基于 Android 平台的流媒体播放器的设计

刘洁彬 ，宋茂强
北京邮电大学软件学院，北京 (100876)
E-mail:liujiebin1986@126.com

摘要：随着移动通信技术和多媒体技术的迅速发展，移动流媒体服务有这很大的市场潜力，结合 FFmpeg 源码的解码流程设计出基于分层结构的流媒体播放器的系统架构，并讨论了 FFmpeg 的修剪优化方法，使其效率等特性适用于移动终端，再将优化后的代码移植到 Android 手机开发平台。该播放器不仅支持本地文件及流媒体文件的播放，还提供了对外部摄像头的控制功能。

关键词：Andorid; FFmpeg; PELCO-D 协议
中图分类号：TP393

1 引 言

随着移动通信技术和多媒体技术的迅速发展，融合手机、网络、多媒体技术为一体的视频监控技术也有了长足的进步，通过移动通信网络提供流媒体服务已经成为可能。全球移动用户数非常庞大，因此移动流媒体服务具有巨大的市场潜力，也正成为移动业务的研究热点之一。在这一背景下，针对移动网络和移动终端的特点，提出移动流媒体客户端的解决方案很有现实意义。

本论文结合 FFmpeg 开源代码中解码流程，提出了移动终端流媒体播放器基于分层体系架构的设计方案。该设计的特点是在底层屏蔽不同类型文件解码时对媒体流处理的差异，并且提供了对外部摄像头设备的控制功能，最终在 Android 平台[1]上实现该播放器。

2 播放器整体设计方案

播放器无论播放本地文件或是网络流媒体文件，都需要有获取媒体数据，解码音视频媒体流，将解码后媒体数据显示给用户三个处理阶段，根据 0 文件播放的流程中这三个明显的处理阶段，本文提出基于层次的播放器结构设计。

由于本地文件和网络流媒体文件的数据获取方式是不相同的，若要保持上层解码的一致性，需要对两类文件进行预处理，形成相同格式的数据提供给上层解码。根据以上特性，结合文件解码流程本文中面向实时监控的播放器设计采用分层结构，每层独立完成任务，使系统的耦合度降低，利于各层独立扩展而不影响上下层的应用。从下至上依次是数据提取层、数据预处理层、音视频解码层和用户界面。该流媒体播放器分层结构如图 1 所示。

用户界面层主要提供用户和播放器之间的交互接口，如播放本地文件时可以实现暂停、快进、快退等功能，在观看流媒体文件时可以通过数字键、导航键或者播放器上方向按钮控制摄像头的焦距、方向等信息。

音视频解码层主要有解码选择组件、各种主流音视频格式的解码器和多路媒体流之间同步的功能。解码选择组件从本地文件或者流媒体文件头中获取到媒体的解码格式信息，根据该格式信息选择相应的解码器对压缩后媒体流进行解码。该部分是由 FFmpeg 修剪优化后作

