WebRTC音视频通话环境搭建

作者：刘琪(sebliu)

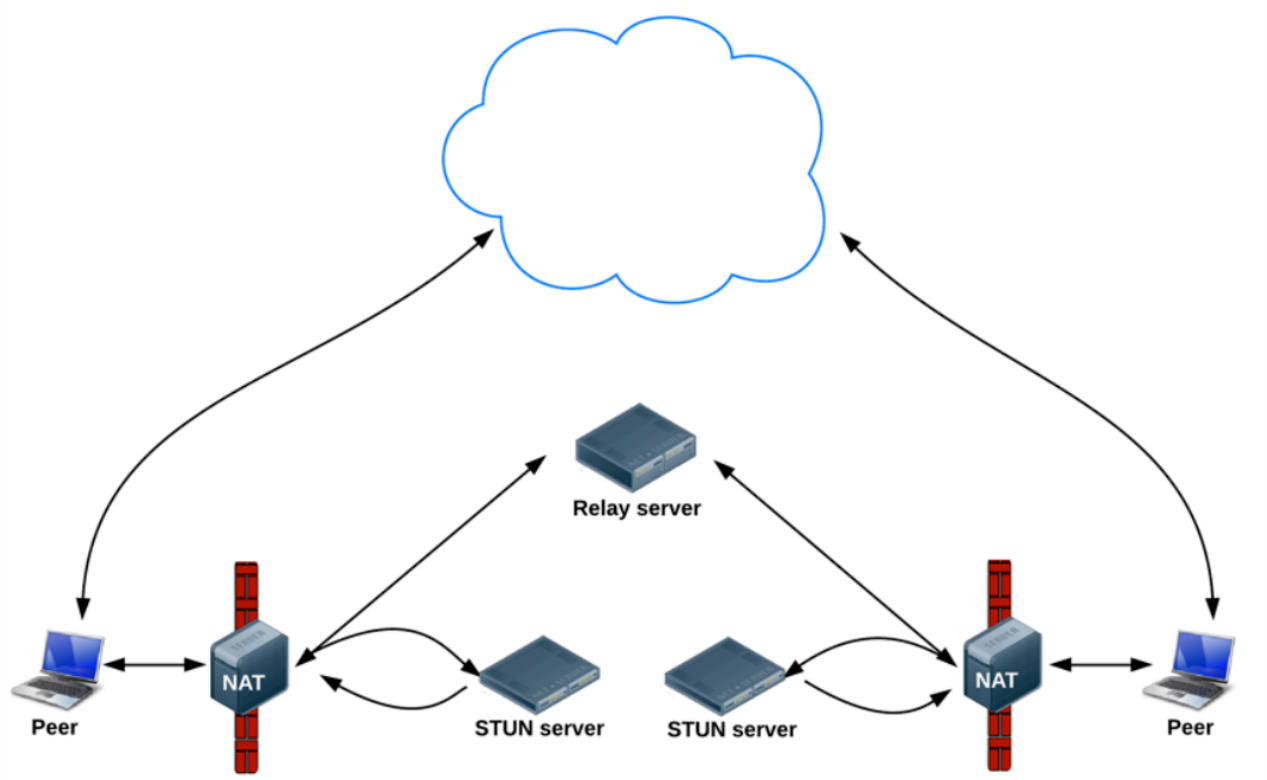
动机

为了进一步加强我组音视频相关技术实力，我与张弘(hongzhang)想要通过学习、汲取webRTC的音视频策略精华的部分来丰富我组当前音视频相关技术知识的积累。学习webRTC的第一步是搭建好可以利用webRTC进行音视频通话的测试环境。为此本人进行了webRTC 音视频通话环境的搭建部署，大体记录如下:

WebRTC 1对1音视频通话环境部署

背景

webRTC的native code为c++编写而成，如今已经内嵌在各个主流浏览器中，如chrome，firefox等(本次搭建的测试环境在chrome和firefox上均可正常运行)。为了让开发者们便于使用webRTC，上述浏览器均提供了供开发者调用webRTC API的javascrip接口，因此本次部署环境代码皆为js脚本，其系统框架如下图所示:



Peer之间由于NAT的限制，需要借助STUN/TURN建立连接通信，STUN server可以用来获取Peer的公网地址，TURN server(也称Relay server)用于对称NAT场景下(STUN失效)中转音视频数据。webRTC连接建立流程可以参考WebRTC信令控制与STUN/TURN服务器搭建

信令服务器的部署

本次部署参考的主要开发文档和代码分别为mozilla MDN开发文档信令与视频通话和其提供的源码，由于其提供的源码不能直接运行，需要增加一些额外的代码，完善后的代码可以参考这里。

首先需要准备一台远程主机，用于作为信令服务器和TURN(Traversal Using Relay NAT)服务器使用。拥有服务器后，登录服务器，安装npm包管理工具

```
sudo yum install nodejs
```

