
2020
北京

多媒体开启新视界
MULTIMEDIA BRIDGE
TO A WORLD OF VISION

# 在Web上全速播放HEVC 1080P 60帧直播流

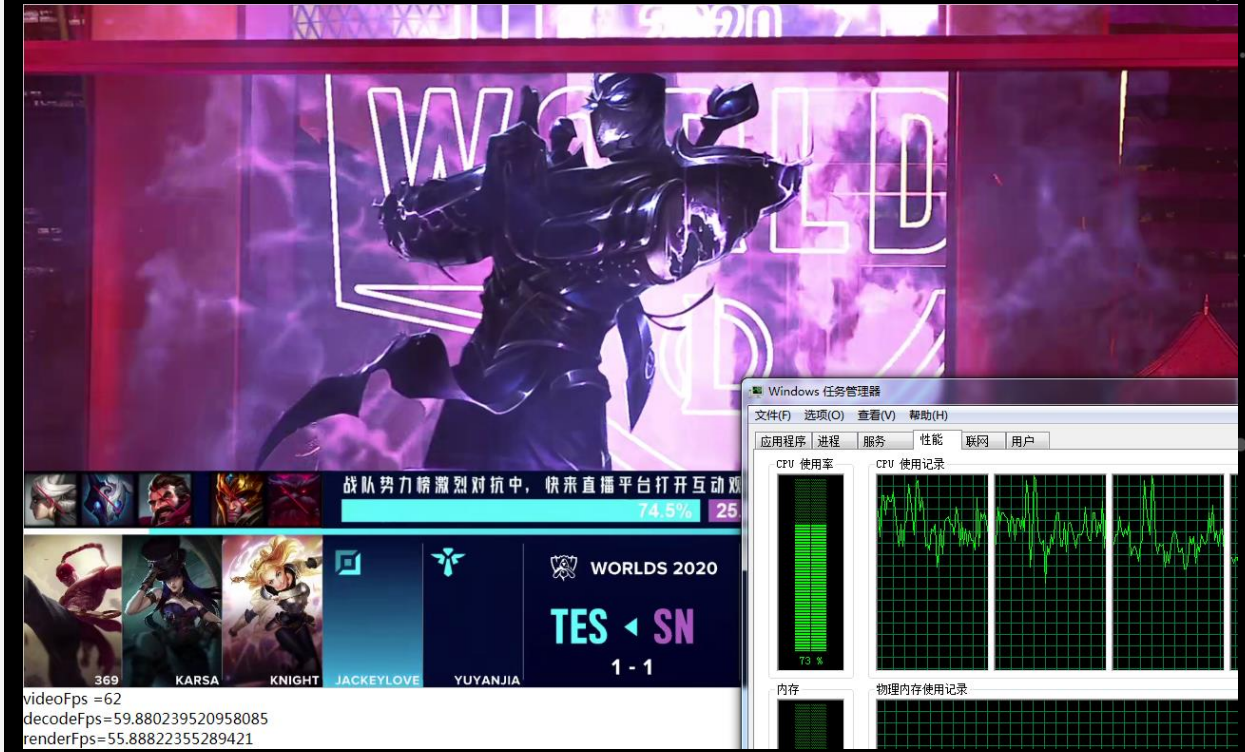
斗鱼直播 - 王兴伟

# 目录

CONTENTS

# 解码能力

- 测试机型：
  - I5 4460@3.2G 集显
- 测试视频：
  - S10测试流
  - HEVC 1080P@60帧



# 解码能力

- 测试机型：
  - I5 4460@3.2G 集显
- 测试视频：
  - 蓝光4M测试流
  - HEVC 1080P@25帧

# 目录

CONTENTS

