

MSHN-无插件H265

2021.01.04 - hrz

Calibri 16 RGB(62,62,62)

Calibri 16 RGB(62,62,62)

Calibri 16 RGB(62,62,62)

**目录**

[目录 - 1 -](#_Toc32163)

[详细设计 - 1 -](#_Toc17746)

[附录 - 3 -](#_Toc4366)

## 详细设计

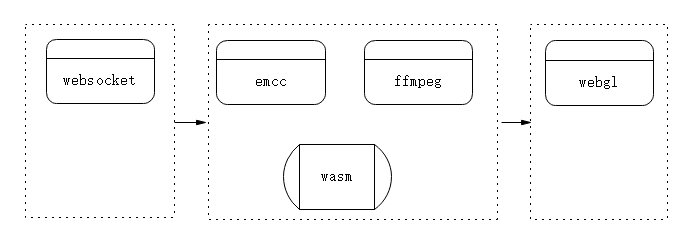
### Webassemble

WebAssembly（缩写为 wasm）是一种使用非 JavaScript 代码，并使其在浏览器中运行的方法。因版权等因素主流浏览器还不支持H.265的解码。

### 实现框架

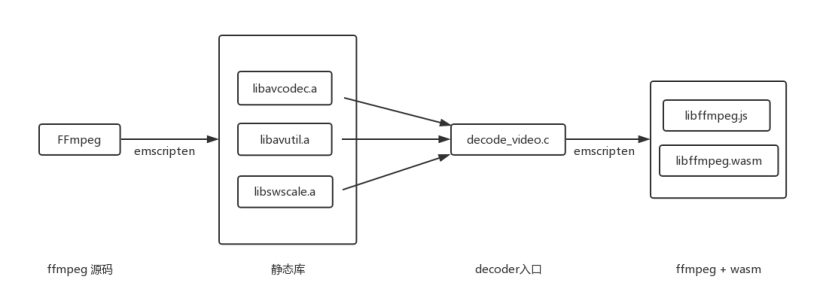
基于wasm封装ffmpeg，JS调用解码，webgl渲染来实现H265的播放。

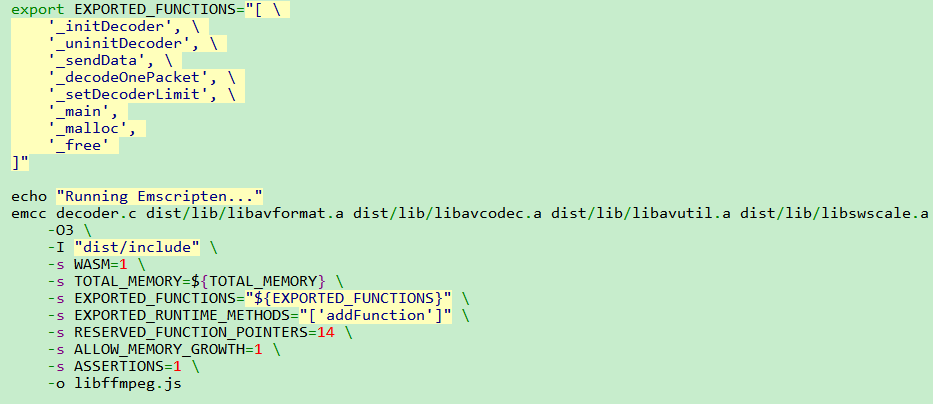
websocket + wasm + webgl



### 编译

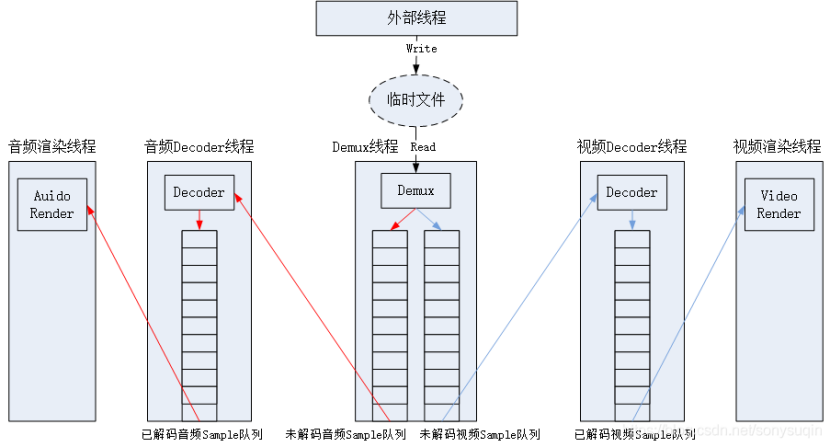
emcc最新版本只支持64位，编译过程遇到的最大问题就是关于线程支持的。去对比了IPC以及以前NVR留下来的资料，以为线程禁用是在将源码编译成wasm文件时配置的，后面发现是用emcc编译ffmpeg库的时候就要配置。

