

19 | WebRTC能不能进行文本聊天呢？

2019-08-27 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 17:56 大小 12.33M



WebRTC 不但可以让你进行音视频通话，而且还可以用它传输普通的二进制数据，比如可以利用它实现文本聊天、文件的传输等等。

WebRTC 的**数据通道 (RTCDataChannel)** 是专门用来传输除了音视频数据之外的任何数据，所以它的应用非常广泛，如实时文字聊天、文件传输、远程桌面、游戏控制、P2P 加速等都是它的应用场景。

像文本聊天、文件传输这类应用，大多数人能想到的通常是通过服务器中转数据的方案，但 WebRTC 则优先使用的是**P2P 方案，即两端之间直接传输数据**，这样就大大减轻了服务器的压力。当然 WebRTC 也可以采用中继的方案，这就需要你根据自己的业务需要进行选择，非常灵活。

RTCDataChannel 介绍

RTCDataChannel 就是 WebRTC 中专门用来传输非音视频数据的类，它的设计模仿了 WebSocket 的实现，使用起来非常方便，关于这一点我将在下面的“RTCDataChannel 的事件”部分向你做更详细的介绍。

另外，RTCDataChannel 支持的数据类型也非常多，包括：字符串、Blob、ArrayBuffer 以及 ArrayBufferView。

实际上，关于这几种类型的联系与区别我在前面[《04 | 可以把采集到的音视频数据录制下来吗？》](#)一文中已经向你做过详细的介绍，如果你现在记不清了，可以再去回顾一下。

WebRTC 的 RTCDataChannel 使用的传输协议为 SCTP，即 Stream Control Transport Protocol。下面图表表示的就是在 TCP、UDP 及 SCTP 等不同传输模式下，数据传输的可靠性、传递方式、流控等信息的对比：

	TCP	UDP	SCTP
可靠性	可靠	不可靠	可配置
传递方式	有序	无序	可配置
传输方式	按字节传输	按消息传输	按消息传输
流控	支持	不支持	支持
拥塞控制	支持	不支持	支持

RTCDataChannel 既可以在可靠的、有序的模式下工作，也可在不可靠的、无序的模式下工作，具体使用哪种模式可以根据用户的配置来决定。下面我们来看看它们之间的区别。

可靠有序模式（TCP 模式）：在这种模式下，消息可以有序到达，但同时也带来了额外的开销，所以在这种模式下**消息传输会比较慢**。


不可靠无序模式（UDP 模式）：在此种模式下，不保证消息可达，也不保证消息有序，但在这种模式下没有什么额外开销，所以它**非常快**。

部分可靠模式（SCTP 模式）：在这种模式下，消息的可达性和有序性可以根据业务需求进行配置。

那接下来我们就来看一下到底该如何配置 RTCDataChannel 对象吧。

配置 RTCDataChannel

在创建 RTCDataChannel 对象之前，首先要创建 RTCPeerConnection 对象，因为 **RTCDataChannel 对象是由 RTCPeerConnection 对象生成的**。有了 RTCPeerConnection 对象后，调用它的 createDataChannel 方法，就可以将 RTCDataChannel 创建出来了。具体操作如下：

 复制代码

```
1 ...
2 var pc = new RTCPeerConnection(); // 创建 RTCPeerConnection 对象
3 var dc = pc.createDataChannel("dc", options); // 创建 RTCDataChannel 对象
4 ...
```

从上面的代码中可以看到 RTCDataChannel 对象是由 RTCPeerConnection 对象创建的，在创建 RTCDataChannel 对象时有两个参数。

第一个参数是一个标签（字符串），相当于给 RTCDataChannel 起了一个名字；

第二个参数是 options，其形式如下：

 复制代码

```
1 var options = {
2     ordered: false,
3     maxPacketLifeTime: 3000
4 };
```

