学号：205544

东南大学专业学位型研究生
学位论文开题报告及论文实施计划

院（系、所） 软件学院

学位类别 工程硕士

专业领域 软件工程

研究生姓名 缑通旺

指导教师（校内） 孔佑勇

指导教师（校外） 刘庭华

开题报告日期 2022 年 5 月

东南大学研究生院制表

填 表 须 知

1、 论文开题报告由研究生本人向审议小组报告并听取意见后，由研究生本人填写此表。

2、 论文开题报告填写完成后，必须经导师审批，通过后方能提交。

3、博士生应在第四学期内、硕士生应在第三学期内完成此开题报告。开题报告经研究生秘书在网上审核确认（硕士生至少半年、博士生至少一年）后方可申请答辩。

4、研究生开题前应填写查新报告。查新报告对专业学位博士作为必要环节。博士生查新工作可委托图书馆负责，也可在完成网络文献检索类研究生课程的学习或参加学校组织的网络文献检索培训后，自行组织查新检索，自行组织查新需要详细文献查新述评作为附件。自行查新报告须经导师审查后由开题报告审核专家组审核签字（或盖章）。硕士生开题查新参考上述办法，不作硬性要求。

5、本表一式两份，一份研究生自留放入本人“研究生档案材料袋”；一份由院（系、所）保存并归入院（系、所）研究生教学档案。

6、学位类别为：工程硕士；公共管理硕士；法律硕士（非法学）；工商管理硕士；建筑学硕士；风景园林硕士；临床医学硕士；公共卫生硕士；旅游管理硕士；会计硕士；国际商务硕士；资产评估硕士；工程管理硕士；艺术硕士；工程博士；医学博士等。

7、本表下载区：http://seugs.seu.edu.cn/3676/list.htm 。本表电子文档打印时用 A4 纸张，格式不变，内容较多可以加页。

一、学位论文开题报告

<table>
<tr><td>论 文
题 目</td><td colspan="9">基于 FFmpeg 的 Web 音视频处理系统的设计与实现</td></tr>
<tr><td>研 究
方 向</td><td colspan="9">软件工程</td></tr>
<tr><td rowspan="2">题 目
来 源</td><td>国家</td><td>部委</td><td>省</td><td>市</td><td>厂、矿</td><td>自选</td><td>有无合同</td><td>经费数</td><td colspan="2">备注</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>√</td><td></td><td></td><td colspan="2"></td></tr>
<tr><td>题 目
类 型</td><td>理论
研究</td><td>应用
研究</td><td>工程
技术</td><td>跨学科
研 究</td><td>其他</td><td colspan="5">工程技术</td></tr>
</table>

开题报告内容（具体要求见《东南大学研究生论文选题和开题报告的原则和要求》）

一、选题依据与工程应用价值

1.1 选题依据

随着互联网技术的不断发展，同时在新冠肺炎疫情的双重影响下，短视频行业、直播行业、线上课程视频等快速增长，越来越多的用户相比较文字等传统传播介质，对于视频这类媒介具有更加广泛的活力和亲和力。短视频的创作者日益增多，人们对短视频的创作兴趣日益浓厚，而短视频的创作门槛、音视频的剪辑处理等都是影响短视频进一步增长的因素之一。虽然短视频平台功能也在不断完善，尽可能的满足创作者的需求。但针对短视频创作者对于音视频快速创作的需求，帮助短视频创作者可以通过在 Web 环境中便捷、快速、简单、易用的完成音视频处理的需求。但是通常情况下，因为各个浏览器对于音视频封装格式、编解码协议支持的程度差异大、不够全面等问题导致一些常见的音视频无法被浏览器所识别和解析。同时如果基于 JavaScript 实现音视频相关处理，会受限于 Web 架构的天然局限性，因为 Web 浏览器底层虚拟机是对代码的解释执行，所以在处理像音视频这类 CPU 密集型任务时会有严重的性能缺陷，导致音视频解析性能低、系统卡顿、延迟高等问题。

所以本课题基于 Fast Forward Moving Picture Experts Group[1-2]（以下简称 FFmpeg）音视频编解码基础库和 WebAssembly[3-5]（以下简称 WASM）编码设计与实现 Web 音视频处理系统，从而实现在 Web 环境下对丰富的音视频格式、编解码协议的安全、高性能的解析处理。其中 FFmpeg 主要用来支持市面上大部分的音视频格式、编解码协议；而 WASM 编码主要解决 Web 浏览器中 JavaScript 实现音视频处理性能低的问题，同时 WASM 可以解决 FFmpeg 这类 C 语言源码无法直接运行在 Web 环境，所以本课题通过设计迁移 WASM 编码将 FFmpeg 源码编译为 WAMS 模块，然后在 Web 浏览器利用 Web Worker[6-7]线程加载音视频文件和 WASM 模块，最后通过 JavaScript 调用该 WASM 模块进行音视频相关的处理。

根据以上思考和实际需求出发，本课题可以总结为基于对 FFmpeg 这款音视频编解码基础库进行二次开发，并迁移 WASM 编码到 FFmpeg C 语言源代码编译时，使其能够安全高效的在 Web 环境对音视频进行解析处理。

1.2 工程价值

基于 Web 环境的强大生态、易用、灵活的技术架构的特点，但不擅长音视频相关的大量数据计算的局限性，以及浏览器对音视频容器格式、编解码协议的支持有限。本课题通过迁移一种新的二进制字节码 WASM 对一些系统级应用程序 FFmpeg 音视频处理基础库库进行编译并在 Web 环境调用运行从而解决 Web 架构自身性能的局限性问题。本课题研究设计的工程价值主要是为以后对 CPU 密集型任务相关的编译型语言库迁移到 Web 这类不擅长处理大量数据的环境下提供了实践依据和宝贵的实践方案，以及实现在

Web 浏览器中对大部分格式的音视频可以高性能的解析处理，同时也体现了 Web 技术在互联网飞速发展的状况下，不断迸发出其多样灵活、丰富强大的能力。

二、国内外发展现状

2.1 音视频技术

随着网络技术的不断发展，音视频技术已经在很多场景下被广泛应用，例如：视频会议、实时直播、视频监控等等。音视频技术其实就是音频技术和视频技术的一个统称，一般音频和视频都是分开处理的。其中视频其实就是一系列连续的图像构成，由于人眼睛的结构，当图像快速切换的时候，画面会有残留，感觉起来就像是连贯的动作，每一张图像也叫做一帧视频帧。音频是通过数字信号记录了声音波的振幅和频率，通过保存声音在各个时间点上的振幅，当然数字信号不能连续保存所有时间点的振幅，但是通过一定采样频率保存的数字信号同样可以还原出人耳分辨不出来的音频。

在音视频处理过程中，最重要的 2 个步骤分别是音视频数据的编解码和压缩后的文件封装与解封装。

因为用户对音视频的品质要求越来越高，音视频在设备采集后的数据量非常庞大，仅仅依靠增加硬盘容量和通信带宽远远不够，尤其是视频随着分辨率、帧率的提升，必须要尽可能的压缩音视频的体积，等到音视频数据传输到客户端再解压缩处理。而音视频可以编解码的最重要的基础是音视频数据信息中普遍存在的大量容余，对视频信号来说会有时间容余、空间容余、编码容余等等。因此针对音视频的传输、存储、压缩出现了很多关于音频和视频的编解码算法、容器格式等等，全球视频编码标准分为很多派系，其中主要以 ITU（国际电信联盟）提出来的 H.26X 系列和 ISO/IEC 提出的 MPEG 系列为主[8]。

而文件封装与解封装是指，为了方便传输和使用将多个编码后的音频和视频存放在同一个容器进行存储。通常像 MP3、MP4、WebM 这些就是容器格式，它们定义了构成媒体文件的音频轨道和视频轨道的储存结构，其中还包含描述这个媒体文件的元数据，以及用于编码的编码译码器等等。

通过音视频技术中的编解码与封装解封装，实现了对巨大的原始音视频大小的压缩，减少了音视频数据对传输带宽、存储的压力，但同样压缩与解压缩数据增加了算力的成本，要求客户端在拿到压缩的音视频数据后能快速的解压缩并分别进行图像渲染和音频播放。

2.2 Web 浏览器音视频处理

2.2.1 音视频格式编码兼容性

随着在 Web 中处理音频和视频成为一种常见的功能需求，从传统的 Flash 技术到 HTML5 中允许通过<video>和<audio>标签将视频和音频嵌入到网页中播放，都不可避免的因为浏览器对音视频支持程度差异，可能需要针对不同的浏览器提供不同的数据源来抹平兼容性问题，这无疑增加了技术成本。

