WebRTC基础介绍

什么是webRTC

WebRTC 全称是（Web browsers with Real-Time Communications (RTC)
大概2011年，谷歌收购了 GIPS，它是一个为 RTC 开发出许多组件的公司，例如编解码和回声消除技术。Google 开源了 GIPS 开发的技术，并希望将其打造为行业标准。

> 在QQ2004版后，通话质量大有改善，就是采用了GIPS的技术。并第一次表明版权：本软件中使用的GIPS语音引擎和相关商标为Global IP Sound AB公司版权所有~~~~

收购花了一大笔钱，谷歌说开源就开源，确实不得不佩服，但显然对于Googl来说，打造音视频的开源生态有着更大的价值。“浏览器 + WebRTC”就是 Google 给出的一个答案。而它的终极目标就是在浏览器之间实现音视频通信。

发展至今日，WebRTC在浏览器的支持性已经大大增强。
WebRTC是一个免费、开放的项目。使web浏览器通过简单的JavaScript api接口实现实时通信功能。

WebRTC应用场景

1. 在线教育
2. 多人音视频实时通话
3. 网络直播

WebRTC原理与架构

 1234

Web API层：面向开发者提供标准API（javascirpt），前端应用通过这一层接入使用WebRTC能力。

C++ API层：面向浏览器开发者，使浏览器制造商能够轻松地实现Web API方案。