其实 **options** 可以指定很多选项，比如像上面所设置的，指定了创建的 RTCDataChannel 是否是有序的，以及最大的存活时间。

下面我就向你详细介绍一下 options 所支持的选项。

ordered: 消息的传递是否有序。

maxPacketLifeTime: 重传消息失败的最长时间。也就是说超过这个时间后，即使消息重传失败了也不再进行重传了。

maxRetransmits: 重传消息失败的最大次数。

protocol: 用户自定义的子协议，也就是说可以根据用户自己的业务需求而定义的私有协议，默认为空。

negotiated: 如果为 true，则会删除另一方数据通道的自动设置。这也意味着你可以通过自己的方式在另一侧创建具有相同 ID 的数据通道。

id: 当 negotiated 为 true 时，允许你提供自己的 ID 与 channel 进行绑定。

在上面的选项中，前三项是经常使用的，也是你要重点搞清楚的。不过需要特别说明的是，**maxRetransmits 与 maxPacketLifeTime 是互斥的**，也就是说这两者不能同时存在，只能二选一。

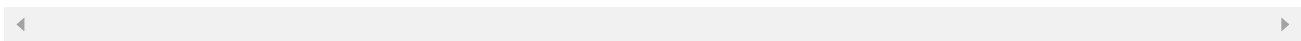
RTCDataChannel 的事件

RTCDataChannel 的事件处理与 WebSocket 的事件处理非常相似，RTCDataChannel 在打开、关闭、接收到消息以及出错时都会有接收到事件。

而当你在使用 RTCDataChannel 时，对上面所描述的这些事件都要进行处理，所以就形成了下面这样的代码模板：

 复制代码

```
1 ...
2 dc.onerror = (error)=> { // 出错
3     ...
4 };
5
6 dc.onopen = ()=> { // 打开
7     ...
8 };
9
10 dc.onclose = () => { // 关闭
11     ...
12 };
13
14 dc.onmessage = (event)=>{ // 收到消息
15     ...
16 };
```



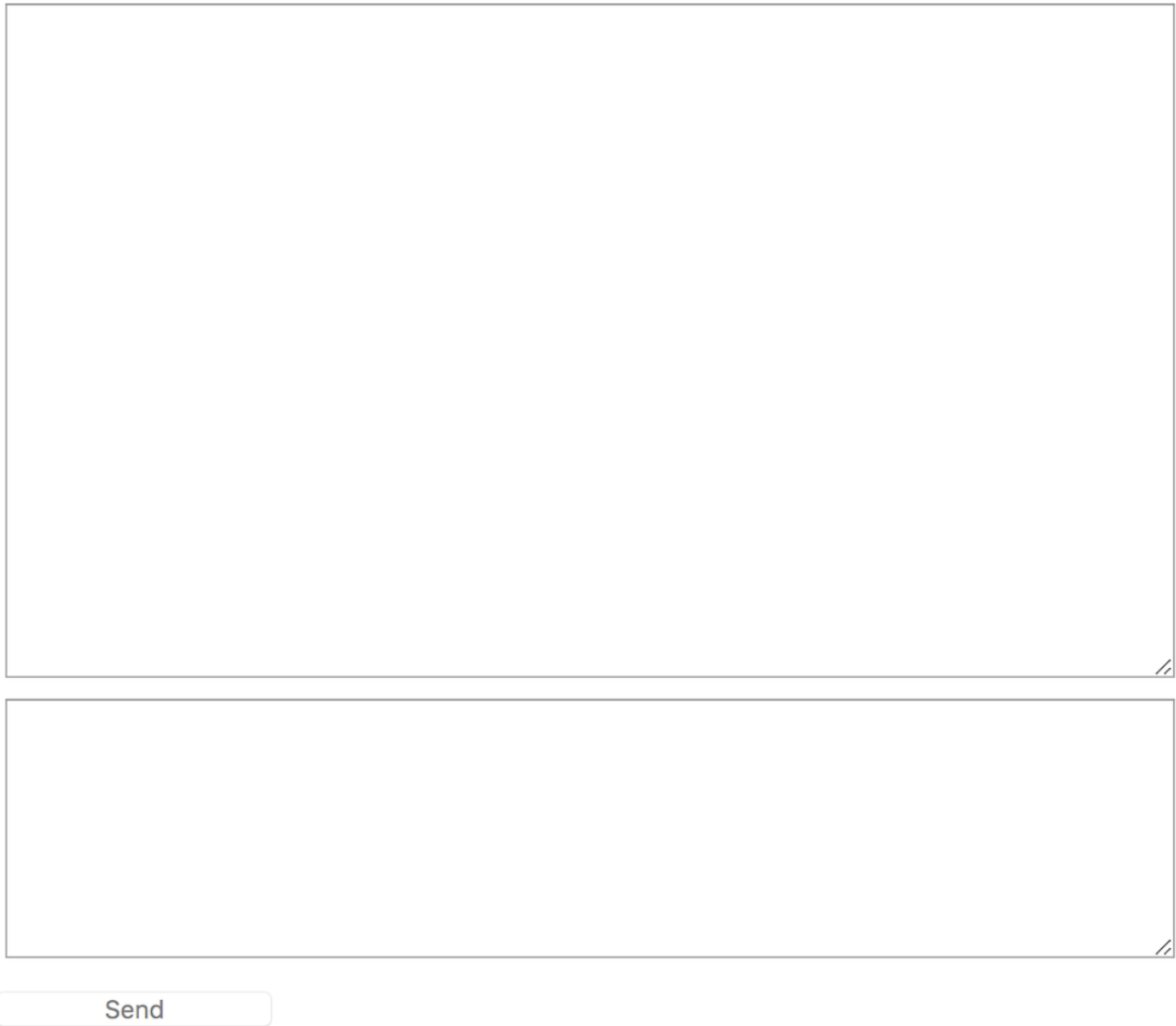
所以在使用 `RTCDataChannel` 对象时，你只要按上面的模板逐一实现其逻辑就好了，是不是很简单？

