# [WebRTC 1.0: Between Browsers中文版](http://www.iwebrtc.com/blog/webrtc-1-0-real-time-communication-between-browsers-1/)

Tuyj整理自：[*iWebRTC about webrtc html5 extjs product*](http://www.iwebrtc.com/blog/)

目录

[WebRTC 1.0: Between Browsers中文版 1](#_Toc353091985)

[前言 2](#_Toc353091986)

[1. 介绍 3](#_Toc353091987)

[2. 一致性 3](#_Toc353091988)

[3. 术语 3](#_Toc353091989)

[4. Network Stream API 4](#_Toc353091990)

[4.1 介绍 4](#_Toc353091991)

[4.2 接口定义 5](#_Toc353091992)

[4.3 AudioMediaStreamTrack 7](#_Toc353091993)

[5. Peer-to-peer connections 8](#_Toc353091994)

[5.1 Connection对象创建 8](#_Toc353091995)

[5.2 连接算法 8](#_Toc353091996)

[5.3 算法补充 9](#_Toc353091997)

[5.4 Media流新增 9](#_Toc353091998)

[5.5 Media流移除 10](#_Toc353091999)

[5.6 Negotiationneeded事件 10](#_Toc353092000)

[5.7 RTCPeerConnection的属性和方法 10](#_Toc353092001)

[6. IANA 注册 22](#_Toc353092002)

[6.1 约束 23](#_Toc353092003)

[7. 简单示例 23](#_Toc353092004)

[8. 高级示例 25](#_Toc353092005)

[9. 浏览器间建立连接 25](#_Toc353092006)

[10. 浏览器与MCU连接 26](#_Toc353092007)

[11. Peer-to-peer Data API 27](#_Toc353092008)

[11.1 DataChannel 27](#_Toc353092009)

[11.2 示例 31](#_Toc353092010)

[12. 垃圾回收机制 32](#_Toc353092011)

[13. 事件定义 32](#_Toc353092012)

[13.1 RTCPeerConnectionIceEvent 32](#_Toc353092013)

[13.2 MediaStreamEvent 32](#_Toc353092014)

[13.3 DataChannelEvent 33](#_Toc353092015)

[14. 事件概要 34](#_Toc353092016)

[15. 安全注意事项 35](#_Toc353092017)

[16. 变更日志 35](#_Toc353092018)

[A. 致谢 37](#_Toc353092019)

[B. 参考 37](#_Toc353092020)

[B.1 Normative references 37](#_Toc353092021)

[B.2 Informative references 38](#_Toc353092022)

## 前言

**说明**

W3c 2012年8月21日的版本：  
This version可以在这里找到:  
<http://www.w3.org/TR/2012/WD-webrtc-20120821/>  
最近公开的版本:  
<http://www.w3.org/TR/webrtc/>

这几个editor挺有意思，一个来自爱立信、一个来自Voxeo（google了一下，是美国最大的viop提供商）、一个来自思科（华为干嘛不掺和一下？），最后一个来自Mazilla。就在我惊诧为什么木有google的人的时候，下面这句话说明了，初始的作者时google的Ian Hickson，而且苹果、Mozilla（应该是叫做谋智，感觉这个公司名称翻译得比谷歌有水平，可还是挺别扭的）还有Opera。而且读者可以使用、重新生成、并且创建这个文档的衍生的工作产品（这说明咱的翻译工作是合法的：）

**摘要**  
为了实现在两个联网的浏览器或者设备之间实现合适的实时协议来传输视频和语音，这篇文档在WebIDL中定义了一系列的ECMAscript的API。这个规格还在与IETF RTCWEB 小组开发的一个协议规以及Media Capture Task Force制定的访问本地媒体设备的API规范协同开发.(反正还在变就对了，google的chromium的代码实现在变，前一阵子微软好像也公开了一个他的webrct规范，好像有点故意与google的不兼容的那么个意思)

**文档状态**

这一节描述了这篇文档在发表时的状态。其它文档可能会取代这一篇.W3C的的技术报告和公开发表物的列表请看这里

这个文档还不全.可能会做大的修改，但是早期的实验是值得鼓励。（炮灰？）。所以这个文档不是用来实现的，API是以早期的工作为基础的。

从上一个版本的文档开始，融合Javascript Session Establishment Protocol(JSEP)到API中，以及将MediaStream和MediaStreamTrack对象融合到Media capture and Stream API中,并且在WebIDL做了大量的清理和整理工作。

注意最近有一篇提案是[另外一个不同的实现](http://lists.w3.org/Archives/Public/public-webrtc/2012Aug/0014.html)，提交到了工作组 ，工作组还没有决定哪一个提案会实质影响现在工作的方向。（这老外明显的不待见微软嘛，另外一个不同的实现不就是微软提交的实现的，呵呵，连名字都不提）

这个文档作为 Web Real-Time Communications工作组的工作稿件公开。这篇文档着力成为W3C的推荐(标准？）.如果你对这篇文档有意见，请发邮件到public-webrtc@w3.org，任何反馈都是欢迎的。

作为工作稿件（working draft）发布，并不意味着被W3C成员认可。这个文档可能在任何时候被更新、替换或者被其它的文档废弃.如果不是作为正在修改中的文档引用是不恰当的。

这篇文档以以为原则，用小组工作方式产生的。

W3C 维护着一个与小组交付物相关的公开的专利列表；那个页面也包含公开专利的步骤。任何个人知道那些包括专利条款的公开信息必须遵守第6节的W3C的专利条款。  
（以上这一段感觉翻译得怪怪的，所以将原文也附在下面）  
W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

## 介绍

这一节是非规范性的(non-normative).

这个规范中覆盖了html视频会议的多个方面：

* 表示一段多媒体流，这个多媒体流来自本地设备（视频摄像机、麦克风、web摄像头）或者一段用户事先录好的文件。
* 使用诸如ICE、STUN和TURN 的NAT穿透技术连接到远端节点。
* 发送本地生成的流到远端以及接受远端节点发送过来的流。
* 直接发送任意的数据到对端。

这篇文档定义了为实现这些特性的一系列的API。这个规格与正在IETE RTCWEB 小组开发的协议规范以及Media Capture Task Force开发的访问本地设备的API在协同开发。

## ****一致性****

这一节也是标记为非规范性的(non-normative).所有的创作指导、图表、示例以及规范都是非规范性的。

在这个规格中的关键字must, must not, required, should, should not, recommended, may, and optional 和 [RFC2119]中描述的一致.

这个规格定义的一致性标尺应用到一个单个产品:实现包含这些接口的用户代理。

用ECMAScrip来实作这个规格中定义的这些API必须和Web IDL 规格中定义的ECMAScript Bindings保持一致，因为这个规格使用了WEBIDL的规格和术语。

## ****术语****

EventHandler interface 表示 在[HTML5]定义的回调事件处理函数.

queue a task 和 fires a simple event 在 [HTML5]中定义.

event handlers 的条款和event handler event 类型在 [HTML5]中定义.

原文：

The [EventHandler](http://dev.w3.org/html5/spec/webappapis.html#eventhandler) interface represents a callback used for event handlers as defined in [[HTML5](http://www.w3.org/TR/webrtc/#bib-HTML5)].

The concepts [***queue a task***](http://dev.w3.org/html5/spec/webappapis.html#queue-a-task) and [***fires a simple event***](http://dev.w3.org/html5/spec/webappapis.html#fire-a-simple-event) are defined in [[HTML5](http://www.w3.org/TR/webrtc/#bib-HTML5)].

The terms [***event handlers***](http://dev.w3.org/html5/spec/webappapis.html#event-handlers) and [***event handler event types***](http://dev.w3.org/html5/spec/webappapis.html#event-handler-event-type) are defined in [[HTML5](http://www.w3.org/TR/webrtc/#bib-HTML5)]

## Network Stream API

[http:/http://www.w3.org/TR/webrtc/#network-stream-api](http://www.w3.org/TR/webrtc/#network-stream-api)

### [4.1 介绍](http://www.w3.org/TR/webrtc/#network-stream-api)

本节摘要：

* MediaStream 定义在[[GETUSERMEDIA](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]中
* MediaStream 表示一段音频和（或者）视频流数据
* 也可被扩展表示一段向远端节点发送的数据流或者从远端节点接受的数据流。
* 一个MediaStream 对象包含零个或者多个MediaStreamTrack 对象
* 对于接收者来说，从远端接收到的MediaStreamTrack是一一对应的。
* Peer定义为遵循webrtc规范的用户代理，可以理解为支持webrtc的浏览器或者设备。
* 通道(Channels) 是 MediaStream 规格中最小的单位.
* 为了传输，所有的通道的数据都需要一起被编码。由一个编解码器共同编码的通道 一定(must) 要在同一个MediaStreamTrack对象中，而且编解码器 应该（should） 可以编码, 或者丢弃track中所有的通道(Channels).
* 一个通过网络传输的MediaStream同样应用了输入输出的概念。由[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象创建的MediaStream会将从远端接收的数据当作input，把本地发到远端的数据(比如来自本地摄像头)当做output。
* 在 [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]中描述的复制 MediaStream 对象的概念在这里也适用。将不同的MediaStream对象上tracks 合并到一个新的MediaStream在某些特定的场景中也是非常有用的。应用场景例如：一个视频会议中，在本地的监视器显示本地的视频和以及播放麦克风的声音，但是只传输声音到远端。

[MediaStream 接口, 定义在 [](http://www.w3.org/TR/webrtc/#network-stream-api)[GETUSERMEDIA](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)] 规格中, 一般情况下表示一段音频和（或者）视频流数据. 一个 MediaStream 可以被扩展为表示一段向远端节点发送的数据流或者从远端节点接受的数据流。(比方说，不仅仅是本地摄像头). 在 MediaStream 对象上开启这个功能的扩展将在本文档中描述。

一个MediaStream 定义在 [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]的对象可以包含零个或者多个MediaStreamTrack 对象. 对于接受者来说一个发送到对端的MediaStreamTrack 对象会表现为一个且只有一个MediaStreamTrack . Peer是定义为一个支持这个规格的用户代理(User Agent可以理解为浏览器或者支持webrtc的设备）

通道(Channels) 是 MediaStream 规格中最小的单位. 为了传输的目的，比如作为RTP的负载类型，所有的通道（Channels）预期在一起编码.由一个编解码器共同编码的通道 一定 要在同一个MediaStreamTrack对象中 而且 编解码器 应该（should） 可以编码, 或者丢弃, 所有track中的通道(Channels).

The concepts of an input and output to 一个给定的的 MediaStream对象的输入输出的概念同样应用到了在网络上传输的 MediaStream 对象上. 一个 MediaStream 由[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)  (稍后描述)创建的对象会作为输入接收远端传过来的数据. 类似的, 一个  从本地源创建的MediaStream对象, 比如一个 [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]描述的摄像头，如果它正在被[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象使用的话，就会有一个表示传输到远端节点的输出.

