# WebRTC Native APIs

**版本2.0 (libjingle r115)**

**2012年2月  
Tuyj译**

**原文：**<http://www.webrtc.org/reference/native-apis>

* WebRTC native APIs文档是基于[WebRTC spec(W3C Editor's Draft)](http://www.w3.org/TR/2012/WD-webrtc-20120821/)文档撰写的.
* 实现WebRTC native APIs的代码(包括Stream 和 PeerConnection APIs) 都可以在开源项目libjingle上找到. 另外在该项目中我们还提供了一个简单的客户端应用.
* 这篇文档的预期读者是想要用WebRTC Native APIs实现WebRTC javascript APIs或者开发本地RTC应用程序的程序员和工程师们。

## 目录

[WebRTC Native APIs 1](#_Toc353115383)

[目录 1](#_Toc353115384)

[1. 新的内容 2](#_Toc353115385)

[2. 模块架构图 2](#_Toc353115386)

[3. 调用序列 3](#_Toc353115387)

[3.1 发起通话 3](#_Toc353115388)

[3.2 接收通话 3](#_Toc353115389)

[3.3 结束通话 4](#_Toc353115390)

[4. 线程模型 4](#_Toc353115391)

[5. Stream APIs (mediastream.h) 4](#_Toc353115392)

[5.1 类 MediaStreamTrackInterface 4](#_Toc353115393)

[5.2 类 VideoTrackInterface 7](#_Toc353115394)

[5.3 类 LocalVideoTrackInterface 7](#_Toc353115395)

[5.4 类 AudioTrackInterface 7](#_Toc353115396)

[5.5 类 LocalAudioTrackInterface 7](#_Toc353115397)

[5.6 类 cricket::VideoRenderer, cricket::VideoCapturer 8](#_Toc353115398)

[5.7 类 webrtc::AudioDeviceModule 8](#_Toc353115399)

[5.8 类 MediaStreamInterface 8](#_Toc353115400)

[5.9 类 LocalMediaStreamInterface 9](#_Toc353115401)

[6. PeerConnection APIs (peerconnection.h) 10](#_Toc353115402)

[6.1 类 StreamCollectionInterface 10](#_Toc353115403)

[6.2 类 PeerConnectionObserver 11](#_Toc353115404)

[6.3 类 PortAllocatorFactoryInterface 13](#_Toc353115405)

[6.4 类 PeerConnectionFactoryInterface 14](#_Toc353115406)

[6.5 函数 CreatePeerConnectionFactory 17](#_Toc353115407)

[6.6 函数 CreatePeerConnectionFactory 17](#_Toc353115408)

[6.7 类 PeerConnectionInterface 17](#_Toc353115409)

[7.引用 22](#_Toc353115410)

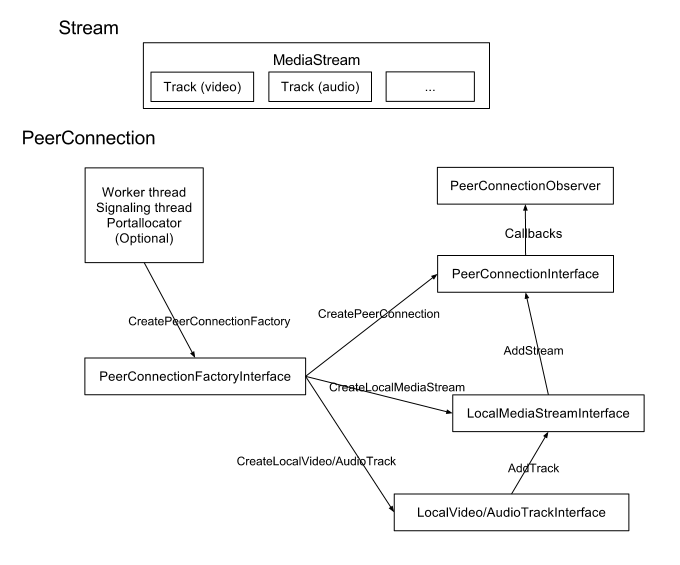
## 新的内容

与WebRTC native APIs的上一个版本 (代码在[这里](https://code.google.com/p/libjingle/source/browse/" \l "svn%2Ftrunk%2Ftalk%2Fapp%2Fwebrtcv1)，开源项目libjingle r115的一部分)相比, 主要不同是新的版本包括了Stream APIs的实现。使用[Stream APIs](http://dev.w3.org/2011/webrtc/editor/webrtc.html" \l "stream-api), 音视频媒体会在MediaTrack对象中处理，而不是直接交给PeerConnection处理。并且现在PeerConnection在调用中能够接收和返回MediaStream对象而不是像先前版本中只能处理媒体流标签。

