# H5直播系列二 MSE(Media Source Extensions)

合肥黑 关注
2 2018.10.09 15:07:25 字数 8,718 阅读 32,585

参考
w3c media-source
Media Source 系列 - 使用 Media Source Extensions 播放视频
全面进阶 H5 直播
无 Flash 时代，让直播拥抱 H5（MSE篇）
使用 MediaSource 搭建流式播放器

## 一、MSE 意义

1.粗识 HTML5 video 标签和MSE媒体源扩展
当前网页上能够搜到的HTML5和MSE相关的内容一抓一大把，本文的目的是尽量用较短的篇幅，简述浏览器为何要使用HTML5的MSE扩展。这也是在我最开始接触有关内容时的最大的疑惑。

以往用户在浏览网页内容尤其是视频内容时，需要使用像Adobe Flash或是微软的Silverlight这样的插件，播放视音频内容即使是电脑小白也知道，需要媒体播放器的支持，前面提到的插件就是起到媒体播放器的作用。但是使用插件这样的方式是很不便捷且很不安全的，一些不法分子会在这些插件上动手脚。因此W3C的最新的HTML5标准中，定义了一系列新的元素来避免使用插件，其中就包含了<video>标签这一大名鼎鼎的元素。

正是使用了<video>标签，支持HTML5的浏览器得以实现无插件就原生支持播放媒体内容，但是对媒体内容的格式有所限制。说到媒体内容，就自然地需要谈到媒体的封装格式和编码格式，这里总结一下，原视频文件通过编码来压缩文件大小，再通过封装将压缩视音频、字幕组合到一个容器内，具体内容请大家自行查阅。

我们可以把<video>标签看做拥有解封装和解码功能的浏览器自带播放器。随着视频点播、直播等视频业务的发展，视频通过流媒体传输协议（目前常用的有两种，MPEG-DASH和Apple的HLS）从服务器端分发给客户端，媒体内容进一步包含在一层传输协议中，这样<video>就无法识别了。以HLS为例，将源文件内容分散地封装到了一个个TS文件中。

仅靠<video>标签无法识别这样的TS文件，那么就引入了MSE拓展来帮助浏览器识别并处理TS文件，将其变回原来可识别的媒体容器格式，这样<video>就可以识别并播放原来的文件了。那么支持HTML5的浏览器就相当于内置了一个能够解析流协议的播放器。

比如在hls.js 源码解读【1】中，介绍的hls.js

> hls实际会先通过 ajax（loader 是可以完成自定义的）请求 m3u8文件，然后会读取到文件的分片列表，以及视频的编码格式，时长等。随后会按照顺序(非 seek)去对分片进行请求，这些也是通过 ajax 请求二进制的文件，然后借助 Media Source Extensions 将 buffer 内容进行合流，然后组成一个可播的媒体资源文件。

2.为什么国内大部分视频厂商不对PC开放HTML5?
视频源存在兼容性问题。原生的 HTML5 <video> 元素在 Windows PC 上仅支持 mp4（H.264 编码）、webm、ogg 等格式视频的播放。而由于历史遗留问题（HTML5 视频标准最终被广泛支持以前，Flash 在 Web 视频播放方面有着统治地位），视频网站的视频源和转码设置，很多都高清源都是适用于 Flash 播放的 FLV 格式，只有少量低清晰度视频是 mp4 格式，webm 和 ogg 更是听都没听说过。比如优酷只有高清和标清才有 MP4 源，超清、1080P 等，基本都是 FLV 和 HLS（M3U8）的视频源（在 Windows PC 上支持 M3U8 比支持 FLV 更复杂，我们不做过多赘述）。而腾讯视频，因为转型 MP4 比较早，视频源几乎全部都是 MP4 和 HLS，所以现在可以在 Mac OS X 上率先支持 PC Web 端的 HTML5 播放器（Safari 下 HLS、Chrome 下 MP4）。

但是 HTML5 是不是就真的没办法播放 FLV 等格式视频了呢？不是。解决方案是 MSE，Media Source Extensions，就是说，HTML5 <video> 不仅可以直接播放上面支持的 mp4、m3u8、webm、ogg 格式，还可以支持由 JS 处理过后的视频流，这样我们就可以用 JS 把一些不支持的视频流格式，转化为支持的格式（如 H.264 的 mp4）。B 站开源的 flv.js 就是这个技术的一个典型实现。B 站的 PC HTML5 播放器，就是用 MSE 技术，将 FLV 源用 JS 实时转码成 HTML5 支持的视频流编码格式（其实就一个文件头的差异（这里文件头改成容器。感谢评论区谦谦的指教，是容器的差异，容器不只是文件头）），提供给 HTML5 播放器播放。

一些人问我为什么不直接采用 MP4 格式，并表示对 FLV 格式的厌恶。这个问题一方面是历史遗留问题，由于视频网站前期完全依赖 Flash 播放而选择 FLV 格式；另一方面，如果仔细研究过 FLV/MP4 封装格式，你会发现 FLV 格式非常简洁，而 MP4 内部 box 种类繁杂，结构复杂固实而又有太多冗余数据。FLV 天生具备流式特征适合网络流传输，而 MP4 这种使用最广泛的存储格式，设计却并不一定优雅。

