

04 | 可以把采集到的音视频数据录制下来吗？

2019-07-23 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 21:05 大小 19.32M



在音视频会议、在线教育等系统中，**录制**是一个特别重要的功能。尤其是在教育系统，基本上每一节课都要录制下来，以便学生可以随时回顾之前学习的内容。

实现录制功能有很多方式，一般分为**服务端录制**和**客户端录制**，具体使用哪种方式还要结合你自己的业务特点来选择。

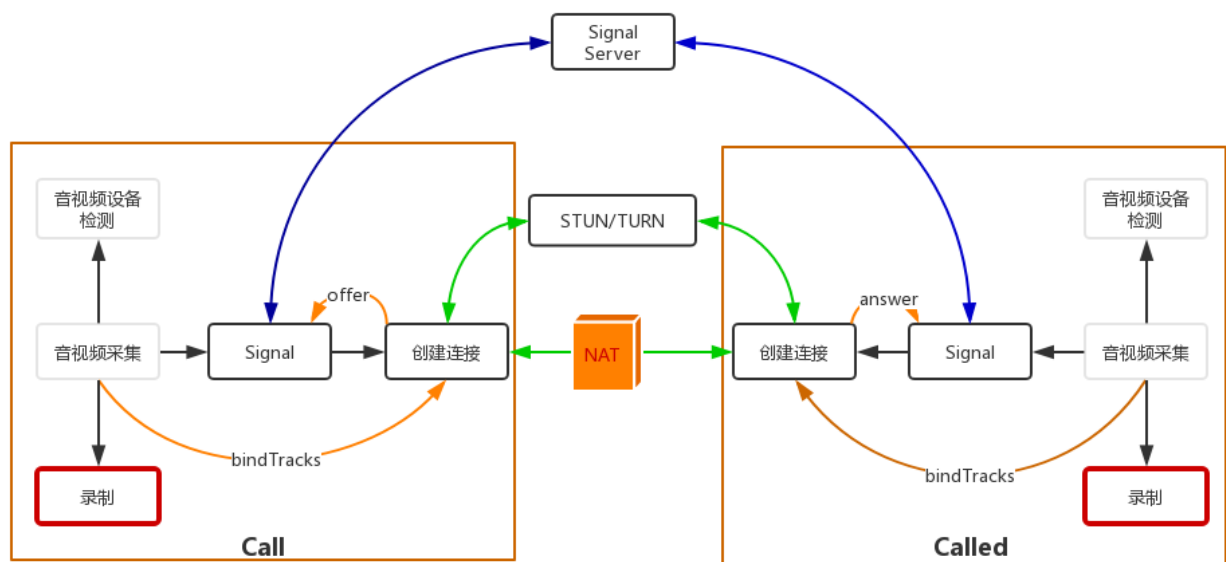
服务端录制：优点是不用担心客户因自身电脑问题造成录制失败（如磁盘空间不足），也不会因录制时抢占资源（CPU 占用率过高）而导致其他应用出现问题等；缺点是实现的复杂度很高。不过由于本文要重点讲解的是接下来的客户端录制，所以这里我就不再深入展开讲解了，你只需要知道有服务端录制这回事就行了，或者如果你感兴趣，也可以自行搜索学习。

客户端录制：优点是方便录制方（如老师）操控，并且所录制的视频清晰度高，实现相对简单。这里可以和服务端录制做个对比，一般客户端摄像头的分辨率都非常高的（如 1280x720），所以客户端录制可以录制出非常清晰的视频；但服务端录制要做到这点就很困难了，本地高清的视频在上传服务端时由于网络带宽不足，视频的分辨率很有可能会被自动缩小到了 640x360，这就导致用户回看时视频特别模糊，用户体验差。不过客户端录制也有很明显的缺点，其中最主要的缺点就是录制失败率高。因为客户端在进行录制时会开启第二路编码器，这样会特别耗 CPU。而 CPU 占用过高后，就很容易造成应用程序卡死。除此之外，它对内存、硬盘的要求也特别高。

