

08 | 有话好商量，论媒体协商

2019-08-01 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 13:53 大小 12.72M



在《[07 | 你竟然不知道 SDP ? 它可是 WebRTC 的驱动核心！](#)》一文中，我向你详细介绍了标准 SDP 规范，以及 WebRTC 与标准 SDP 规范的一些不同，而本文我们将重点学习一下 WebRTC 究竟是如何使用 SDP 规范进行**媒体协商**的。

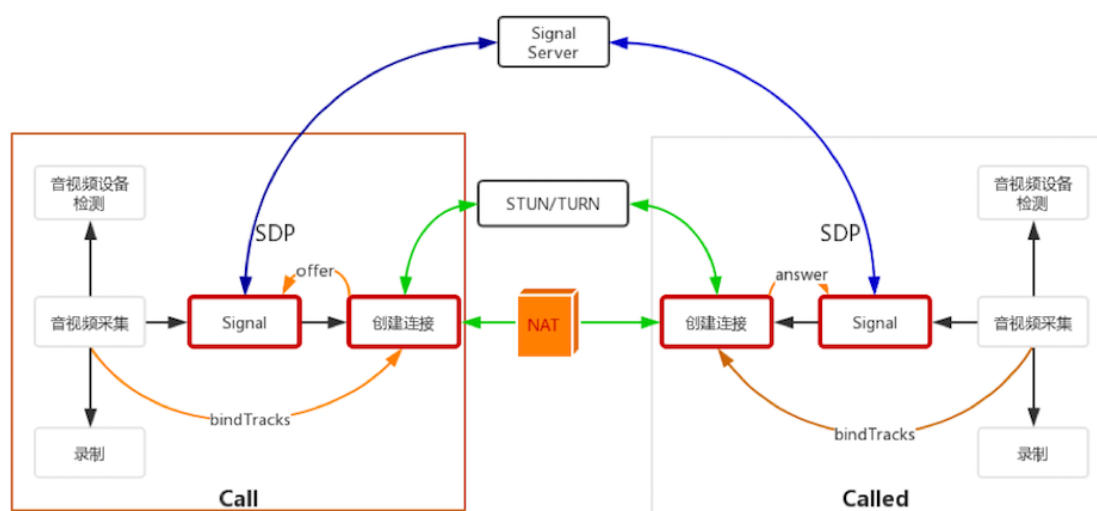
我们平时所说的**协商**你应该清楚是什么意思，说白了就是讨价还价。以买白菜为例，商家说 5 元一颗，买家说身上只有 4.5 元卖不卖？商家同意卖，这样一次协商就完成了。

而**媒体协商**也是这个意思，只不过它们讨价还价的不是一般商品，而是与媒体相关的能力。那**媒体能力**是什么呢？实际就是你的设备所支持的音视频编解码器、使用的传输协议、传输的速率是多少等信息。

所以简单地说，**媒体协商**就是看看你的设备都支持那些编解码器，我的设备是否也支持？如果我的设备也支持，那么咱们双方就算协商成功了。

在 WebRTC 处理过程中的位置

在正式进入主题之前，我们还是来看看本文在整个 WebRTC 处理过程中的位置，如下图所示：



WebRTC 处理过程图

通过这张图你可以了解到，本文所涉及的内容包括**创建连接**和**信令**两部分。

创建连接，指的是创建 `RTCPeerConnection`，它负责端与端之间彼此建立 P2P 连接。在后面 `RTCPeerConnection` 一节中，我们还会对其做进一步的介绍。

信令，指的是客户端通过信令服务器交换 SDP 信息。

所以从本文开始，我们就开始讲解 WebRTC 最核心的一部分知识了，下面就让我们开始吧。

WebRTC 中媒体协商的作用

在 WebRTC 1.0 规范中，在双方通信时，双方必须清楚彼此使用的编解码器是什么，也必须知道传输过来的音视频流的 SSRC（SSRC 的概念参见[《06 | WebRTC 中的 RTP 及 RTCP 详解》](#)一文）信息，如果连这些最基本的信息彼此都不清楚的话，那么双方是无法正常通信的。

举个例子，如果 WebRTC 不清楚对方使用的是哪种编码器编码的数据，比如到底是 H264，还是 VP8？那 WebRTC 就无法将这些数据包正常解码，还原成原来的音视频帧，这将导致音视频无法正常显示或播放。

同样的道理，如果 WebRTC 不知道对方发过来的音视频流的 SSRC 是多少，那么 WebRTC 就无法对该音视频流的合法性做验证，这也将导致你无法观看正常的音视频。因为对于无法验证的音视频流，WebRTC 在接收音视频包后会直接将其抛弃。