3.Media Source Extensions
我们已经可以在 Web 应用程序上无插件地播放视频和音频了。但是，现有架构过于简单，只能满足一次播放整个曲目的需要，无法实现拆分/合并数个缓冲文件。流媒体直到现在还在使用 Flash 进行服务，以及通过 RTMP 协议进行视频串流的 Flash 媒体服务器。

MSE 使我们可以把通常的单个媒体文件的 src 值替换成引用 MediaSource 对象（一个包含即将播放的媒体文件的准备状态等信息的容器），以及引用多个 SourceBuffer 对象（代表多个组成整个串流的不同媒体块）的元素。MSE 让我们能够根据内容获取的大小和频率，或是内存占用详情（例如什么时候缓存被回收），进行更加精准地控制。它是基于它可扩展的 API 建立自适应比特率流客户端（例如DASH 或 HLS 的客户端）的基础。

Download ---》 Response.arrayBuffer(适用fetch/xhr等异步获取流媒体数据) ---》 SourceBuffer(添加到MediaSource的buffer中) ---》 <vedio/> or <autio/>

## 二、运行DEMO

参考MDN在线DEMO bufferAll，将HTML代码及所用的文件frag_bunny.mp4下载到本地，即可运行

```
<html><head>
    <meta charset="utf-8">
  </head>
  <body>
    <video controls=""></video>
    <script>
      var video = document.querySelector('video');

      var assetURL = 'frag_bunny.mp4';
```

推荐阅读
流式播放器的实现原理
阅读 160
超全！iOS 面试题汇总（清晰易懂）
阅读 647
OpenGL ES +MediaCodec音视频采集录制录
阅读 256
常问的技术问题
阅读 176
50道前端笔面试题及参考答案
阅读 723

```
      // Need to be specific for Blink regarding codecs
      // ./mp4info frag_bunny.mp4 | grep Codec
      var mimeCodec = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"';

      if ('MediaSource' in window &&
      MediaSource.isTypeSupported(mimeCodec)) {
        var mediaSource = new MediaSource;
        //console.log(mediaSource.readyState); // closed
        video.src = URL.createObjectURL(mediaSource);
        mediaSource.addEventListener('sourceopen', sourceOpen);
      } else {
        console.error('Unsupported MIME type or codec: ', mimeCodec);
      }

      function sourceOpen (e) {
        //console.log(this.readyState); // open
        var mediaSource = e.target;
        var sourceBuffer = mediaSource.addSourceBuffer(mimeCodec);
        fetchAB(assetURL, function (buf) {
          sourceBuffer.addEventListener('updateend', function (_) {
            mediaSource.endOfStream();
            video.play();
            //console.log(mediaSource.readyState); // ended
          });
          console.log("buf",buf);
          sourceBuffer.appendBuffer(buf);
        });
      };

      function fetchAB (url, cb) {
        console.log(url);
        var xhr = new XMLHttpRequest;
        xhr.open('get', url);
        xhr.responseType = 'arraybuffer';
        xhr.onload = function () {
          cb(xhr.response);
        };
        xhr.send();
      };
    </script>
  </body></html>
```

1.参考MSE（Media Source Extensions）的一点尝试
遇到的坑是：一开始用的是自己本地随便找的一个视频文件，结果报错：Uncaught DOMException: Failed to execute 'endOfStream' on 'MediaSource': The MediaSource's readyState is not 'open'.原因是该MP4文件不是 framented mp4，不支持这种MSE的播放形式。这里也提供一个转换的工具，支持将普通MP4转为 framented mp4:Bento4 MP4 & DASH Class Library, SDK and Tools

2.参考mp4文件格式之fragment mp4
对于普通 MP4 文件，整个mp4文件的的meta数据都在文件头，所有媒体数据为整体一块。当文件比较大的时候，meta数据就比较大。这样对mp4文件的本地播放是没有问题。但对于一些视频播放网站而言，用户的播放器必须下载全meta数据才能开始播放，这就意味着用户的缓冲时间将因为mp4文件的存储结构而延长。目前一种解决方法是将大的mp4文件切成物理分离的多段，使得每段的meta都比较小，从而在一定程度上减少缓冲时间。

对于fragment mp4，mp4文件被分成多个frag分片，而原来的meta数据大大变小，且没个frag都可以单独索引、传输和播放，这样就可以解决mp4不能流式传输播放的问题。对用户体验比较好。然而目前这种格式并不被多数解码器完整支持，部分播放器加载文件时间过长，而且浏览器内嵌播放器也可能不支持播放。

3.参考WebSocket+MSE——HTML5 直播技术解析

```
       0       1c ftyp
      1c      757 moov
     773      260 moof
     9d3    21dbf mdat
   22792      220 moof
   229b2     5fb2 mdat
   28964      258 moof
   28bbc     3874 mdat
   2c430      140 moof
   2c570      738 mdat
   2cca8      260 moof
   2cf08    2c5bb mdat
   594c3      228 moof
   596eb     5bd8 mdat
   5f2c3      258 moof
   5f51b     3974 mdat
   62e8f      140 moof
   62fcf      748 mdat
   63717      260 moof
   63977    2ee9a mdat
```

fragment mp4 文件

```
       0       20 ftyp
      20        8 free
      28     c6c1 mdat
    c6e9     1e84 moov
```

non-fragment mp4 文件

我们可以看到 non-fragment mp4 的最顶层 box 类型非常少，而 fragment mp4 是由一段一段的 moof+mdat 组成的，它们已经包含了足够的 metadata 信息与数据，可以直接 seek 到这个位置开始播放。也就是说 fMP4 是一个流式的封装格式，这样更适合在网络中进行流式传输，而不需要依赖文件头的metadata。