在 [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]中描述的复制 MediaStream 对象的概念在这里也适用。 举例子来说，这个功能能够在这样的场景使用：这个一个视频会议中，在本地的监视器显示本地的视频和以及播放麦克风的声音，但是只传输声音到远端。(e.g. 对应用户所用的 “视频静音”的功能). 将不同的MediaStream对象上tracks 合并到一个新的MediaStream在某些特定的场景中也是非常有用的。

### 4.2 接口定义

注意：

在本小节, 我们仅仅阐述与  [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)相关方面。如果想要知道MediaStream 和 MediaStreamTrack使用方面的基本信息，请参考在[[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)] 文档中参考原始的对象定义。

#### 4.2.1 MediaStream

**4.2.1.1 label属性**

MediaStream中详细描述了label这个属性，label用以唯一标示一个流，这样当流通过[RTCPeerConnection](http://www.w3.org/TR/webrtc/#rtcpeerconnection)API发送到远端后还能被识别。

当[MediaStream](http://www.w3.org/TR/webrtc/#mediastream)被创建的时候，为了标示这个流，会用来自远端的信息初始化label属性。

注：MediaStream 对象的label属性对于源stream是唯一标示的，但并不是标示它不能被复制。比如，一个本地生成的stream从本地用户代理通过[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)发送到远端，然后远端用同样的方式发送回来，那么原始的用户代理就会多个stream有相同的label(一个本地生成，一个来自远端)

**4.2.1.2 MediaStream的事件**

**事件：新的 media track与一个存在的MediaStream关联。**

以下情况需要触发这个事件：

* 如果远端Peer在通过 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)连接的MediaStream 对象的track列表中新增一个 MediaStreamTrack对象, 这个过程被本地的用户代理观察到.
* 任何原因导致的add() [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]方法被 MediaStreamTrackList对象在本地调用
* 随着stream创建而添加tracks(例如stream用tracks初始化)

**具体操作：**

用户代理（user agent） must 必须运行一下的步骤:

1. 创建一个 MediaStreamTrack 对象来表示新的媒体组件.
2. 如果这个Track的[kind](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtrack-kind) 属性等于 “audio“, 将它加入 MediaStream 对象的[audioTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-audiotracks) MediaStreamTrackList中.
3. 如果这个Track的[kind](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtrack-kind) 属性等于”video“, 则将它加入 MediaStream 对象的 [videoTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-videotracks) MediaStreamTrackList中。
4. 在MediaStreamTrackList对象上触发一个叫做 [addtrack](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtracklist-addtrack) 的track事件，附带一个新创建的 track 作为参数.

**事件：一个存在的media track 也从MediaStream上剥离。**

以下情况需要触发这个事件：

* remove() [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)] 方法在本地的  MediaStreamTrackList对象上被调用
* 一个stream对象被销毁

**具体操作：**

 用户代理（user agent）必须执行 must 如下的步骤:

1. 准备移除媒体组件的MediaStreamTrack对象track.

（Let *track* be the MediaStreamTrack object representing the media component about to be removed.）

1. 从 MediaStreamTrackList对象移除track.
2. 在MediaStreamTrackList 对象上触发一个名字叫做 [removetrack](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtracklist-removetrack)的事件，将 track 变量作为参数 .

在网络实例中，持续不断(onended)的事件是来自 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection).

#### 4.2.2 MediaStreamTrack

**事件：*ended***

在一个非本地媒体源的情况中（例如一个rtp源，一般MediaStream 都是通过[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接收数据），MediaStreamTrack 对这个MediaStream 的引用总是牢靠的。

当一个track属于一个来自远端的MediaStream，而且远端不再发送数据，那么*必须在*track上出发一个*ended*事件，该事件在[[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]中有详细描述。

问题 1

ISSUE: 你怎么知道它什么时候停止？这像是一个SDP问题,不是一个媒体层的问题。

**属性：readyState**

* 源自 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)的MediaStream中的track一定 有readyState属性 [[*GETUSERMEDIA*](http://www.w3.org/TR/webrtc/#bib-GETUSERMEDIA)]， 在没有接收到媒体数据之前，这个属性将设置为[muted](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtrack-muted).(1)
* 另外,如果本地的用户代理disable了(正在发送的)MediaStream上的对应MediaStreamTrack， 那么对应的远端的MediaStreamTrack 的readyState 属性将会设置为[muted](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtrack-muted)。
* 当在[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)上触发了**addstream** 事件, 所有的MediaStream中产生的MediaStreamTrack将被设置为[muted](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtrack-muted)，直到从RTP 源中收到数据为止。

问题 2

ISSUE: 你如何知道何时他被disable了？ 这好像是个SDP的问题, 而不是媒体级的问题.

### 4.3 AudioMediaStreamTrack

注意：DTMF API有大量讨论列表而且可能会改动。

[AudioMediaStreamTrack](http://www.w3.org/TR/webrtc/#idl-def-AudioMediaStreamTrack)是 MediaStreamTrack的具体类，它只传输音频而且扩展了发送/接收DTMF codes的能力。

**WebIDL：**

interface **AudioMediaStreamTrack** : *MediaStreamTrack* {

readonly attribute boolean [canInsertDTMF](http://www.w3.org/TR/webrtc/#widl-AudioMediaStreamTrack-canInsertDTMF);

void [insertDTMF](http://www.w3.org/TR/webrtc/#widl-AudioMediaStreamTrack-insertDTMF-void-DOMString-tones-long-duration) (DOMString *tones*, optional long *duration*);

};

#### 4.3.1 属性

**canInsertDTMF  --** boolean, readonly

该属性*必须* 指明[AudioMediaStreamTrack](http://www.w3.org/TR/webrtc/#idl-def-AudioMediaStreamTrack) 是否有能力发送DTMF

#### 4.3.2 Methods

void [insertDTMF](http://www.w3.org/TR/webrtc/#widl-AudioMediaStreamTrack-insertDTMF-void-DOMString-tones-long-duration) (DOMString *tones*, optional long *duration*);

说明：

* 当 [AudioMediaStreamTrack](http://www.w3.org/TR/webrtc/#idl-def-AudioMediaStreamTrack) 对象的insertDTMF()的方法被调用时,用户代理*必须* 将[发送 DTMF tones]的任务放到队列中去。
* **tone** 参数会被当做一系列的字符处理的。DTMF tones由字符 0 到 9, A 到 D, #, 和 \*组成(不区分大小写)。这个字符,指明延迟2秒处理下一个tones中的字符。其它不识别的字符将被忽略。
* **duration**参数指明播放**tones**中的各个DTMF的duration ，单位为ms（毫秒）。 duration大于70小于6000。 duration缺省值是100。tones的间隔 必须至少为 50 ms 但是在此基础上越短越好.

问题 3

ISSUE: 非法的值是如何处理的?

* 如果insertDTMF调用时，这个对象上已经有产生DTMF的任务在运行, 现有的任务将被取消。So如果调用insertDTMF 并且传一个空的tones参数就可以取消所有正在发送的tones。

**编者注**: 我们需要在这个对象上增加一个回调函数，这个回调函数将在tones发送之后被调用。这个对应用来说是非常需要的，因为他们要知道什么时候可以发送新的tones而不会取消现在正在发送的tones.

**编者注**: 看来我们需要一个回调函数/事件来通知我们收到了tones。草案是作为音频在扬声器上播放，但是我不知道是否有用。

## 5. Peer-to-peer connections

[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)允许两个用户之间直接通讯，,浏览器到浏览器。通信通过一个信号通道来协调，这个信号通道没有指定方式，但是一般情况下是网页script 经由服务器来交互，比如**XMLHttpRequest**方式。

译者注：iwebrtc现有版本的实现没有通过XMLHttpRequest，而是通过NPAPI的插件实现了一个socket连接到服务器端，同时，这个连接还有可能会用来中转语音和视频的数据。[iwebrtc的详细描述可以看这里](http://www.tokbox.com/opentok/api/documentation/v2/reference)

注：这一章节中所提及的任务的源是网络任务源。

为了防止第四方通过网络嗅探来截获的信息，创建带外连接来欺骗其他客户端，configuration 信息 应该 始终用加密连接传输。

### 5.1 Connection对象创建

以下5.1-5.6节为原文5.1之前部分

* 调用new [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)(configuration ) 会创建一个新的 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象.
* configuration变量 包含了需要访问的 [[*STUN*](http://www.w3.org/TR/webrtc/#bib-STUN)] 和 [[*TURN*](http://www.w3.org/TR/webrtc/#bib-TURN)] 服务器.TURN server 和STUN server可以由多台服务器组成，角色也可以相互转变.
* RTCPeerConnection 对象还与 **ICE agent**, **RTCPeerConnection readiness state(就绪状态)**, 和ICE state有关. 当对象被创建时这些将会被初始化。

### 5.2 连接算法

当RTCPeerConnection() 构造函数被调用时, 用户代理（user agent一般指浏览器） 必须 运行以下的步骤。 这个算法有一个同步的部分，它将作为事件循环处理算法—event loop的一部分而运行。

1. 创建一个ICE Agent并将其作为RTCPeerConnection的[RTCPeerConnection  ICE Agent](http://www.w3.org/TR/webrtc/#rtcpeerconnection-ice-agent) ，然后通过configuration将STUN 和TURN服务器传给RTCPeerConnection。只要IceTransports约束一取消(not set none)，[[*ICE*](http://www.w3.org/TR/webrtc/#bib-ICE)] 就会处理搜集。这个时候，ICE Agent 不知道它需要多少个ICE 组件(因此**候选人**的数量也需要搜集)，但是它可以做一个合理的假设而且随着RTCPeerConnection对象获取了更多信息的时它可以随时调节组件的数量。
2. 设置RTCPeerConnection的readiness state 为 new .
3. 设置[RTCPeerConnection的ice state](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state) 为 new .
4. 设置[localStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localStreams) 属性的值为空的只读的MediaStream 数组。
5. 设置 [remoteStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams)属性的值为空的只读 MediaStream 数组。
6. 返回 RTCPeerConnection, 但继续异步执行这些步骤。
7. 等待一个稳当的状态。同步的部分包含该算法的剩下步骤。

### 5.3 算法补充

在RTCPeerConnection对象的生命周期中,如下的规则需要遵从:

* 如果 iceState 值是”new” 且IceTransports 约束没有被设置为 “none”,  必须 在队列中增加一个任务来开始采集ICE 地址且将iceState 变量值设为 “gathering”.
* 如果ICE Agent已经为所有MediaStreamTrack找到一对或者多对候选人，这样就形成了一条有效的连接, ICE state 将被修改为”connected”.
* 当ICE Agent检查完所有的候选人对后, 如果为一部分MediaStreamTrack找到至少一条连接,  iceState将被修改为”completed”，如果任何一个MediaStreamTrack都没有找到一条连接, iceState将被修改为 “failed”.