通过上面的描述，我想你已经非常清楚媒体协商的作用是什么了。没错，**媒体协商的作用就是让双方找到共同支持的媒体能力**，如双方都支持的编解码器，从而**最终实现彼此之间的音视频通信**。

那 WebRTC 是怎样进行媒体协商的呢？这就要用到[《07 | 你竟然不知道 SDP？它可是 WebRTC 的驱动核心！》](#)文章中讲解的 SDP 了。

首先，通信双方将它们各自的媒体信息，如编解码器、媒体流的 SSRC、传输协议、IP 地址和端口等，按 SDP 格式整理好。

然后，通信双方通过信令服务器交换 SDP 信息，并待彼此拿到对方的 SDP 信息后，找出它们共同支持的媒体能力。

最后，双方按照协商好的媒体能力开始音视频通信。

WebRTC 进行媒体协商的步骤基本如上所述。接下来，我们来看看 WebRTC 具体是如何操作的。


RTCPeerConnection

讲到媒体协商，我们就不得不介绍一下 RTCPeerConnection 类，顾名思义，它表示的就是端与端之间建立的**连接**。

该类是整个 WebRTC 库中**最关键**的一个类，通过它创建出来的对象可以做很多事情，如 NAT 穿越、音视频数据的接收与发送，甚至它还可以用于非音视频数据的传输等等。

而在这里我们之所以要介绍 `RTCPeerConnection`，最主要的原因是**我们今天要讲的端到端之间的媒体协商，就是基于 `RTCPeerConnection` 对象实现的**。

首先，我们来看一下如何创建一个 `RTCPeerConnection` 对象：

 复制代码

```
1 ...  
2 var pcConfig = null;  
3 var pc = new RTCPeerConnection(pcConfig);  
4 ...
```

在 JavaScript 下创建 `RTCPeerConnection` 对象非常简单，如上所述，只要通过 `new` 关键字创建即可。

在创建 `RTCPeerConnection` 对象时，还可以给它传一个参数 **`pcConfig`**，该参数的结构非常复杂，这里我们先将其设置为 `null`，后面在《12 | `RTCPeerConnection`：音视频实时通讯的核心》一文中我再对其做更详尽的描述。

有了 `RTCPeerConnection` 对象，接下来，让我们再来看看端与端之间是如何进行媒体协商的吧！

媒体协商的过程

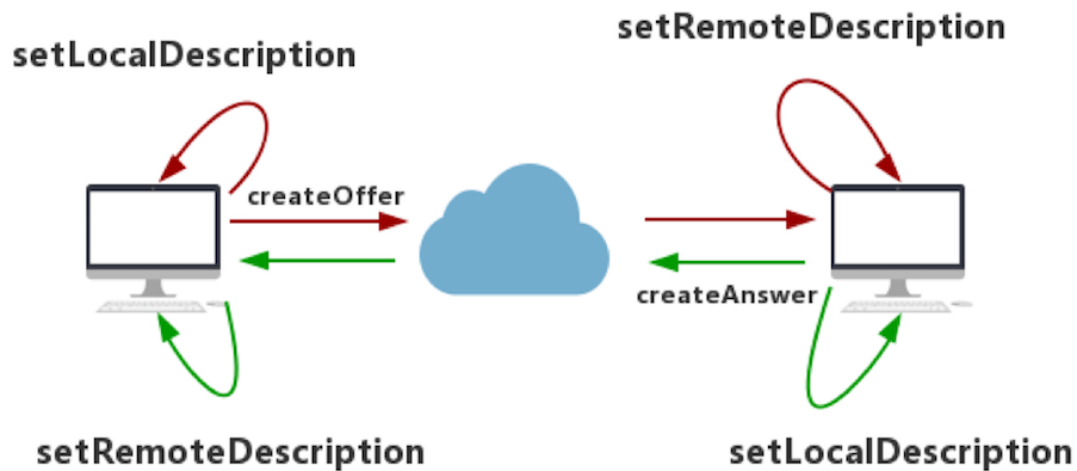
在通讯双方都创建好 `RTCPeerConnection` 对象后，它们就可以开始进行媒体协商了。不过在进行媒体协商之前，有两个重要的概念，即 **Offer** 与 **Answer**，你必须要弄清楚。

Offer 与 Answer 是什么呢？对于 1 对 1 通信的双方来说，我们称首先发送媒体协商消息的一方为**呼叫方**，而另一方则为**被呼叫方**。

Offer，在双方通讯时，呼叫方发送的 SDP 消息称为 Offer。

Answer，在双方通讯时，被呼叫方发送的 SDP 消息称为 Answer。

在 WebRTC 中，双方协商的整个过程如下图所示：



媒体协商过程图

首先，呼叫方创建 Offer 类型的 SDP 消息。创建完成后，调用 `setLocalDescription` 方法将该 Offer 保存到本地 Local 域，然后通过信令将 Offer 发送给被呼叫方。