实时文字聊天

有了上面的知识，下面我们就来看一个具体的例子，看看如何通过 `RTCDataChannel` 对象实现一个实时文字聊天应用。

你可以想像这样一个场景，在两台不同的 PC 上（一个称为 A，另一个称为 B），用户打开浏览器，在页面上显示两个 `textarea`，一个作为文本输入框，另一个作为聊天记录的显示框。如下图所示：

Chat:



The diagram illustrates a chat interface. It consists of two large, empty rectangular text input boxes stacked vertically. Each box has a small double-slash icon in its bottom right corner. Below the second input box is a rounded rectangular button labeled "Send".

文本聊天图

当 A 向 B 发消息时，JavaScript 会从输入框中提取文本，然后通过 `RTCDataChannel` 发送出去。实际上，文本通过 `RTCDataChannel` 发送出去后，最终是经过 `RTCPeerConnection` 传送出去的。同理，B 向 A 发送文本数据时也是同样的流程。另一方面，当 B 收到 A 发送过来的文本数据后，也要通过 `RTCDataChannel` 对象来接收文本数据。

对于 `RTCDataChannel` 对象的创建主要有 **In-band 协商** 和 **Out-of-band 协商** 两种方式。

1. In-band 协商方式

此方式是默认方式。那什么是 In-band 协商方式呢？假设通信双方中的一方调用 **createDataChannel** 创建 **RTCDDataChannel** 对象时，将 options 参数中的 **negotiated** 字段设置为 false，则通信的另一方就可以通过它的 **RTCPeerConnection** 对象的 **ondatachannel** 事件来得到与对方通信的 **RTCDDataChannel** 对象了，这种方式就是 In-band 协商方式。