Apple在WWDC 2016 大会上宣布会在 iOS 10、tvOS、macO S的 HLS 中支持 fMP4，可见fMP4 的前景非常的好。

值得一提的是，fMP4、CMAF、ISOBMFF 其实都是类似的东西。

把一个 non-fragment MP4 转换成 fragment MP4。可以使用 FFmpeg 的 -movflags 来转换。

对于原始文件为非 MP4 文件：`ffmpeg -i trailer_1080p.mov -c:v copy -c:a copy -movflags frag_keyframe+empty_moov bunny_fragmented.mp4`

对于原始文件已经是 MP4 文件：`ffmpeg -i non_fragmented.mp4 -movflags`

`frag_keyframe+empty_moov fragmented.mp4`

或者使用 mp4fragment：`mp4fragment input.mp4 output.mp4`

### 三、URL.createObjectURL

在H5直播系列一 Blob File FileReader URL曾经介绍过URL.createObjectURL方法。

```
//blob参数是一个File对象或者Blob对象.
var objecturl =  window.URL.createObjectURL(blob);
```

上面的代码会对二进制数据生成一个 URL，这个 URL 可以放置于任何通常可以放置 URL 的地方，比如 img 标签的 src 属性。需要注意的是，即使是同样的二进制数据，每调用一次 URL.createObjectURL 方法，就会得到一个不一样的 URL。这个 URL 的存在时间，等同于网页的存在时间，一旦网页刷新或卸载，这个 URL 就失效。(File 和 Blob 又何尝不是这样呢)除此之外，也可以手动调用 URL.revokeObjectURL 方法，使 URL 失效。

```
window.URL.revokeObjectURL(objectURL);
```

举个简单的例子。

```
var blob = new Blob(["Hello hanzichi"]);
var a = document.createElement("a");
a.href = window.URL.createObjectURL(blob);
a.download = "a.txt";
a.textContent = "Download";

document.body.appendChild(a);
```

页面上生成了一个超链接，点击它就能下载一个名为 a.txt 的文件，里面的内容是 Hello hanzichi。

### 四、使用createObjectURL将MediaSource和video标签连接起来

```
        var mediaSource = new MediaSource;
        //console.log(mediaSource.readyState); // closed
        video.src = URL.createObjectURL(mediaSource);
        mediaSource.addEventListener('sourceopen', sourceOpen);
```

这里传入createObjectURL的不是File或Blob了，而是MediaSource。MS 的实例通过 URL.createObjectURL() 创建的 url 并不会同步连接到 video.src。换句话说，URL.createObjectURL() 只是底层流 (MS) 和 video.src 的连接中间者，一旦两者连接到一起之后，该对象就没用了。那么什么时候 MS 才会和 video.src 连接到一起呢？创建实例都是同步的，但是底层流和 video.src 的连接是异步的。MS 提供了一个 sourceopen 事件给我们进行这项异步处理。一旦连接到一起之后，该 URL object 就没用了，处于内存节省的目的，可以使用 URL.revokeObjectURL(vidElement.src) 销毁指定的 URL object。

```
mediaSource.addEventListener('sourceopen', sourceOpen);
function sourceOpen(){
    URL.revokeObjectURL(vidElement.src)
}
```

MSE 支持具体的事件

- sourceopen 绑定到媒体元素后开始触发
- sourceclosed 未绑定到媒体元素后开始触发
- sourceended 所有数据接收完成后触发

对应的属性mediaSource.readyState

- open MSE 实例，已经绑定到了媒体元素上，等待接受数据或者正在接受数据
- closed MSE 实例未绑定到了媒体元素上。MS刚创建时就是该状态。
- ended MSE 实例，已经绑定到了媒体元素上，并且所有数据都已经接受到了。当endOfStream() 执行完成，会变为该状态。

### 五、设置编码类型mime 字符串

```
function sourceOpen(e) {
  URL.revokeObjectURL(videoMp4.src);
  var mime = 'video/webm; codecs="opus, vp9"';
  // e.target refers to the mediaSource instance.
  // Store it in a variable so it can be used in a closure.
  var mediaSource = e.target;
  var sourceBuffer = mediaSource.addSourceBuffer(mime);
  // Fetch and process the video.
}
```

```
var mime = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"'
```

首先，前面的 video/mp4 代表这是一段 mp4 格式封装的视频，同理也存在类似 video/webm、audio/mpeg、audio/mp4 这样的 mime 格式。一般情况下，可以通过 canPlayType 这个方法来判断浏览器是否支持当前格式。