这里需要再次说明的是，由于本专栏的文章是**聚焦在 WebRTC 的使用上**，所以我们只讲**如何使用 WebRTC 库实现客户端录制音视频流**。至于服务端录制的相关知识，由于与 WebRTC 库无关，所以我们就不做进一步讲解了。

在 WebRTC 处理过程中的位置

在正式进入主题之前，咱们依旧先来看一下本文在 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

通过上图可以了解到，咱们本文所讲的内容就是**音视频采集**下面的**录制功能**。

基本原理

录制的基本原理还是蛮简单的，但要做好却很不容易，主要有以下三个重要的问题需要你搞清楚。

第一个，录制后音视频流的存储格式是什么呢？比如，是直接录制原始数据，还是录制成某种多媒体格式（如 MP4）？你可能会想，为什么要考虑存储格式问题呢？直接选择一个不就好了？但其实**存储格式的选择对于录制后的回放至关重要**，这里我先留个悬念不细说，等看到后面的内容你自然就会理解它的重要性与局限性了。

第二个，录制下来的音视频流如何播放？是使用普通的播放器播放，还是使用私有播放器播呢？其实，这与你的业务息息相关。如果你的业务是多人互动类型，且回放时也要和直播时一样，那么你就必须使用私有播放器，因为普通播放器是不支持同时播放多路视频的。还有，如果你想让浏览器也能观看回放，那么就需要提供网页播放器。

第三个，启动录制后多久可以回放呢？这个问题又分为以下三种情况。


边录边看，即开始录制几分钟之后用户就可以观看了。比如，我们观看一些重大体育赛事时（如世界杯），一般都是正式开始一小段时间之后观众才能看到，以确保发生突发事件时可以做紧急处理。

录制完立即回放，即当录制结束后，用户就可以回看录制了。比较常见的是一些技术比较好的教育公司的直播课，录制完成后就可以直接看回放了。

录完后过一段时间可观看。大多数的直播系统都是录制完成后过一段时间才可以看回放，因为录制完成后还要对音视频做一些剪辑、转码，制作各种不同的清晰度的回放等等。

接下来，咱们就针对上面的问题详细分析一下。我们先从存储格式的重要性的局限性切入，也正好解答下前面所留的悬念。

录制原始数据的优点是不用做过多的业务逻辑，来什么数据就录制什么数据，这样录制效率高，不容易出错；并且录制方法也很简单，可以将音频数据与视频数据分别存放到不同的二进制文件中。文件中的每一块数据可以由下面的结构体描述：

 复制代码

```
1 struct data
2     int media_type; // 数据类型，0：音频 1：视频
3     int64_t ts;      // timestamp，记录数据收到的时间
4     int data_size;   // 数据大小
5     char* data;      // 指定具体的数据
```

当录制结束后，再将录制好的音视频二进制文件整合成某种多媒体文件，如 FLV、MP4 等。

但它的弊端是，录制完成后用户要等待一段时间才能看到录制的视频。因为在录制完成后，它还要做音视频合流、输出多媒体文件等工作。

那直接录制成某种多媒体格式会怎么样呢？如果你对多媒体文件格式非常熟悉的话，应该知道 **FLV** 格式特别适合处理这种流式数据。因为 FLV 媒体文件本身就是流式的，你可以在 FLV 文件的任何位置进行读写操作，它都可以正常被处理。因此，如果你使用 FLV 的话，就不用像前面录制原始数据那样先将二进制数据存储下来，之后再进行合流、转码那么麻烦了。

不仅如此，采用 FLV 媒体格式进行录制，你还可以进一步优化，将直播的视频按 **N** 分钟为单位，录制成一段一段的 FLV，然后录完一段播一段，这样就实现了上面所讲的**边录边看**的效果了。

但 FLV 也不是万能的。如果你的场景比较复杂（如多人互动的场景），即同时存在多路视频，FLV 格式就无法满足你的需求了，因为 FLV 只能同时存在一路视频和一路音频，而不能同时存在多路视频这种情况。此时，最好的方法还是录制原始数据，然后通过实现私有播放器来达到用户的需求。

