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3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的Web跨平台视频处理系统的研究与实现 | | | | | | | | |
| 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环境下便捷、快速、简单、易用的完成视频创作的需求，针对音视频处理操作中最常见的三类视频操作功能音视频剪裁、合并；增加弹幕、音效；视频帧提取功能， 本课题基于Fast Forward Moving Picture Experts Group[3]（以下简称FFmpeg）在Web跨平台环境下进行系统设计与实现。  通常情况，Web运行环境下是不可以运行系统级的应用程序，即就是像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。基于此，本课题提出通过迁移WebAssembly（以下简称WASM）编码到FFmpeg源码编译过程，对FFmpeg源码编译成相对应的WASM编码，然后在Web环境中加载并通过JavaScript调用该WASM模块。  WASM是一种新的二进制字节码，在2019年W3C正式发布WASM的标准草案。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[4]（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，同时WASM字节码已经被大多数浏览器厂商以及多种高级语言所支持。  同时针对跨平台[5]的系统实现方案，本课题通过结合Web的跨平台技术解决方案Electron框架来实现视频操作系统平台对操作系统的低依赖性，实现可以一份代码多平台编译安装执行系统整体需求，基于Electron[6]架构本课题设计一种可以嵌入WASM模块以及对视频文件进行加载的Web Worker[7]线程加载机制，来解决WASM模块以及视频文件过大导致的内存占用不够的问题。  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这款音视频编解码工具进行二次开发，并迁移WASM编码到FFmpeg C语言源代码编译时，使其能够正常高效的在Web环境中运行。同时结合Web跨平台技术框架Electron对Chromium浏览器内核以及Node进行封装，从而实现一次编码可以编译成多个平台的目标代码进行安装和使用。  **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，以及实现Web跨平台解决方案上的缺陷。本课题通过研究提出迁移一种新的二进制字节码WASM，对一些系统级应用或者程序迁移到Web环境运行的可能性提供了有效的实践基础，并通过跨平台的解决方案对系统进行实现。本课题研究设计的价值是可以为以后对CPU密集型任务相关的编译型语言迁移到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，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 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[8]。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[9]。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。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[10]。  如图1.1 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[11]是用来探测音视频文件的各种基本信息。ffplay[12]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  **2.2 跨平台解决方案**  传统的桌面应用程序开发在一段时间内主导了软件开发的进程，随着Web应用程序的发展，传统的桌面应用开发因为其不够灵活、组件审美不丰富、迭代周期慢、复杂度高等缺点导致传统的桌面应用程序也在吸收着Web技术的优势，产生了跨平台的桌面应用开发技术。传统的桌面应用开发是强依赖于不同的操作系统底层架构，所以在不同的操作系统上都有特定的开发框架和开发语言。桌面应用需要更加关注与操作系统的交互以及多线程的使用，需要严格系统文件权限、系统托盘、剪贴板以及系统软硬件的的差异性。  如表2.1所示，据国外Statcounter调查统计全球桌面操作系统的市场占有情况，其中Windows操作系统和MacOS操作系统几乎占据了90%多的市场份额，也导致大部分开发框架的发展由这两种操作系统主导，而Linux操作系统通常用作服务器，对桌面应用的需求不高。  表2.1 操作系统市场份额   |  |  | | --- | --- | | 操作系统 | 市场占有率 | | Windows | 74.27% | | MacOS | 16.05% | | Linux | 2.09% |   表2.2 桌面开发框架详情   |  |  |  | | --- | --- | --- | | 框架/类库 | 语言/环境 | 适用平台 | | QT | C/C++ | 跨平台 | | WinForm | .NET/C# | Windows | | MFC | C/C++ | Windows | | Swift | Objective-C | Mac | | Swing/JavaFx/SWT | Java | 跨平台 | | NW.JS | JavaScript | 跨平台 | | Electron | JavaScript | 跨平台 |   如表2.2所示，我们可以简单将桌面应用开发分为三个方面： 第一，原生桌面应用开发解决方案  直接将程序编译为目标平台的二进制可执行文件，调用系统API，完成界面绘制。Windows平台主要是C#、.NET语言框架，从早期的MFC方案到WPF和WinForm,而MacOS系统则主要是Objective-C语言和Cocoa框架。但是原生桌面应用开发的缺点就是，不跨平台、复杂度高、迭代周期长，但是往往运行效率对操作系统的结合性都非常高，更适合大型复杂的桌面应用开发。  第二，QT、JavaFx一类的跨平台的框架解决方案  首先QT，是一个GUI的框架，支持跨平台，易移植，语法结构简单清晰，相比较原生更加容易简单。而且QT不仅仅支持C++，同时也支持Python等[13]。但是QT学习成本比较大，涉及到协议、QML等，开发周期也会比较长。但是其开源、丰富的UI库和文档生态，也是很多桌面客户端的首选。  以Java为首的JavaFx、Swing也是一类比较重要的开发模式[14]，优势是和Java的天然结合，但是其生态较差，并且Java运行时还要通过JVM来管理和维护Java类对象的内存分配，性能上不如C++等，开发周期又不如Web，同时组件也相对较少。  第三，Web桌面应用开发解决方案  Web技术开发桌面应用程序，从早期的node-webkit[15]到NW.js在到如今的Electron，目前有很多桌面应用都是基于Electron框架开发，例如Visual Studio Code、Atom、WordPress等等。随着Chrome V8引擎的出现，提高的JavaScript的执行性能，让JavaScript可以承担更复杂的应用开发。  Electron是一个基于Web构建桌面应用程序的底层工具框架。 它允许使用 Node.js 和Chromium (V8 引擎内核库)完成桌面 GUI 应用程序的开发[16]。通过嵌入Chromium 和 Node.js 到二进制的 Electron 可以构建跨平台桌面程序[17]。为了提供原生系统的GUI支持，Electron内置了原生应用程序接口，对调用一些系统功能，如调用系统通知、打开系统文件夹提供支持。  