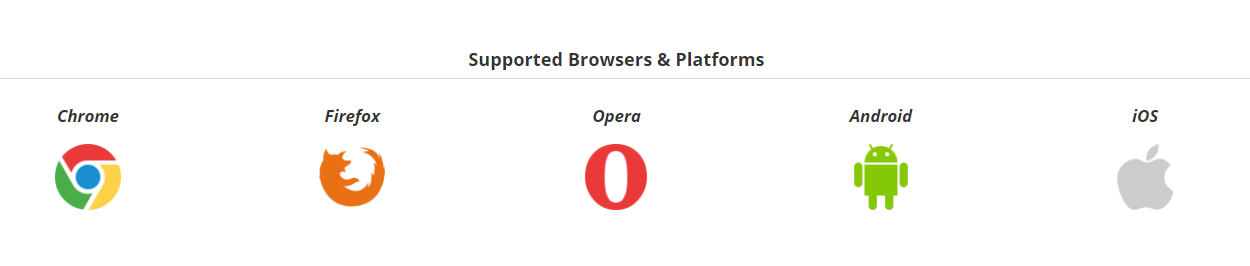
**WebRTC介绍**

**1、下载和示例：**

官网： <https://webrtc.org/>



**2、WebRTC是什么？**

WebRTC是一项可以让数据，视频，甚至屏幕共享通过P2P连接的形式进行传播的技术；

这就意味着数据不需要通过服务器进行传输。

peer跟peer之间一旦建立连接就可以直接传输音视频数据流，并不需要借助第三方服务器中转。

WebRTC协议被内置在Chrome和Firefox浏览器中，并且包括这三个部分：

# getUserMedia：其功能是允许网页浏览器获得摄像头和麦克风的使用权限，并且捕获媒体。用于获取媒体数据，例如来自摄像头和麦克风的视频流和音频流；

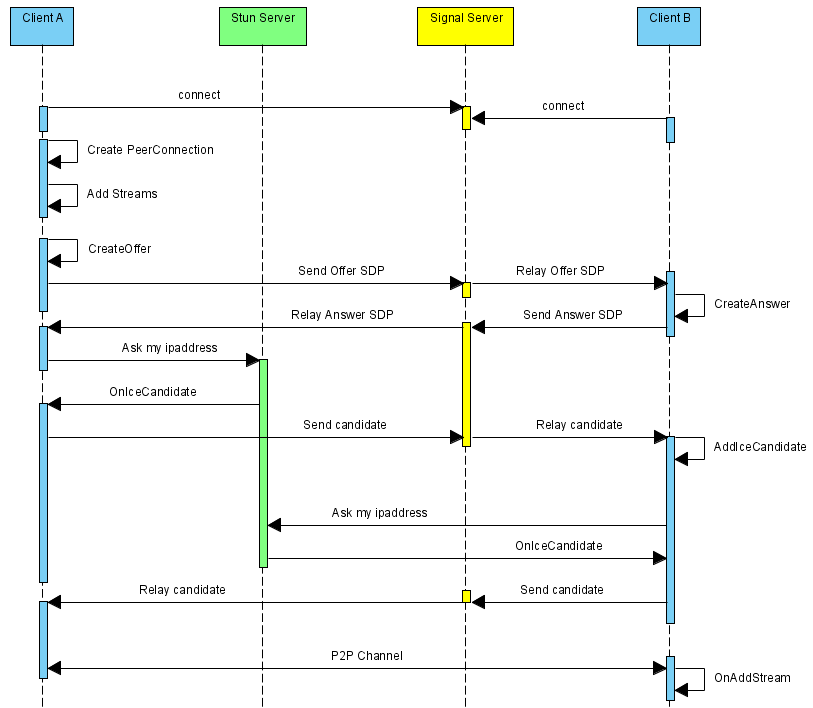
# RTCPeerConnection：负责管理端到端连接,用于peer跟peer之间呼叫和建立连接以便传输音视频数据流。

# RTCDataChannel：其功能是允许浏览器分享任意数据,用于peer跟peer之间传输音视频之外的一般数据。

需要注意的是这几个API的名称在不同浏览器及同一浏览器的不同版本之间略有差异，比如PeerConnection在FireFox上叫做mozRTCPeerConnection，而在当前版本的Chrome上叫做webkitRTCPeerConnection

**3、序列图**

场景：ClientA向ClientB发起对聊请求



图中Stun服务器和Signal服务器需要自己实现，WebRTC并不提供。

Stun服务器可以用google提供的实现stun协议的测试服务器（stun:stun.l.google.com:19302），Signal服务器则完全需要自己实现了，它需要在ClientA和ClientB之间传送彼此的SDP信息和candidate信息，ClientA和ClientB通过这些信息建立P2P连接来传送音视频数据。由于网络环境的复杂性，并不是所有的客户端之间都能够建立P2P连接，这种情况下就需要有个relay服务器做音视频数据的中转（这种情况暂时不考虑）。

**步骤：**

• ClientA首先创建PeerConnection对象，然后打开本地音视频设备，将音视频数据封装成MediaStream添加到PeerConnection中。

• ClientA调用PeerConnection的CreateOffer方法创建一个用于offer的SDP对象，SDP对象中保存当前音视频的相关参数。ClientA通过PeerConnection的SetLocalDescription方法将该SDP对象保存起来，并通过Signal服务器发送给ClientB。

• ClientB接收到ClientA发送过的offer SDP对象，通过PeerConnection的SetRemoteDescription方法将其保存起来，并调用PeerConnection的CreateAnswer方法创建一个应答的SDP对象，通过PeerConnection的SetLocalDescription的方法保存该应答SDP对象并将它通过Signal服务器发送给ClientA。送过来的音视频流，会通过PeerConnection的OnAddStream回调接口返回一个标识ClientA

• ClientA接收到ClientB发送过来的应答SDP对象，将其通过PeerConnection的SetRemoteDescription方法保存起来。

• 在SDP信息的offer/answer流程中，ClientA和ClientB已经根据SDP信息创建好相应的音频Channel和视频Channel并开启Candidate数据的收集，Candidate数据可以简单地理解成Client端的IP地址信息（本地IP地址、公网IP地址、Relay服务端分配的地址）。

• 当ClientA收集到Candidate信息后，PeerConnection会通过OnIceCandidate接口给ClientA发送通知，ClientA将收到的Candidate信息通过Signal服务器发送给ClientB，ClientB通过PeerConnection的AddIceCandidate方法保存起来。同样的操作ClientB对ClientA再来一次。

• 这样ClientA和ClientB就已经建立了音视频传输的P2P通道，ClientB接收到ClientA传端音视频流的MediaStream对象，在ClientB端渲染出来即可。同样操作也适应ClientB到ClientA的音视频流的传输。