> 最近一直在研究WebRTC相关的知识，学习了P2P的链接建立的方式实现了两台终端的互联。也学习了经过服务器中转的广播的工作的模式。最后自己实现了一个经过服务器中转的多人通信的语音的demo。

WebRTC官方网站

WebRTC是什么

WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose.

Our mission: To enable rich, high-quality RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allow them all to communicate via a common set of protocols.

The WebRTC initiative is a project supported by Google, Mozilla and Opera, amongst others. This page is maintained by the Google Chrome team.

以上内容是官网的简介，简单的说WebRTC是一个免费的，开源的，提供在浏览器和手机等终端之间的实时通信的协议。它提供很多渠道和简单的API。rtc团队的最大的愿望就是提供最好的服务给我们。

我们的使命：使更大量、高效的rtc的应用被开发出来给浏览器，手机等平台和更多的设备，使他们之间沟通通过这个协议。

rtc是由谷歌、火狐、欧朋甚至更多共同支持的，目前主要由谷歌的团队来维护。

WebRTC所需要的三个服务器

> 虽然rtc可以实现端对端通信，也可以实现利用服务器中转的通信，但是它并没有我们想象中的那么 简单，我们需要有三个服务器来维护我们的通信的过程

- 房间服务器(Room Server) 房间服务器是一个用来创建和维护我们通话状态的服务器，可以通过http协议进行通信，我们在加入房间和离开房间等过程，需要用到这个服务器，这个服务器可以将信令的配置信息告诉本地
- 信令服务器(Signaling Server) 信令服务器是用来管理和协助通话终端建立点对点通话的工作
- 用来控制通信发起或者结束的链接控制信息
- 当发生异常时会进行转发
- 各自一方媒体流元数据，可以是一些流编码与解码等功能
- 可以使各个终端之间建立安全的链接
- 提供外界所能看到的网络上的数据，例如广域网上面的IP地址、端口等
- 防火墙打洞服务器(STUN/TURN/ICE Server) 我们大部分人在互联网中都处在防火墙后面或者处在私有子网的路由器的后面，这样导致我们的终端的IP地址并不是广域网中的IP地址，所以导致我们不能直接进行通信，所以我们需要一个穿越防火墙或者路由(NAT)路由器，让两个同时处在私有网络中的计算机能够通讯起来。
- STUN协议可以解决家用(NAT)路由器环境的打洞问题，但是对于大部分企业网络环境不是很好
- TURN协议可以很好的弥补STUN的不足
- ICE协议是结合了以上两种的综合性的解决方案，是通过offer/answer模型建立基于UDP的通讯。ICE是offer/answer模型的扩展，通过在offer和answer的SDP(Session Description Protocol)里面包含多种IP地址和端口，然后对本地SDP和远程SDP里面的IP地址进行配对，然后通过P2P连通性检查进行连通性测试工作，如果测试通过即表明该传输地址对可以建立连接。其中IP地址和端口（也就是地址）有以下几种：本机地址、通过STUN服务器反射后获取的server-reflexive地址（内网地址被NAT映射后的地址）、relayed地址（和TURN转发服务器相对应的地址）及Peer reflexive地址等。

WebRTC的使用

简介

其实WebRTC是在全平台提供较为类似的接口的，逻辑更是完全一样，所以做别的开发的，也可以了解一下这个流程。

这里介绍的是经过服务器中转的多端语音的大概的流程：

首先说一下，为什么目前我只做了语音，因为经过服务器中转之后，每个终端需要做的事情其实很简单，就是将本地的视频流或者语音流进行上传，然后接收来自远端的流，在上传方面语音流和视频流并没有什么区别但是涉及到接收的时候，语音流和视频流就有了区别，因为语音流，即使是再多我们也可以只接收一个，因为语音流是非常容易合并在一起的，我们可以在服务端做一个语音流合并的操作，所以我们每一个终端只需要接收一个流就可以，但是视频流不可以这么做，我们需要同时维护多个视频流。

