



千帆直播

看看现在的我 QF.COM.CN

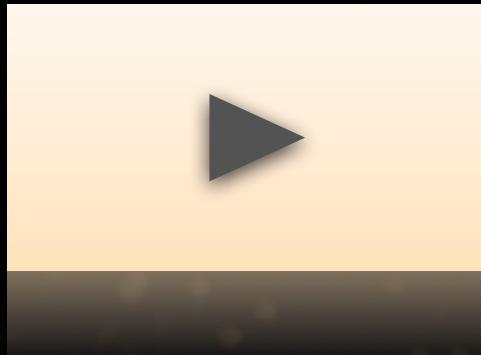
直播播放器，从Flash到HTML5

by weiko
2017-09-05



PC Play Side

- 1、现状分析
 - 2、思考、探索与实践
 - 3、实践方案的技术实现
 - 4、Q&A
- 



目前视频直播播放端常见的实现方案

Flash + RTMP

Flash + HTTP-FLV

一些现象

点按以使用 Flash →



一些现象



按住 Ctrl 键的同时点击即可运行 Adobe Flash Player



一些现象



Flash正由于它自身的性能问题、安全问题，已逐步走向衰老死亡
同时各大浏览器对其的封杀也不断升级

一些信息

2014年10月28日，W3C的HTML工作组正式发布了HTML5的正式推荐标准

2015年11月30日，Adobe在其官方博客发表申明：
“建议用户通过新的网页标准创建内容”

YouTube、Facebook、直播站点Twitch相继完成了HTML5视频播放器的过渡

2017年7月26日，Adobe宣布Flash技术将于2020年底退役

思考



是时候对 Flash say goodbye 了

思考

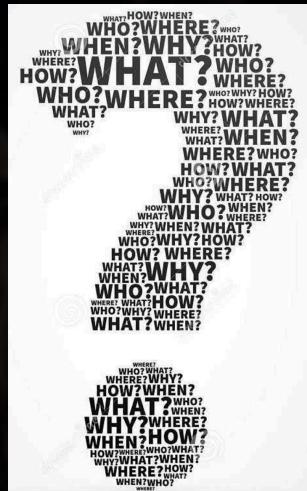
H5技术是我们探索的方向,拥抱H5

video

重点研究对象

HLS、HTTP-FLV、MSE和WebSocket

以及



HLS

HTTP Live Streaming , 简称 HLS
是由苹果提出基于HTTP的流媒体传输协议

HLS

服务器把视频流分成以 **ts** 为后缀名的小文件
生成一个**m3u8**的纯文本索引文件
客户端请求 **m3u8**文件并解释
拿到相应 **ts** 文件地址，再请求**ts**进行播放
m3u8索引实时更新
客户端再请求**m3u8**