后面的这一段 codecs="...." 比较特别，以逗号相隔，分为两段：

第一段，'avc1.42E01E'，即它用于告诉浏览器关于视频编解码的一些重要信息，诸如编码方式、分辨率、帧率、码率以及对解码器解码能力的要求。

在这个例子中，**'avc1' **代表视频采用 H.264 编码，随后是一个分隔点，之后是 3 个两位的十六进制的数，这 3 个十六进制数分别代表：

1. AVCProfileIndication(42)
2. profile_compability(E0)
3. AVCLevelIndication(1E)

第一个用于标识 H.264 的 profile，后两个用于标识视频对于解码器的要求。

对于一个 mp4 视频，可以使用 mp4file 这样的命令行工具：

```
mp4file --dump xxx.mp4
```

找到 avcC Box 后，就可以看到这三个值：

```
mp4file --dump movie.mp4
...
    type avcC (moov.trak.mdia.minf.stbl.stsd.avc1.avcC) // avc1
     configurationVersion = 1 (0x01)
     AVCProfileIndication = 66 (0x42)    // 42
     profile_compatibility = 224 (0xe0)  // E0
     AVCLevelIndication = 30 (0x1e)      // 1E
...
```

有一处要注意，后面两个值(profile_compability、AVCLevelIndication)只是浏览器用于判断自身的解码能力能否满足需求，所以不需要和视频完全对应，更高也是可以的。

下面来看 codecs 的第二段 'mp4a.40.2'，这一段信息是关于音频部分的，代表视频的音频部分采用了 AAC LC 标准：'mp4a' 代表此视频的音频部分采用 MPEG-4 压缩编码。随后是一个分隔点，和一个十六进制数(40)，这是 ObjectTypeIndication，40 对应的是 Audio ISO/IEC 14496-3 标准。(不同的值具有不同的含义，详细可以参考官方文档)

然后又是一个分隔点，和一个十进制数(2)，这是 MPEG-4 Audio Object Type，维基百科中的解释是 "MPEG-4 AAC LC Audio Object Type is based on the MPEG-2 Part 7 Low Complexity profile (LC) combined with Perceptual Noise Substitution (PNS) (defined in MPEG-4 Part 3 Subpart 4)"，具体是什么意思就不翻译了，其实就是一种 H.264 视频中常用的音频编码规范。

这一整段 codecs 都有完善的官方文档，可以参考：The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types

## 六、请求资源

sourceBuffer对象提供了一系列接口，这里用到的是 appendBuffer 方法，可以动态地向 MediaSource 中添加视频/音频片段(对于一个 MediaSource，可以同时存在多个 SourceBuffer)

如果视频很长，存在多个chunk 的话，就需要不停地向 SourceBuffer 中加入新的 chunk。这里就需要注意一个问题了，即 appendBuffer 是异步执行的，在完成前，不能 append 新的 chunk：

```
sourceBuffer.appendBuffer(buffer1)
sourceBuffer.appendBuffer(buffer2)
// Uncaught DOMException:
Failed to set the 'timestampOffset' property on 'SourceBuffer':
This SourceBuffer is still processing
an 'appendBuffer' or 'remove' operation.
```

而是应该监听 SourceBuffer 上的 updateend 事件，确定空闲后，再加入新的 chunk：

```
sourceBuffer.addEventListener('updateend', () => {
    // 这个时候才能加入新 chunk
    // 先设定新chunk加入的位置，比如第20秒处
    sourceBuffer.timestampOffset = 20
    // 然后加入
    sourceBuffer.append(newBuffer)
}
```

## 七、SourceBuffer简介

SourceBuffer 是由 mediaSource 创建，并直接和 HTMLMediaElement 接触。简单来说，它就是一个流的容器，里面提供的 append()，remove() 来进行流的操作，它可以包含一个或者多个 media segments。

```
interface SourceBuffer : EventTarget {
             attribute AppendMode          mode;
    readonly attribute boolean             updating;
    readonly attribute TimeRanges          buffered;
             attribute double              timestampOffset;
    readonly attribute AudioTrackList      audioTracks;
    readonly attribute VideoTrackList      videoTracks;
    readonly attribute TextTrackList       textTracks;
             attribute double              appendWindowStart;
             attribute unrestricted double appendWindowEnd;
             attribute EventHandler        onupdatestart;
             attribute EventHandler        onupdate;
             attribute EventHandler        onupdateend;
             attribute EventHandler        onerror;
             attribute EventHandler        onabort;
    void appendBuffer(BufferSource data);
    void abort();
    void remove(double start, unrestricted double end);
};
```

1.mode
上面说过，SB(SourceBuffer) 里面存储的是 media segments(就是你每次通过 append 添加进去的流片段)。SB.mode 有两种格式：

- segments: 乱序排放。通过 timestamps 来标识其具体播放的顺序。比如：20s的 buffer，30s 的 buffer 等。
- sequence: 按序排放。通过 appendBuffer 的顺序来决定每个 mode 添加的顺序。timestamps 根据 sequence 自动产生。