因为音视频技术的相关专利问题、硬件问题等等，导致如今市场上常见的浏览器对音视频格式和编解码的支持程度也非常有限。各个浏览器厂商在 HTML5 实现中对音视频容器格式的支持也存在差异，其中音频格式兼容性如表 2.1 所示，对 OGG、MP3、WAV 这三种音频格式的支持，除了 MP3 这种最常见的音频是基本上所有浏览器支持，其他的音频格式浏览器支持都存在兼容性问题。同样如表 2.2 所示，各个浏览器对视频容器格式的兼容性支持除了 MP4 其他的都存在不一致的情况。除此之外，还有一些其他的未在表中罗列的音视频格式浏览器兼容性支持更差，这就造成相对应的音视频格式无法直接在浏览器中被解析识别。

表 2.1 浏览器 HTML5 音频格式兼容性

音频	Chrome6+	Firefox3.6+	IE9+	Safari5+	Opera10+
OGG	支持	支持	不支持	不支持	支持
MP3	支持	支持	支持	支持	支持
WAV	支持	支持	不支持	不支持	支持

表 2.2 浏览器 HTML5 视频格式兼容性

视频	Chrome6+	Firefox3.6+	IE9+	Safari5+	Opera10+
MP4	支持	支持	支持	支持	支持
WebM	支持	支持	不支持	不支持	支持
OGG	支持	支持	不支持	不支持	支持

对于视频的编解码算法，浏览器支持的也很差，目前对最主流也是使用免费的编解码算法 AVC/H.264，其兼容性在各个浏览器表现较好，但是对于一些专利收费、硬件不支持的算法各个浏览器的支持程度都很一般。

表 2.3 浏览器 HTML5 视频解码算法兼容性

视频	Chrome6+	Firefox3.6+	IE9+	Safari5+	Opera10+
AVC/H.264	支持	支持	支持	支持	支持
H.263	不支持	不支持	不支持	不支持	不支持
H.265	不支持	不支持	支持	支持	不支持
MPEG-1	不支持	不支持	不支持	支持	不支持
MPEG-2	不支持	不支持	不支持	不支持	不支持

目前因为浏览器音视频兼容性问题，一般的解决的办法有下面几种，第一种是通过插件进行转码、转格式。比如 IE 浏览器上安装 ActiveX 插件或 YLC 插件能够播放 RTSP 协议或者私有协议的码流[9]。或者在目前主流的浏览器上安装 Flash 插件播放 RTMP(Real Time Messageing Protocol)协议的视频流，在网页端调用 Native 的代码，获得良好的用户体验，但是 Flash 已经停止更新，谷歌也在 2020 年 12 月起不再支持 Flash Player[8]。第二种基于无插件的 Web 浏览器音视频处理，大多是通过服务端解码后再通过网络协议和私有数据协议通过自研的播放器来处理兼容性问题。第三种就是对不同的客户端提供不同的播放数据源，这样对服务提供方增加了成倍的存储、传输成本。

2.2.2 Web 音视频处理性能

由于音视频数据需要大量的计算力，单靠纯 CPU 计算往往效率不是很高。在当前的电子产品里，对音视频的解码分为硬件解码和软件解码，硬件解码往往需要客户端自身的硬件内置一些支持音视频的编解码算法，缺点就是和具体的平台相关性太强。而软件解码就是通过程序来处理音视频，相比较硬件解码对特定的音视频算法的数据处理性能较差。

软解码带来的就是复杂的计算，这个计算可以是音视频编解码，也可以是各种图像处理、音效处理。但解码后的数据再使用 WebGL 和 WebAudio 等技术进行图像和音效处理，不再受限于系统提供的播放能力。但是软解码性能的缺陷主要体现在 JavaSciprt 是解释性语言且受限于 B/S 架构的局限性导致通过程序的解码性能会比较低。比如 HTML5 音视频播放器 flv.js，它通过纯 JavaScript 编写，没有使用 Flash。它的工作原理是将音视频数据流通过 JavaScript 流式解析为 flv 文件流，并实时封装为 mp4，再通过 Media Source Extensions 分段将数据塞给 Video 实现对音视频的处理。但是 flv 使用原生的 JavaScript 去解码 FLV 数据，其性能会有一定的损失，所以 flv.js 已经在加入 WASM 编码来提升自身性能。

还有 HLS 是 Apple 首先提出的流媒体分发协议，浏览器原生支持也不错，但是延时比较大，由于 HLS 协议本身的切片原理，基本延迟都在 10 秒以上，这对于一些低延时场景非常不友好，虽然 HLS 也在努力优化，但是想达到秒级延迟还是不现实的。

2.3 相关技术现状

2.3.1 FFmpeg

FFmpeg 库[10-17]是一个开源免费的跨平台音视频分离、转换、解码于一体的音视频工具，方便音视频

的相关，同时包含了对流媒体的格式转换，媒体协议的转变、音视频的码率控制，采样率的改变以及色彩格式的修改。FFmpeg 源代码采用 LGPL(Lesser General Public Lisense)或 GPL(General Public Lisense)许可证。FFmpeg 支持 MPEG、Divx、MPEG-4、FLV 等 40 多种编码方式，以及 AVI、OGG、Matroska、ASF 等 90 多种解码方式。FFmpeg 的开发基于 Linux 操作系统，并且可以在大多数操作系统中编译和使用，包括 Windows 平台、MacOS 平台甚至是安卓平台等[10]。因为其开源性、良好的跨平台性以及可移植等特点，得到了广泛应用，MPlayer、VLC 以及国内 QQ 影音等等播放器都用到了 FFmpeg 库。

FFmepg 为了达到可移植性的目的，提高视频编解码的质量。FFmpeg 适用于多种编码和解码方式，如 H.264 编码和 MPEG-4 等编码标准及 MPEG 解码。FFmpeg 包含以下几个重要方面，具体的模块如下：首先是 FFmpeg 的解码封装模块 AVFormat。此模块主要作用为实现多种媒体的音视频封装和解封装的格式，而且包含音视频的解析，并将解析后的视频流进行分离。然后是音视频的编解码模块 AVCodec，此模块包含多种原始音视频码流的编解码，并且能够满足多种操作系统运行使用的需求。FFmpeg 的滤镜处理模块 AVFilter，能够音视频及字幕进行滤镜处理，而且提供多输入，多输出的接口。最后的模块为视频图像转换计算模块 swscale，可以对图像进行图像像素的缩放和对音视频进行格式的转换，如图像 RGB 格式与 YUV 格式的互相转换[11]。

如图 2.1 FFmpeg 架构图所示，除了以上介绍的 FFmpeg 源码中的核心二进制 Library 库之外，在核心库上层是根据核心库依赖构建出来的简单易用的工具包，帮助二次开发者实现一些简单基础的功能。其中 ffmpeg 是 CLI 命令工具，一个强大的媒体文件转换工具，它可以转换大多数格式的媒体文件；ffprobe[12]是用来探测音视频文件的各种基本信息；ffplay[13]是一个播放媒体文件的工具，支持多种不同格式的音视频文件的解码播放。

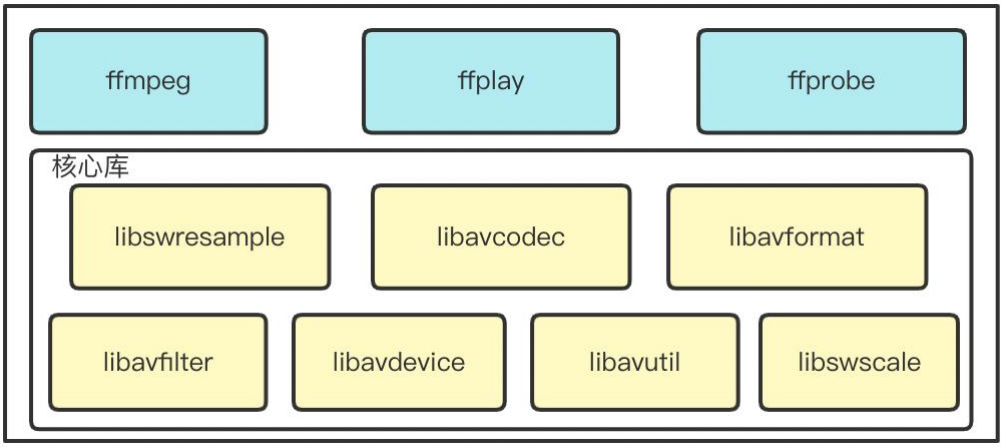

图 2.1 FFmpeg 架构图

2.3.2 WASM

WASM 是一个可移植、体积小、加载快、兼容型强，且拥有全新编码格式的二进制字节码，它可以在现代网络浏览器中直接运行，是一种低级的类汇编语言，具有紧凑的字节码格式，接近原生的性能运行，并在 2019 年 W3C 正式发布 WASM 的标准草案[18-22]。