除此之外，在这个版本中信令处理协议改用了[ROAP](http://tools.ietf.org/html/draft-jennings-rtcweb-signaling-01)，然而对使用这些APIs的用户来说是透明的。

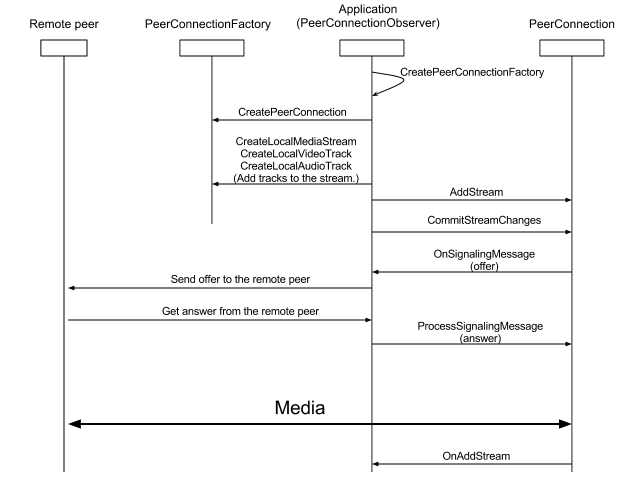
如果你开发的应用是基于以前的版本，请查看这个[补丁](http://codereview.chromium.org/9284020)来把你的应用迁移到新的APIs来。上面提到的简单的客户端应用也可以作为参考。

## 模块架构图

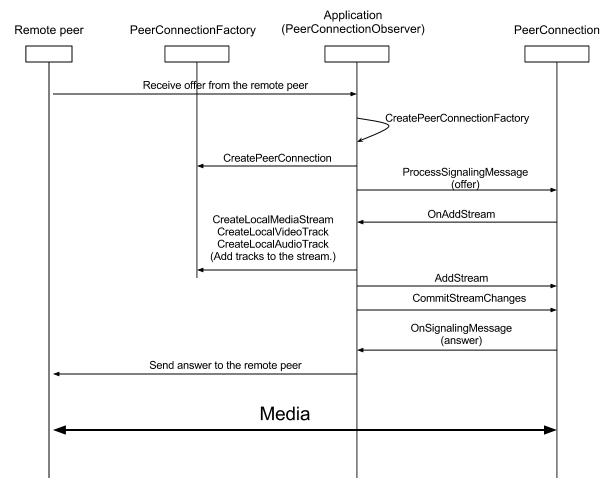


## 调用序列

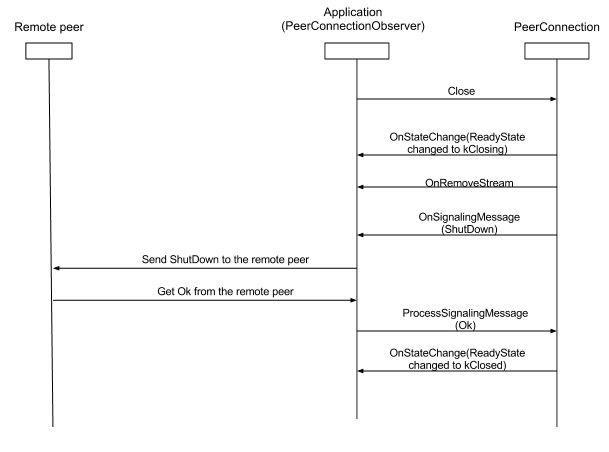
### 发起通话



### 接收通话



### 结束通话



## 线程模型

WebRTC native APIs 拥有两个全局线程：信令线程（signaling thread）和工作者线程（worker thread）。根据PeerConnection factory被创建的方式，应用程序可以提供这两个线程或者直接使用内部创建好的线程。(either provide those 2 threads or just let them be created internally)

