

03 | 如何使用浏览器给自己拍照呢？

2019-07-20 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 10:50 大小 9.93M



在之前的文章中，我向你介绍了如何在浏览器中利用 WebRTC 采集音视频数据。那么，是否可以通过相同的技术进行拍照呢？没错，这是完全可行的。

现代的浏览器功能越来越强大，你不光可以通过它进行拍照，而且还可以对拍下来的照片进行各种滤镜处理。

为了实现上述功能，你需要了解并掌握以下三个知识点：

如何从采集到的视频中获取到图片？

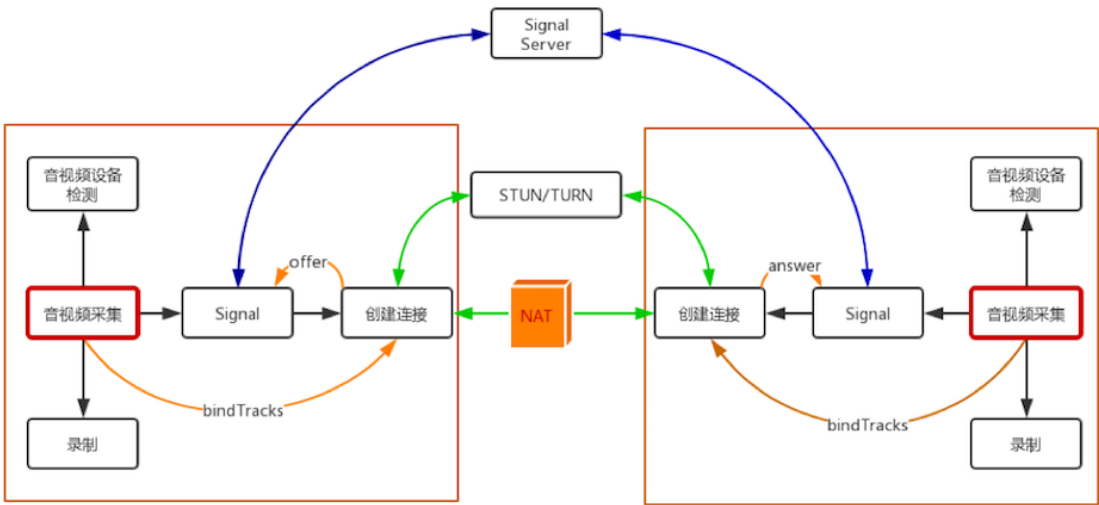
如何将处理后的图片保存成文件？

如何对获取到的图片进行滤镜处理？

这三方面的知识点就是本文要交付的重点内容。下面我们先学习与之相关的基础知识和原理，然后再对这几个知识点逐一进行讲解，各个击破。

在 WebRTC 处理过程中的位置

在正式进入主题之前，咱们仍然按老规矩，看看本篇文章所介绍的内容在整个 WebRTC 处理过程中的位置。如下图所示：



WebRTC 1 对 1 音视频实时通话过程示意图

你可以看到，这张图与[《01 | 原来通过浏览器访问摄像头这么容易》](#)文章中的图一模一样。没错，咱们本篇文章所涉及的知识点仍然属于音视频数据采集的部分。

基础知识

在正式讲解如何进行**拍照**之前，你需要先了解非编码帧（解码帧）和编码帧这两个知识点，这会有利于你对后面拍照实现内容的理解。

1. 非编码帧

好多人小时候应该都学过，在几张空白的纸上画同一个物体，并让物体之间稍有一些变化，然后连续快速地翻动这几张纸，它就形成了一个小动画。

音视频播放器就是利用这样的原理来播放音视频文件的。当你要播放某个视频文件时，播放器会按照一定的时间间隔连续地播放从音视频文件中**解码后的视频帧**，这样视频就动起来了。同理，播放从摄像头获取的视频帧也是如此，只不过从摄像头获取的本来就是**非编码视频帧**，所以就不需要解码了。

