.createObjectURL方法把stream转换成blob，作为video的src属性进行播放
    video.src = window.URL.createObjectURL(stream);
```

```
    video.play();
}, function (error) {
    console.log(error);
});
```

- **RTCPeerConnection**

这是WebRTC的架构图，看完这张图，我表示完全不懂这是什么鬼，太复杂了。不过不懂没关系，RTCPeerConnection已经帮我们做了很多事情，我们只需要把RTCPeerConnection理解成一种p2p传输音视频数据的通道，但是我们仍然还需要服务器来为我们传递消息，因为在建立p2p之前需要先互相交换session、网络配置、媒体适配等信息。另外，WebRTC还需要服务器帮助完成NAT穿越，以及在p2p建立失败的时候作为中转服务器。具体用法将在流程讲解中说明。

- **RTCDataChannel**
  WebRTC除了支持视频和音频流之外，还支持其他类型的数据。RTCDataChannel用于点到点的任意数据交换，具有低延迟和高吞吐量的特点。RTCDataChannel必须建立在RTCPeerConnnection之上，没办法单独工作。

  这个API潜在的应用场景很多，例如：

  - 游戏
  - 远程桌面应用
  - 实时文字聊天
  - 文件传输

  创建一个RTCDataChannel的方式如下：

```
var RTCPeerConnection = webkitRTCPeerConnection || mozRTCPeerConnection;
var peerConn = new RTCPeerConnection();
var dc = peerConn.createDataChannel("label");
```

流程讲解

看完上面的内容，大家应该还是处于云里雾里的状态，不知道WebRTC应用要如何开发，3个API如何配合使用。没关系，这里才是重点，看完这一段，相信你的所有疑惑就都解开了。

要用WebRTC建立一个p2p通道需要经历2个步骤：

1. 获得本机SDP描述符并交换
   - A、B均实例化一个RTCPeerConnection(以下简称rpcA和rpcB),调用rpcA的`createOffer()`方法建立一个offer信令，并且拿到A的SDP
   - 通过rpcA的`setLocalDescription()`方法设置A机器的本地描述
   - A通过服务器将offer信令发给B

- B接收到A的offer信令，通过rpcB的`setRemoteDescription()`方法设置远程机器（即A）的描述
- B调用rpcB的`createAnswer()`方法建立一个answer信令，并且拿到B的SDP
- 通过rpcB的`setLocalDescription()`方法设置B机器的本地描述
- B通过服务器将answer信令发送给A
- A接收到B的answer信令，通过rpcA的`setRemoteDescription()`方法设置远程机器B的描述

这个过程完成后，A和B就都拿到各自的SDP描述符了

2. 通过ICE框架连接两段主机的网络地址

ICE框架具体内容我不清楚，咱们姑且先了解在WebRTC中如何使用，对ICE感兴趣的同学可以自行谷歌。

在实例化RTCPeerConnection对象的时候可以传入ICE服务器的地址，我们可以使用谷歌提供的『stun:stun.l.google.com:19302』或者Mozilla提供的『stun:stun.services.mozilla.com』，代码如下：

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

在rpc上绑定onicecandidate事件的回调函数，当网络候选可用时这个函数会被调用，在这个回调函数中，本机可以拿到ice candidate信令，然后通过服务器发给远程机器，远程机器通过自己的rpc实例的`addIceCandidate()`方法添加，同样地，远程机器也应该将自己的ice candidate信令通过服务器发送给本机。

当双方的ice candidate交换完成时，连接就建立成功了，可以在rpc示例上调用`addStream()`来添加流，另一边通过绑定onaddstream事件就可以获取到传过去的流。

服务器通信部分，没有规定必须要用某种协议，所以只要能在两边传输消息的技术可以使用，例如*WebSocket*、*XHR*等，大家自行选择即可。

示例

本来想自己写个例子给大家参考的，后来发现网上有个非常棒的实例教程，我就直接上地址了 https://bitbucket.org/webrtc/codelab

小结

写这篇文章的目的是为了让学习WebRTC的同学能够快速上手，对WebRTC技术有个整体的概念，所以文章写的比较简短，专注于讲解WebRTC的运作流程，对一些相关技术，比如ICE、SDP等没有深入探讨（当然我也不太懂）。如果你想快速上手制作WebRTC应用，看完这篇文章再对着codelab提供的例子一步一步跟着做应该就没问题了。Good luck!