WASM 字节码是一种抹平了不同 CPU（Central Processing Unit，中央处理器）架构的机器码，WASM 字节码不能直接在任何一种 CPU 架构上运行，但由于其非常接近机器码，可以非常快地被翻译为目标 CPU 架构的机器码，因此 WASM 运行速度和底层机器码接近。每个高级语言源码编译到不同平台的机器码的转换工作都是重复的，高级语言只需要生成底层虚拟机 LLVM[14]（Low Level Virtual Machine）认识的中间语言 LLVM IR（LLVM Intermediate Representation），这样 LLVM 就能实现 LLVM IR 到不同 CPU 架构机器码的生成、机器码编译时性能和大小的优化。除此之外，LLVM 还能实现 LLVM IR 到 WASM 字节码的编译功能，也就是说只要高级语言能转换成 LLVM IR，就能被编译成 WASM 字节码[21]。

WASM 可以在 Web 端运行其他语言（C、Rust 等）编写的程序模块，从而获得比较好的计算性能。目前 WASM 仍处于发展阶段，但针对 WASM 的研究和应用一直处于广泛关注的状态。WASM 适合用于大量计算的场景[17]，例如：Tensorflow.js 一种在浏览器中训练和推理模型的技术也利用了 WASM 来加快

模型训练、推理、可视化等等场景。WASM 目前被大多数浏览器厂商、多种编程语言支持，并且广泛应用于各种高性能容器场景，嵌入式系统以及边缘计算，同时尤其是给在 Web 技术架构下处理 CPU 密集型任务打开了一扇大门。

WASM 打包体积小是因为它采用了一种基于小端模式的编码算法进行可变长编码。WASM 对不同的数据类型选择了不同的编码方案，最大限度的利用各个编码方案的优点。其中主要有部分整数类型数值编码基于 LEB-128 的整数编码、浮点数编码基于 IEEE-754 编码、字符串编码基于 UTF-8 字符编码[6]。其中基于 LEB-128 的整数编码，是一种用于整数的、基于小端模式的可变长编码，所以可变长编码是指待编码的源数据在经过编码算法后得到的编码结果长度是不固定的。通过使用可变长对源数据进行无损数据压缩，并且被压缩后的数据也可以随时被再次解压缩回源数据，通过合理编码压缩，保证了编译后的模块体积处于最优[6]。

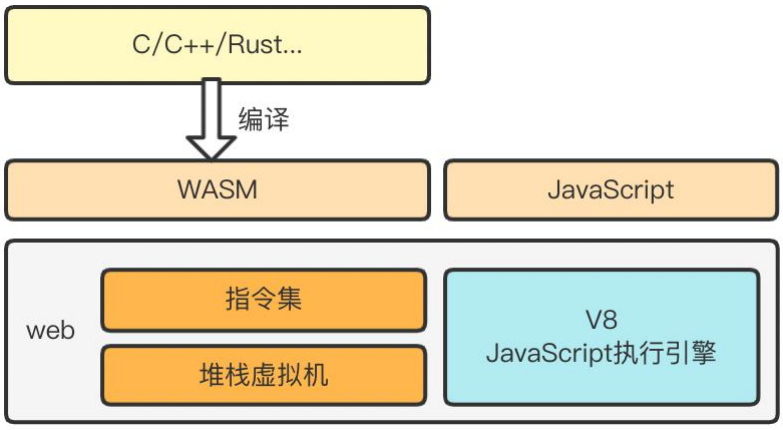

图 2.2 WASM 基本流程模型

如图 2.2 所示，WASM 运行效率高是因为它通过自定义虚拟指令集和拥有独立的堆栈虚拟机，并且不需要管理垃圾回收等问题。虚拟指令集是用于构成 WASM 模块核心功能的关键元素之一。需要让执行环境理解编码的意思，必须要让执行环境先理解指令集，然后根据指令集再对二进制编码进行编译，进而被执行。通过这两项核心 WASM 的执行效率几乎和 native 的源码运行效率差不多。

三、研究目标与研究内容

3.1 研究目标

针对 Web 浏览器中对音视频处理的兼容性问题、性能问题，本课题研究目标是实现音视频兼容性强，对多种音视频格式、编码协议进行识别解析的高性能 Web 音视频处理系统。

功能上实现音视频数据帧的编码、解码、拼接、丢弃、转换，并进一步实现音视频的播放与剪辑；非功能上利用 FFmpeg 的强大兼容能力提高 Web 浏览器对音视频格式、编解码的兼容性、并通过迁移 WASM 编码编译音视频处理程序嵌入 Web 浏览器中调用执行来提高处理性能。

3.2 研究内容

结合研究背景与研究目标，本课题内容主要是设计实现 Web 浏览器音视频处理系统，同时基于 FFmpeg 解决浏览器兼容性问题，并通过迁移编译 WASM 模块和多线程加载方案提高处理性能。针对实现后系统测试，分为功能性测试和非功能性测试，功能性测试采用黑盒测试方法，非功能性测试包含性能测试、PESQ[23]音频质量测试方法以及 PSNR[24]和 SSIM[25]视频图像质量测试方法。研究内容分为 3 个部分介绍:1、功能上研究 Web 音视频播放与编辑处理，包含基于音视频的解封装解码后的帧拼接、帧丢弃、帧编解码的音视频转换、增加字幕音频等功能；2、研究基于 FFmpeg 来解决多种格式、编解码算法的音视频处理的兼容性问题；3、研究 Web 浏览器如何调用执行 C 语言的音视频处理程序并提升其处理性能。

3.2.1 Web 音视频的播放与编辑

本小节主要是对系统音视频处理的帧拼接、编解码、裁剪的基础上，实现上层功能模块的方案。

（1）不同格式音视频解码播放

音视频往往有不同格式的封装，如 map4、mkv，不同编码的数据流，如 H.264、H.265 等等，针对不同编码的数据流，浏览器可能无法识别所以也就无法正常播放，因为本课题设计通过 FFmpeg 解码后的视频流数据数组通过 WebGL 2D 渲染每一帧的画面，如果浏览器不支持 WebGL 再转为 Canvas 去渲染，而音频流数据通过浏览器原生支持的 Audio 多媒体进行播放。设计如图 3.1 所示：

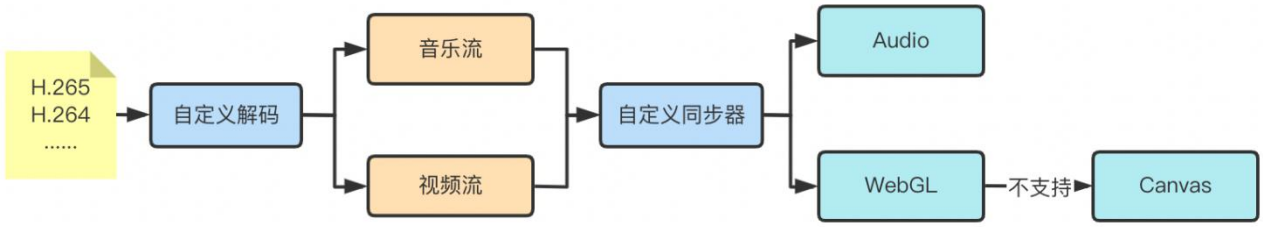

图 3.1 音视频播放设计

（2）音视频裁剪与合并

对多个不同格式、编码的音视频裁剪、合并的功能。

核心是首先要收敛不同格式、不同编码的音视频数据，针对音视频数据帧按时间戳对原视频进行裁剪或者对多个视频片段进行合成拼接。

（3）视频帧提取

对上传视频需要对视频进行提帧生成推荐封面，生成规则比较简单，根据视频总时长，每隔一定步长就对当前帧进行提取保留，传给渲染器渲染，用户可以从其中选择一张图片作为视频封面。这个功能的难点在于要遍历整个音视频的帧，如果音视频文件过大可能会影响处理的速度和生成的速度。

