**使用WebRTC创建一个网络摄像头通信App**

WebRTC是一个协议，允许人们使用JavaScript在两点之间创建实时通讯。

我们可以用这个结构使两个或更多浏览器之间实现直接交流，而不需要中心服务器。

服务器只需要在连接的时候被使用，因此每个客户端知道如何连接彼此。

我们可以使用这个特性创建什么类型的App呢？例如，直接网络摄像头连接。点对点通话，文件共享，还有更多。

本教程我会介绍一个当你第一次使用的时候，会发出惊呼的App：一个网络摄像头通信App.

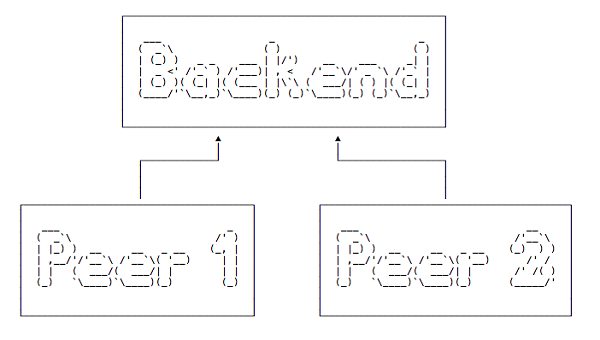
我们不会使用原始WebRTC API,然而，我们需要注意很多细节。这就是library的作用，它们做出了很好的抽象，人们可以集中精力创建App而不是将精力花费的底层API上。

其中一个library是PeerJS,它使得实时通信变得非常简单。简单来说就是WebRTC,它的抽象使得结果获取更快，之后你可以学习内部是如何运作的。

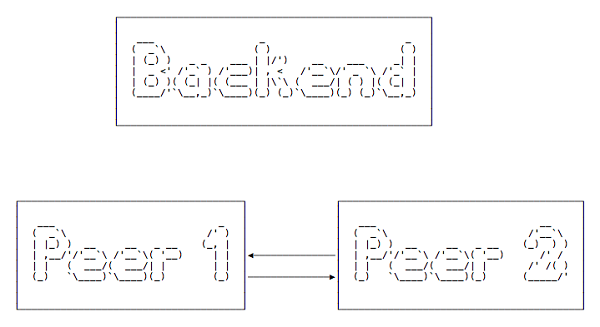
建议：当你建立一个App时候使用Bit分享到一个可重复利用的集合中，并且在你所有的项目中同步它们！不妨试一下，可以加快你的工作速度。

**后端**

首先我们需要创建后端。尽管我们会实现直接点对点通信，起初的握手和合作需要中心服务器。

[](http://webrtc.org.cn/wp-content/uploads/2018/12/111.png)

一旦握手完成，点点之间会直接交流，而不再需要依靠后端。

[](http://webrtc.org.cn/wp-content/uploads/2018/12/222.png)

PeerJS为我们提供了这样一个服务器，安装过程很简单，容易运行。

在一个文件夹下，初始化一个npm项目使用npm init命令，使用npm install peer指令安装PeerJS，接着你可以使用npx运行它：

Npx peerjs –port 9000

使用npx peerjs –help查看更多选择。

这就是你的后端。

现在我们可以创建一个最简单的App，我们设置一个接收端和一个发送端。

**前端**

首先创建一个接收端，连接PeerJS服务器，等待接收信息。第一个参数new Peer()为我们的端点名称，我们叫做receiver，使得表意更清晰。：  
导入PeerJS客户端：

1. <script src="https://cdnjs.cloudflare.com/ajax/libs/peerjs/0.3.16/peer.min.js"></script>

接着初始化Peer对象。当另一个端点连接我们的时候，Connection事件被调用。当接收到一些信息之后，data事件被调用：

1. const peer = new **Peer**('receiver', { host: 'localhost', port: 9000, path: '/' })
2. peer.**on**('connection', (conn) => {
3. conn.**on**('data', (data) => {
4. console.**log**(data);
5. })
6. })

让我们创建通信的另一端。我们称作sender，因为它会连接并发送信息到接收端。

初始化Peer对象，接着请求端点连接receiver端，接收端我们之前已经创建好。一旦连接建立，open事件启动，接着调用send()方法来向接收端发送信息：

1. const peer = new **Peer**('sender', { host: 'localhost', port: 9000, path: '/' })
2. const conn = peer.**connect**('receiver')
3. conn.**on**('open', () => {
4. conn.**send**('hi!')
5. })