那么上面两个哪个是默认值呢？看情况，讲真，没骗你。当 media segments 天生自带 timestamps，那么 mode 就为 segments，否则为 sequence。所以，一般情况下，我们是不用管它的值。不过，你可以在后面，将 segments 设置为 sequence 这个是没毛病的。反之，将 sequence 设置为 segments 就有问题了。

```
var bufferMode = sourceBuffer.mode;
if (bufferMode == 'segments') {
  sourceBuffer.mode = 'sequence';
}
```

segments 表示 A/V 的播放时根据你视频播放流中的 pts 来决定，该模式也是最常使用的。因为音视频播放中，最重要的就是 pts 的排序。因为，pts 可以决定播放的时长和顺序，如果一旦 A/V 的 pts 错开，有可能就会造成 A/V sync drift。

sequence 则是根据空间上来进行播放的。每次通过 appendBuffer 来添加指定的 Buffer 的时候，实际上就是添加一段 A/V segment。此时，播放器会根据其添加的位置，来决定播放顺序。还需要注意，在播放的同时，你需要告诉 SB，这段 segment 有多长，也就是该段 Buffer 的实际偏移量。

而该段偏移量就是由 timestampOffset 决定的。整个过程用代码描述一下就是：

```
sb.appendBuffer(media.segment);
sb.timestampOffset += media.duration;
```

另外，如果你想手动更改 mode 也是可以的，不过需要注意几个先决条件：

- 对应的 SB.updating 必须为 false.
- 如果该 parent MS 处于 ended 状态，则会手动将 MS readyState 变为 open 的状态。

2.buffered
返回一个 timeRange 对象。用来表示当前被存储在 SB 中的 buffer。

3. updating

返回 Boolean，表示当前 SB 是否正在被更新。例如: SourceBuffer.appendBuffer(), SourceBuffer.appendStream(), SourceBuffer.remove() 调用时。

- true：当前 SB 正在处理添加或者移除的 segment
- false：当前 SB 处于空闲状态。当且仅当 updating = false 的时候，才可以对 SB 进行额外的操作。

SB 内部的 buffer 管理主要是通过 appendBuffer(BufferSource data) 和 remote() 两个方法来实现的。当然，并不是所有的 Buffer 都能随便添加给指定的 SB，这里面是需要条件和相关顺序的。

- 该 buffer，必须满足 MIME 限定的类型
- 该 buffer，必须包含 initialization segments（IS）和 media segments（MS）

下图是相关的支持 MIME：



image.png

这里需要提醒大家一点，MSE 只支持 fmp4 的格式。具体内容可以参考: 学好 MP4，让直播更给力。上面提到的 IS 和 MS 实际上就是 FMP4 中不同盒子的集合而已。

4.事件
在 SB 中，相关事件触发包括：

- updatestart：当 updating 由 false 变为 true。
- update：当 append()/remove() 方法被成功调用完成时，updating 由 true 变为 false。
- updateend: append()/remove() 已经结束
- error: 在 append() 过程中发生错误，updating 由 true 变为 false。
- abort: 当 append()/remove() 过程中，使用 abort() 方法废弃时，会触发。此时，updating 由 true 变为 false。

注意上面有两个事件比较类似：update 和 updateend。都是表示处理的结束，不同的是，update 比 updateend 先触发。

```
sourceBuffer.addEventListener('updateend', function (e) {
    // 当指定的 buffer 加载完后，就可以开始播放
      mediaSource.endOfStream();
      video.play();
    });
```

5.添加/移除 buffer
在添加 Buffer 的时候，你需要了解你所采用的 mode 是哪种类型，sequence 或者 segments。这两种是完全两种不同的添加方式。

(1)segments
这种方式是直接根据 MP4 文件中的 pts 来决定播放的位置和顺序，它的添加方式极其简单，只需要判断 updating === false，然后，直接通过 appendBuffer 添加即可。

```
if (!sb.updating) {
    let MS = this._mergeBuffer(media.tmpBuffer);

    sb.appendBuffer(MS); // ****

    media.duration += lib.duration;
    media.tmpBuffer = [];
}
```

(2)sequence
如果你是采用这种方式进行添加 Buffer 进行播放的话，那么你也就没必要了解 FMP4 格式，而是了解 MP4 格式。因为，该模式下，SB 是根据具体添加的位置来进行播放的。所以，如果你是 FMP4 的话，有可能就有点不适合了。针对 sequence 来说，每段 buffer 都必须有自己本身的指定时长，每段 buffer 不需要参考的 baseDts，即，他们直接可以毫无关联。那 sequence 具体怎么操作呢？

简单来说，在每一次添加过后，都需要根据指定 SB 上的 timestampOffset。该属性，是用来控制具体 Buffer 的播放时长和位置的。

```
if (!sb.updating) {
    let MS = this._mergeBuffer(media.tmpBuffer);

    sb.appendBuffer(MS); // ****

    sb.timestampOffset += lib.duration; // ****
    media.tmpBuffer = [];
}
```

上面两端打 * 号的就是重点内容。该方式比较容易用来直接控制 buffer 片段的添加，而不用过度关注相对 baseDTS 的值。

6.控制播放片段

如果要在 video 标签中控制指定片段的播放，一般是不可能的。因为，在加载整个视频 buffer 的时候，视频长度就已经固定的，剩下的只是你如果在 video 标签中控制播放速度和音量大小。而在 MSE 中，如何在已获得整个视频流 Buffer 的前提下，完成底层视频 Buffer 的切割和指定时间段播放呢？