为播放器的解码模块的。多路媒体之间同步包括视频流和音频流的同步，在播放本地文件时可能还需要字幕的同步。

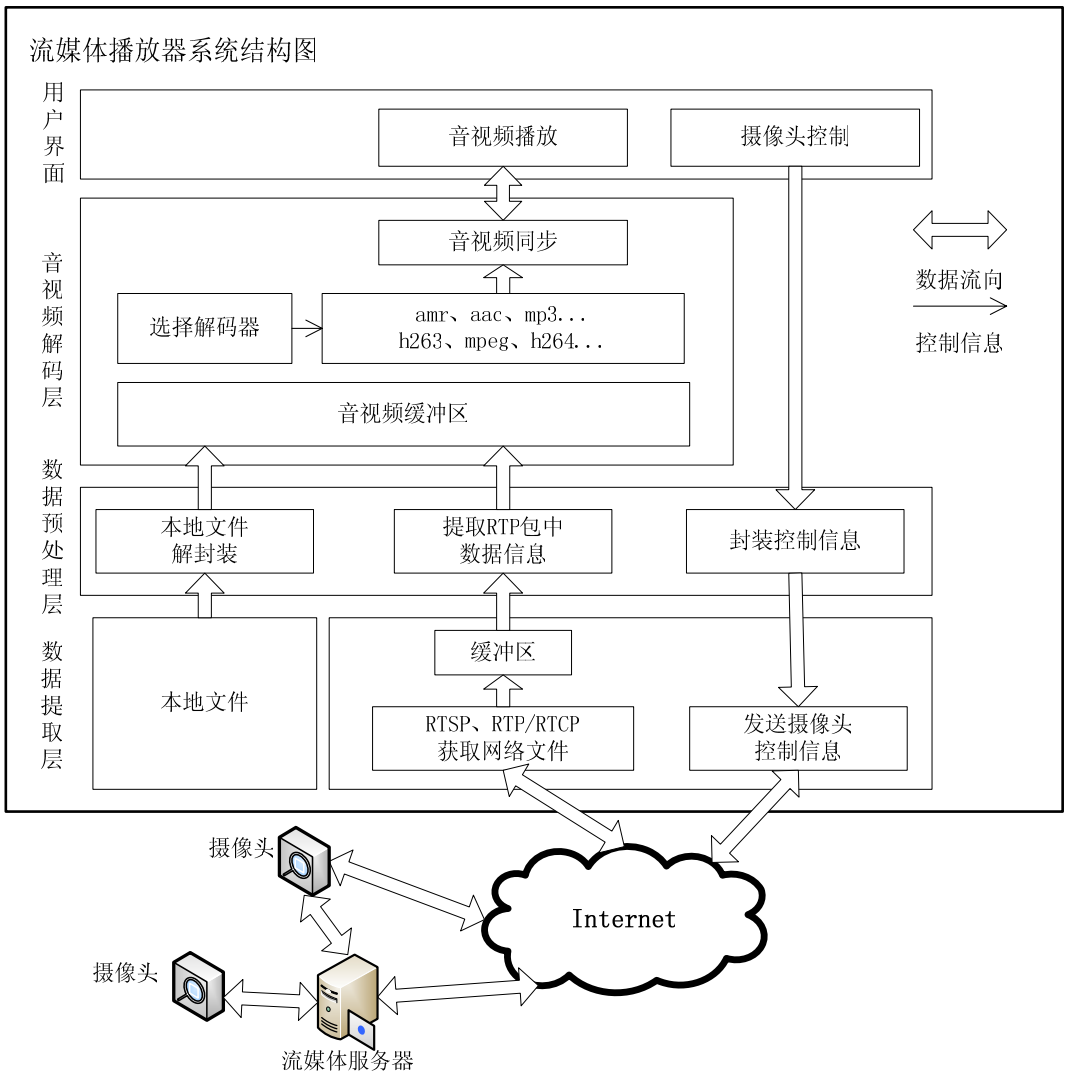

图1 面向实时监控播放器的分层结构

数据预处理层对本地文件按照其媒体格式解封装，获取文件的音视频或字幕等信息并将其按帧放入相应上层待解码缓冲区。对流媒体文件将去除 RTP 的包头信息，并将 RTP 中音视频信息组帧，将完整的数据帧传给上层待解码缓冲区。而封装控制信息组件按照 PELCO-D/P 协议规定的文本格式封装用户的控制输入，并将控制信息传递给下层。

数据获取层的功能包括本地文件、流媒体文件的获取和摄像头控制信息的发送，前者只需读取本地文件即可，流媒体文件的获取需要从流媒体服务器获取媒体数据信息。流媒体文件获取部分包括前期会话协商部分、数据发送部分和数据缓冲部分。其中媒体信息协商部分需要使用 RTSP 协议[2]协商媒体流常规信息，如媒体类型（视频和音频）、传输协议（RTP/UDP/IP…）和媒体格式（H263、mpeg…）和媒体传输端口等信息。

3 FFmpeg 到 Android 平台的移植

FFmpeg 是一个集录制、转换、音/视频编码解码功能为一体的完整的开源解决方案。但

本文中播放器只需要 FFmpeg 中对文件解封装及音视频解码部分的功能，若将 FFmpeg 整个解决方案全部移植到目标平台上会造成大量的代码冗余。并且 FFmpeg 代码的开发时基于 Linux 操作系统的，并没有考虑到手机平台的处理能力小，能源不足等限制，因此针对手机上特定功能需求将 FFmpeg 代码进行修剪及优化是十分重要的。