（4）融合音频、字幕

对当前视频增加音频或者字幕，尤其是字幕来说需要判断目标生成的容器封装是否支持字幕流，相 mp4 类型的封装只支持音频、视频流，所以对于不支持单独字幕流的需要将字幕硬编码到视频流中，这种方式的缺点就是无法视频。对于 mkv 这种支持单独字幕流的，只需要将字幕流和视频流的时间进行同步就能在视频容器中支持是否打开字幕的选项，比较有良好的视频体验。整体功能如图 3.2 所示，

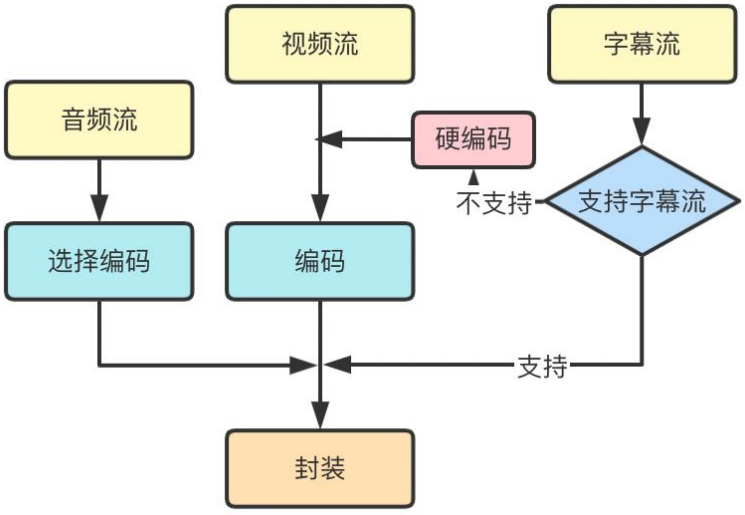

图 3.2 融合音频、字幕流

3.2.2 基于 FFmpeg 的音视频兼容性设计

研究 FFmpeg 在音视频转码、解析、拼接等音视频流操作的基本流程和功能原理，针对 FFmpeg 进行二次开发，设计与实现满足兼容性需求的流程模块设计，FFmpeg 的主要工作流程包含以下四个部分：解封装、解码、编码、封装。

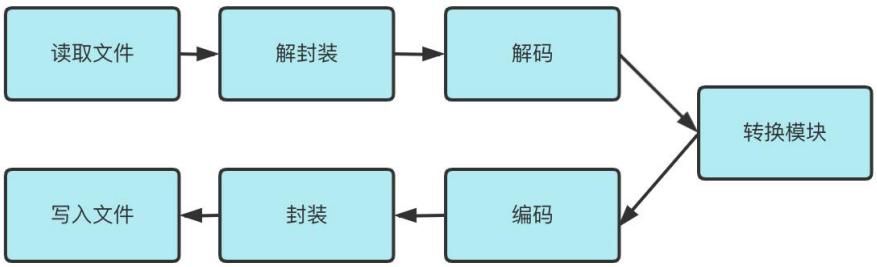

图 3.3 FFmpeg 处理流程模型

如图 3.3 所示，这四个部分在音视频处理又可以细化为以下 6 个步骤：读取输入源、进行音视频的解封装、解码每一帧的音视频数据、编码每一帧音视频数据、进行音视频的重新封装、输出到目标文件。

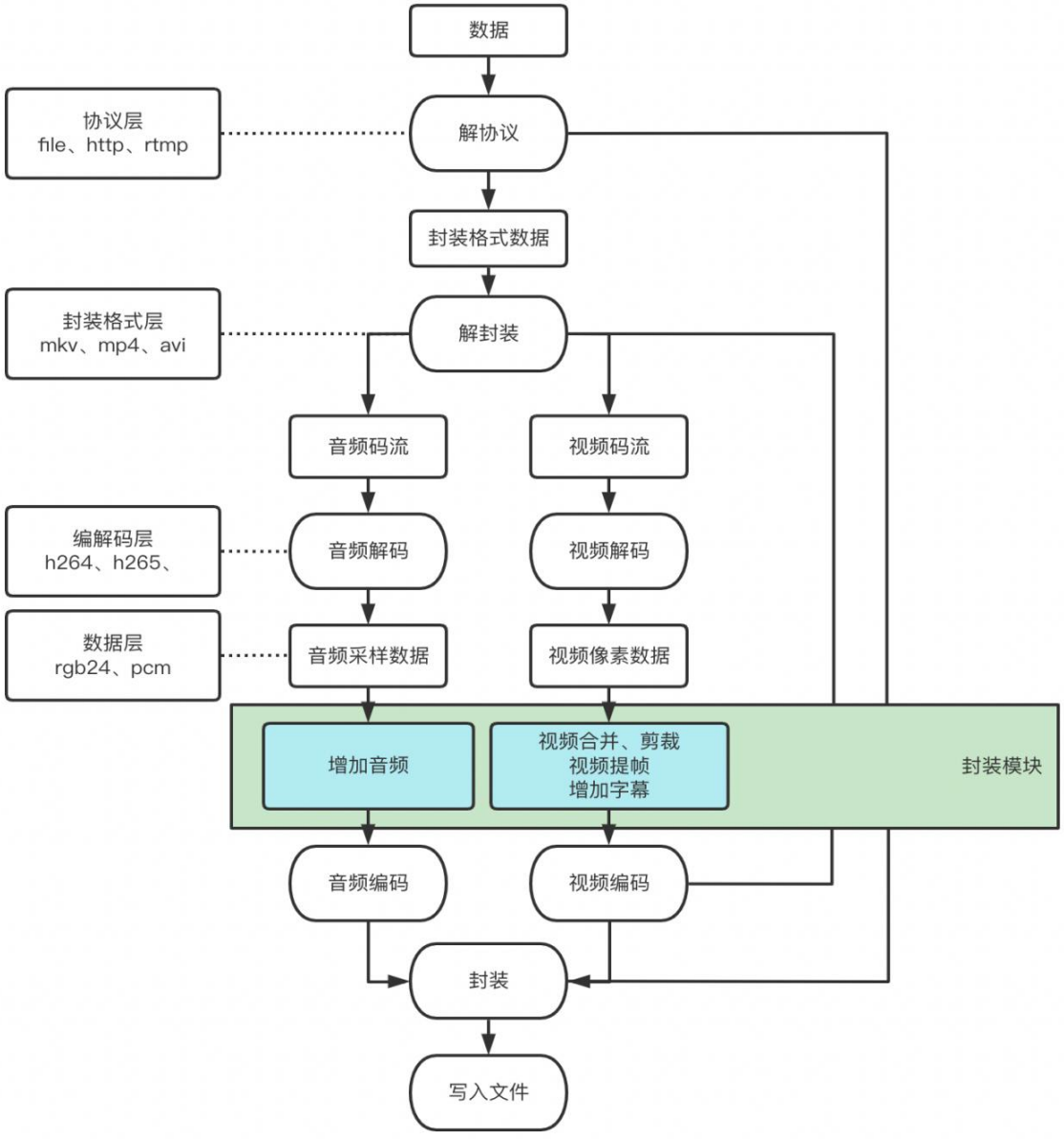

图 3.4 音视频兼容性处理设计

基于 FFmpeg 音视频操作的基本处理流程框架设计如图 3.4 所示系统音视频兼容性处理，从上到下，分别是协议层、封装格式层、编解码层和数据层，最后是对数据处理的功能层。对于音视频兼容性来说，协议层处理不同网络文件协议、格式层解析不同的音视频容器格式、编解码层对不同的音视频流编码进行解码，从而提高 Web 浏览器中音视频的兼容性。

基于 FFmpeg 音视频处理模块的内容主要分为以下 2 点：

1、实现 Web 浏览器兼容多种音视频的封装协议、编解码协议以及对音视频流的处理流程。

2、掌握 FFmpeg 开源库封装在音视频处理不同阶段的数据结构，利用暴露的抽象音视频数据结构进行二次开发。

3.2.3 音视频处理模块的编译与加载

因为 C 语言音视频处理程序无法直接在 Web 中调用执行，所以本课题通过移植 WASM 编码编译 C 程序，并通过 Web 浏览器加载编译后的模块进行调用，从而解决 Web 环境下处理 CPU 密集型任务的性能局限性和 Web 浏览器无法直接调用执行编译型语言程序的问题。通过搭建 WASM 的编译环境，对 FFmpeg 和二次开发的功能模块进行定制化编译，最终编译的模块再嵌入 Web 浏览器中被调用使用，如图 3.5 所示，