那 In-band 协商方式到底是如何工作的呢？下面我们就来详细描述一下。

A 端调用 **createDataChannel** 创建 **RTCDDataChannel** 对象。

A 端与 B 端交换 SDP，即进行媒体协商（offer/answer）。

媒体协商完成之后，双方连接就建立成功了。此时，A 端就可以向 B 端发送消息了。

当 B 端收到 A 端发的消息后，B 端的 **ondatachannel** 事件被触发，B 端的处理程序就可以从该事件的参数中获得与 A 端通信的 **RTCDDataChannel** 对象。需要注意的是，该对象与 A 端创建的 **RTCDDataChannel** 具有相同的属性。

此时双方的 **RTCDDataChannel** 对象就可以进行双向通信了。

该方法的**优势是 RTCDDataChannel 对象可以在需要时自动创建，不需要应用程序做额外的逻辑处理。**

2. Out-of-band 协商方式

RTCDDataChannel 对象还能使用 Out-of-band 协商方式创建，这种方式不再是一端调用 **createDataChannel**，另一端监听 **ondatachannel** 事件，从而实现双方的数据通信；而是两端都调用 **createDataChannel** 方法创建 **RTCDDataChannel** 对象，再通过 ID 绑定来实现双方的数据通信。具体步骤如下：

A 端调用 **createDataChannel({negotiated: true, id: 0})** 方法；

B 也调用 **createDataChannel({negotiated: true, id: 0})** 方法；

双方交换 SDP，即进行媒体协商（offer/answer）；

一旦双方连接建立起来，数据通道可以被立即使用，它们是通过 ID 进行匹配的（这里的 ID 就是上面 options 中指定的 ID，ID 号必须一致）。

这种方法的优点是，B 端不需等待有消息发送来再创建 `RTCDataChannel` 对象，所以双方发送数据时不用考虑顺序问题，即谁都可以先发数据，这是与 In-band 方式的最大不同，这也**使得应用代码变得简单**，因为你不需要处理 `ondatachannel` 事件了。

另外，需要注意的是，你选的 ID 不能是任意值。ID 值是从 0 开始计数的，也就是说你第一次创建的 `RTCDataChannel` 对象的 ID 是 0，第二个是 1，依次类推。所以这些 ID 只能与 WebRTC 实现协商的 SCTP 流数量一样，如果你使用的 ID 太大了，而又没有那么多的 SCTP 流的话，那么你的数据通道就不能正常工作了。


具体例子

了解完相关理论后，接下来我们就实践起来，结合具体例子将这些理论应用起来。

在本文的例子中，我们使用的是 In-band 协商方式来创建 `RTCDataChannel` 对象。下面我们就来一步一步操作，看看一个文本聊天应用是如何实现的。

1. 添加事件

为页面上的每个按钮添加 **onclick 事件**，具体如下面的示例代码所示：

 复制代码

```
1  var startButton = document.querySelector('button#startButton');
2  var callButton = document.querySelector('button#callButton');
3  var sendButton = document.querySelector('button#sendButton');
4  var closeButton = document.querySelector('button#closeButton');
5
6  startButton.onclick = connectServer; //createConnection;
7  callButton.onclick = call;
8  sendButton.onclick = sendData;
9  closeButton.onclick = closeDataChannels;
```

在这个段代码中定义了 4 个 button，其中 Start 按钮用于与信令服务器建立连接；Call 用于创建 `RTCDataChannel` 对象；Send 用于发送文本数据；Close 用于关闭连接释放资源。

2. 创建连接

用户在页面上点击 **Start** 按钮时，会调用 **connectServer** 方法。具体代码如下：


```
1  function connectServer(){
2
3      socket = io.connect(); // 与服务器建立连接
4
5      ...
6
7      socket.on('created', function(room) { // 第一个用户加入后收到的消息
8          createConnection();
9      });
10
11     socket.on('joined', function(room) { // 第二个用户加入后收到的消息
12         createConnection();
13     });
14
15     ...
16 }
```

