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

目录

[WebRTC 1.0: Between Browsers中文版 1](#_Toc353181709)

[前言 2](#_Toc353181710)

[1. 介绍 4](#_Toc353181711)

[2. 一致性 4](#_Toc353181712)

[3. 术语 4](#_Toc353181713)

[4. Peer-to-peer connections 5](#_Toc353181714)

[4.1. 介绍 5](#_Toc353181715)

[4.2. Configuration 5](#_Toc353181716)

[4.3. RTCPeerConnection Interface 5](#_Toc353181717)

[4.4. State定义 5](#_Toc353181718)

[4.5. Callback定义 5](#_Toc353181719)

[4.6. 错误处理 5](#_Toc353181720)

[4.7. 会话描述Model 5](#_Toc353181721)

[4.8. 建立连接接口 5](#_Toc353181722)

[5. Peer-to-peer Data API 5](#_Toc353181723)

[5.1. RTCPeerConnection接口扩展 5](#_Toc353181724)

[5.2. RTCDataChannel 5](#_Toc353181725)

[5.3. RTCDataChannelEvent 5](#_Toc353181726)

[5.4. 垃圾回收机制 5](#_Toc353181727)

[6. Peer-to-peer DTMF 5](#_Toc353181728)

[6.1. RTCPeerConnection接口扩展 5](#_Toc353181729)

[6.2. RTCDTMFSender 6](#_Toc353181730)

[6.3. RTCDTMFToneChangeEvent 6](#_Toc353181731)

[7. 统计Model 6](#_Toc353181732)

[7.1. 介绍 6](#_Toc353181733)

[7.2. RTCPeerConnection接口扩展 6](#_Toc353181734)

[7.3. RTCStatsCallback 6](#_Toc353181735)

[7.4. RTCStatsReport 对象 6](#_Toc353181736)

[7.5. RTCStats字典 6](#_Toc353181737)

[7.6. 其他派生字典 6](#_Toc353181738)

[7.7. 示例 6](#_Toc353181739)

[8. Network Stream API 6](#_Toc353181740)

[4.1 介绍 6](#_Toc353181741)

[4.2 接口定义 7](#_Toc353181742)

[4.3 AudioMediaStreamTrack 9](#_Toc353181743)

[9. Peer-to-peer connections 10](#_Toc353181744)

[5.1 Connection对象创建 10](#_Toc353181745)

[5.2 连接算法 11](#_Toc353181746)

[5.3 算法补充 11](#_Toc353181747)

[5.4 Media流新增 11](#_Toc353181748)

[5.5 Media流移除 12](#_Toc353181749)

[5.6 Negotiationneeded事件 13](#_Toc353181750)

[5.7 RTCPeerConnection的属性和方法 13](#_Toc353181751)

[10. IANA 注册 25](#_Toc353181752)

[6.1 约束 25](#_Toc353181753)

[11. 简单示例 26](#_Toc353181754)

[12. 高级示例 27](#_Toc353181755)

[13. 浏览器间建立连接 27](#_Toc353181756)

[14. 浏览器与MCU连接 28](#_Toc353181757)

[15. Peer-to-peer Data API 29](#_Toc353181758)

[11.1 DataChannel 29](#_Toc353181759)

[11.2 示例 34](#_Toc353181760)

[16. 垃圾回收机制 34](#_Toc353181761)

[17. 事件定义 34](#_Toc353181762)

[13.1 RTCPeerConnectionIceEvent 34](#_Toc353181763)

[13.2 MediaStreamEvent 35](#_Toc353181764)

[13.3 DataChannelEvent 36](#_Toc353181765)

[18. 事件概要 36](#_Toc353181766)

[19. 安全注意事项 37](#_Toc353181767)

[20. 变更日志 37](#_Toc353181768)

[A. 致谢 41](#_Toc353181769)

[B. 参考 41](#_Toc353181770)

[B.1 Normative references 41](#_Toc353181771)

[B.2 Informative references 42](#_Toc353181772)

## 前言

**说明**

W3c 2013年3月22日的版本：  
This version可以在这里找到:  
<http://dev.w3.org/2011/webrtc/editor/webrtc.html>  
最近公开的版本:  
<http://www.w3.org/TR/webrtc/>

Editors:

Adam Bergkvist, Ericsson

Daniel C. Burnett, Voxeo

Cullen Jennings, Cisco

Anant Narayanan, Mozilla (until November 2012)

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

这个文档还不完整和稳定，因此作为商业应用是不合适的，但鼓励作为早期的试验。API基于早期WHATWG中的工作。WebRtc工作小组希望通过以下几点来不断完善这个文档：

* The outcome of ongoing exchanges in the companion RTCWEB group at IETF to define the set of protocols that, together with this document, will enable real-time communications in Web browsers.
* Privacy issues that arise when exposing local capabilities and local streams.
* Technical discussions within the group.
* Experience gained through early experimentations.
* Feedback received from other groups and individuals.

这个文档作为 [Web Real-Time Communications工作组](http://www.w3.org/2011/04/webrtc/)的编辑稿件公开。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。

作为编辑稿件（Editor's 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。(Working draft)~~
* 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)]

## ****Peer-to-peer connections****

### 介绍

[RTCPeerConnection](#_4.3.2_Interface_Definition)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)

### [Configuration](http://dev.w3.org/2011/webrtc/editor/webrtc.html#configuration)配置

#### 4.2.1 RTCConfiguration字典

dictionary RTCConfiguration {

[RTCIceServer](#_4.2.2_RTCIceServer_Type)[] iceServers;

};

**字典成员：**

**iceServers** [*RTCIceServer*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceServer)类型的数组

可用于ICE的STUN和TURN服务器数组

#### 4.2.2 RTCIceServer字典

dictionary RTCIceServer {

DOMString url;

DOMString? credential;

};

**字典成员：**

**credential** DOMString类型 可空

如果url元素是TURN URI，这个就是使用TURN服务器的凭证

**url** DOMString类型

STUN或者TURN服务器URI，格式定义参照[[STUN-URI](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-STUN-URI)] and [[TURN-URI](http://dev.w3.org/2011/webrtc/editor/webrtc.html#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数组示例：

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

### [RTCPeerConnection接口](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-interface)

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)].

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。

注：这一章节中所提及的任务的源是网络任务源。

#### 4.3.1 操作--operations

* 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。(修改如下)~~
* RTCPeerConnection 对象还与 *ICE agent*[[ICE](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)], RTCPeerConnection signaling state(*发信号状态)*, ICE gathering state(收集状态), 和 ICE connection state有关. 当对象被创建时这些将会被初始化。
* (*新增条目*) RTCPeerConnection对象有两个相关的流集合：通过这个connection正在发送的本地流集合和正在接收的远端流集合。当RTCPeerConnection对象创建时，这些集合会初始化为空。

##### 4.3.1.1 RTCPeerConnection创建

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(in [[ICE](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)])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 Agent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（例如2个）.而且随着RTCPeerConnection对象获取了更多信息的时ICE Agent可以随时调节组件的数量。
2. ~~设置RTCPeerConnection的readiness state 为 new .~~

设置连接的RTCPeerConnection signalingState(4.3.2.1)为stable

1. ~~设置~~[~~RTCPeerConnection的ice state~~](http://www.w3.org/TR/webrtc/#rtcpeerconnection-readiness-state)~~为 new .~~

设置连接的RTCPeerConnection ice connection state(4.3.2.1)为new。

1. ~~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。~~

设置RTCPeerConnection ice gathering state(4.3.2.1)为new

1. ~~设置~~[~~remoteStreams~~](http://www.w3.org/TR/webrtc/#widl-RTCPeerConnection-remoteStreams)~~属性的值为空的只读 MediaStream 数组。~~

用空的集合来初始化一组operations

1. 返回 RTCPeerConnection, 但继续异步执行这些步骤。
2. 等待一个稳当的状态。同步的部分包含该算法的剩下步骤。

##### 4.3.1.2 操作队列--operations数组

当RTCPeerConnection对象初始化之后，为了调用(每次调用…之后？) createOffer, setLocalDescription, createAnswer和setRemoteDescription，需要执行如下步骤：

1. 向operations数组中添加一个对象，这个对象表示当前正在被处理的调用(也就是函数名和其对应的参数)

2. 如果operations数组的长度正好是1，异步执行这个序列的前部的那个函数。

3. 当异步操作完成(无论成功与否)，都将对应的那个对象从operations数组中移除。然后如果数组不空，继续异步执行序列的第一个对象(其实是任务)，以此循环执行知道结束。

大概的设想是,在任意时间点只会执行createOffer, setLocalDescription, createAnswer和setRemoteDescription这些操作中的某一个。如果(当有调用正在执行)又有后续调用发生，那么新来的这些将会被添加到一个序列中，等到之前的调用执行完了会处理新加的这些。对于普通的错误处理程序，这样做是有效而且希望的。

##### 4.3.1.3 其他规则

此外，在RTCPeerConnection对象的生命周期中,如下的规则需要遵从:

1. 如果 ~~iceState~~ *iceConnectionState*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”。
2. 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, *iceConnectionState*将被修改为”connected”.
3. 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,*iceConnectionState*将被修改为”completed”，如果任何一个MediaStreamTrack都没有找到一条连接, *iceConnectionState*将被修改为 “failed”.

问题 (来自工作草案)

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?

