## 基于FFmpeg的Web音视频处理系统的设计与实现

## 绪论

**1.1 研究背景与意义**

**1.2 研究现状**

Web音视频处理的研究现状

相关框架技术的研究现状

**1.3 论文的研究目标和内容**

本论文的主要工作为：

1. 研究学习音视频处理技术以及相关理论，分析Web音视频处理系统的需求，设计Web音视频处理系统的主要框架和满足需求的技术实现方案。
2. 研究学习FFmpeg音视频处理库以及WebAssembly二进制字节码编译技术。掌握FFmpeg处理音视频的核心流程和原理，并利用WebAssembly技术编译FFmpeg以实现Web浏览器中强兼容、高性能的音视频处理能力。
3. 设计实现基于FFmpeg的Web音视频编解码模块，再根据数据实时性消费划分为弱实时性音视频剪辑处理和强实时性音视频播放处理。针对弱实时性处理模块实现基于DAG拓扑图的多剪辑处理任务功能。针对强实时性处理模块实现多级缓存、同步控制、回退补偿的WebGL渲染视频图像数据和ContextAudio播放音频采样数据的实时同步功能。
4. 针对本文最终设计实现的Web音视频处理系统进行功能性测试和非功能性测试。功能性测试主要包括测试音视频编解码模块、弱实时性音视频剪辑功能以及强实时性音视频播放功能。非功能性测试主要包括系统的性能、代码质量等。

**1.4 论文组织结构**

## 二、相关技术

**2.1音视频技术**

**2.2 FFmpeg**

**2.3 WebAssembly**

**2.4 本章小结**

## 系统需求分析与总体设计

**3.1 基于FFmpeg的Web音视频处理系统需求分析**

基于FFmpeg的Web音视频处理系统的最终目标是在Web浏览器上下文环境中实现本地音视频多媒体文件的高效、强兼容的处理操作，因此本文提出的Web音视频处理系统需要支持在浏览器上下文环境中嵌入扩展的FFmpeg音视频编解码处理功能以及扩展兼容原生浏览器对多种音视频格式的识别与解析能力。

本文设计实现的Web音视频处理系统的功能性需求依次分为三个部分：

1、Web音视频编解码处理功能性需求。因为浏览器环境不能直接运行由C语言编写的FFmpeg音视频处理库，所以通过WebAssembly二进制字节码技术定制化编译FFmpeg为WASM编解码模块，再加载到浏览器中被调用执行，从而实现在Web浏览器中对本地音视频文件的解封装、解码、编码、封装的基础处理功能。

在Web音视频编解码功能基础之上，根据音视频处理在Web浏览器运行侧和WASM处理模块运行侧的数据转移实时性划分为弱实时性音视频剪辑处理功能和强实时性音视频播放处理功能。

2、弱实时性音视频剪辑处理功能性需求。在Web音视频编解码处理功能的基础之上，通过将剪辑操作抽象为多个处理单元，例如剪切处理单元、压缩处理单元。并通过构建DAG任务图的方式来灵活定制和拓扑执行多个音视频剪辑处理任务。

3、强实时性音视频播放处理功能性需求。相比较弱实时性处理，这部分功能要求WASM模块运行侧解码后的音视频数据需要实时传递到浏览器运行侧，在浏览器环境中将解码后的视频YUV图像数据和音频采样数据分别通过WebGL和AudioContext进行渲染和播放。同时本系统设计实现多级缓存策略和回退补偿策略来协调解决实时音频帧和视频帧的同步问题。

最后是Web音视频处理系统的非功能性需求，主要包括安全性、易用性、性能、可靠性以及可维护性五部分。

**3.1.1 系统功能性需求**

根据系统的具体设计目标，本小节主要分为三个部分描述系统的功能性需求：Web浏览器本地音视频编解码功能、弱实时性音视频剪辑处理功能以及强实时性音视频播放处理功能。

**1、Web音视频编解码功能**

Web音视频编解码功能是本系统中最基础的、也是架构设计最核心的部分，因为弱实时性处理模块和强实时性处理模块都是基于此功能做的进一步扩展。Web音视频编解码功能是通过移植FFmpeg功能模块到浏览器中再进一步扩展的。最重要的步骤是将基于FFmpeg扩展后的音视频处理程序定制化编译为WASM模块。因为WASM这种新型二进制字节码程序相比较JavaScript解释型高级抽象语言程序的执行速度要更加高效，所以通过实例化WASM处理模块并将实例传递给JavaScript运行侧来实现对音视频的高性能处理的调用是本文设计实现系统的基石。

Web音视频编解码模块通过利用两个Web Worker子线程来加载本地文件和初始化WASM处理模块。在文件Worker子线程中利用FileReader标准文件读取API流式读取本地音视频媒体数据，以浏览器主线程为数据传递中间层，将读取到的文件数据流式的传递给处理Worker子线程中的WASM模块实现对音视频数据的解封装、解码、编码、封装四个重要功能。最后因为文件读取、多媒体数据编解码处理都是在Web Worker子线程中处理的，这就需要设计实现主线程和子线程之间的协调通讯机制来完成用户在页面上的交互操作得到正确响应。