安装好后，在工作目录 git clone代码

```
git clone https://github.com/surpass007/webrtc-chat
```

clone完代码后，需要先安装好依赖组件的如websocket等，执行startup.sh脚本

```
./startup.sh
```

会安装好组件并启动后台服务程序，也可以自行启动后台服务如下：

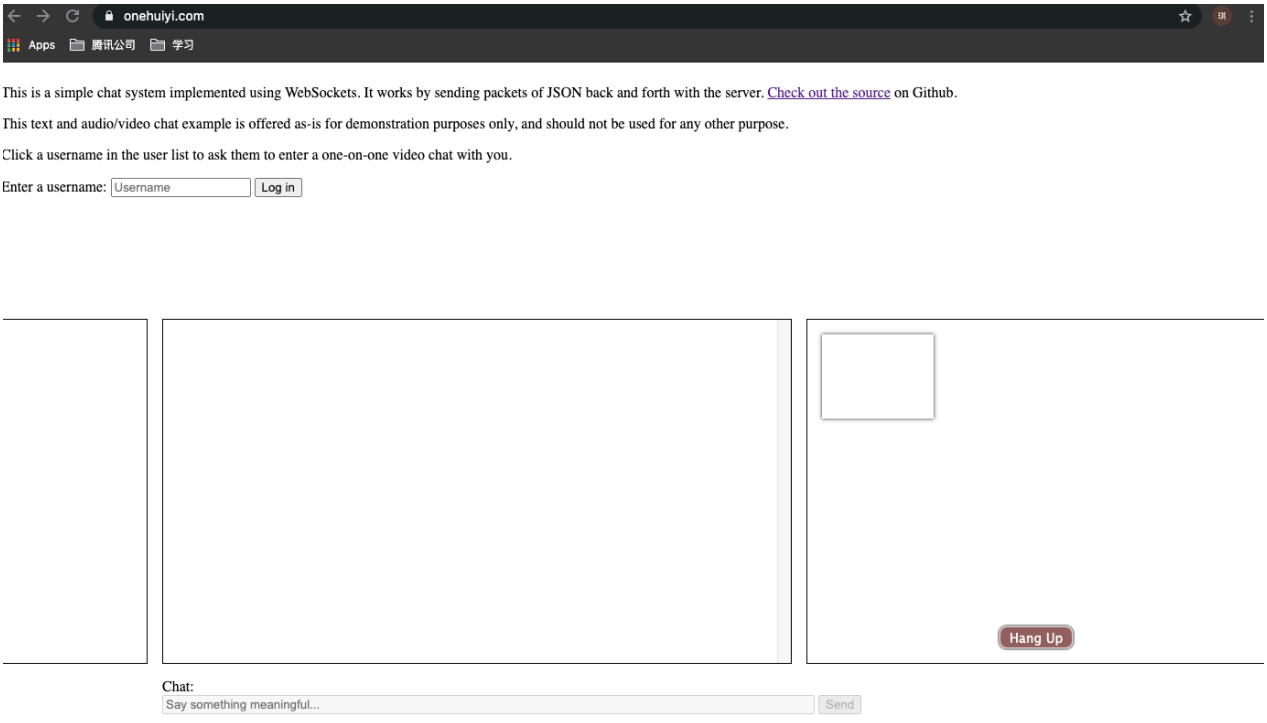
```
node chatserver.js
```

这个时候后台服务会监听443端口(https默认端口)，由于当前chrome浏览器会禁止http协议获取本机麦克风和摄像头，因此需要给后台服务部署SSL证书，一般证书是和域名绑定的，因此需要先为该服务器申请一个域名，由于国内域名申请步骤繁琐，可以去这里申请域名，证书可以去这里进行申请，证书申请成功后会获得一个youdomain.zip文件和一个私钥文件，解压文件后可以获得`yourdomain.ca-bundle`，`yourdomain.crt`和`yourdomain.p7b`这三个文件，将key私钥文件和证书.crt文件路径拷贝至chatserver.js脚本中如下：

```
const keyFilePath = "onehuiyi_com_key.txt";
const certFilePath = "onehuiyi.com.crt";
```

证书的原理大致为：站点向CA购买私钥，CA颁发证书给站点，证书的生成是通过私钥对购买者的信息进行签名(就是利用privite key对明文签名/加密)，当访问者访问该站点时，先去向CA获取该站点域名的公钥，站点方提供明密文对供访问者校验，访问者利用获取的公钥对密文解密并和明文比较，如果信息一致，说明站点是CA授权站点，可以安全访问。