1. 如果 *iceConnectionState*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”.(改成如下)~~

而且本地和远端的会话描述已经接收到了有效的SDPoffer/answer对(意思应该就是sdp被设置), RTCPeerConnection的状态将被置为 “stable”。

1. 如果 *iceConnectionState*等于 “failed”, 队列里将插入一个任务来调用closel方法。

问题 1

讨论中的问题: CJ – 这个我认为是错的。--只是因为网络连接失败不代表那台PC就必须变成不可恢复的状态。

##### 4.3.1.4 ICE Agent任务

当ICE Agent需要通知脚本关于候选人收集的情况，用户代理就需要在队列中添加一个任务来执行如下步骤：

1. 使这个connection与这个ICE Agent相关联。
2. 如果这个connection的*signalingState*是closed，中止操作。
3. 根据ICE Agent通知脚本不同的事情执行：

* 一个新的候选人是有效的

将那个候选人添加到connection的localDescription(4.3.2.1)中，然后创建一个[**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) (4.8.1)对象(newCandidate )来表示这个候选人。

* 候选人收集完成

设置connection的iceGatheringState为completed，然后使newCandidate 为null。

1. 在这个connection上触发一个名叫[icecandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-icecandidate)(参见事件概要章)的事件，并将这个newCandidate 作为参数。

##### 4.3.1.5 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：

“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。

(来自工作草案)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。

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对象来表示这个接受的媒体流.
3. 用mediatrack来表示媒体流中的每个rtp media流，按此[算法](#_9.2.2_Events_on)执行。

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。
* （来自工作草案）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)~~RTCPeerConnection signalingState为closed, 那么就放弃这些步骤.
* 将新创建的MediaStream 对象加入到connection的 *remote streams set*中。
* 在connection上触发一个流事件(9.4),名为 [addstream](http://www.w3.org/TR/webrtc/#event-mediastream-addstream)(11.)， 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。

##### 4.3.1.6 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RTCPeerConnection signalingState等于*closed* , 则中止这些步骤。
* 从connection的*remote streams set*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.

##### 4.3.1.7 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[addTrack()](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-addtrack)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。

#### 4.3.2 接口定义

[Constructor (RTCConfiguration configuration, optional MediaConstraints constraints)]

interface **RTCPeerConnection** : *EventTarget*  {

void [createOffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-createOffer-void-RTCSessionDescriptionCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback-MediaConstraints-constraints) ([**RTCSessionDescriptionCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback) *successCallback*, [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*, optional MediaConstraints *constraints*);

void [createAnswer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-createAnswer-void-RTCSessionDescriptionCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback-MediaConstraints-constraints) ([**RTCSessionDescriptionCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback) *successCallback*, [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*, optional MediaConstraints *constraints*);

void [setLocalDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-setLocalDescription-void-RTCSessionDescription-description-VoidFunction-successCallback-RTCPeerConnectionErrorCallback-failureCallback) ([**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) *description*, VoidFunction *successCallback*, [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

readonly attribute [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) [localDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-localDescription);

void [setRemoteDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-setRemoteDescription-void-RTCSessionDescription-description-VoidFunction-successCallback-RTCPeerConnectionErrorCallback-failureCallback) ([**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) *description*, VoidFunction *successCallback*, [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

readonly attribute [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) [remoteDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-remoteDescription);

readonly attribute [**RTCSignalingState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSignalingState) [signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-signalingState);

void [updateIce](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-updateIce-void-RTCConfiguration-configuration-MediaConstraints-constraints) (optional [**RTCConfiguration**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCConfiguration) *configuration*, optional MediaConstraints *constraints*);

void [addIceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-addIceCandidate-void-RTCIceCandidate-candidate) ([**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) *candidate*);

readonly attribute [**RTCIceGatheringState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceGatheringState) [iceGatheringState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-iceGatheringState);

readonly attribute [**RTCIceConnectionState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceConnectionState) [iceConnectionState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-iceConnectionState);

sequence<MediaStream> [getLocalStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-getLocalStreams-sequence-MediaStream) ();

sequence<MediaStream> [getRemoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-getRemoteStreams-sequence-MediaStream) ();

MediaStream? [getStreamById](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-getStreamById-MediaStream-DOMString-streamId) (DOMString *streamId*);

void [addStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-addStream-void-MediaStream-stream-MediaConstraints-constraints) (MediaStream *stream*, optional MediaConstraints *constraints*);

void [removeStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-removeStream-void-MediaStream-stream) (MediaStream *stream*);

void [close](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-close-void) ();

attribute EventHandler [onnegotiationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onnegotiationneeded);

attribute EventHandler [onicecandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onicecandidate);

attribute EventHandler [onsignalingstatechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onsignalingstatechange);

attribute EventHandler [onaddstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onaddstream);

attribute EventHandler [onremovestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onremovestream);

attribute EventHandler [oniceconnectionstatechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-oniceconnectionstatechange);

};

##### 4.3.2.1 属性

###### iceConnectionState

类型 RTCIceConnectionState, readonly

This attribute must return the state of the RTCPeerConnection ICE Agent ICE state.

###### iceGatheringState

类型 RTCIceGatheringState, readonly

This attribute must return the gathering state of the RTCPeerConnection ICE Agent connection state.

###### localDescription

类型 RTCSessionDescription, readonly

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。

###### onaddstream

类型 EventHandler

这是addstream(11.)类型的事件处理器。所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象 *必须* 支持这个事件。只要通过远端节点增加一个媒体流，就必须触发这个事件。具体的说只有当setRemoteDescription调用后才会触发这个事件，而且要尽快。这个回调不等待指定流的SDP协商结果(即无论接受还是拒绝)。

###### onicecandidate

类型 EventHandler

这是 onicecandidate(11.) 类型的事件处理器, 所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象 *必须* 支持这个事件。~~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。~~

###### oniceconnectionstatechange

类型 EventHandler,

这是 [iceconnectionstatechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-iceconnectionstatechange)（11.）类型的事件处理器，所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象*必须*支持。只要当iceConnectionState 改变后调用。

###### onnegotiationneeded

类型 EventHandler,

这是[negotiationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-negotiation-needed)（11.）类型的事件处理器，所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象*必须*支持。

###### onremovestream

类型 EventHandler

这是 removestream(11.)类型的事件处理器, 所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象 *必须* 支持这个事件。当一个媒体流被远端移除时，这个事件将被触发。具体的说只有当setRemoteDescription调用后才会触发这个事件

###### onsignalingstatechange

类型 EventHandler,

这是[signalingstatechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-signalingstatechange)(11.)类型的事件处理器，所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象 *必须* 支持这个事件。只要当readyState 改变后调用，也就是说调用setLocalDescription或者setRemoteDescription或者code之后。(It is called any time the readyState changes, i.e., from a call to setLocalDescription, a call to setRemoteDescription, or code.).当从初始状态变成new状态，不需要触发该事件。

###### remoteDescription

类型 RTCSessionDescription, readonly

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。

###### signalingState

类型 RTCSignalingState, readonly

该属性must 返回[RTCPeerConnection signaling state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state)。

##### 4.3.2.2 方法

###### addIceCandidate

该方法给ICE Agent提供了一个远端的 candidate,。除了会把(这个candidate)添加到远端描述符中，在“IceTransports”约束没有设置为’none’之前还会向这个新的候选者发送连通性检测。这个方法调用将造成ICE Agent的connection状态改变,而且如果因此创建了另外一条连接，媒体状态也会改变 。

如果candidate参数格式不正确，将会抛出INVALID\_CANDIDATE\_TYPE 类型的RTCError异常。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state)状态是 closed,抛出一个 INVALID\_STATE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 signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state) to closed.

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

###### 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

###### getLocalStreams

Returns a sequence of MediaStream objects representing the streams that are currently sent with this[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object.

The ***getLocalStreams()*** method must return a new sequence that represents a snapshot of all the MediaStreamobjects in this [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s [local streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html#local-streams-set). The conversion from the streams set to the sequence, to be returned, is user agent defined and the order does not have to stable between calls.

No parameters.

Return type:sequence<MediaStream>

###### getRemoteStreams

Returns a sequence of MediaStream objects representing the streams that are currently received with this[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object.

The ***getRemoteStreams()*** method must return a new sequence that represents a snapshot of all the MediaStreamobjects in this [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s [remote streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html#remote-streams-set). The conversion from the streams set to the sequence, to be returned, is user agent defined and the order does not have to stable between calls.

No parameters.

Return type:sequence<MediaStream>

###### getStreamById

If a MediaStream object, with an [id](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-id) equal to trackId, exists in this [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s stream sets ([local streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html#local-streams-set) or [remote streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html#remote-streams-set)), then the ***getStreamById()*** method must return that MediaStreamobject. The method must return null if no stream matches the streamId argument.

NOTE

For this method to make sense, we need to make sure that ids are unique within the two stream sets of a PeerConnection. This is not the case today when a peer re-adds a stream that is received. Two different stream instances will now have the same id at both peers; one in the remote stream set and one in the local stream set.

One way to resolve this is to not allow re-adding a stream instance that is received (guard on id). If an application really needs this functionality it's really easy to make a clone of the stream, which will give it a new id, and send the clone.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| streamId | DOMString | ✘ | ✘ |  |

Return type:MediaStream, nullable

###### 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

#### 4.3.3 Garbage collection

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.)