根据Web音视频编解码处理功能的需求，得到其需求用例图，如图3-1所示：

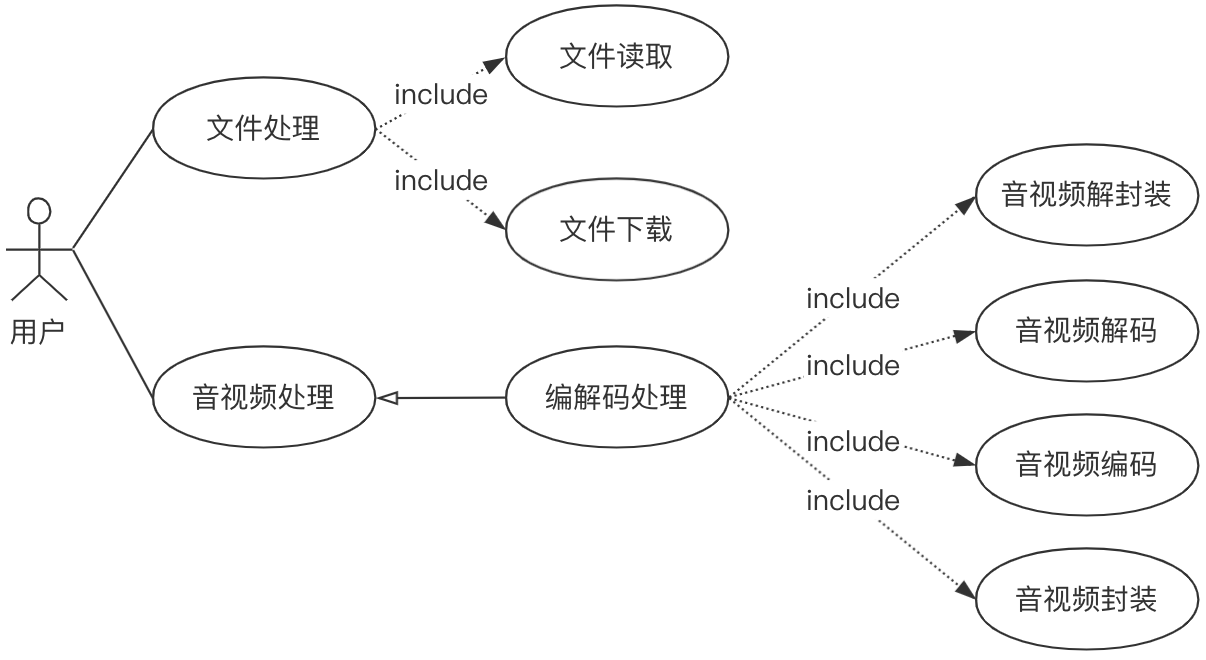


图3-1 Web音视频编解码处理功能需求用例图

用例名称:Web音视频编解码处理。

说明:在Web浏览器环境中，实现文件读取子线程将音视频文件数据流式地传递到音视频处理子线程中FFmpeg定制化编译后的WASM处理模块中，实现有效兼容解析多种音视频格式，以及包括音视频解封装、音视频解码、音视频编码、音视频封装四个重要处理功能。

参与者:Web音视频编解码模块。

前置条件:本地音视频文件读取成功。

后置条件:本地音视频文件可以被正确识别解析、解封装、解码、编码、封装。

**2、弱实时性剪辑处理功能**

基于上一小节Web浏览器上下文环境中对音视频文件的读取、解封装、编解码功能的基础上，实现弱实时性的音视频剪辑处理功能。弱实时性音视频处理功能的弱实时性主要体现在Web浏览器JavaScript运行侧与WASM处理模块运行侧的数据交互实时性不强。数据流向主要为单向流动，由JavaScript运行侧读取音视频原始数据，再将数据发送到WASM子线程模块中进行解析处理。在WASM模块解析处理音视频数据的同时，解码后的数据并不会实时地从WASM子线程中传递出去。直到WASM模块处理完用户定制的音视频剪辑任务后，才会将重新编码、封装的音视频数据从WASM子线程发送到Web浏览器JavaScript运行侧。最后用户就可以下载查看经过剪辑处理的音视频文件。

弱实时性剪辑处理模块将音视频剪辑功能抽象为五个基本的原子处理单元，分别是剪切处理单元、抽取处理单元、合并处理单元、压缩处理单元和转换处理单元。以及两个辅助处理节点分别是输入处理单元和输出处理单元，输入处理单元是模块根据本地读取文件数量一一对应生成的，而输出处理单元用于标识一个剪辑处理任务的结束。最后用户可以通过拖拽成功注册的处理单元组件来构建DAG多剪辑任务流程图。本小节介绍的剪辑处理模块再经过以下三个步骤来执行组合的多个剪辑任务：

1、以输入处理单元为无入度根节点，检测多剪辑任务图是否为多个有向无环图。

2、初始化系统任务执行器，并通过拓扑排序算法并行执行多个剪辑任务的各个处理单元。

3、以输出处理单元为无出度结束节点，提供下载功能。

弱实时性音视频剪辑处理模块整体功能流程上主要是用户先通过定制化拖拽抽象的处理单元来构建多DAG剪辑任务图，通过多个处理单元的组合来实现一个完整的音视频剪辑任务。最后利用拓扑排序算法让模块任务执行器来处理音视频剪辑需求。