问题 4

ISSUE:这就意味着：如果可以通过ICE成功协商音频，但是视频协商失败， 那么 iceState == “completed”. 这个真的是想要的结果吗?

* 如果 iceState 等于 “connected”或者 “completed” 而且本地和远端的 session 描述被设置, RTCPeerConnection的状态将被置为 “active”.
* 如果 iceState 等于 “failed”, 队列里将插入一个任务来调用closel方法。

问题 5

ISSUE:: CJ – 这个我认为是错的。

### 5.4 Media流新增

用户代理协商编解码器的分辨率, 比特率,以及其它媒体参数.推荐 用最大的分辨率为视频流进行协商。对于那些将要被渲染render的视频流 (通过一个 video 元素),推荐 以将要渲染的分辨率进行协商。

注意：如果web应用开始传输数据时，用本地分辨率和对端协商，同时对端用相应的分辨率准备video元素，那么就不需要如下一般再次重新商定流了。

“components”这个词在这个上下文里指一个 RTP media 流而与 [[*ICE*](http://www.w3.org/TR/webrtc/#bib-ICE)] 中术语”component”毫无关系。

用户代理须按如下**步骤**创建MediaStream，用以表示即将接收的rtp媒体流：

1. 用[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)来接收这个媒体流.
2. 创建MediaStream对象来表示这个媒体流.
3. 为媒体流中的每个rtp media流执行如下步骤：

* 创建MediaStreamTrack 对象来表示rtp流. [[编辑的话:这里我们可以直接参考4.2.1.2吗？]]
* 如果 track的 [kind](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtrack-kind)属性为 “audio“,将它加到MediaStream的[audioTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-audiotracks) MediaStreamTrackList中。
* 如果track的 [kind](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtrack-kind)属性为 “video“,将它加到MediaStream的 [videoTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-videotracks) MediaStreamTrackList中。

注意:

* 创建MediaStream实例这一步会被[SDP 协商]或者是[随着指定媒体流的到来]而触发。
* 接收端MediaStreamTrackList中元素的顺序反映了发送端的顺序。可以通过指定sdp中的顺序来确保这个对应关系。

1. 在队列中创建一个任务以执行下列的子步骤：

* 如果connection的 RTCPeerConnection readiness state是CLOSED (3), 那么就放弃这些步骤.
* 将新创建的MediaStream 对象加入到connection的 [remoteStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams) 数组尾部。
* [触发一个stream的](http://www.w3.org/TR/webrtc/#fire-a-stream-event)名为 [addstream](http://www.w3.org/TR/webrtc/#event-mediastream-addstream)的事件， 并将新创建的 MediaStream对象作为参数传给它。

当已经用MediaStream对象来表示一个媒体流，为这媒体流中的rtp流协商好媒体参数，用户代理 必须  将rtp流与这个MediaStream 对象关联起来。

### 5.5 Media流移除

当一个[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象发现一个从远端节点过来的stream被移除时 , 用户代理必须 执行如下的步骤:

1. 将[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)和这个stream关联的变量移除.
2. 如果有MediaStream对象表示该媒体流，则将该对象移除，如果没有，则中止这些步骤。
3. 根据定义 ，stream 现在结束，因此执行下一步。
4. 在队列中增加一个任务来执行下列的子步骤:

* 如果变量RTCPeerConnection readiness state 等于closed (3), 则中止这些步骤。
* 从connection变量的[remoteStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams) 数组中移除 stream.
* 在connection 上[触发一个stream事件](http://www.w3.org/TR/webrtc/#fire-a-stream-event) 名字叫做 [removestream](http://www.w3.org/TR/webrtc/#event-mediastream-removestream)，并将该stream作为参数.

### 5.6 Negotiationneeded事件

* 如果浏览器中一些修改引起 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象需要初始化、协商新的会话描述符，那么将在 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)上触发一个[negotiationneeded](http://www.w3.org/TR/webrtc/#event-negotiationneeded)事件。
* 具体来说, 如果一个[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)正在占用使用（consuming） 一个 MediaStream对象，同时一个track被添加到这个MediaStream的 MediaStreamTrackList对象中(比如调用[add()](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtracklist-add)方法), [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象must 触发 “negotiationneeded” 事件。
* 同理，在这种情形下移除媒体rtp流也 *必须* 触发 “negotiationneeded”事件。

### 5.7 RTCPeerConnection的属性和方法

RTCPeerConnection的一般操作在这个文档中描述： [[*RTCWEB-JSEP*](http://www.w3.org/TR/webrtc/#bib-RTCWEB-JSEP)].

#### 5.7.1 枚举--RTCSdpType

枚举[RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription)实例的类型

enum RTCSdpType {

"offer",

"pranswer",

"answer"

};

**枚举类型描述**

offer 表明为[[SDP](http://www.w3.org/TR/webrtc/#bib-SDP)] offer.

pranswer 表明为[SDP] answer,但不是final answer。一般应用于SDP offer的应答或者对上次发送的pranswer的更新。

answer 表明为[SDP] answer,而且是offer-answer交互过程的完结。一般应用于SDP offer的应答或者对上次发送的pranswer的更新。

#### 5.7.2 类--RTCSessionDescription

RTCSessionDescription()构造函数有一个可选的“字典”类型的参数, descriptionInitDict, 它的内容是用来初始化新的[RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription)对象。如果对应的属性的key不在descriptionInitDict中，值就会被初始化为 null。如果没有给构造函数传递字典参数,那么所有的属性都会被初始化为null。这个类是可扩展的，不需要做实质性的改动、处理。

[Constructor (optional RTCSessionDescriptionInit descriptionInitDict)]

interface **RTCSessionDescription** {

attribute [**RTCSdpType**](http://www.w3.org/TR/webrtc/#idl-def-RTCSdpType)? [type](http://www.w3.org/TR/webrtc/#widl-RTCSessionDescription-type);

attribute DOMString? [sdp](http://www.w3.org/TR/webrtc/#widl-RTCSessionDescription-sdp);

stringifier [DOMString](http://www.w3.org/TR/webrtc/#widl-RTCSessionDescription-DOMString-stringifier) ();

};  
dictionary **RTCSessionDescriptionInit** {

[**RTCSdpType**](http://www.w3.org/TR/webrtc/#idl-def-RTCSdpType) [type](http://www.w3.org/TR/webrtc/#widl-RTCSessionDescriptionInit-type);

DOMString [sdp](http://www.w3.org/TR/webrtc/#widl-RTCSessionDescriptionInit-sdp);

};

##### 5.7.2.1 属性

**type** of type [RTCSdpType](http://www.w3.org/TR/webrtc/#idl-def-RTCSdpType) nullable sdp的类型

**sdp** of type DOMString nullable sdp内容

##### 5.7.2.2 方法

DOMString()

没有参数

返回类型:字符串( stringifier注：此处应该是json类型的字符串)

说明：返回json类型的(字符串)属性列表，包含type和sdp。

返回值格式(算法)：

({name:”value”[,name:”value”]})

( == U+0028 ) == U+0029

{ == U+007B } == U+007D

: == U+003A “ == U+0022

, == U+002C

##### 5.7.2.3 字典--RTCSessionDescriptionInit

成员：

sdp  类型是DOMString

type 类型是 [RTCSdpType](http://www.w3.org/TR/webrtc/#idl-def-RTCSdpType)

#### 5.7.3 回调--RTCSessionDescriptionCallback

callback **RTCSessionDescriptionCallback** = void ([**RTCSessionDescription**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) *sdp*);

#### 5.7.4 回调--RTCVoidCallback

callback **RTCVoidCallback** = void ();

#### 5.7.5 回调--RTCPeerConnectionErrorCallback

callback **RTCPeerConnectionErrorCallback** = void (DOMString *errorInformation*)

问题：

*errorInformation类型变成枚举怎样？*

#### 5.7.6 枚举--RTCPeerState

enum **RTCPeerState** {

"new",

"opening",

"active",

"closing",

"closed"

};

**枚举类型描述**

new 对象刚刚创建，而且还没有任何网络事件。

opening 用户代理正在尝试通过ICE agent 建立连接并等待设置本地和远端的SDP。

问题 7

我们需要在 “opening” 和”active”之间加上更多的状态吗?

active ICE Agent已经发现了一个连接而且本地和远端的SDP已经被设置好了。可以开始媒体流程了。

closing RTCPeerConnection对象结束了所有的媒体流，连接关闭中。

closed 连接已经关闭了。

#### 5.7.7 枚举--RTCIceState

注：这些状态还正在讨论过程中，有可能被修改。

enum **RTCIceState** {

"new",

"gathering",

"waiting",

"checking",

"connected",

"completed",

"failed",

"closed"

};

**枚举类型描述**

new RTCPeerConnection 对象刚刚被创建,没有任何网络事件。

gathering ICE Agent正在尝试收集地址。

waiting ICE Agent没有在收集任何地址，它在等待另外一端的candidates完成之后它才会开始检查。

checking ICE Agent正在检查candidate对，但是还没有找到一个(可用)连接。除了检查，它有可能还在搜集地址。

connected ICE Agent已经找到一条连接，但是还在检查其它的candidate对，看能否找到一条更好的连接。它也可能还在收集地址。

completed ICE Agent 已经完成搜集工作而且已经找到一条连接（对比上一个状态可以知道这是最好的一条连接）

failed ICE Agent完成检查所有的 candidate 对，但是没有找到一条连接。

closed ICE Agent已经关闭了而且不再响应STUN请求。

#### 5.7.8 类--RTCIceCandidate

RTCIceCandidate()，一个可选的字典参数，candidateInitDict,用来初始化新的[RTCIceCandidate](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate) 对象。如果一个字典中的key在candidateInitDict不存在,对应的属性将被初始化为空。如果没有将字典参数传给构造函数,所有的属性将初始化为空。这个类是可以为未来的携带的数据做扩充扩展而不需要执行任何实质性的处理。

[Constructor (optional RTCIceCandidateInit candidateInitDict)]

interface **RTCIceCandidate** {

attribute DOMString? [candidate](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-candidate);

attribute DOMString? [sdpMid](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-sdpMid);

attribute unsigned short? [sdpMLineIndex](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-sdpMLineIndex);

stringifier [DOMString](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-DOMString-stringifier) ();

};  
dictionary **RTCIceCandidateInit** {

DOMString [candidate](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidateInit-candidate);

DOMString [sdpMid](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidateInit-sdpMid);

unsigned short [sdpMLineIndex](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidateInit-sdpMLineIndex);

};

##### 5.7.8.1 属性

**candidate ：类型DOMString, 可能为空**

这个候选人属性定义在 [[*ICE*](http://www.w3.org/TR/webrtc/#bib-ICE)]的15.1 节.