### State定义

#### 4.4.1 RTCPeerState Enum—RTCSignalingState

enum RTCSignalingState {

"stable",

"have-local-offer",

"have-remote-offer",

"have-local-pranswer",

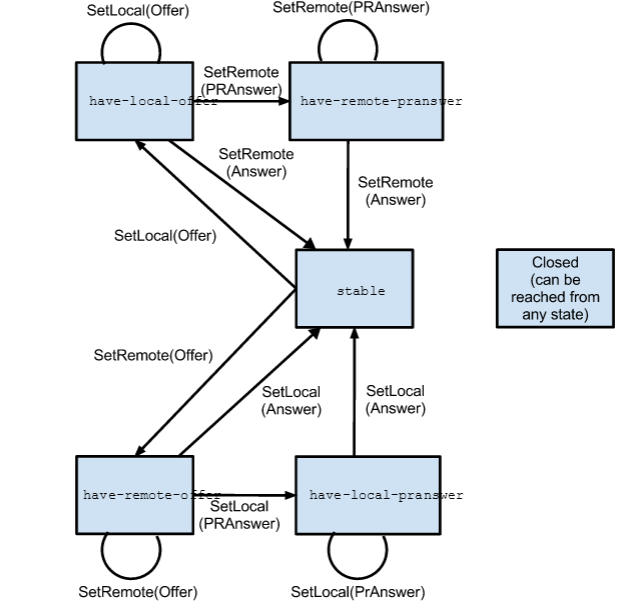
"have-remote-pranswer",

"closed"

};

|  |  |
| --- | --- |
| **Enumeration description** | |
| stable | There is no offer­answer exchange in progress. This is also the initial state in which case the local and remote descriptions are empty. |
| have-local-offer | A local description, of type "offer", has been supplied. |
| have-remote-offer | A remote description, of type "offer", has been supplied. |
| have-local-pranswer | A remote description of type "offer" has been supplied and a local description of type "pranswer" has been supplied. |
| have-remote-pranswer | A local description of type "offer" has been supplied and a remote description of type "pranswer" has been supplied. |
| closed | The connection is closed. |

非规范化的peer状态转换图：



一组状态变化的示例：

呼叫方转换:

* new PeerConnection(): stable
* setLocal(offer): have-local-offer
* setRemote(pranswer): have-remote-pranswer
* setRemote(answer): stable
* close(): closed

被叫方转换:

* new PeerConnection(): stable
* setRemote(offer): have-remote-offer
* setLocal(pranswer): have-local-pranswer
* setLocal(answer): stable
* close(): closed

#### 4.4.2 RTCIceGatheringState Enum

enum RTCIceGatheringState {

"new",

"gathering",

"complete"

};

|  |  |
| --- | --- |
| **Enumeration description** | |
| new | The object was just created, and no networking has occurred yet. |
| gathering | The ICE engine is in the process of gathering candidates for this RTCPeerConnection. |
| complete | The ICE engine has completed gathering. Events such as adding a new interface or a new TURN server will cause the state to go back to gathering. |

#### 4.4.3 RTCIceConnectionState Enum

enum RTCIceConnectionState {

"new",

"checking",

"connected",

"completed",

"failed",

"disconnected",

"closed"

};

|  |  |
| --- | --- |
| **Enumeration description** | |
| new | The ICE Agent is gathering addresses and/or waiting for remote candidates to be supplied. |
| checking | The ICE Agent has received remote candidates on at least one component, and is checking candidate pairs but has not yet found a connection. In addition to checking, it may also still be gathering. |
| connected | The ICE Agent has found a usable connection for all components but is still checking other candidate pairs to see if there is a better connection. It may also still be gathering. |
| completed | The ICE Agent has finished gathering and checking and found a connection for all components. Open issue: it is not clear how the non controlling ICE side knows it is in the state. |
| failed | The ICE Agent is finished checking all candidate pairs and failed to find a connection for at least one component. Connections may have been found for some components. |
| disconnected | Liveness checks have failed for one or more components. This is more aggressive than failed, and may trigger intermittently (and resolve itself without action) on a flaky network. |
| closed | The ICE Agent has shut down and is no longer responding to STUN requests. |

该状态取值如下所述：

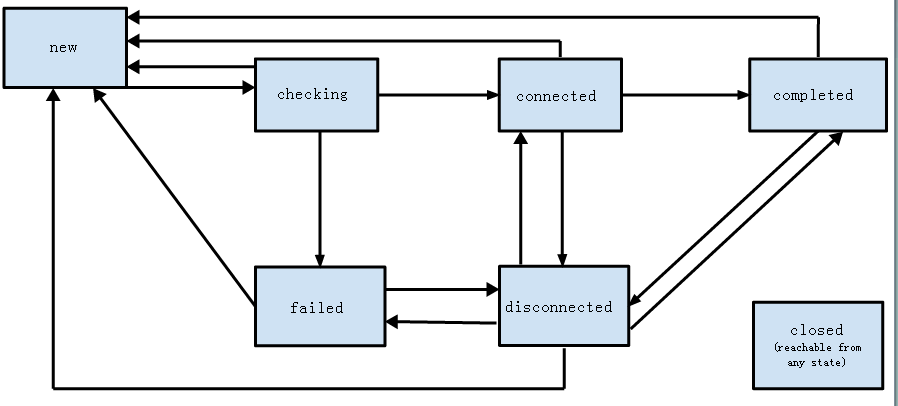
* checking occurs if ANY component has received a candidate and can start checking
* connected occurs if ALL components have established a working connection
* completed occurs if ALL components have finalized the running of their ICE processes
* failed occurs if ANY component has given up trying to connect
* disconnected occurs if ANY component has failed liveness checks
* closed occurs only if PeerConnection.close() has been called.

If a component is discarded as a result of signaling (e.g. RTCP mux or BUNDLE), the state may advance directly fromchecking to completed.

**状态转变的示例：**

* new PeerConnection(): new
* (new, remote candidates received): checking
* (checking, found usable connection): connected
* (checking, gave up): failed
* (connected, finished all checks): completed
* (completed, lost connectivity): disconnected
* (any state, ICE restart occurs): new
* close(): closed

**非规范化的ICE(connection)状态转变图：**

****

### Callback定义

#### 4.5.1 RTCPeerConnectionErrorCallback

callback RTCPeerConnectionErrorCallback = void (RTCError error);

参数：

error 类型[**RTCError**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCError)(4.6.2)

封装了错误信息的对象

### 错误处理

#### 4.6.1 一般规则

Errors are indicated in two ways: exceptions and objects passed to error callbacks. Both forms of error reporting must provide an object of type RTCError. An exception must be thrown in the following cases:

* The type of any argument passed to a function did not match what was expected. An appropriate string from the RTCExceptionName enum must be used as the error name.
* A function call was made when the RTCPeerConnection is in an invalid state, or a state in which that particular function is not allowed to be executed. In this case, the string INVALID\_STATEmust be used as the error name.

In all other cases, an error object must be provided to the failure callback. The error name in the object provided must be picked from either the RTCExceptionName or RTCErrorName enums.

#### 4.6.2 RTCError

待解决问题：RTCError 需要扩展为DOMError嘛？

interface RTCError {

readonly attribute DOMString name;

readonly attribute DOMString? message;

};

##### 4.6.2.1 属性

**message** of type DOMString, readonly , nullable

A human readable description of the error. This string may vary between different user agents.

**name** of type DOMString, readonly

A string representing the type of error. This string must be one of those defined by theRTCExceptionName or RTCErrorName enums for the error object to be valid.

#### 4.6.3 RTCSdpError

interface RTCSdpError : RTCError {

readonly attribute long sdpLineNumber;

};

##### 4.6.3.1 属性

**sdpLineNumber** of type long, readonly

The line number of an [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) at which the error was encountered.

#### 4.6.4 RTCExceptionName

enum RTCExceptionName {

"INVALID\_CONSTRAINTS\_TYPE",

"INVALID\_CANDIDATE\_TYPE",

"INVALID\_MEDIASTREAM\_TRACK",

"INVALID\_STATE"

};

|  |  |
| --- | --- |
| **Enumeration description** | |
| INVALID\_CONSTRAINTS\_TYPE | The provided constraints object is not a dictionary with either themandatory or optional keys. |
| INVALID\_CANDIDATE\_TYPE | The provided candidate is not an object of type RTCIceCandidate. |
| INVALID\_MEDIASTREAM\_TRACK | The provided track is not an object of type MediaStreamTrack. |
| INVALID\_STATE | The function was called on a RTCPeerConnection that is an invalid state, or a state in which the function is not allowed to be executed. |

#### 4.6.5 RTCErrorName

enum RTCErrorName {

"INVALID\_SESSION\_DESCRIPTION",

"INCOMPATIBLE\_SESSION\_DESCRIPTION",

"INCOMPATIBLE\_CONSTRAINTS",

"INCOMPATIBLE\_MEDIASTREAMTRACK",

"INTERNAL\_ERROR"

};

|  |  |
| --- | --- |
| **Enumeration description** | |
| INVALID\_SESSION\_DESCRIPTION | The provided [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) contained invalid SDP, or the[type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCSessionDescription-type) was wrong for the current state of the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection). User agents should provide as much additional information in the error message as possible, including the sdpLineNumber, if appropriate. |
| INCOMPATIBLE\_SESSION\_DESCRIPTION | The provided [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) contained SDP that could not be correctly applied to the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) due to its current state. User agents should provide as much additional information in the error message as possible, including the sdpLineNumber, if appropriate. |
| INCOMPATIBLE\_CONSTRAINTS | The provided MediaConstraints could not be correctly applied to theRTCPeerConnection due to its current state. User agents should provide as much additional information in the error message as possible. |
| INCOMPATIBLE\_MEDIASTREAMTRACK | The provided MediaStreamTrack is not an element of a MediaStream that is currently in the RTCPeerConnection's localStreams attribute. |
| INTERNAL\_ERROR | The [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) encountered an error that it could not recover from. |

### 会话描述Model

#### 4.7.1 RTCSdpType

表示[**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#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。

#### 4.7.2 RTCSessionDescription Class

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。

实现RTCSessionDescription接口的对象*必须*用序列样式"{ attribute }"序列化自己。见4.7.2.2

(Objects implementing the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) interface must serialize with the serialization pattern "{ attribute }".)