从代码中可以看到，`connectServer` 函数首先调用 `io.connect()` 连接信令服务器，然后再根据信令服务器下发的消息做不同的处理。

需要注意的是，在本例中我们使用了 `socket.io` 库与信令服务器建立连接。


如果消息是 **created** 或 **joined**，则调用 `createConnection` 创建 `RTCPeerConnection`。其代码如下：

```
1  var servers = {'iceServers': [{
2      'urls': 'turn:youdomain:3478',
3      'credential': "passwd",
4      'username': "username"
5  }]}
6  };
7
8  pc = new RTCPeerConnection(servers, pcConstraint);
9  pc.onicecandidate = handleIceCandidate; // 收集候选者
10 pc.ondatachannel = onDataChannelAdded; // 当对接创建数据通道时会回调该方法。
```

通过上面的代码就将 `RTCPeerConnection` 对象创建好了。


3. 创建 RTCDataChannel

当用户点击 Call 按钮时，会创建 RTCDataChannel，并发送 offer。具体代码如下：

 复制代码

```
1  dc = pc.createDataChannel('sendDataChannel',
2      dataConstraint); // 一端主动创建 RTCDataChannel
3
4  ...
5  dc.onmessage = receivedMessage; // 当有文本数据来时，回调该函数。
6
7  pc.createOffer(setLocalAndSendMessage,
8      onCreateSessionDescriptionError); // 创建 offer，如果成功，则在 setLocalAndSendMessage
```

当其中一方创建了 RTCDataChannel 且通信双方完成了媒体协商、交换了 SDP 之后，另一端收到发送端的消息，ondatachannel 事件就会被触发。此时就会调用它的回调函数 onDataChannelAdded，通过 onDataChannelAdded 函数的参数 event 你就可以获取到另一端的 RTCDataChannel 对象了。具体如下所示：


 复制代码

```
1  function onDataChannelAdded(event) {
2      dc = event.channel;
3      dc.onmessage = receivedMessage;
4      ...
5  }
```

至此，双方就可以通过 RTCDataChannel 对象进行双向通信了。


4. 数据的发送与接收

数据的发送非常简单，当用户点击 Send 按钮后，文本数据就会通过 RTCDataChannel 传输到远端。其代码如下：

 复制代码

```
1  function sendData() {
2      var data = dataChannelSend.value;
3      dc.send(data);
4  }
```

而对于接收数据，则是通过 RTCDataChannel 的 onmessage 事件实现的。当该事件触发后，会调用 receivedMessage 方法。通过其参数就可以获取到对端发送的文本数据了。具体代码如下：

 复制代码

```
1 function receivedMessage(e) {
2     var msg = e.data;
3     if (msg) {
4         dataChannelReceive.value += "<- " + msg + "\n";
5     }
6 };
```

以上就是文本聊天的大体逻辑。具体的代码你可以到（文末的）[GitHub 链接](#)上获取。

小结

本文我们结合具体的例子——实时文字聊天，向你详细介绍了如何使用 RTCDataChannel 进行非音视频数据的传输。

RTCDataChannel 的创建有两种方式，一种是默认的 In-band 协商方式，另一种是 Out-of-band 协商方式。在本文例子的实践部分，我们主要应用的是第一种方式。但一般情况下我更推荐使用第二种方式，因为它更高效、更简洁。

另外，在使用 RTCDataChannel 时，还有两点你需要注意：

1. RTCDataChannel 对象的创建要在媒体协商（offer/answer）之前创建，否则 WebRTC 就会一直处于 connecting 状态，从而导致数据无法进行传输。
2. RTCDataChannel 对象是可以双向传输数据的，所以接收与发送使用一个 RTCDataChannel 对象即可，而不需要为发送和接收单独创建 RTCDataChannel 对象。

当然本文只是介绍了 RTCDataChannel 的“一种”具体应用，若你有兴趣还可以自行实践其他更有趣的实现。

思考时间

今天留给你的思考题是：SCTP 协议是运行在 TCP 协议之上还是 UDP 协议之上呢？

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。

参考

具体代码地址：https://github.com/avdance/webrtc_web/tree/master/19_chat/

 极客时间

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 18 | 如何使用Canvas绘制统计图表（下）？

下一篇 20 | 原来WebRTC还可以实时传输文件？

精选留言 (5)

