播放器技术方案

1. 版本修订历史

1.1. 版本历史

版本	修订日期	修订描述	修改人
v1.	2024-08-09	第一版	彭金龙

1.2. 责任人职责

人员	职责
负责人	1、负责技术方案的完整性、一致性、合理性。 2、方案发生变更及时同步的第一责任人 2、负责本模块的整体联调、上线等。 3、作为本模块线上问题的第一接口人。

1.3. 责任人确定

模块	负责人
整体	彭金龙

2. 整体介绍

2.1. 业务背景

实习demo开发播放器，作为考核使用，对于播放器来说，简洁、直观的用户界面非常重要。用户希望能够方便地进行播放、暂停等功能

2.2. 需求详情&设计详情

设计一款播放器程序，实现播放MP4等视频格式文件播放，确保在播放过程中音视频能够保持同步，避免卡顿或不同步的问题，实现播放、暂停、停止，通过文件选择对话框让用户选择要播放的视频文件

2.3. 技术名词解释

名词	解释
FFmpeg	开源的多媒体框架，能够解码、编码、转码、mux、demux、流式传输、过滤和播放几乎所有的音视频格式。它提供了强大的命令行工具和库，用于处理多媒体数据
SDL	提供了对图形、声音、输入设备等底层功能的访问。SDL 旨在为游戏开发和其他需要低级别硬件访问的应用程序提供简化的接口。SDL 提供了跨平台的支持，可以在 Windows、macOS、Linux、iOS 和 Android 等多个操作系统上运行。

3. 需求分析

3.1. 需求拆解

将播放器需求拆解成可以通过研发实现的子模块，可以考虑以下模块：

文件加载模块

功能：选择和加载视频文件。

实现：使用 Qt 的 `QFileDialog` 选择文件，并将文件路径传递给播放器。

初始化模块

功能：初始化 SDL 和 FFmpeg，设置播放器的基本环境。

实现：`InitializePlayer` 方法进行 SDL 和 FFmpeg 的初始化，包括创建窗口、渲染器和解码器。

解码模块

功能：解码视频和音频数据。

实现：`DecodeThread` 方法解码视频和音频数据，并将解码后的帧存入队列。

视频播放模块

功能：将解码后的视频帧渲染到屏幕上。

实现：`VideoPlayThread` 方法从队列中取出视频帧并更新 SDL 纹理。

音频播放模块

功能：播放解码后的音频数据。

实现：`AudioPlayThread` 方法使用 SDL 音频回调函数播放音频数据。

暂停和恢复模块

功能：控制视频和音频的播放/暂停状态。

实现：`TogglePause` 方法切换播放和暂停状态，并相应调整音频和视频的播放。

清理模块

功能：释放资源并停止所有线程。

实现：`StopPlayer` 方法清理资源，停止解码、视频播放和音频播放线程。

3.2. 技术难点

音视频同步

技术难点：确保音频和视频保持同步，即视频播放与音频播放的时 间一致。可能需要精确计算播放时间和处理延迟。

解决方案：使用时间戳和同步机制来调整播放时间和延迟。

多线程处理

技术难点：正确管理多线程之间的通信和同步，避免竞争条件和死锁。

解决方案：使用互斥锁（`std::mutex`）和条件变量（`std::condition_variable`）来实现线程间的协调。

性能优化

技术难点：处理高分辨率视频和高比特率音频时，保持播放器的流畅性和响应速度。

解决方案：优化解码和渲染代码，减少不必要的计算和内存操作。

动态资源管理

技术难点：动态调整 SDL 纹理大小和格式，处理不同分辨率的视频流。

解决方案：在 `VideoPlayThread` 中动态创建和销毁 SDL 纹理，以适应不同的视频帧尺寸。

错误处理

技术难点：处理 FFmpeg 和 SDL 中可能发生的各种错误，如文件无法打开、解码失败等。

解决方案：添加错误检查和处理代码，确保播放器在出现错误时能够安全退出或恢复正常。

3.3. 兼容性

向前兼容性:

需要考虑不同版本的操作系统和不同的硬件环境，确保播放器在较旧的版本上也能正常工作。

对于 FFmpeg 和 SDL 的版本，确保所使用的 API 兼容，并测试在多个版本上进行兼容性测试。

向后兼容性:

考虑不同的音视频格式和编码方式，确保播放器能够支持旧版视频文件格式。提供一定的版本适配层，处理不同的文件格式和编码方式。

3.4. 性能

解码性能

确保使用的解码器和编码器能够处理高分辨率和高比特率的视频流，避免解码瓶颈。

渲染性能

使用高效的渲染技术和优化的 SDL 绘制代码，避免由于图形渲染导致的性能问题。

内存管理

确保及时释放不再使用的资源，避免内存泄漏。特别是在动态创建和销毁 SDL 纹理时，要小心内存管理。

线程管理

合理分配线程的优先级和资源，确保解码、播放和用户交互的平衡，避免线程间的竞争和冲突。

4. 技术方案选型

4.1. 方案一

方案概要设计：

- **音视频解码**:

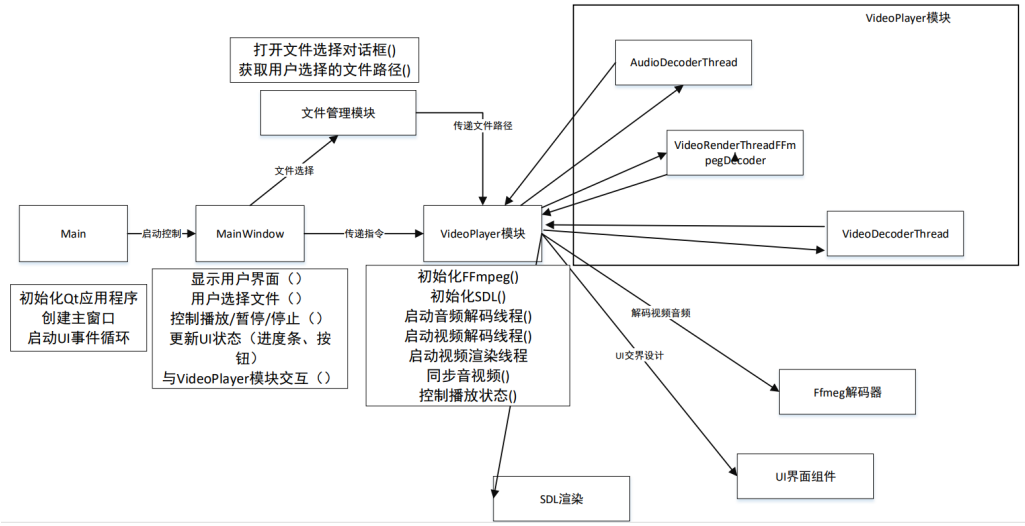
    - 使用 FFmpeg 作为音视频解码库。FFmpeg 提供了广泛的编解码支持，可以处理多种格式的音视频数据。
- **音视频播放**：
    - 使用 SDL 作为音视频渲染库。SDL 提供了跨平台的图形和音频处理能力。
    - 创建 SDL 窗口和渲染器，用于显示视频画面。
    - 配置 SDL 音频回调函数，用于播放音频数据。
- **线程管理**：
    - 使用多个线程分别处理音视频解码、视频播放和音频播放。线程间通过条件变量和互斥锁进行同步。
    - 主要线程包括：
        - 解码线程：解码音视频数据并将解码后的帧存入队列。
        - 视频播放线程：从队列中取出视频帧并更新 SDL 纹理。
        - 音频播放线程：使用 SDL 音频回调播放音频数据。
- **暂停和恢复**：
    - 通过 `TogglePause` 方法切换播放器的播放和暂停状态。调整音视频的播放状态，以同步播放和暂停。
- **资源管理**：
    - 在播放器停止时，释放所有动态分配的资源，包括 SDL 窗口、纹理、解码器和 FFmpeg 的上下文。