被呼叫方收到 Offer 类型的 SDP 消息后，调用 `setRemoteDescription` 方法将 Offer 保存到它的 Remote 域。作为应答，被呼叫方要创建 Answer 类型的 SDP 消息，Answer 消息创建成功后，再调用 `setLocalDescription` 方法将 Answer 类型的 SDP 消息保存到本地的 Local 域。最后，被呼叫方将 Answer 消息通过信令发送给呼叫方。至此，被呼叫方的工作就完部完成了。

接下来是呼叫方的收尾工作，呼叫方收到 Answer 类型的消息后，调用 `RTCPeerConnecton` 对象的 `setRemoteDescription` 方法，将 Answer 保存到它的 Remote 域。

至此，**整个媒体协商过程处理完毕。**

当通讯双方拿到彼此的 SDP 信息后，就可以进行媒体协商了。媒体协商的具体过程是在 WebRTC 内部实现的，我们就不去细讲了。你只需要记住本地的 SDP 和远端的 SDP 都设

置好后，协商就算成功了。

媒体协商的代码实现


了解了 WebRTC 的媒体协商过程之后，我们再看一下如何使用 JavaScript 代码来实现这一功能。浏览器提供了几个非常方便的 API，这些 API 是对底层 WebRTC API 的封装。如下所示：

- `createOffer`，创建 Offer；
- `createAnswer`，创建 Answer；
- `setLocalDescription`，设置本地 SDP 信息；
- `setRemoteDescription`，设置远端的 SDP 信息。

接下来，我们就结合上述的协商过程对这几个重要的 API 做下详细的讲解。

1. 呼叫方创建 Offer

当呼叫方发起呼叫之前，首先要创建 Offer 类型的 SDP 信息，即调用 **RTCPeerConnection** 的 `createOffer()` 方法。代码如下：

 复制代码

```
1  function doCall() {  
2      console.log('Sending offer to peer');  
3      pc.createOffer(setLocalAndSendMessage, handleCreateOfferError);  
4  }
```

如果 `createOffer` 函数调用成功的话，浏览器会回调我们设置的 `setLocalAndSendMessage` 方法，你可以在 `setLocalAndSendMessage` 方法里获取到 **RTCSessionDescription 类型的 SDP 信息**；如果出错则会回调 `handleCreateOfferError` 方法。

最终，在 `setLocalAndSendMessage` 回调方法中，通过 **`setLocalDescription()`** 方法将本地 SDP 描述信息设置到 WebRTC 的 Local 域。然后通过信令通道将此会话描述发送给被呼叫方。代码如下所示：

```

1  function setLocalAndSendMessage(sessionDescription) {
2      pc.setLocalDescription(sessionDescription);
3      sendMessage(sessionDescription);
4  }

```

2. 被呼叫方收到 Offer

被呼叫方收到 Offer 后，调用 **setRemoteDescription** 方法设置呼叫方发送给它的 Offer 作为远端描述。代码如下：

```

1  socket.on('message', function(message) {
2      ...
3      } else if (message.type === 'offer') {
4
5          pc.setRemoteDescription(new RTCSessionDescription(message));
6          doAnswer();
7      } else if (...) {
8          ...
9      }
10     ....
11 });

```

3. 被呼叫方创建 Answer

然后，被呼叫方调用 `RTCPeerConnection` 对象的 `createAnswer` 方法，它会生成一个与远程会话兼容的本地会话，并最终将该会话描述发送给呼叫方。


```

1  function doAnswer() {
2      pc.createAnswer().then(
3          setLocalAndSendMessage,
4          onCreateSessionDescriptionError
5      );
6  }

```

4. 呼叫方收到 Answer

当呼叫方得到被呼叫方的会话描述，即 SDP 时，调用 `setRemoteDescription` 方法，将收到的会话描述设置为一个远程会话。代码如下：

 复制代码

```
1  socket.on('message', function(message) {
2      ...
3      } else if (message.type === 'answer') {
4
5          pc.setRemoteDescription(new RTCSessionDescription(message));
6      } else if (...) {
7          ...
8      }
9      ....
10 });
```

此时，媒体协商过程完成。紧接着在 WebRTC 底层会收集 Candidate，并进行连通性检测，最终在通话双方之间建立起一条链路来。

以上就是通信双方交换媒体能力信息的过程。对于你来说，如果媒体协商这个逻辑没搞清楚的话，那么，你在编写音视频相关程序时很容易出现各种问题，最常见的就是音视之间不能互通。