目前为止，https部署成功，不出意外，打开本地浏览器(chrome或firefox)输入https://yourdomain可以加载出登录页面如下：



此时的远程server，只能作为信令服务器使用来初始化浏览器之间的连接，对于NAT场景还无法直接进行音视频通话，因此还需要利用一台STUN服务器来打洞，但是在对称NAT场景下，STUN打洞不起作用，因此还需要TURN服务器作为 relay中转音视频数据包。幸运的是coturn已经了将STUN和TURN两个服务集成到了一起。本次搭建环境将信令服务器和STUN/TURN服务器同机器部署。coturn的部署主要参考WebRtc之搭建 coturn(STUN/TURN)和WebRTC信令控制与STUN/TURN服务器搭建，主要步骤为：

获取源码

```
git clone https://github.com/coturn/coturn.git
```

编译安装

```
cd coturn
./configure --prefix=/usr/local/coturn
sudo make -j 4 && make install
```

配置coturn

进入/usr/local/etc/目录

```
cd /usr/local/etc/
```

编辑turnserver.conf文件，填写如下配置

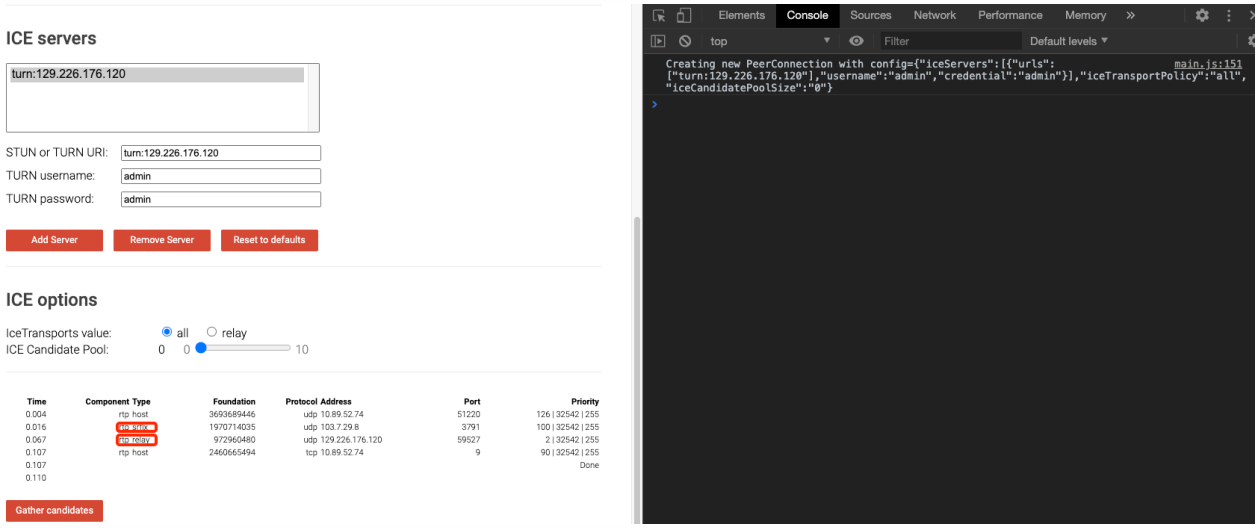
```
listening-port=3478  #turn default listener port
listening-device=eth0     # ethernet nic
listening-ip=172.19.0.5    #internal ip
external-ip=129.226.176.120  #public ip
user=admin:admin    #username:credential
realm=onehuiyi.com  #yourdomain
```

编辑后保存，然后启动STUN/TURN服务

```
turnserver -c /user/local/etc/turnserver.conf
```

测试STUN/TURN服务

打开trickle-ice，输入RTCPeerConnection中的iceService配置，点击Gather candidates 如下：



可以看到rtp srflx，说明STUN正常工作，看到rtp relay说明TURN正常工作，从右侧的cosole也可以看出，这个页面其实相当于接受输入的信息并构造一个RTCPeerConnection对象发起连接请求。