音频引擎（VoiceEngine）：音频引擎是一系列音频多媒体处理的框架，包括从视频采集卡到网络传输端等整个解决方案。

1. iSAC/iLBC/Opus等编解码。
2. NetEQ语音信号处理。

3. 回声消除和降噪。

视频引擎（VideoEngine）： 是一系列视频处理的整体框架，从摄像头采集视频、视频信息网络传输到视频显示整个完整过程的解决方案。

1. VP8编解码。
2. jitter buffer：动态抖动缓冲。
3. Image enhancements：图像增益。

传输（Transport)：传输 / 会话层，会话协商 + NAT穿透组件。

1. RTP 实时协议。
2. P2P传输 STUN+TRUN+ICE实现的网络穿越。

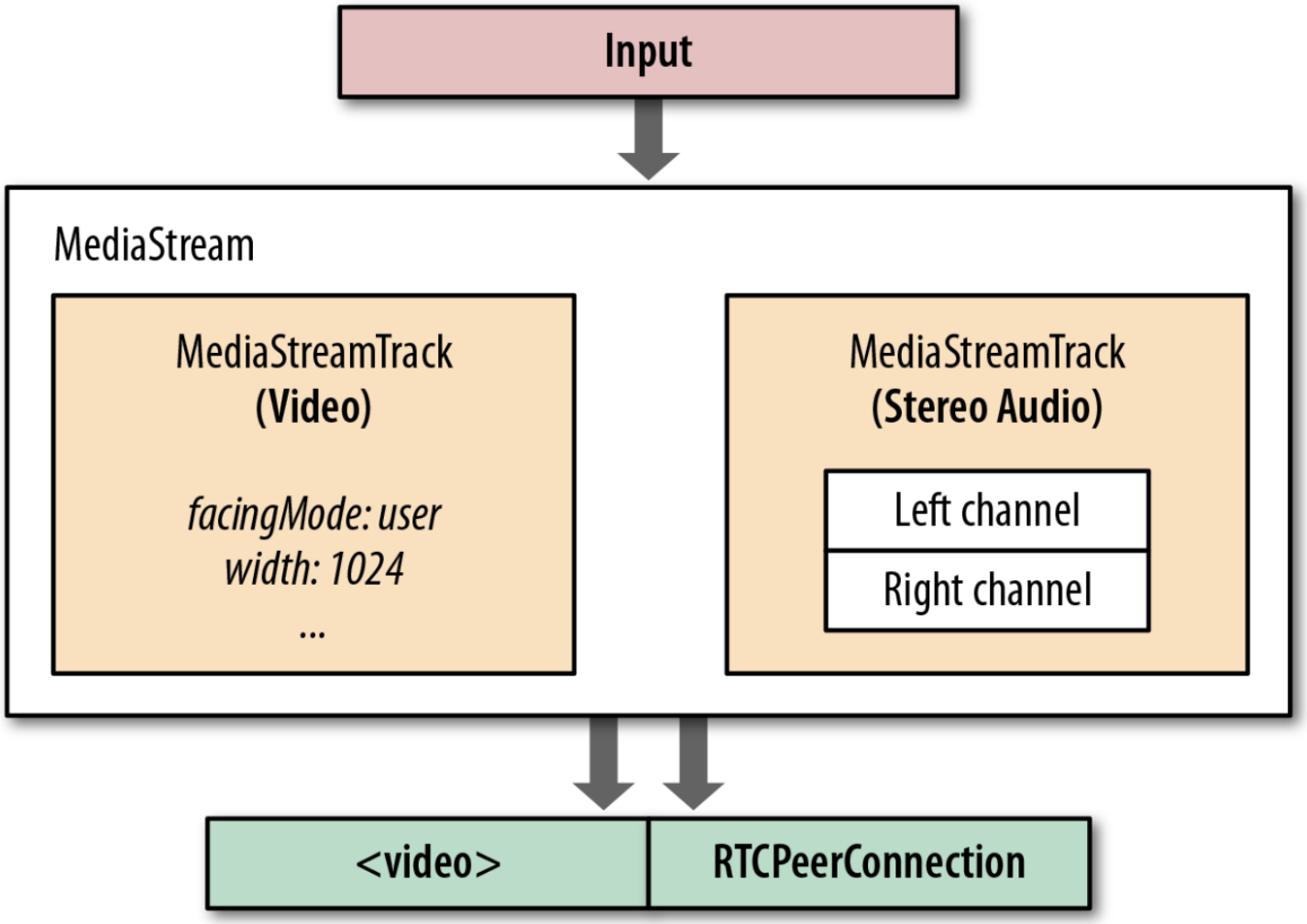
硬件模块：音视频的硬件捕获以及NetWork IO相关。

WebRTC重要的类及概念

Network Stream API - MediaStream（媒体流）/MediaStreamTrack（媒体轨）

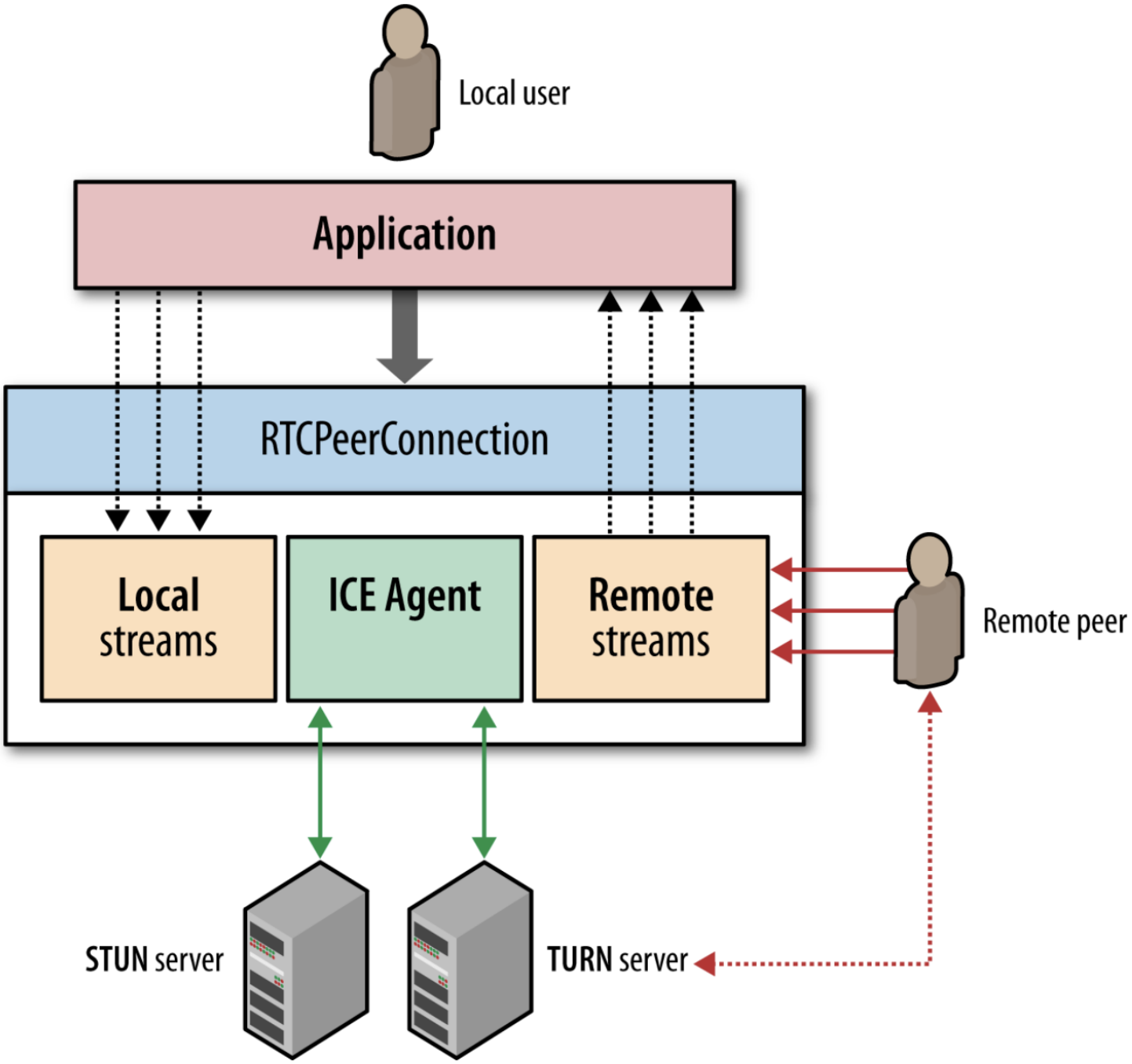
这个类并不完全属于WebRTC的范畴，但是在本地媒体流获取，及远端流传到vedio标签播放都与WebRTC相关。
MS 由两部分构成： MediaStreamTrack 和 MediaStream。