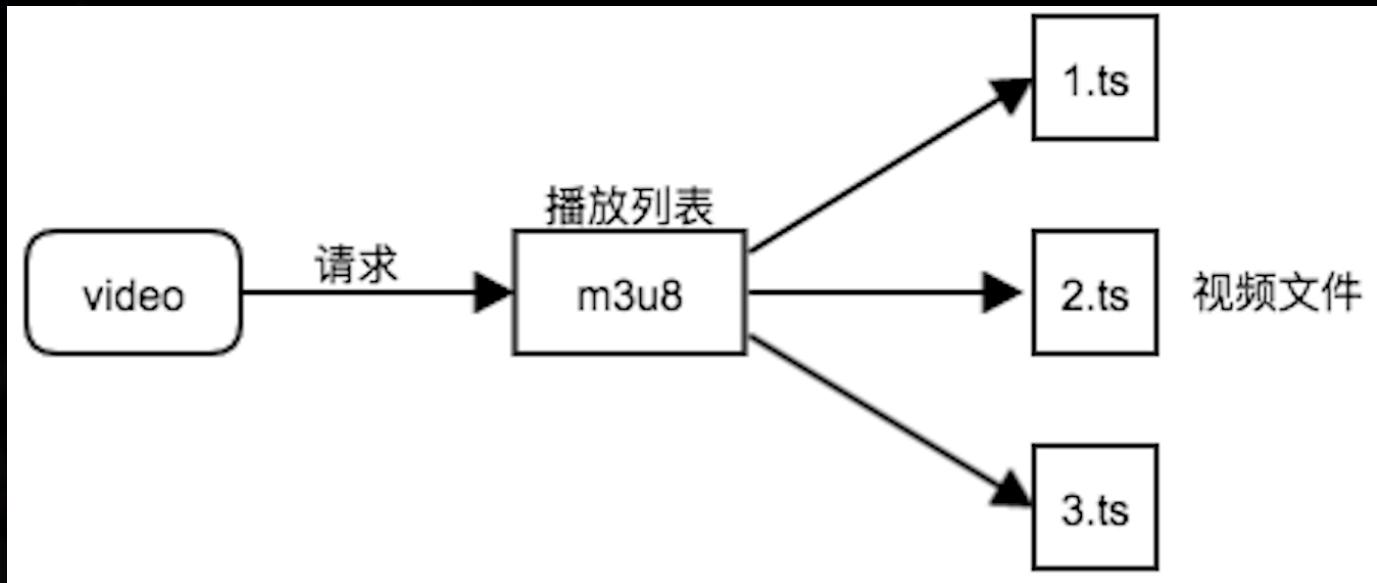
playlist.m3u8

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:NO
#EXT-X-TARGETDURATION:4
#EXT-X-MEDIA-SEQUENCE:155

#EXTINF:1.935
504071.ts
#EXTINF:3.975
504072.ts
#EXTINF:1.972
504073.ts
```

HLS

```
<div class="pc_video_wrap" style="display: none;>
  > <video id="videoPlayer" width="100%" src="https://hls-v-ngb.qf.56.com/live/5089788_1500684053747/playlist.m3u8?wsSecret=ef7b9996c160dd34d64a81a6e890c654&wsTime=5972A71C&get_url=6" playsinline preload="auto" controls>...</video>
</div>
```



欢迎 松林H54V 进入房间
欢迎 微信网友03263 进入房间

HLS优势

客户端支持简单，支持 HTTP 请求并按顺序下载媒体片段即可

HTTP 协议网络兼容性好，CDN 支持良好

Apple 的全系列产品不需要安装任何插件就可以原生支持播放 HLS，
现在，Android 也加入了对 HLS 的支持

自带多码率自适应

HLS缺点

PC 端浏览器基本不支持

延时高

HTTP-FLV

将音视频数据封装成FLV
然后通过HTTP协议传输给客户端

HTTP-FLV优点

低延迟

HTTP 协议网络兼容性好
FLV 天生就是流媒体格式

HTTP-FLV缺点

播放器问题

播放器解决方案

video + MSE

HTML5 video特点

- video H5标准，符合我们的预期；
- 各大浏览器厂商实现了高性能硬解；