写留言

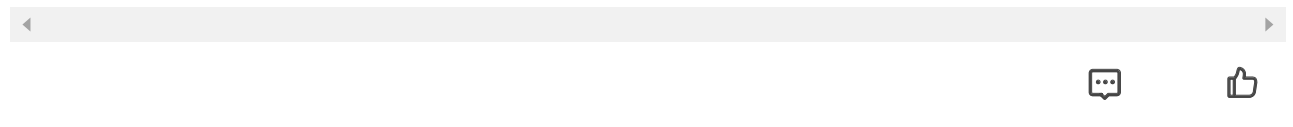


希望改名字不被发现的...

2019-08-29

同一个peerconnection, datachannel发送的文字能和视频stream保持同步吗

作者回复: 这是两个不同的通道无法同步

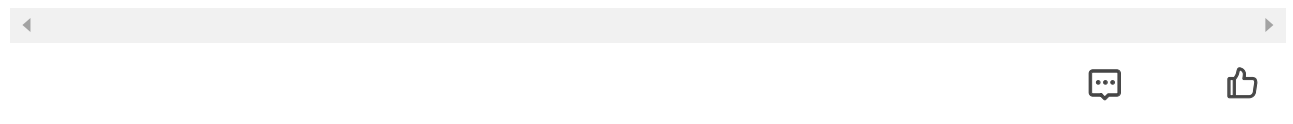


峰

2019-08-28

老师, 一个题外话, 这么多可作后端的语言, c++、python、go、java、c#该如何选择了?

作者回复: 信令服务器或者说业务服务器对性能要高不要的话使用 go/java 比较好。流媒体服务器由于对性能要求特别高, 所以要使用 C/C++ 开发

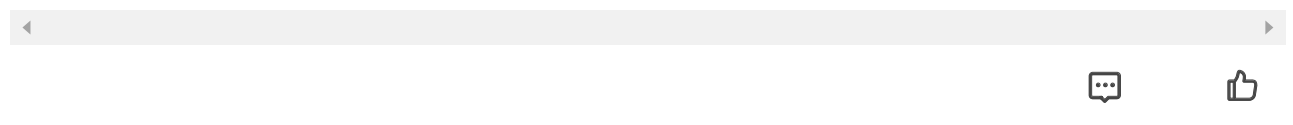


Geek_c1c44a

2019-08-27

老师您好, 请问zoom和直播技术相关吗? zoom可能使用什么协议呢?

作者回复: zoom 是 Zoom 公司开发的一款直播软件, 它底层也用的 UDP 协议, 他的老板袁征以前是 Webex 的高管。Webex 是第一家在美国上市的, 专门做音视频会议的公司。创建于 1996 年, 2000 年左右在美国上市, 后 2007 年被 cisco 公司收购。Webex 出走的人很多都创业做音视频相关的事情, 像国内的 声网都是 Webex 的人创建的。



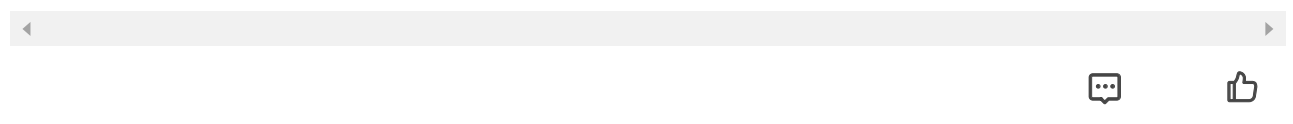
许童童

2019-08-27

SCTP 协议基于UDP, 自行实现TCP相关的功能。

展开 ∨

作者回复: 赞!



木木

2019-08-27

SCTP是运行在UDP上的，本质上是对UDP的封装，在应用层实现了有序性与可靠性的配置。

作者回复: 赞