- MediaStreamTrack 媒体轨，代表一种单类型数据流，可以是音频轨或者视频轨。
- MediaStream 是一个完整的音视频流。它可以包含 >=0 个 MediaStreamTrack。它主要的作用就是确保几个媒体轨道是同步播放。



RTCPeerConnection

WebRTC使用RTCPeerConnection，用于实现peer跟peer之间RTC连接，继而**无需服务器**就能传输音视频数据流的连接通道。（直播的实际生产中还是需要服务器）。



这么说过于抽象，为了帮助理解，可以用一个**不太恰当**但有助于理解的比喻： `RTCPeerConnection` 就是一个**高级且功能强大**的用于**传输音视频数据**而建立类似Websocket链接通道，只不过它不仅可以client to server还可以

之所以说是高级且强大，是因为它作为WebRTC web层核心API，让你无须关注数据传输延迟抖动、音视频编解码，音画同步等问题。直接使用PeerConnection 就能自动用上这些浏览器提供的底层封装好的能力。

Peer-to-peer Data API

RTCDataChannel可以建立浏览器之间的点对点通讯。常用的通讯方式有websocket, ajax和等方式。websocket虽然是双向通讯，但是无论是websocket还是ajax都是客户端和服务器之间的通讯，你必须配置服务器才可以进行通讯。

