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4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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 论 文 题 目 | 基于FFmpeg和WebAssembly的Web音视频处理系统的设计与实现 | | | | | | | | |
| 研 究  方 向 | 软件工程 | | | | | | | | |
| 题 目 来 源 | 国家 | 部委 | 省 | 市 | 厂、矿 | 自选 | 有无合同 | 经费数 | 备注 |
|  |  |  |  |  | √ |  |  |  |
| 题 目  类 型 | 理论  研究 | 应用  研究 | 工程  技术  **√** | 跨学科  研 究 | 其他 |  | | | |
| 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》）   1. **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。根据《2021中国网络视听发展研究报告》[1]（以下简称《报告1》）和《2021年短视频用户价值研究报告》[2]（以下简称《报告2》）中分别指出，当前短视频深度渗透、全面融入用户生活，从看视频到拍视频，视频逐渐成为网民表达自身情感、生活、想法的工具。其中《报告1》指出，2020年6月至12月，我国新增网民4915万。其中，25.2%的新网民因使用网络视听类应用而接触互联网，短视频对网民的吸引力最大，20.4%的人第一次上网时使用的是短视频应用，仅次于即时通信，排在第二位。46.1%的用户在过去半年上传过短视频，这一比例大幅增长，较2019年增长28.6%。针对短视频的价值研究，《报告2》指出，短视频在媒介生态中的地位持续攀升，从具体使用时长看，日均观看短视频超过60分钟的用户占比达56.5%，人均每天使用时长升至87分钟，预期观看短视频时长增加的用户占比升至57.9%。同时，短视频成为用户碎片化时间的黏合剂，“晚上睡觉前”观看的用户占比上升最快，2021年升至61.3%；20.7%的短视频用户选择在“看电视时”看短视频。  根据以上报告可以总结以下，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[3-4]（以下简称FFmpeg）音视频编解码基础库和WebAssembly[5-7]（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主要用来实现支持市面上大部分的音视频格式、编解码协议；而WASM编码主要解决的是Web浏览器中JavaScript实现音视频处理性能低的问题。因为FFmpeg这类C语言源码编写的二进制程序无法直接运行在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[8-9]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。   1. **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[10]。  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2.1 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[11]。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[10]。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+，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[12-19]。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[12]。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[13]。  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[14]是用来探测音视频文件的各种基本信息；ffplay[15]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[20-24]。  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[16]（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[23]。  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为什么可以运行效率高且打包体积小？  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。   1. **研究目标与研究内容**   **3.1 研究目标**  本课题针对Web浏览器中对音视频兼容性问题、处理性能问题，实现在Web浏览器对多种音视频格式、编码协议进行识别解析处理，最终可以总结为以下3点研究目标：  (1) 针对Web浏览器对音视频的容器格式、编解码协议支持兼容性差的问题，本课题通过迁移FFmpeg音视频处理基础库提高多种音视频格式的兼容性。  (2) 针对Web浏览器JavaScript实现的音视频处理性能差的问题，本课题通过设计对C音视频处理程序编译为WASM模块，并设计WASM模块的加载方案来提高Web浏览器中处理音视频封装、解封装以及编解码的性能。  (3) 针对本课题的功能需求出发，本课题对音视频解码后数据流进行二次开发，实现对音画同步渲染播放、视频帧提取、增加字幕音频等功能目标。  **3.2 研究内容**  结合研究背景与研究目标，本课题的研究内容可以概括为以下3个方面：基于FFmpeg的Web音视频处理的兼容性提升、设计音视频处理程序的WASM模块的编译和加载方案、实现Web浏览器音视频处理系统的开发与测试。最终整体总结如图3.1所示：    图3.1 研究目标和研究内容概览  对于系统的功能需求分析如图3.2所示，系统主要实现的功能包含音视频的同步渲染播放、音视频剪辑、音视频提帧。其中音视频剪辑又分为视频增加字幕、增加音频、视频剪裁以及视频合并四个功能点。以上功能需求中最重要的是以音视频同步播放为主，兼容多个浏览器对不同容器格式与编解码算法的音视频。除此之外，音视频剪辑主要是对音视频数据流帧的处理，音视频提帧主要是以一定步长对整个视频进行帧截取，最后生成图片供用户选择。    图3.2 系统功能需求实例图  最后基于功能需求的三点研究内容涉及的主要工作进行叙述：  **3.2.1 基于FFmepeg的音视频处理模块实现**  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，设计与实现满足需求的功能改写流程方案。  FFmpeg的主要工作流程包含以下四个部分：解封装（Demuxing），解码（Decoding），编码（Encoding），封装（Muxing）。