Stream APIs和PeerConnection APIs的调用会被**代理到信令线程**，这就意味着应用程序可以在**任何线程调用这些APIs**。

**所有的回调函数都在信令线程调用。**应用程序应当尽快地跳出回调函数以避免阻塞信令线程。严重消耗资源的过程都应当其他的线程执行。

工作者线程被用来处理资源消耗量大的过程，比如说数据流传输。

## 5. Stream APIs (mediastream.h)

### 5.1 类 MediaStreamTrackInterface

这个类声明了一个媒体流传输的抽象接口，用来代表用户代理（UA）的媒体源。

class MediaStreamTrackInterface : public talk\_base::RefCountInterface,  
                                 public NotifierInterface {  
public:  
 enum TrackState {  
   kInitializing,  
   kLive = 1,  
   kEnded = 2,  
   kFailed = 3,  
 };  
 virtual std::string kind() const = 0;  
 virtual std::string label() const = 0;  
 virtual bool enabled() const = 0;  
 virtual TrackState state() const = 0;  
 virtual bool set\_enabled(bool enable) = 0;  
 virtual bool set\_state(TrackState new\_state) = 0;  
};

#### 5.1.1 enum TrackState

这个枚举类型定义了传输轨道的状态。

语法：

enum TrackState {  
   kInitializing,  
   kLive = 1,  
   kEnded = 2,  
   kFailed = 3,  
 };

kInitializing - 传输正在协商.  
kLive - 传输轨道可用.  
kEnded - 传输轨道关闭.  
kFailed - 传输协商失败.

#### 5.1.2 method kind

如果是传输音频轨道则返回"audio”，如果传输视频轨道则返回"video" ，或者用户代理定义的其他字符串。

语法：

virtual std::string kind() const = 0;

#### 5.1.3 method label

如果有的话返回轨道的标签，否则返回空串。

语法:

virtual std::string label() const = 0;

#### 5.1.4 method enabled

如果轨道可用返回"true”，否则返回"false”。

语法:

virtual bool enabled() const = 0;

#### 5.1.5 method state

返回轨道的当前状态。

语法:

virtual TrackState state() const = 0;

#### 5.1.6 method set\_enabled

开启(true) 或者关闭(false) 媒体管道.

语法:

virtual bool set\_enabled(bool enable) = 0;

备注:未实现

#### 5.1.7 method set\_state

设置媒体管道的状态。

语法:

virtual bool set\_state(TrackState new\_state) = 0;

备注:这个方法应该被PeerConnection内部调用，应用程序不应当直接调用。

### 5.2 类 VideoTrackInterface

类VideoTrackInterface继承自类MediaStreamTrackInterface，额外新增了设置和得到视频渲染器的两个接口。

class VideoTrackInterface : public MediaStreamTrackInterface {  
public:  
 virtual void SetRenderer(VideoRendererWrapperInterface\* renderer) = 0;  
 virtual VideoRendererWrapperInterface\* GetRenderer() = 0;  
protected:  
 virtual ~VideoTrackInterface() {}

};

### 5.3 类 LocalVideoTrackInterface

类LocalVideoTrackInterface继承自类VideoTrackInterface，增加了一个接口得到视频捕获设备。

class LocalVideoTrackInterface : public VideoTrackInterface {

public:

 virtual cricket::VideoCapturer\* GetVideoCapture() = 0;

protected:

 virtual ~LocalVideoTrackInterface() {}

};

### 5.4 类 AudioTrackInterface

类AudioTrackInterface继承自类MediaStreamTrackInterface，目前没有其他接口。

class AudioTrackInterface : public MediaStreamTrackInterface {

public:

protected:

 virtual ~AudioTrackInterface() {}

};

### 5.5 类 LocalAudioTrackInterface

类LocalAudioTrackInterface继承自AudioTrackInterface，增加了一个接口来得到音频设备。

class LocalAudioTrackInterface : public AudioTrackInterface {

public:

 virtual AudioDeviceModule\* GetAudioDevice() =  0;

protected:

 virtual ~LocalAudioTrackInterface() {}

};

### 5.6 类 cricket::VideoRenderer, cricket::VideoCapturer

这些类定义在开源工程libjingle，在这里不再详细陈述。

### 5.7 类 webrtc::AudioDeviceModule

类AudioDeviceModule定义在开源工程[webrtc](https://code.google.com/p/webrtc/). 请点击链接参考详细定义。(即webrtc/source/browse/trunk/src/modules/audio\_device/main/interface/audio\_device.h)

