# Webrtc阐述

## 总述

### 现状

目前WebRTC的开发有两个方向，一个是基于浏览器的应用开发，编程语言主要是JavaScript、HTML等，这也是WebRTC作为HTML5标准的原本目的；另一个是C层面的移植和开发，主要是利用google的开源项目。

**Webrtc HML5标准**

由Google等几家公司的大牛提出，交由W3C组织标准化，初衷是在浏览器上能够容易(开发简单、不需要插件等等)的实现RTC(实时通讯),所以所谓的W3C的webrtc标准就是指web层(JS/HTML)接口定义，已经有很多机构都实现了这些接口，比如google、Mozilla等；

**Google的Webrtc项目**

这个项目是由Google收购的GIPS项目和libjingle项目融合而成。其中GIPS部分主要提供媒体的处理的功能。libjingle项目部分主要提供P2P传输部分的功能。具体的项目框架参见< WebRTC框架.docx>。

Google开源了这个项目，所以我们就可以使用、借鉴它的内部实现(C/C++),封装自己的程序:Win/Linux/IOS(还不够支持)/安卓系统上的可执行程序。

### API分类

不管是Web层还是C++层，API都主要分为三大类:

**MediaStream类**，用于表示(本地/远端)媒体数据。Web层主要有getUserMedia接口，可以从摄像头和麦克风获取音视频数据并将其封装成MediaStream；

**PeerConnection类**，用于表示peer to peer之间音视频数据传输的通道，在web层叫RTCPeerConnection。在创建这个媒体通道之前需要信令来协商通道参数/信息，这个信令没有指定方式，但是一般情况下是JS/HTML经由信令服务器来转发，比如XMLHttpRequest方式，也可以采用sip/xmpp/Websocket协议来协商。

**DataChannel类**，用于peer跟peer之间传输音视频之外的一般数据，在web层叫RTCDataChannel，目前实际开发中使用比较少。

注：这几个API的名称在不同浏览器及同一浏览器的不同版本之间略有差异，比如PeerConnection在FireFox上叫做mozRTCPeerConnection，而在当前版本的Chrome上叫做webkitRTCPeerConnection，将来WebRTC标准化完成后会把这些前缀去掉使用统一的名称。

## C/C++层面

这里谈到的C/C++层面的东西都源自Google的Webrtc开源项目。

### 层次

**WebRTC Native APIs**是C++层接口,初衷是给浏览器厂商来实现JS/HTML接口用的，也可用于封装自己的Webrtc基础库；中文文档< WebRTC-NativeAPIs.docx >，原文参见<http://www.webrtc.org/reference/native-apis>

**WebRTC Internals** 是webrtc的内部实现。

### Libjingle/Webrtc

官方介绍和学习文档

<https://developers.google.com/talk/libjingle/index?hl=zh-CN>

#### 2.1项目介绍

因为Webrtc项目是融合在Libjingle项目中的,所以需要了解下Libjingle中各工程功能:

* **libjingle**：原libjingle项目的一些基础功能类，如线程，队列，消息，Socket等。p2p和会话组件都基于这些类
* **libjingle\_media**：原libjingle项目的媒体相关类。提供了基本的设备管理功能以及部分媒体抽象类。其中的webrtc目录是为了让原libjingle接口匹配webrtc的媒体处理接口做的中间类。这些中间类继承了libjingle原接口，并用webrtc/gips的相关类实现了媒体处理功能。
* **libjingle\_P2P**：原libjingle的P2P传输相关类，可以NAT设备和防火墙，在大多数网络环境下协商、建立、维持p2p连接。其中抽象了session类，用于控制p2p。
* **libjingle\_peerconneciton**：webrtc项目加入的，提供简单的外部接口， 方便使用。
* **third\_party/webrtc**：原GIPS的媒体处理相关功能类。

#### 基本模块关系

以视频传输为例，图2.2中虚线表示媒体数据流向。

**cricket::VideoCapturer**: 视频捕捉功能抽象，通过SignalFrameCaptured传递原始数据，通过SignalVideoFrame信号传递I420数据。(**ibjingle\_media,talk/media/base**)