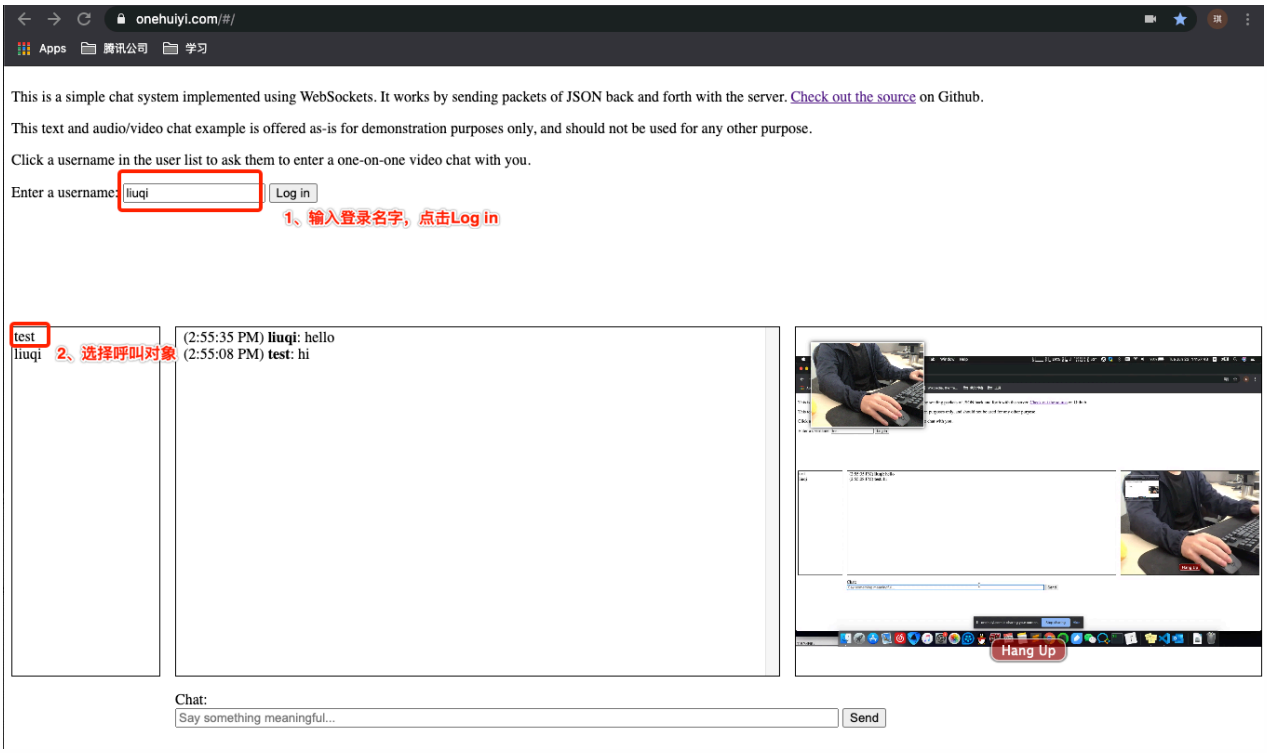
最后，将配置好的STUN/TURN信息作为初始化RTCPeerConnection对象的参数，具体为修改chatclient.js脚本如下：

```
iceServer:[
  {
    urls: "turn:" + myHostname,
    username:"admin",
    credential:"admin"
  }
]
```

重新打开浏览器访问 https://yourdomain，输入名字，点击log in 进入房间，选择要通过的对象(不能选自己)，便可以通过turn进行音视频通话了。

试一下

打开chrome或者firefox浏览器，在地址栏输入 https://onehuiyi.com可以进入已经搭建好的音视频通话页面。输入名字后点击log in登录房间，选择要呼叫的对象，并允许浏览器获取本机麦克风和摄像头，便可以开始视频通话了。如下图所示：