### 5.8 类 MediaStreamInterface

这个类声明了一个MediaStream的抽象接口，该类典型但不是必须的，表示视音频数据流。

每一个MediaStream对象可以包含零到多个轨道，尤其视音/频轨道。MediaStream对象的所有媒体轨道在渲染的时候必须是同步的。不同的MediaStream对象不必同步。

class MediaStreamInterface : public talk\_base::RefCountInterface,

                            public NotifierInterface {

public:

 virtual std::string label() const = 0;

 virtual AudioTracks\* audio\_tracks() = 0;

 virtual VideoTracks\* video\_tracks() = 0;

 enum ReadyState {

   kInitializing,

   kLive = 1,  // Stream alive

   kEnded = 2,  // Stream have ended

 };

 virtual ReadyState ready\_state() = 0;

protected:

 virtual ~MediaStreamInterface() {}

};

#### 5.8.1 method label

返回这个媒体流的唯一标签，目的是在这些媒体流通过PeerConnection APIs传输后能够被区别清楚。

语法

virtual std::string label() const = 0;

#### 5.8.2 method audio\_tracks

返回当前MediaStream对象的音频流轨道列表的指针(return a pointer to a list)

语法

virtual AudioTracks\* audio\_tracks() = 0;

#### 5.8.3 method video\_tracks

返回当前MediaStream对象的视频流轨道列表的指针(return a pointer to a list)

语法

virtual VideoTracks\* video\_tracks() = 0;

#### 5.8.4 method ready\_state

返回当前MediaStream的状态是否就绪。

语法

virtual ReadyState ready\_state() = 0;

### 5.9 类 LocalMediaStreamInterface

类LocalMediaStreamInterface继承自类MediaStreamInterface，新增两个接口添加视音频轨道。

class LocalMediaStreamInterface : public MediaStreamInterface {  
public:  
 virtual bool AddTrack(AudioTrackInterface\* track) = 0;  
 virtual bool AddTrack(VideoTrackInterface\* track) = 0;

};

## 6. PeerConnection APIs (peerconnection.h)

### 6.1 类 StreamCollectionInterface

这个类定义了一个MediaStream容器接口。

class StreamCollectionInterface : public talk\_base::RefCountInterface {  
public:  
 virtual size\_t count() = 0;  
 virtual MediaStreamInterface\* at(size\_t index) = 0;  
 virtual MediaStreamInterface\* find(const std::string& label) = 0;  
protected:  
 ~StreamCollectionInterface() {}  
};

#### 6.1.1 method count

返回MediaStreams集合的数量。

语法

size\_t count() = 0;

#### 6.1.2 method at

返集合中指定位置的MediaStream对象指针。

语法

MediaStreamInterface\* at(size\_t index) = 0;

参数

index [in] Position of a MediaStream in the collection.

#### 6.1.3 method find

用标签查询MediaStream对象，如果找到返回对象指针，否者返回NULL。

语法

MediaStreamInterface\* find(const std::string& label) = 0;

参数

label [in] The label value to be searched for.

### 6.2 类 PeerConnectionObserver

这个类为用户定义的观察中(observer)声明了一个抽象接口。它取决于PeerConnection用户实现的子类。当用PeerConnectionFactoryInterface创建PeerConnection的时候会注册observer对象。

class PeerConnectionObserver {  
public:  
 enum StateType {  
   kReadyState,  
   kIceState,  
   kSdpState,  
 };  
 virtual void OnError() = 0;  
 virtual void OnMessage(const std::string& msg) = 0;  
 virtual void OnSignalingMessage(const std::string& msg) = 0;  
 virtual void OnStateChange(StateType state\_changed) = 0;  
 virtual void OnAddStream(MediaStreamInterface\* stream) = 0;  
 virtual void OnRemoveStream(MediaStreamInterface\* stream) = 0;  
protected:  
 ~PeerConnectionObserver() {}  
};

#### 6.2.1 enum StateType

这个枚举定义了状态机状态。

语法

enum StateType {  
   kReadyState,  
   kIceState,  
   kSdpState,  
 };

#### 6.2.2 method OnError

当PeerConnection执行中出错时调用此方法。

语法

void OnError() = 0;

备注:尚未实现。

#### 6.2.3 method OnMessage

当收到对端的一条文本消息时此方法被调用。

语法

void OnMessage(const std::string& msg) = 0;

#### 6.2.4 method OnSignalingMessage

当准备发送信令消息时调用此函数(This method is called once a signaling message is ready.)