# WebAssembly

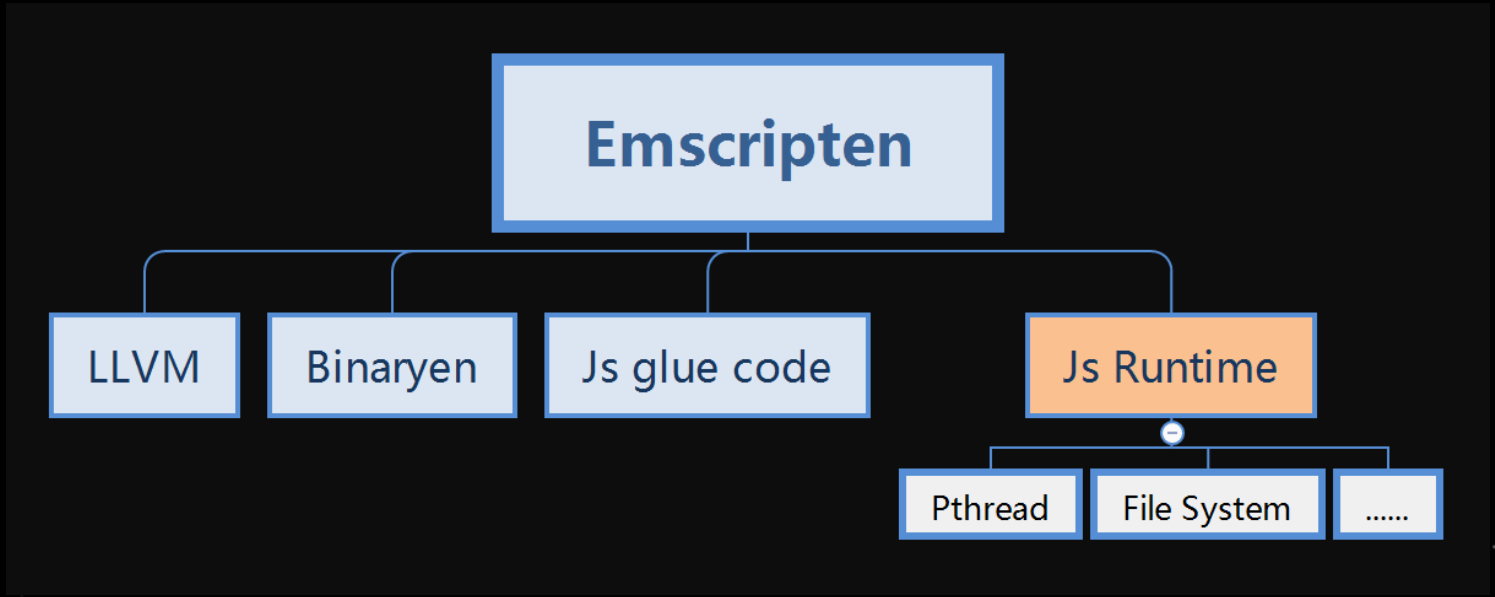

2020
北京

```
int add(int a, int b)
{
    return a + b;
}
```

```
(module
  (type $t0 (func (param i32 i32) (result i32)))
  (func $add (type $t0) (param $p0 i32) (param $p1 i32) (result i32)
    local.get $p0
    local.get $p1
    i32.add)
  (export "add" (func $add)))
```

