WebRTC native APIs文档 是基于 WebRTC spec 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。 内容

∙ 1 新的内容 ∙ 2 模块架构图 ∙

3 调用序列

o 3.1 发起通话 o 3.2 接收通话 o 3.3 结束通话 ∙ 4 线程模型

∙

5 Stream APIs (mediastream.h)

o 5.1 类 MediaStreamTrackInterface o 5.2 类 VideoTrackInterface

o 5.3 类 LocalVideoTrackInterface o 5.4 类 AudioTrackInterface

o 5.5 类 LocalAudioTrackInterface

o 5.6 类 cricket::VideoRenderer, cricket::VideoCapturer o 5.7 类 webrtc::AudioDeviceModule o 5.8 类 MediaStreamInterface

o 5.9 类 LocalMediaStreamInterface ∙

6 PeerConnection APIs (peerconnection.h)

o 6.1 类 StreamCollectionInterface o 6.2 类 PeerConnectionObserver

o 6.3 类 PortAllocatorFactoryInterface o 6.4 类 PeerConnectionFactoryInterface o 6.5 函数 CreatePeerConnectionFactory o 6.6 函数 CreatePeerConnectionFactory o 6.7 类 PeerConnectionInterface ∙

7 引用

新的内容

与WebRTC native APIs的上一个版本 (代码在这里，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, 视音频媒体会在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使用的协议改用了ROAP ，这对APIs 的用户应当是透明的。

如果你开发的应用是基于以前的版本，请查看这个补丁来把你的应用迁移大新的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

接受通话

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 ）。取决于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。

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。

Stream APIs (mediastream.h)

Class 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; };

MediaStreamTrackInterface::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, }; 备注

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.

MediaStreamTrackInterface::kind

如果传输音频轨道返回"audio”，如果传输视频轨道则返回"video" ，或者一个用户代理定义的其他字符串。 语法

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

MediaStreamTrackInterface::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;

MediaStreamTrackInterface::enabled

如果轨道可用返回"true”，否则返回"false”. 语法

virtual bool enabled() const = 0;

MediaStreamTrackInterface::state

返回轨道的当前状态。 语法

virtual TrackState state() const = 0;

MediaStreamTrackInterface::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

Not implemented.

MediaStreamTrackInterface::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。

Class VideoTrackInterface

类VideoTrackInterface 继承自类MediaStreamTrackInterface ，增加了两个设置和得到视频渲染器的接口。

class VideoTrackInterface : public MediaStreamTrackInterface { public:

virtual void SetRenderer(VideoRendererWrapperInterface\* renderer) = 0; virtual VideoRendererWrapperInterface\* GetRenderer() = 0; protected:

virtual ~VideoTrackInterface() {} };

Class 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() {} };

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

class AudioTrackInterface : public MediaStreamTrackInterface { public: protected:

virtual ~AudioTrackInterface() {} };

Class 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() {} };

Class cricket::VideoRenderer, cricket::VideoCapturer

这些类定义在开源工程libjingle ，在这里不再详细陈述。

Class webrtc::AudioDeviceModule

类AudioDeviceModule 定义在开源工程webrtc. 请点击链接参考详细定义。

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

每一个MediaStream 对象可以包含零到多个音频轨道，尤其视音频轨道。在一个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() {} };

MediaStreamInterface::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;

MediaStreamInterface::audio\_tracks

返回MediaStreamTrack 对象列表的指针，代表与当前MediaStream 对象相关的音频流轨道列表。 语法

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

MediaStreamInterface::video\_tracks

返回MediaStreamTrack 对象列表的指针，代表与当前MediaStream 对象相关的视频流轨道列表。 语法

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

MediaStreamInterface::ready\_state

返回当前MediaStream 的状态是否就绪。 语法

virtual ReadyState ready\_state() = 0;

Class LocalMediaStreamInterface

类LocalMediaStreamInterface 继承自类MediaStreamInterface ，定义了两个方法添加视音频轨道。

class LocalMediaStreamInterface : public MediaStreamInterface { public:

virtual bool AddTrack(AudioTrackInterface\* track) = 0; virtual bool AddTrack(VideoTrackInterface\* track) = 0; } ;

PeerConnection APIs (peerconnection.h)

Class 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() {} };

StreamCollectionInterface::count

返回MediaStreams 集合的数量。 语法

size\_t count() = 0;

StreamCollectionInterface::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.

StreamCollectionInterface::find

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

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

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

Class PeerConnectionObserver

这个类为用户定义的observer 声明了一个抽象接口。它取决于PeerConnection 用户实现的子类。当PeerConnection 被创建的时候用

PeerConnectionFactoryInterface 类注册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() {} };

PeerConnectionObserver::StateType

这个枚举定义了状态机状态。 语法

enum StateType { kReadyState, kIceState, kSdpState, };

PeerConnectionObserver::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。

PeerConnectionObserver::OnMessage

当收到对端的一条文本消息是此方法被调用。 语法

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

PeerConnectionObserver::OnSignalingMessage

当收到信令是调用此方法。 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。

The user should send the signaling message from the callback to the remote peer.

PeerConnectionObserver::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。

PeerConnectionObserver::OnAddStream 该方法当从对端收到新的媒体流时被调用。 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。

PeerConnectionObserver::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.