dictionary RTCSessionDescriptionInit {

RTCSdpType type;

DOMString sdp;

};

[Constructor (optional RTCSessionDescriptionInit descriptionInitDict)]

interface RTCSessionDescription {

attribute RTCSdpType? type;

attribute DOMString? sdp;

serializer = {attribute};

};

##### 4.7.2.1 属性

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

sdp的类型

**sdp** 类型DOMString nullable

sdp内容

##### 4.7.2.2 串化器Serializer

这个接口的实例都用一个map序列化，这个map中存放可序列化的属性。

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”]})

注：

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

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

: == U+003A “ == U+0022

, == U+002C

##### 4.7.2.3 字典RTCSessionDescriptionInit

**sdp** 类型DOMString

**type** 类型 [*RTCSdpType*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSdpType)

#### 4.7.3 RTCSessionDescriptionCallback

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

##### 4.7.3.1 参数

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

包含sdp的对象

### 建立连接接口

#### 4.8.1 RTCIceCandidate Type

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。

实现RTCSessionDescription接口的对象*必须*用序列样式"{ attribute }"序列化自己。见4.7.2.2

(Objects implementing the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) interface must serialize with the serialization pattern "{ attribute }".)

dictionary RTCIceCandidateInit {

DOMString candidate;

DOMString sdpMid;

unsigned short sdpMLineIndex;

};

[Constructor (optional RTCIceCandidateInit candidateInitDict)]

interface RTCIceCandidate {

attribute DOMString? candidate;

attribute DOMString? sdpMid;

attribute unsigned short? sdpMLineIndex;

serializer = {attribute};

};

##### 4.8.1.1 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。

##### 4.8.1.2 串化器Serializer

这个接口的实例都用一个map串化，这个map中存放可串化的属性。

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”]})

注：

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

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

: == U+003A “ == U+0022

, == U+002C

##### 4.8.1.3 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

#### 4.8.2 RTCPeerConnectionIceEvent

* RTCPeerConnection的**icecandidate**事件使用[**RTCPeerConnectionIceEvent**](http://www.w3.org/TR/webrtc/#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。

dictionary RTCPeerConnectionIceEventInit : EventInit {

RTCIceCandidate candidate;

};

[Constructor(DOMString type, RTCPeerConnectionIceEventInit eventInitDict)]

interface RTCPeerConnectionIceEvent : Event {

readonly attribute RTCIceCandidate candidate;

};

##### 4.8.2.1 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。

##### 4.8.2.2 字典RTCPeerConnectionIceEventInit

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

## ****Peer-to-peer Data API****

Peer-to-peer Data API让 web应用程序能够发送和接收一般的(generic)基于应用的peer-to-peer数据。用于收发数据的API参考模仿了WebSockets [[WEBSOCKETS-API](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-WEBSOCKETS-API)]中的方式。

问题 11: 更多开放性的问题(摘自工作草案)

* 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？

### [RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口扩展

Peer-to-peer Data API扩展自[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口，定义如下：

partial interface RTCPeerConnection {

RTCDataChannel createDataChannel([TreatNullAs=EmptyString] DOMString label,

optional RTCDataChannelInit dataChannelDict);

attribute EventHandler ondatachannel;

};

#### 5.1.1 属性

**ondatachannel** 类型EventHandler

这是[datachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-peerconnection-datachannel)类型的事件处理器 , 所有实现[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 接口的对象 *必须* 支持这个事件。

#### 5.1.2 方法

**createDataChannel**

用指定的标签（label)创建一个新的 **[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**对象。 [**RTCDataChannelInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannelInit)字典参数可以用来配置底层的传输通道属性，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 [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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 signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state)是closed, 抛出一个 INVALID\_STATE异常。
2. 让channel 成为一个新创建的 [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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 | [**RTCDataChannelInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannelInit) | ✘ | ✔ |  |

返回类型: **[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**

### [RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "rtcdatachannel)

#### 5.2.00 通道创建与销毁

* 数据通道( [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel))接口表示一个于两个节点之间的双向的数据通道 。数据通道( [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel))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 [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象也将会在对端节点部署。

每个 [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象*必须* 初始化为connecting 状态。如果[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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)[[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**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 signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state) 是 closed , 放弃执行下面的步骤。
2. 宣告该通道为[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)。
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 signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state)是 closed , 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**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**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:**

**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**对象的底层数据传输可能通过***closing procedure（下面步骤）***的方式正常关闭，如果这样，那么用户代理 *必须*(除非程序已经被[close()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-close)方法初始化)在队列中添加一个任务来设置 [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象的[readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate)属性为closing。这将最终导致数据传输被关闭。

注意:

需要参考协议说明文档

当底层数据传输被关闭，用户代理 *必须* 在队列中添加一个任务来执行下面步骤：

1. 将已经把数据传输通道关闭的[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象称为channel 。

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.

1. 将 channel的 [readyState](http://www.w3.org/TR/webrtc/#dom-datachannel-readystate) 属性设置为closed.
2. 如果底层数据传输关闭返回错误，那么在channel上触发一个错误事件。
3. 在channel上触发一个 [close](http://www.w3.org/TR/webrtc/#event-datachannel-close) 的简单事件。

#### 5.2.01 定义

dictionary RTCDataChannelInit {

boolean reliable;

};

interface RTCDataChannel : EventTarget {

readonly attribute DOMString label;

readonly attribute boolean reliable;

readonly attribute RTCDataChannelState readyState;

readonly attribute unsigned long bufferedAmount;

attribute EventHandler onopen;

attribute EventHandler onerror;

attribute EventHandler onclose;

void close ();

attribute EventHandler onmessage;

attribute DOMString binaryType;

void send (DOMString data);

void send (Blob data);

void send (ArrayBuffer data);

void send (ArrayBufferView data);

};

#### 5.2.1 属性

##### binaryType

类型DOMString

The ***binaryType*** attribute must, on getting, return the value to which it was last set. On setting, the user agent must set the IDL attribute to the new value. When a **[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**object is created, the [binaryType](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-binarytype) attribute must be initialized to the string "blob".

This attribute controls how binary data is exposed to scripts. See the [[WEBSOCKETS-API](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-WEBSOCKETS-API)] for more information.

##### bufferedAmount

类型unsigned long, readonly

The ***bufferedAmount*** attribute must return the number of bytes of application data (UTF-8 text and binary data) that have been queued using [send()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-send) but that, as of the last time the event loop started executing a task, had not yet been transmitted to the network. (This thus includes any text sent during the execution of the current task, regardless of whether the user agent is able to transmit text asynchronously with script execution.) This does not include framing overhead incurred by the protocol, or buffering done by the operating system or network hardware. If the channel is closed, this attribute's value will only increase with each call to the [send()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-send) method (the attribute does not reset to zero once the channel closes).

##### label

类型DOMString, 只读

该标签属性用来区分不同的[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象。这个值*必须*为对象创建时设置的值。

##### onclose

类型EventHandler

[close](http://www.w3.org/TR/webrtc/#event-datachannel-close)类型的事件处理器，所有实现[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)接口的类*必须*支持。

##### onerror

类型EventHandler

[error](http://www.w3.org/TR/webrtc/#event-datachannel-error)类型的事件处理器，所有实现[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)接口的类*必须*支持。

##### onmessage

类型EventHandler

[message](http://www.w3.org/TR/webrtc/#event-datachannel-message)类型的事件处理器，所有实现[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)接口的类*必须*支持。

##### onopen

类型EventHandler

[open](http://www.w3.org/TR/webrtc/#event-datachannel-open)类型的事件处理器，所有实现[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)接口的类*必须*支持。

##### readyState

类型[*RTCDataChannelState*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannelState), 只读

该属性表示**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**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boolean, 只读

True表示该**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**是可靠的，反之亦然。这个值*必须*为对象创建时设置的值。

#### 5.2.2 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。

当该方法被调用，用户代理 *必须* 执行下面步骤：

1. 称将要关闭的**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**对象为channel 。

2. 如果channel 的readyState属性为closing或者closed,中止。

3. 设置channel 的readyState属性为closing。

4. 如果[closing procedure](http://dev.w3.org/2011/webrtc/editor/webrtc.html#data-transport-closing-procedure)(5.2.00)没有启动，那么现在启动

没有参数.