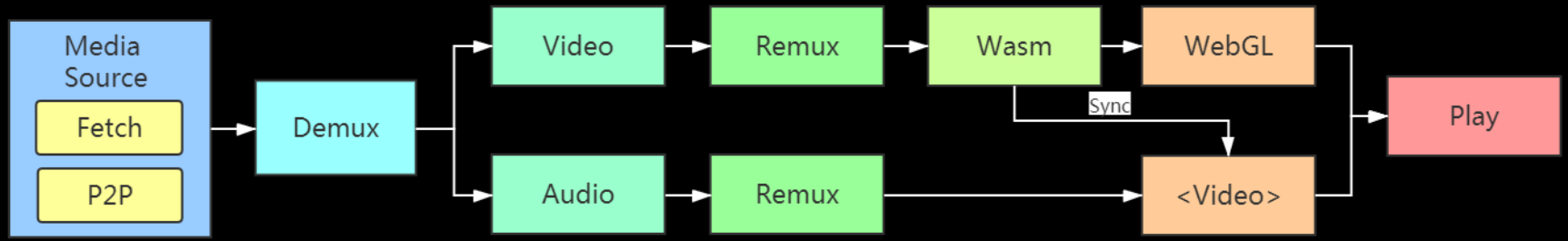
# Emscripten



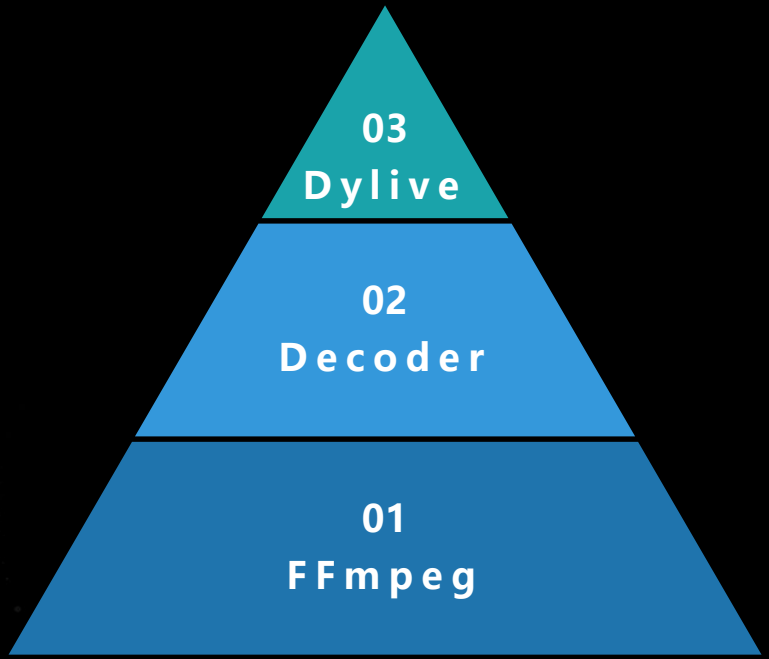



大型C/C++项目移植到Web的**重要工具**

# 整体架构


Media Source
Fetch
P2P
Demux
Video
Audio
Remux
Remux
Wasm
WebGL
Sync
<Video>
Play

# 整体架构




- Typescript
  自研H5播放器
- C + Emscripten
  自研解码模块
- C + Emscripten
  定制FFmpeg

# 定制FFMpeg

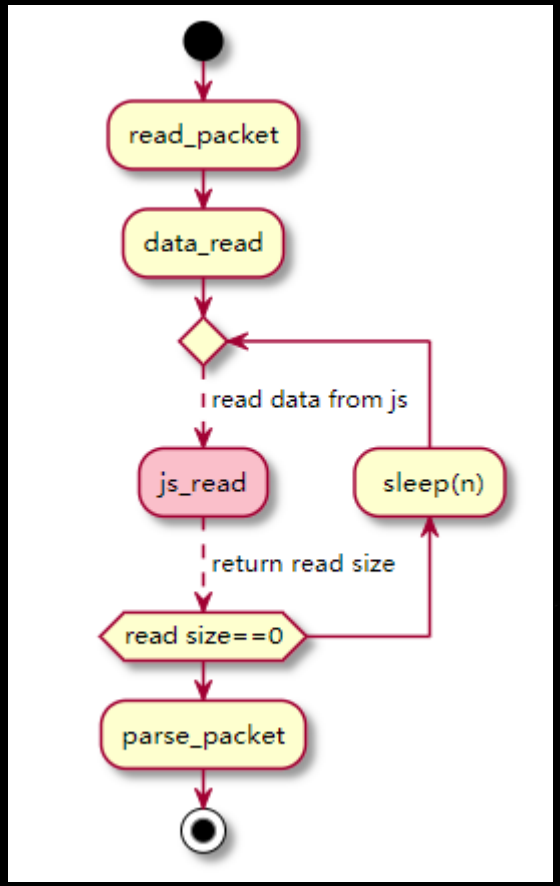
# 自定义protocol


2020
北京

- protocol是FFmpeg获取媒体数据的方式
- web应用，数据来自Js（Network or **SDK**）

# 自定义protocol的实现

```
const URLProtocol ff_custom_protocol = {
  .name = "douyu_custom",
  .url_open = data_open,
  .url_close = data_close,
  .url_read = data_read,
  .priv_data_size = sizeof(DataContext)
};
```



# sleep的问题


2020
北京

```
function _usleep(useconds) {
  // int usleep(useconds_t useconds);
  // http://pubs.opengroup.org/onlinepubs/000095399/functions/usleep.html
  // We're single-threaded, so use a busy loop. Super-ugly.
  var start = _emscripten_get_now();
  while (_emscripten_get_now() - start < useconds / 1000) {
    // Do nothing.
  }
}
```