**sdpMLineIndex ：类型unsigned short, 可能为空**

这个指定index (从0开始的) ，是SDP m-line对应的数组下标。

**sdpMid ：类型DOMString, 可以为空**

如果存在这个属性,它包含了 “media stream identification”的标识符，与候选人关联的m-line部分在 [RFC 3388]中有定义。

##### 5.7.8.2 方法

DOMString()

没有参数

返回类型: 字符串（stringifier）

说明：

stringify化的方法，定义参见5.7.2.2.

包含[candidate](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-candidate), [sdpMid](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-sdpMid), [sdpMLineIndex](http://www.w3.org/TR/webrtc/#widl-RTCIceCandidate-sdpMLineIndex) 作为元素的属性列表。

##### 5.7.8.3 字典--RTCIceCandidateInit

candidate**：类型是DOMString，**DOMString sdpMid

sdpMLineIndex**：类型是unsigned short**

sdpMid**：类型是DOMString，**无符号短整型的sdpMLineIndex

#### 5.7.9 字典—RTCIceServer

dictionary **RTCIceServer** {

DOMString [url](http://www.w3.org/TR/webrtc/#widl-RTCIceServer-url);

nullable DOMString [credential](http://www.w3.org/TR/webrtc/#widl-RTCIceServer-credential);

};

credential：可以为空的DOMString

如果内部数组的url元素是一个TURN URI,那么这个就是TURN server的凭证。

url ：类型是 DOMString

一个stun或者turn的 URI，定义在这两个文档中：[[*STUN-URI*](http://www.w3.org/TR/webrtc/#bib-STUN-URI)] 和 [[*TURN-URI*](http://www.w3.org/TR/webrtc/#bib-TURN-URI)]。

在多层NAT的网络拓扑图中，除了TURN Server之外，需要在每两层NAT之间设置一个STUN Server，用来减少网络延时。

一个RTCIceServer对象数组的例子是:

[ { url:"stun:stun.example.net"] } , { url:"turn:user@turn.example.org", credential:"myPassword"} ]

#### 5.7.10 字典—RTCConfiguration

dictionary **RTCConfiguration** {

[**RTCIceServer**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceServer)[] [iceServers](http://www.w3.org/TR/webrtc/#widl-RTCIceServer-iceServers);

};

iceServers**：**[RTCIceServer](http://www.w3.org/TR/webrtc/#idl-def-RTCIceServer)类型的数组

用于ICE的JS提供了STUN和TURN 服务器存放在该数组中。

#### 5.7.11 接口—RTCPeerConnection

typedef MediaStream[] MediaStreamArray;

[Constructor (RTCConfiguration configuration, optional MediaConstraints

constraints)]

interface **RTCPeerConnection** : *EventTarget*  {

void [createOffer](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-createOffer-void-RTCSessionDescriptionCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback-MediaConstraints-constraints) ([**RTCSessionDescriptionCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescriptionCallback) *successCallback*,

optional [**RTCPeerConnectionErrorCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*,

optional MediaConstraints *constraints*);

void [createAnswer](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-createAnswer-void-RTCSessionDescription-offer-RTCSessionDescriptionCallback-successCallback-RTCPeerConnectionErrorCallback--failureCallback---null-optional-MediaConstraints-constraints---null-optional-boolean-createProvisionalAnswer) ([**RTCSessionDescription**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) *offer*, [**RTCSessionDescriptionCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescriptionCallback) *successCallback*,

optional RTCPeerConnectionErrorCallback? failureCallback = *null*,

optional optional MediaConstraints constraints = *null*,

optional optional boolean createProvisionalAnswer = *false*);

void [setLocalDescription](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-setLocalDescription-void-RTCSessionDescription-description-RTCVoidCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback) ([**RTCSessionDescription**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) *description*,

optional [**RTCVoidCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCVoidCallback) *successCallback*,

optional [**RTCPeerConnectionErrorCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

void [setRemoteDescription](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-setRemoteDescription-void-RTCSessionDescription-description-RTCVoidCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback) ([**RTCSessionDescription**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) *description*,

optional [**RTCVoidCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCVoidCallback) *successCallback*,

optional [**RTCPeerConnectionErrorCallback**](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

void [updateIce](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-updateIce-void-RTCConfiguration--configuration---null-optional-MediaConstraints--constraints---null-boolean-restart---false) (optional RTCConfiguration? configuration = *null*,

optional optional MediaConstraints? constraints = *null*,

optional boolean restart = *false*);

[**DataChannel**](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) [createDataChannel](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-createDataChannel-DataChannel-DOMString-label-DataChannelInit-dataChannelDict) ([TreatNullAs=EmptyString] DOMString *label*, optional [**DataChannelInit**](http://www.w3.org/TR/webrtc/#idl-def-DataChannelInit) *dataChannelDict*);

void [addIceCandidate](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-addIceCandidate-void-RTCIceCandidate-candidate) ([**RTCIceCandidate**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate) *candidate*);

void [addStream](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-addStream-void-MediaStream-stream-MediaConstraints-constraints) (MediaStream *stream*, optional MediaConstraints *constraints*);

void [removeStream](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-removeStream-void-MediaStream-stream) (MediaStream *stream*);

void [close](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-close-void) ();

readonly attribute [**RTCIceState**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceState) [iceState](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-iceState);

readonly attribute [**MediaStreamArray**](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamArray) [localStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localStreams);

readonly attribute [**MediaStreamArray**](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamArray) [remoteStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams);

readonly attribute [**RTCSessionDescription**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) [remoteDescription](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteDescription);

readonly attribute [**RTCSessionDescription**](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) [localDescription](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localDescription);

readonly attribute [**RTCPeerState**](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerState) [readyState](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-readyState);

attribute EventHandler [ondatachannel](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-ondatachannel);

attribute EventHandler [onnegotationneeded](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onnegotationneeded);

attribute EventHandler [onicecandidate](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onicecandidate);

attribute EventHandler [onopen](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onopen);

attribute EventHandler [onstatechange](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onstatechange);

attribute EventHandler [onaddstream](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onaddstream);

attribute EventHandler [onremovestream](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onremovestream);

attribute EventHandler [onicechange](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-onicechange);

};

##### 5.7.11.1属性

<http://www.w3.org/TR/webrtc/#widl-RTCIceServer-url>

iceState**类型是**[RTCIceState](http://www.w3.org/TR/webrtc/#idl-def-RTCIceState)**, 只读**

必须 为[RTCPeerConnection的 ICE Agent](http://www.w3.org/TR/webrtc/#rtcpeerconnection-ice-agent)的 ICE state.

localDescription**的类型是**[RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription)**, 只读**

必须为最近传给[setLocalDescription()](http://www.w3.org/TR/webrtc/#dom-peerconnection-setlocaldescription)的参数,加上任何的从那时起ICE Agent生成的本机的candidates。如果本地sdp没有设置则返回空。

localStreams**类型是**[MediaStreamArray](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamArray)**, 只读**

返回一个包含本地媒体流的(live)数组(那些通过[addStream()](http://www.w3.org/TR/webrtc/#dom-peerconnection-addstream)加入的流媒体 ).

onaddstream**类型是 EventHandler**

这是一个事件处理器，事件处理器的类型是[addstream](http://www.w3.org/TR/webrtc/#event-mediastream-addstream),当远端节点增加一个媒体流时 实现了[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 接口的所有对象 必须 实现触发这个事件 。

ondatachannel**类型是 EventHandler**

这个事件处理器,类型是 [datachannel](http://www.w3.org/TR/webrtc/#event-rtcpeerconnection-datachannel) ,   所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 接口的必须支持.

onicecandidate**类型是 EventHandler**

这个事件处理器,事件处理器的类型是 [onicecandidate](http://www.w3.org/TR/webrtc/#event-icecandidate) , 所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接口的对象必须支持。当一个新的ICE candidate加入之前的offer或者answer中将会触发这个事件。

onicechange类型是 EventHandler

这个事件处理器,事件处理器的类型是  [icechange](http://www.w3.org/TR/webrtc/#event-icechange), 所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接口的对象必须支持. 当 iceState 变化时这个事件将触发。

onnegotationneeded**类型是 EventHandler**

这个事件处理器,事件处理器的类型是  [negotiationneeded](http://www.w3.org/TR/webrtc/#event-negotiationneeded) , 所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接口的对象必须支持。

onopen**of 类型是 EventHandler**

这个事件处理器,事件处理器的类型是 [open](http://www.w3.org/TR/webrtc/#event-mediastream-open) , 所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接口的对象必须支持。

onremovestream**of type EventHandler**

这个事件处理器,事件处理器的类型是 [removestream](http://www.w3.org/TR/webrtc/#event-mediastream-removestream), 所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接口的对象必须支持。当一个媒体流被移除时，这个事件将被触发。

onstatechange**of type EventHandler**

这个事件处理器,事件处理器的类型是 [statechange](http://www.w3.org/TR/webrtc/#event-mediastream-open) , 所有实现了 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)接口的对象必须支持。当 readyState属性被修改时，这个事件触发。

readyState**类型是**[RTCPeerState](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerState)**, 只读**

readyState 属性 必须返回 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [RTCPeerConnection readiness 状态](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state).

remoteDescription**类型是**[RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription)**, 只读**

remoteDescription 属性 必须 返回 最近传给[setRemoteDescription()](http://www.w3.org/TR/webrtc/#dom-peerconnection-setremotedescription)方法的[RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription)对象, 加上从那时起任何 通过 [addIceCandidate()](http://www.w3.org/TR/webrtc/#dom-peerconnection-addicecandidate)添加远端的 candidates.

如果没有设置过远端描述符，将会返回空。

remoteStreams**类型是**[MediaStreamArray](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamArray)**, 只读**

返回一个包含远端流活动(live)数组(那些远端添加的媒体流).

当[addstream](http://www.w3.org/TR/webrtc/#event-mediastream-addstream) 和 [removestream](http://www.w3.org/TR/webrtc/#event-mediastream-removestream) 事件被触发时数组将会更新.

##### 5.7.11.2方法

**addIceCandidate**

给ICE Agent提供了一个远端的 candidate, 这个candidate将会被加到远端的描述符中去。连接检查将在”IceTransports” 约束没有被设置为 “none”之前发送到新的candidates 去。 这个方法调用将造成ICE Agent的状态改变,而且如果因此创建了另外一条连接有，媒体状态也会改变 。

如果candidate格式不正确，那么TBD 异常（exception）会抛出。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **是否可选** | **描述** |
| candidate | [RTCIceCandidate](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate) | ✘ | ✘ |  |

Return type: void

**addStream**

给RTCPeerConnection添加一个流。

当addStream() 方法被调用时,用户代理（user agent）必须执行如下的步骤:

1. 如果[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [RTCPeerConnection readiness](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state)状态是closed (3), 则抛出 INVALID\_STATE\_ERR异常。
2. 如果这个流(stream )已经在 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [localStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localStreams) 对象中存在, 则放弃执行下面的步骤。
3. 将流(stream)加入 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象的 [localStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localStreams) 对象的尾部.
4. 分析应用程序提供的约束（ constraints),如果可能将它们应用到MediaStream。注意 – 这里需要处理抛出的异常。
5. 触发 negotiationneeded 事件.