这就是最简单的例子。

首先打开接收端，接着打开发送端。接受者从发送者直接获取信息，不需要中心服务器。服务器只需要交换信息，确保两端连接。之后，就不会干预两端之间的交流。

这是一个非常基础的信息连接。

下一步，我们不会发送信息，而是让两端彼此共享网络摄像头流。

在客户端，我们没有使用peer.connect()连接，而是使用了peer.call():

1. const call = peer.**call**('receiver', localStream)
2. })

在接收端，当接收到一个通话事件反馈之后，必须对此作出应答：

1. peer.**on**('call', call => {
2. call.**answer**(localStream)
3. })

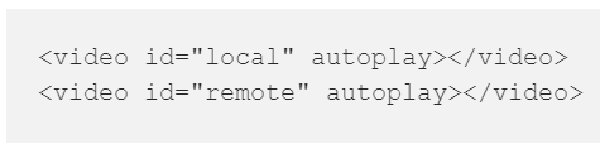
就像电话交流一样，我们不能自动回复每个来电，必须明确的作出应答。

每个通话中的LocalStream是什么？它是网络摄像头产生的流，我们必须通过调用navigator.mediaDevices.getUserMedia()得到它，这是一个浏览器API。

这是一个异步通讯，因此我们使用async/await来等待执行，我们需要将通话包裹在一个异步函数中，首先：

1. const startChat = **async** () => {
2. const localStream = await navigator.mediaDevices.**getUserMedia**({
3. video: true
4. })
5. }
6. **startChat**()

一旦我们得到了localStream对象，可以将它分配给HTML网页中的一个video元素。我们可以创建本地和远程视频元素：

[](http://webrtc.org.cn/wp-content/uploads/2018/12/333.png)

将流分配给video#local元素：

1. document.**querySelector**('video#local').srcObject = localStream

调用receiver端，传递localStream对象：

1. const call = peer.**call**('receiver', localStream)

接收端的代码如下：

1. peer.**on**('call', call => {
2. call.**answer**(localStream)
3. })

我们也需要得到媒体流。代码非常简单，和发送端类似，只是我们把所有代码包裹在call事件反馈中：

1. peer.**on**('call', call => {
2. const startChat = **async** () => {
3. const localStream = await navigator.mediaDevices.**getUserMedia**({
4. video: true
5. })
6. document.**querySelector**('video#local').srcObject = localStream
7. call.**answer**(localStream)
8. }
9. **startChat**()
10. })

**展示远程流**

我们还需要添加最后一部分到发送端和接收端中。一旦从call对象的stream事件中得到远程流，我们需要将它附加在video#remote元素上。

1. call.**on**('stream', remoteStream => {
2. document.**querySelector**('video#remote').srcObject = remoteStream
3. })

接收端代码如下：

1. peer.**on**('call', call => {
2. const startChat = **async** () => {
3. const localStream = await navigator.mediaDevices.**getUserMedia**({
4. video: true
5. })
6. document.**querySelector**('video#local').srcObject = localStream
7. call.**answer**(localStream)
8. call.**on**('stream', remoteStream => {
9. document.**querySelector**('video#remote').srcObject = remoteStream
10. })
11. }
12. **startChat**()
13. })

发送端代码如下：

1. const startChat = **async** () => {
2. const localStream = await navigator.mediaDevices.**getUserMedia**({
3. video: true
4. })
5. document.**querySelector**('video#local').srcObject = localStream
6. const call = peer.**call**('receiver', localStream)
7. call.**on**('stream', remoteStream => {
8. document.**querySelector**('video#remote').srcObject = remoteStream
9. })
10. }
11. **startChat**()

当其中一端通过导向新网页或关闭浏览器标签关闭了连接之后，另一端停止接收流，远程视频流停止。

**总结**

我们使用WebRTC创建了一个非常简单的网络摄像头通信App。创建了两个文件来处理两端通信，但是没必要这样做。你可以建立一个用户接口，允许用户自己决定是否需要调用，更重要的是，他们想与谁通话。可以允许用户输入用户名或者从列表中选择来实现这个功能。

原文标题：Build a Webcam Communication App using WebRTC  
作者：[‘Flavio Copes’](https://blog.bitsrc.io/build-a-webcam-communication-app-using-webrtc-9737384e84be)