当然，即使是单视频的情况下，FLV 的方案看上去很完美，但实际情况也不一定像你想象的那样美好。因为将音视频流存成 FLV 文件的前提条件是音视频流是按顺序来的，而实际上，音视频数据经过 UDP 这种不可靠传输网络时，是无法保证音视频数据到达的先后顺序的。因此，在处理音视频录制时，你不仅要考虑录制的事情，还要自己做音视频数据排序的工作。除此之外，还有很多其他的工作需要处理，这里我就不一一列举了。

不过，好在 WebRTC 已经处理好了一切。有了 WebRTC，你不用再考虑将音视频保存成什么媒体格式的问题；有了 WebRTC，你不用再考虑网络丢包的问题；有了 WebRTC，你不用再考虑音视频数据乱序的问题……这一系列恼人的问题，WebRTC 都帮你搞定了。下面就让我们赶紧应用起来，看一下 WebRTC 是如何进行客户端录制的吧！

基础知识

为便于你更好地理解，在学习如何使用 WebRTC 实现客户端录制之前，你还需要先了解一些基础知识。

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。那这三者与我们今天要讲的录制音视频流有什么关系呢？


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；而 Blob 又与 ArrayBuffer 有着很密切的关系。那 ArrayBuffer 与 ArrayBufferView 又有什么联系呢？接下来，我们就了解一下这 3 种二进制数据类型，以及它们之间的关系吧。

1. 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。


但你并不能直接对 ArrayBuffer 对象进行访问，类似于 Java 语言中的抽象类，在物理内存中并不存在这样一个对象，必须使用其**封装类**进行实例化后才能进行访问。

也就是说，ArrayBuffer 只是描述有这样一块空间可以用来存放二进制数据，但在计算机的内存中并没有真正地为其分配空间。只有当具体类型化后，它才真正地存在于内存中。如下所示：

 复制代码

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

或

 复制代码

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


在上面的例子中，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。

2. 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。


以 `Int8Array` 为例，当你对其实例化时，计算机就会在内存中为其分配一块空间，在该空间中的每一个元素都是 8 位的整数。再以 `Uint8Array` 为例，它表达的是在内存中分配一块每个元素大小为 8 位的无符号整数的空间。

通过这上面的描述，你现在应该知道 `ArrayBuffer` 与 `ArrayBufferView` 的区别了吧？`ArrayBufferView` 指的是 `Int8Array`、`Uint8Array`、`DataView` 等类型的总称，而这些类型都是使用 `ArrayBuffer` 类实现的，因此才统称他们为 `ArrayBufferView`。

3. 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。

`Blob` 对象的格式如下：

 复制代码


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

其中，`array` 可以是 **`ArrayBuffer`**、**`ArrayBufferView`**、**`Blob`**、**`DOMString`** 等类型；`option`，用于指定存储成的媒体类型。

介绍完了这几个你需要了解的基本概念之后，接下来，我们书归正传，看看如何录制本地音视频。

如何录制本地音视频

WebRTC 为我们提供了一个非常方便的类，即 **MediaRecorder**。创建 MediaRecorder 对象的格式如下：

 复制代码

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

其参数含义如下：

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'`。


MediaRecorder 对象还有一个特别重要的事件，即 **ondataavailable** 事件。当 MediaRecorder 捕获到数据时就会触发该事件。通过它，我们才能将音视频数据录制下来。

有了上面这些知识，接下来，我们看一下具体该如何使用上面的对象来录制音视频流吧！

1. 录制音视频流

首先是获取本地音视频流。在 [《01 | 原来通过浏览器访问摄像头这么容易》](#) 一文中，我已经讲过如何通过浏览器采集音视频数据，具体就是调用浏览器中的 `getUserMedia` 方法。所以，这里我就不再赘述了。


获取到音视频流后，你可以将该流当作参数传给 MediaRecorder 对象，并实现 **ondataavailable** 事件，最终将音视频流录制下来。具体代码如下所示，我们先看一下 HTML 部分：

 复制代码

```
1 <html>
2 ...
3 <body>
```

```
4     ...
5     <button id="record">Start Record</button>
6     <button id="recplay" disabled>Play</button>
7     <button id="download" disabled>Download</button>
8     ...
9 </body>
10 </html>
```