问题 9

ISSUE:如果 RTCPeerConnection是一个新的对象，还要触发事件吗?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **是否可选** | **描述** |
| 流(stream) | MediaStream | ✘ | ✘ |  |
| 约束(constraints) | MediaConstraints | ✘ | ✔ |  |

返回值类型: void

**close**

当 close()方法被调用时,用户代理（ user agent）必须执行如下的步骤:

1. 如果[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [RTCPeerConnection readiness](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state)状态是 closed (3),抛出一个 INVALID\_STATE\_ERR 异常。
2. 销毁 [RTCPeerConnection ICE Agent](http://www.w3.org/TR/webrtc/#rtcpeerconnection-ice-agent), 快速结束任何活跃的 ICE 处理过程和任何活跃的流（ streaming), 且释放相关的资源 (比如. TURN 资源).
3. 设置对象的 [RTCPeerConnection readiness state](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state) to closed (3).

没有参数

返回值类型: void

**createAnswer**

 createAnswer方法根据传入的configuration 参数为会话（session)产生一个 [[*SDP*](http://www.w3.org/TR/webrtc/#bib-SDP)] answer 消息，此消息与offer的参数保持兼容。和createOffer类似,返回的 blob包含了管理了本地 MediaStreams 的RTCPeerConnection对象,协商过的编解码器/RTP/RTCP选项,和任何ICE Agent搜集的candidates 。可能会传入约束参数来提供额外的控制信息来生成应答（answer）.

作为一个应答（answer）,生成的SDP会包含指定的配置(configuration）,和offer消息一起, 指定媒体层（media plane）应该如果建立。SDP的生成必须遵循如下的适当的过程来生成一个应答(answer)消息或者 临时应答(provisional answer)。

如果setLocalDescription方法在sucessCallback函数中被调用的话，不需要生成错误消息 ，createAnswer 方法生成的会话描述符（Session descriptions）必须立即作为可用状态提供给setLocalDescription方法。类似于createOffer方法,返回的描述符（description）应该反映当前系统的状态。会话描述符 必须 remain 通过setLocalDescription 方法保持可用且不产生任何错误，直到successCallback function结束。获取ICE 用户名和密码需要调用这个方法。只有当 createProvisionalOffer 标志为真时会创建在 [[*RTCWEB-JSEP*](http://www.w3.org/TR/webrtc/#bib-RTCWEB-JSEP)]中描述的临时的offer（ Provisional offers）。

当系统不能为指定的offer消息生成应答消息（answer)时，failureCallback 函数将被触发。

如果约束参数格式不正确，TBD异常 将被抛出。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **可选** | **描述** |
| offer | [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) | ✘ | ✘ |  |
| successCallback | [RTCSessionDescriptionCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescriptionCallback) | ✘ | ✘ |  |
| null | RTCPeerConnectionErrorCallback? failureCallback = | ✘ | ✔ |  |
| null | optional MediaConstraints constraints = | ✘ | ✔ |  |
| false | optional boolean createProvisionalAnswer = | ✘ | ✔ |  |

Return type: void

**createDataChannel**

    用指定的标签（label)创建一个新的 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象。  [DataChannelInit](http://www.w3.org/TR/webrtc/#idl-def-DataChannelInit) 字典参数可以用来配置底层的传输通道，比如数据的可靠性。如果通道创建设置成功的话，一个对应的 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)对象将在对端的节点部署。

当调用 createDataChannel() 方法时, 用户代理（user agent）必须执行如下的步骤：

1. 如果[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [RTCPeerConnection readiness 状态](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state)是closed (3), 抛出一个 INVALID\_STATE\_ERR 异常。
2. 让channel 成为一个新创建的 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象。
3. 用第一个参数来初始化channel的[label](http://www.w3.org/TR/webrtc/#dom-datachannel-label) 属性。
4. 将channel 的[reliable](http://www.w3.org/TR/webrtc/#dom-datachannel-reliable)属性设置为true。
5. 如果第二个参数存在而且它包含一个 [reliable](http://www.w3.org/TR/webrtc/#widl-DataChannelInit-reliable) 的字典成员, 那么就将channel的 [reliable](http://www.w3.org/TR/webrtc/#dom-datachannel-reliable) 属性设置为对应的字典成员的值。
6. 返回channel然后在后台继续执行余下的步骤。
7. 创建通道 channel的关联的底层数据传输通道（[underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport)）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **是否可选** | **描述** |
| label | DOMString | ✘ | ✘ |  |
| dataChannelDict | [DataChannelInit](http://www.w3.org/TR/webrtc/#idl-def-DataChannelInit) | ✘ | ✔ |  |

返回类型: [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)

**createOffer**

     createOffer 方法生成一个SDP数据块,包含了 RFC 3264 协议中的offer消息以及会话(session)的配置，包括关联到这个 RTCPeerConnection对象的本地媒体流(local MediaStreams) 描述符(description), 这个实现支持的编解码器/RTP/RTCP选项，以及ICE Agent搜集的candidates。约束参数可能为offer消息的生成提供了附加的控制信息。

作为一个offer消息，生成的SDP将会包含会话支持的完整的功能。(和answer消息不同,它只包含指定的协商过的子集）;对于每行SDP消息，SDP的生成offer消息的过程必须正确。在会话建立之后createOffer方法被调用的事件中，, createOffer会生成一个offer消息与当前的会话(session)兼容, 自从上次完整的offer-answser消息交换过程完成后作用在这个会话上变化将会合并进来，比如增加或者减少流(stream)。如果没有任何变化，offer消息将会包含当前本地的描述符的功能（capabilities ）以及任何附加的可用在更新的offer消息协商的功能。

createOffer 生成的会话(Session)描述符must必须 be立即置为可用提供给setLocalDescription使用，并保证在setLocalDiscription方法被 successCallback 函数调用时不出错。如果系统资源有限(比如：有限的解码器数量）, createOffer需要返回一个能够反映当前系统状态的 offer消息, 因此当setLocalDescription获取那些系统资源能够成功。 直到successCallback函数结束时会话（session）描述符must必须 remain为setLocalDescription保持可用不出错。调用这个方法需要获取 ICE的用户名和密码。

failureCallback 方法将在系统不能在给定的RTCPeerConnection状态下生成正确的offer消息时被调用。

如果约束参数的格式不正确，一个TBD 异常将被抛出。

To Do: Discuss privacy aspects of this from a finger printing point of view – it’s probably around as bad as access to a canvas :-) （这个To Do暂时保留不翻译，因为感觉不太重要，第一眼也看得太明白）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **是否可选** | **描述** |
| successCallback | [RTCSessionDescriptionCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescriptionCallback) | ✘ | ✘ |  |
| failureCallback | [RTCPeerConnectionErrorCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |
| constraints | MediaConstraints | ✘ | ✔ |  |

返回值类型: void

**removeStream**

    在RTCPeerConnection 对象中从LocalStream 数组移除指定流(stream )并触发negotiationneeded事件。

当另外一个节点（对端）用这样的方式停止发送流(stream),一个[removestream](http://www.w3.org/TR/webrtc/#event-mediastream-removestream) 事件将在 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象上触发。

当 removeStream() method 方法被触发时，用户代理（user agent）必须执行如下的步骤:

1. 如果[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [RTCPeerConnection readiness](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state)的状态是closed (3), 抛出一个 INVALID\_STATE\_ERR 异常。
2. 如果流(stream) 不存在于 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [localStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localStreams) 对象中, 那么放弃下面的步骤。. TODO:需要在这里抛出异常吗?
3. 从[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的 [localStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-localStreams) 对象中移除此流（stream)。
4. 触发negotiationneeded事件。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **是否可选** | **描述** |
| 流 | MediaStream | ✘ | ✘ |  |

返回值类型: void

**setLocalDescription**

setLocalDescription() method 方法命令 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)将传递的 [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) 参数作为本地描述符。

这个API修改本地媒体状态(local media state),为了能成功处理这样的场景：应用程序想要从一种媒体格式转换到另外一种不兼容的格式。 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 必须能够同时支持使用老的和新的本地描述符(比如支持在两者中存在的编解码器)一直到收到一个最终的应答消息(final answer),此时[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)能够完全采纳新的本地描述符（local description）,如果对端不同意修改则回滚（ roll back )到老的描述符。

问题 8

ISSUE: 如何指定回滚?

To Do: 指定SDP的那一部分在createOffer和setLocalDescription之间可以修改。

修改到媒体传输状态会引起最终的应答(final answer)将会成功应用。 [localDescription](http://www.w3.org/TR/webrtc/#dom-peerconnection-localdescription) 必须 return 返回之前的描述符直到新的描述符成功应用。

如果 [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) 是合法的但是没能应用到媒体层( cannot be applied at the media layer),那么failureCallback 将被触发，比如没有足够的资源应用SDP。当失败发生时而新的描述符部分已经应用，用户代理( user agent) 必须 根据需要回滚 。

如果SDP的内容非法，一个 TBD 异常将被抛出。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **可选** | **描述** |
| description | [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) | ✘ | ✘ |  |
| successCallback | [RTCVoidCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCVoidCallback) | ✘ | ✔ |  |
| failureCallback | [RTCPeerConnectionErrorCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |

返回值类型: void

**setRemoteDescription**

    setRemoteDescription() 方法指令[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象应用传递的 [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) 作为远端offer或者answer消息。这个API修改本地媒体状态(local media state)

修改到媒体传输(media transmission)将会造成最终的应答( final answer)成功设置。[remoteDescription](http://www.w3.org/TR/webrtc/#dom-peerconnection-remotedescription) must必须 返回之前的描述符直到新的描述符被成功应用。

如果 [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) 是合法的但是没能应用到媒体层( cannot be applied at the media layer),那么failureCallback 将被触发，比如没有足够的资源应用SDP。当失败发生时而新的描述符部分已经应用，用户代理( user agent) 必须 根据需要回滚 。

如果SDP的内容非法，一个 TBD 异常将被抛出。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **可选** | **描述** |
| description | [RTCSessionDescription](http://www.w3.org/TR/webrtc/#idl-def-RTCSessionDescription) | ✘ | ✘ |  |
| successCallback | [RTCVoidCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCVoidCallback) | ✘ | ✔ |  |
| failureCallback | [RTCPeerConnectionErrorCallback](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |

返回值类型: void

**updateIce**

    updateIce方法重启或者更新ICE Agent搜集本地candidate和pinging远端candidate的流程。如果有一个固定的名为”IceTransports”约束，它将会控制ICE 引擎如何工作。这个可以对被调用者用来限制使用TURN candidates来防止泄露本地信息优先于接受调用。

这个调用可能导致 ICE Agent状态变化, 如果它导致连接被建立的话，就可能导致需要修改媒体状态。

如果restart这个参数被设置为true，ICE状态机( state machine)将丢弃它搜集的所有candidates,重新为主机candidate分配新的端口，如果没有之前的ICE 会话(session)然后重启ICE and restarts ICE。当有些事情已经发生严重错误时，应用程序可以用这个方法来重置所有的ICE协商。

如果参数格式不正确，一个TBD异常将被抛出。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否可以为空** | **是否可选** | **描述** |
| null | RTCConfiguration? configuration = | ✘ | ✔ |  |
| null | optional MediaConstraints? constraints = | ✘ | ✔ |  |
| false | boolean restart = | ✘ | ✔ |  |

返回值类型: void

## 6. IANA 注册

IANA 要求按照 [[*RTCWEB-CONSTRAINTS*](http://www.w3.org/TR/webrtc/#bib-RTCWEB-CONSTRAINTS)]所述在 [Constraints Section](http://www.w3.org/TR/webrtc/#sec-constraints) 中注册约束。

### 6.1 约束

TOOD:当约束的草稿进一步修改时，需要修改这些约束的命名和声明来符合要求。这里的命名现在好像是不太正确，但是是作为占位符来使用的。

问题 10

ISSUE: 有多种方法来增加约束，如何来协调他们的兼容性?

下面这些新定义的约束用以配合RTCPeerConnection对象的使用：

**OfferToReceiveVideo**

这是一个枚举类型的约束，它可以取值”true” and “false”。对传输videostream的RTCPeerConnection来说，该值非强制的默认为true。

在某些情况下, RTCPeerConnection对象可能希望只接收video，但不发送。这时这个RTCPeerConnection（根据这个约束值）知道它是否应该发信号通知远端它是否愿意接收video。这个约束允许应用程序来在创建一个offer的时候指定它的“是否接收video”的设置(preferences)。

**OfferToReceiveAudio**

这是一个枚举类型的约束，它可以取值”true” and “false”。对传输audiostream的RTCPeerConnection来说，该值非强制的默认为true。

在某些情况下, RTCPeerConnection对象可能希望只接收audio，但不发送。这时这个RTCPeerConnection（根据这个约束值）知道它是否应该发信号通知远端它是否愿意接收audio。这个约束允许应用程序来在创建一个offer的时候指定它的“是否接收audio”的设置(preferences)。

**VoiceActivityDetection**

这是一个枚举类型的约束，它可以取值”true” and “false”。 缺省值是非强制性的 “true”。

许多编解码器和系统在检测出“静止(silence)后,可能会改变操作，比如不再发送任何(audio)媒体数据。在许多情况下，比如当处理除了讲话声音以外的voice或者处理紧急呼叫时,就需要能够关闭先前的操作(可能指发送媒体流)。这个约束允许应用程序设置该值来决定是否需要该功能生效。

**IceTransports**

是一个枚举类型的约束，它可以取值 “none”, “relay”, and “all”。缺省值是非强制性的 “all”。

这个约束指定ICE引擎允许用哪些 candidates 。 “none”取值意味着ICE 引擎 *不能* 发送或者接收任何包。 “relay”取值指明ICE 引擎 *must*只可以用媒体中继(转播)candidates，比如通过TURN server中转的candidates ，这个可以用来在某些情况下减少IP地址的泄露 。 “all”取值则表示所有取值都可以用。

TODO items – 需要在IANA注册。

## 7. 简单示例

当两个节点将要相互建立连接时，它们都得进行如下步骤。可以通过STUN/TURN服务器配置来获取各端的公网ip或者建立NAT穿透。它们也必须通过信号通道（这个信号通道是指通过web server中转的连接，比如通过ajax）按照相同的（带外）传输格式经行交互，以便建立起初的通信连接。

EXAMPLE 1

var signalingChannel = createSignalingChannel();

var pc;

var configuration = ...;

// run start(true) to initiate a call

function start(isCaller) {

pc = new RTCPeerConnection(configuration);

// send any ice candidates to the other peer

pc.onicecandidate = function (evt) {

signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));

};

// once remote stream arrives, show it in the remote video element

pc.onaddstream = function (evt) {

remoteView.src = URL.createObjectURL(evt.stream);

};

// get the local stream, show it in the local video element and send it

navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {

selfView.src = URL.createObjectURL(stream);

pc.addStream(stream);

if (isCaller)

pc.createOffer(gotDescription);

else

pc.createAnswer(pc.remoteDescription, gotDescription);

function gotDescription(desc) {

pc.setLocalDescription(desc);

signalingChannel.send(JSON.stringify({ "sdp": desc }));

}

});

}

signalingChannel.onmessage = function (evt) {

if (!pc)

start(false);

var signal = JSON.parse(evt.data);

if (signal.sdp)

pc.setRemoteDescription(new RTCSessionDescription(signal.sdp));

else

pc.addIceCandidate(new RTCIceCandidate(signal.candidate));

};

## 8. 高级示例

这个例子展示更加复杂的功能.

例子 2

TODO

## 9. 浏览器间建立连接

注意

编者注: 这个例子中的流程还需要讨论而且好像在很多方面有问题。

这里展示了一个例子，描述了一种可能方法来建立两个浏览器之间的呼叫。这个例子没有展示获得本地媒体流的过程和每个事件回调函数，只是显示了关键的事件和信息。



图9.1

如下的流程展示了更完全的过程中产生的回调和事件：



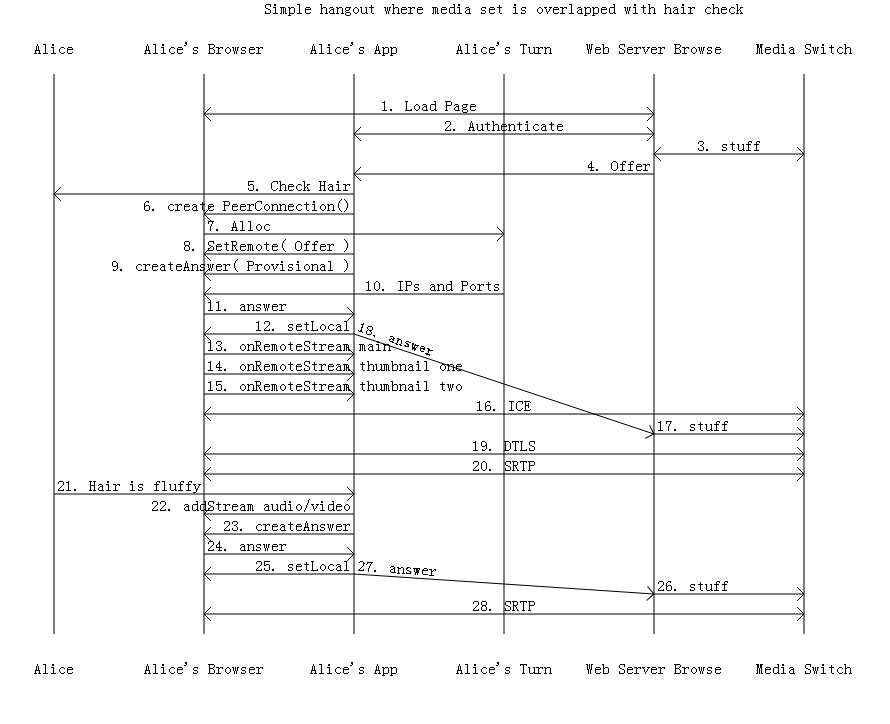
图9.2

## 10. 浏览器与MCU连接

注意

编者注:这个流程例子需要在list中讨论而且可能在很多方面有问题。

这里展示的例子，说明了一个可能的在中心会议服务器和浏览器之间的呼叫流程。这里并没有展示每一个触发的回调函数而是只显示了关键的事件和信息。



## 11. Peer-to-peer Data API

Peer-to-peer Data API让 web应用程序发送和接收一般的(generic)基于应用的点对点的数据。

问题 11: 更多开放性的问题

* 数据通道发送信号(signaling)的问题(通过SDP以及应用程序指定的信号通道发送信号或者第一条通道通过SDP，之后的通道通过内部信号传输）
* 对于实际的一些接口，哪些能够共享Websocket的API？

### 11.1 DataChannel

#### 11.1.0-0 通道创建与销毁

* 数据通道( [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel))接口表示一个在两个节点之间的双向的数据通道 。一个数据通道( [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel))是通过 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象的工厂方法来创建的。如果通道创建成功，对应的 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象也将会在对端节点部署。

每个 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)对象都有一个对应的**底层数据传输工具** (**underlying data transport**)来传输实际数据到对端 。 **底层数据传输工具** （ [underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport)）的传输属性,比如可靠传输模式,是由主动创建通道的一端来设置的。另外一端无法修改数据通道的传输属性( data channel)。实际的物理层传输协议超出了本文的范围。

* 通过 [createDataChannel()](http://www.w3.org/TR/webrtc/#dom-peerconnection-createdatachannel)创建的[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)对象*必须* 初始化为connecting 状态。如果[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象的底层数据传输工具([underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport))成功创建，用户代理(user agent)必须宣告(announce)[DataChannel对象的状态是open](http://www.w3.org/TR/webrtc/#announce-datachannel-open)。

**(主动创建方)触发任务：**

当用户代理（user agent）宣告一个DataChannel状态是 **open**,就*必须*将一个任务加入队列来执行如下步骤:

1. 如果相关联的 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 的 [RTCPeerConnection readiness 状态](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state) 是 closed (3), 放弃执行下面的步骤。
2. 宣告该通道为[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)。
3. 将 channel的 [readyState](http://www.w3.org/TR/webrtc/#dom-datachannel-readystate) 属性置为 open。
4. 在这个channel上触发一个名叫[open](http://www.w3.org/TR/webrtc/#event-datachannel-open)的简单事件。

**(被动创建方)触发任务:**

当一个底层数据传输工具( [underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport))成功建立，用户代理( user agent），必须将一个任务放到队列里面来执行如下的步骤:

1. 如果相关联的 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 的 [RTCPeerConnection readiness 状态](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state) 是 closed (3), 放弃执行下面的步骤。
2. 随着底层数据传输的建立，会从对端接收到一扎(bundle)的信息，将这些信息用键值对的形式来设置configuration。
3. 新创建[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象。
4. 初始化这个channel的[label](http://www.w3.org/TR/webrtc/#dom-datachannel-label)属性，值与configuration对应。
5. 初始化这个channel的 [reliable](http://www.w3.org/TR/webrtc/#dom-datachannel-reliable)属性为true。
6. 如果configuration 中有key为”reliable“的值, 将channel的 [reliable](http://www.w3.org/TR/webrtc/#dom-datachannel-reliable) 属性设置为对应的值。
7. 将channel的 [readyState](http://www.w3.org/TR/webrtc/#dom-datachannel-readystate) 的属性设置为 open。
8. 在[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection)对象上触发一个名叫[datachannel](http://www.w3.org/TR/webrtc/#event-rtcpeerconnection-datachannel) 的datachannel  event 事件，并将这个channel 作为参数。

**通道销毁:**

当 **销毁**[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)**的底层数据传输工具**[***underlying data transport***](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport) 的过程开始以后, 用户代理(user agent) 必须 执行如下的步骤:

1. 如果相关联的 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 的 [RTCPeerConnection readiness 状态](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state) 是 closed (3), 放弃执行下面的步骤：
2. 设置通道类型为[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)。
3. 如果channel的[readyState属性](http://www.w3.org/TR/webrtc/#dom-datachannel-readystate) 是 closing 或者 closed, 则放弃执行下面的步骤.
4. 将 channel的 [readyState](http://www.w3.org/TR/webrtc/#dom-datachannel-readystate) 属性设置为closing.
5. 将一个任务加入队列来执行下面的步骤:
   * 关闭channel的[底层数据传输通道(underlying data transport)](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport).