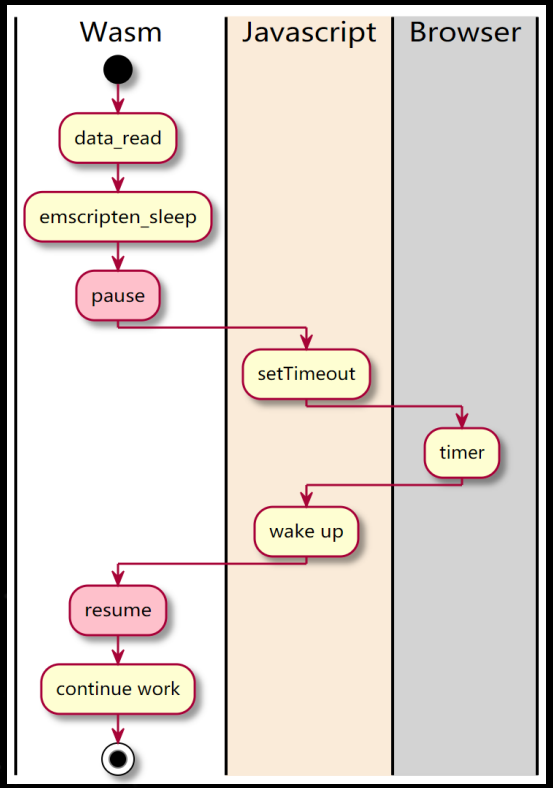
使用循环模拟sleep，**cpu占用高，线程无响应**

# Asyncify


2020
北京

- Asyncify让你在C代码中，可以**同步调用异步的JS代码**
- 使用 **emscripten_sleep** 代替 sleep

# Asyncify原理




- wasm体积会增大
- 会降低运行性能

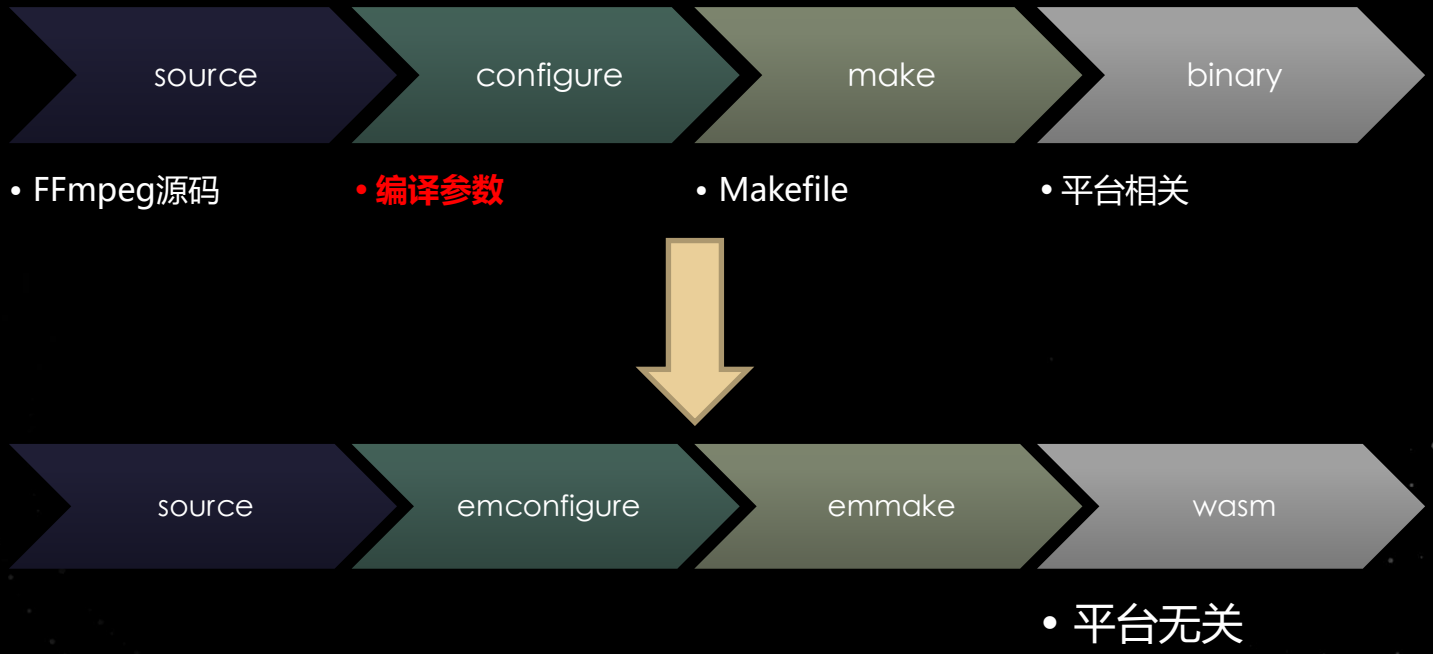
# Asyncify使用及优化



- 编译时使用参数 ” -s ASYNCIFY=1 “ 开启功能
- 设置 ASYNCIFY_ONLY 白名单，白名单中要**包括所有调用路径上的函数名**

```
-s ASYNCIFY=1
-s ASYNCIFY_ONLY="[ ..., 'read_packet', 'data_read'  ]"
```

# FFmpeg wasm编译



# FFmpeg编译参数配置


2020
北京

```
// wasm编译
--nm=llvm-nm
--strip=llvm-strip
--cc=emcc
--cxx=em++
--ar=emar
--ranlib=emranlib
--objcc=emcc
--dep-cc=emcc
--enable-cross-compile
--disable-asm
--disable-runtime-cpudetect
--disable-autodetect
```

```
// 裁剪
--disable-everything
--disable-programs
--disable-doc