上面的 HTML 代码片段定义了三个 **button**，一个用于开启录制，一个用于播放录制下来的内容，最后一个用于将录制的视频下载下来。然后我们再来看一下 JavaScript 控制部分的代码：

 复制代码

```
1  ...
2
3  var buffer;
4
5  ...
6
7  // 当该函数被触发后，将数据压入到 blob 中
8  function handleDataAvailable(e){
9      if(e && e.data && e.data.size > 0){
10         buffer.push(e.data);
11     }
12 }
13
14 function startRecord(){
15
16     buffer = [];
17
18     // 设置录制下来的多媒体格式
19     var options = {
20         mimeType: 'video/webm;codecs=vp8'
21     }
22
23     // 判断浏览器是否支持录制
24     if(!MediaRecorder.isTypeSupported(options.mimeType)){
25         console.error(`${options.mimeType} is not supported!`);
26         return;
27     }
28
29     try{
30         // 创建录制对象
31         mediaRecorder = new MediaRecorder(window.stream, options);
32     }catch(e){
33         console.error('Failed to create MediaRecorder:', e);
```



```

34         return;
35     }
36
37     // 当有音视频数据来了之后触发该事件
38     mediaRecorder.ondataavailable = handleDataAvailable;
39     // 开始录制
40     mediaRecorder.start(10);
41 }
42
43 ...

```

当你点击 Record 按钮的时候，就会调用 **startRecord** 函数。在该函数中首先判断浏览器是否支持指定的多媒体格式，如 webm。如果支持的话，再创建 MediaRecorder 对象，将音视频流录制成指定的媒体格式文件。


实际存储时，是通过 ondataavailable 事件操作的。每当 ondataavailable 事件触发时，就会调用 handleDataAvailable 函数。该函数的实现就特别简单了，直接将数据 push 到 buffer 中，实际在浏览器底层使用的是 Blob 对象。

另外，在开启录制时，可以设置一个毫秒级的时间片，这样录制的媒体数据会按照你设置的值分割成一个个单独的区块，否则默认的方式是录制一个非常大的整块内容。分成一块一块的区块会提高效率和可靠性，如果是一整块数据，随着时间的推移，数据块越来越大，读写效率就会变差，而且增加了写入文件的失败率。

2. 回放录制文件

通过上面的方法录制好内容后，又该如何进行回放呢？让我们来看一下代码吧！

在 HTML 中增加下面的代码：


 复制代码

```

1 ...
2 <video id="recvideo"></video>
3 ...

```

在 HTML 中只增加了一个 `<video>` 标签，用于播放录制的内容。下面的 JavaScript 是将录制内容与 `<video>` 标签联接到一起：


 复制代码

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

在上面的代码中，首先根据 `buffer` 生成 `Blob` 对象；然后，根据 `Blob` 对象生成 `URL`，并通过 `<video>` 标签将录制的内容播放出来了。

3. 下载录制好的文件

那如何将录制好的视频文件下载下来呢？代码如下：

 复制代码

```
1 btnDownload.onclick = ()=> {
2     var blob = new Blob(buffer, {type: 'video/webm'});
3     var url = window.URL.createObjectURL(blob);
4     var a = document.createElement('a');
5
6     a.href = url;
7     a.style.display = 'none';
8     a.download = 'aaa.webm';
9     a.click();
10 }
```

将录制好的视频下载下来还是比较简单的，点击 **download** 按钮后，就会调用上面的代码。在该代码中，也是先创建一个 `Blob` 对象，并根据 `Blob` 对象创建 `URL`；然后再创建一个 `<A>` 标签，设置 `A` 标签的 `href` 和 `download` 属性。这样当用户点击该标签之后，录制好的文件就下载下来了。

小结

在直播系统中，一般会包括服务端录制和客户端录制，它们各有优劣，因此好点的直播系统会同时支持客户端录制与服务端录制。

通过上面的介绍，你应该已经了解了在浏览器下如何录制音视频流了，你只需要使用浏览器中的 MediaRecorder 对象，就可以很方便地录制本地或远程音视频流。

今天我们介绍的内容只能够实现一路视频和一路音视频流的情况，但在实际的项目中还要考虑多路音视频流的情况，如有多人同时进行音视频互动的情况，在这种复杂场景下该如何将它们录制下来呢？即使录制下来又该如何进行回放呢？