选择方案一的依据包括：

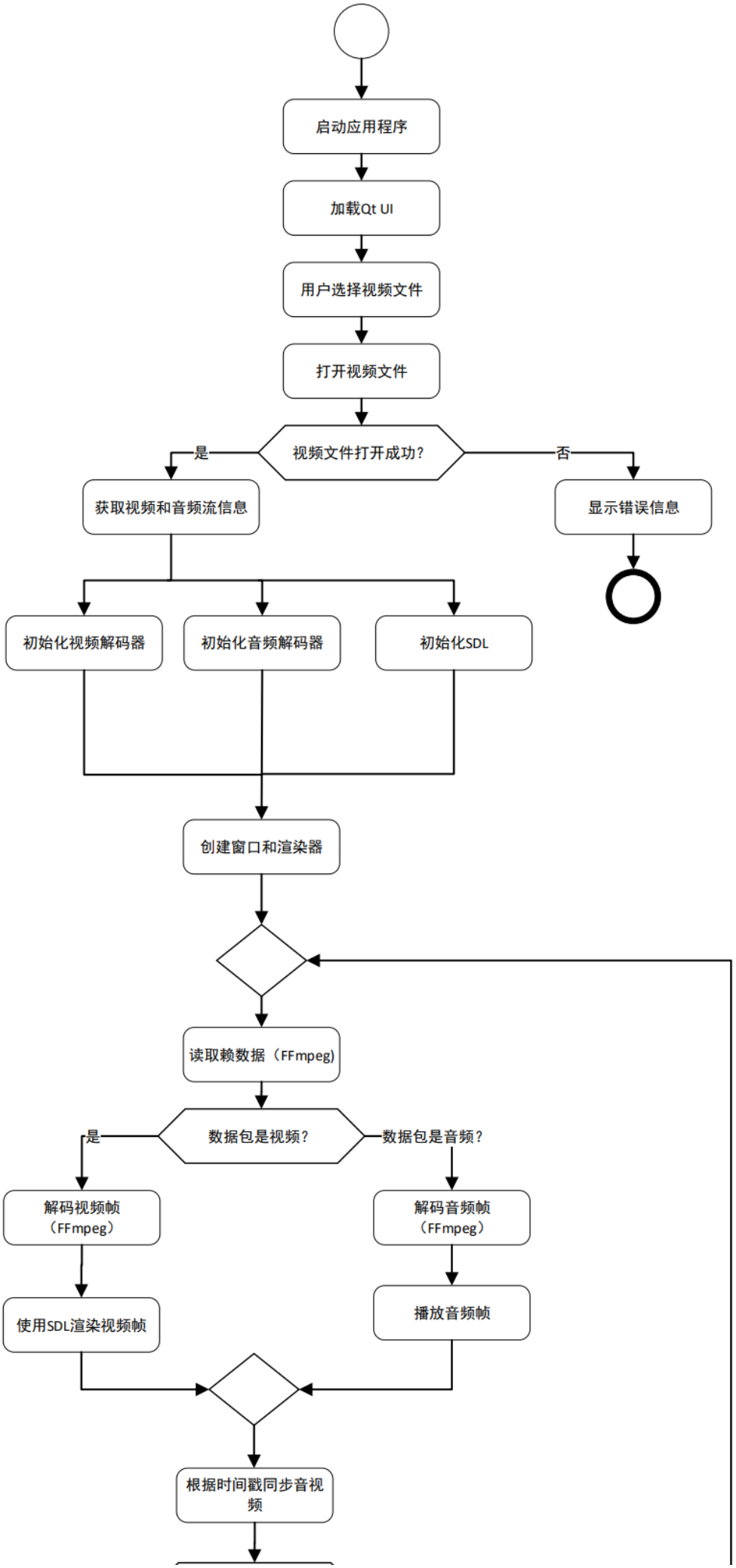
- **功能需求**：FFmpeg 和 SDL 能够满足项目的所有功能需求，包括音视频解码、渲染、播放控制等。
- **技术成熟度**：FFmpeg 和 SDL 都是成熟的开源库，拥有广泛的社区支持和文档资源，减少了开发中的不确定性。
- **跨平台兼容性**：SDL 的跨平台支持使得播放器可以在多种操作系统上运行，适应不同的用户环境。
- **性能要求**：尽管方案一对性能有一定要求，但通过优化和合理的线程管理，可以满足高性能播放的需求。