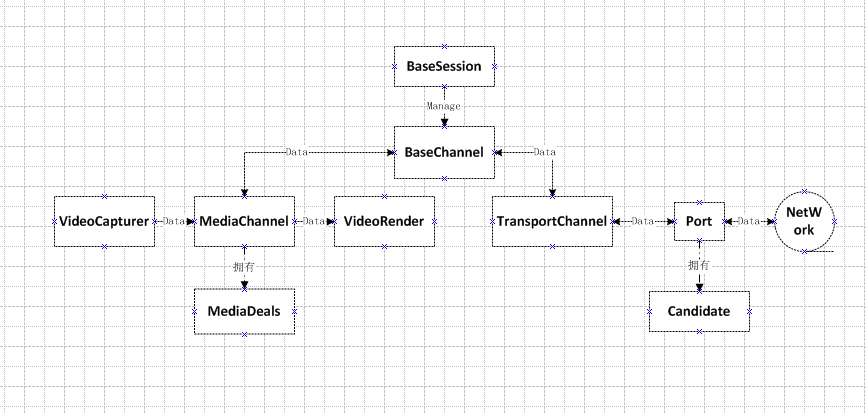
**cricket::MediaChannel**: 媒体相关功能的抽象，接收capture/network的数据，处理后发送给其他模块。实际的处理需要有继承类调用其他模块实现(**ibjingle\_media,talk/media/base**)

**cricket::****BaseChannel**: 管理MeidaChannel和TransportChannel并维护本端和对端的媒体描述信息（description）(**ibjingle\_ P2P,talk/session/media**)

**cricket::TransportChannel**: 提供抽象的P2P传输功能。实现ICE协议，ICE协商后，会选择最好的方法发送。对外只暴露发送和接收的接口。(**ibjingle\_ P2P,talk/p2p/base**)

**cricket::Port**: 具体的传输模块。派生出各种端口类型，如UDP/Stun/TCP/Turn等。 具体的端口根据相应的协议发送数据。(**ibjingle\_ P2P,talk/p2p/base**)

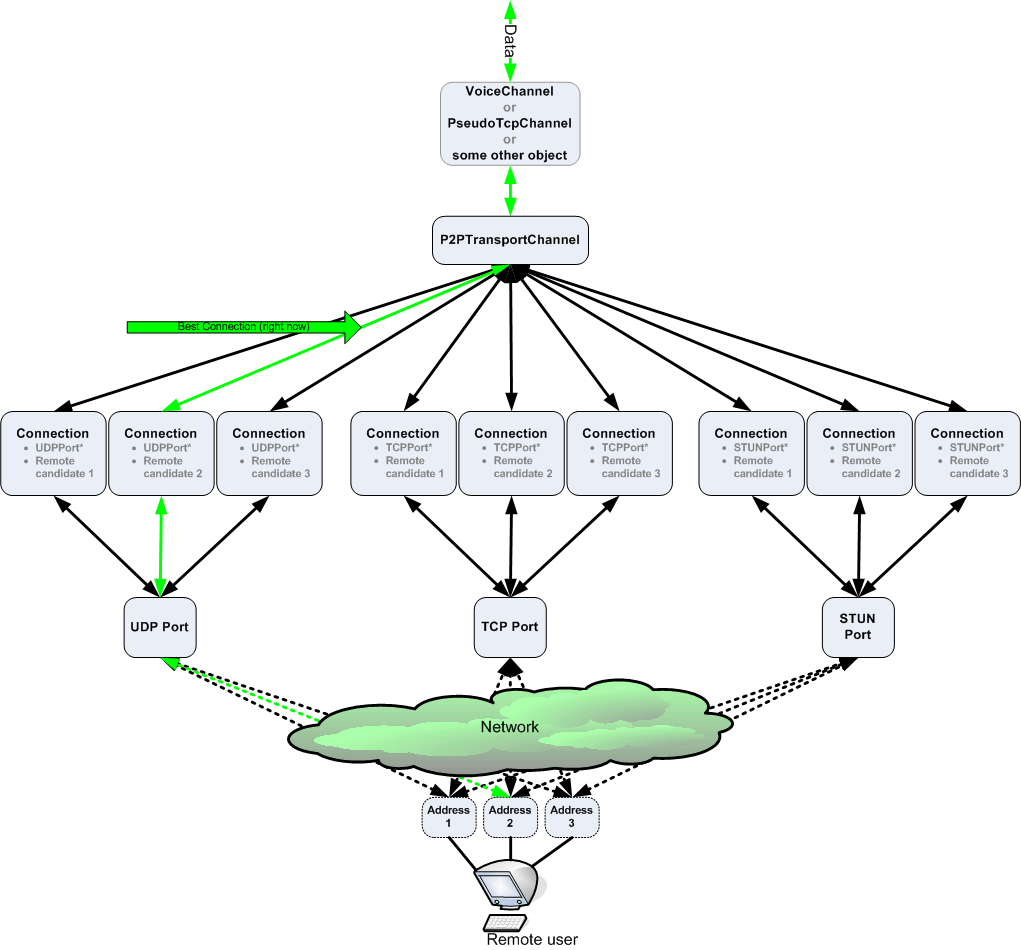
**cricket::Candidate**: 对本端网络通信能力的一种描述。对于UDP/Stun协议，candidate仅包含IP及端口信息，对于trun，包含turn server的IP，端口，以及用户名密码等。candidate由本端代码生成。生成后通过信令发送给对端。对端会在本端所有的candidate中选择一个最好的建立与本端的连接。(**ibjingle\_ P2P,talk/p2p/base**)