通过上面的描述，你应该能得到以下两点信息：

播放的视频帧之间的时间间隔是非常小的。如按每秒钟 20 帧的帧率计算，每帧之间的间隔是 50ms。

播放器播的是**非编码帧（解码后的帧）**，这些非编码帧就是一幅幅独立的图像。

从摄像头里采集的帧或通过解码器解码后的帧都是**非编码帧**。非编码帧的格式一般是 YUV 格式或是 RGB 格式。关于 YUV 与 RGB 的相关知识，我在[上一篇文章](#)中已向你做过简要介绍，这里就不再赘述了。

2. 编码帧

相对于**非编码帧**，通过编码器（如 H264/H265、VP8/VP9）压缩后的帧称为**编码帧**。这里我们以 H264 为例，经过 H264 编码的帧包括以下三种类型。

I 帧：关键帧。压缩率低，可以单独解码成一幅完整的图像。

P 帧：参考帧。压缩率较高，解码时依赖于前面已解码的数据。

B 帧：前后参考帧。压缩率最高，解码时不光依赖前面已经解码的帧，而且还依赖它后面的 P 帧。换句话说就是，**B 帧后面的 P 帧要优先于它进行解码，然后才能将 B 帧解码。**


关于编码这块的内容，目前你只需了解上面这些知识即可。

通过上面的介绍，现在你应该已经清楚地知道了：**从播放器里获取的视频帧一定是非编码帧。也就是说，拍照的过程其实是从连续播放的一幅幅画面中抽取正在显示的那张画面。**

如何获取视频流


在获得照片之前，你首先要通过浏览器的 API 获取视频流，并通过 HTML5 的 <video> 标签将视频播放出来。实际上，这些知识我已经在[《01 | 原来通过浏览器访问摄像头这么容易》](#)中介绍过了，这里我就不做过多的描述了。咱们还是直接上代码吧。

HTML 部分代码如下：

 复制代码

```
1 <html>
2 <head>
3   <title>WebRTC take picture</title>
4 </head>
5 <body>
6   <video autoplay playsinline id="player">
7   <script src="./js/client.js"></script>
8 </body>
9 </html>
```

上面这段代码很简单，就是定义了一个 video 标签，用于播放从摄像头获取到的视频流。另外，它还引入了一段 JavaScript 脚本：

 复制代码

```
1 'use strict'
2
3 // 获取 HTML 页面中的 video 标签
4 var videoplay = document.querySelector('video#player');
5
6 // 播放视频流
7 function gotMediaStream(stream){
8     videoplay.srcObject = stream;
9 }
10
11 function handleError(err){
12     console.log('getUserMedia error:', err);
13 }
14
15 // 对采集的数据做一些限制
16 var constraints = {
17     video : {
18         width: 1280,
19         height: 720,
20         frameRate:15,
21     },
22     audio : false
```

```
23         }
24
25 // 采集音视频数据流
26 navigator.mediaDevices.getUserMedia(constraints)
27     .then(gotMediaStream)
28     .catch(handleError);
```

在这段脚本中，我们调用了之前所讲的**getUserMedia**方法，该方法会打开摄像头，并通过它采集音视频流。然后再将采集到的视频流赋值给 HTML 中定义的**video**标签的**srcObject**字段，这样**video**标签就可以从摄像头源源不断地获得视频帧，并将它播放出来了。

以上这些内容，你应该都非常熟悉了。下面的关键点是，获取到视频流后如何从中获取正在显示的视频帧或图片呢？现在就让我们来解决这个问题吧！

如何拍照

实际上，浏览器提供了一个非常强大的对象，称为**Canvas**。你可以把它想像成一块画布，既可以在这块画布上画上点、线或各种图形，也可以将一幅画直接绘制到上面。

在浏览器中，Canvas 的功能非常强大，可以处理很多图表方面的事情，对于这部分知识我们后面还会做详细的介绍。而这里你只需关注它获取图片这一个知识点。


我们还是通过代码来讲解，这样更一目了然。首先，在 HTML 中增加以下代码：

 复制代码

```
1 ...
2 <button id="TakePhoto">Take</button>
3 ...
4 <canvas id="picture"></canvas>
5 ...
```

上面的 HTML 代码段，包括一个 `<canvas>` 标签和一个 `<button>` 标签。我们的设想是，当点击拍照按钮时，就可以从视频流中获取到一张当时正在显示的图片了。