这里，需要利用 SB 下的 appendWindowStart 和 appendWindowEnd 这两个属性。

他们两个属性主要是为了设置，当有视频 Buffer 添加时，只有符合在 [start,end] 之间的 media frame 才能 append，否则，无法 append。例如：

```
sourceBuffer.appendWindowStart = 2.0;
sourceBuffer.appendWindowEnd = 5.0;
```

设置添加 Buffer 的时间戳为 [2s,5s] 之间。appendWindowStart 和 appendWindowEnd 的基准单位为 s。该属性值，通常在添加 Buffer 之前设置。

6.SB 内存释放
SB 内存释放其实就和在 JS 中，将一个变量指向 null 一样的过程。

```
var a = new ArrayBuffer(1024 * 1000);
a = null; // start garbage collection
```

在 SB 中，简单的来说，就是移除指定的 time ranges' buffer。需要用到的 API 为：

```
remove(double start, unrestricted double end);
```

具体的步骤为：

- 找到具体需要移除的 segment。
- 得到其开始(start)的时间戳(以 s 为单位)
- 得到其结束(end)的时间戳(以 s 为单位)
- 此时，updating 为 true，表明正在移除
- 完成之后，出发 updateend 事件

如果，你想直接清空 Buffer 重新添加的话，可以直接利用 abort() API 来做。它的工作是清空当前 SB 中所有的 segment，使用方法也很简单，不过就是需要注意不要和 remove 操作一起执行。更保险的做法就是直接，通过 updating===false 来完成：

```
if(sb.updating===false){
    sb.abort();
}
```

这时候，abort 的主要流程为：

- 确保 MS.readyState==="open"
- 将 appendWindowStart 设置为 pts 原始值，比如，0
- 将 appendWindowEnd 设置为正无限大，即，Infinity。

abort(): 用来放弃当前 append 流的操作。不过，该方法的业务场景也比较有限。它只能用在当 SB 正在更新流的时候。即，此时通过 fetch,已经接受到新流，并且使用 appendBuffer 添加，此为开始的时间。然后到 updateend 事件触发之前，这段时间之内调用 `abort()`。有一个业务场景是，当用户移动进度条，而此时 fetch 已经获取前一次的 media segments，那么可以使用 `abort` 放弃该操作，转而请求新的 media segments。具体可以参考：abort 使用

7.appendBuffer(ArrayBuffer)
用来添加 ArrayBuffer。该 ArrayBuffer 一般是通过 fetch 的 `response.arrayBuffer();` 来获取的。在使用 addSourceBuffer 创建之前，还需要保证当前浏览器是否支持该编码格式。当然，不支持也行，顶多是当前 MS 报错，断掉当前 JS 线程。

## 八、MediaSource简介

```
[Constructor]
interface MediaSource : EventTarget {
    readonly attribute SourceBufferList    sourceBuffers;
    readonly attribute SourceBufferList    activeSourceBuffers;
    readonly attribute ReadyState          readyState;
             attribute unrestricted double duration;
             attribute EventHandler        onsourceopen;
             attribute EventHandler        onsourceended;
             attribute EventHandler        onsourceclose;
    SourceBuffer addSourceBuffer(DOMString type);
    void         removeSourceBuffer(SourceBuffer sourceBuffer);
    void         endOfStream(optional EndOfStreamError error);
    void         setLiveSeekableRange(double start, double end);
    void         clearLiveSeekableRange();
    static boolean isTypeSupported(DOMString type);
};
```

1.isTypeSupported
isTypeSupported 主要是用来检测 MS 是否支持某个特定的编码和容器盒子。例如：

```
MediaSource.isTypeSupported('video/mp4; codecs="avc1.42E01E, mp4a.40.2"')
```

这里有一份具体的 mimeType 参考列表。

2.addSourceBuffer
用来返回一个具体的视频流 SB，接受一个 mimeType 表示该流的编码格式。例如：

```
var mimeType = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"';
var sourceBuffer = mediaSource.addSourceBuffer(mimeType);
```

3.removeSourceBuffer
用来移除某个 sourceBuffer。比如当前流已经结束，那么你就没必要再保留当前 SB 来占用空间，可以直接移除。具体格式为：

```
mediaSource.removeSourceBuffer(sourceBuffer);
```

4.endOfStream()
用来表示接受的视频流的停止，注意，这里并不是断开，相当于只是下好了一部分视频，然后你

可以进行播放。此时，MS 的状态变为：ended。例如：

```
  var mediaSource = this;
  var sourceBuffer = mediaSource.addSourceBuffer(mimeCodec);
  fetchAB(assetURL, function (buf) {
    sourceBuffer.addEventListener('updateend', function (_) {
      mediaSource.endOfStream(); // 结束当前的接受
      video.play(); // 可以播放当前获得的流
    });
    sourceBuffer.appendBuffer(buf);
  });
```

5.sourceBuffers

sourceBuffers 是 MS 实例上的一个属性，它返回的是一个 SourceBufferList 的对象，里面可以获取当前 MS 上挂载的所有 SB。不过，只有当 MS 为 open 状态的时候，它才可以访问。具体使用为：

```
let SBs = mediaSource.sourceBuffers;
```