返回值类型: void

##### send

按照[send()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-send)算法(5.2.5)执行，参数为string 对象。

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

返回值类型: void

##### send

按照[send()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-send)算法(5.2.5)执行，参数为Blob 对象。

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

返回值类型: void

##### send

按照[send()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-send)算法(5.2.5)执行，参数为ArrayBuffer 对象。

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

返回值类型: void

##### send

按照[send()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-send)算法(5.2.5)执行，参数为ArrayBufferView对象。

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

返回值类型: void

#### 5.2.3 字典RTCDataChannelInit

**reliable**

类型boolean

#### 5.2.4 枚举RTCDataChannelState

enum RTCDataChannelState {

"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 [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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[**RTCDataChannelEvent**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannelEvent)事件派生出(dispatched)的**[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**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。 |

#### 5.2.5 Send算法

Send()重载了不同的参数类型，但不管哪个版本的Send被调用，用户代理都需要执行如下步骤：

1. 命名需要发送数据的[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象称为*channel* 。

2. 如果*channel* 的[readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate)属性是connecting，抛出INVALID\_STATE 异常并中止。

3. 跟着参数类型执行相应的子步骤：

* + string

将参数转换成Unicode编码的字符串*data*，然后[bufferedAmount](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "dom-datachannel-bufferedamount)属性加上*data用*UTF-8表示时的字节数。

* + Blob

将Blob参数变现(how)成原始数据*data*，然后[bufferedAmount](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-bufferedamount)属性加上data的字节大小。

* + ArrayBuffer

将ArrayBuffer 参数命名为*data，*然后[bufferedAmount](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-bufferedamount)属性加上这个数组的字节长度。

* + ArrayBufferView

将ArrayBufferView参数引用的ArrayBuffer片段命名为*data*，然后[bufferedAmount](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-bufferedamount)属性加上该参数的字节长度。

4. 如果*channel* 的底层数据传输还没有建立，或者[closing procedure](http://dev.w3.org/2011/webrtc/editor/webrtc.html#data-transport-closing-procedure)已经开始，中止。

5. 尝试通过*channel* 的底层数据传输通道发送*data*；如果数据不能发送(例如因为缓存数据的buffer满了)，用户代理 *必须* 立即关闭*channel* 的底层数据传输通道(with error)

### [RTCDataChannelEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcdatachannelevent)

* [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)事件使用 [**RTCDataChannelEvent**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannelEvent)接口。
* ***触发一个名为e的***[***RTCDataChannel***](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)***事件***，并将[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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。

dictionary RTCDataChannelEventInit : EventInit {

RTCDataChannel channel;

};

[Constructor(DOMString type, RTCDataChannelEventInit eventInitDict)]

interface RTCDataChannelEvent : Event {

readonly attribute RTCDataChannel channel;

};

#### 5.3.1 属性

##### channel

**类型**[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)**, 只读**

和这个事件相关的那个channel

#### 5.3.2 字典RTCDataChannelEventInit

channel

**类型****[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**

昝略，备注

### 垃圾回收机制

[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象在如下情况下不被垃圾回收：

* [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "dom-datachannel-readystate)为connecting ，并且至少有一个事件监听器注册监听open、message、error或close事件。
* [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate)为open ，并且至少有一个事件监听器注册监听message、error、close事件。
* [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate)为closing ，并且至少有一个事件监听器注册监听error或close事件。
* 底层传输通道已经建立，而且数据正在排队发送。

## [Peer-to-peer DTMF](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-to-peer-dtmf)

为了通过**[RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCPeerConnection)**发送DTMF值(电话键盘)，用户代理需要知道在哪个[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)上的MediaStreamTrack会承载DTMF。这一章节介绍了[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)的扩展接口，该接口描述了MediaStreamTrack上DTMF能力集的问题。

### [RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口扩展

Peer-to-peer DTMF API扩展了[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口，定义如下：

partial interface RTCPeerConnection {

RTCDTMFSender createDTMFSender (MediaStreamTrack track);

};

#### 6.1.1 方法

##### createDTMFSender

该方法以MediaStreamTrack为参数创建了一个RTCDTMFSender对象，MediaStreamTrack *必须* 是对应[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)的[local streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html#local-streams-set)中的MediaStream 的一个元素。如果这个MediaStreamTrack不在其中，需要上抛一个INVALID\_MEDIASTREAMTRACK类型的RTCError异常。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否为空** | **是否可选** | **描述** |
| track | MediaStreamTrack | ✘ | ✘ |  |

Return type:[**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender)

### [RTCDTMFSender](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcdtmfsender)

由createDTMFSender()创建，用以发送DTMF的方法设计如下：

[NoInterfaceObject]

interface RTCDTMFSender {

readonly attribute boolean canInsertDTMF;

void insertDTMF (DOMString tones, optional long duration, optional long interToneGap);

readonly attribute MediaStreamTrack track;

attribute EventHandler ontonechange;

readonly attribute DOMString toneBuffer;

readonly attribute long duration;

readonly attribute long interToneGap;

};

#### 6.2.1 属性

##### canInsertDTMF

type boolean, readonly

The ***canInsertDTMF*** attribute must indicate if the [**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender) is capable of sending DTMF.

##### duration

type long, readonly

The ***duration*** attribute must return the current tone duration value. This value will be the value last set via the [insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF) method, or the default value of 100 ms if [insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF) was called without specifying the duration.

##### interToneGap

type long, readonly

The ***interToneGap*** attribute must return the current value of the between-tone gap. This value will be the value last set via the [insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF) method, or the default value of 50 ms if[insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF) was called without specifying the interToneGap.

##### ontonechange

type EventHandler,

This event handler uses the [**RTCDTMFToneChangeEvent**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFToneChangeEvent) interface to return the character for each tone as it is played out. See [**RTCDTMFToneChangeEvent**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFToneChangeEvent) for details.

##### toneBuffer

type DOMString, readonly

The ***toneBuffer*** attribute must return a list of the tones remaining to be played out. For the syntax, content, and interpretation of this list, see insertDTMF.

##### track

type MediaStreamTrack, readonly

The track attribute must return the MediaStreamTrack given as argument to the createDTMFSender()method.

#### 6.2.2 方法

##### insertDTMF

An [**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender) object’s ***insertDTMF()*** method is used to send DTMF tones.

The tones parameter is treated as a series of characters. The characters 0 through 9, A through D, #, and \* generate the associated DTMF tones. The characters a to d are equivalent to A to D. The character ',' indicates a delay of 2 seconds before processing the next character in the tones parameter. Unrecognized characters are ignored.

The duration parameter indicates the duration in ms to use for each character passed in the tones parameters. The duration cannot be more than 6000 ms or less than 70 ms. The default duration is 100 ms for each tone.

The interToneGap parameter indicates the gap between tones. It must be at least 50 ms. The default value is 50 ms.

ISSUE 4

ISSUE: How are invalid values handled?

When the [insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF) method is invoked, the user agent must run the following steps:

1. If the associated MediaStreamTrack is not connected to the associated [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection), return.
2. If the [canInsertDTMF](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-caninsertdtmf) attribute is false, return.
3. Set the value of the [toneBuffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-tonebuffer) attribute to the value of the tones argument, the value of the [duration](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-duration) attribute to the duration argument if specified, and the value of the[interToneGap](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-intertonegap) to the interToneGap argument, if specified.
4. If [toneBuffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-tonebuffer) is an empty string, return.
5. If a Playout task is scheduled to be run; abort these steps; otherwise queue a task that runs the following steps (Playout task):
   1. If [toneBuffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-tonebuffer) is an empty string, fire an event named [tonechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-RTCDTMFSender-tonechange) with an empty string at the [**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender) object and abort these steps.
   2. Remove the first character from [toneBuffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-tonebuffer) and let that character be tone.
   3. Start playout of tone for [duration](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-duration) ms on the associated RTP media stream, using the appropriate codec.
   4. Queue a task to be executed in [duration](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-duration) + [interToneGap](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-intertonegap) ms from now that runs the steps labelled Playout task.
   5. Fire an event named [tonechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-RTCDTMFSender-tonechange) with a string consisting of tone at the[**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender) object.

Calling [insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF) with an empty tones parameter can be used to cancel all tones queued to play after the currently playing tone.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| tones | DOMString | ✘ | ✘ |  |
| duration | long | ✘ | ✔ |  |
| interToneGap | long | ✘ | ✔ |  |

Return type:void

### [RTCDTMFToneChangeEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcdtmftonechangeevent)

* [tonechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "event-RTCDTMFSender-tonechange)(11.)事件使用的就是[**RTCDTMFToneChangeEvent**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFToneChangeEvent)接口。
* ***触发一个***[tonechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-RTCDTMFSender-tonechange)***类型的名为e的事件***，并将DOMString类型的tone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tone属性被设置为参数tone，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。

[Constructor(DOMString type, RTCDTMFToneChangeEventInit eventInitDict)]

interface RTCDTMFToneChangeEvent : Event {

readonly attribute DOMString tone;

};

#### 6.3.1 属性

##### tone

type DOMString, readonly

The ***tone*** attribute contains the character for the tone that has just begun playout (see[insertDTMF()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-RTCDTMFSender-insertDTMF)). If the value is the empty string, it indicates that the previous tone has completed playback.

## 统计Model

### 介绍

基础的统计模型是，由浏览器根据一个*selector*（选择器\挑选者）来维护一组统计数据。比如，这个selector可能是一个MediaStreamTrack。为了让这个Track是有效的挑选着，这个track必须属于[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)对象接收或发送的media流，并且统计请求已经发布(issued)。发起呼叫的Web应用将这个选择器设置给[getStats()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-getstats)方法，然后浏览器发布(在JS中)一组和这个选择器相关统计数据。

注意：

评估其他比MediaStreamTrack更有需要的目标

The statistics returned are designed in such a way that repeated queries can be linked by the[**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) [id](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-rtcstats-id) dictionary member.因此web应用可以在一个测量周期的开始和结束时请求测量，以便得到测量数据。

### [RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口扩展

Statistics API扩展了[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口，定义如下：

partial interface RTCPeerConnection {

void getStats (MediaStreamTrack? selector, RTCStatsCallback successCallback, RTCPeerConnectionErrorCallback failureCallback);

};

#### 7.2.1 方法

##### getStats

Gathers stats for the given [selector](http://dev.w3.org/2011/webrtc/editor/webrtc.html#stats-selector) and reports the result asynchronously.