注意

当这个通道关闭，数据传输协议会引发一些特定的事情：比如缓冲的数据。

The data transport protocol will specify what happens to, e.g. buffered data, when the data transport is closed.

* + 将 channel的 [readyState](http://www.w3.org/TR/webrtc/#dom-datachannel-readystate) 属性设置为closed (3).
  + 在channel上触发一个 [close](http://www.w3.org/TR/webrtc/#event-datachannel-close) 的简单事件

#### 11.1.0-1 接口定义

interface **DataChannel** {

readonly attribute DOMString [label](http://www.w3.org/TR/webrtc/#widl-DataChannel-label);

readonly attribute boolean [reliable](http://www.w3.org/TR/webrtc/#widl-DataChannel-reliable);

readonly attribute [**DataChannelState**](http://www.w3.org/TR/webrtc/#idl-def-DataChannelState) [readyState](http://www.w3.org/TR/webrtc/#widl-DataChannel-readyState);

readonly attribute unsigned long [bufferedAmount](http://www.w3.org/TR/webrtc/#widl-DataChannel-bufferedAmount);

attribute EventHandler [onopen](http://www.w3.org/TR/webrtc/#widl-DataChannel-onopen);

attribute EventHandler [onerror](http://www.w3.org/TR/webrtc/#widl-DataChannel-onerror);

attribute EventHandler [onclose](http://www.w3.org/TR/webrtc/#widl-DataChannel-onclose);

attribute EventHandler [onmessage](http://www.w3.org/TR/webrtc/#widl-DataChannel-onmessage);

attribute DOMString [binaryType](http://www.w3.org/TR/webrtc/#widl-DataChannel-binaryType);

void [close](http://www.w3.org/TR/webrtc/#widl-DataChannel-close-void) ();

void [send](http://www.w3.org/TR/webrtc/#widl-DataChannel-send-void-DOMString-data) (DOMString *data*);

void [send](http://www.w3.org/TR/webrtc/#widl-DataChannel-send-void-ArrayBuffer-data) (ArrayBuffer *data*);

void [send](http://www.w3.org/TR/webrtc/#widl-DataChannel-send-void-Blob-data) (Blob *data*);

};

#### 11.1.1 属性

**binaryType 类型是 DOMString**

注意：

FIXME: 与 WebSocket API的行为保持一致

**bufferedAmount 类型是 unsigned long, 只读**

注意：

FIXME: 与 WebSocket API的行为保持一致

**label  类型是 DOMString, 只读**

该标签属性用来区分不同的[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象。这个值*必须*为对象创建时设置的值。

**onclose 类型是 EventHandler**

[close](http://www.w3.org/TR/webrtc/#event-datachannel-close)类型的事件处理器，所有实现[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)接口的类*必须*支持。

**onerror 类型是 EventHandler**

[error](http://www.w3.org/TR/webrtc/#event-datachannel-error)类型的事件处理器，所有实现[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)接口的类*必须*支持。

**onmessage 类型是 EventHandler**

[message](http://www.w3.org/TR/webrtc/#event-datachannel-message)类型的事件处理器，所有实现[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)接口的类*必须*支持。

**onopen 类型是 EventHandler**

[open](http://www.w3.org/TR/webrtc/#event-datachannel-open)类型的事件处理器，所有实现[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)接口的类*必须*支持。

**readyState 类型是**[**DataChannelState**](http://www.w3.org/TR/webrtc/#idl-def-DataChannelState)**, 只读**

该属性表示DataChannel对象的状态。这个值*必须*为用户代理上次设置的值。 (正如处理模型算法定义的那样).

**reliable 类型是 boolean, 只读**

True表示该DataChannel是可靠的，反之亦然。这个值*必须*为对象创建时设置的值。

#### 11.1.2 方法

**close**

用来关闭 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)。无论这个 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象是本节点主动创建还是被动创建的都可以调用。

当close()方法被调用时, 用户代理(user agent) must必须 开始DataChannel对象的底层数据传输器( [underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport))的销毁流程.

没有参数.

返回值类型: void

**send**

注意

FIXME: 与 WebSocket API的行为保持一

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **可选** | **描述** |
| data | DOMString | ✘ | ✘ |  |

返回值类型: void

**send**

注意

FIXME: 与 WebSocket API的行为保持一致

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **可选** | **描述** |
| data | ArrayBuffer | ✘ | ✘ |  |

返回值类型: void

**send**

注意

FIXME: 与 WebSocket API的行为保持一致

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **可选** | **描述** |
| data | Blob | ✘ | ✘ |  |

返回值类型: void

#### 11.1.3 其他类型

**DataChannelInit字典**

dictionary **DataChannelInit** {

boolean [reliable](http://www.w3.org/TR/webrtc/#widl-DataChannelInit-reliable);

};

reliable 类型是boolean

FIXME: 描述暂缺

**DataChannelStat枚举**

enum **DataChannelState** {

"connecting",

"open",

"closing",

"closed"

};

|  |  |
| --- | --- |
| **枚举类型描述(Enumeration description)** | |
| connecting | 此时用户代理(user agent)试图创建底层数据通道。由 [createDataChannel()](http://www.w3.org/TR/webrtc/#dom-peerconnection-createdatachannel)创建的 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象的初始状态。 |
| open | 此时底层数据通道已经创建而且可以进行通讯。由[DataChannelEvent](http://www.w3.org/TR/webrtc/#idl-def-DataChannelEvent) 事件派生出(dispatched)的[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)对象的初始状态。 |
| closing | 开始关闭底层数据通道([underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport))。 |
| closed | 底层数据通道（ [underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport)）已经关闭或者无法建立。 |

### 11.2 示例

这个简单的例子展示了如何创建一个DataChannel对象,然后注册一个事件监听器来处理收到的数据，以及如何发送数据(主动)。

EXAMPLE 3

var chan = peerConn.createDataChannel("mylabel");

chan.onmessage = function (evt) {

// use evt.data };

chan.send("hello");

}

这个简单的例子展示了如何注册一个事件监听器来处理[对端创建DataChannel对象]的情况(被动)。

EXAMPLE 4

peerConn.ondatachannel = function (evt) {

var chan = evt.channel;

chan.onmessage = function (evt) {

// use evt.data

};

chan.onclose = function () {

// remote side closed the data channel

};

};

## 12. 垃圾回收机制

作为构造器的全局变量，Window对象强制引用[RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) .

(A  Window object has a strong reference to any [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) objects created from the constructor whose global object is that Window object.)

## 13. 事件定义

### 13.1 [RTCPeerConnectionIceEvent](http://www.w3.org/TR/webrtc/#rtcpeerconnectioniceevent)

[Constructor(DOMString type, RTCPeerConnectionIceEventInit eventInitDict)]

interface **RTCPeerConnectionIceEvent** : *Event* {

readonly attribute [**RTCIceCandidate**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate) [candidate](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnectionIceEvent-candidate);

};  
dictionary **RTCPeerConnectionIceEventInit** : *EventInit* {

[**RTCIceCandidate**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate) [candidate](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnectionIceEventInit-candidate);

};

* RTCPeerConnection的**onicecandidate**事件使用**[RTCPeerConnectionIceEvent](http://www.w3.org/TR/webrtc/" \l "idl-def-RTCPeerConnectionIceEvent)**接口。
* ***触发一个***[***RTCPeerConnectionIceEvent***](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionIceEvent)***类型的名为e的事件***，并将[**RTCIceCandidate**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate)类型的候选人作为其参数，意味着这个名为*e*的事件既不能bubble(冒泡)也不能被取消(除非其他地方另有说明)；这个事件的candidate属性被设置为一个新的ICE候选人，那么这个事件 *必须* 在指定的目标(我认为就是这个New候选人)上创建、分发。

#### 13.1.1 属性

candidate**类型是**[RTCIceCandidate](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate)**, 只读**

发gua用所有的了如何候数据通道被关闭candidate 属性是一个新的 [RTCIceCandidate](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate)对象，它引发了这个事件。

#### 13.1.2 字典—RTCPeerConnectionIceEventInit

candidate**类型是**[RTCIceCandidate](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate)

### 13.2 MediaStreamEvent

[Constructor(DOMString type, MediaStreamEventInit eventInitDict)]

interface **MediaStreamEvent** : *Event* {

readonly attribute MediaStream? [stream](http://www.w3.org/TR/webrtc/#widl-MediaStreamEvent-stream);

};  
dictionary **MediaStreamEventInit** : *EventInit* {

MediaStream [stream](http://www.w3.org/TR/webrtc/#widl-MediaStreamEventInit-stream);

};

* [**addstream**](http://www.w3.org/TR/webrtc/#event-mediastream-addstream) 和[**removestream**](http://www.w3.org/TR/webrtc/#event-mediastream-removestream)事件使用 [MediaStreamEvent](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamEvent) 接口。
* ***触发一个名为e的流事件***，并将[**MediaStream流**](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate)作为其参数，意味着这个名为*e*的事件既不能bubble(冒泡)也不能被取消(除非其他地方另有说明)；这个事件的[stream](http://www.w3.org/TR/webrtc/#dom-mediastreamevent-stream)属性被设置为一个stream，那么这个事件 *必须* 在指定的目标(我认为就是这个stream)上创建、分发。

#### 13.2.1 属性

stream**类型是 MediaStream, 只读, 为空**

和这个事件相关的那个stream

#### 13.2.2 字典—MediaStreamEventInit

stream**类型是 MediaStream**

### 13.3 DataChannelEvent

[Constructor(DOMString type, DataChannelEventInit eventInitDict)]

interface **DataChannelEvent** : *Event* {

readonly attribute [**DataChannel**](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) [channel](http://www.w3.org/TR/webrtc/#widl-DataChannelEvent-channel);

};  
dictionary **DataChannelEventInit** : *EventInit* {

[**DataChannel**](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) [channel](http://www.w3.org/TR/webrtc/#widl-DataChannelEventInit-channel);

};

* [datachannel](http://www.w3.org/TR/webrtc/#event-rtcpeerconnection-datachannel) 事件使用 [DataChannelEvent](http://www.w3.org/TR/webrtc/#idl-def-DataChannelEvent) 接口。
* ***触发一个名为e的datachannel事件***，并将[[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)](http://www.w3.org/TR/webrtc/#idl-def-RTCIceCandidate)作为其参数，意味着这个名为*e*的事件既不能bubble(冒泡)也不能被取消(除非其他地方另有说明)；这个事件的[channel](http://www.w3.org/TR/webrtc/#dom-datachannelevent-channel)属性被设置为一个channel，那么这个事件 *必须* 在指定的目标(我认为就是这个channel)上创建、分发。

#### 13.3.1 属性

channel**类型是**[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)**, 只读**

和这个事件相关的那个channel

#### 13.3.2 字典—DataChannelEventInit

channel**类型是**[DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel)

## 14. 事件概要

这一节是非标准化的(non-normative).

