# 一、Jitsi视频会议系统

**1.1. 基于WebRTC的多人视频会议**

25 July 2014  
最近两周在调研和搭建基于WebRTC的多人视频会议系统。目前已经搭建成功，可以在http://jitsi.shengbin.me/试用。  
这个系统无需注册和登录，只要多人访问同一个URL（含有系统为每个房间分配的特定ID），就可以进行视频会议。  
如果上面那个链接失效，可以尝试国外一个同样的系统：https://meet.jit.si/。  
使用视频会议系统需要客户端电脑提供摄像头功能；至于带宽，当然是越大越好了。  
下面总结一下该系统的组成。

**1.2. 客户端**

客户端是一个Web App的形式，包括HTML、CSS、JavaScript代码组成的网页。HTML和CSS来构造聊天室的界面，JavaScript来实现功能。由于功能比较复杂，JS代码也较多。  
通过WebRTC，客户端从用户摄像头获取图像并传给服务器，来实现视频会议。由于WebRTC只在Chrome、Opera、Firefox上支持，而Firefox有相关的bug尚未解决，所以客户端只能运行于Chrome或者Opera浏览器。

**1.3. 服务器**

服务器端包含多个部分。下面分别介绍。  
Nginx(http://nginx.org/)  
Nginx是一个Web服务器，与著名的Apache同类。它的用途是提供网页访问。  
  
  
Prosody(http://prosody.im/)  
Prosody是一个XMPP服务器。XMPP全称是Extensible Messaging and Presence Protocol，即可扩展通信和表示协议。  
它是一种即时通信协议，主要是实现文字聊天。  
XMPP的前身是Jabber，一个开源的即时通信协议。Jabber被IETF标准化为XMPP。Google Talk用的就是它。  
  
  
Jitsi-Videobridge(https://jitsi.org/Projects/JitsiVideobridge)  
Jitsi-Videobridge用于处理视频传输，也就是视频流在各参与者之间的转发。  
如果没有这个组件，各参与者能文字聊天，但无法互相看见。  
转发意味着服务器要从N个参与者那里接受视频流，然后给每个参与者发送其他N-1个参与者的视频数据，  
这对服务器带宽要求很高。但由于未对视频做任何处理，CPU负载并不高。  
  
  
Restund(http://www.creytiv.com/restund.html)  
这是一个STUN/TURN服务器。STUN是一种NAT穿透技术，用于帮助处在内网的主机确定自己的公网IP和端口，  
从而与别的主机建立直接连接（WebRTC中PeerConnection）。  
TURN是STUN的增强版，可以在无法穿透NAT进行直连的情况下提供数据的转发。  
上述整个系统都是开源的，更多信息可参见相关的GitHub代码库和Jitsi主页。

**二、Janus 视频直播系统**

前面说过，WebRTC 是用来解决端到端的实时通信问题，也就是说它很适合用在网络电话这种需要双向视频通话的场景上。网上大部分 WebRTC 的 Demo 也都是在页面上放两个 Video，分别来播 localStream 和 RemoteStream。

**2.1 无MediaServer的视频直播系统**

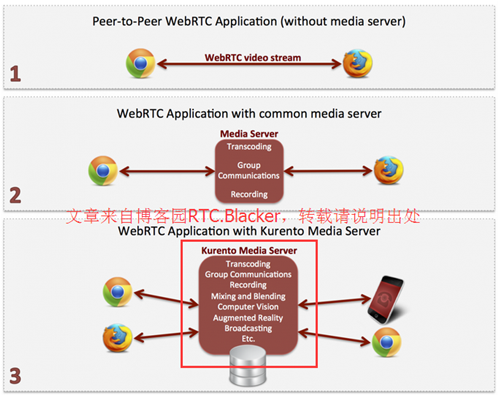
那么究竟 WebRTC 能否用来实现单向一对多直播呢？当然可以，而且貌似还很简单：  
Step1: 首先必须有一个专门负责调用 getUserMedia 采集音视频的页面，我称之为信源服务；  
Step2: 打开直播页面时，建立到信源服务的 PeerConnection，并通过 DataChannel 通知信源服务；  
Step3: 信源服务收到通知后，通过对应 PeerConnection 的 addStream 方法提供直播流；  
Step4: 直播页面监听 PeerConnection 的 onaddstream 事件，将获得的直播流用丢给 Video 播放；  
  