3.1 FFmpeg 修剪及优化

从 FFmpeg 如此庞大并且代码结构复杂的源代码中找出本文需要的代码确实是一项非常艰难的工作。在 Linux 下编译运行 FFmpeg 代码时需要经过 configure、make、make install 三步才能将 FFmpeg 正确的编译到 Linux 系统当中。其中 configure 阶段会生成一个 configure.h 和 make 文件，从这两个文件中可以查找出该次编译都编译了那些文件。

经研究发现在 configure 源代码的时候可以加入很多配置参数，其中参数分为基本选项参数、高级选项参数还有专门提供的优化参数。优化参数主要负责配置在编译时需编译的内容。对 FFmpeg 的修剪也恰是将本系统中不需要的文件去除，因此本文利用选择适当的优化参数的方法找出播放器所需文件。对这些参数仔细研究后，得出编译时设置的参数如下：

./configure --ena ble-version3 --disa ble-encoders --e nable-encoder=h263 --en able-encoder=amr_nb --disa ble-parsers --disa ble-bsfs --ena ble-muxer=tgp --disa ble-protocols --enab le-protocol=file。

以上面所示的参数配置编译源文件时，系统只将 h263、amr_nb 的编码方法和 3gp 的文件封装格式及其所有的解码格式、解封装文件的源代码部分编译到了链接库。

此时被编译到链接库的源代码集合即为本文所需的源代码有效集，通过查找 configure.h 和 make 文件中的后缀名为.o 文件，后缀名为.o 的文件是编译.c 代码时生成的目标文件，每一个被编译的.c 文件都会生成.o 文件，所以通过查看所有的后缀名为.o 的文件名，便可得知在该配置参数下被编译源文件有哪些，因此可以得出本文所需编译的源文件最小集合。

FFmpeg 开源代码虽然能够跨平台编译运行，但其代码的设计都是针对于 PC 机而言的，PC 机和手机从 CPU 处理能力、能源、内存等各方面的资源都具有很多大差异，本文中针对手机的特点主要从以下几个方面优化代码：

1. 去除冗余代码、规范程序结构、减少 if-else 的判断、调整局部和全局变量、使用寄存器变量代替局部变量，减少不必要的代码冗余，去除 FFmpeg 调试过程中的打印语句；

2. 用逻辑移位运算代替乘除操作，因为乘除运算指令的执行时间远远大于逻辑移位指令，尤其是除法指令，使用逻辑移位运算可以减少指令的运行时间；

3. 注意循环函数的调用，尽量减少多重循环的使用，编写代码时尽量减弱上次循环与下次循环的相关性，减少不必要的代码计算量；

4. 设置合理的缓存。针对 FFmpeg 移植的目标平台 Android 平台，设置适合此本台的缓存大小；。

这里对具体代码的修改就不一一重复了。

3.2 FFmpeg 移植

Google 发布的 NDK 的 makefile 文件即 Android.mk 文件语法和普通的 makefile 文件有很多不同之处，在跨平台编译 FFmpeg 源代码时并不能使用原有的 makefile 文件。所以移植的先决条件就是将 FFmpeg 里的 makefile 文件全部替换为 NDK 中的 Android.mk 文件。

通过分析 FFmpeg 的模块结构得知 avutil 是基础模块，avcodec 模块的编译基于已经编译好的 avutil 模块，avformat 基于前两者，按照这种模块结构本文编译移植的顺序为 avutil、avcoedec、avformat，编译的步骤详细说明如下：

1. 关于 config.h 和 config.mak

首先说明一下 FFmpeg 自带的 ma kefile 的框架，FFmpeg 在经过 configure 命令之后会产生一个 config.h 文件和一个 config.mak 文件，这两个文件加起来共有 6 00-700 个宏定义，用来描述编译后代码的各个方面参数设置，其中有关于体系架构、编译器、链接库、头文件、版本、编解码器等等相关的宏定义。在这一部分必须要修改关于平台差异方面的定义，比如必须把体系架构改成 Android 平台的 ARMv5TE，这时文件编译的时候指令集就会选择 ARM 的指令集而不是 X86 的指令集。这两个文件很重要，以后很多文件都要 include c onfig.h 这个文件，编译器会根据这个文件而选择性对代码进行编译。