相比较原生C++等原生开发框架、QT等跨平台框架来说，Web技术跨平台桌面应用开发带来的是更加丰富的组件、更加灵活的技术架构、更快的开发周期以及更加繁荣的生态环境。   1. **研究目标与研究内容**   **3.1 研究目标**  本课题针对基于FFmpeg视频处理这类CPU计算密集型任务，迁移到Web客户端运行环境并且需要保持高效的执行效率。针对音视频处理本课题选取短视频创作常见的三种操作需求分别是：音视频裁剪、合并；增加字幕、音频；视频帧提取三类代表性的操作，其中视频帧提取主要是通过对最终的目标视频进行在一定步长内对当前帧画面提取，并通过canvas渲染出来，方便使用者选取短视频封面图的功能。以上这些视频处理是无法在Web客户端独立完成的，因为Web客户端性能缺陷，导致这类处理任务的执行时间长、内存占用高，从而无法达到生产使用的程度。基于以上原因，本课题研究目标分为以下三点：  (1)本课题提出通过迁移WebAssembly（以后简称WASM）编码方式到FFmpeg，通过对FFmpeg的二次开发实现上文提到的三类代表性操作后编译为WASM模块从而能够在Web环境下直接使用。  (2) 针对Web运行环境内存限制影响到因为视频大小从而导致视频解析加载失败的问题，本课题提出通过WebWorker线程加载编译好的FFmpeg的WASM模块的方式，并通过worker内Blob对视频文件进行只读操作，防止系统一次性将视频全部加载到内存。  (3) 针对系统平台的移植性、普适性，本课题结合Web跨平台的开发解决方案来满足对视频相关操作的封装与使用。  **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几个方面：   1. 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，搭建编译环境，针对FFmpeg进行二次开发，提出满足需求的功能改写方案，重点是构建可以使用的WASM编码格式的模块。 2. 研究针对视频操作占用内存过大的情况，利用Web Worker设计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。 3. 研究结合端到端的Web跨平台开发解决方案，来降低系统对操作系统平台的依赖性与增强系统的可移植性，同时又可以可以拓展系统对操作系统底层API使用的功能。   根据研究目标以及研究内容，本课题整体总结如图所示:    图3.1 研究目标和研究内容概览  **3.3 研究相关技术**  **3.3.1 FFmpeg**  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3.2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模块做进一步的设计与实现。    图3.2 FFmpeg处理流程模型  **3.3.2 WebAssembly编码**  本小节重点讲述关于WASM编码模型的机制。  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[18]。  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[19]，例如以下场景：处理音视频，Flv.js用WASM重写后性能有很大提升；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[20]。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[20]。    图3.3 WASM基本流程模型  如图3.3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。  **3.3.3 Electron web跨平台技术**  Electron结合了基于V8引擎的轻量浏览器内核Chromium和NodeJs丰富强大的系统层面的接口，高效利用了操作系统的能力，使得可以通过JavaScript来创建跨平台的桌面应用。  （1）、NodeJs  NodeJs是一个JavaScript运行环境，是对Google V8引擎进行了封装，用于方便地搭建响应速度快、易于扩展的网络应用。NodeJs使用事件驱动、非阻塞I/O模型而得以轻量和高效，非常适合在分布式环境中运行数据密集时实时应用。  （2）、系统API  为了提供原生系统的GUI支持，Electron内置了原生应用程序接口，对调用一些系统功能，如调用系统通知、打开系统文件夹、访问操作系统剪贴板等提供支持。  （3）、进程类别  Electron区分了两种进程：主进程和渲染进程。一个Electron应用总是有且只有一个主进程，主进程职责：  (1) 创建渲染进程（可多个）；  (2) 控制应用生命周期（启动、退出APP以及对APP做一些事件监听）；  (3) 调用系统底层功能，调用原生资源。  一个渲染进程相当于一个桌面应用窗口，其主要职责：  (1) 用HTML和CSS渲染界面；  (2) 用JavaScript做一些界面交互。  Electron技术优势是通过桌面应用应用Web技术的丰富性并通过其他V8引擎等保障运行时的效率和性能，可以即时启动，不需要等待资源从网络下载下来。可以访问计算机的操作系统和硬件资源，包括可以读写用户计算机中的文件系统。可以更好地控制软件的用户体验，不需要担心兼容性问题。可以用Web前端技术开发跨平台的桌面应用：使用纯JavaScript语言开发，只需要写一份代码，打包出来的应用可以同时在Windows、Linux、Mac操作系统上运行。可以从NodeJs的生态获得极大的助力：NodeJs这个大生态下很多成熟模块可以直接引入使用，避免重复造轮子，提高开发效率。进程隔离：基于Chromium多进程模式的应用模块集成，天然提供了应用模块之间的隔离性，其中某一应用模块的故障不影响其他应用模块及整个应用软件。   1. **实施方案和可行性分析**   **4.1 基于WASM构建FFmpeg.wasm**  **4.1.1 搭建编译环境**  构建WASM编码，首先要下载Emscripten工具链[21]、CMake高级语言编译工具，本课题采用C所以使用GCC或者Clang编译器以及python，python主要是用来充当编译过程的一些脚本功能。  (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4.1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；同样如果4.2所示在浏览器环境下打开控制台，也同样可以运行成功。    图4.1 Node执行结果    图4.2 浏览器执行结果   1. 斐波那契数列函数   针对上一小节的实验，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相比较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。  **4.1.2 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。  **4.1.3 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4.3所示，    图4.3 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”]’ \  除此之外，还可以增加getValue、setValue、writeAsciiToMemory等方法来丰富C程序与JavaScript程序的调用方法，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。  **4.2 基于Web Worker的模块文件加载方案**  **4.2.1 加载方案**  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[23]。