那我们怎么获取到具体的 SB 对象呢？因为，其返回值是 SourceBufferList 对象，具体格式为：

```
interface SourceBufferList : EventTarget {
    readonly attribute unsigned long length;
             attribute EventHandler  onaddsourcebuffer;
             attribute EventHandler  onremovesourcebuffer;
    getter SourceBuffer (unsigned long index);
};
```

简单来说，你可以直接通过 index 来访问具体的某个 SB:

```
let SBs = mediaSource.sourceBuffers;

let SB1 = SBs[0];
```

SBL 对象还提供了 addsourcebuffer 和 removesourcebuffer 事件，如果你想监听 SB 的变化，可以直接通过 SBL 来做。这也是为什么 MS 没有提供监听事件的一个原因。所以，删除某一个 SB 就可以通过 SBL 查找，然后，利用 remove 方法移除即可：

```
let SBs = mediaSource.sourceBuffers;

let SB1 = SBs[0];

mediaSource.removeSourceBuffer(SB1);
```

6.activeSourceBuffers

activeSourceBuffers 实际上是 sourceBuffers 的子集，返回的同样也是 SBL 对象。为什么说也是子集呢？

因为 ASBs 包含的是当前正在使用的 SB。因为前面说了，每个 SB 实际上都可以具体代表一个 track，比如，video track，audio track，text track 等等，这些都算。那怎么标识正在使用的 SB 呢？很简单，不用标识啊，因为控制哪一个 SB 正在使用是你来决定的。如果非要标识，就需要使用到 HTML 中的 video 和 audio 节点。通过

```
audioTrack = media.audioTracks[index]
videoTrack = media.videoTracks[index]

// media 为具体的 video/audio 的节点
// 返回值就是 video/audio 的底层 tracks

audioTrack = media.audioTracks.getTrackById( id )
videoTrack = media.videoTracks.getTrackById( id )

videoTrack.selected // 返回 boolean 值，标识是否正在被使用
```

上面的代码只是告诉你，正在使用 的含义是什么。对于我们实际编码的 SB 来说，并没有太多关系，了解就好。上面说了 ASBs 返回值也是一个 SBL。所以，使用方式可以直接参考 SBL 即可。

7.状态切换

要说道状态切换，我们得先知道 MS 一共有几个状态值。MS 本身状态并不复杂，一共只有三个状态值：

```
enum ReadyState {
    "closed",
    "open",
    "ended"
};
```

- closed: 当前的 MS 并没有和 HTMLMedia 元素连接
- open: MS 已经和 HTMLMedia 连接，并且等待新的数据被添加到 SB 中去。
- ended: 当调用 endOfStream 方法时会触发，并且此时依然和 HTMLMedia 元素连接。

记住，closed 和 ended 到的区别关键点在于有没有和 HTMLMedia 元素连接。

其对应的还有三个监听事件：

- sourceopen: 当状态变为 open 时触发。常常在 MS 和 HTMLMedia 绑定时触发。
- sourceended: 当状态变为 ended 时触发。
- sourceclose: 当状态变为 closed 时触发。

那哪种条件下会触发呢？

(1)sourceopen 触发

sourceopen 事件相同于是一个总领事件，只有当 sourceopen 时间触发后，后续对于 MS 来说，才是一个可操作的对象。通常来说，只有当 MS 和 video 元素成功绑定时，才会正常触发：

```
let mediaSource = new MediaSource();
vidElement.src = URL.createObjectURL(mediaSource);
```

其实这简单的来说，就是给 MS 添加 HTML media 元素。其整个过程为：

- 先延时 media 元素的 load 事件，将 delaying-the-load-event-flag 设置为 false
- 将 readyState 设置为 open。

- 触发 MS 的 sourceopen 事件

(2)sourceended 触发
sourceended 的触发条件其实很简单，只有当你调用 endOfStream 的时候，会进行相关的触发。`mediaSource.endOfStream();` 这个就没啥需要过多讲的了。

(3)sourceclose 的触发
sourceclose 是在 media 元素和 MS 断开的时候，才会触发。那这个怎么断开呢？难道直接将 media 的元素的 src 直接设置为 null 就 OK 了吗？要是这样，我就日了狗了。MS 会这么简单么？实际上并不，如果要手动触发 sourceclose 事件的话，则需要下列步骤：

- 将 readyState 设置为 closed
- 将 MS.duration 设置为 NaN
- 移除 activeSourceBuffers 上的所有 Buffer
- 触发 activeSourceBuffers 的 removesourcebuffer 事件
- 移除 sourceBuffers 上的 SourceBuffer。
- 触发 sourceBuffers 的 removesourcebuffer 事件
- 触发 MediaSource 的 sourceclose 事件

到这里，三个状态事件基本就介绍完了。不过，感觉只有 sourceopen 才是最有用的一个。

8.track 的切换
track 这个概念其实是音视频播放的轨道，它和 MS 没有太大的关系。不过，和 SB 还是有一点关系的。因为，某个一个 SB 里面可能会包含一个 track 或者说是几个 track。所以，推荐某一个 SB 最好包含一个值包含一个 track，这样，后面的 track 也方便更换。在 track 中的替换里，有三种类型，audio，video，text 轨道。
(1)video 切换
切换的含义有两种，一种是移除原有的，一种是添加新的。这里，我们需要分两部分来讲解。
(a)移除原有不需要 track