  
为了方便，我使用了 PeerJS 这个开源项目来验证上面这个过程。  
PeerJS 对 WebRTC Api 进行了封装，使用更简单。  
它还提供了用来辅助建立连接的 Signaling 服务，在官网注册一个 Api Key 就能用。  
也可以通过 PeerJS Server 搭建自己的服务，只需要通过   
npm install peer 装好 peer 后，再通过下面这行命令启动就可以了：  
peerjs --port 9000 --key peerjs  
启动好 Peer Server，在页面中引入 peer.js 就可以开始玩了。  
首先, 实现信源服务：  
//由于其它端都要连它，指定一个固定的 ID  
var peer = new Peer('Server', {  
    host: 'qgy18.imququ.com',   
    port: 9003,   
    path: '/',  
    config: {  
        'iceServers': [  
              { url: 'stun:stun.l.google.com:19302' }  
        ]  
    }  
});  
  
  
navigator.getUserMedia({ audio: false, video: true }, function(stream) {  
    window.stream = stream;  
}, function() { /\*...\*/ });  
  
  
peer.on('connection', function(conn) {  
    conn.on('data', function(clientId){  
        var call = peer.call(clientId, window.stream);  
  
  
        call.on('close', function() { /\*...\*/ });  
    });  
});  
  
  
然后,就是直播服务：  
//随机生成一个 ID  
var clientId = (+new Date).toString(36) + '\_' + (Math.random().toString()).split('.')[1];  
  
  
var peer = new Peer(clientId, {  
    host: 'qgy18.imququ.com',   
    port: 9003,   
    path: '/',  
    config: {  
        'iceServers': [  
              { url: 'stun:stun.l.google.com:19302' }  
        ]  
    }  
});  
  
  
var conn = peer.connect('Server');  
  
  
conn.on('open', function() {  
    conn.send(clientId);  
});  
  
  
peer.on('call', function(call) {  
    call.answer();  
    call.on('stream', function(remoteStream) {  
        var video = document.getElementById('video');  
        video.src = window.URL.createObjectURL(remoteStream);  
    });  
  
  
    call.on('close', function() { /\*...\*/ });  
});  
  
  
直播页面通过指定 ID 的方式跟信源服务建立端到端连接，  
然后通过 DataChannel 告诉信源服务自己的 ID，信源服务收到消息后，  
主动把直播流发过来，直播页面应答后播放就可以了。整个过程原理就这么简单，  
这里有一个「完整的 Demo」。  
看完上面的 Demo，你也许会想原来使用 WebRTC 直播这么简单，随便找台带摄像头的电脑，  
开个浏览器就能提供直播服务，那还搞 HLS、RTMP 什么的干嘛。

**2.2 Janus视频直播系统应用简介**

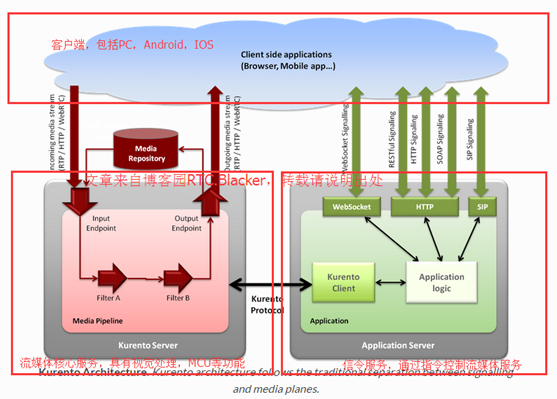
实际上，现实并没有那么美好，这个 Demo 也就玩玩儿还可以，真正使用起来问题还大着呢！  
首先，虽然说在 WebRTC 直播方案中，服务端只扮演桥梁的工作，  
实际数据传输直接发生在端到端之间，但前面说过仍然会有 8% 的情况完全不能直连。  
要保证服务的高可用性，还是得考虑部署 TURN 这种复杂而昂贵的中转服务。  
其次，Chrome 对每个 Tab 允许连接的终端数有限制，最多 256 个。  
实际上，在我最新的 Retina Macbook Pro 上，差不多有 10 个连接时，  
Chrome 就开始变得无比卡，风扇呼呼地转，内存被吃掉 6G，CPU 一直跑满，  
网络吞吐开始忙不过来，直播服务也开始变得极其不稳定。  
  