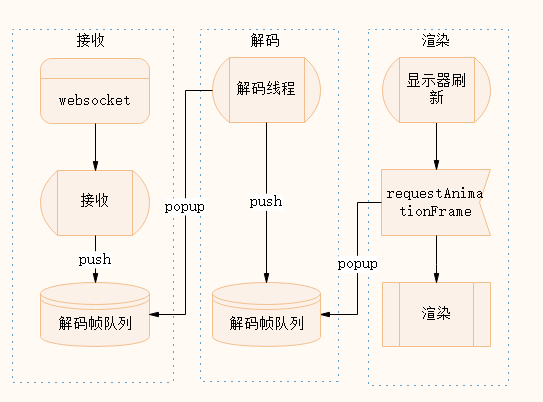




### 线程模型

理论上来说，播放器应该使用这样的线程模型，各个模块在各自线程各司其职，但是WASM目前对多线程(pthread)的支持不够好，各个浏览器的WASM多线程支持还处于试验阶段，火狐和safari则是直接报错。因此原生代码里不能编写pthread的代码，封装成工具接口，简化接口数，初始化，帧数据push，解码，策略配置，卸载，线程模型则在上层体现。





通过3条线控制接收解码渲染

**接收**：前端创建websocket连接到后端，接收帧数据，将接收到的数据添加到解码帧队列。接收独立，能保证数据传输畅通，连接更稳定。

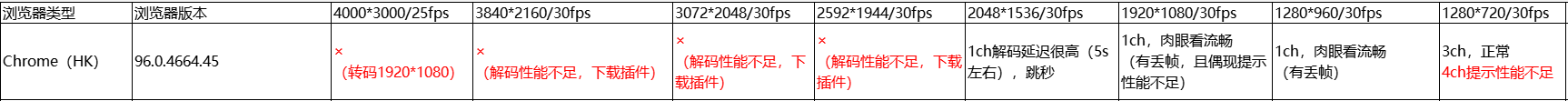
**解码**：定时器定时从解码帧队列取数据，调用ffmpeg接口解码，并将解码YUV数据添加到解码帧队列。解码独立，能根据实际情况调整解码策略，通过解码帧队列来判断当前播放的延迟情况，对每秒解码帧数做限制。

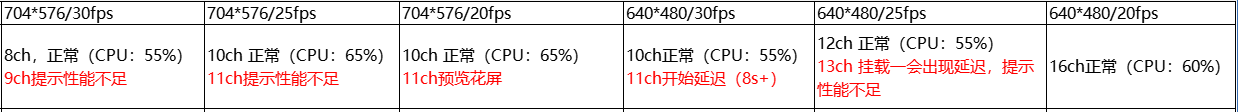
**渲染**：通过注册window.requestAnimationFrame函数，等待浏览器刷新回调，将解码帧队列数据取出来通过webgl渲染。通过raf函数，除了类似于定时器的效果执行渲染外，能在一次重绘或回流中集中完成DOM操作，另外还能在浏览器处于后台时，该函数会被暂停调用，意味着更少的的cpu，gpu和内存使用量，提升性能和电池寿命。(处于后台时，解码帧队列不会无限堆积，达到一定在帧数后会集中清理一次)。

将接收、解码、渲染从主线程拆分出来，既能保证连接的流畅，也能根据实际情况来调整解码策略。

### 解码策略

wams编译的c语言运行效率还是比较低，软解耗时且占用cpu，通过PC设备性能的提升，(比如i7处理器)，可以明显降低cpu，但是高分辨率的软解用时依旧不低，因此265更倾向于显示次码流。

友商情况：看起来是通过延迟来判断解码性能，达到一定延迟后直接提示解码性能不足。



1、主码流几乎都是直接提示解码性能不足，可能是根据解码时间设置的性能限制。

2、鱼眼主码流有概率转码显示来避免解码性能不足，只在鱼眼通道出现过，且不是必现操作。不确定转码条件。

3、次码流有遇到减帧的情况，没有提示性能不足，但是明显帧数比配置的要少一点点，猜测海康是代码有做了一些猫腻，减少帧数来实现流畅。(海康NVR预览窗口没办法配置码流信息显示，这点是通过预览效果来判断的)。