--disable-avdevice
--disable-swresample
--disable-avfilter
--disable-swscale
--disable-sdl2
--disable-network
--disable-pixelutils
--disable-hwaccels
--enable-protocol=custom
--enable-demuxer=mov
--enable-parser=hevc
--enable-decoder=hevc
```

# Decoder

# 日志输出

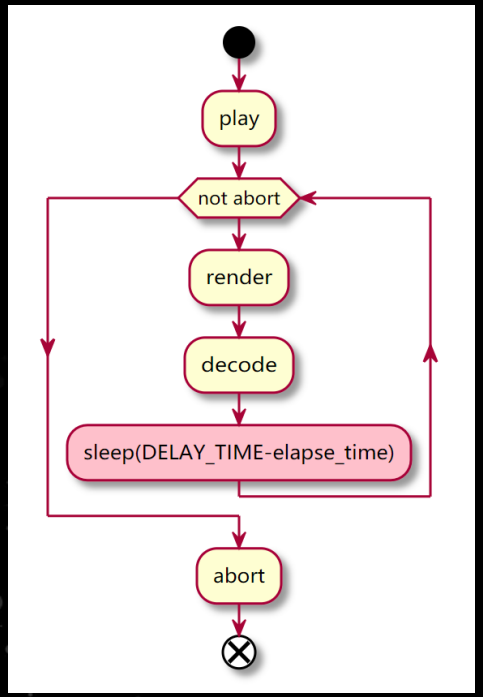



```
void log_callback(void* avcl, int level, const char* fmt, va_list arg)
{
  ...
  if (level > AV_LOG_WARNING)
      emscripten_console_log(buf);
  else
      emscripten_console_warn(buf);
}

void log_init(int level)
{
  log_level = level;
  av_log_set_level(level);
#if __EMSCRIPTEN__
  av_log_set_callback(log_callback);
#endif
}
```

- printf调用链接较长，使用**emscripten_console_log/warn代替**
- emscripten_console_warn可以看到调用堆栈，**包括wasm堆栈**，对于排查问题很有帮助