所以，对于大多数采用客户端录制方案的公司，在录制时一般不是录制音视频数据，而是录制桌面加音频，这样就大大简化了客户端实现录制的复杂度了。我想你也一定知道这是为什么，没错就是因为在桌面上有其他参与人的视频，所以只要将桌面录制下来，所有的视频就一同被录制下来了哈！

思考时间

上面我已经介绍了很多录制音视频可能出现的问题。现在你思考一下，是否可以将多路音视频录制到同一个多媒体文件中呢？例如在多人的实时直播中，有三个人同时共享视频，是否可以将这三个人的视频写入到一个多媒体文件中呢（如 MP4）？这样的 MP4 在播放时会有什么問題吗？

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。

[所做 Demo 的 GitHub 链接（有需要可以点这里）](#)

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 03 | 如何使用浏览器给自己拍照呢？

下一篇 05 | 原来浏览器还能抓取桌面？

精选留言 (9)

写留言



xiao豪

2019-07-23

示例代码

```
var buffer;  
// 创建录制对象  
var mediaRecorder;...
```

展开 ▾

作者回复: 非常棒！



3



耳东

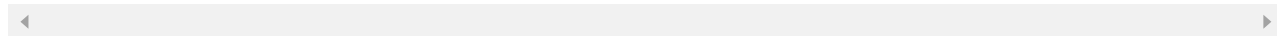
2019-07-23

React版示例代码：

<https://github.com/baayso/react-tic-tac-toe/commits/master/src/components/Video/Video.js>

展开 ▾

作者回复: 给你个大大的赞！



lvxus

2019-07-23

老师，想请问一下录制桌面具体是什么操作

展开 ▾

作者回复: 后面的课程有讲，别着急！



K

2019-07-24

视屏可以下载播放，但是无法回放。

rePlayVideo.src = window.URL.createObjectURL(blob); 提示不能赋值src为null t.js:75

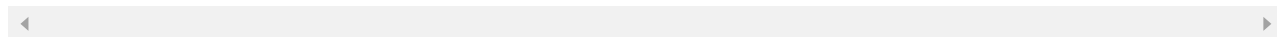
Uncaught TypeError: Cannot set property 'src' of null

控制台输出window.URL.createObjectURL(blob); 是

blob:http://localhost:63342/15434d88-60de-44fc-bd2e-b1ecd453cc5a...

展开 ▾

作者回复: 明天例子就更新了，你对照着例子看一下



恋着歌

2019-07-24

交作业 <https://codepen.io/htkar/pen/OKMbzV>

展开 ▾

作者回复: 不错！不错！



K

2019-07-24

为啥我回放的时候提示我recvideo 是null blob的size是0

作者回复: 明天demo会被放到 git 上，到时候你对照一下 demo看是不是哪里写错了



CHY

2019-07-23

在安卓端也有这种现成的录制视频的类吗？如果没有的话是不是需要自己来合流并控制时间戳呢。

作者回复: 一般都不使用移动端进行录制。如果真的要录制的话，需要使用 Native 的方法。



李跃爱学习

2019-07-23

从产品的角度来讲，应该是设置保存路径，开始录制，结束录制这么操作。但是看老师介绍的是要点击保存，就将当前blob的内容保存下来，真实场景中，要录制很长时间，内存肯定是放不下吧，所以有点困惑，录制开始后是写到磁盘了，还是保存在内存中呢？

展开 ▾

作者回复: 我看一下 handleDataAvailable 这个函数，在这个函数中我是直接将它保存到内存中了。如果你想保存成文，可以直接在这里修改代码！





剑衣清风

2019-07-23

github 里的代码可以运行么？

关于这讲，推荐看看这个 <https://cloud.tencent.com/developer/article/1366886> 里的
MediaRecorder使用示例 - 摄像头版，或者直接打开
<https://wendychengc.github.io/media-recorder-video-...>

展开 ∨

作者回复: 可以运行，你再试一下