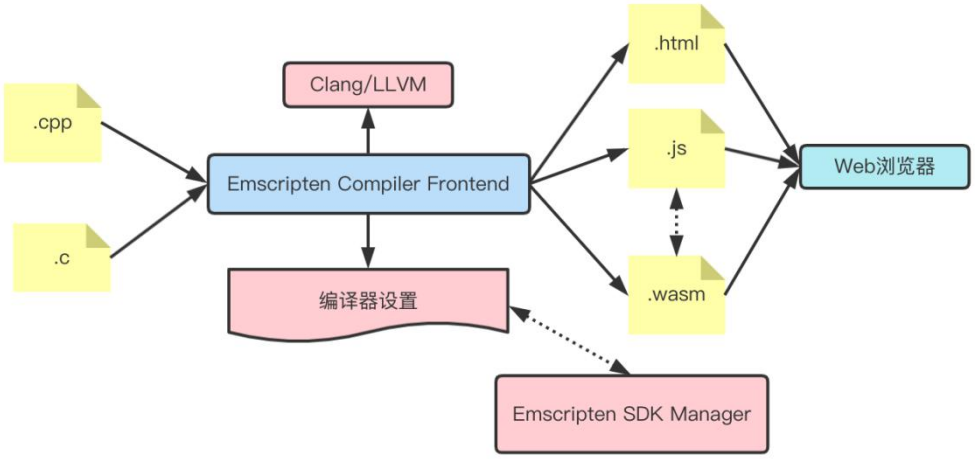

图 3.5 WASM 编译图

如图 3.6 对于 C 程序利用 Emscripten[26]工具链编译成二进制代码，因为底层二进制代码相比较高级语言的运行速度是更快的。同时 Web 浏览器底层对 WASM 代码有独立的堆栈虚拟机和自定义的虚拟指令集，可以更快的对代码进行执行，利用这一特性提高 Web 浏览器中的音视频处理性能提升。

C	Wat (.wat file)	Binary (.wasm file)

int factorial(int n) {
  if (n == 0)
  return 1;
  else
  return n * factorial(n-1);
}
	(func factorial
	  get_local 0
	  i64.const 0
	  i64.eq
	  if i64
	  i64.const 1
	  else
	  get_local 0
	  get_local 0
	  i64.const 1
	  i64.sub
	  call 0
	  i64.mul
	  end)
		20 00 42 00 51 04 7e 42 01 05 20 00
		20 00 42 01 7d 10 00 7e 0b

图 3.6 WASM 编译优化图

但是在 B/S 架构中，因为 JavaScript 是单线程语言，在加载音视频文件和 WASM 处理模块的时候，如果通过主线程去加载，就会导致加载文件时间过长，而阻塞主线程的功能正常使用。

所以针对 WASM 模块以及音视频文件过大导致的内存占用不够的情况。本课题设计 Web Worker 线程处理音视频文件的读取，对本地文件实时读取不需要等待所有文件加载到内存，再对文件数据流流式地通过 JS 主线程传递给 WASM 模块的 Web Worker 线程。对于 WASM 模块的加载，在必要的时候先对 WASM 模块进行 base64 编码，等到初始化的时候再通过 ArrayBuffer 对 base64 编码的 WASM 进行解码解析并加载到内存，同时对 WASM 模块暴露的功能，通过消息机制进行封装，暴露给 JS 主线程使用，达到模块之间高内聚、低耦合的架构设计，结合系统目标对加载文件的设计如图 3.7 所示：

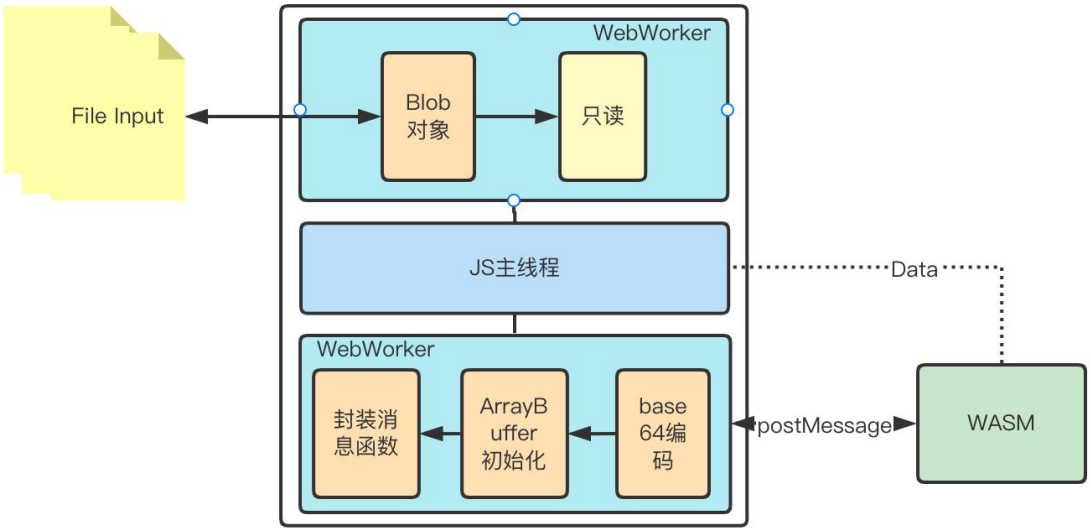

图 3.7 Web Worker 加载方案

最终设计音视频高性能处理模块的编译加载流程如图 3.8 所示：

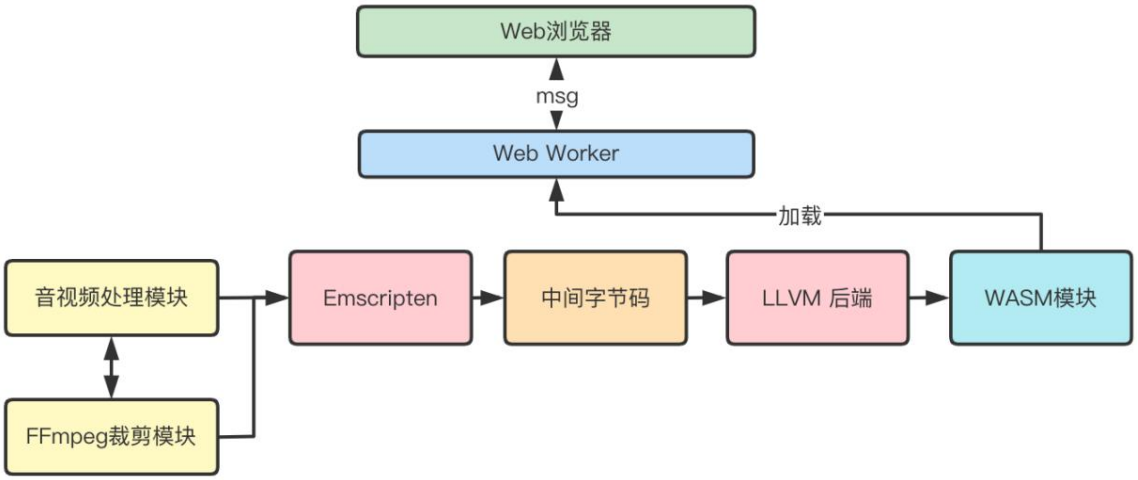

图 3.8 WASM 编译加载流程设计图

四、实施方案和可行性分析

4.1 系统实现

针对课题系统的实现过程如图 4.1 所示，其中核心分为三个实施步骤：

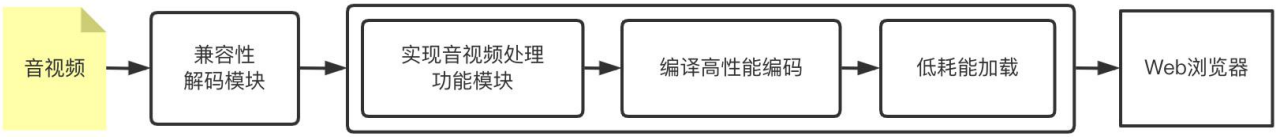

图 4.1 实施步骤图

4.1.1 实现高兼容性的音视频处理

针对 FFmpeg 二次开发，设计开发实现封装模块对音视频处理操作功能的需求，同时封装模块要暴露出来对加解封装、加解密码的配置化接口，以便对不同格式的音视频进行处理操作。其中需要掌握的是常用的音视频不同阶段的抽象数据结构[27-28]：