主要参考资料

WebRTC信令控制与STUN/TURN服务器搭建

mozilla开发文档：信令与视频通话

mozilla 示例代码 sampler-server

STUN/TURN测试地址 trickle-ice

域名申请/购买链接

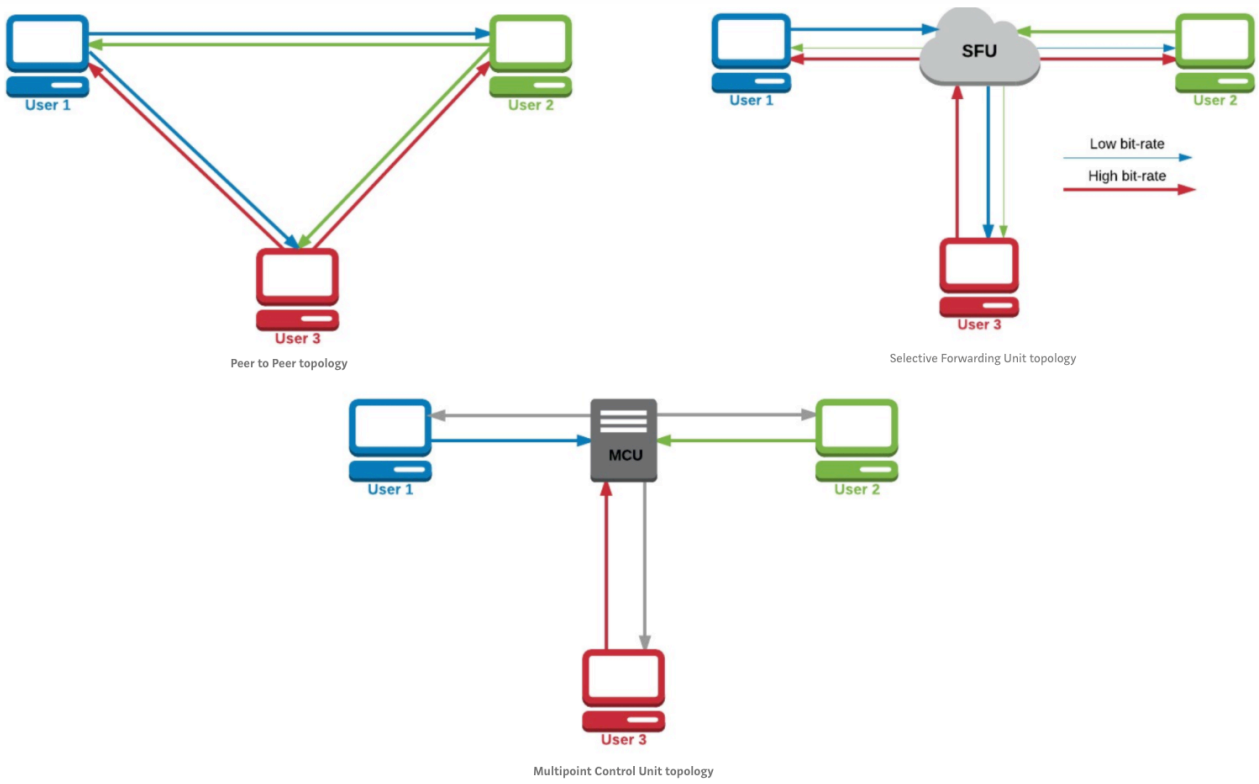
证书购买链接

mozilla webRTC API

WebRTC多人视频通话服务搭建

背景

一般来说，实现多人视频通话服务有三种方式，分别为mesh，SFU(Selective Forwarding Unit)，MCU(MultiPoint Control Unit)。其对应结构如下图所示：



- **mesh**(图中Peer to Peer topology)结构：其特点是对于每个用户来说，都有n-1条发送流和n-1条接收流(类似于有向完全图)，该方案优点是实现起来比较简单，但是用户带宽占用巨大(每个用户有$2n-2$条链路)，在弱网情况下时延将非常明显，整体网络链路数为$2\cdot\binom{n}{2}=n^2-n$条链路。
- **SFU**结构：该模式下每个用户只需要发送一条上行流，由媒体服务器管理转发的流至用户。用户可以选择想要观看的下行流，与mesh结构相比，该方案显著减少了用户端的上行带宽，但是相应的增加了媒体服务器的复杂性。当有n个用户，将共有$n+n\cdot(n-1)=n^2$条链路
- **MCU**结构：媒体服务器通过将单个用户的多条下行流合并为一条下行流，使得每个用户只需要一条下行流，极大的降低了用户接收带宽，且该方案有效的降低了网络的整体带宽，链路数为$2n$。

在实际应用场景中，语音信号为时域上的振幅点，对于人耳来说，人所处世界环境听到的声音本身就是各个声源通过空气分子振动的叠加，因此，对多路语音信号进行混音合并为一条语音信号是符合人耳的听声模式。因此语音信号多采用MCU结构，而图像信号为空间域上的像素点，视频混流需要对多路视频图像像素进行重新排列，而实际场景中用户更希望能定制化不同视频流的显示位置，因此一般对视频信号采用SFU结构。

SFU媒体服务器部署

为了方便部署媒体服务器，采用开源媒体服务器框架，下图列举了当前比较流行的媒体服务器框架

Name	SFU	MCU	Recording	Document	License	Comments
Kurento	✓	✓	✓	✓	Apache v2.0	elasticRTC
Janus	✓	✓	✓	✓	GPLv3	Meetecho Slack
Jitsi	✓		✓	✓	Apache v2.0	
Licode	✓	✓	✓	✓	MIT	Intel CS
Intel CS for webrtc	✓	✓	✓	✓	Free	Licode iqiyi
MediaSoup	✓			✓	ISC	Node.js

这里选择了Kurento框架部署媒体服务器。

由于目前Kurento只支持ubuntu平台，而本人搭建环境为CentOS，Kurento官方提供了docker镜像版本，且已经加入到了docker hub中，因此理论上借助docker可以部署在任何平台，这里采用Docker容器方式部署，部署参考这里

首先安装Docker

```
sudo yum install docker
```

安装完成Docker后，获取Kurento-media-server(以下简称KMS)镜像源，输入如下命令获取

```
docker pull kurento/kurento-media-server:latest
```

获取镜像源后，运行该镜像

```
docker run --name kms -d -p 8888:8888 kurento/kurento-media-server:latest
```

该命令的含义为启动一个以镜像名为 kurento/kurento-media-server:latest 的容器，容器命名为 kms，并设置端口转发，将主机中的端口8888(左侧)映射为docker中的容器占用端口8888(右侧，默认情况下容器与外部进程隔离，无法相互通信，因此需要设置端口转发)。

获取KMS进程运行日志

```
docker logs --follow kms >"kms-$(date '+%Y%m%dT%H%M%S').log" 2>&1
```

测试KMS是否启动成功

```
curl \
    --include \
    --header "Connection: Upgrade" \
    --header "Upgrade: websocket" \
    --header "Host: 127.0.0.1:8888" \
    --header "Origin: 127.0.0.1" \
    http://127.0.0.1:8888/kurento
```