流程

假设我们现在具有多个终端和一个服务器，想要通信，那么我们只要所有的终端都做同一件事情就可以，那就是上传本地流，接收服务器的流，就可以了。

假设我们想要实现客户端A和服务器之间的通信：

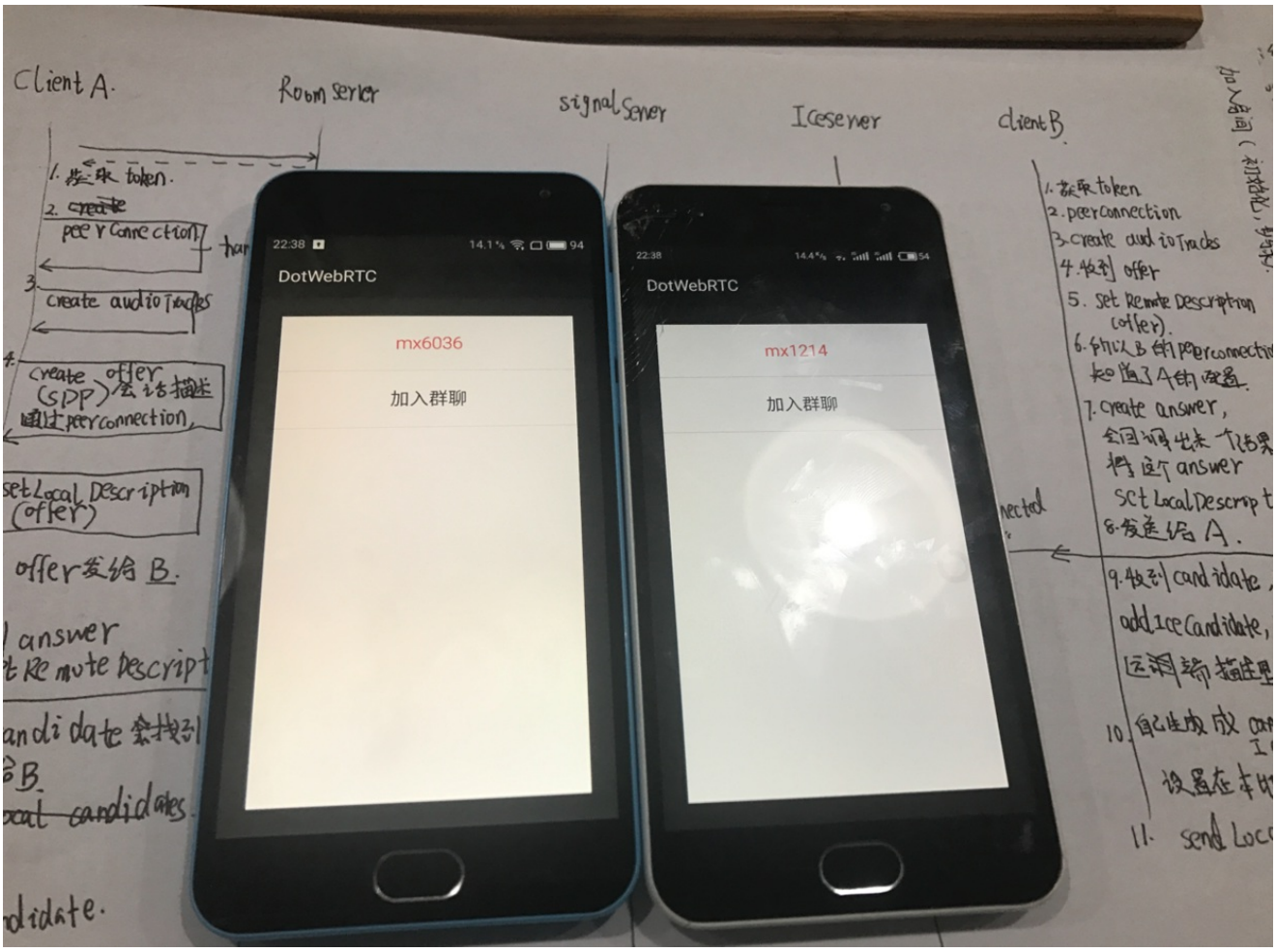
1.我们需要在我们的RoomServer上面获取token 2.我们需要建立本地的PeerConnertion 3.创建本地流createAudioTrack(将本地流创建，这个流就是我们之后要上传用到的流) 4.创建会话描述createOffer(SDP)，sdp是一个会话描述，它包含着我们通信的标准

```
v=0
o=carol 28908764872 28908764872 IN IP4 100.3.6.6        //会话ID号和版本
s=-                                      //用于传递会话主题
t=0 0                                    //会话时间，一般由其它信令消息控制，因此填0
c=IN IP4 192.0.2.4               //描述本端将用于传输媒体流的IP
m=audio 0 RTP/AVP 0 1 3      //媒体类型 端口号 本端媒体使用的编码标识（Payload）集
a=rtpmap:0 PCMU/8000 //rtpmap映射表，各种编码详细描述参数，包括使用带宽（bandwidth）
a=rtpmap:1 1016/8000
a=rtpmap:3 GSM/8000
a=sendonly      //说明本端媒体流的方向，取值包括sendonly/recvonly/sendrecv/inactive
a=ptime:20                               //说明媒体流打包时长
m=video 0 RTP/AVP 31 34
a=rtpmap:31 H261/90000
a=rtpmap:34 H263/90000
```