- 仅支持播放 mp4/webm 格式；
- 不支持实时流；

问题

HTML5 video支持播放什么格式视频？

格式	IE	Firefox	Opera	Chrome	Safari
Ogg	No	3.5+	10.5+	5.0+	No
MPEG 4	9.0+	No	No	5.0+	3.0+
WebM	No	4.0+	10.6+	6.0+	No

HTML5 video

我们目前主要的直播流

RTMP 以及 FLV

video 不支持 怎么办？

问题

什么是MSE？

MSE

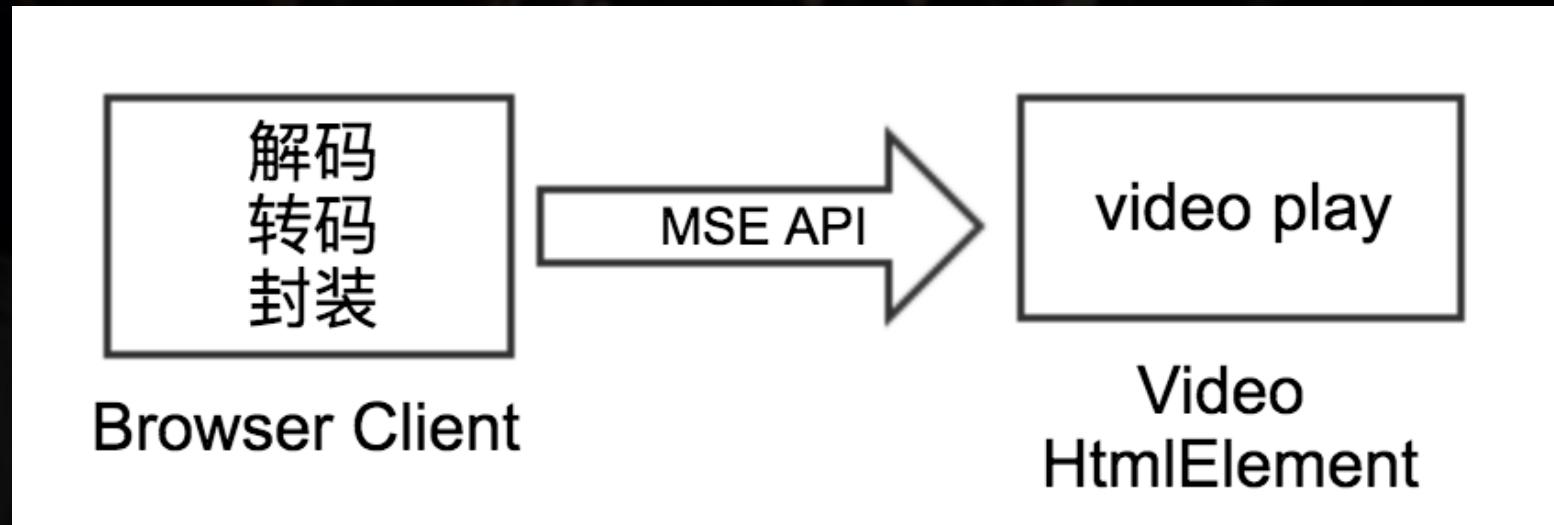
MSE : Media Source Extensions, 媒体源扩展；

主要作用是解决HTML5流的问题

如何解决 H5 流的问题？

使用 JavaScript 进行动态构建媒体流

通过 MSE API 把流推送给 HTML 媒体元素，如 video

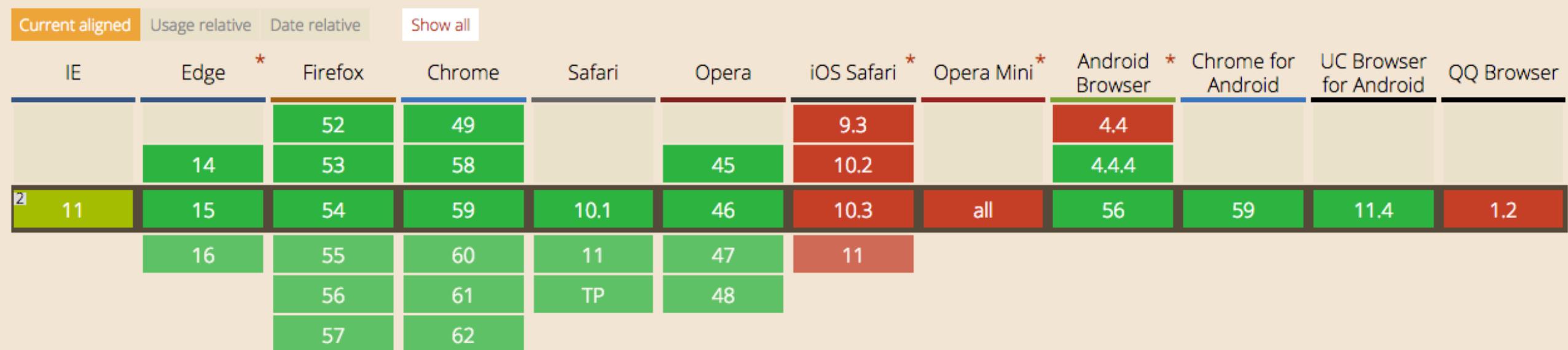


MSE浏览器兼容

Media Source Extensions - CR

API allowing media data to be accessed from HTML **video** and **audio** elements.

Global	$74.53\% + 3.32\% = 77.85\%$
unprefixed:	$74.43\% + 3.32\% = 77.75\%$
China	$72.58\% + 3.43\% = 76.02\%$
unprefixed:	$72.39\% + 3.43\% = 75.82\%$



MSE API

MediaSource

媒体对象，连接到 HTML 媒体元素

SourceBuffer

用于保存媒体数据的缓冲区

addSourceBuffer

创建一个新的SourceBuffer，并将其添加到MediaSource 的 SourceBuffers

appendBuffer

向 SourceBuffer 添加将指定的媒体片段

MSE API