根据弱实时性音视频剪辑处理功能的需求，得到其需求用例图，如图3-2所示：

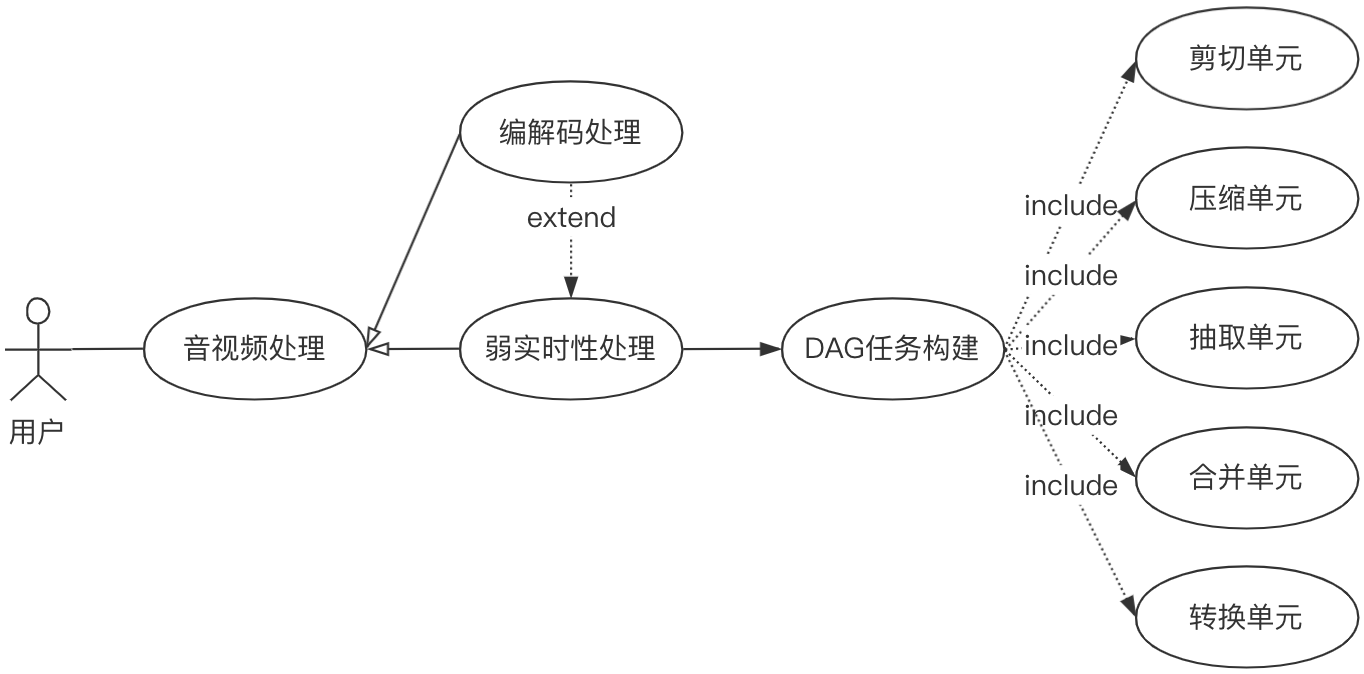


图3-2 弱实时性剪辑处理模块需求用例图

用例名称:弱实时性剪辑处理。

说明:基于Web音视频编解码模块，通过组合抽象的剪辑处理单元和基本输入输出处理单元来构建多DAG剪辑任务图，实现多音视频剪辑处理功能。剪辑处理单元抽象为剪切处理单元、压缩处理单元、抽取处理单元、合并处理单元以及转换处理单元。

参与者:Web音视频编解码模块、弱实时性剪辑处理模块。

前置条件:本地音视频文件成功读取、解析并成功解码。

后置条件:剪辑处理后的音视频文件可以下载查看，并符合预期处理效果。

1. **强实时性播放处理功能**

强实时性播放处理模块同样是基于Web音视频编解码功能进行扩展。与上一小节弱实时性音视频剪辑处理不同在于，强实时性音视频播放处理主要表现为WASM处理模块运行侧会实时的将解码后的每一帧音频数据和视频数据通过Web Worker消息数据机制传递给Web浏览器JavaScript运行侧，最后在浏览器中利用WebGL将视频YUV图像数据进行渲染播放以及利用ContextAudio对音频采样数据进行同步播放。

强实时性播放处理模块主要分为简单顺序播放、高级播放，高级播放包括倍速播放、全屏播放以及Seek播放。在播放处理过程中，需要考虑协调音频帧和视频帧的同步问题，确保在任何播放状态下音频帧和视频帧都会同步被消费，也就是通常所说的音画同步。在播放处理模块中设计音频帧和视频帧的交替存储数据结构，以及设计实现多级缓存机制来确保音频帧和视频帧的同步。同时针对出现音画不同步的问题，强实时性播放处理模块需要一定的回退补偿和丢弃补偿策略来修正音画不同步的情况。

根据强实时性音视频播放处理功能模块的需求，得到其需求用例图，如图3-3所示：

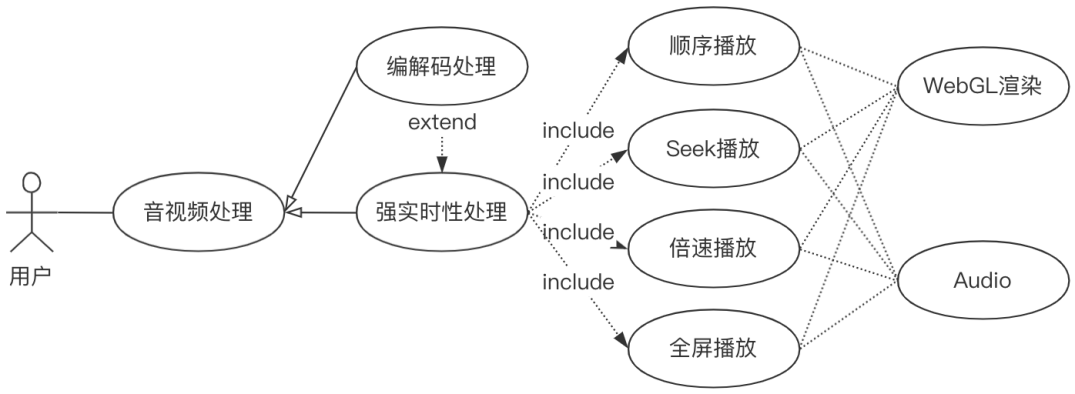


图3-3 强实时性处理模块需求用例图

用例名称:强实时性播放处理。

说明:基于Web音视频编解码模块的，通过将web编解码模块处理后的解码音视频数据从WASM运行侧转移给JavaScript运行侧，然后通过WebGL来进行yuv图像的绘制、Audio播放音频数据。强实时性播放处理模块包含顺序播放、倍速播放、全屏播放以及Seek播放。

参与者:web音视频编解码模块、强实时性播放处理模块。

前置条件:本地音视频文件成功读取、解析并成功解码。

后置条件:解码后的音视频数据可以实时地从WASM处理模块子线程转移至Web浏览器JavaScript主线程中，并成功地同步对视频帧YUV图像数据渲染和对音频帧数据播放。