#### ICE问题

在peerconnction创建阶段，webrtc层会通过libjingle创建UDP/STUN/TCP Port，STUN port会向STUN服务器发起绑定请求。libjingle层会把创建的port信息回调给webrtc，从而生成

candidate。每一个candidate实际上描述了和自己的通信方式，比如一个Stun类型的candidate会包含本端在防火墙外的IP和端口类型。本端会通过信令协议（sip/xmpp/http）将自己的所有的candidate发送给对端。对端接收到后，会尝试找到一个最好的连接方式建立和本端的连接，之后的数据将通过此连接传输，如图:



## 三、HTML/JS层面

### 学习资料

1. <http://www.w3.org/TR/webrtc/>: Working Draft 这是公布的正式文档，描述webrtc定义，主要指js/html接口的定义，并不阐述webrtc内部实现，用于开发web应用；<http://dev.w3.org/2011/webrtc/editor/webrtc.html> :Editor's Draft 这个是作者草案，可能有技术预言等，非正式。
2. <http://www.html5rocks.com/en/tutorials/webrtc/basics/>官网上推荐的学习WebRTC基础知识比较好的网站。

### 封装

WebRTC的目的是为了简化基于浏览器的实时数据通信的开发工作量，但实际应用编程还是有点复杂，尤其调用RTCPeerConnection必须对如何建立连接、交换信令的流程和细节有较深入的理解。因此有高人为我们开发了WebRTC封装库，将WebRTC的调用细节封装起来，包装成更简单的API，使开发应用程序更简单。封装库的另一个目的是为了屏蔽不同浏览器之间的差异，例如上面说的API名称的差异。当然，这些库都是开源的，可以根据自己的需要重新修改。

目前网上找到的有两种WebRTC封装库，一个是[webrtc.io](https://github.com/webRTC/webRTC.io)，有很多demo使用它；另一个是[SimpleWebRTC](https://github.com/HenrikJoreteg/SimpleWebRTC)，貌似比webrtc.io用起来更简单。

### Demos

转自<http://blog.sina.com.cn/s/blog_69a04cf401016gz4.html>第三节,没有自己做过测试。

**测试环境**：手机上的浏览器是Chrome for Android 26.0.1410.49，运行在Android 4.1.2上，这个Chrome版本本身是beta版，支持WebRTC但缺省是关闭WebRTC功能的，需要在chrome://flags中使能WebRTC并重启Chrome，或者在启动Chrome时增加命令行选项--enable-webrtc。PC上运行WebRTC的浏览器是Chrome 26.0.1410.43，操作系统是Windows 7。

#### 3.1 官方Demo

http://www.webrtc.org/demo（https://apprtc.appspot.com/）

这是官方的demo，功能很全，可惜不知为何https://apprtc.appspot.com/这个网址已经连不上了，不过其源码还是可以下载到的，在https://code.google.com/p/webrtc-samples/。此demo没有用任何封装库。

这个demo所使用的信令机制使用了XHR和Google App Engine Channel API ，具体我不懂。

在我的公司局域网环境里无法运行该demo。

#### 3.2爱立信Demo

据说是第一个基于浏览器的WebRTC视频通信demo，爱立信为此还开发了一个浏览器用于支持WebRTC，好象也是基于WebKit的，叫做Bowser browser（当时市场上可能还没有支持WebRTC的浏览器），该项目网址是https://labs.ericsson.com/apps/bowser。这个Bowser browser好象只支持Ubuntu 11.04 and 11.10（见https://labs.ericsson.com/apis/web-real-time-communication/downloads）。