语法

void OnSignalingMessage(const std::string& msg) = 0;

参数

msg [in] A ROAPformat signaling message.

备注:用户应当从回调函数向对端发送信令。

#### 6.2.5 method OnStateChange

这个方法当状态机状态(ReadyState, SdpState or IceState)改变时被调用。

语法

virtual void OnStateChange(StateType state\_changed) = 0;

参数

state\_changed [in] Specify which state machine’s state has changed.

备注:IceState尚未实现。

#### 6.2.6 method OnAddStream

当从对端收到新的媒体流时该方法被调用。

语法

virtual void OnAddStream(MediaStreamInterface\* stream) = 0;

参数

stream [in] The handler to the remote media stream.

备注:用户可以用这个事件为收到的媒体流设置渲染器。

#### 6.2.7 method OnRemoveStream

当对端关闭媒体流时调用此方法。

语法

virtual void OnRemoveStream(MediaStreamInterface\* stream) = 0;

参数

stream [in] The handler to the closed remote media stream.

### 6.3 类 PortAllocatorFactoryInterface

这个类声明了一个工厂接口来创建cricket::PortAllocator对象(用来做ICE协商)。PeerConnectionFactory使用这个接口（如果提供）为自己创建的PeerConnection创建PortAllocator。应用程序也可以通过实现PortAllocatorFactoryInterface(主要是CreatePortAllocator方法)来提供自己的PortAllocator。

class PortAllocatorFactoryInterface : public talk\_base::RefCountInterface {  
public:  
 struct StunConfiguration {  
   StunConfiguration(const std::string& address, int port)  
       : server(address, port) {}  
   talk\_base::SocketAddress server;  
 };  
 struct TurnConfiguration {  
   TurnConfiguration(const std::string& address,  
                     int port,  
                     const std::string& user\_name,  
                     const std::string& password)  
       : server(address, port),  
         username(username),  
         password(password) {}  
   talk\_base::SocketAddress server;  
   std::string username;  
   std::string password;  
 };  
 virtual cricket::PortAllocator\* CreatePortAllocator(  
     const std::vector<StunConfiguration>& stun\_servers,  
     const std::vector<TurnConfiguration>& turn\_configurations) = 0;  
protected:  
 PortAllocatorFactoryInterface() {}  
 ~PortAllocatorFactoryInterface() {}  
};

#### 6.3.1 method CreatePortAllocator

这个方法返回PortAllocator类的实例.

语法

 virtual cricket::PortAllocator\* CreatePortAllocator(  
     const std::vector<StunConfiguration>& stun\_servers,  
     const std::vector<TurnConfiguration>& turn\_configurations) = 0;

参数

stun\_servers [in] A configuration list of the STUN servers.

turn\_servers [in] A configuration list of the TURN servers.

备注:TURN 尚未实现。

### 6.4 类 PeerConnectionFactoryInterface

PeerConnectionFactoryInterface是一个工厂接口用来创建PeerConnection对象、媒体流、媒体轨道。

class PeerConnectionFactoryInterface : public talk\_base::RefCountInterface {  
public:  
 virtual talk\_base::scoped\_refptr<PeerConnectionInterface>  
     CreatePeerConnection(const std::string& config,  
                          PeerConnectionObserver\* observer) = 0;  
 virtual talk\_base::scoped\_refptr<LocalMediaStreamInterface>  
     CreateLocalMediaStream(const std::string& label) = 0;  
 virtual talk\_base::scoped\_refptr<LocalVideoTrackInterface>  
     CreateLocalVideoTrack(const std::string& label,  
                           cricket::VideoCapturer\* video\_device) = 0;  
 virtual talk\_base::scoped\_refptr<LocalAudioTrackInterface>  
     CreateLocalAudioTrack(const std::string& label,  
                           AudioDeviceModule\* audio\_device) = 0;  
protected:  
 PeerConnectionFactoryInterface() {}  
 ~PeerConnectionFactoryInterface() {}  
};

#### 6.4.1 method CreatePeerConnection

该方法创建PeerConnection类的实例。

语法

virtual talk\_base::scoped\_refptr<PeerConnectionInterface>  
     CreatePeerConnection(const std::string& config,  
                        PeerConnectionObserver\* observer) = 0;