```
// 创建并分配源  
var mediaSource = new MediaSource();  
var video = document.getElementById("video");  
video.src = URL.createObjectURL(mediaSource);
```

```
// 添加 MIME 类型  
var audioSourceBuffer = mediaSource.addSourceBuffer('video/mp4; codecs="mp4a.40.2"');  
var videoSourceBuffer = mediaSource.addSourceBuffer('video/mp4; codecs="avc1.42E01E"');
```

```
// 添加音视频数据到 SourceBuffer  
audioSourceBuffer.appendBuffer(segment.data.buffer);  
videoSourceBuffer.appendBuffer(segment.data.buffer);
```

如何传输视频流？

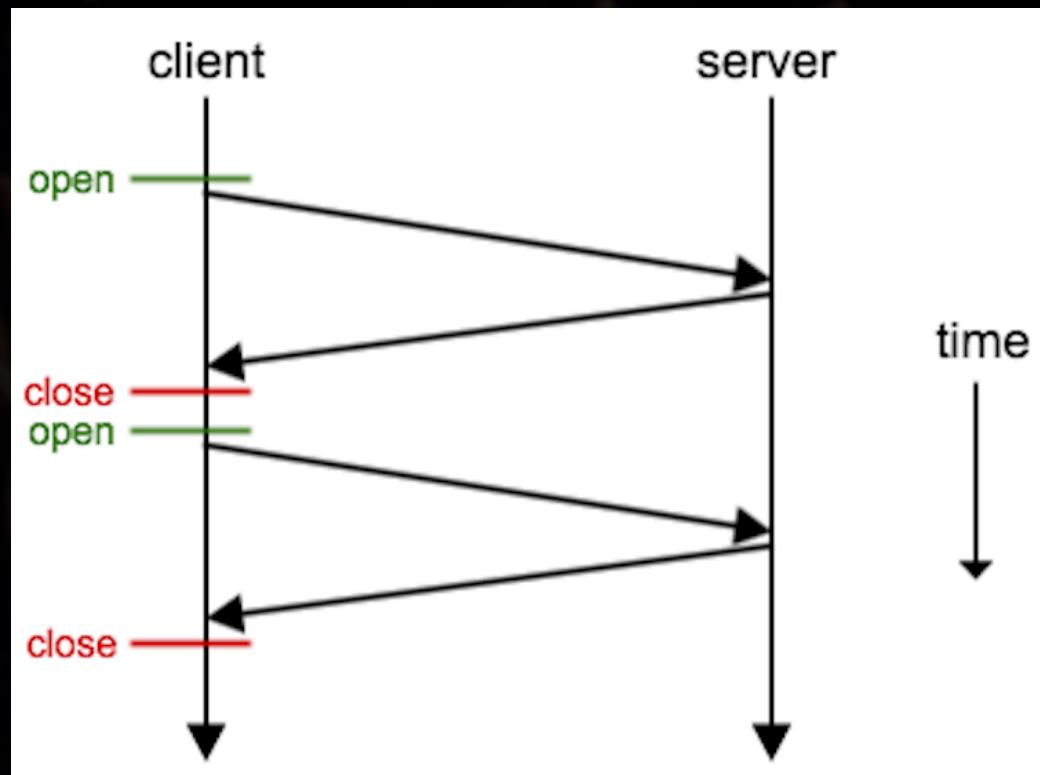
WebSocket

WebSocket特点