When the ***getStats()*** method is invoked, the user agent must queue a task to run the following steps:

1. If the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [RTCPeerConnection signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state) is closed, throw anINVALID\_STATE exception.
2. Return, but continue the following steps in the background.
3. Let selectorArg be the methods first argument.
4. If selectorArg is an invalid [selector](http://dev.w3.org/2011/webrtc/editor/webrtc.html#stats-selector), the user agent must queue a task to invoke the failure callback (the method's third argument).
5. Start gathering the stats indicated by selectorArg. In case selectorArg is null, statsmust be gathered for the whole [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object.
6. When the relevant stats have been gathered, queue a task to invoke the success callback (the method's second argument) with a new [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) object, representing the gathered stats, as its argument.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| selector | MediaStreamTrack | ✔ | ✘ |  |
| successCallback | [**RTCStatsCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsCallback) | ✘ | ✘ |  |
| failureCallback | [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✘ |  |

Return type:void

### [RTCStatsCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcstatscallback)

callback RTCStatsCallback = void (RTCStatsReport report);

#### 7.3.1 参数

**report**

**type**[**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport)

[**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) 表示收集到的统计数据

### [RTCStatsReport 对象](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcstatsreport-object)

The [getStats()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-getstats) method delivers a successful result in the form of a [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) object. A[**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) object represents a map between strings, identifying the inspected objects ([RTCStats.id](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-rtcstats-id)), and their corresponding [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) objects.

An [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) may be composed of several [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) objects, each reporting stats for one underlying object that the implementation thinks is relevant for the [selector](http://dev.w3.org/2011/webrtc/editor/webrtc.html#stats-selector). One achieves the total for the [selector](http://dev.w3.org/2011/webrtc/editor/webrtc.html#stats-selector) by summing over all the stats of a certain type; for instance, if aMediaStreamTrack is carried by multiple SSRCs over the network, the [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) may contain oneRTCStats object per SSRC (which can be distinguished by the value of the "ssrc" stats attribute).

interface RTCStatsReport {

getter RTCStats (DOMString id);

};

#### 7.4.1 方法

##### RTCStats

Getter to retrieve the [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) objects that this stats report is composed of.

The set of supported property names [[WEBIDL](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-WEBIDL)] is defined as the ids of all the [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats)objects that has been generated for this stats report. The order of the property names is left to the user agent.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| id | DOMString | ✘ | ✘ |  |

Return type:getter

### RTCStats字典

An [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) dictionary represents the stats gathered by inspecting a specific object relevant to a[selector](http://dev.w3.org/2011/webrtc/editor/webrtc.html#stats-selector). The [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) dictionary is a base type that specifies as set of default attributes, such as [timestamp](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-rtcstats-timestamp) and [type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-rtcstats-type). Specific stats are added by extending the [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) dictionary.

Note that while stats names are standardized, any given implementation may be using experimental values or values not yet known to the Web application. Thus, applications must be prepared to deal with unknown stats.

NOTE

OPEN ISSUE: Need to define an IANA registry for this and populate with pointers to existing things such as the RTCP statistics.

Statistics need to be synchronized with each other in order to yield reasonable values in computation; for instance, if "bytesSent" and "packetsSent" are both reported, they both need to be reported over the same interval, so that "average packet size" can be computed as "bytes / packets" - if the intervals are different, this will yield errors. Thus implementations must return synchronized values for all stats in a [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) object.

dictionary RTCStats {

DOMHiResTimeStamp timestamp;

RTCStatsType type;

DOMString id;

};

#### 7.5.1 成员

##### id

type DOMString

A unique ***id*** that is associated with the object that was inspected to produce this [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats)object. Two [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) objects, extracted from two different [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) objects, must have the same id if they were produced by inspecting the same underlying object. User agents are free to pick any format for the id as long as it meets the requirements above.

NOTE

Consider naming id something that indicates that the id refers to the underlying object that was inspected to produce the stats, instead of being an id for the JavaScript object. Suggestions: statsObjectId, reporterId, srcId.

##### timestamp

type DOMHiResTimeStamp

The ***timestamp***, of type DOMHiResTimeStamp [[HIGHRES-TIME](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-HIGHRES-TIME)], associated with this object. The time is relative to the UNIX epoch (Jan 1, 1970, UTC).

##### type

type [*RTCStatsType*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsType)

The type of this object.

The ***type*** attribute must be initialized to the name of the most specific type this [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats)dictionary represents.

enum **RTCStatsType** {

"inbound-rtp",

"outbound-rtp"

};

|  |  |
| --- | --- |
| **Enumeration description** | |
| inbound-rtp | Inbound RTP. |
| outbound-rtp | Outbund RTP. |

### [其他派生字典](http://dev.w3.org/2011/webrtc/editor/webrtc.html#derived-stats-dictionaries)

#### 7.6.1 字典RTCRTPStreamStats

dictionary RTCRTPStreamStats : RTCStats {

DOMString ssrc;

DOMString remoteId;

};

remoteId

type DOMString

The remoteId can be used to look up the corresponding [**RTCStats**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStats) object that represents stats reported by the other peer.

ssrc

type DOMString

...

#### 7.6.2 字典RTCInboundRTPStreamStats

dictionary RTCInboundRTPStreamStats : RTCRTPStreamStats {

unsigned long packetsReceived;

unsigned long bytesReceived;

};

**bytesReceived** of type unsigned long

...

**packetsReceived** of type unsigned long

...

#### 7.6.3 RTCOutboundRTPStreamStats

dictionary RTCOutboundRTPStreamStats : RTCRTPStreamStats {

unsigned long packetsSent;

unsigned long bytesSent;

};

**bytesSent** of type unsigned long

...

**packetsSent** of type unsigned long

...

### 示例

试想这样的情形，用户感觉到声音效果不好，应用程序想要判断是否由丢包导致的，那么可能就用到下面的示例代码了：

EXAMPLE 1

var baselineReport, currentReport;

var selector = pc.getRemoteStreams()[0].getAudioTracks()[0];

pc.getStats(selector, function (report) {

baselineReport = report;

});

// ... wait a bit

setTimeout(function () {

pc.getStats(selector, function (report) {

currentReport = report;

processStats();

});

}, aBit);

function processStats() {

// compare the elements from the current report with the baseline

for each (var now in currentReport) {

if (now.type != "outbund-rtp")

continue;

// get the corresponding stats from the baseline report

base = baselineReport[now.id];

if (base) {

remoteNow = currentReport[now.remoteId];

remoteBase = baselineReport[base.remoteId];

var packetsSent = now.packetsSent - base.packetsSent;

var packetsReceived = remoteNow.packetsReceived - remoteBase.packetsReceived;

// if fractionLost is > 0.3, we have probably found the culprit

var fractionLost = (packetsSent - packetsReceived) / packetsSent;

}

}

}

## 身份认证([Identity](http://dev.w3.org/2011/webrtc/editor/webrtc.html#identity))

### 认证交互

WebRTC offers and answers (and hence the channels established by RTCPeerConnection objects) can be authenticated by using web-based Identity Providers. The idea is that the entity sending the offer/answer acts as the Authenticating Party (AP) and obtains an identity assertion from the IdP which it attaches to the offer/answer. The consumer of the offer/answer (i.e., the RTCPeerConnectionon which setRemoteDescription() is called acts as the Relying Party (RP) and verifies the assertion.

The interaction with the IdP is designed to decouple the browser from any particular identity provider; the browser need only know how to load the IdP's JavaScript -- which is deterministic from the IdP's identity -- and the generic protocol for requesting and verifying assertions. The IdP provides whatever logic is necessary to bridge the generic protocol to the IdP's specific requirements. Thus, a single browser can support any number of identity protocols, including being forward compatible with IdPs which did not exist at the time the browser was written. The generic protocol details are described in [RTCWEB-SECURITY-ARCH]. This document specifies the procedures required to instantiate the IdP proxy, request identity assertions, and consume the results.

#### 8.1.1 Peer-Connection/Idp 通讯

In order to communicate with the IdP, the browser must instantiate an isolated interpreted context [TODO: What's the technical term?], such as an invisible IFRAME. The initial contents of the context are loaded from a URI derived from the IdP's domain name. [RTCWEB-SECURITY-ARCH; Section XXX].

For purposes of generating assertions, the IdP shall be chosen as follows:

1. If the setIdentityProvider() method has been called, the IdP provided shall be used.
2. If the setIdentityProvider() method has not been called, then the browser shall use an IdP configured into the browser. If more than one such IdP is configured, the browser should provide the user with a chooser interface.

In order to verify assertions, the IdP domain name and protocol shall be equal to the "domain" and "protocol" fields of the identity assertion.

The context must have a MessageChannel named window.TBD which is "entangled" to the RTCPeerConnection and is unique to that subcontext. This channel is used for messaging between the RTCPeerConnection and the IdP. All messages sent via this channel are strings, specifically the JSONified versions of JavaScript structs.

All messages sent from the RTCPeerConnection to the IdP context must have an origin ofrtcweb://peerconnection/. The fact that ordinary Web pages cannot set their origin values arbitrarily is an essential security feature, as it stops attackers from requesting WebRTC-compatible identity assertions from IdPs. For this reason, the origin must be included in the identity assertion and verified by the consuming RTCPeerConnection.

#### 8.1.2 请求断言

The identity assertion request process involves the following steps.

1. The RTCPeerConnection instantiates an IdP context as described in the previous section.
2. The IdP serves up the IdP JavaScript code to the IdP context.
3. Once the IdP is loaded and ready to receive messages it sends a "READY" message [RTCWEB-SECURITY-ARCH; Section 5.6.5.2]. Note that this does not imply that the user is logged in, merely that enough IdP state is booted up to be ready to handle PostMessage calls.
4. The IdP sends a "SIGN" message (Section 5.6.5.2.2) to the IdP proxy context. This message includes the material the RTCPeerConnection desires to be bound to the user's identity.
5. If the user is not logged in, at this point the IdP will initiate the login process. For instance, it might pop up a dialog box inviting the user to enter their (IdP) username and password.
6. Once the user is logged in (potentially after the previous step), the IdP proxy generates an identity assertion (depending on the authentication protocol this may involve interacting with the IDP server).
7. Once the assertion is generated, the IdP proxy sends a response (Section 5.6.5.2.2) containing the assertion to the RTCPeerConnection over the message channel.
8. The RTCPeerConnection stores the assertion for use with future offers or answers. If the identity request was triggered by a createOffer() or createAnswer(), then the assertion is inserted in the offer/answer.