参数

config [in] STUN或TURN服务器地址，用来建立连接的配置字符串。格式定义参考[webrtc-api](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-to-peer-connections)。

observer [in] 继承PeerConnectionObserver类的实例的指针。

备注:

**可接受的配置字符串格式：**

"TYPE 203.0.113.2:3478"

     指定服务器的IP和端口号。

"TYPE relay.example.net:3478

     表明服务器的域名和端口号，用户代理会从DNS查询对应IP。

"TYPE example.net"

     表明服务器的域名，用户代理会从DNS查询对应IP和端口。

**类型"TYPE"可以是一下的一种:**

STUN 表明一个STUN服务器。

STUNS表明一个STUN 服务器并用TLS会话连接。

TURN表明一个TURN服务器。

TURNS表明一个TURN服务器并用TLS会话连接。

#### 6.4.2 method CreateLocalMediaStream

创建本端媒体流的实例。

语法

 virtual talk\_base::scoped\_refptr<LocalMediaStreamInterface>  
     CreateLocalMediaStream(const std::string& label) = 0;

参数

label [in] Desired local media stream label.

#### 6.4.3 method CreateLocalVideoTrack

创建本端视频轨道的对象。

语法

virtual talk\_base::scoped\_refptr<LocalVideoTrackInterface>  
     CreateLocalVideoTrack(const std::string& label,  
                           cricket::VideoCapturer\* video\_device) = 0;

参数

label [in] Desired local video track label.  
video\_device [in] Pointer to the video capture device that is going to associate with this track.

#### 6.4.4 method CreateLocalAudioTrack

创建本端音频轨道的对象。

语法

virtual talk\_base::scoped\_refptr<LocalVideoTrackInterface>  
     CreateLocalAudioTrack(const std::string& label,  
                           AudioDeviceModule\* audio\_device) = 0;

参数

label [in] Desired local audio track label.  
audio\_device [in] Pointer to the audio device that is going to associate with this track.

### 6.5 函数 CreatePeerConnectionFactory

创建PeerConnectionFactoryInterface实例。

语法

talk\_base::scoped\_refptr<PeerConnectionFactoryInterface>  
CreatePeerConnectionFactory();

备注:通过该函数生成的PeerConnectionFactoryInterface对象会创建必要的内部资源，包括libjingle线程、socket，网络部分的network manager factory。

### 6.6 函数 CreatePeerConnectionFactory

用给定的libjingle线程和portallocatoFactory创建PeerConnectionFactoryInterface实例。

语法

talk\_base::scoped\_refptr<PeerConnectionFactoryInterface>  
CreatePeerConnectionFactory(talk\_base::Thread\* worker\_thread,  
                           talk\_base::Thread\* signaling\_thread,  
                           PortAllocatorFactoryInterface\* factory,  
                           AudioDeviceModule\* default\_adm);

备注

当应用程序想要使用自己实现的线程和portallocator对象时调用该方法。

这些参数的所有权没有转移到新建的PeerConnectionFactoryInterface对象上，必须在PeerConnectionFactoryInterface的生命周期范围内(使用)。

### 6.7 类 PeerConnectionInterface

class PeerConnectionInterface : public talk\_base::RefCountInterface {  
public:  
 enum ReadyState {  
   kNew,  
   kNegotiating,  
   kActive,  
   kClosing,  
   kClosed,  
 };  
 enum SdpState {  
   kSdpNew,  
   kSdpIdle,  
   kSdpWaiting,  
 };  
 virtual void ProcessSignalingMessage(const std::string& msg) = 0;  
 virtual bool Send(const std::string& msg) = 0;  
 virtual talk\_base::scoped\_refptr<StreamCollectionInterface>  
     local\_streams() = 0;  
 virtual talk\_base::scoped\_refptr<StreamCollectionInterface>  
     remote\_streams() = 0;  
 virtual void AddStream(LocalMediaStreamInterface\* stream) = 0;  
 virtual void RemoveStream(LocalMediaStreamInterface\* stream) = 0;  
 virtual void CommitStreamChanges() = 0;  
 virtual void Close() = 0;  
 virtual ReadyState ready\_state() = 0;  
 virtual SdpState sdp\_state() = 0;  
protected:  
 ~PeerConnectionInterface() {}  
};

