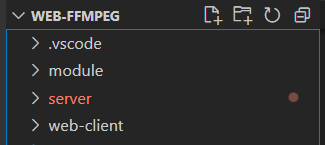
Server端功能介绍:

提供websocket server服务，当网页端通过websocekt协议连接进来后，就开启线程，不断的从h264文件中，提取出h264 nalu,然后通过 websocket 传递给网页前端，网页前端调用ffmpeg.js进行解码，得到YUV数据后，进行渲染显示.

目录结构如下:



module

web-client和server依赖的第三方模块，比如ffmpeg,sdl,libwebsocket等.

FFmpeg.

在我们的server端，我们主要依靠ffmpeg来从h264中读取出h264 nalu.

官网地址: <https://ffmpeg.org/download.html>

git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg

./configure --prefix=$PWD/../server/ffmpeg --disable-everything --disable-yasm --enable-muxer=h264

make

make install

SDL

官网: http://www.libsdl.org/

介绍:SDL是一个跨平台的库，实现了对windows/linux跨平台的支持，比如:线程,锁,信号量等.

./configure --prefix=$PWD/../server/sdl/

make

make install

libwebsockets

官网: <https://libwebsockets.org/>

介绍: libwebsockets是一个基于C的websocekt开源库,基于它我们制作服务端.

git clone https://libwebsockets.org/repo/libwebsockets

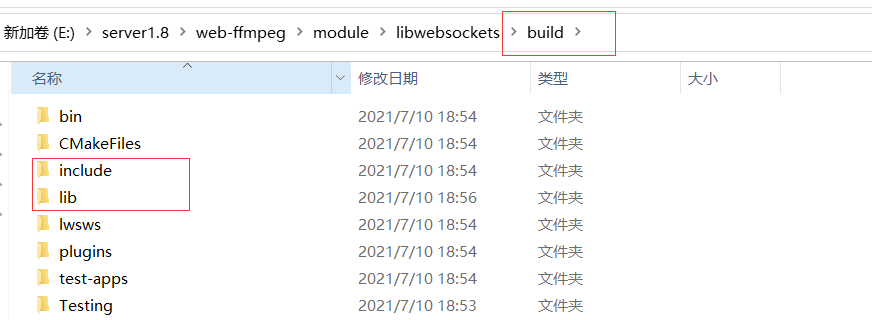
mkdir build

cd build

cmake ..

make

在我们的build目录下，include/lib 目录，可以得到头文件和静态库.



Server:

简介: Server端，我们要实现的功能是当用户连接上来后，我们便从文件中读取一帧数据，然后不断地通过websocket协议传递给前端.

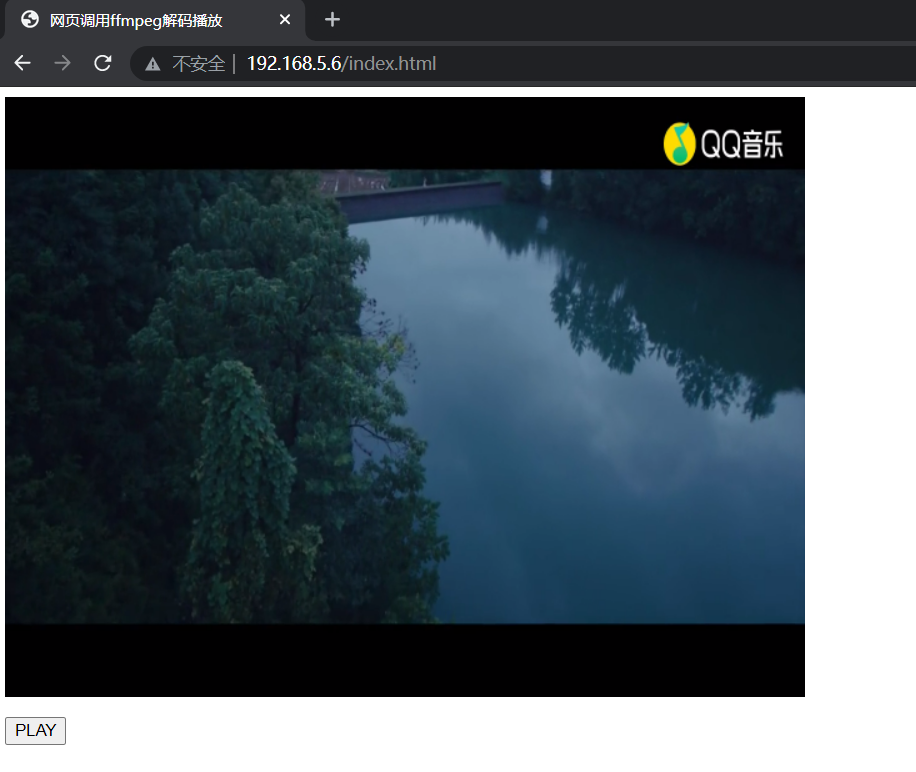
运用技术:连接池.