显然，光有 HTML 部分肯定是不行的，还需要下面的 JavaScript 脚本进行控制。增加 JavaScript 代码如下：

 复制代码

```
1 ...
2
3 var picture = document.querySelector('canvas#picture');
4 picture.width = 640;
5 picture.height = 480;
6
7 ...
8
9 picture.getContext('2d').drawImage(videoplay, 0, 0, picture.width, picture.height);
10
11 ...
```

在上面的 JavaScript 代码中，首先获得 HTML 中的 Canvas 标签，并设置了 Canvas 的宽高；然后调用 Canvas 上下文的 drawImage 方法，这样就可以从视频流中抓取当时正在显示的图片了。

这里最关键的点就是 **drawImage** 方法，其方法格式如下：

 复制代码

```
1 void ctx.drawImage(image, dx, dy, dWidth, dHeight);
```

image：可以是一幅图片，或 HTMLVideoElement。

dx, dy：图片起点的 x、y 坐标。

dWidth：图片的宽度。

dHeight：图片的高度。

该方法的第一个参数特别重要，它既可以是一幅图片，也可以是一个 Video 元素。而 HTML 中的 <video> 标签就是一个 video 元素，所以它可以当作是 drawImage 方法的第一个参数。这样就可以通过 Canvas 获取到照片了。

通过上面的方法，你就拍照成功了哈，是不是很简单？

如何保存图片


照片拍好后，如何将它保存到本地文件系统中呢？浏览器同样给我们提供了非常方便的方法，让我们来看一下具体代码吧。

HTML 要先增加如下代码：

 复制代码

```
1 ...
2 <div>
3     <button id="save"> 保存 </button>
4 </div>
5 ...
```

也就是当你点击保存这个 `<button>` 的时候，就可以将前面 Canvas 抓取的图片保存下来。不过，`<button>` 只是触发一个事件，真正做事儿的是下面的 JavaScript 代码。具体逻辑如下：

 复制代码

```
1 ...
2
3 function download(url){
4     var oA = document.createElement("a");
5     oA.download = 'photo';// 设置下载的文件名，默认是'下载'
6     oA.href = url;
7     document.body.appendChild(oA);
8     oA.click();
9     oA.remove(); // 下载之后把创建的元素删除
10 }
11
12 ...
13 document.querySelector("button#save").onclick = function (){
14     download(canvas.toDataURL("image/jpeg"));
15 }
16 ....
```

在上面的代码中，当用户点击**保存**按钮时，会调用一个匿名函数。该函数的逻辑如下：

首先，通过 Canvas 的 toDataURL 方法获得图片的 URL 地址；

然后，将该 URL 地址当作参数传给 download 函数；


最后，download 函数做的事儿比较简单，就是创建一个<a>标签，当用户点击时就将图片下载下来。

通过上面的代码，你就可以通过浏览器为自己拍照，并同时将拍下来的照片保存到文件系统中了。

如何实现滤镜

从视频流中获取到照片后，你还可以通过滤镜为照片增加点特效，这样会让你的照片更加特别。

在浏览器中对于图片的滤镜处理是通过 CSS 来控制的。像前面一样，首先在 HTML 中增加 CSS 的滤镜代码如下：

 复制代码

```
1  ...
2  <head>
3      <style>
4          .none {
5              -webkit-filter: none;
6          }
7
8          .blur {
9              -webkit-filter: blur(3px);
10         }
11
12         .grayscale {
13             -webkit-filter: grayscale(1);
14         }
15
16         .invert {
17             -webkit-filter: invert(1);
18         }
19
20         .sepia {
21             -webkit-filter: sepia(1);
22         }
23
24     </style>
25 </head>
26 <body>
```



```
27 ...
28 <select id="filter">
29     <option value="none">None</option>
30     <option value="blur">blur</option>
31     <option value="grayscale">Grayscale</option>
32     <option value="invert">Invert</option>
33     <option value="sepia">sepia</option>
34 </select>
35 ...
36 </body>
```

上面的 HTML 代码中定义了以下四种 CSS 滤镜。


blur : 模糊度

grayscale : 灰度 (黑白)

invert : 反转

sepia : 深褐色

并增加了一个 `<select>` 标签，以便让用户选择使用不同的滤镜。但最终的控制还是由下面的 JavaScript 脚本来做的，JavaScript 代码如下：