**3.1.2 系统非功能性需求**

Web音视频处理系统的非功能需求主要体现除了功能性需求以外的其他特性，包括系统的安全性、易用性、性能、可靠性以及可维护性需求，具体说明如下：

1. 安全性

系统的安全性主要体现在两个方面，分别是编码安全和用户内容安全两部分。编码安全指的是系统自身的代码需要经过严格的测试，尽量对编码过程中容易出现的内存泄漏、数据为空进行校验与捕获。用户内容安全是指系统在处理用户本地音视频数据时，保证可见性原则和不侵入原则，安全有效的处理用户指定的音视频文件。通过以上两个方面保障系统自身的安全性。

1. 易用性

Web音视频处理系统借助了B/S架构的易用优势，无需下载其他软件实现音视频在线处理功能。系统面对不同用户群体，尤其是针对不具备音视频相关专业知识基础的用户在使用过程中，提供对应的默认参数以及相关参数的友好提示来帮助用户更好的使用平台功能。同时要做到系统页面逻辑清晰、功能操作简单方便等。

1. 性能

性能需求主要指在Web浏览器中提高音视频处理性能。因为Web浏览器的音视频处理过程是非常消耗系统性能，因为Web浏览器并不适合处理CPU密集型计算任务，并且Web浏览器作为应用软件对操作系统做了封装，导致不能直接利用操作系统的资源。如果可以通过提高音视频处理在Web中的性能，可以缓解服务端压力，更好的利用客户端的计算资源。

1. 可靠性

系统可靠性需求指的是使用该Web音视频处理系统处理音视频时，需要严格按照使用者定义的规则进行处理。同时系统自身的稳定性需要控制在极小的波动范围内，不会对使用者的浏览器应用产生过大的内存负担导致页面崩溃等问题。

5. 可维护性

系统可维护性需求主要指系统在后期的更新维护过程中，能够根据新的业务场景对原有系统进行扩展。同时系统本身运行过程中，通过埋点收集关键的日志信息，方便根据日志信息排查系统出现的问题并解决。同时在系统功能的设计实现上，尽量运用六大软件设计原则，做到系统中各个模块高内聚、低耦合的关系，提高系统的可维护性。

**3.2 系统总体设计**

根据上一节对系统的功能性需求和非功能性需求为基础，以需求分析为设计目标，本节主要是对本文提出的高性能、强兼容性的Web音视频处理系统进行总体设计，具体包括系统架构设计和系统功能设计。

**3.2.1 系统架构设计**

因为Web浏览器JavaScript运行侧对音视频直接编解码处理的低性能，以及无法直接在Web浏览器中运行C语言程序的局限性。所以本小节介绍通过引入WebAssembly二进制字节码技术，将FFmpeg音视频库编译为WASM音视频处理模块并通过Web Worker线程加载到Web浏览器中调用来解决低性能、局限性的问题。

基于FFmpeg和WebAssembly的Web音视频处理系统的整体架构主要分为四层：音视频渲染交互层、音视频数据控制层、Web Worker线程加载层以及音视频处理层。其中音视频处理层是系统核心，是Web音视频处理的性能保证。Web Worker线程加载层主要通过子线程加载本地音视频文件和WASM音视频处理模块，使本地音视频文件的读取和音视频的复杂处理两种耗时操作在子线程中执行而不会阻塞主线程的运行来提高系统的整体性能。最后设计实现主线程和子线程之间的消息通讯机制来实现各个线程之间的交互调用逻辑协作完成音视频处理任务，通过这种方式可以解耦模块和模块之间的关系，提高了系统的可维护性和稳定性。音视频数据控制层主要通过连接渲染交互层和Worker线程加载层，将渲染交互层的操作逻辑指令解释为Worker线程加载层提供的消息指令组合，再调用下层的核心处理功能。渲染交互层是使用者直接操控和查看的交互页面的统称，主要用来抽象音视频处理任务的定制化过程以及处理结果的渲染播放。