# 解码流程




- 每次循环，需要sleep，**给予时间响应其他任务**
- 可根据解码耗时**动态调整sleep时长**

# 控制内存



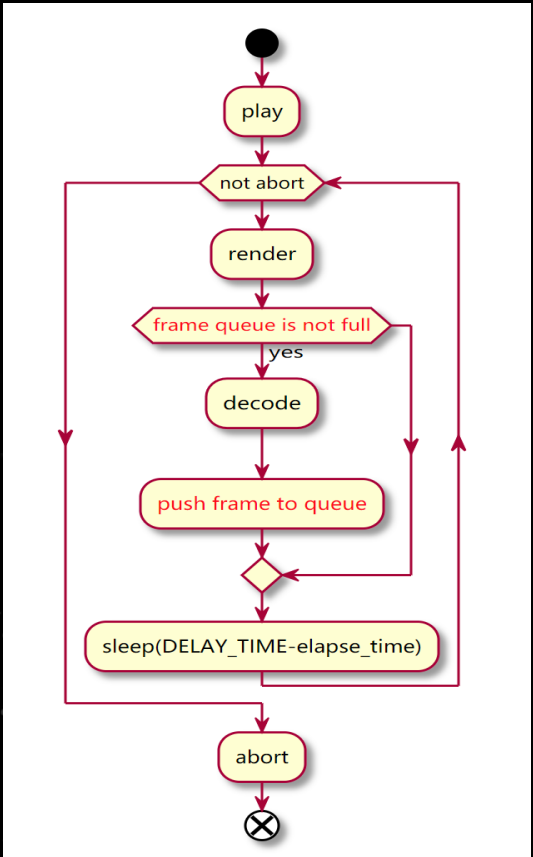


1920 x 1080分辨率的视频一帧占多少内存?

- RGBA = 1920 x 1080 x 4Byte = **8100K** （Canvas 唯一支持格式）
- YUV420P = 1920 x 1080 x 1.5Byte = **3037.5K**

对解码需要精细控制，否则容易**撑爆内存**

# 帧缓存




使用**帧队列**控制已解码帧的最大值，

同时保留一定的抗抖动能力

# 音视频同步



- 视频同步到音频
  - 音频数据较稳定
  - 视频比音频更好调整
- 以音频时间为基准
  - 视频 - 音频 > 阈值，慢放一定倍数等待
  - 视频 - 音频 < 阈值，快放一定倍数追赶

# 问题排查



- 先在**native环境调试**，保证代码的正确性
- wasm编译采用**-g3选项**，拥有尽量多的调试信息
- 关键位置输出日志，**必要时使用warn**以观察调用堆栈

# Dylive

# Javascript接口实现

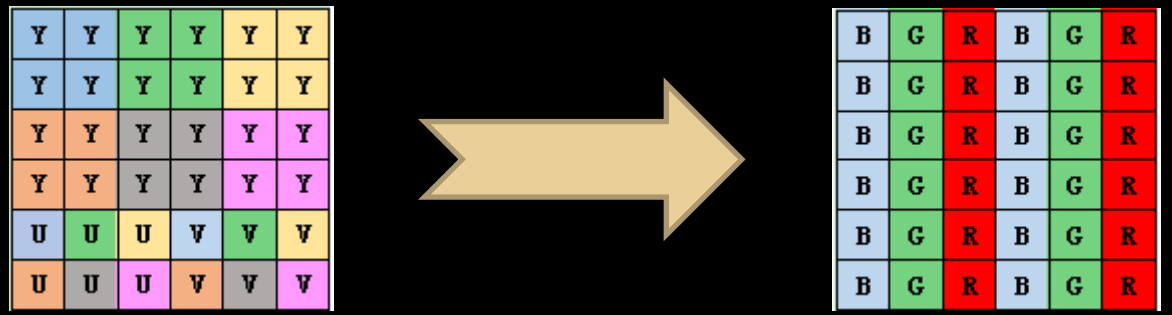


- **outer_read**
- outer_close
- **frame_out**
- get_play_time
- play_exit

```
outer_read(bufPtr, size) {
  if (this._videoBytes.byteAvailable > 0) {
    const realSize = Math.min(this._videoBytes.byteAvailable, size);
    const u8a = new Uint8Array(this._videoBytes.read(realSize));
    this._wasm.HEAPU8.set(u8a, bufPtr);
    return realSize;
  }
  ...
  return 0;
}
```

```
frame_out(width, height, pts, stride_y, stride_u, stride_v, y, u, v) {
  const ySize = stride_y * height;
  const yBuffer = this._wasm.HEAPU8.buffer.slice(y, y + ySize);
  ...
  this.render(
    {
      name: "frame",
      params: {
        width, height, pts,
        stride_y, stride_u, stride_v,
        y: yBuffer, u: uBuffer, v: vBuffer
      }
    }
  );
}
```