AVFormatContext：描述了媒体文件的构成及基本信息，是统领全局的基本结构体，贯穿程序始终，很多函数都要用它作为参数；

AVCodecContext：描述编解码器上下文的数据结构，包含了众多编解码器需要的参数信息；

AVCodec：编解码器对象，每种编解码格式(例如 H.264、AAC 等）对应一个该结构体，如 libavcodec/aacdec.c 的 ff_aac_decoder。每个 AVCodecContext 中含有一个 AVCodec；

AVPacket：存放编码后、解码前的压缩数据；

AVFrame：存放编码前、解码后的原始数据，如 YUV 格式的视频数据或 PCM 格式的音频数据等；

基于 FFmpeg 对音视频流数据的获取、解码，再读取解码数据帧后实现音视频处理功能。基本逻辑流程图如图 4.2 所示，其中最核心的是利用 avformat_open_input 接口获取到输入的音视频数据流，通过指针循环遍历数据流，找到并判断该数据流的类型、编码信息等，然后指定不同的解码器来实现解码，最后通过 av_read_frame 接口获取到音视频帧，再对音视频帧进行如需求分析中的拼接、裁剪、转换等处理，最后再实现上层功能，返回给前端再解析渲染。

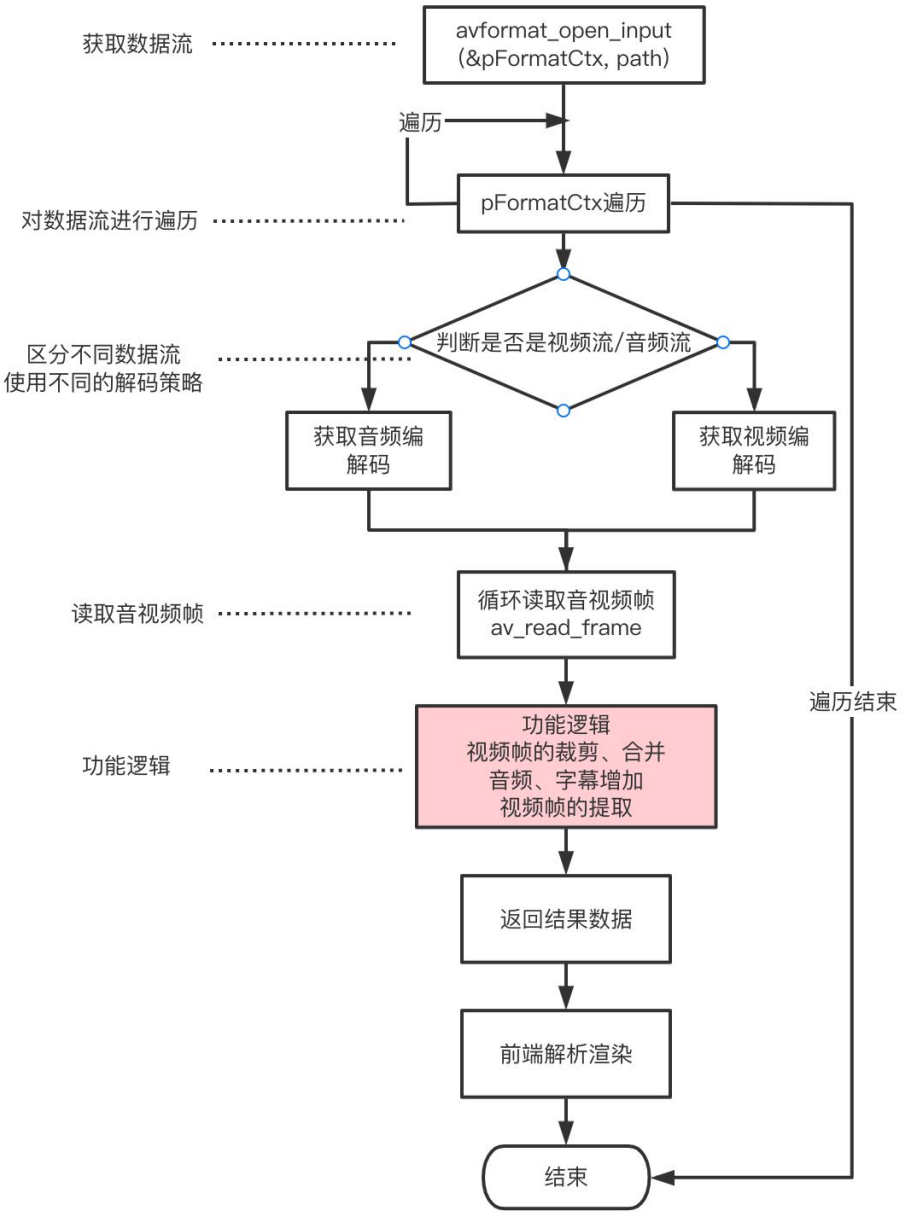

图 4.2 基于 FFmpeg 的音视频处理实现

4.1.2 定制化编译高性能音视频处理模块

本课题不采用整体编译使用封装好的 FFmpeg 命令功能以及参数，因为其不够灵活无法满足本课题对视频处理操作的二次开发。通过定制化编译，选取 FFmpeg 中本课题所需功能模块以及底层 lib 库进行构建编译，裁剪掉 ffplay、ffprobe 等不需要的模块，这样做的好处可以减少最终 WASM 模块的体积，提高其加载速度与执行速度。最终定制化编译 FFmpeg 和二次开发模块的流程，如图 4.3 所示，

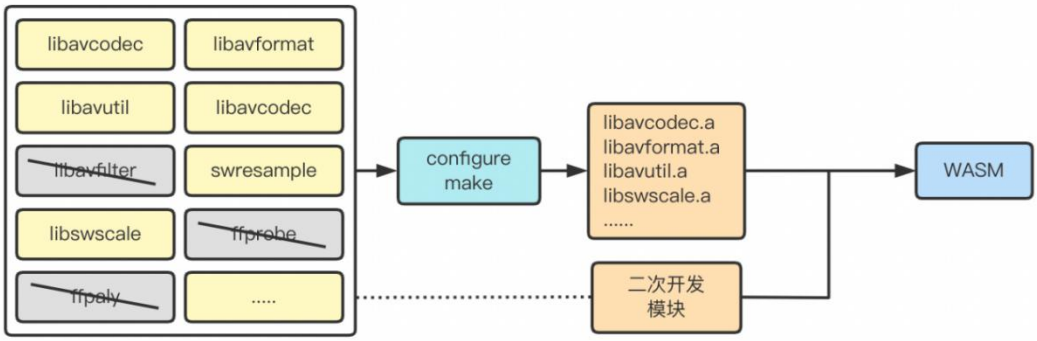

图 4.3 定制化编译流程

在 configure make 配置编译参数的模块，系统需要实现 C 语言程序和浏览器环境中的部分函数可以相互调用，从而保障浏览器可以完整调用自定义解码功能。如表 4.1 所示，通过 EXPORTED_FUNCTIONS 关键字指定 WASM 模块中函数名都可以在浏览器环境中通过 WASM 模块的引用直接获取并使用。通过 EXTRA_EXPORTED_RUNTIME_METHODS 指定的 addFunction 关键字，可以通过 addFunction 定义函数并在 WASM 模块中动态获取并行。

表 4.1 JavaScript 和 WASM 实现相互调用的方式

功能	具体参数
JavaScript 调用 WASM	EXPORTED_FUNCTIONS=[“methodName”]
WASM 调用 JavaScript	EXTRA_EXPORTED_RUNTIME_METHODS=[“addFunction”]

通过关键字 EXPORTED_RUNTIME_METHODS 可以扩展 WASM 模块中对源码的动态修改。除此之外，还可以增加 getValue、setValue、writeAsciiToMemory 等方法来丰富 C 程序与 JavaScript 程序相互调用的方法，满足音视频处理的复杂场景的需求。

本小节通过的 FFmpeg 源码阅读对定制化编译有了初步了解和设计方案，对于定制化编译的具体实施和详细设计，还需要进一步对 FFmpeg 编码学习和测试。

4.1.3 实现模块的多线程加载调用

JavaScript 是一种单线程的解释性编程语言，在加载 WASM 的过程中，如果 WASM 模块文件过大，会阻塞 JavaScript 主进程导致在一段时间内 CPU 都在加载编译 WASM 模块，所以系统的可用性、交互性等体验都急剧下降，尤其是当一段时间后 WASM 模块如果加载失败，也会导致系统崩溃影响系统的其他部分，比如主进程中的交互、渲染等。