该demo的网址是http://webrtc.labs.ericsson.net:8082。在我的公司局域网环境里无法运行该demo。

#### 3.3人脸检测识别

利用WebRTC的getUserMedia从摄像头获取图像进行人脸识别的demo，例如这两个：http://neave.com/webcam/html5/face/

http://www.raymondcamden.com/demos/2012/mar/29/test1.html

这两个demo在PC和手机上的Chrome上都可运行。

#### 3.4 Simpl Demo

<http://www.simpl.info>, 这个demo演示HTML, CSS and JavaScript的各种feature和使用方法，包括WebRTC的3个API：getUserMedia、RTCPeerConnection、RTCDataChannel的演示，但遗憾的是RTCPeerConnection的演示只是本地camera的画面传回给本地，并没有实现真正的设备之间音视频通信。

该项目的源码在https://github.com/samdutton/simpl。

#### 3.5 Framegrabber

这是一个基于WebRTC实现屏幕共享（screensharing）的Chrome扩展，源码在https://github.com/samdutton/rtcshare，有关介绍可参考这篇文章：http://blog.sina.com.cn/s/blog\_51396f890102es7k.html。

本人没有试用过。

#### 3.6 Webrtc.io --官方Demo

<http://webrtc.dennis.is> 这个demo是基于库webrtc.io实现的，是webrtc.io官方的demo，使用WebSocket作为信令手段。

在我的公司局域网环境里无法运行该demo；在家里无线路由器环境下可成功运行，但只能单向传输视频。

#### 3.7 Webrtc.io -- 国产kainy Demo

<http://v.kainy.cn> 国内牛人做的，相当于是汉化版的http://webrtc.dennis.is，自然也是基于webrtc.io实现的，但使用的webrtc.io版本较老，不支持新版本Chrome所使用的API名称webkitRTCPeerConnection，所以无法在新版本Chrome上运行。具体介绍在http://blog.kainy.cn/2013/01/webrtc实现的视频聊天室应用。

#### 3.8 Webrtc.io -- 国产视频聊天室应用

这个demo是国内牛人赵书剑开发的视频聊天室，基于webrtc.io实现。

该项目源码和文档下载地址是http://ishare.iask.sina.com.cn/f/35083616.html，源码在https://github.com/zsj2145676。

经修改后在本人公司局域网成功运行，试过两个手机和一个笔记本电脑同时视频通信OK。修改和运行步骤：

1.从http://ishare.iask.sina.com.cn/f/35083616.html下载webrtc.chatdemo.zip，解压缩，修改其中public\javascripts\client.js中的rtc.connect一行，将实际的服务器地址写进去，例如改为：rtc.connect("ws://10.100.156.83:8001", room);

2.同上文3.8节步骤1、2安装nodejs

3.运行node app.js，注意该demo本身已包含http server，不需要其他的比如apache server

4.在同一局域网中的其他设备上打开Chrome浏览器，地址栏输入http://10.100.156.83:8000，输入相同的room名称（随便起）即可开始多方视频通信。

#### 3.9 SimpleWebRTC -- 官方conversat Demo

http://conversat.io这个demo是基于库SimpleWebRTC实现的，是SimpleWebRTC官方的demo，使用WebSocket作为信令手段。

在我的公司局域网环境里无法运行该demo；在家里无线路由器环境下可成功运行，且可双向传视频，支持多个peer同时视频通信。

经修改后在本人公司局域网成功运行，试过两个手机和一个笔记本电脑同时视频通信OK。修改和运行步骤：

1.从http://www.nodejs.org/download/下载nodejs最新版并安装，我是在Windows7 64位上安装的；

2.在命令行下依次运行如下命令（安装运行signaling server所需的模块）：

npm install express

npm install yetify

npm install getconfig

npm install node-uuid

npm install socket.io

3.从https://github.com/andyet/signalmaster下载信令服务器源码，该信令服务器是SimpleWebRTC缺省使用的，解开该源码后运行node server.js，该服务器监听8888端口，通过WebSocket与浏览器通信。

4.在同一台PC上运行apache server，将从http://conversat.io网站扒下来的文件放到该server上（可在Chrome浏览器中打开http://conversat.io然后鼠标右键单击在菜单中选“另存为”、“网页，全部”即可），修改其中的 index.html 和 simplewebrtc.js，将其中 url 改为 'http://10.100.156.83:8888'（其中IP地址是我的PC在公司局域网中的IP地址）。

5.在同一局域网中的其他设备上打开Chrome浏览器，地址栏输入http://10.100.156.83，输入相同的room名称（随便起）即可开始多方视频通信，也可修改上述index.html中的“var room”一行，设定为固定的room名称，就不需要每次在每个设备上手工输入room名称了。

## 四、Android/IOS部分

1、安卓发行版中Webrtc可能在/external/webrtc/