而由于RTCDATAChannel借助RTCPeerConnection无需经过服务器，就可以提供点对点之间的通讯，无需/(避免)服务器了这个中间件。

浏览器的音视频采集及设备管理

音视频采集基本概念

在讲浏览器提供的用JS 采集音视频API之前，需要先了解音视频采集的基本概念。

- **摄像头**：用于捕捉（采集）图像和视频。
- **帧率**:
  Frame rate 摄像头一秒钟采集图像的次数称为帧率。帧率越高，视频越流畅，但每秒传输率也越大，宽带占用就越高。而在显示器上，同样的概念称之为刷新率，就越高越好。
- **分辨率**:
- 分辨率是用于度量视频图像内数据量多少的一个参数，通常表示成ppi。一般有1080P、720P、320P 等。宽高比一般为4:3或16:9。和帧率相同，分辨率越高越清晰，但在直播中占用的宽带越多。因此分辨率应该根据网络情况进行动态调整。
- **麦克风**: 用于采集音频数据。
- **采样率**：指录音设备在一秒钟内对声音信号的采样次数，采样率越高还原声音越真实。8,000 Hz 是电话通话所用采样率, 对于人的说话已经足够。
- **轨（Track）**: WebRTC 中的“轨”借鉴了多媒体的概念。“轨”在多媒体中表达的就是每条轨数据都是独立的，分为音频轨、视频轨。
- **流（Stream）**: 可以理解为容器。在 WebRTC 中，“流”可以分为媒体流（MediaStream）和数据流（DataStream）。

音视频设备与采集

getUserMedia

getUserMedia 方法在浏览器中访问音视频设备非常简单。

```
var promise = navigator.mediaDevices.getUserMedia(constraints);
```

结果会通过**Promise**返回stream，用URL.createObjectURL/srcObject转换后，设置为Video或Audio元素的src属性来进行播放。

失败时promise catchError 可能的异常有：

1. AbortError：硬件问题
2. NotFoundError：找不到满足请求参数的媒体类型。
3. NotReadableError：操作系统上某个硬件、浏览器或者网页层面发生的错误导致设备无法被访问。
4. TypeError：类型错误，constraints对象未设置空，或者都被设置为false。
5. OverConstrainedError：指定的要求无法被设备满足。
6. SecurityError：安全错误，需要用户浏览器设置中开启。
7. NotAllowedError：用户拒绝了当前的浏览器实例的访问请求；或者用户拒绝了当前会话的访问；或者用户在全局范围内拒绝了所有媒体访问请求。

MediaStreamConstraints 参数

媒体约束-MediaStreamConstraints，可以在 getusermedia时指定 MediaStream 中要包含哪些类型的媒体轨，并且设置一些限制。

1. 可以指定采集音频还是视频，或是同时对两者进行采集。

```
const mediaStreamContrains = {
    video: true,
    audio: true
};
```

2. 也可以进一步对媒体做限制。

```
const mediaStreamContrains = {
    video: {
        frameRate: {
            min: 20
        },
        width: {
            min: 640,
            ideal: 1280
        },
        height: {
            min: 360,
            ideal: 720
        },
        aspectRatio: 16/9
    },
    audio: {
        echoCancellation: true,
        noiseSuppression: true,
        autoGainControl: true
    }
};
```

使用采集到的音视频媒体流

通过getUserMedia采集到的媒体流,可以再本地直接播放使用。

```
<video autoplay playsinline></video>


const mediaStreamContrains ={
    video: true
};
const localVideo = document.querySelector('video');

function gotLocalMediaStream(mediaStream) {
  localVideo.srcObject = mediaStream;
}

function handleLocalMediaStreamError(error) {
  console.log('navigator.getUserMedia失败: ', error);
}
navigator.mediaDevices.getUserMedia(mediaStreamContrains).then(gotLocalMediaStream).cat
```

音视频设备管理

MediaDevices接口提供了访问（连接到计算机上的）媒体设备（如摄像头、麦克风）以及屏幕分享的方法。

我们可以通过它，获取可用的音视频设备列表。

MediaDeviceInfo，它表示的是每个输入 / 输出设备的信息:

- deviceID： 设备的唯一标识
- label： 设备名称
- kind：设备种类：可用于识别出是音频设备还是视频设备，是输入设备还是输出设备。

需要注意的是，出于安全原因，**除非用户已被授予访问媒体设备的权限（要想授予权限需要使用 HTTPS 请求），否则 label 字段始终为空。** 另外，label 可以用作指纹识别机制的一部分，以识别是否是合法用户。

获取音视频设备列表

```
MediaDevices.enumerateDevices().then((deviceList)=>{console.log(deviceList)})
```

通过调用 `navigator.MediaDevices.enumerateDevices()` 返回每一个 MediaDeviceInfo，并将每个 MediaDeviceInfo 中的基本信息打印出来，也就是我们想要的每个音视频设备的基本信息。

可以通过 kind 字段再将设备区分为：音频/视频设备，输入/输出设备。
根据deviceid，能知道该设备是否为**默认设备**。

> 这个音频设备为例，将耳机插入电脑后，耳机就变成了音频的默认设备；将耳机拔出后，默认设备又切换成了系统的音频设备。

浏览器端的音视频录制

相关概念

录制从端来说， 可以分为**服务端录制**和**客户端录制**。

- 服务端录制：无须心客户端因电脑问题造成录制失败（如磁盘空间不足、CPU 占用率过高等问题）；缺点是实现的复杂度很高。
- 客户端录制：优点是方便录制者（如老师）操控，所录制的视频清晰度高,实现相对简单。缺点是，录制时会开启的编码器，很耗CPU，且对内存和硬盘要求也高，一旦硬件占高负载会容易造成程序卡死。

在 JavaScript 中，有很多用于存储二进制数据的类型，这些类型包括：ArrayBuffer、ArrayBufferView 和 Blob。WebRTC 录制音视频流之后，最终是通过 Blob 对象将数据保存成多媒体文件的。

ArrayBuffer

ArrayBuffer 对象表示通用的、固定长度的二进制数据缓冲区。因此，你可以直接使用它存储图片、视频等内容。

```
let buffer = new ArrayBuffer(16); // 创建一个长度为 16 的 buffer
let view = new Uint32Array(buffer);
```

```
let buffer = new Uint8Array([255, 255, 255, 255]).buffer;
let dataView = new DataView(buffer);
```

一开始生成的 buffer 是不能被直接访问的。只有将 buffer 做为参数生成一个具体的类型的新对象时（如 Uint32Array 或 DataView），这个新生成的对象才能被访问。

ArrayBufferView

ArrayBufferView 并不是一个具体的类型，而是代表不同类型的 Array 的描述。这些类型包括：Int8Array、Uint8Array、DataView 等。也就是说 Int8Array、Uint8Array 等才是 JavaScript 在内存中真正可以分配的对象。

Blob

Blob（Binary Large Object）是 JavaScript 的大型二进制对象类型，WebRTC 最终就是使用它将录制好的音视频流保存成多媒体文件的。而它的底层是由上面所讲的 ArrayBuffer 对象的封装类实现的，即 Int8Array、Uint8Array 等类型。

```
var aBlob = new Blob( array, options );
```

浏览器录制方法

浏览器为我们提供了一个录制音视频的类，即 MediaRecorder。

```
var mediaRecorder = new MediaRecorder(stream[,options]);
```

参数解释：

- stream: 通过 getUserMedia 获取的本地视频流或通过 RTCPeerConnection 获取的远程视频流。
- options:可选项，指定视频格式、编解码器、码率等相关信息，如 mimeType: 'video/webm;codecs=vp8'。