 复制代码

```
1 ...
2 picture.className = filtersSelect.value;
3 ...
```

没错，只需要这样简单的一行代码，你就可以将不同的滤镜应用于获取的照片上，是不是非常简单？学习完这些，你就可以很快实现你想要的滤镜效果了，赶快上手尝试下吧！

小结

本文我向你介绍了如何在浏览器上从视频流中抓取一幅图片，即**照片**。通过上面的介绍，你应该也了解到通过浏览器的 Canvas 来实现这个功能会特别简单。

在文章的最后，我还讲述了如何通过 CSS 对捕获到的图片做特效处理，当然这种特效是比较简单的实现。如果想实现更多更复杂的效果，就要用 WebGL 来实现了。而 WebGL 的知识相对比较复杂，甚至可以另起一个专栏专门来介绍这部分知识。所以，这里我就不做过多的讲解了，有需要的同学可以自行搜索资料学习。

思考时间

文章的最后是通过 CSS 对照片做了特效处理，但当你将这张照片保存下来，并单独打开后你会发现，之前在浏览器上设置的效果没有了，这是什么原因呢？你要如何来解决该问题呢？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。



从 0 打造音视频直播系统

手把手教你打造实时互动音视频直播系统

李超

新东方音视频直播技术专家
前沪江音视频架构师



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 如何通过WebRTC进行音视频设备检测呢？

精选留言 (6)

💬 写留言

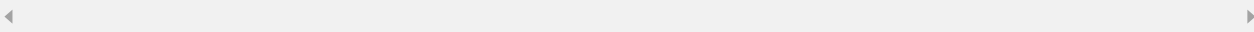


Jim

2019-07-20

代码片段有点太零散，还好有点基础可以自己处理，对新手看起来估计就有点懵逼了。

作者回复: 会有整体的代码供参考的！下周一就会出来哈

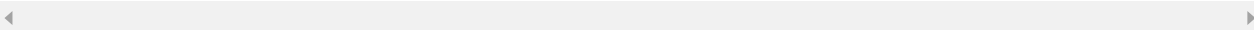


salmon

2019-07-20

老师可以整理代码作为demo传到github吗？前端新手有点整不明白。

作者回复: 下周一就会有有了，正在进行中.....



流浪剑客

2019-07-20

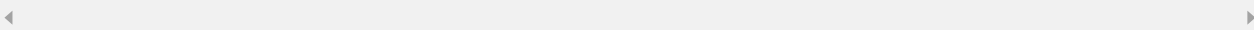
老师，代码没跑起来。需要做如下修改。

画布渲染图片需要改成触发动作，如下：

```
document.querySelector("button#TakePhoto").onclick = function (){  
    picture.getContext('2d').drawImage(videoplay, 0, 0, picture.width,  
    picture.height);...
```

展开 ∨

作者回复: 下周一我会将相关的代码放到github上，耐心等待一下哈！



LongXiaJun

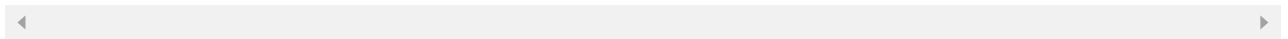
2019-07-20

现在每天都在等老师更新，凌晨快速看一遍，白天写笔记再看一遍，对于没有基础的同学可能会比较吃力，附上我的GitHub代码地址，供大家参考学习：

https://github.com/KuKuXia/Real_Time_Communication_using_WebRTC

展开 ∨

作者回复: 赞！加油！

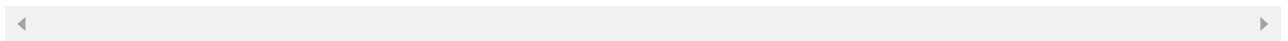


xiao豪

2019-07-20

调用drawImage方法时图片已经画好了，css只是对图片的渲染形式改变而已，所以下载的图片还是原图

作者回复: 可以继续思考，如果能让下载的图片生效呢？



LongXiaJun

2019-07-20

打卡~

展开 ∨

作者回复: 打卡~