基于TCP连接之上的通信协议
浏览器与服务器**全双工**通信

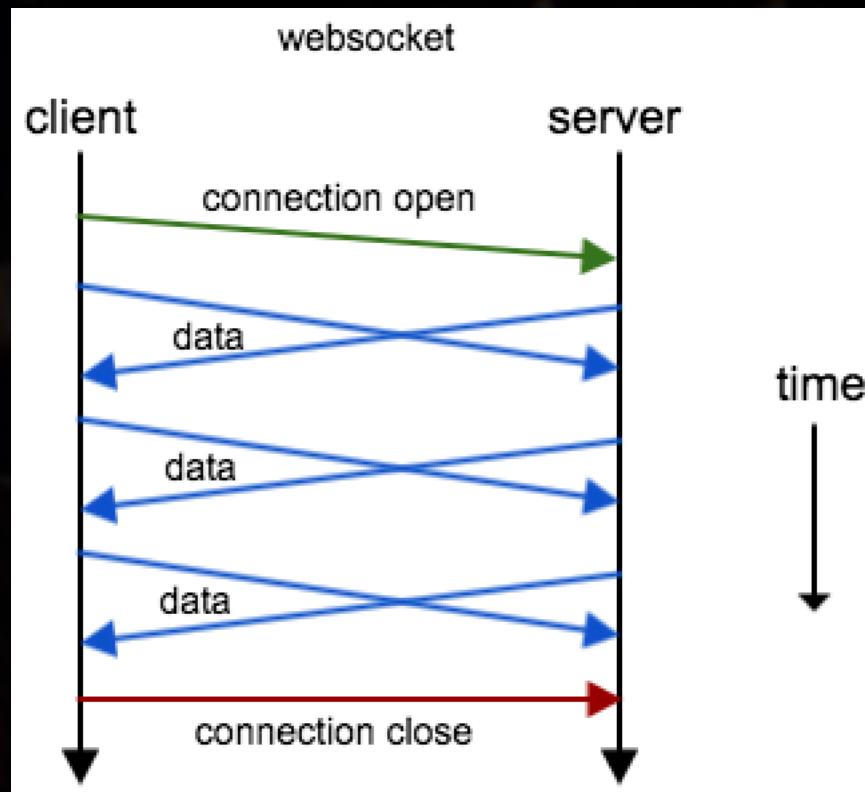
WebSocket

通常的web应用中HTTP请求/响应交互图



WebSocket

WebSocket协议是基于TCP连接之上的一种新的通信协议



WebSocket握手

HTTP/1.1引入了 Upgrade 机制；

客户端必须在请求头部中指定两个字段：

Connection: Upgrade

Upgrade: websocket

WebSocket握手

服务端同意升级，响应头如下：

HTTP/1.1 101 Switching Protocols

Connection: upgrade

Upgrade: websocket

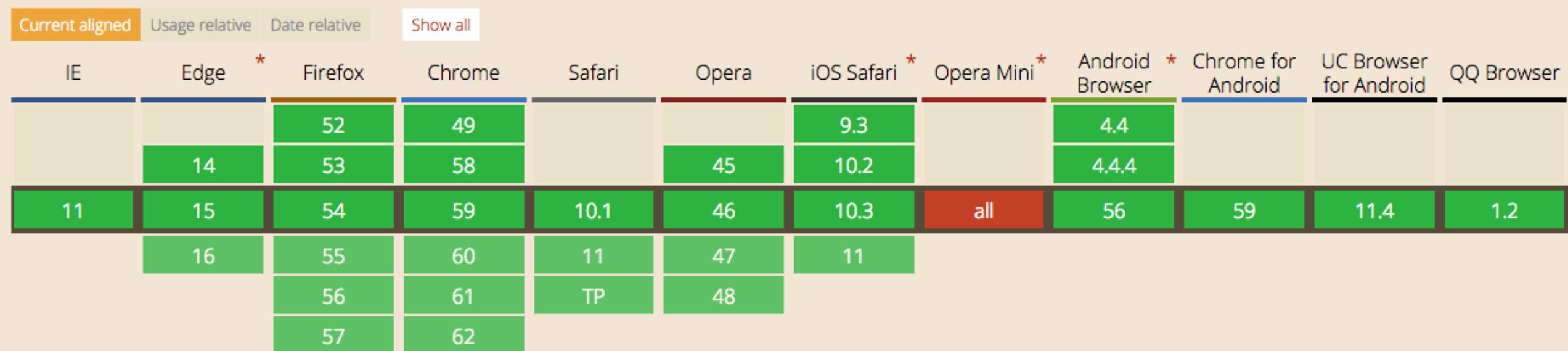
客户端与服务端的 websocket 连接握手成功，后续的数据传输直接在 TCP 上完成

WebSocket浏览器兼容

Web Sockets - LS

Bidirectional communication technology for web apps

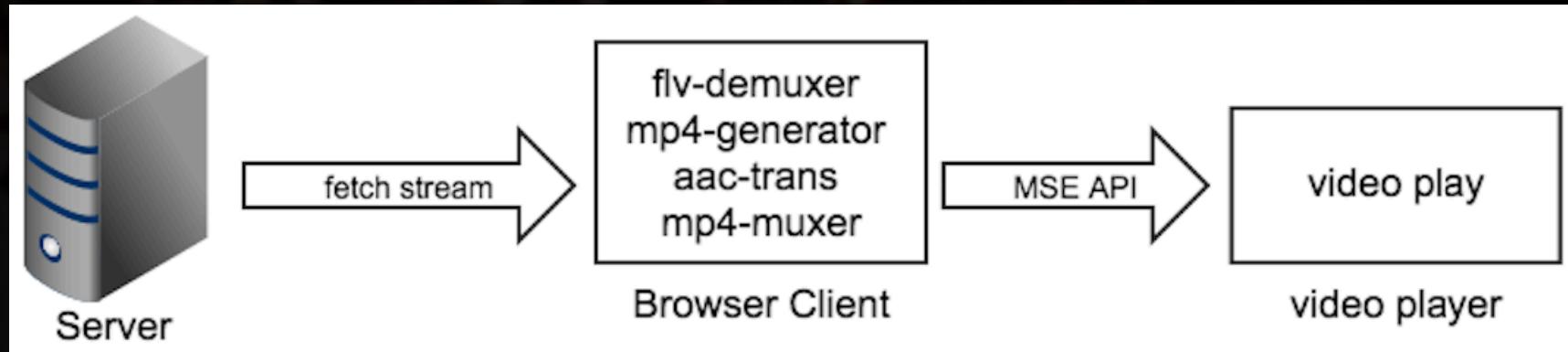