Web Worker 为 Web 内容在后台线程中运行脚本提供了可能。通过 Web Worker 线程去执行任务而不影响 JavaScript 主进程。Web Workers 和主线程数据传递是通过消息机制进行通讯和同步，使用 onmessage 事件处理函数来响应消息。所以本课题针对 FFmpeg WASM 模块大小，采用 Web Worker 加载 WASM，通过主线程发送消息 Worker 线程处理消息，并根据回调函数返回处理结果。

4.2 系统整体设计

根据研究目标和研究内容，设计如图 4.4 所示的系统整体架构设计图，其中核心层就是迁移 WASM 编码以及音视频模块文件加载处理。

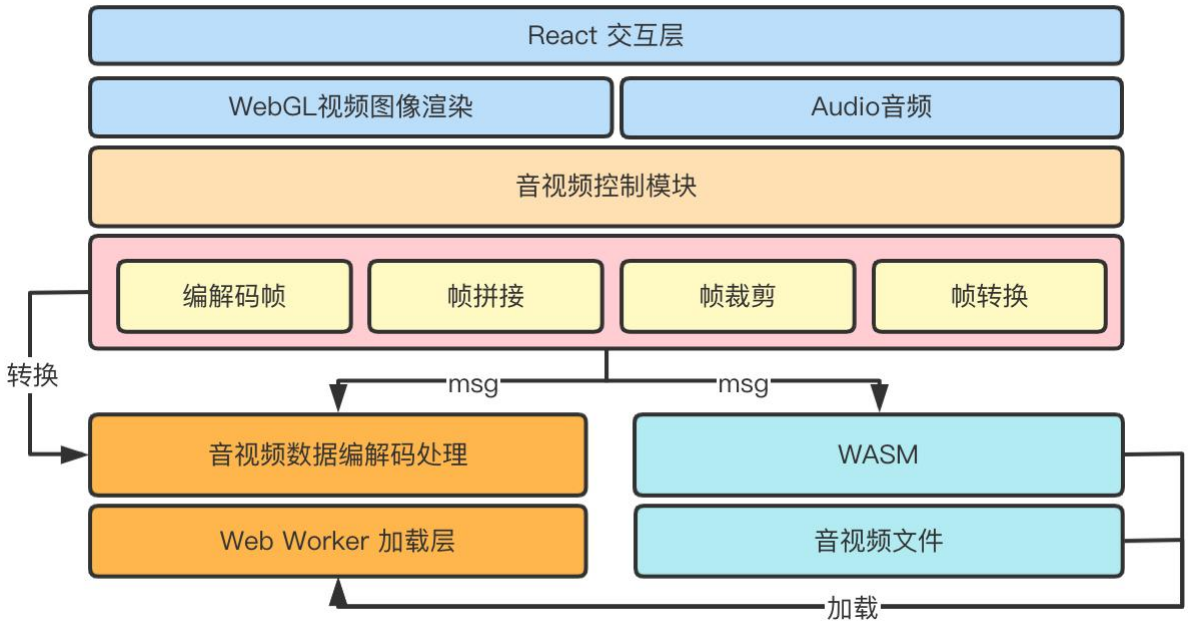

图 4.4 跨平台音视频处理系统总体架构设计

本课题采用 React 框架、npm 包依赖管理、webpack 前端代码打包器进行 Web 系统的开发与实现[29-31]，底层核心是通过 Web Worker 加载层去加载 WASM 模块和音视频文件到内存，在通过编解码处理层对压缩的音视频数据进行解码，再对解码后的数据帧进行帧拼接、帧裁剪、帧转换，处理后的数据帧通过同步控制模块传递给 WebGL 进行图像渲染以及 Audio 音频输出。

4.3 系统评估标准

(1) 功能性测试:

1、黑盒功能与兼容性测试

针对多种格式、编码的音视频，系统在多个浏览器上处理表现是否一致且正常。

(2) 非功能性测试:

1、性能测试

利用浏览器开发者工具可以测试当前页面的内存、CPU 使用情况。来衡量系统对音视频的编码、处理是否符合正常 Web 应用的刷新率、内存消耗、CPU 使用率等。同时对于是否通过 WASM 模块来处理音视频的速度做自对照实验，得出 WASM 模块对性能的影响程度。

2、音频质量测试

采用 PESQ 客观音频质量测试标准，将处理前音频作为参考信号，处理后的音频作为目标信号，将两个音频信号输入 PESQ 程序，得到对应的分数。PESQ 得分范围在-0.5~4.5 之间，得分越高说明音频质量越高。

3、视频质量测试

因为人肉眼在很多情况下无法直观的查看到视频图像的质量差异，所以本课题采用 PSNR（峰值信噪比）和 SSIM（结构相似性）两种衡量视频图像质量的标准来对音视频系统处理前后的视频图像质量进行比较。

其中 PSNR 用于衡量两张图像之间差异，值越大，视频图像质量越好。如公式（1）所示，其中 MSE 代表图像的像素值的均方差。

$$PSNR = 10log_{10}\frac{(2^{bits}-1)^2}{MSE} \tag{1}$$

而 SSIM 用于衡量两幅图像相似度的指标，对比的指标一般是两张图像的亮度、对比度以及结构。同时其值如果越大，代表视频图像相似度越高。如公式（2）所示，x，y 分别是指两种图像，其中 μ_x 是 x 的

平均值，μ_y是 y 的平均值，σ_x，σ_y是标准差,σ_{xy}是协方差。c1、c2 分别是稳定系数。

$$SSIM(x,y)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)} \tag{2}$$

比较系统处理前后的视频图像，主要目的是测试处理后的视频图像的质量不会因为处理而下降，如果系统处理前后的视频图像的 PSNR 值相近以及 SSIM 值大于标准值则代表视频图像的质量符合系统处理要求。同时测试处理后的音视频的各项指标比如码率、大小、编码、容器格式是否和预期一致。

4.4 可行性分析

搭建 WASM 编译环境，下载相关的编译工具，包含 Emscripten 工具链、CMake 高级语言编译工具，本课题采用 C 语言所以使用 GCC 或者 Clang 编译器以及 python2.7 版本以上，python 主要是用来充当编译过程的一些脚本功能。

(1) Hello World 实验

利用搭建好的编译环境，先对简单的 C 语言 Hello World 程序进行编译，并使其运行在 Web 浏览器下查看结果；通过 emcc 命令也就是安装 Emscrpten 工具链增加的全局命令对 hello.c 进行编译，编译后增加两个文件一个是 hello.js，另一个是 hello.wasm，其中 hello.js 生成了一些外壳胶水函数用来调用 hello.wasm 模块。通过 Node，一种服务端的 JavaScript 执行框架来执行 node hello.js，如图 4.5 所示通过 C 编写的函数代码编码为 WASM 后可以通过 JavaScript 执行调用并运行正确；同样在浏览器环境下打开控制台，也可以运行成功。

```
$ node hello.js
Hello, world!
```

图 4.5 Node 执行结果

(2) 斐波那契数列函数性能实验

针对上一小节的实验，进一步针对 JavaScript、C 语言以及 C 语言编译为 WASM 后的斐波那契额数列函数进行入参梯度对比，查看针对这种超深递归的 CPU 密集型计算 WASM 编码能否带来性能上的提升与优化。通过 emcc 命令将 C 语言版本的斐波那契数列函数单独编译为 WASM 模块，并在 JavaScript 中引入调用。最终经过在 Google 浏览器、Mac 2GHz 四核 Intel Core i5 的同等环境下测试，得出如表 4.2 所示的相关数据，可以显著的观察到 C 以及 C-WASM 相比 JavaScript 的执行时间几乎提高了 45 ~ 47%的范围程度。

表 4.2 JavaScript、C、C-WASM 运行结果对照

斐波那契函数	JavaScript	C	C-WASM
20	0.70ms	0.00ms	0.00ms
40	1284.20ms	682.443ms	669.60ms
45	14155.60ms	7513.88ms	7983.00ms

根据本小节内容，可以总结以下两点：

（1）、C 语言等编译型高级语言确实可以通过 WASM 编码为新型的独立二进制字节码，并可以在 JavaScipt 执行环境中正常使用。

（2）、WASM 编码格式的程序相比较纯 JavaScript 代码执行上保留了编译型语言高效的执行效率。

尽管本小节的实验存在一定的误差因素，考虑的纬度还不够全面。但足以证明 WASM 编码可以对 FFmpeg 源代码进行编译处理后，可以在 Web 浏览器中运行且可以拥有良好的执行效率。

通过实验、文献阅读、工业界实践可以得出通过迁移 WASM 编码到一些 CPU 计算密集型任务处理的代码库的可行性和可操作性，以及编译后的 WASM 编码在 Web 客户端的执行效率上保留了 native 源码的性能，对比 JavaScript 的性能提升是理论和实践都相互印证的。同时对系统测试有完备的功能性测试与非

功能性测试。最后个人在企业实践过程中，积累了对 Web 系统开发中用到的 React、Node 技术框架的实战经验。

五、参考文献

[1] Ken Tsutsuguchi. FFmpeg[J]. The Journal of The Institute of Image Information and Television Engineers,2010,64(3).