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。如图3.3所示，我们需要根据系统功能目标结合FFmpeg提供的处理流程模型再对转换功能模块做进一步的设计与实现。    图3.3 FFmpeg处理流程模型  基于FFmpeg音视频处理模块的内容主要分为以下三点  1、实现音视频的封装协议、编解码协议以及对视频流、音频流、字幕流的处理流程。  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、基于功能需求分析图，实现系统功能的音视频处理模块。  **3.2.2 音视频模块的WASM编译和加载流程**  因为FFmpeg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FFmpeg，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。但是在B/S架构中，JavaScript语言单线程语言，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，本课题利用Web Worker，一种Web浏览器提供的多线程方案，设计编译好的WASM模块的加载与调用逻辑，实现与JavaScript语言相互调用、内存共享、安全加载的方式。最终音视频处理模块的编译加载流程如图3.4所示：    图3.4 WASM编译和加载流程  **3.2.3 Web音视频处理系统的开发与测试**  基于以上两个小节，本系统的核心模块编译加载流程已经设计好。在此基础上，本课题采用React框架[18]、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，并针对音视频处理的相关功能需求包含：音视频同步播放；音视频裁剪、合并；增加字幕、音频；视频帧提取等操作进行进一步的设计与实现，如图3.4系统整体的需求实例图。  同时针对最终的Web音视频处理系统，要进行对应的系统测试，测试系统功能是否完善、测试本系统是否解决了Web浏览器对音视频格式、编码算法的兼容性问题、测试本系统在音视频解码、处理的操作上内存、CPU、耗时等性能指标是否符合预期。  **3.3 小结**  本小节重点介绍了本课题的研究目标和研究内容，以及研究内容中最终的三个部分，分别是基于FFmpeg的音视频处理，Web浏览器中音视频模块的WASM编译和加载流程，以及Web音视频处理系统开发与测试。通过明确本课题的研究目标和研究内容，为本课题系统实施方案提供了清晰的思路和方向。   1. **实施方案和可行性分析**   **4.1 基于FFmpeg的音视频处理模块**  **4.1.1 音视频处理流程设计**  首先对FFmpeg音视频操作的处理流程框架进行预处理，设计对音视频处理功能的统一封装，再对本课题的重点二次开发的目标封装模块进行具体功能上的实现，如图4.1所示    图4.1 FFmpeg加解码音视频流程  系统音视频的处理流程图，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。需要针对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，以方便用户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：  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；然后利用C语言编写程序，实现自定义开发模块对音视频流数据的获取，具体逻辑流程图如图4.2所示：    图4.2 FFmpeg转换模块设计  基本上，基于FFmpeg二次开发所要实现的功能都是要基于此流程基础上开发，最终搭建好流程框架后，针对上图中功能逻辑部分再具体开发系统目标中的音视频渲染播放；视频裁剪、合并；视频帧提取；音频、字幕融合等功能。  **4.1.2 功能模块设计**  本小节主要是对系统中涉及到的功能模块目标进行简单介绍和设计。   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4.3所示：    图4.3 视频播放设计   1. 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。   1. 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4.4所示，    图4.4 融合音频、字幕流  **4.2 基于WASM构建FFmpeg.wasm**  **4.2.1 搭建编译环境**  构建WASM编码，首先要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，最终实现如图4.5所示：    图4.5 WASM编译图  (1) Hello World  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；  // hello.c  #include <stdio.h>  int main() {  printf("Hello, world!\n");  return 0;  }  通过emcc命令也就是安装Emscrpten工具链增加的全局command命令，执行emcc hello.c -o hello.js，当前目录下hello.c就会被编译，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4.6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，也同样可以运行成功。    图4.6 Node执行结果   1. 斐波那契数列函数   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。  // fib.c  #include <stdio.h>  int fib() {  If(n <= 1) {  return 1;  }  return fib(n - 1) + fib(n - 2);  }  int main (int argc, char \*\*argv) {  return 0;  }  // fib.js  function fib() {  if(n < 2) return 1;  return fib(n - 2) + fib(n - 1);  }  [20, 40, 45].forEach(n => { const start = performance.now();  const end = performance.now();  console.log(`time: ${(end - start).toFixed(2)} ms`)  })  通过emcc命令将fib.c文件单独编译为WASM模块，并在JavaScript中引入调用。  // script  fetch(’./fib.wasm’)  .then(wasm => wasm.arrayBuffer())  .then(WebAssembly.instantiate)  .then(module => {  const start = performance.now();  module.instance.exports.fib(value)  const end = performance.now();  console.log(`time: ${(end - start).toFixed(2)} ms`);  });  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4.1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4.1 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，在JavaScript中运行且可以拥有良好的执行效率，为本课题的研究以及后续工作打下了坚实的基础。  **4.2.2 整体编译FFmpeg**  整体编译FFmpeg这种方式是将FFmpeg源码全部构建为二进制产物后，再作为Emscripten工具的输入构建出FFmpeg.wasm模块以及FFmpeg.js胶水层代码。FFmpeg.js内部会导出函数或者全局变量，供外部使用，结果放在回调函数中。开发者可以通过在PostMessage传递任务参数以及目标文件的实例对象，从而调用胶水代码层的ffmpeg\_run函数，进而调用wasm模块中的实际被调用方，最后再一层一层的返回结果。  但这种编译方法，虽然不需要开发者关注FFmpeg内部的实现细节，但是也无法进行二次开发，并且缺点还有就是编译后产物体积太大，同时内部功能不可控，浏览器崩溃等问题都无法快速定位并解决，这些都非常影响系统生产环境下正常运行。  **4.2.3 定制化编译FFmpeg**  本课题不采用整体编译，使用封装好的FFmpeg命令功能[22]以及参数约定，因为其不够灵活，可能无法满足本课题对视频处理操作的一些定制化功能。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。FFmpeg框架的基本组成包含AVFormat、AVCodec、AVFilter、AVDevice以及AVUtil等模块库组成，本课题不涉及到滤镜方面的操作，所以可以裁剪掉AVFilter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。例如，还有FFmpeg自身构建的工具包ffplay、ffprobe，本课题都不需要，因为这些需要通过自身定制化去生成。  通过阅读FFmpeg源码，FFmpeg提供了对自身模块的定制化编译选项。通过--disable-ffplay，--disable--ffprobe可以实现编译裁剪FFmpeg的目的。同时需要指定--cc=”emcc” --arch --cpu等一系列编译参数。最终定制化编译FFmpeg和二次开发模块的流程，如图4.7所示，    图4.7 定制化编译流程模型  因为 WASM默认的调用c 函数的传参中只能传输 int 类型，所以需要通过 cwrap 的方式来帮助传输字符串类型, cwrap可以帮助JavaScript定义对C语言暴露的函数进行封装定义参数类型。从而实现将字符串参数传给 wasm, 关键代码如下:  -lworkerfs.js \  -s EXPORTED\_RUNTIME\_METHODS=’[“ccall”, “cwrap”]’ \  通过关键字EXPORTED\_RUNTIME\_METHODS可以扩展WASM模块中对C程序的动态修改。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，满足不同场景的需求。  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。  **4.3 基于Web Worker的模块文件加载方案**  **4.3.1 Web Worker加载流程**  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。本课题结合系统目标对加载架构设计如图4.8所示：    图4.8 Web Worker加载方案  本课题设计多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。  **4.3.2 基于React的Web Worker使用**  上一小节，介绍了关于如何利用Web Worker加载文件、WASM的整体设计流程。本小节介绍如何在React框架下使用Web Worker。  基于React前端开发框架在使用视频文件、WASM文件时，需要通过引入webpack打包机制结合filer-loader、worker-loader两个插件，其中worker-loader插件是专门用来处理Web Worker文件引入和初始化操作的loader，最终将上一小节编码完成的web worker加载模块通过该插件一起打包，最终生成浏览器可识别的前端代码。  **4.4 Web音视频处理系统的整体设计与测试**  **4.4.1 系统整体设计**  根据研究目标和研究内容，本课题将视频处理操作主要划分为：视频剪辑、合并；视频帧提取；增加字幕、音频三种代表性的功能操作。同时基于Electron Web跨平台解决方案，将浏览器环境、Node环境、React、WASM、视频文件模块结合起来，设计如图4.9所示的系统整体架构设计图，其中核心层就是结合4.1和4.2提出的迁移WASM编码和模块加载设计方案组成二进制编码功能层，作为整个系统应用的核心研究。    图4.9 跨平台音视频处理系统总体架构设计  根据系统的整体设计，可以实现系统一份代码根据不同操作系统编译为目标操作系统可以识别的二进制码，从而使得本系统可以在多个操作系统平台上直接运行，其中功能核心就是音视频相关处理的实现。  **4.4.2 系统测试**  最终系统开发后测试主要是黑盒测试，测试目标主要分为：   1. 针对多种格式、编码的音视频，系统在多个浏览器上表现是否正常。 2. 本系统对音视频的编码、处理是否符合正常Web应用的刷新率、内存使用率等。 3. 相比较处理前的音视频文件，对处理后的音视频的各项数据比如码率、大小、编码、容器格式是否和预期一致。   **4.5 可行性分析**  根据技术和时间，本课题的可行性分析如下。  （1）技术可行性  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。同时个人在企业实践过程中，积累了对跨平台系统开发中用到的React、Node技术框架的实战经验。  （2）时间可行性  本课题的研究工作将在开题后开始，从开题至毕业有13个月的时间，具体工作分为文献整理、音视频技术架构研究与编译方案实验、跨平台系统开发与测试以及论文撰写4个部分，各项工作将稳步推进，并按时汇报工作进度。  **五、目前进度**  目前课题的进度  (1)对于FFmepg源码进行了部分阅读以及改写，并掌握了基本的音视频处理流程。  (2)对于WASM编码，搭建完成了编译环境；并进行了一些小实验验证Web浏览器的可使用与性能部分；最后基于FFmpeg尝试编译了WASM解码模块。  (3)对于Web Worker加载方案、前端系统的搭建还处于准备阶段。  **六、参考文献**   1. 中国网络视听节目服务协会. 