Global	$93.62\% + 0.23\% =$	93.85%
unprefixed:	$93.62\% + 0.18\% =$	93.81%
China	$88.5\% + 0.54\% =$	89.04%
unprefixed:	$88.5\% + 0.17\% =$	88.66%



WebSocket + MSE + video 配合工作

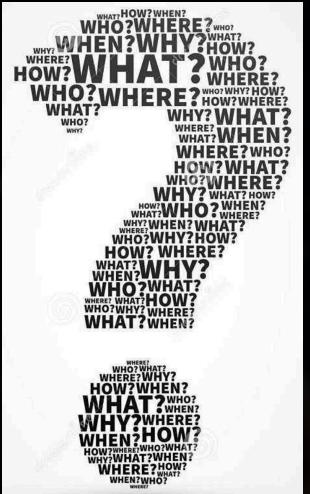
WebSocket不间断传输 FLV 视频流

客户端解码、转码、封装，使用MSE推送给 video
video 播放视频和音频



所有问题都解决了吗？

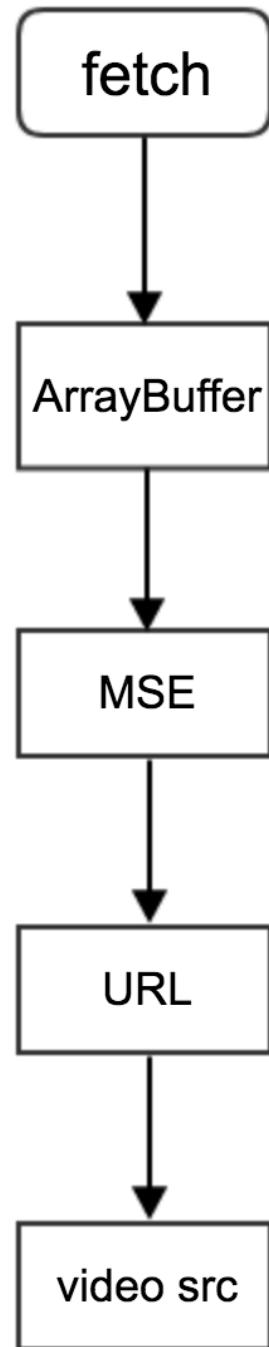
WebSocket是否可行？



= fetch

Access-Control-Allow-Origin: https://qf.56.com

Access-Control-Allow-Credentials: true



FLV 文件分析

FLV包括文件头（File Header）和文件体（File Body）两部分，其中文件体由一系列的Tag组成

每个Tag前面还包含了Previous Tag Size字段，表示前面一个Tag的大小。Tag的类型可以是视频、音频和Script