另外，需要特别注意的是，通信双方链路的建立是在设置本地媒体能力，即调用 `setLocalDescription` 函数之后才进行的。

小结

在本文中，我向你详细介绍了 WebRTC 进行媒体协商的过程，这个过程是你必须牢记在脑子里的。如果对这块不熟悉的话，后面你在真正使用 WebRTC 开发音视频应用程序时就会遇到各种困难，如音视频不通、单通等情况。

另外，本文还向你简要介绍了 `RTCPeerConnection` 对象，它是 WebRTC 的核心 API，媒体协商的具体操作都是通过该对象来完成的。对于该对象，我会在后面的文章中做更详尽的解答。

`RTCPeerConnection` 除了会在端与端之间建立连接、传输音视频数据外，还要进行两次绑定：一次是与媒体源进行绑定，以解决数据从哪里来的问题；另外一次是与输出进行绑定，

以解决接收到的音视频数据显示 / 播放的问题。

思考时间

在 WebRTC 中，SDP 消息的交换是使用 RTCPeerConnection 对象完成的吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。



从 0 打造音视频直播系统

手把手教你打造实时互动音视频直播系统

李超

新东方音视频直播技术专家
前沪江音视频架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | 你竟然不知道SDP？它可是WebRTC的驱动核心！

下一篇 09 | 让我们揭开WebRTC建立连接的神秘面纱

精选留言 (8)

写留言



wuqilv

2019-08-01

客户端通过信令服务器交换 SDP 信息。

展开 ▾



2



Jian

2019-08-01

呼叫方和被呼叫方的角色是如何确认的呢？会否存在两端都向对方发送offer的情况？是由服务器确定的？

作者回复: 谁先发起呼叫谁就发offer,另一方发answer;这完全有应用层控制；比如第一个人进入房间后，就在哪里等待，当发现第二个人上来的时候它就给对方发offer 就好了。如果两个人同时进入房间，就在服务器端建个队列，让他们顺序进入就好了，非常好处理对吧？另外两端都发offer 那协商必然失败。



1



scorio

2019-08-02

老师，从NVR推流到Web端展示监控的实时视频，有什么好的解决方案吗？系统运行在内网上

作者回复: 应该要做协议转换，要看 NVR用的啥协议，SIP？



君

2019-08-02

请问老师哪些开源的sip框架支持webrtc的吗

展开 ▾

作者回复: 你可以使用sip 协议做信令，但sip 协议用的人比较少，一般都在监控系统中使用，目前开源的基本没有人使用sip 与WebRTC 结合

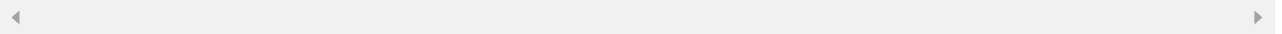


Geek_e39e6f

2019-08-01

WebRTC 底层会收集 Candidate，就是通过stun，turn服务获取候选地址吗？这个流程不是在交换sdp之前就应该获取，然后记录在sdp里，发送给对端的吗？

作者回复: 你说的那是以前了, 现在方式已经改了!



1

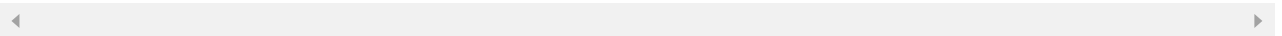


花果山の酸梅汤

2019-08-01

感谢讲解, 请问李老师webrtc的c++ API是如何映射为JS API的, 另外官方JS API规范定义的功能, 如何可以确认那些是在浏览器中实现, 那些是在webrtc c++中实现?

作者回复: Js 与 c++的互调是通过V8 引擎, 至于那些是WebRTC 实现的, 那些是浏览器实现的, 你后面慢慢熟悉了就清楚了, 别着急



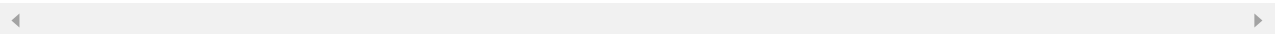
Jason

2019-08-01

思考题: 从老师的讲解来看, SDP 消息的交换不是使用 RTCPeerConnection对象完成的, RTCPeerConnection对象负责创建offer、设置本地SDP描述信息、设置远端SDP描述信息、创建answer。交换SDP消息应该是socket对象完成的, 但socket的类型啥呢, 还不知道。

展开

作者回复: 有没有想过用http? 它可是浏览器天然的哈



Beast-Of-Prey

2019-08-01

发送信令用socket?

展开

作者回复: 由于信令数据量不大, 所以你可选择的协议就比较多了, TCP、HTTP/HTTPS、WS/WSS, 都可以, 底层实现都是用的socket