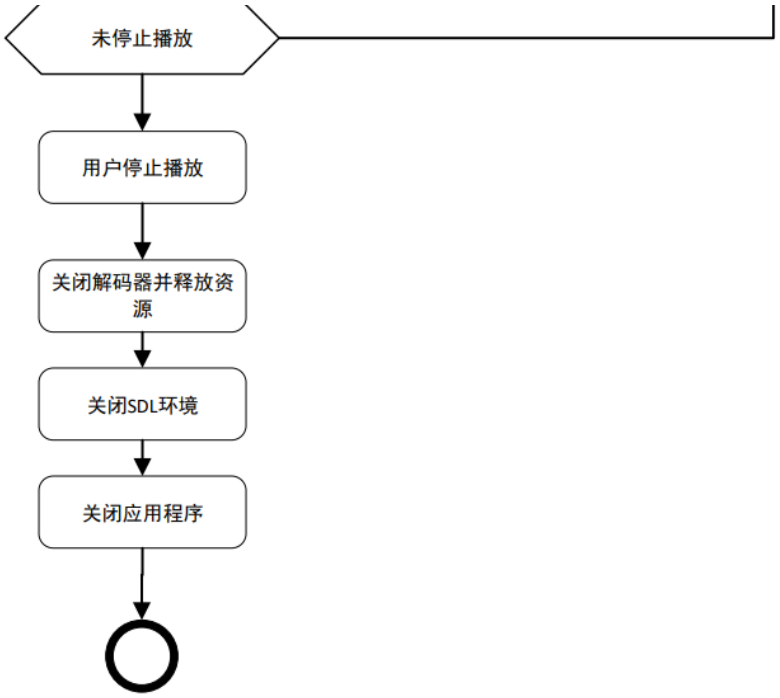
5. 系统整体架构

5.1. 整体架构图


VideoPlayer模块
打开文件选择对话框()
获取用户选择的文件路径()
AudioDecoderThread
文件管理模块
传递文件路径
VideoRenderThreadFFmpegDecoder
文件选择
Main
启动控制
MainWindow
传递指令
VideoPlayer模块
VideoDecoderThread
初始化Qt应用程序
创建主窗口
启动UI事件循环
显示用户界面（）
用户选择文件（）
控制播放/暂停/停止（）
更新UI状态（进度条、按钮）
与VideoPlayer模块交互（）
初始化FFmpeg()
初始化SDL()
启动音频解码线程()
启动视频解码线程()
启动视频渲染线程
同步音视频()
控制播放状态()
解码视频音频
UI交界设计
Ffmeg解码器
UI界面组件
SDL渲染

5.2. 模块时序图


启动应用程序
加载Qt UI
用户选择视频文件
打开视频文件
视频文件打开成功?
是
否
获取视频和音频流信息
显示错误信息
初始化视频解码器
初始化音频解码器
初始化SDL
创建窗口和渲染器
读取赖数据（FFmpeg)
数据包是视频?
是
数据包是音频?
解码视频帧（FFmpeg）
解码音频帧（FFmpeg）
使用SDL渲染视频帧
播放音频帧
根据时间戳同步音视频



6. 详细设计–客户端

6.1. 重点逻辑&算法

逻辑描述：

6.1.1.音视频解码

- **输入**：媒体文件（如 MP4、AVI、MKV）。
- **步骤**：
  - **打开文件**：使用 `avformat_open_input` 打开媒体文件并初始化 `AVFormatContext`。
  - **查找流信息**：使用 `avformat_find_stream_info` 查找音视频流信息。
  - **初始化解码器**：根据流信息，使用 `avcodec_find_decoder` 查找解码器，使用 `avcodec_open2` 打开解码器。
  - **读取数据包**：使用 `av_read_frame` 从媒体文件中读取数据包（`AVPacket`）。
  - **解码数据包**：将数据包发送给解码器（`avcodec_send_packet`），并接收解码后的帧（`avcodec_receive_frame`）。

算法细节：

- **时间基准转换**：视频帧的时间戳（PTS）需要根据流的时间基准进行转换，以计算实际播放时间。
- **延迟计算**：通过计算帧的预期播放时间与当前时间的差值，确定需要的播放延迟时间。