```
// 检测是否 FLV 格式
if (data[0] !== 0x46 || data[1] !== 0x4C ||
    data[2] !== 0x56 || data[3] !== 0x01) {
    return mismatch;
}
```

```
// 是否存在音频、视频数据
let hasAudio = ((data[4] & 4) >>> 2) !== 0;
let hasVideo = (data[4] & 1) !== 0;
```

FLV Header	Signature(3字节)文件标识。FLV(0x46, 0x4c, 0x66)		
	Version(1字节)版本，0x01		
	Flags(1字节)前5位保留为0，第6位表示是否有音频Tag，第7位保留，为0，第8位是否有视频Tag		
	Headersize(4字节)从 File Header 开始到 File Body 开始的字节数		
FLV Body	Previous Tag Size #0 (4字节) 表示前一个 Tag 的长度		
	Tag #1	Type(1字节) 表示 Tag 类型，音频(0x08), 视频(0x09), script data(0x12)	
FLV Body		Datasize(3字节) 表示该 Tag Data 部分大小	
		Timestamp (3字节) 表示该 Tag 的时间戳	
		Timestamp_ex (1字节) 时间戳扩展字节	
		StreamID (3字节) 表示 stream id	
Tag Data	不同类型 Tag 的 data 部分结构不相同 (Audio、Video、Script Tag Data)		
Previous Tag Size #1 Tag#1的大小 (11+Datasize)			
Tag #2			
Previous Tag Size #2			
.....			

FLV 文件分析

```
// 不断解释 tag
if (tagType !== 8 && tagType !== 9 && tagType !== 18) {
    Log.w(this.TAG, `Unsupported tag type ${tagType}, skipped`);
    offset += 11 + dataSize + 4;
    continue;
}
....
```

```
switch (tagType) {
    case 8: // Audio
        this._parseAudioData(chunk,
        break;
    case 9: // Video
        this._parseVideoData(chunk,
        break;
    case 18: // ScriptDataObject
        this._parseScriptData(chunk
        break;
}
```

FLV Header	Signature(3字节)文件标识。FLV(0x46, 0x4c, 0x66)	
	Version(1字节)版本, 0x01	
	Flags(1字节)前5位保留为0, 第6位表示是否有音频 Tag, 第7位保留, 为0, 第8位是否有视频Tag	
	Headersize(4字节)从 File Header 开始到 File Body 开始的字节数	
Previous Tag Size #0 (4字节) 表示前一个 Tag 的长度		
FLV Body	Tag #1	Type(1字节) 表示 Tag 类型, 音频(0x08), 视频(0x09), script data(0x12)
		Datasize(3字节) 表示该 Tag Data 部分大小
		Timestamp (3字节) 表示该 Tag 的时间戳
		Timestamp_ex (1字节) 时间戳扩展字节
		StreamID (3字节) 表示 stream id
	Tag Data	不同类型 Tag 的 data 部分结构不相同 (Audio、Video、Script Tag Data)
	Previous Tag Size #1 Tag#1的大小 (11+Datasize)	
Tag #2		
Previous Tag Size #2		
.....		

根据MP4格式进行封装

```
// Generate a box
static box(type) {
    let size = 8;
    let result = null;
    let datas = Array.prototype.slice.call(arguments);
    let arrayCount = datas.length;

    for (let i = 0; i < arrayCount; i++) {
        size += datas[i].byteLength;
    }

    return new Uint8Array(size);
}
```

```
// emit ftyp & moov
static generateInitSegment(m
    let ftyp = MP4.box(MP4.t
    let moov = MP4.moov(meta

        let result = new Uint8Ar
        result.set(ftyp, 0);
        result.set(moov, ftyp.by
        return result;
    }
}
```