录制流

```
var buffer;

//当该函数被触发后，将数据压入到blob中
function handleDataAvailable(e){
        if(e && e.data && e.data.size > 0){
                buffer.push(e.data);
        }
}

function startRecord(){
        buffer = [];
        //设置录制下来的多媒体格式
        var options = {
                mimeType: 'video/webm;codecs=vp8'
        }

        //判断浏览器是否支持录制
        if(!MediaRecorder.isTypeSupported(options.mimeType)){
                console.error(`${options.mimeType} is not supported!`);
                return;
        }

        try{
                //创建录制对象
                mediaRecorder = new MediaRecorder(window.stream, options);
        }catch(e){
                console.error('Failed to create MediaRecorder:', e);
                return;
        }

        //当有音视频数据来了之后触发该事件
        mediaRecorder.ondataavailable = handleDataAvailable;
        //开始录制
        mediaRecorder.start(10);
}

...
```

播放录制文件

首先根据 buffer 生成 Blob 对象；然后，根据 Blob 对象生成 URL，并通过 video标签进行播放。

```
<video id="playback"></video>
```

```
var blob = new Blob(buffer, {type: 'video/webm'});
playback.src = window.URL.createObjectURL(blob);
playback.srcObject = null;
playback.controls = true;
playback.play();
```

屏幕分享

桌面分享可以当做特殊音视频数据来看待，在实时音视频，尤其是在在线教育场景中，尤为常见。

对于屏幕分享者：每秒钟多次抓取的屏幕，每一屏数据取它们的差值，然后对差值进行压缩；如果差值超过一定程度，则单独对这一屏数据进行帧内压缩，该压缩方法，类似于视频编码中GOP的I帧。

对于远端观看/控制者：收到数据进行解压缩，还原成画面播放即可。另外如果有操控指令，需要实现对应的信令系统，并自行信令控制。

总结为以下流程：
抓屏、压缩编码、传输、解码、显示、控制。和音视频流程基本一致。

屏幕分享的协议有：

- RDP（Remote Desktop Protocal）：windows下的桌面共享协议。
- VNC（Virtual Network Console）：在不同的操作系统上共享远程桌面，像 TeamViewer、RealVNC 都是在使用这个协议。

WebRTC的屏幕分享

由于webrtc不包含控制部分，因此他的处理过程只使用了视频方式，而不需要信令控制。因此它和RDP/VNC还是存在差异。

1. 桌面数据的采集
   在桌面数据采集上，和VNC一样是通过各平台提供的API实现的。
   BitBlt、Hook、DirectX等。最新的 WebRTC 都是使用的这种方式GetWindowDC：可以通过它来抓取窗口。
2. 桌面数据的编码
   WebRTC 对桌面的编码使用的是视频编码技术，即 H264/VP8 等（好处就是压缩率高）；但 RDP/VNC 则不一样，它们使用的是图像压缩技术。
3. 传输
   webrtc有根据网络情况的调节能力，网络差时会进行丢数据保证实时性。
4. 解码渲染
   解码同第二点，渲染一般会通过 OpenGL/D3D 等 GPU进行渲染。

通过 `getDisplayMedia` API 来采集桌面：

> 特别注意: 在桌面采集的参数里，不能对音频进行限制了。也就是说，不能在采集桌面同时采集音频。

```
//get桌面数据流
function getDeskStream(stream){
    localStream = stream;
}

//采集桌面
function shareDesktop(){
        //只有在 PC 下才能抓取桌面
  if(utils.isPC){
                //开始捕获桌面数据
       navigator.mediaDevices.getDisplayMedia({video: true})
            .then(getDeskStream)
            .catch(handleError);
      return true;
  }
  return false;

}
```

而展示和录制和音视频的展示和录制是实现方式基本相同。

以上就是浏览器端WebRTC通过设备采集音视频数据及其播放与录制的相关介绍。有了数据，接下来才可以使用WebRTC来实现实时音视频通讯。

信令通道服务

SDP协议与WebRTC媒体协商offer/answer

WebRTC的NAT打洞与连接：STUN/TUN/ICE

WebRTC核心API之RTCPeerConnection

WebRTC网络传输协议及安全加密

WebRTC的质量分析

多人音视频通讯的常见架构