#### 8.1.3 验证断言

The identity assertion request process involves the following steps.

1. The RTCPeerConnection instantiates an IdP context as described in the previous section.
2. The IdP serves up the IdP JavaScript code to the IdP context.
3. Once the IdP is loaded and ready to receive messages it sends a "READY" message [RTCWEB-SECURITY-ARCH; Section 5.6.5.2]. Note that this does not imply that the user is logged in, merely that enough IdP state is booted up to be ready to handle PostMessage calls.
4. The IdP sends a "VERIFY" message (Section 5.6.5.2.2) to the IdP proxy context. This message includes assertion from the offer/answer which is to be verified.
5. The IdP proxy verifies the identity assertion (depending on the authentication protocol this may involve interacting with the IDP server).
6. Once the assertion is verified the IdP proxy sends a response containing the verified assertion results (Section 5.6.5.2.3) to the RTCPeerConnection over the message channel.
7. The RTCPeerConnection displays the assertion information in the browser UI and stores the assertion in the [peerIdentity](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-peerIdentity) attribute for availability to the JavaScript application. The assertion information to be displayed shall contain the domain name of the IdP and the identity returned by the IdP and must be displayed via some mechanism which cannot be spoofed by content. [[OPEN ISSUE: The identity information should also be available in the inspector interface defined in [RTCWEB-SECURITY-ARCH; Section 5.5].

### [RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口扩展

Identity API扩展了[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)接口，定义如下：

partial interface RTCPeerConnection {

void setIdentityProvider (DOMString provider, optional DOMString protocol, optional DOMString username);

void getIdentityAssertion ();

readonly attribute RTCIdentityAssertion? peerIdentity;

attribute EventHandler onidentityresult;

};

#### 8.2.1 属性

##### onidentityresult

type EventHandler,

This event handler, of event handler event type [identityresult](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-identityresult), must be fired by all objects implementing the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) interface. It is called any time an identity verification succeeds or fails.

##### peerIdentity

type [*RTCIdentityAssertion*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIdentityAssertion), readonly , nullable

Contains the peer identity assertion information if an identity assertion was provided and verified.

#### 8.2.2 方法

##### getIdentityAssertion

Initiates the process of obtaining an identity assertion. Applications need not make this call. It is merely intended to allow them to start the process of obtaining identity assertions before a call is initiated. If an identity is needed, either because the browser has been configured with a default identity provider or because the setIdentityProvider()method was called, then an identity will be automatically requested when an offer or answer is created.

Queue a task to run the following substeps.

1. If the connection's [RTCPeerConnection signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-signaling-state) is closed, abort these steps.
2. Instantiate a new IdP proxy and request an identity assertion.

No parameters.

Return type:void

##### setIdentityProvider

Sets the identity provider to be used for a given PeerConnection object. Applications need not make this call; if the browser is already configured for an IdP, then that configured IdP will be used to get an assertion.

When the ***setIdentityProvider()*** method is invoked, the user agent must run the following steps:

1. Set the current identity values to the triplet (provider, protocol, username).
2. If the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [RTCPeerConnection signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-signaling-state) is stable, and any of the identity settings have changed, queue a task to run the following substeps:
   1. If the connection's [RTCPeerConnection signalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-signaling-state) is closed, abort these steps, and throw an exception with an RTCError object of type INVALID\_STATE.
   2. Instantiate a new IdP proxy and request an identity assertion.
   3. If/when the assertion is obtained, fire a [negotiationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-negotiation) event.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| provider | DOMString | ✘ | ✘ |  |
| protocol | DOMString | ✘ | ✔ |  |
| username | DOMString | ✘ | ✔ |  |

Return type:void

### RTCIdentityAssertion字典

dictionary RTCIdentityAssertion {

DOMString idp;

DOMString name;

};

#### 8.3.1 成员

**idp**

type DOMString

A domain name representing the identity provider.

**name**

type DOMString

An RFC822-conformant [TODO: REF] representation of the verified peer identity. This identity will have been verified via the procedures described in [RTCWEB-SECURITY-ARCH].

### 示例

The identity system is designed so that applications need not take any special action in order for users to generate and verify identity assertions; if a user has configured an IdP into their browser, then the browser will automatically request/generate assertions and the other side will automatically verify them and display the results. However, applications may wish to exercise tighter control over the identity system as shown by the following examples.

This example shows how to configure the identity provider and protocol.

EXAMPLE 2

pc.setIdentityProvider("example.com", "default", "alice@example.com");

This example shows how to consume identity assertions inside a Web application.

EXAMPLE 3

pc.onidentityresult = function(result) {

console.log("IdP= " + pc.peerIdentity.idp +

" identity=" + pc.peerIdentity.name);

};

## Media Stream API扩展(网络应用)

### 介绍

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。

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。

### MediaStream

#### 9.2.1 id

MediaStream中详细描述了id这个属性，id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id属性。

注：MediaStream 对象的id属性对于源stream是唯一标示的，但并不是表示它不能被复制。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id(一个本地生成，一个来自远端)

#### 9.2.2 Events on MediaStream

**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addTrack()方法被 MediaStream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must  运行一下的步骤

1. 目标MediaStream称作stream。

2. 创建一个track来表示即将接收的媒体流，子步骤如下：

1. 创建MediaStreamTrack对象track来表示流

2. 根据媒体流的类型初始化track的kind属性为’audio’或者’video’

3. 初始化track的id属性为媒体流的track id

4. 初始化track的label属性为“remote audio”或者“remote video”。

5. 初始化track的readyState属性为‘muted’。

6. 将track添加到stream的[track set](http://dev.w3.org/2011/webrtc/editor/getusermedia.html" \l "track-set)中。

3. 在stream上触发一个名为[addtrack](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastream-addtrack)的track事件，并将新创建的MediaStreamTrack对象作为其参数

**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：

* [removeTrack()](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-removetrack)方法在本地的  MediaStream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. 目标MediaStream称作stream。
2. 将准备移除的MediaStreamTrack的对象称作track。
3. 将track从stream的[track set](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#track-set)中移除。
4. 在stream上触发一个名叫[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).

### [MediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#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.

### [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastreamevent)

* [**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。

dictionary MediaStreamEventInit : EventInit {

MediaStream stream;

};

[Constructor(DOMString type, MediaStreamEventInit eventInitDict)]

interface MediaStreamEvent : Event {

readonly attribute MediaStream? stream;

};

#### 9.4.1 属性

stream**类型是 MediaStream, 只读, 为空**

和这个事件相关的那个stream

#### 9.4.2 字典[MediaStreamEventInit](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEventInit)

stream**类型是 MediaStream**

## 例子和呼叫流程

这一章是非规范性的。

### 简单Peer-to-peer例子

这一章是非规范性的。

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 4

var signalingChannel = new SignalingChannel();

var configuration = { "iceServers": [{ "url": "stun:stun.example.org" }] };

var pc;

// call start() to initiate

function start() {

pc = new RTCPeerConnection(configuration);

// send any ice candidates to the other peer

pc.onicecandidate = function (evt) {

if (evt.candidate)

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

};

// let the "negotiationneeded" event trigger offer generation

pc.onnegotiationneeded = function () {

pc.createOffer(localDescCreated, logError);

}

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

pc.onaddstream = function (evt) {

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

};

// get a local stream, show it in a self-view and add it to be sent

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

selfView.src = URL.createObjectURL(stream);

pc.addStream(stream);

});

}

function localDescCreated(desc) {

pc.setLocalDescription(desc, function () {

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

}, logError);

}

signalingChannel.onmessage = function (evt) {

if (!pc)

start();

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

if (message.sdp)

pc.setRemoteDescription(new RTCSessionDescription(message.sdp), function () {

// if we received an offer, we need to answer

if (pc.remoteDescription.type == "offer")

pc.createAnswer(localDescCreated, logError);

}, logError);

else

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

};

function logError(error) {

log(error.name + ": " + error.message);

}

### 高级的Peer-to-peer例子

这个例子展示更加复杂的功能.