因此Web音视频处理系统的整体架构设计如图3-4所示：

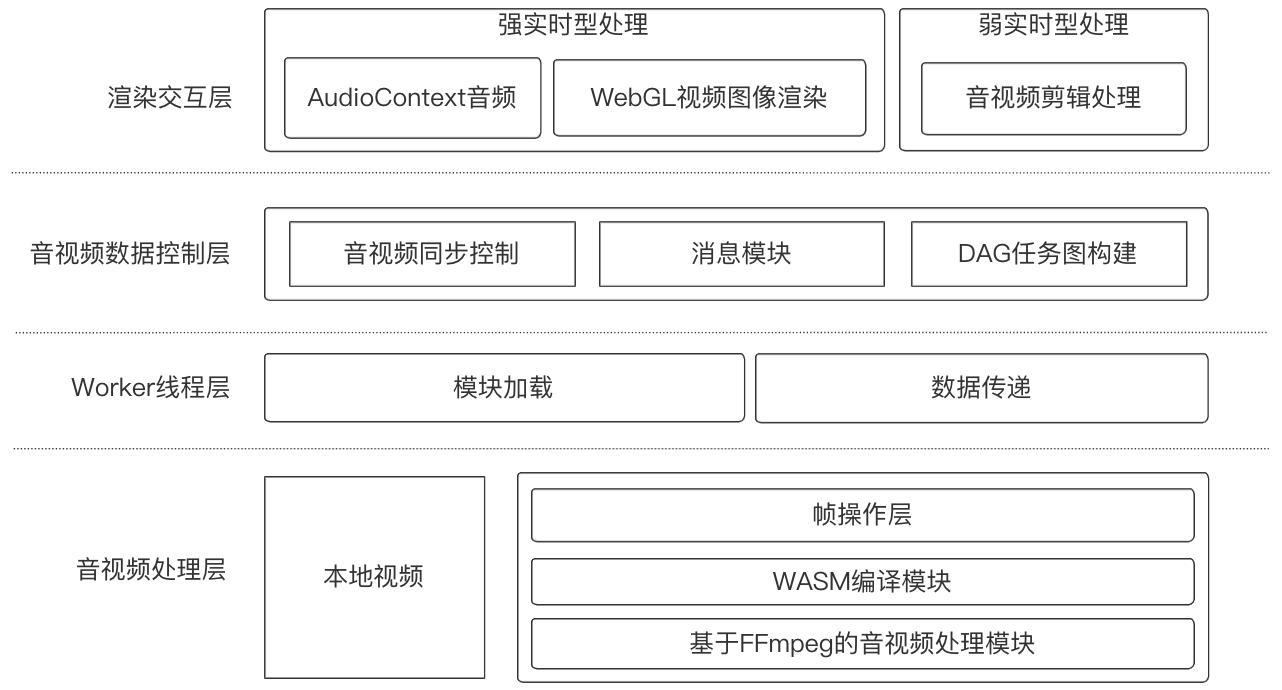


图3-4 音视频处理系统总体架构图

在Web音视频处理系统中，弱实时性音视频剪辑处理模块和强实时性音视频播放处理模块的前端都基于Web音视频编解码模块。根据不同的音视频处理需求，Web音视频编解码模块将解码后数据发送给具体的模块来进一步处理。如图3-5展示了Web音视频处理系统针对不同功能需求的具体工作流程。

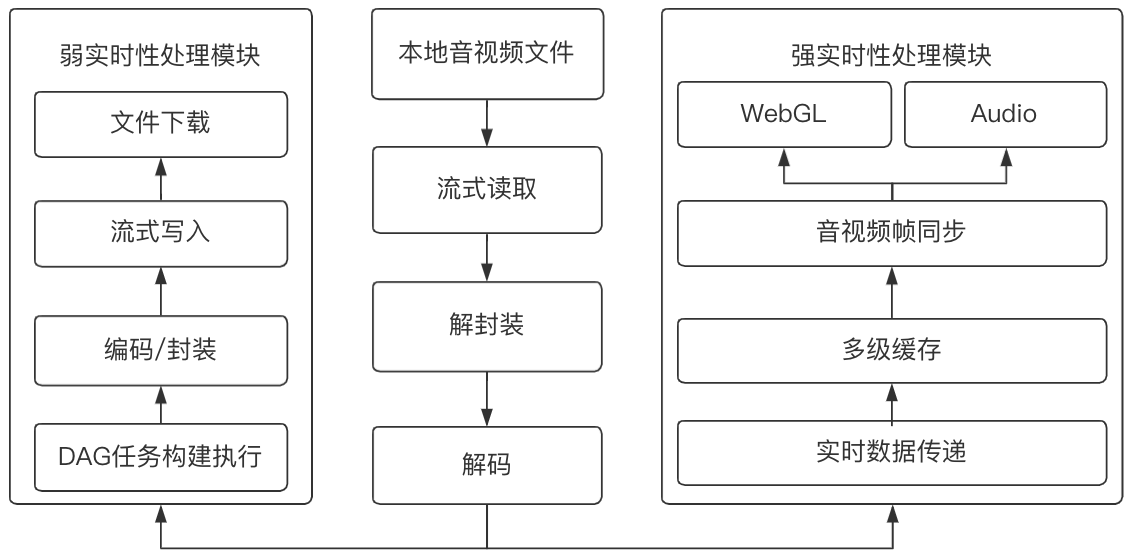


图3-5 系统整体数据流图

首先Web音视频编解码模块第一步是读取本地音视频文件数据，再根据音视频文件的头信息加载对应的解封装器、解码器来保证正确地解码不同格式的音视频，最后将解码后的数据发送给弱实时性处理模块或者强实时性处理模块来完成剪辑功能和播放功能。

弱实时性处理模块根据DAG剪辑任务图自定义的各个节点处理单元来拓扑执行解码后的音视频数据，将处理完成的音视频数据通过参数指定的编码器和封装器来生成新的音视频文件，最后流式写入本地文件实现文件下载。强实时性处理模块则需要实时地获取到解码后的音频帧和视频帧数据，通过缓存模块和同步控制模块的处理，进一步把帧数据发送给Web浏览器主线程的WebGL模块和Audio模块来消费。

**3.2.2 系统功能设计**

根据Web音视频处理系统各相关功能之间的关系，将系统整体功能划分为文件模块和音视频处理模块，音视频处理模块又分为Web编解码模块、弱实时性剪辑处理模块和强实时性播放处理模块三个模块。