#### 6.7.2 enum ReadyState

该枚举定义了就绪状态的几种类别状态。

* kNew - 对象刚被创建并且他的ICE和SDP代理尚未启动。
* kNegotiating - peerConenction 对象正在尝试联系传输媒体的一端。
* kActive - 连接建立成功，如果任何媒体流协商成功，相关的媒体就可以传输了。
* kClosing - close()方法被调用后对象正在被关闭。
* kClosed - 对象关闭成功。

语法  
enum ReadyState {  
   kNew,  
   kNegotiating,  
   kActive,  
   kClosing,  
   kClosed,  
};

#### 6.7.2 enum SdpState

该枚举定义了SDP的状态。

* kSdpNew - 对象刚被创建，SDP代理尚未开始。
* kSdpIdle - 有效交互了请求/应答，SDP代理正在等地下一个SDP事务。
* kSdpWaiting - SDP代理发送了一个SDP请求正在等待响应。

语法

enum SdpState {

   kSdpNew,  // TODO(ronghuawu): kSdpNew is not defined in the spec.

   kSdpIdle,

   kSdpWaiting,

};

#### 6.7.3 method ProcessSignalingMessage

该方法用来处理来自对端的信令。

语法

virtual void ProcessSignalingMessage(const std::string& msg) = 0;

参数

msg[in] A ROAPformat signaling message.

备注:信令的顺序是很重要的，如果传输(返回?)信令与对端产生信令的顺序不同的话会使会话失去连接或降低会话质量。

#### 6.7.4 method Send

用方法可以通过data流给对端发送一个文本信息。

语法

virtual bool Send(const std::string& msg) = 0;  // TODO(ronghuawu): This is not defined in the spec.

参数

msg[in] The text message to be sent to the remote peer.

备注：尚未实现。

#### 6.7.5 method local\_streams

返回用户代理正在尝试发往远端的媒体流数组（由方法AddStream()添加的流）。

语法

virtual talk\_base::scoped\_refptr<StreamCollectionInterface>

     local\_streams() = 0;

#### 6.7.6 mothod remote\_streams

返回用户代理当前正在从对端接收的媒体流数组。

当OnAddStream和OnRemoveStream回调函数被调用时这个数组被更新。

语法

virtual talk\_base::scoped\_refptr<StreamCollectionInterface>

     remote\_streams() = 0;

#### 6.7.7 mthod AddStream

添加一个本地媒体流到发送媒体流数组。这个函数只是添加媒体流并不触发任何变化直到CommitStreamChanges方法被调用。

语法

virtual void AddStream(LocalMediaStreamInterface\* stream) = 0;

参数

stream[in] Pointer to the local media stream to be added.

#### 6.7.8 method RemoveStream

从本地发送媒体流数组中删除指定媒体流。这个函数只是删除媒体流并不触发任何变化直到CommitStreamChanges方法被调用。

语法

virtual void RemoveStream(LocalMediaStreamInterface\* stream) = 0;

参数

stream[in] Pointer to the local media stream to be removed.

#### 6.7.9 method CommitStreamChanges

提交AddStream和RemoveStream调用后造成的媒体流的变化。新媒体流被添加后开始发送媒体，媒体流被移除后停止发送媒体流。

语法

virtual void CommitStreamChanges() = 0;

#### 6.7.10 method Close

关闭当前的会话。这会触发发送一个Shutdown消息并且ready\_state会切换到kClosing。

语法

virtual void Close() = 0;

#### 6.7.11 method ready\_state

返回PeerConnection对象的readiness state， 由枚举类型ReadyState表示。

语法

virtual ReadyState ready\_state() = 0;

#### 6.7.12 method sdp\_state

返回PeerConnection SDP代理的状态， 由枚举类型SdpState表示。

语法

virtual SdpState sdp\_state() = 0;

## 7.引用

目前[HTML5](http://bbs.189works.com/)的WebRTC规范说明:

http://dev.w3.org/2011/webrtc/editor/webrtc.html

WebRTC Native API 的源码:

https://code.google.com/p/libjingle/source/browse/trunk/talk/app/webrtc/

客户端和服务器Demo:

https://code.google.com/p/libjingle/source/browse/trunk/talk/#talk%2Fexamples%2Fpeerconnection