如下的事件在 [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象上触发:

|  |  |  |
| --- | --- | --- |
| **事件名称** | **接口** | **何时触发…** |
| open | Event | [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象的底层数据传输通道( [underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport)) 已经建立 (或者已经重建). |
| MessageEvent | Event | 一个消息成功接收。 TODO: 参考在哪儿定义的MessageEvent? |
| error | Event | TODO. |
| close | Event | [DataChannel](http://www.w3.org/TR/webrtc/#idl-def-DataChannel) 对象的底层数据传输通道( [underlying data transport](http://www.w3.org/TR/webrtc/#dfn-underlying-data-transport)) 已经关闭。 |

如下的事件在 [RTCPeerConnection](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnection) 对象上触发:

|  |  |  |
| --- | --- | --- |
| **事件名称** | **接口** | **何时触发…** |
| connecting | Event | TODO |
| open | Event | TODO |
| addstream | [MediaStreamEvent](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamEvent) | 一个新的流已经加到 [remoteStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams) 数组中. |
| removestream | [MediaStreamEvent](http://www.w3.org/TR/webrtc/#idl-def-MediaStreamEvent) | 一个新的流已经从[remoteStreams](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams)数组中移除。 |
| datachannel | [DataChannelEvent](http://www.w3.org/TR/webrtc/#idl-def-DataChannelEvent) | TODO |
| negotiationneeded | Event | 浏览器想要通知应用程序在不久之后需要做会话协商 |
| statechange | Event | TODO |
| icechange | Event | TODO |
| icecandidate | [RTCPeerConnectionIceEvent](http://www.w3.org/TR/webrtc/#idl-def-RTCPeerConnectionIceEvent) | TODO |

## 15. 安全注意事项

TBD.

## 16. 变更日志

这部分将在定稿前被删除(所以下面的也就不翻译）

Changes since Aug 13, 2012

Made the RTCSessionDescription and RTCIceCandidate constructors take dictionaries instead of a strings. Also added detailed stringifier algorithm.

Went through the list of issues (issue numbers are only valid with HEAD at fcda53c460). Closed (fixed/wontfix): 1, 8, 10, 13, 14, 16, 18, 19, 22, 23, 24. Converted to notes: 4, 12. Updated: 9.

Incorporate [changes proposed](http://lists.w3.org/Archives/Public/www-archive/2012Aug/0015.html) by Li Li.

Use an enum for DataChannelState and fix IDLs where using an optional argument also requires all previous optional arguments to have a default value.

Changes since Jul 20, 2012

Added RTC Prefix to names (including the notes below).

Moved to new defintion of configuration and ice servers object.

Added correlating lines to candidate structure.

Converted setLocalDescription and setRemoteDescription to be asynchronous.

Added call flows.

Changes since Jul 13, 2012

Removed peer attribute from RTCPeerConnectionIceEvent (duplicates functionality of Event.target attribute).

Removed RTCIceCandidateCallback (no longer used).

Removed RTCPeerConnectionEvent (we use a simple event instead).

Removed RTCSdpType argument from setLocalDescription() and setRemoteDescription(). Updated simple example to match.

Changes since May 28, 2012

Changed names to use RTC Prefix.

Changed the data structure used to pass in STUN and TURN servers in configuration.

Updated simple RTCPeerConnection example (RTCPeerConnection constructor arguments; use icecandidate event).

Initial import of new Data API.

Removed some left-overs from the old Data Stream API.

Renamed “underlying data channel” to “underlying data transport”. Fixed closing procedures. Fixed some typos.

Changes since April 27, 2012

Major rewrite of RTCPeerConnection section to line up with IETF JSEP draft.

Added simple RTCPeerConnection example. Initial update of RTCSessionDescription and RTCIceCandidate to support serialization and construction.

Changes since 21 April 2012

Moved MediaStream and related definitions to getUserMedia.

Removed section “Obtaining local multimedia content”.

Updated getUserMedia() calls in examples (changes in Media Capture TF spec).

Introduced MediaStreamTrackList interface with support for adding and removing tracks.

Updated the algorithm that is run when RTCPeerConnection receives a stream (create new stream when negotiated instead of when data arrives).

Changes since 12 January 2012

Clarified the relation of Stream, Track, and Channel.

Changes since 17 October 2011

Tweak the introduction text and add a reference to the IETF RTCWEB group.

Changed the first argument to getUserMedia to be an object.

Added a MediaStreamHints object as a second argument to RTCPeerConnection.addStream.

Added AudioMediaStreamTrack class and DTMF interface.

Changes since 23 August 2011

Separated the SDP and ICE Agent into separate agents and added explicit state attributes for each.

Removed the send method from PeerConenction and associated callback function.

Modified MediaStream() constructor to take a list of MediaStreamTrack objects instead of a MediaStream. Removed text about MediaStream parent and child relationship.

Added abstract.

Moved a few paragraphs from the MediaStreamTrack.label section to the MediaStream.label section (where they belong).

Split MediaStream.tracks into MediaStream.audioTracks and MediaStream.videoTracks.

Removed a sentence that implied that track access is limited to LocalMediaStream.

Updated a few getUserMedia()-examples to use MediaStreamOptions.

Replaced calls to URL.getObjectURL() with URL.createObjectURL() in example code.

Fixed some broken getUserMedia() links.

Introduced state handling on MediaStreamTrack (removed state handling from MediaStream).

Reintroduced onended on MediaStream to simplify checking if all tracks are ended.

Aligned the MediaStreamTrack ended event dispatching behavior with that of MediaStream.

Updated the LocalMediaStream.stop() algorithm to implicitly use the end track algorithm.

Replaced an occurrence the term finished track with ended track (to align with rest of spec).

Moved (and extended) the explanation about track references and media sources from LocalMediaStream to MediaStreamTrack.

## A. 致谢

The editors wish to thank the Working Group chairs, Harald Alvestrand and Stefan Håkansson, for their support.

## B. 参考

### B.1 Normative references

**[GETUSERMEDIA]**

D. Burnett, A. Narayanan. [getusermedia: Getting access to local devices that can generate multimedia streams](http://dev.w3.org/2011/webrtc/editor/getusermedia.html) 13 August 2012. W3C Editors draft (Work in progress.) URL: <http://dev.w3.org/2011/webrtc/editor/getusermedia.html>

**[HTML5]**

Ian Hickson; David Hyatt. [HTML5.](http://www.w3.org/TR/html5) 29 March 2012. W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/html5>

**[ICE]**

J. Rosenberg. [Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.](http://tools.ietf.org/html/rfc5245) April 2010. Internet RFC 5245. URL: <http://tools.ietf.org/html/rfc5245>

**[RFC2119]**

S. Bradner. [Key words for use in RFCs to Indicate Requirement Levels.](http://www.ietf.org/rfc/rfc2119.txt) March 1997. Internet RFC 2119. URL: <http://www.ietf.org/rfc/rfc2119.txt>

**[RTCWEB-CONSTRAINTS]**

D. Burnett. [IANA Registry for RTCWeb Media Constraints.](http://datatracker.ietf.org/doc/draft-burnett-rtcweb-constraints-registry/) URL: <http://datatracker.ietf.org/doc/draft-burnett-rtcweb-constraints-registry/>

**[SDP]**

J. Rosenberg, H. Schulzrinne. [An Offer/Answer Model with the Session Description Protocol (SDP).](http://tools.ietf.org/html/rfc3264) June 2002. Internet RFC 3264. URL:<http://tools.ietf.org/html/rfc3264>

**[STUN]**

J. Rosenberg, R. Mahy, P. Matthews, D. Wing. [Session Traversal Utilities for NAT (STUN).](http://tools.ietf.org/html/rfc5389) October 2008. Internet RFC 5389. URL: <http://tools.ietf.org/html/rfc5389>

**[STUN-URI]**

S. Nandakumar, G. Salgueiro, P. Jones, and M. Petit-Huguenin. [URI Scheme for Session Traversal Utilities for NAT (STUN) Protocol.](http://tools.ietf.org/html/draft-nandakumar-rtcweb-stun-uri) 12 March 2012. Internet Draft (work in progress). URL: <http://tools.ietf.org/html/draft-nandakumar-rtcweb-stun-uri>

**[TURN]**

P. Mahy, P. Matthews, J. Rosenberg. [Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).](http://tools.ietf.org/html/rfc5766) April 2010. Internet RFC 5766. URL: <http://tools.ietf.org/html/rfc5766>

**[TURN-URI]**

M. Petit-Huguenin, S. Nandakumar, G. Salgueiro, and P. Jones. [Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers.](http://tools.ietf.org/html/draft-petithuguenin-behave-turn-uris) 12 March 2012. Internet Draft (work in progress). URL: <http://tools.ietf.org/html/draft-petithuguenin-behave-turn-uris>

**[WEBIDL]**

Cameron McCormack. [Web IDL.](http://www.w3.org/TR/2011/WD-WebIDL-20110927/) 27 September 2011. W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/2011/WD-WebIDL-20110927/>

### B.2 Informative references

**[RTCWEB-JSEP]**

J. Uberti, C. Jennings. [Javascript Session Establishment Protocol.](http://datatracker.ietf.org/doc/draft-ietf-rtcweb-jsep/) URL: <http://datatracker.ietf.org/doc/draft-ietf-rtcweb-jsep/>