  
所以实际使用方案中，一般还是需要 Media Server 的支持，  
把「端到多端」变成「端到 Media Server 到多端」的架构。  
Media Server 可以有更好的性能和带宽，可以自己实现 WebRTC 协议，  
也就有了支持更多用户的可能。  
  
  
我找到一个名为 Janus 的 WebRTC Gateway，这个开源项目用 C 语言实现了对 WebRTC 的支持。  
Janus 自身实现得很简单，提供插件机制来支持不同的业务逻辑，  
配合官方自带插件就可以用来实现高效的 Media Server 服务。  
Janus 官方提供的 Demo 在这里，我也尝试在我的 VPS 上部署了一套。  
  
  
Janus 有个 Streaming 插件，可以接受 GStreamer 推送的音视频流，  
然后通过 PeerConnection 推送给所有的用户。由于 GStreamer 可以直接读摄像头，  
也就不用再走 WebRTC 的 MediaStream 获取视频，这样架构就变成了传统的服务器到端了。  
整个过程比较复杂和曲折，这里不写了，有兴趣的同学可以单独找我讨论。

# 三、Kurento视频直播系统

## 4.1  应用实例

搞视频会议就会涉及一对多、多对多、广播、转码、混音、合屏、录制，这就需要用到流媒体服务器，而kurento就具有这些功能。  
他主要用来作为webrtc的流媒体服务器，因为BUG多，目前不适于商用，不过前景可期，   
图1:  
  
说明：  
1、看到这里您可不要讲他的功能和ICE服务器的功能给搞混了哦，后者主要用来做NAT穿透和转发的。

### 4.1.1 、kurento架构

图2:  
  
  
说明：  
1、客户端对音视频数据的采集和播放等是通过webrtc来处理的，传输模块就是kurento的。  
2、流媒体服务是他的核心服务，可以进行编解码，混音，录制，计算机视觉，视觉增强等等。

### 4.1.2  、特色功能---计算机视觉

图3:  
  
  
说明：  
1、服务端可以对收到的视频流进行处理，如人脸识别，这些扩展下去应用前景就很广泛了，期待！  
2、因为他对图像进行了处理，所以延迟会比较大，识别率还存在些问题，而且会造成图像闪动（可能也是跟延迟有关）。  
3、其他功能如一对一，广播就不重复了，很多其他流媒体服务都具有这些功能。  
   
最后：虽然kurento目前问题很多，但我看好他，后面会继续分享相关内容，也会和他们一起去完善这个东西。

## 4.2  Kureto官网

链接：http://www.kurento.org/whats-kurento  
What's Kurento?  
Kurento is a WebRTC media server and a set of client APIs making simple the development   
of advanced video applications for WWW and smartphone platforms.   
Kurento Media Server features include group communications, transcoding,   
recording, mixing, broadcasting and routing of audiovisual flows.  
  
  
Kurento 是一个 WebRTC 流媒体服务器以及一些客户端API，有了它，  
开发WWW及智能手机平台的高级视频应用就变得更加容易。  
可以利用Kurento开发的应用类型包括，视频会议，音视频广播，音视频录制、转码等。  
  
  
As a differential feature, Kurento Media Server also provides advanced media   
processing capabilities involving computer vision, video indexing,   
augmented reality and speech analysis. Kurento modular architecture   
makes simple the integration of third party media processing algorithms   
(i.e. speech recognition, sentiment analysis, face recognition, etc.),   
which can be transparently used by application developers as the rest of Kurento built-in features.  
.  
   
Kurento Media Server adds advanced media processing to the capabilities of common media servers.