2021中国网络视听发展研究报告[EB/OL]. 北京: 2021.6 <http://www.cnsa.cn/attach/0/2112271351275360.pdf> 2. 中国广视索福瑞媒介研究（CSM）. 2021年短视频用户价值研究报告[EB/OL]. 北京: 2021.10 <http://www.cnsa.cn/attach/0/2112271351275360.pdf> 3. Ken Tsutsuguchi. FFmpeg[J]. The Journal of The Institute of Image Information and Television Engineers,2010,64(3). 4. 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). 5. 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). 6. Romano A.,Wang W.. WasmView: Visual testing for webassembly applications[J]. Proceedings - International Conference on Software Engineering,2020. 7. Sven Groppe, Niklas Reimer. Code Generation for Big Data Processing in the Web using WebAssembly[J]. Open Journal of Cloud Computing, 2019, 6(1). 8. 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). 9. 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. 10. 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. 11. 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 &amp; Technology,2018,7(4.44). 12. Dave Rodriguez. Introduction to Audiovisual Transcoding, Editing, and Color Analysis with FFmpeg[J]. The Programming Historian,2018,7. 13. 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. 14. 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. 15. 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. 16. 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. 17. 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. 18. 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. 19. 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. 20. 于航，著. 深入浅出WebAssembly[M]. 北京:电子工业出版社，2018 21. 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. 22. 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. 23. 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. 24. Paul Krill. WebAssembly may go live in browsers this year[J]. InfoWorld.com,2016. 25. React. 2019. React - a javascript library for building user interfaces, <https://reactjs.org/> 26. 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 27. 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. 28. 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. 29. 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). 30. Jan Ozer. Six FFmpeg Commands You Can't Live Without[J]. Streaming Media Magazine,2019. 31. 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/> 32. 李亚男. 基于微信小程序的轻量化AR关键技术研究与系统实现[D].北京邮电大学,2021.DOI:10.26969/d.cnki.gbydu.2021.000465.   研究生签名: WechatIMG22  2022年 04月 25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

|  |
| --- |
| **实验环境：**  硬件环境：PC机一台（Inter Core I5 CPU, 8G内存, 64位操作系统）  **预期成果：** |

（二）论文工作进度与安排

|  |  |  |
| --- | --- | --- |
| 起讫  日期 | 工 作 内 容 和 要 求 | 备 注 |
| 2022年3-4月 | 搜集文献资料 |  |
| 2022年5-7月 | 设计与实现FFmpeg的WASM编码 |  |
| 2022年7-9月 | 设计与实现二次开发模块和加载方案 |  |
| 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 |  |
| 2023年5月 | 论文答辩准备 |  |
| 学校指导教师对开题报告的综合意见 | 指导教师（签字）  2022年 03月 日 | |
| 校外指导  教师对开题报告的综合意见 | 指导教师（签字）  2022年 03月 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：  成员：  ２、审议小组意见  ３、投票表决结果  审议小组出席 人；通过 人；不通过 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）  年 月 日 | |
| 院（系、所）意见：  院（系、所）负责人签名（或印章）  年 月 日 | | |
| 备注： | | |