最终系统整体功能划分如图3-6所示：

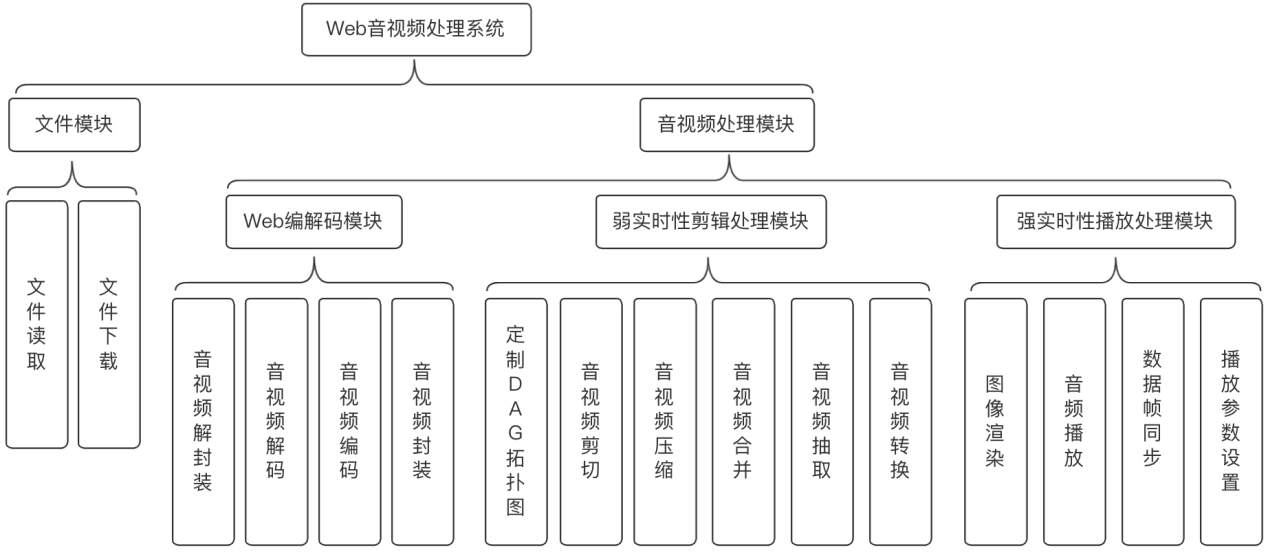


图3-6 音视频处理系统整体功能图

文件模块主要作用是根据音视频处理模块的指令来控制流式读取文件的开始与停止，并提供处理完成后生成的新音视频文件的下载功能。音视频处理模块中Web编解码模块是核心基础模块，主要负责音视频数据的解封装、解码、编码、封装功能。弱实时性剪辑处理模块主要功能是通过DAG拓扑图构建并执行组合的音视频剪辑处理单元，包含剪切、压缩、合并、抽取与转换。最后是强实时性处理模块是基于WebGL渲染引擎、Audio引擎以及数据帧同步策略实现音视频的播放功能，再通过播放参数设置功能可以把播放细分为简单顺序播放、倍速播放以及Seek播放。

**3.4 本章小结**

本章第一部分内容根据Web音视频处理系统的最终目标逐次对三个功能性需求Web音视频编解码、弱实时性剪辑处理、强实时性播放处理进行需求分析，并对系统非功能性提出了五点要求，包括系统的安全性、易用性、性能、可靠性以及可维护性。其次本章第二部分内容是以需求分析为设计目标进行系统总体设计，包括系统架构设计和系统功能设计。

## 系统详细设计与实现

本章

## 4.1 Web编解码模块的设计与实现

**4.1.1 FFmpeg编译移植**

FFmpeg是一个开源免费的、跨平台的、集音视频编解码、转换于一体的基础C程序库，所以在Web浏览器上下文环境中不能直接调用FFmpeg提供的相关功能。因此本文设计实现的系统通过引入WebAssembly二进制字节码编译技术，首先将基于FFmpeg扩展的处理程序编译为WASM音视频处理模块，再通过Web Worker加载使用来提高Web浏览器对CPU密集型计算的运行速度。

因为FFmpeg源代码由多个基础核心模块和上层工具模块共同组成，其中一些模块对于本文Web音视频处理系统是非必需的，所以针对FFmpeg采取定制化裁剪，只选取核心模块libavformat、libavcodec、libavutil等，再通过链接自定义扩展的音视频处理程序进行编译。通过这种方式不但可以减少编译后的产物大小，而且提高了编译时速度、编译后模块的加载速度以及系统整体程序的大小。

在MacOS操作系统下实现FFmpeg的定制化编译需要借助相关的编译工具链，FFmpeg的编译依赖于pkg-config，而它又依赖于GLib，而GLib又依赖于gettext，所以必须先安装并编译相关依赖。因此具体编译步骤如下：

1. 下载FFmpeg源码以及fdk-aac、x264、x265等音视频编解码器，确保在编译过程是可以找到这些编解码器进行链接。
2. 下载安装emscripten、emmake，用于替换gcc编译器来编译C程序。
3. 编译FFmpeg: 切换到FFmpeg源码目录下，执行编译配置命令

在编译FFmpeg时，通过不同的参数选项组合来实现具体的定制化编译方案。FFmpeg提供的参数选项作用一般分为以下几个方面：基本参数包含编译输入输出目录、静态库目录动态库目录等设置；开闭参数主要作用是开启或者关闭相关模块或者编译方法；高级参数用来指定编译工具、编译宿主系统的CPU、架构信息以及一些编译时内存大小分配等等参数。部分编译时配置的优化参数如表4-1所示：

表4-1 FFmpeg编译配置参数说明

|  |  |
| --- | --- |
| 配置参数 | 参数作用 |
| --disable-ffplay  --disable-ffprobe  --disable-ffmpeg  --disable-doc  --disable-small  --disable-programs  --disable-network  --enable-cross-compile  –enable-encoder=NAME  –enable-decoder=NAME  –enable-muxer=NAME  –enable-demuxer=NAME | 禁止编译ffplay播放器模块  禁止编译ffprobe嗅探器模块  禁止编译ffmpeg命令工具模块  禁止生成doc文件  优化减少FFmpeg的体积  禁止生成可执行文件  禁止编译网络模块  启动交叉编译  启用NAME编码器  启用NAME解码器  启用NAME复用器  启用NAME解复用器 |

在根据不同配置参数功能的分析后，最后确定如下的编译配置参数。

emconfigure ./configure --cc="emcc" --cxx="em++" --ar="emar" --ranlib="emranlib" --prefix=$(pwd)/dist --enable-cross-compile --target-os=none --arch=x86\_32 \ --cpu=generic --enable-gpl --enable-version3 --disable-avdevice \ --disable-swresample --disable-postproc --disable-avfilter --disable-programs \ --disable-logging --disable-everything --enable-avformat --enable-decoder=hevc \ --enable-decoder=h264 --enable-decoder=aac --disable-ffplay --disable-ffprobe \ --disable-ffserver --disable-asm --disable-doc --disable-devices --disable-network \ --disable-hwaccels --disable-parsers --disable-bsfs --disable-debug \ --enable-protocol=file --enable-demuxer=mov --enable-demuxer=flv --disable-indevs \ --disable-outdevs

开启交叉编译目的可以在MacOS操作系统上编译出能运行在体系结构不同的其他操作系统上。同时禁止目标构建系统中不需要的模块，最后增加了多种音视频的复用器、解复用器以及编解码器来提高Web音视频处理的兼容性。