# 四、一个基于WebRTC的混合CDN/P2P直播开源项目

最近在GitHub看到一个基于WebRTC的混合CDN/P2P直播开源项目，觉得很有前景。  
相比传统P2P直播软件而言，该项目的优势是不需要安装任何插件，而且主流浏览器都支持。

出于好奇，我也在服务器上创建了一个实例，看看效果如何。

如果你稍微有那么一点的感兴趣，我邀请你一起参与测试。这里是一些注意事项：  
1. 测试地址 http://www.zoptv.com/player/p2p/  
2. 支持的浏览器 Chrome, Firefox, Opera  
3. 测试页面和视频，之后可能被删掉  
4. 期待你的建议，我会不定期的在此帖回复  
  
P.S 感谢原作者 http://github.com/bemtv (貌似github被墙了？)  
  
官方链接：  
https://github.com/bemtv  
  
http://bem.tv/  
BemTV is an attempt to scale live video streaming using peer-to-peer  
 without the need of an external plug-in. It uses the powers of WebRTC to   
build swarms, enabling the possibility to flow chunks between users.

# 基于WebRTC开源MCU框架—1 **licode**

官网地址：  
[http://lynckia.com/licode/index.html](http://lynckia.com/licode/index.html" \t "_blank)  
  
官方demo地址：  
[https://chotis2.dit.upm.es/](https://chotis2.dit.upm.es/" \t "_blank)  
  
Github地址：  
[https://github.com/lynckia/licode](https://github.com/lynckia/licode" \t "_blank)  
  
Github推荐指数：  
1483（star）  
  
功能（官方原文）：  
Licode is based on WebRTC technologies. It is **100% compatible(兼容)** with latest stable versions of Google Chrome. Your users will be able to**talk from their web browsers** with no need to installing anything.  
You don't need to care about **complicated real-time infrastructures（复杂的实时传输内部结构）**. It provides a fast development of videoconference features based on HTML5. And we make it 100%**scalable（可扩展）**.  
Licode allows you to **include videoconference rooms on your web**. But you can also implement**streaming**,**recording**and **any other real-time multimedia features** you dreamt of  
  
  
主要模块及实现语言：  
1.Erizo：MCU模块，实现语言C++  
2.erizoAPI：Node.js addon wrapper for Erizo，实现语言C++  
3.erizo\_controller：跟Rooms管理相关的服务，实现语言JS  
4.nuve：跟Rooms管理相关的API，实现语言JS  
  
接口（SDK）语言：  
Client API: Javascript  
Sercer API: node.js  
  
优点：  
接口简洁，服务轻量级，支持h264  
  
缺点：  
API文档比较简单且其他文档较少；客户端接口只有js的，没有android和IOS原生API；不是很稳定，经常中断

# 基于WebRTC开源MCU框架—**2 kurento**

官网地址：  
[http://www.kurento.org/](http://www.kurento.org/" \t "_blank)  
  
官方demo地址：  
无  
  
Github地址：  
[https://github.com/Kurento](https://github.com/Kurento" \t "_blank)  
  
Github推荐指数：  
735（star）  
  
功能（官方原文）：  
Networked streaming protocols, including **HTTP**, **RTP**and **WebRTC**.  
Group communications (MCUs( Multipoint Conferencing Unit) and SFUs(Selective Forwarding Unit.) functionality) supporting both**media mixing and media routing/dispatching**.  
Generic support for **computational vision（计算机视觉）** and **augmented reality（现实增强）** filters.  
**Media storage** supporting writing operations for **WebM** and**MP4**and playing in all formats supported by **GStreamer**.  
Automatic media transcodification between any of the codecs supported by GStreamer including VP8, H.264, H.263, AMR, OPUS, Speex, G.711, etc.  
   
主要模块及实现语言：  
Kurento media server：KMS流媒体服务器，实现语言C++  
  
接口（SDK）语言：  
服务端SDK：java，node.js  
客户端SDK：java，js  
支持Json-Rpc  
  
优点：  
文档齐全，功能、封装API都比较齐全，对Android和IOS也有原生API支持，支持h264  
  
缺点：  
bug较多，不是很稳定，接口太多，所以使用起来相对复杂，Android和IOS缺少官方demo，因其中增加了视觉增强等图像处理功能，所以会有视频延迟风险

# 基于WebRTC开源MCU框架—**3 jitsi**

官网地址：  
[https://jitsi.org/](https://jitsi.org/" \t "_blank)  
  
官网demo地址：  
[https://meet.jit.si/](https://meet.jit.si/" \t "_blank)  
  