# 颜色转换



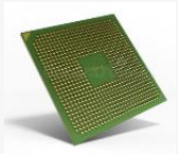

YUV转换到RGB**非常消耗性能**

# 使用WebGL shader加速



```
// Quickie YUV conversion
// https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.2020_conversion
// multiplied by 256 for integer-friendliness
multCrR  = (409 * colorCr | 0) - 57088 | 0;
multCbCrG = (100 * colorCb | 0) + (208 * colorCr | 0) - 34816 | 0;
multCbB  = (516 * colorCb | 0) - 70912 | 0;

multY = 298 * bytesY[YPtr++] | 0;
output[outPtr  ] = (multY + multCrR) >> 8;
output[outPtr + 1] = (multY - multCbCrG) >> 8;
output[outPtr + 2] = (multY + multCbB) >> 8;
outPtr += 4;
```



```
precision lowp float;

uniform sampler2D uTextureY;
uniform sampler2D uTextureCb;
uniform sampler2D uTextureCr;
varying vec2 vLumaPosition;
varying vec2 vChromaPosition;
void main() {
   // Y, Cb, and Cr planes are uploaded as LUMINANCE textures.
   float fY = texture2D(uTextureY, vLumaPosition).x;
   float fCb = texture2D(uTextureCb, vChromaPosition).x;
   float fCr = texture2D(uTextureCr, vChromaPosition).x;

   // Premultipy the Y...
   float fYmul = fY * 1.1643828125;

   // And convert that to RGB!
   gl_FragColor = vec4(
     fYmul + 1.59602734375 * fCr - 0.87078515625,
     fYmul - 0.39176171875 * fCb - 0.81296875 * fCr + 0.52959375,
     fYmul + 2.017234375   * fCb - 1.081390625,
     1
   );
}
```

# 浏览器侧优化


2020
北京

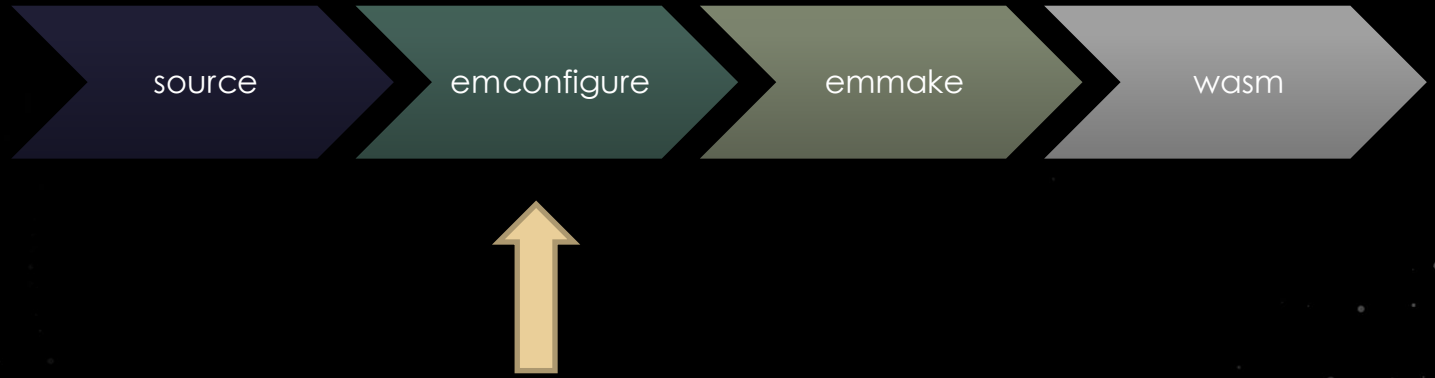
- 放到WebWorker中运行
- 使用RAF优化
- 多Tab优化
- 解码性能监测，即时降级

# 多线程

# 编译多线程wasm



- 使用”**-s USE_PTHREADS=1**“参数编译所有c文件



# 使用多线程



- FFmpeg解码开启多线程

```
AVDictionary* opt = NULL;
av_dict_set(&opt, "threads", "4", 0); // 开启多线程解码
ret = avcodec_open2(codec_ctx, codec, &opt);
```

- 使用多线程参数编译

```
-s USE_PTHREADS=1          // 开启多线程
-s PTHREAD_POOL_SIZE=4     // 预初始化4个线程
```