Class PortAllocatorFactoryInterface

这 个类声明了一个工厂接口来创建cricket::PortAllocator对象，该对象用来做ICE 协商。PeerConnection 工厂使用这个接 口（如果提供）为自己创建PortAllocator 。应用程序也可以提供自己的PortAllocator 实现，只要实现了PortAllocatorFactoryInterface 的CreatePortAllocator 方法。 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& stun\_servers,

const std::vector& turn\_configurations) = 0; protected:

PortAllocatorFactoryInterface() {} ~PortAllocatorFactoryInterface() {} };

PortAllocatorFactoryInterface::CreatePortAllocator

这个方法返回PortAllocator 类的实例. 语法

virtual cricket::PortAllocator\* CreatePortAllocator( const std::vector& stun\_servers,

const std::vector& 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。

Class PeerConnectionFactoryInterface

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

class PeerConnectionFactoryInterface : public talk\_base::RefCountInterface { public:

virtual talk\_base::scoped\_refptr

CreatePeerConnection(const std::string& config,

PeerConnectionObserver\* observer) = 0;

virtual talk\_base::scoped\_refptr

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

CreateLocalVideoTrack(const std::string& label,

cricket::VideoCapturer\* video\_device) = 0;

virtual talk\_base::scoped\_refptr

CreateLocalAudioTrack(const std::string& label,

AudioDeviceModule\* audio\_device) = 0; protected:

PeerConnectionFactoryInterface() {} ~PeerConnectionFactoryInterface() {} };

PeerConnectionFactoryInterface::CreatePeerConnection

该方法创建PeerConnection 类的实例。 语法

virtual talk\_base::scoped\_refptr

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 。

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

备注

TURN 尚未支持。 接受的配置字符串： "TYPE 203.0.113.2:3478"

表名服务器的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"

表明服务器的域名和端口号，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。

PeerConnectionFactoryInterface::CreateLocalMediaStream

创建本端媒体流的实例。 语法

virtual talk\_base::scoped\_refptr

CreateLocalMediaStream(const std::string& label) = 0; 参数

label [in] Desired local media stream label.

PeerConnectionFactoryInterface::CreateLocalVideoTrack

创建本端视频轨道的对象。 语法

virtual talk\_base::scoped\_refptr

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.

PeerConnectionFactoryInterface::CreateLocalAudioTrack

创建本端音频轨道的对象。 语法

virtual talk\_base::scoped\_refptr

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.

函数 CreatePeerConnectionFactory

创建一个PeerConnectionFactoryInterface 对象的实例。 语法

talk\_base::scoped\_refptr

CreatePeerConnectionFactory();

备注

这个生成的PeerConnectionFactoryInterface 对象会创建需求的内部资源，包括libjingle 线程，socket ，管理网络的network manager factory。

函数 CreatePeerConnectionFactory

用给出的libjingle 线程和portallocator 对象工厂创建一个PeerConnectionFactoryInterface 对象的实例。 Syntax

talk\_base::scoped\_refptr

CreatePeerConnectionFactory(talk\_base::Thread\* worker\_thread,

talk\_base::Thread\* signaling\_thread,

PortAllocatorFactoryInterface\* factory,

AudioDeviceModule\* default\_adm); 备注

当应用程序想要提供自己实现的线程和portallocator 对象时调用该方法。 这些参数的所有权不能转移到这个对象上，必须在

PeerConnectionFactoryInterface 的声明周期范围内。

Class 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 local\_streams() = 0; virtual talk\_base::scoped\_refptr 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() {} };

PeerConnectionInterface::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 对象正在尝试得到媒体的传输方式。 ∙ kActive - 连接建立成功，如果任何媒体流协商成功，相关的媒体就可以被传输了。

∙ 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, };

PeerConnectionInterface::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 - 有效的请求回答已经交换，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, };

PeerConnectionInterface::ProcessSignalingMessage

该方法用来处理对端的信令。 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，如果传输的信令与对端产生信令的顺序不同的话会使会话的建立失败或会话连接质量降低。 PeerConnectionInterface::Send 该方法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。（好奇怪老外为什么会release 这么多未实现的interface 。）

PeerConnectionInterface::local\_streams

返回一个用户代理正在尝试向对端传送的媒体流数组（由方法AddStream()添加的视频流）。 语法

virtual talk\_base::scoped\_refptr local\_streams() = 0;

PeerConnectionInterface::remote\_streams

返回一个用户代理当前正在从对端接收的媒体流数组。

这个数组当OnAddStream 和OnRemoveStream 回调函数被调用时被更新。 语法

virtual talk\_base::scoped\_refptr remote\_streams() = 0;

PeerConnectionInterface::AddStream

添加一个本地媒体流到用户代理正在尝试向对端传送的媒体流数组。这个函数只是添加媒体流并不触法任何状态变化直到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.

PeerConnectionInterface::RemoveStream

删除一个本地媒体流到用户代理正在尝试向对端传送的媒体流数组。这个函数只是添加媒体流并不触法任何媒体流状态变化直到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.

PeerConnectionInterface::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;

PeerConnectionInterface::Close

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

virtual void Close() = 0;

PeerConnectionInterface::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;

PeerConnectionInterface::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;

引用

目前HTML5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

本文档下载自文档之家，www.doczj.com-免费文档分享平台，众多试卷、习题答案、公务员考试、英语学习、法语学习、人力资源管理、电脑基础知识、学习计划、工作计划、工作总结、活动策划、企业管理等文档分类免费下载；乐于分享，共同进步，转载请保留出处:http://www.doczj.com/doc/53f4fcb483d049649a665813.html