**4.1.2 Web编解码模块流程设计**

Web编解码功能模块依赖于Web Worker多线程来加载本地音视频文件和编解码音视频的WASM模块，从而避免Web浏览器JavaScript主线程、UI渲染线程的阻塞。以主线程为中间层，控制UI交互指令消息发送给子线程并协调文件读取子线程和WASM音视频处理子线程的工作。完整的编解码模块线程协作关系如图4-1所示，主线程作为主控模块分别与文件读取子线程和音视频处理子线程通过消息、数据进行协作交互。子线程之间不能直接通讯，必须要通过主线程过渡。之后会根据音视频处理子线程解码后的数据是否实时发送到主线程进行渲染消费来划分为弱实时性剪辑处理和强实时性播放处理。

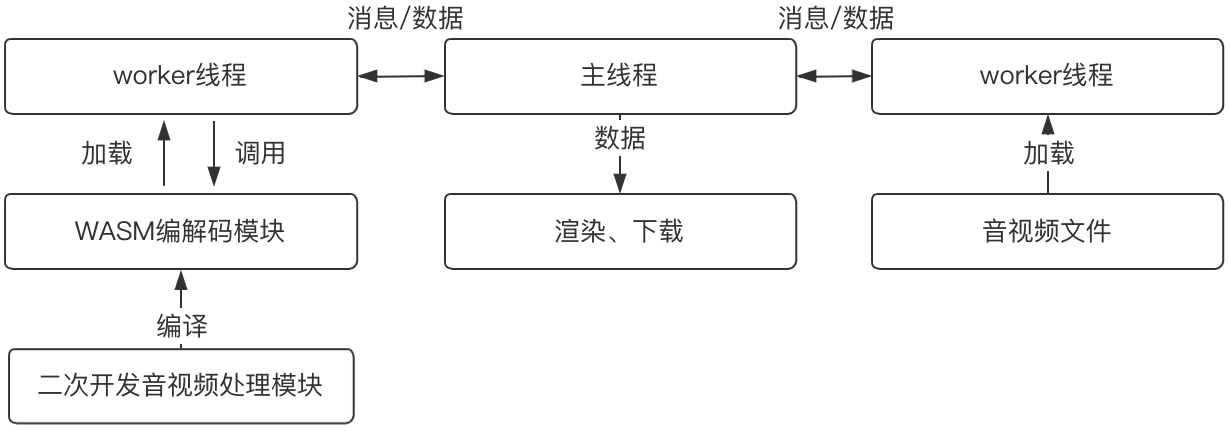


图4-1 Web编解码模块线程关系图

Web音视频解码模块基于多线程的并发协作模式完成音视频的解封装、解码、编码、封装流程。首先主线程通过UI交互控制文件子线程加载本地音视频文件，如果加载成功则会预读取音视频文件头信息，一般地通过头信息可以获取到音视频文件的封装格式、多媒体数据的压缩标准和规范信息，通过这些信息可以帮助音视频编解码模块选择合适的解复用器以及编解码器来进行处理。如果正确识别到支持的音视频格式，就可以通过解复用器对音视频进行解封装后获取到独立的音频流和视频流。如果存在音视频没有音频流意味着音频流为空。再通过编解码器就可以对音频或者视频数据进行解码和编码。整体Web音视频编解码流程设计如图4-2所示：