以上只是一个简单的sdp的例子，供大家参考

5.将会话描述设置在本地(setLocalDescription) 6.将offer发送给服务器 7.收到answer，这个就是远程流的标准 8.将远程流的标准设置在远程流标准上面setRemoteDescription 9.我们第二步初始化PeerConnection的时候会产生一个回调，系统会自动为我们收集candidate，candidate是我们在公网中的位置信息，以及服务器到我们的多种较优的路径，我们要将这个信息发送给服务端

这时我们的链接已经建立完成了，我们便可以将本地的流发送到服务器，也可以下载到服务器的流了。就可以实现多端的通信了。



主要代码

初始化PeerConnectionFactory

```
private PeerConnectionFactory peerConnectionFactory;
if (!PeerConnectionFactory.initializeAndroidGlobals(_context,true,false,false)){
            Log.i("lin","---*lin*--->   initInternal    error   ");
            userListener.onError();
            return;
        }
peerConnectionFactory = new PeerConnectionFactory();
```

初始化PeerConnection

```
PeerConnection peerConnection = peerConnectionFactory.createPeerConnection(configuration,constraints,

//具有的方法
peerConnection.createOffer(SdpObserver var1, MediaConstraints var2);

peerConnection.createAnswer(SdpObserver var1, MediaConstraints var2);

peerConnection.createOffer(SdpObserver var1, MediaConstraints var2);

peerConnection.setRemoteDescription(SdpObserver var1, SessionDescription var2);

peerConnection.updateIce(List<PeerConnection.IceServer> var1, MediaConstraints var2);

peerConnection.addIceCandidate(IceCandidate candidate);

peerConnection.removeStream(MediaStream stream) ;

peerConnection.getStats(StatsObserver observer, MediaStreamTrack track) ;

//以上方法都是有回调的
```

本地流的初始化

```
private MediaStream localStream;
localStream = peerConnectionFactory.createLocalMediaStream("ARDAMS");
audioSource =  peerConnectionFactory.createAudioSource(new AudioMediaConstrains());
localAudioTrack =  peerConnectionFactory.createAudioTrack("ARDAMSa0", audioSource);
localAudioTrack.setEnabled(true);
localPeer.addLocalStream(localStream);
```

源码链接

以上便是根据最近看到的所总结的东西吧。