例子 5

TODO

### Peer-to-peer Data例子

这个例子展示里如何创建一个[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)对象，执行offer/answer交互，完成和对端连接。[**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)应用于一个简单的聊天程序的上下文，当通道准备就绪并收到消息，或者当通道关闭，监听器将被附着到监视器上。

{The [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel) is used in the context of a simple chat application and listeners are attached to monitor when the channel is ready, messages are received and when the channel is closed.}

注意：

这个例子用negotiationneeded事件初始化offer/answer会话。引发negotiationneeded事件的上下文行为目前还没有详细定义。这个例子希望能够促进该问题的讨论。在这个例子里做了一个假设：只有当需要开始协商新的会话，这个事件才会触发。这就意味着在offer/answer会话过程中本该触发negotiationneeded事件的行为(例如addStream())将不再触发。

EXAMPLE 6

var signalingChannel = new SignalingChannel();

var configuration = { "iceServers": [{ "url": "stun:stun.example.org" }] };

var pc;

var channel;

// call start(true) to initiate

function start(isInitiator) {

pc = new RTCPeerConnection(configuration);

// send any ice candidates to the other peer

pc.onicecandidate = function (evt) {

if (evt.candidate)

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

};

// let the "negotiationneeded" event trigger offer generation

pc.onnegotiationneeded = function () {

pc.createOffer(localDescCreated, logError);

}

if (isInitiator) {

// create data channel and setup chat

channel = pc.createDataChannel("chat");

setupChat();

} else {

// setup chat on incoming data channel

pc.ondatachannel = function (evt) {

channel = evt.channel;

setupChat();

};

}

}

function localDescCreated(desc) {

pc.setLocalDescription(desc, function () {

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

}, logError);

}

signalingChannel.onmessage = function (evt) {

if (!pc)

start(false);

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

if (message.sdp)

pc.setRemoteDescription(new RTCSessionDescription(message.sdp), function () {

// if we received an offer, we need to answer

if (pc.remoteDescription.type == "offer")

pc.createAnswer(localDescCreated, logError);

}, logError);

else

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

};

function setupChat() {

channel.onopen = function () {

// e.g. enable send button

enableChat(channel);

};

channel.onmessage = function (evt) {

showChatMessage(evt.data);

};

}

function sendChatMessage(msg) {

channel.send(msg);

}

function logError(error) {

log(error.name + ": " + error.message);

}

### 浏览器间呼叫流程

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。

图见：

<http://dev.w3.org/2011/webrtc/editor/images/ladder-2party-simple.svg>

### DTMF流程

在例子中设定*pc*表示一个RTCPeerConnection，track 表示该连接上的音频track。

用每个音500ms的duration发送DTMF信令“1234“：

EXAMPLE 7

var sender = pc.createDTMFSender(track);

if (sender.canInsertDTMF) {

var duration = 500;

sender.insertDTMF("1234", duration);

} else

log("DTMF function not available");

(假定lightKey("")可以使所有的key失效、熄灭darken) 发送DTMF信令“1234“后，当tone在播放的时候，用lightKey(key)点亮(light up)对应的key：

EXAMPLE 8

var sender = pc.createDTMFSender(track);

sender.ontonechange = function (e) {

if (!e.tone)

return;

// light up the key when playout starts

lightKey(e.tone);

// turn off the light after tone duration

setTimeout(lightKey, sender.duration, "");

};

sender.insertDTMF("1234");

在发送完一个1秒的“1“音之后发送一个2秒的“2“音：

EXAMPLE 9

var sender = pc.createDTMFSender(track);

sender.ontonechange = function (e) {

if (e.tone == "1")

sender.insertDTMF("2", 2000);

};

sender.insertDTMF("1", 1000);

向tone buffer中添加音总是安全的。这个例子中，在任何音播出之前或者播放期间，都可以添加音:

EXAMPLE 10

var sender = pc.createDTMFSender(track);

sender.insertDTMF("123");

// append more tones to the tone buffer before playout has begun

sender.insertDTMF(sender.toneBuffer + "456");

sender.ontonechange = function (e) {

if (e.tone == "1")

// append more tones when playout has begun

sender.insertDTMF(sender.toneBuffer + "789");

};

先发送DTMF信令“123“，然后在发送完”2“之后中止：

EXAMPLE 11

var sender = pc.createDTMFSender(track);

sender.ontonechange = function (e) {

if (e.tone == "2")

// empty the buffer to not play any tone after "2"

sender.insertDTMF("");

};

sender.insertDTMF("123");

## ****事件概要****

这一节是非标准化的(non-normative).

如下的事件在 **[RTCDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "idl-def-RTCDataChannel)**对象上触发:

|  |  |  |
| --- | --- | --- |
| **事件名称** | **接口** | **何时触发…** |
| open | Event | [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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 | [**RTCDataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDataChannel)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://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)对象上触发:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| connecting | Event | TODO |
| addstream | [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEvent) | 一个新的流已经加到 [remote streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "remote-streams-set)中 |
| removestream | [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEvent) | 一个流从[remote streams set](http://dev.w3.org/2011/webrtc/editor/webrtc.html#remote-streams-set)中移除。 |
| 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. |
| signalingstatechange | Event | 可能由于[setLocalDescription()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-setlocaldescription)or[setRemoteDescription()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-setremotedescription)被调用而导致[RTCPeerConnectionsignalingState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-signaling-state)改变。 |
| iceconnectionstatechange | Event | [RTCPeerConnectionice connection state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-ice-connection-state)已经改变。 |
| icecandidate | [RTCPeerConnectionIceEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionIceEvent) | [**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate)被设置为对脚本有效。 |
| identityresult | RTCIdentityEvent | TODO |

如下的事件在 [**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender) 对象上触发:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| ***tonechange*** | Event | The [**RTCDTMFSender**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCDTMFSender) object has either just begun playout of a tone (returned as the [tone](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-tonechangeevent-tone) attribute) or just ended playout of a tone (returned as an empty value in the[tone](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-tonechangeevent-tone) attribute). |

## 安全注意事项

TBD

## 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。

### 13.1 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 7

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。

**RequestIdentity**

是一个枚举类型的约束，它可以取值 “yes”, “no”, and “ifconfigured”。缺省值是非强制性的 “ifconfigured”。

这个约束指定是否需要身份认证 。这个约束可能用于createOffer()或createAnswer ()，亦或者其构造函数。Yes值表示必须需要身份认证；No表示不需要身份识别；ifconfigured的意识是，如果用户在浏览器中配置了身份识别或者在JS中调用了setIdentityProvider()，就需要身份认证。因为默认值为ifconfigured，所以当且仅当用户通过某种方式配置了Idp，才需要身份认证。值得注意的是，随着DTLS-SRTP的使用，不管这个约束值是什么，都将发送指纹识别。

TODO items – 需要在IANA注册。

## ****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 Feb 22, 2013**

Example review: Updated DTMF and Stats examples. Added text about when to fire "negotiationneeded" event to align with examples.

Updated PeerConnection state machine. Added a shared processing model for setLocalDescription()/setRemoteDescription().

Updated simple callflow to match the current API.

**Changes since Jan 16, 2013**

Initial import of Statistics API to version 2.

Integration of Statistics API version 2.5 started.

Updated Statistics API to match Boston/list discussions.

Extracted API extensions introduced by features, such as the P2P Data API, from the PeerConnection API.

Updated DTMF algorithm to dispatch an event when insertDTMF() is called with an empty string to cancel future tones.

Updated DTMF algorithm to not cancel and reschedule if a playout task is running (only update toneBuffer and other values).

**Changes since Dec 12, 2012**

Changed AudioMediaStreamTrack to RTCDTMFSender and gave it its own section. Updated text to reflect most recent agreements. Also added examples section.

Replaced the localStreams and remoteStreams attributes with functions returning sequences of MediaStream objects.

Added spec text for attributes and methods adopted from the WebSocket interface.

Changed the state ENUMs and transition diagrams.

Aligned the data channel processing model a bit more with WebSockets (mainly closing the underlying transport).

**Changes since Nov 13, 2012**

Made some clarifications as to how operation queuing works, and fixed a few errors with the error handling description.

Introduced new representation of tracks in a stream (removed MediaStreamTrackList). Added algorithm for creating a track to represent an incoming network media component.

Renamed MediaStream.label to MediaStream.id (the definition needs some more work).

**Changes since Nov 03, 2012**

Added text describing the queuing mechanism for RTCPeerConnection.

Updated simple P2P example to include all mandatory (error) callbacks.

Updated P2P data example to include all mandatory (error) callbacks. Also added some missing RTC prefixes.

**Changes since Oct 19, 2012**

Clarified how createOffer() and createAnswer() use their callbacks.

Made all failure callbacks mandatory.

Added error object types, general error handling principles, and rules for when errors should be thrown.

**Changes since Sept 23, 2012**

Restructured the document layout and created separate sections for features like Peer-to-peer Data API, Statistics and Identity.

**Changes since Aug 16, 2012**

Replaced stringifier with serializer on RTCSessionDescription and RTCIceCandidate (used when JSON.stringify() is called).

Removed offer and createProvisionalAnswer arguments from the createAnswer() method.

Removed restart argument from the updateIce() method.

Made RTCDataChannel an EventTarget

Updated simple PeerConnection example to match spec changes.

Added section about RTCDataChannel garbage collection.

Added stuff for identity proxy.

Added stuff for stats.

Added stuff peer and ice state reporting.

Minor changes to sequence diagrams.

Added a more complete RTCDataChannel example

Various fixes from Dan's Idp API review.

Patched the Stats API.

**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 definition 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 and Team Contact, Harald Alvestrand, Stefan Håkansson and Dominique Hazaël-Massieux, for their support. Substantial text in this specification was provided by many people including Harald Alvestrand, Justin Uberti, and Eric Rescorla.

## B. 参考

### B.1 Normative references

**[GETUSERMEDIA]**

D. Burnett; A. Narayanan. [Media Capture and Streams](http://www.w3.org/TR/2012/WD-mediacapture-streams-20120628/). 28 June 2012. W3C Working Draft. URL: <http://www.w3.org/TR/2012/WD-mediacapture-streams-20120628/>

**[HIGHRES-TIME]**

Jatinder Mann. [High Resolution Time Specification Editors Draft, Oct 18 2012. URL: http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HighResolutionTime/Overview.html](http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HighResolutionTime/Overview.html)

**[HTML5]**

Robin Berjon et al. [HTML5](http://www.w3.org/TR/html5/). 17 December 2012. W3C Candidate Recommendation. 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. 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. 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. 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. 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/2012/CR-WebIDL-20120419/). 19 April 2012. W3C Candidate Recommendation. URL: <http://www.w3.org/TR/2012/CR-WebIDL-20120419/>

**[WEBSOCKETS-API]**

I. Hickson. [The WebSocket API.](http://www.w3.org/TR/websockets/) W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/websockets/>

### 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/>