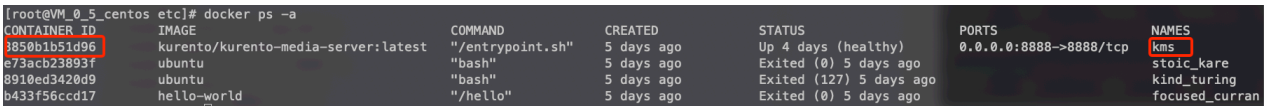
正常启动的情况下会得到如下响应

```
HTTP/1.1 500 Internal Server Error
Server: WebSocket++/0.7.0
```

注：Server Error 为正常显示，可以忽略

接下来需要通过修改其相应的配置文件来指定STUN/TURN server，具体步骤为查看当前运行容器(如果已经进入容器可以忽略该步骤)

```
docker ps -a
```



找到之前启动的容器名字(NAMES)或容器ID(CONTAINER ID)，进入容器

```
docker exec -it kms bash # your container name
```

或者

```
docker exec -it 8850b1b51d96 bash  # your container id
```

然后打开配置文件

```
vim /etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini
```

添加如下配置

```
stunServerAddress=129.226.176.120
stunServerPort=3478
turnURL=admin:admin@129.226.176.120:3478
```

该配置与前述的webRTC的1对1音视频通信中的coturn部署的配置文件相一致，修改完保存后，退出当前Docker容器

```
exit
```

然后重新启动KMS容器使配置生效

```
docker restart kms # your container name
```

或则

```
docker restart 8850b1b51d96  # your container id
```

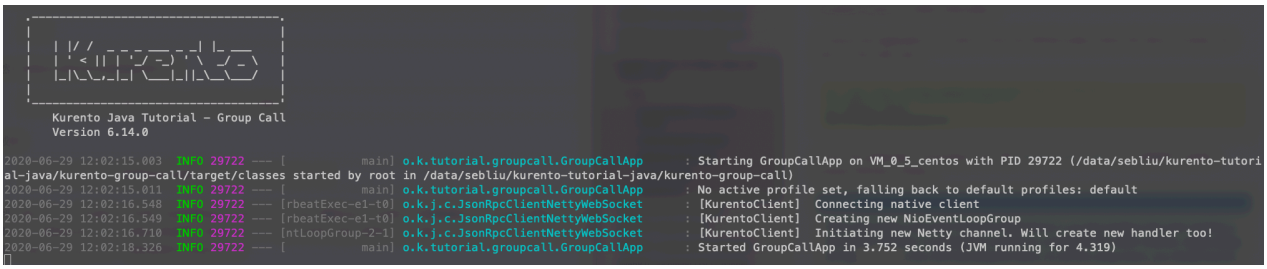
到目前为止，不出意外，KMS已经部署成功，接下来可以运行测试demo来进行音视频通话了，这里采用Kurento官方提供的demo，源码参考这里，其相应的部署文档参考这里。由于KMS目前只提供了java和javascript接口，且Kurento官方提供的demo基于spring boot，因此需要先安装maven

```
yum install -y maven
```

安装完成后，clone demo程序并启动

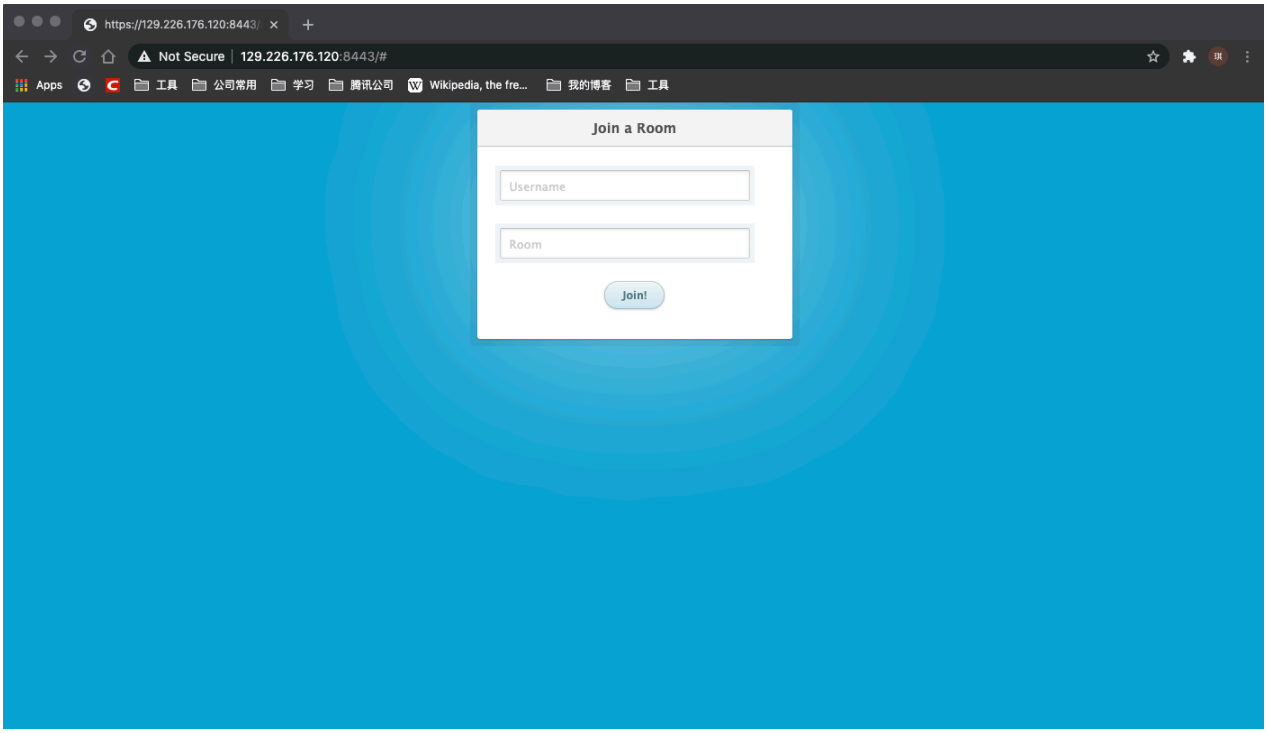
```
git clone https://github.com/Kurento/kurento-tutorial-java.git
cd kurento-tutorial-java/kurento-group-call
git checkout master
mvn -U clean spring-boot:run
```