**MS NVR解码策略：**划分解码等级：

L0:不限制每秒解码帧数，帧缓存上限为80帧。

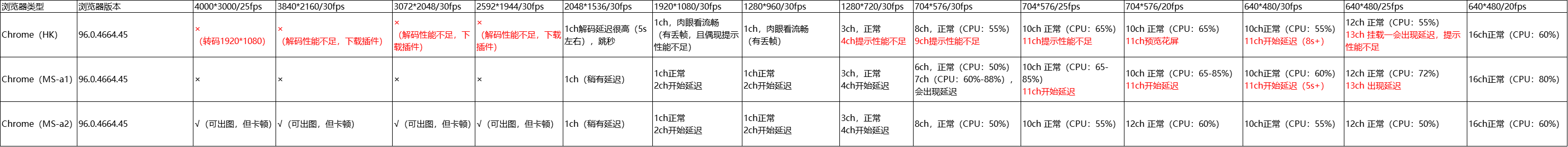
L1:每秒最多解18帧，超过则丢弃，等待下一个I帧，帧缓存上限为50帧。

L2:每秒最多解10帧，超过则丢弃，等待下一个I帧，帧缓存上限为25帧。

L3:仅解I帧，帧缓存上限为5帧

预览（包括预览界面以及小窗配置界面预览），触发到L2，弹出解码性能提示。预览会自动从L0逐级递增到L3

回放（包括回放界面以及多个录像/事件搜索回放），触发到L1，弹出解码性能提示。回放仅弹出提示，不会自动递增解码等级。



## 附录

### emcc环境搭建

需要64位ubuntu系统

#1 Get the emsdk repo

git clone https://github.com/emscripten-core/emsdk.git

#2 Enter that directory

cd emsdk

#3 Fetch the latest version of the emsdk (not needed the first time you clone)

git pull

#4 Download and install the latest SDK tools.

./emsdk install latest

#5 Make the "latest" SDK "active" for the current user. (writes .emscripten file)

./emsdk activate latest

#6 Activate PATH and other environment variables in the current terminal

source ./emsdk\_env.sh

### emcc编译ffmpeg串口命令

ffmpeg源码直接从git上导出，9.0.14用的是release/4.4版本

CPPFLAGS="-D\_POSIX\_C\_SOURCE=200112 -D\_XOPEN\_SOURCE=600" \

emconfigure ./configure --cc="emcc" --cxx="em++" --ar="emar" --prefix=/usr/hrz/aswm/demo/webfile/js/ffmpeg-wasm/dist --enable-cross-compile --target-os=none --arch=x86\_64 --cpu=generic --enable-gpl --enable-version3 --disable-avdevice --disable-swresample --disable-postproc --disable-avfilter --disable-avformat --disable-parsers --disable-everything --disable-programs --disable-ffplay --disable-ffprobe --disable-asm --disable-doc --disable-devices --disable-hwaccels --disable-bsfs --disable-debug --disable-iconv --disable-xlib --disable-zlib --disable-sdl2 --disable-bzlib --enable-small --disable-indevs --disable-outdevs --disable-encoders --disable-decoders --disable-decoder=h263 --enable-ffmpeg --enable-static --disable-shared --enable-lto --enable-decoder=pcm\_mulaw --enable-decoder=pcm\_alaw --enable-decoder=adpcm\_ima\_smjpeg --enable-decoder=aac --enable-decoder=hevc --enable-decoder=h264 --enable-avformat --disable-pthreads

make && make install

### emcc编译生成wasm文件

export TOTAL\_MEMORY=671088640

export EXPORTED\_FUNCTIONS="[ \

'\_initDecoder', \

'\_uninitDecoder', \

'\_sendData', \

'\_decodeOnePacket', \

'\_setDecoderLimit', \

'\_main',

'\_malloc',

'\_free'

]"

echo "Running Emscripten..."

emcc decoder.c dist/lib/libavformat.a dist/lib/libavcodec.a dist/lib/libavutil.a dist/lib/libswscale.a \

-O3 \

-I "dist/include" \

-s WASM=1 \

-s TOTAL\_MEMORY=${TOTAL\_MEMORY} \

-s EXPORTED\_FUNCTIONS="${EXPORTED\_FUNCTIONS}" \

-s EXPORTED\_RUNTIME\_METHODS="['addFunction']" \

-s RESERVED\_FUNCTION\_POINTERS=14 \

-s ALLOW\_MEMORY\_GROWTH=1 \

-s ASSERTIONS=1 \

-o libffmpeg.js