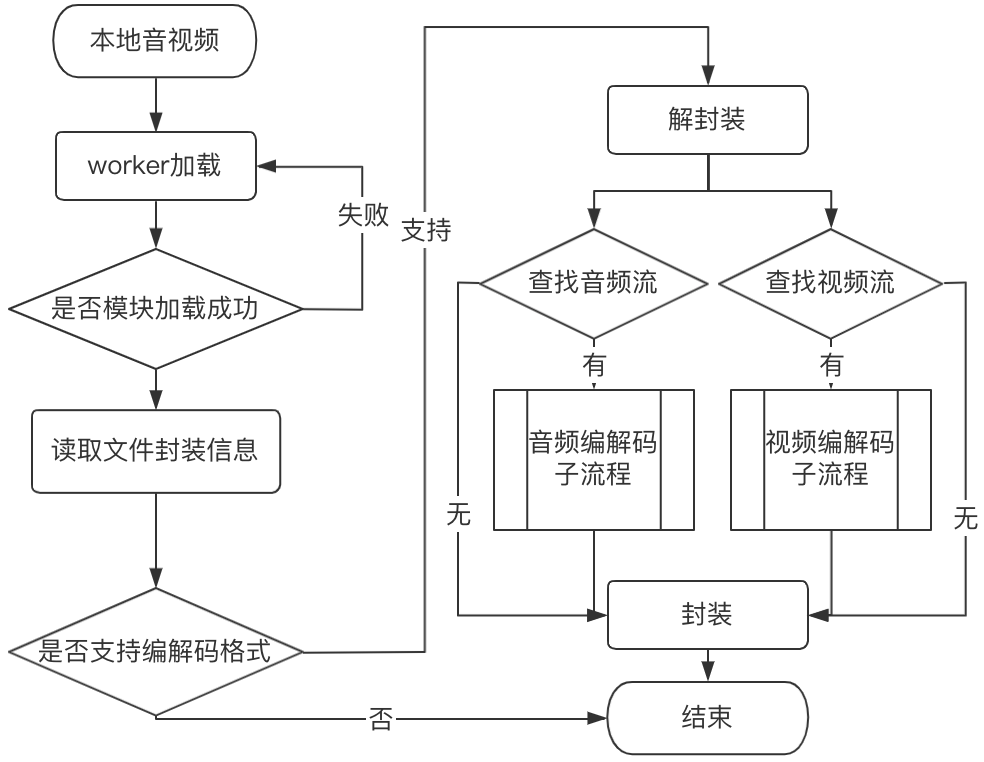
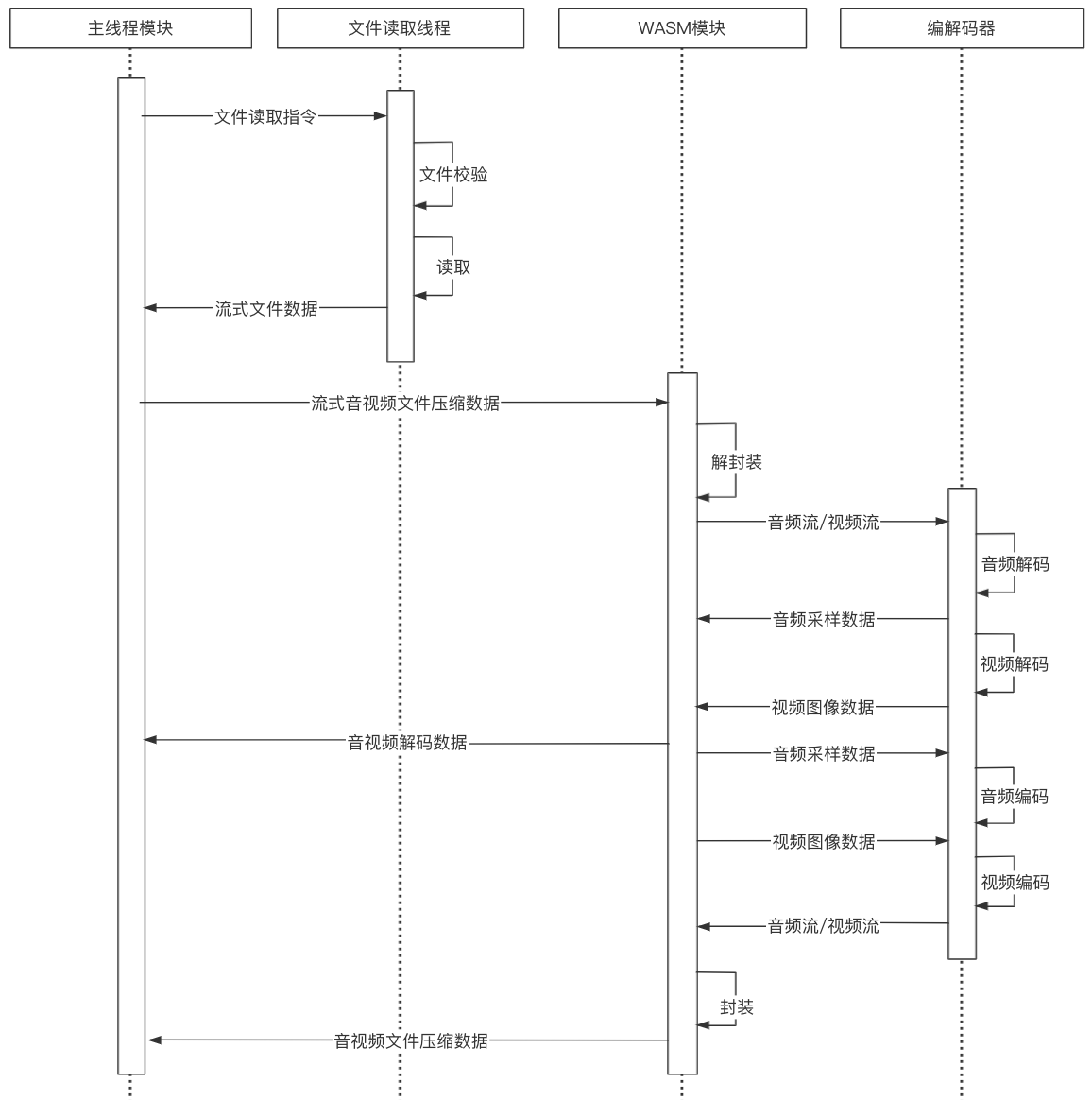


图4-2 Web编解码流程

**4.1.3 Web编解码模块实现**



## 4.2弱实时性处理模块的设计与实现

本章主要介绍弱实时性音视频处理模块的详细设计与实现。

**4.2.1 抽象剪辑处理单元**

4.1.1 抽象处理单元

在弱实时性处理中，将音视频处理功能抽象设计为多个剪辑元件，共包含Compress处理单元、Transform处理单元、Input处理单元、Output处理单元，最后通过构建DAG图来组合多个元件来实现多任务处理的构建。

首先在系统内部注册相关元件，完成元件注册之后，就可以针对音视频实例或者中间实例进行创建和编辑相关参数。

1. Input处理单元

Input处理单元主要是有选择的初始音视频决定，每一个初始化的音视频都可以作为一个Input处理单元。其中包含每一个音视频的详细信息，包括音视频大小、格式、类型等等。

1. Compress处理单元

Compress处理单元主要用来对音视频进行压缩。

1. Extract处理单元
2. Merge处理单元

Merge处理单元用于两个及两个以上音频、视频的合并。

1. Cut处理单元

Cut处理单元对音频或视频进行裁剪操作，以时间为维度进行分割操作。

1. Transform处理单元

Transform处理单元主要用于对音频或视频格式、编解码协议等进行转换。

1. Output处理单元

Output处理单元主要作用是指定我们最终输出的音视频数量，因为每个阶段的操作都可以作为音视频输出。

**4.2.2 基于DAG的多剪辑任务处理实现**

基于DAG构建音视频的处理任务，可以帮助我们流程化的构建有向无环图。

4.2.1 构建DAG任务图

基于React-flow-render库构建Web拖拽层，用户可以通过预设的处理单元进行拖拽添加。

4.2.2 Task调度器

## 4.3强实时性处理模块的设计与实现

本章主要介绍关于在Web音视频处理系统中，强实时性播放处理模块的设计与实现。

**4.3.1 音视频速率同步控制模块的设计与实现**

多级缓存

交替数组

**4.3.2 简单播放的设计与实现**

**4.3.1 高级播放的设计与实现**

倍速播放

Seek播放

## 系统测试与分析

## 总结与展望

## 致谢

## 参考文献