```
// Movie header box
static mvhd(timescale, duration) {
    return MP4.box(MP4.types.mvhd, [
        0x00, 0x00, 0x00, 0x00, // 4 bytes
        0x00, 0x00, 0x00, 0x00, // 4 bytes
        0x00, 0x00, 0x00, 0x00, // 4 bytes
        (timescale >> 24) & 0xFF,
        (timescale >> 16) & 0xFF,
```

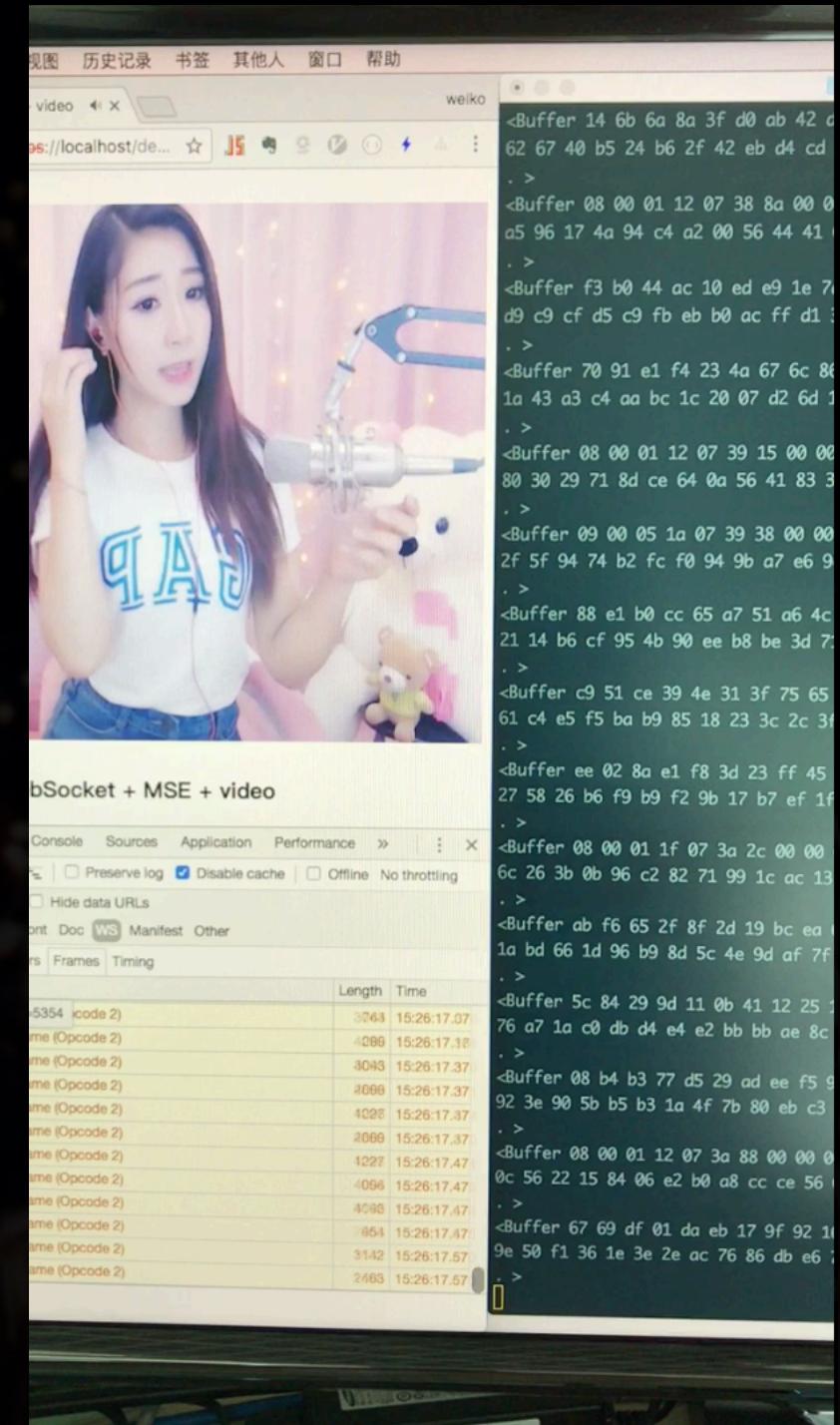
```
// Media Box
static mdia(meta) {
    return MP4.box(MP4.types.mdia, MF
}

// Media header box
static mdhd(meta) {
    let timescale = meta.timescale;
    let duration = meta.duration;
```



WebSocket + MSE + video

Demo



问题

JS 解码、转码和封装的性能问题；
以及传输接收Buffer控制问题；
WebSocket 传输流稳定性问题；
各浏览器兼容问题；
为什么传输的是flv；

为什么是 FLV

FLV 封装格式简洁，易于解释；
一边传输一边解包；
不需要整个文件、不需要通过索引分包；
基于目前成熟的直播架构方案；

Q & A



谢谢 !