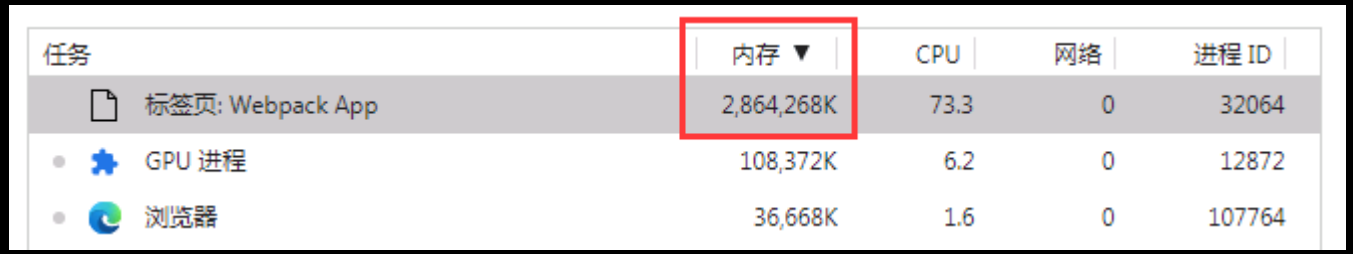
# 多线程浏览器兼容


2020
北京

- 浏览器需要支持 SharedArrayBuffer 以及 WebAssembly threads support

```
export function isPthreadsSupported() {
  return !!(
    isWasmSupported() &&
    self.SharedArrayBuffer &&
    self.Atomics &&
    //@ts-ignore
    new WebAssembly.Memory({ initial: 1, maximum: 1, shared: true })
      .buffer instanceof SharedArrayBuffer
  );
}
```

# 内存泄漏



| 任务 | 内存 ▼ | CPU | 网络 | 进程 ID |
| --- | --- | --- | --- | --- |
| 标签页: Webpack App | 2,864,268K | 73.3 | 0 | 32064 |
| GPU 进程 | 108,372K | 6.2 | 0 | 12872 |
| 浏览器 | 36,668K | 1.6 | 0 | 107764 |

```
frame_out(width, height, pts, stride_y, stride_u, stride_v, y, u, v) {
  const ySize = stride_y * height;
  const yBuffer = this._wasm.HEAPU8.buffer.slice(y, y + ySize);
  ...
```

- **不要对SharedArrayBuffer做slice操作**

# 内存泄漏解决办法

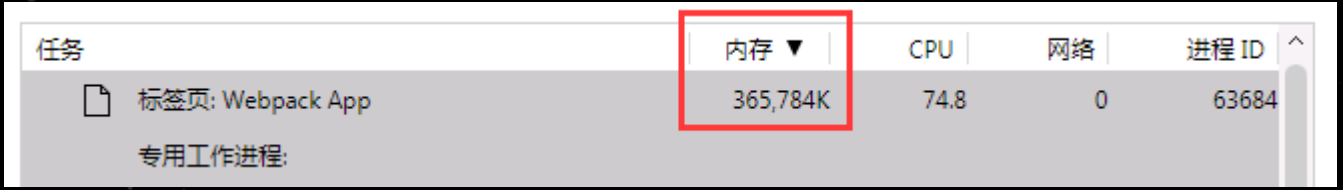


- 转换为ArrayBuffer

```
const ySize = stride_y * height;
const yBuffer = new Uint8Array(this._wasm.HEAPU8.subarray(y, y + ySize)).buffer;
```

- **使用固定内存**

```
const ySize = stride_y * height;
if (!this.yBuffer || this.yBuffer.byteLength !== ySize) {
  this.yBuffer = new SharedArrayBuffer(ySize);
}
new Uint8Array(this.yBuffer).set(this._wasm.HEAPU8.subarray(y, y + ySize), 0);
```



# 目录

CONTENTS

# 总结

- 自定义protocol
- Asyncify技术解决sleep问题
- FFmpeg裁剪及wasm编译
- 使用多线程解码
- 使用帧缓存控制内存占用
- 音视频同步
- 解决SharedArrayBuffer内存泄漏问题
- 使用WebGL渲染
- 使用Web端特有技术优化

# 目录

CONTENTS

# 意义和未来

- 不止是为了播HEVC，而是让Web端拥有FFmpeg强大的媒体处理能力
- 探索WebAssembly的调优之道，让Web端的高性能计算成为可能
- 音视频处理、云剪辑、AI、加密……

# Thank you