[2] Olivera Solís, Rafael Alejandro, López Pérez, et al. Codificación de video en HEVC/H.265 utilizando FFMPEG[J]. Ingeniería Electrónica, Automática y Comunicaciones, 2019, 40(2).

[3] Andreas Rossberg,Ben L. Titzer,Andreas Haas,Derek L. Schuff,Dan Gohman,Luke Wagner,Alon Zakai,J. F. Bastien,Michael Holman. Bringing the web up to speed with WebAssembly[J]. Communications of the ACM,2018,61(12).

[4] Romano A.,Wang W.. WasmView: Visual testing for webassembly applications[J]. Proceedings - International Conference on Software Engineering,2020.

[5] Sven Groppe, Niklas Reimer. Code Generation for Big Data Processing in the Web using WebAssembly[J]. Open Journal of Cloud Computing, 2019, 6(1).

[6] Javier Verdú,Juan José Costa,Alex Pajuelo. Dynamic web worker pool management for highly parallel javascript web applications[J]. Concurrency and Computation: Practice and Experience,2016,28(13).

[7] 邱珊. 使用 HTML5 Web Worker 提高 Web 的应用性能研究[J]. 软件导刊,2013(12):47-51.

[8] 薛粤桂. 支持 Web 直播的视频监控系统的研究与开发[D]. 广东:华南理工大学,2020.

[9] Watequlis Syaifudin Yan,Fahrur Rozi Imam,Ariyanto Rudy,Rohadi4 Erfan,Adhisuwignjo Supriatna. Study of Performance of Real Time Streaming Protocol (RTSP) in Learning Systems[J]. International Journal of Engineering & Technology,2018,7(4.44).

[10] Dave Rodriguez. Introduction to Audiovisual Transcoding, Editing, and Color Analysis with FFmpeg[J]. The Programming Historian,2018,7.

[11] Gaohe Li. Special Treatment of Video Image Based on FFmpeg[C]//.2018 联合国际先进工程与技术研究国际会议论文集.,2018:270-275.

[12] 薛芳芳,王凯悦,郭玉洁,马浩.基于 FFmpeg 的机载视频监控与通信功能设计[J].航空计算技术,2021,51(02):108-111.

[13] XIUYU ZHONG, ZHONGYI LUO. Design Of Video Bitrate Analyzer Based On Swift[C]. //2018 2nd International Conference on Electronic Information Technology and Computer Engineering (EITCE 2018)(2018 第二届电子信息技术与计算机工程国际会议)(EITCE2018)论文集. 2018:1-4.

[14] YUN CHENG, QINGTANG LIU, CHENGLING ZHAO, et al. Design and Implementation of MediaplayerBased on FFmpeg[C]. //Software engineering and knowledge engineering. Volume 2.:Springer, 2009:867-874.

[15] 陶奎印. 基于 FFmpeg 的教育直播系统设计与实现[D].大连理工大学, 2021. DOI: 10.26991/d.cnki.gdllu.2021.001671.

[16] 余海鑫,丁航,李文邦.基于 Vapoursynth 和 ffmpeg 的视频编辑[J].电子世界, 2022(01): 164-165+167.

DOI:10.19353/j.cnki.dzsj.2022.01.076.
[17] 岳瑞. 基于 FFmpeg 的音视频转码系统的设计与实现[D].西安电子科技大学,2021.
[18] 于航，著. 深入浅出 WebAssembly[M]. 北京:电子工业出版社，2018
[19] 薛超. 基于 WebAssembly 的 JavaScript 性能优化方案研究与实现[D]. 陕西:西北大学,2019.
[20] 匡开圆. 基于 WebAssembly 的 JavaScript 代码虚拟化保护方法研究与实现[D]. 陕西:西北大学,2018.
[21] Manuel Rigger,Matthias Grimmer,Christian Wimmer,Thomas Würthinger,Hanspeter Mössenböck. Bringing low-level languages to the JVM: efficient execution of LLVM IR on Truffle[P]. Virtual Machines and Intermediate Languages,2016.
[22] Paul Krill. WebAssembly may go live in browsers this year[J]. InfoWorld.com,2016.
[23] A. Olatubosun,Patrick O. Olabisi. An Improved Logistic Function for Mapping Raw Scores of Perceptual Evaluation of Speech Quality (PESQ)[J]. Journal of Engineering Research and Reports,2018.
[24] Bosse S, Siekmann M, Samek W, et al. A perceptually relevant shearlet-based adaptation of the PSNR[C]. 2017 IEEE International Conference on Image Processing (ICIP). IEEE, 2017: 315-319.
[25] Hore A, Ziou D. Image quality metrics: PSNR vs. SSIM[C]//2010 20th international conference on pattern recognition. IEEE, 2010: 2366-2369.
[26] Jiang Chen,Jin Xi. Quick Way to Port Existing C/C++ Chemoinformatics Toolkits to the Web Using Emscripten.[J]. Journal of chemical information and modeling,2017,57(10).
[27] 蒙敏荣.多媒体通信中的音视频同步问题探讨[J].电子技术与软件工程,2016(07):86.
[28] Jan Ozer. Six FFmpeg Commands You Can't Live Without[J]. Streaming Media Magazine,2019.
[29] React. 2019. React - a javascript library for building user interfaces, https://reactjs.org/
[30] 蔡兵,王啸楠.移动 Web 应用的前端工程化实现[J].长沙大学学报,2020,34(05):48-51
[31] 周伟，郑世珏.Web 前端工程化解决方案研究[J]. 信息技术，2018, 42(08): 44-47.DOI: 10.13274/j.cnki.hdzj.2018.08.010.
[32] Judy McConnell. 2019. WebAssembly support now shipping in all major browsers - The Mozilla Blog. https://blog.mozilla.org/blog/2017/11/13/webassembly-in-browsers/

研究生签名: 张通旺

2022 年 05 月 01 日

二、学位论文工作实施计划

（一）论文的理论分析与硬件要求及其预期达到的水平与结果

本课题是基于 FFmpeg 开源音视频处理库实现的基础上设计与实现兼容性强、性能高的 Web 音视频处理系统。主要流程是通过对多种格式的音视频兼容性解封装、解码之后，实现对于解码后的音视频帧同步播放以及实现多个音视频的合并、转换格式、添加字幕音频等功能；同时在解封装、编解码等耗费性能的操作上，本课题通过将 C 语言的音视频处理程序编译为高性能的 WASM 编码模块,并设计多线程加载方式来实现在 Web 浏览器中的加载调用。最后也要确保处理后的音视频质量均在系统测试标准范围之内。

（二）论文工作进度与安排

起讫日期	工 作 内 容 和 要 求	备 注
2021 年 7 月-2022 年 2 月	实践积累的相关技术经验，实现 H.265 的音视频 Web 播放的 Demo	
2022 年 3-4 月	搜集文献资料，搭建编译环境并尝试编译 FFmpeg	
2022 年 5-7 月	设计与实现 FFmpeg 的功能模块	
2022 年 7-9 月	设计与实现二次开发模块的编译和加载方案	
2022 年 10-12 月	基于已有模块设计与开发 Web 系统	
2023 年 1-3 月	整理撰写学位论文与完成系统开发	
2023 年 4 月	论文修改和完善	

学校指导教师对开题报告的综合意见	同意开题。 指导教师（签字）孔佑勇 2022 年 5月 1日
校外指导教师对开题报告的综合意见	同意开题。 指导教师（签字）刘庭华 2022年 5月 1日

开题报告审议情况记录	1、审议小组成员（硕士至少5人，博士5—7人，其中1人须为校外导师）： 组长：何洁月 成员：陈飞 王世杰 方效林 魏通 2、审议小组意见 同意开题 3、投票表决结果 审议小组出席 5 人；通过 5 人；不通过 0 人。 开题报告质量 （优、良、中、通过） 4、审议小组组长（签名） 审议小组成员（签名） 陈飞 王世杰 魏通 方效林 2022年5月4日
院（系、所）意见： 院（系、所）负责人签名（或印章） 2022年5月 4日	
备注：	