Github:

Client

c/c++层定义的接口，JS端如何调用

效果如下:



在js中，导入ffmpeg.js

</script><script async src=ffmpeg.js></script>

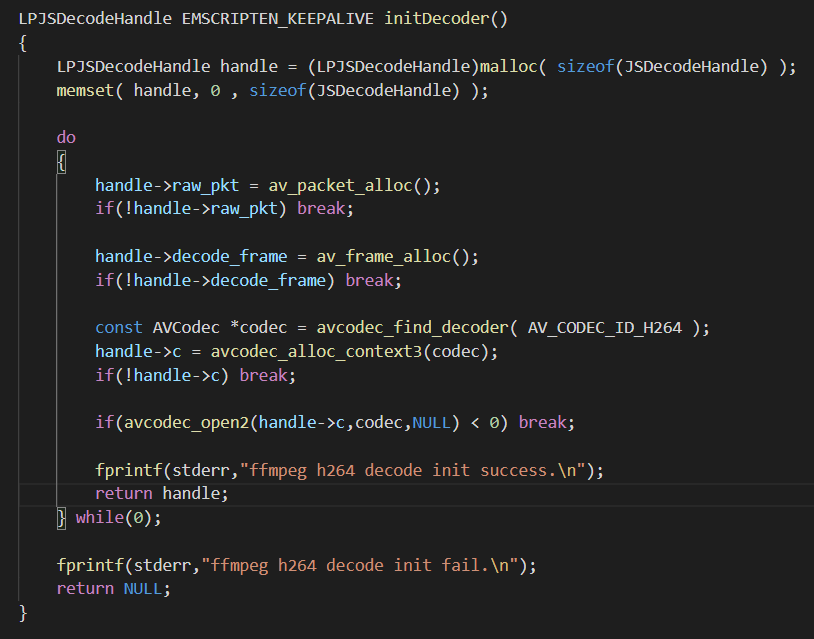
比如，我们在C++层定义初始化如下:



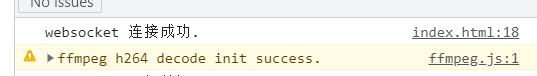
那么对应的前端调用则为:



函数的调用方式直接就是: Module.\_函数名类似的方式.



网上JS输出:



可以看到这个时候，我们的初始化已经成功了.

JS层传递数据到C/C++层

现在我们是在JS层收到的H264数据，这个时候我们需要实现的功能是: 把JS层收到的数据传递给C/C++.

数据从JS层到C/C++:

1. 我们首先在C/C++层malloc一块内存,用来存放JS层传递过来的数据,
2. 这块分配的内存，JS层主动获取，C/C++同样返回句柄.
3. 通过 HEAPU8.set() 函数，把js收到的数据传递到C++层.

数据从C/C++层到JS层.

解码成功后，我们是在C/C++层得到了解码后的YUV数据，所以我们需要把YUV数据从C/C++ 层传递回JS层.

1. 同样的我们把解码成功后的YUV内存句柄传回给JS层.
2. 通过 HEAPU8.subarray() 拷贝数据到JS层.



HEAPU8.subarray()，可以把我们C/C++层的数据传入到JS层.