6.1.2. 音视频播放

逻辑描述：

- **视频播放**：
  - **更新纹理**：将解码后的视频帧数据更新到 SDL 纹理中。
  - **渲染帧**：使用 SDL 渲染器绘制纹理并显示到窗口。
- **音频播放**：

- **音频回调**：使用 SDL 提供的音频回调函数（`AudioCallback`）获取解码后的音频数据并将其写入音频缓冲区。
- **音频同步**：确保音频播放与视频播放的同步。

算法细节：

- **视频帧显示**：需要根据帧的实际尺寸更新 SDL 纹理，确保显示的画面正确。
- **音频缓冲管理**：在音频回调中，管理音频缓冲区，确保连续播放音频数据。

6.1.3. 音视频同步

逻辑描述：

- **同步机制**：
  - **时间戳计算**：使用视频帧的 PTS 和当前系统时间计算延迟。
  - **暂停和恢复**：处理暂停状态下的时间记录和恢复播放时的时间调整。

算法细节：

- **延迟调整**：通过计算当前时间与预期播放时间之间的差值来调整播放延迟。
- **暂停时间管理**：在暂停时记录时间，并在恢复时调整视频播放的起始时间。

6.1.4 .用户交互

逻辑描述：

- **用户输入处理**：
  - **检测键盘输入**：使用 SDL 事件处理机制，检测用户按键（如空格键用于播放/暂停）。
  - **播放控制**：根据用户输入，调用 TogglePause 方法切换播放状态。

算法细节：

- **事件处理**：使用 SDL 的事件循环处理用户输入事件，并相应地调整播放器状态。

6.1.5. 线程管理

逻辑描述：

- **线程设计**：
  - **解码线程**：负责从文件中读取数据包并解码。
  - **视频播放线程**：负责将解码后的视频帧渲染到屏幕上。
  - **音频播放线程**：负责播放解码后的音频数据。

算法细节：

- **线程同步**：使用互斥锁（std::mutex）和条件变量（std::condition_variable）来协调线程之间的工作。
- **队列管理**：使用线程安全的队列存储解码后的音视频帧，并在播放线程中进行处理。

6.2. 模块一设计

6.2.1. 类图

6.2.2. 接口访问流程

在多媒体播放器项目中，API接口的请求主要用于以下场景：

6.2.2.1. 访问频次

- **媒体文件的解码**：通过FFmpeg API解码音频和视频数据。这涉及到多个函数调用，包括 `avformat_open_input` 、 `avcodec_send_packet` 、 `avcodec_receive_frame` 等。
- **同步控制**：利用SDL API控制音频的播放和暂停，确保音视频同步。主要使用 `SDL_OpenAudio` 、 `SDL_PauseAudio` 、 `SDL_RenderPresent` 等函数。
- **用户交互**：响应用户操作，如播放、暂停等，通过SDL事件处理机制。

6.2.2.2. 访问频次

- **高频访问**：在解码过程中，API函数调用频次较高，如 `av_read_frame` 和 `avcodec_receive_frame` ，这些函数需要连续调用来解码视频和音频数据帧。
- **中等频次**：SDL的音频和视频渲染相关API函数，如 `SDL_UpdateYUVTexture` 、 `SDL_RenderCopy` ，在每一帧的播放过程中都会被调用。
- **低频访问**：播放器的初始化和关闭过程中的API调用，如 `avformat_open_input` 、 `SDL_CreateWindow` ，只在程序启动和结束时调用。

6.2.2.3. 容错处理

为了确保播放器的稳定运行，必须进行全面的容错处理：

- **错误检测与日志记录**：在每个API函数调用之后，必须检查返回值是否指示了错误。对于FFmpeg函数，可以使用 `av_err2str` 来获取错误信息并记录到日志中。
- **重试机制**：在解码过程中，如果某些API调用失败（如网络抖动导致的视频流中断），可以实现重试机制。
- **安全退出**：在发生不可恢复的错误时（如文件损坏或格式不支持），播放器应当安全地退出，并释放所有资源。

- **异常处理**：对于SDL音频和视频的异常处理，如果出现异常状态（如设备不支持的格式），应及时释放资源并通知用户.

7. 整体评分

类目	评分
技术复杂度	80
技术文档质量	80