Github地址：  
[https://github.com/jitsi](https://github.com/jitsi" \t "_blank)  
  
Github推荐指数：  
1678（star）  
  
功能（官方原文翻译）：  
不像其他的视频会议技术，Jitsi Videobridge是jitsi的核心，将所有人的音视频传送给所有的参与者，而不是将他们混起来  
能达到更低的延迟，更高的质量，并且如果你运行你自己的服务，这将是一个非常便于扩展和廉价的解决方案  
Jitsi完全兼容webRTC这个开放的web通信标准  
Jitsi支持高级的视频路由功能，比如同步广播、带宽检测、可扩展的视频编码等  
Jitsi是用java和一些原生代码编写  
  
主要模块及实现语言：  
1. Jitsi Video-Bridge (Software video-bridge 实现语言java)  
2. Jitsi Jicofo (Component mandatory for jitsi conference 实现语言java)  
3. Prosody ( XMPP Server 实现语言lua)  
4. Nginx (Web Server)  
5. Jitsi Meet (Web application – to which the end user will interact. 实现语言js)  
  
接口（SDK）语言：  
Libjitsi：java接口SDK，用于服务端开发  
lib-jitsi-meet：javascript接口SDK，用于web客户端的开发  
  
优点：  
比较稳定，家族产品较多，即时通讯，电子白板，文件共享都有。2017年8月发布android和IOS原生API接口。  
  
缺点：  
协议用的是SIP和XMPP，编译部署过程过于复杂，依赖库较多，且文档比较少。缺少android和IOS的demo和文档。多人对讲时采用的是单路分发机制，对服务器网络等要求较高。

**有哪些开源媒体服务器可以选择使用？**

正如上文提到过的，WebRTC生态系统是非常庞大的，在市场上有一些开源产品可以选择。

下面是一些最成熟/最知名的产品：

# [Jitsi平台](https://jitsi.org/jitsi-meet/" \t "_blank)

[](http://webrtc.org.cn/wp-content/uploads/2017/12/MediaServers3.png)

Jitsi不仅是一个WebRTC媒体服务器，而且还是一个围绕其搭建的完整的平台。Jitsi的产品家族包括了Jitsi Videobridge（媒体传输，SFU），Jitsi Meet（会议网络客户端），Jicofo（Jitsi会议焦点），Jitsi SIP电话等等。Jitsi平台最吸引人的特点是它包含了一个交流平台的一切，可以在几个小时内就启动和实现运行。它还使用了Jingle（XMPP）和全功能的Web界面实现自己的信号传输。遗憾的是，最大的缺点就是实施媒体录制，因为Jitsi没有可靠的，易于使用的解决方案。

[Kurento媒体服务器](https://www.kurento.org/" \t "_blank)

[](http://webrtc.org.cn/wp-content/uploads/2017/12/MediaServers4.png)

这是目前最通用的解决方案之一。它不仅仅是一个媒体服务器，还是一个用来搭建媒体服务器的工具箱。Kurento的主要优势在于它的多功能性，通过引入媒体工作流的概念允许在代码中定义媒体要如何进行流动，以及向哪里流动。这使得WebRTC开发者能够编写和集成注入机器视觉（例如二维码扫描，人脸识别），实时媒体修改，以及与RTP（VoIP）服务互操作的非常有趣的特性。Kurento在一个实例中也可以配置充当SFU或者MCU，或者两者。

# [Janus WebRTC网关](https://janus.conf.meetecho.com/" \t "_blank)

[](http://webrtc.org.cn/wp-content/uploads/2017/12/MediaServers5.png)

尽管在它的描述中没有任何地方提到“媒体服务器”，但是Janus可以很容易地就被设置为一个SFU。其最显著的特点之一是它的插件架构，可以增强服务的核心功能。在这个[demo页](https://janus.conf.meetecho.com/demos.html" \t "_blank)中，展示了Janus的一些有趣的用例。

# [mediasoup](https://mediasoup.org/" \t "_blank)

[](http://webrtc.org.cn/wp-content/uploads/2017/12/MediaServers6.png)

这是一个相对较新而且比较有趣的媒体服务器，与其他媒体服务器不同的是，它被设计为一个库（用于Node），允许它集成到更大的应用程序中去。