看到如下界面说明多人视频服务端启动成功

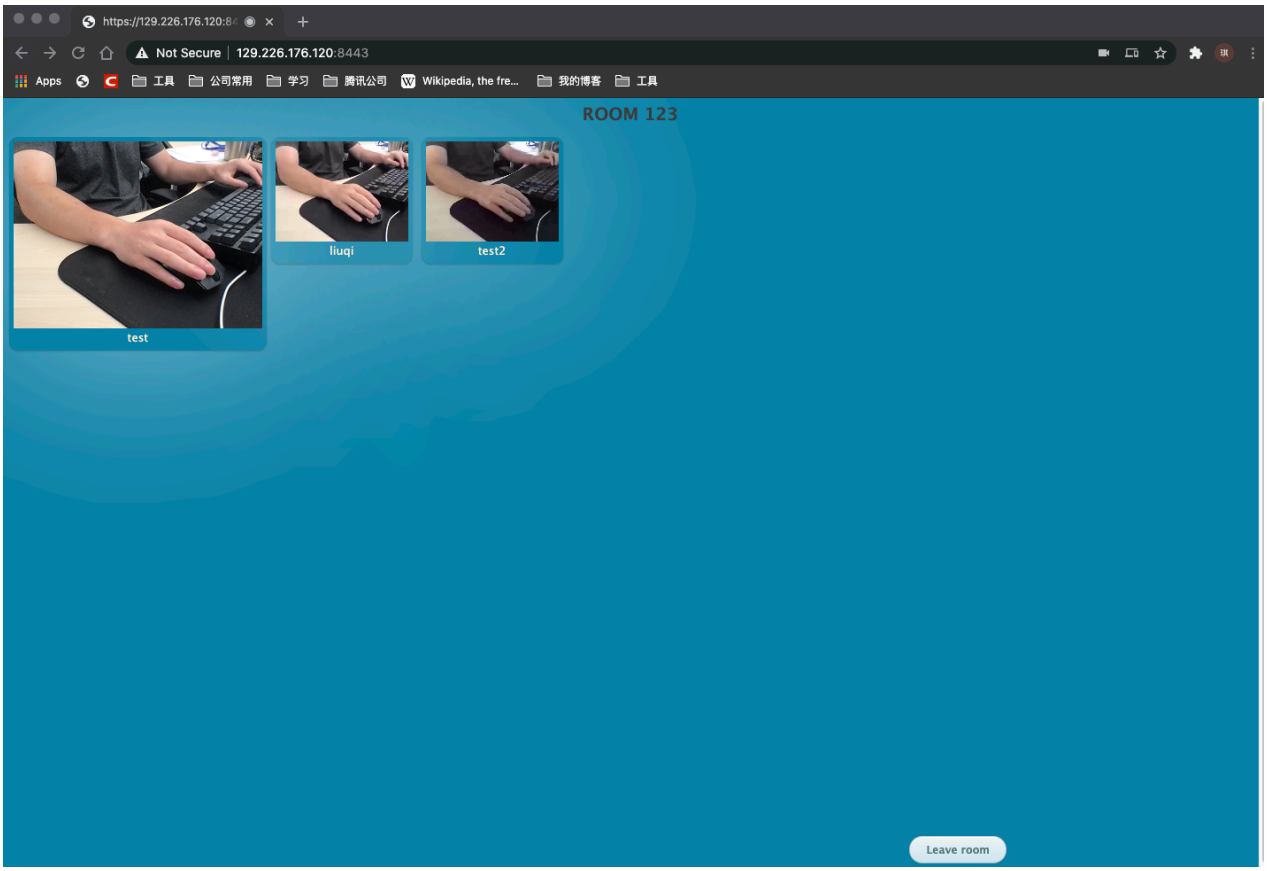


多人视频测试

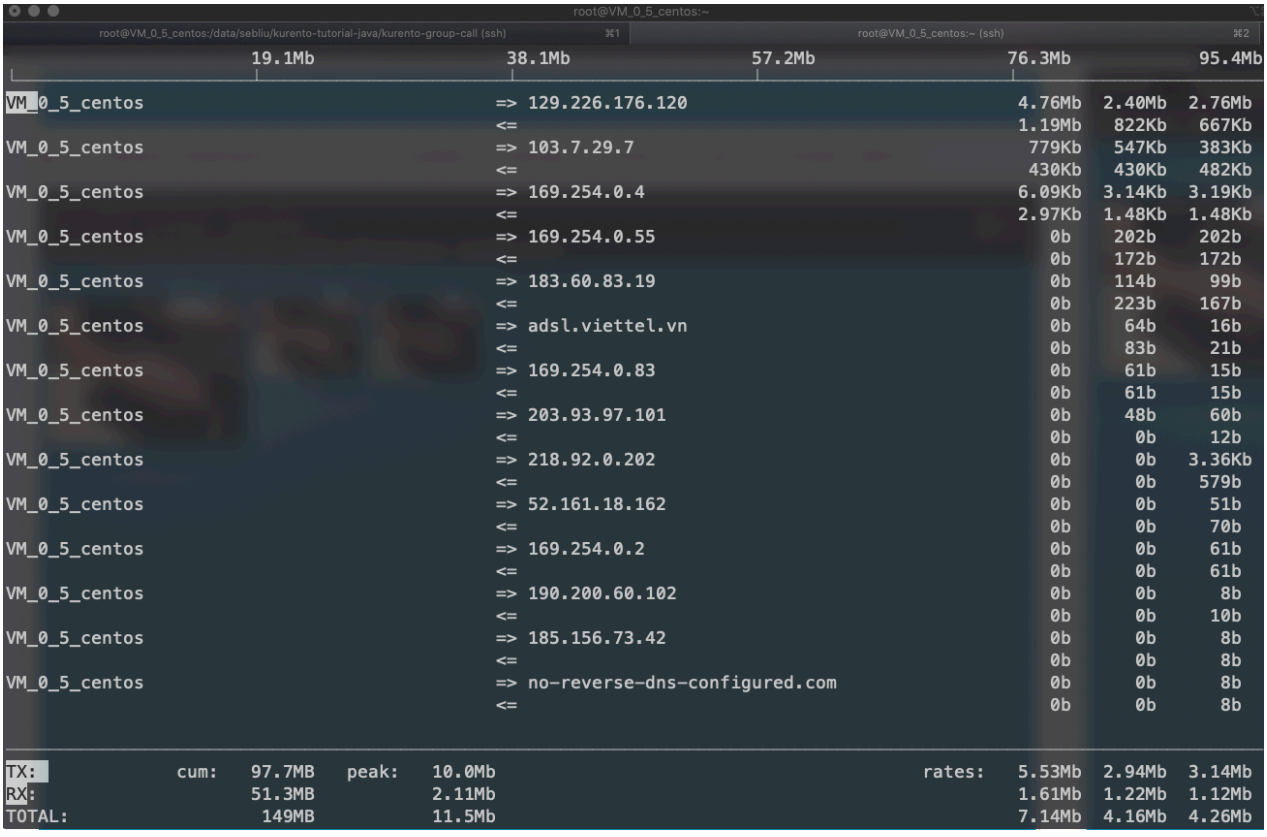
接着打开chrome或firefox浏览器输入https://serverIP:8443可以看到进入房间页面



输入名字和房间号(如果房间号不存在后台会自动创建一个房间)进入会议房间。三人进入同一房间如下图所示



使用iftop工具查看后台服务器带宽占用情况



右下角rate处有三列三行，第一行为发送流量，第二行为接收流量，第三行为总流量(发送+接收)，第一列为2s平均流量，第二列为10s平均流量，第三列为40s平均流量，可以看到由于采用的是SFU模型，当接入n个用户时，媒体服务器接收n条上行链路，并且发送$n(n-1)$条下行链路，所以发送流量约为接收流量的$n-1$倍。

主要参考链接

WebRTC网关服务器搭建：开源技术vs自行研发

Scalability in video-conferencing

Kurento Media Server docker镜像部署

Kurento官方提供的多人视频会议测试demo源码

Kurento多人视频会议demo使用该说明