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4.4所示：    图4.4 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。  **4.2.2 基于Electron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Electron跨平台框架下使用Web Worker。  基于Electron框架的前端开发在使用视频文件、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加载模块文件通过该插件一起打包，最终生成浏览器可识别的前端代码。  **4.3 跨平台音视频处理系统的开发与测试**  **4.3.1 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[24]、WASM、视频文件模块结合起来，设计如图4.5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4.5 跨平台音视频处理系统总体架构设计  **4.3.2 FFmpeg转换模块设计与实现**  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4.6所示    图4.6 FFmpeg加解码音视频流程  需要针对FFmpeg二次开发，设计开发实现封装模块对三类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FFmpeg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，即ES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编写程序，实现外层对音视频流数据的获取，具体逻辑流程图如图4.7所示    图4.7 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视频裁剪、合并；视频帧提取；音频、字幕融合三种功能。  **4.4 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、Electron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14个月的时间，具体工作分为文献整理、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、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. 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. 5. Vassallo Keith Garg Lalit Prakash Vijay Ramesh K.. Contemporary Technologies and Methods for Cross-Platform Application Development[J]. Journal of Computational and Theoretical Nanoscience,2019,16(9). 6. RAY VILLALOBOS. NW.JS VS ELECTRON[J]. Net,2016(Sep. TN.284):111. 7. 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). 8. Dave Rodriguez. Introduction to Audiovisual Transcoding, Editing, and Color Analysis with FFmpeg[J]. The Programming Historian,2018,7. 9. 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. 10. 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. 11. 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. 12. 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. 13. 闫锋欣,牛子杰,杜烁炜,潘天丽.基于Qt的Android应用程序C/C++开发方法与实践[J].计算机系用,2018,27(07):96-102.DOI:10.15888/j.cnki.csa.006423. 14. Robillard Martin P.,Kutschera Kaylee. Lessons Learned in Migrating From Swing to JavaFX[J]. IEEE Software,2019,37(3). 15. 朱丽英. 基于Node-Webkit平台的JavaScript工具集研究与实现[D]. 四川:电子科技大学,2016. DOI:10.7666/d.D00988796. 16. 褚孔统,朱勇. 开发跨平台桌面应用的探讨[J]. 机电信息,2019(33):55-56. DOI:10.3969/j.issn.1671-0797.2019.33.030. 17. Paul B. Jensen，著. 跨平台桌面应用开发:基于 Electron 与 NW.JS[M]. Goddy Zhao，译. 北京:电子工业出版社，2018 18. 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). 19. Paul Krill. WebAssembly may go live in browsers this year[J]. InfoWorld.com,2016. 20. 于航，著. 深入浅出WebAssembly[M]. 北京:电子工业出版社，2018 21. 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). 22. Jan Ozer. Six FFmpeg Commands You Can't Live Without[J]. Streaming Media Magazine,2019. 23. 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. 24. React. 2019. React - a javascript library for building user interfaces, https://reactjs.org/ 25. 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/> 26. 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. 27. 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. 28. 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. 29. 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. 30. 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  2022年 03月 12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 | 研究与实现FFmpeg的WASM编码 |  |
| 2022年7-9月 | 研究与实现二次开发模块和加载方案 |  |
| 2022年10-12月 | 基于已有模块开始跨平台系统设计与开发 |  |
| 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： | | |