- 从 activeSourceBuffers 移除与当前 track 相关的 SB
- 触发 activeSourceBuffers 的 removesourcebuffer 事件

(b)添加指定的 track

- 从 activeSourceBuffers 添加指定的 SourceBuffer
- 触发 activeSourceBuffers 的 addsourcebuffer 事件

(2)audio 切换
audio 的切换和 video 的过程一模一样。这里我就不过多赘述了。

9.MS duration 修正机制
MS 的 duration 实际上就是 media 中播放的时延。通常来说，A/V track 实际上是两个独立的播放流，这中间必定会存在先关的差异时间。但是，media 播放机制永远会以最长的 duration 为准。这种情况对于 live stream 的播放，特别适合。因为 liveStream 是不断动态添加 buffer，但是 buffer 内部会有一定的时长的，而 MS 就需要针对这个 buffer 进行动态更新。整个更新机制为：

- 当前 MS.duration 更新为 new duration。
- 如果 new duration 比 sourceBuffers 中的最大的 pts 小，这时候就会报错。
- 让最后一个的 sample 的 end time 为所后 timeRanges 的 end time。
- 将 new duration 设置为当前 SourceBuffer 中最大的 endTime。
- 将 video/audio 的播放时长（duration）设置为最新的 new duration。

10.如何界定 track
这里先声明一下，track 和 SB 并不是一一对应的关系。他们的关系只能是 SB：track = 1: 1 or 2 or 3。即，一个 SB可能包含，一个 A/V track(1)，或者，一个 Video track，一个Audio track(2)，或者再额外加一个 text track(3)。

上面也说过，推荐将 track 和 SB 设置为一一对应的关系，应该这样比较好控制，比如，移除或者同步等操作。具体编码细节我们有空再说，这里先来说一下，SB 里面怎么决定 track 的播放。

track 最重要的特性就是 pts，duration，access point flag。track 中 最基本的单位叫做 Coded Frame，表示具体能够播放的音视频数据。它本身其实就是一些列的 media data，并且这些 media data 里面必须包含 pts，dts，sampleDuration 的相关信息。在 SB 中，有几个基本内部属性是用来标识前面两个字段的。

- last decode timestamp: 用来表示最新一个 frame 的编码时间（pts）。默认为 null 表示里面没有任何数据
- last frame duration: 表示 coded frame group 里面最新的 frame 时长。
- highest end timestamp: 相当于就是最后一个 frame 的 pts + duration
- need random access point flag: 这个就相当于是同步帧的意思。主要设置是根据音视频流 里面具体字段决定的，和前端这边编码没关系。
- track buffer ranges: 该字段表示的是 coded frame group 里面，每一帧对应存储的 pts 范围。

这里需要特别说一下 last frame duration 的概念，其实也就是 Coded Frame Duration 的内容。Coded Frame Duration 针对不同的 track 有两种不同的含义。一种是针对 video/text 的 track，一种是针对 audio 的 track:

- video/text: 其播放时长（duration）直接是根据 pts 直接的差值来决定，和你具体播放的 samplerate 没啥关系。虽然，官方也有一个计算 refsampelDuration 的公式：duration = timescale / fps，不过，由于视频的帧率是动态变化的，没什么太大的作用。
- audio: audio 的播放时长必须是严格根据采样频率来的，即，其播放时间必须和你自己定制的 timescale 以及 sampleRate 一致才行。针对于 AAC，因为其采样频率常为 44100Hz，其固定播放时长则为：duration = 1024 / sampleRate * timescale

所以，如果你在针对 unstable stream 做同步的话，一定需要注意这个坑。有时候，dts 不同步，有可能才是真正的同步。

我们再回到上面的子 title 上-- 如果界定 track。一个 SB 里面是否拥有一个或者多个 track，主要是根据里面的视频格式来决定的。打个比方，比如，你是在编码 MP4 的流文件。它里面的 track 内容，则是根据 moov box 中的 trak box 来判断的。即，如果你的 MP4 文件只包含一个，那么，里面的 track 也有只有一个。

## 九、MSE兼容性 caniuse

1.iOS Safari 不支持 Media Source Extensions 因此无法使用 flv.js

以下摘自HTML5 媒体源扩展（MSE）：把影视制作级别的视频格式带入 Web

> 要覆盖99%的用户，我们需要做一个视频流兼容设置，这样也可以让那些不支持MSE的浏

览器也能顺利播放，比如一些旧版本的浏览器，和iOS上的Safari。老的浏览器可以使用Flash播放器来提供服务，Flash播放器是可以直接播放MSE的MPEG-DASH格式内容的，如Bitdash player播放器。为了支持iOS设备，我们必须要使用Apple的HLS流媒体格式，这是苹果在HTML5中强推的另一种方式。Apple并不喜欢支持开放标准（如MSE），不过Mac OSX上的Safari还是支持MSE的。

2.以下摘自X5内核视频之问答汇总--本帖最后由 YongLing 于 2018-07-05 14:45:51 编辑

Q:X5内核支持MSE吗？
A:X5内核MSE正在支持中，预计TBS44200版本及以后，QQ浏览器8.6版本及以后支持。