2. 编译 libavutil.a

在 libavutil 建立一个 Android.mk 的文件，libavutil 里的 makefile 文件需要调用 subdir.mak，这个其实就是真正的编译，但是书写在 Android.mk 下，这个 make 文件可以不要，但需要直接把对应的源文件引入，标准的 makefile 是指定.o 目标文件，但在 Android.mk 中需要直接指定.c 源文件，Android.mk 文件如下所示：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := avutil
LOCAL_SRC_FILES:=adler32.c  \
                    ……      \
include $(BUILD_STATIC_LIBRARY)
```

编译时可能会出现很多错误，但这些问题归结起来大部分都是因为有些头文件没有引入而产生的问题，只要引入相应的头文件后就可以了。比如不识别某些文件的 size_t 关键字，在该文件 include stdio.h 后就不报错了，其他类似错误就不一一例举了。

其它模块按照相同的方法书写 Android.mk 文件，移植到 Android 平台最为本文中播放器的解码模块。

4 各层模块详解

4.1 数据获取层

该层完成主要功能为与流媒体服务器协商媒体信息细节，并根据协商结果从服务器端获取流媒体数据，将流媒体数据存入缓冲区，按照本文中缓冲策略将数据包发送给数据预处理

层，其结构图如图 2 所示：

本文中该层一共启动五个线程，其中一个线程中启动 TCP 连接，用于 RTSP 会话协商，并且在 RTP 数据传输期间，该 TCP 连接必须一直保留。两个线程分别为接收音频和视频 RTP 数据的线程，另外两个线程分别为接收以及发送音频和视频的 RTCP 数据包。

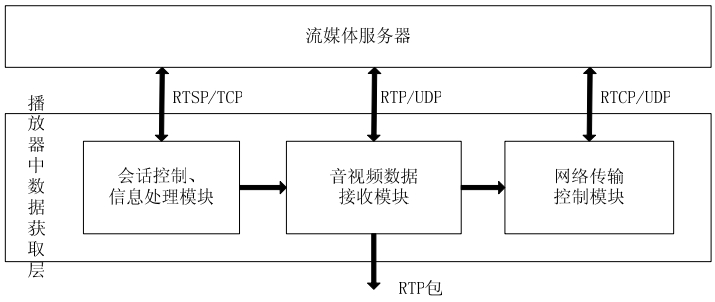

图 2 数据获取模块结构图

4.2 数据预处理层

本层对本地文件的预处理完全依赖于 FFmpeg 提供的功能文件解封装功能，而流媒体文件的预处理需将一个或多个 RTP 数据包整合在一起，这部分技术已经相对成熟，本文将不再复述。

本文中流媒体播放器区别于其他普通流媒体播放器的最大特点即为能对外部带有云台的摄像头进行控制，例如焦距、上、下、左、右等方面的设置。所以本文中使用 PELCO-D 协议作为云台控制协议[3]。其协议格式如表 1 所示：

表 1 P ELCO-D 协议格式

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
同步字	地址码	命令字 1	命令字 2	数据 1	数据 2	校验码

表 1 中第一字节为同步字也称起始符号，通常都是 0xFF。该符号字节用来检测所采用的收发方式正确与否。第二字节填写为目标设备的地址，在命令字 1 字节中为对摄像头光圈及焦距的控制。在命令字 2 字节为焦距及变倍控制，其中 Bit4, Bit3, Bit2, Bitl 为上下左右控制位，最后一个 Bit0 位总是 0。数据 1 字节中，水平方向速度（00-3F）。数据 2 字节，垂直方向速度，其数值同数据字节 1。校验码字节为前六个字节之和。

本文设计的 PELCO-D 协议文本，最初默认情况下位命令字 1、命令字 2 全部为 0，数据字 1 和数据字 2 值为 20H。通过上层发送的按键消息修改相应命令字 1、命令字 2 的相应位。根据用户通常输入习惯，本文设计按键消息和屏幕上导航键与摄像头控制对应表如表 2 所示：

表 2 按键与控制信息对照表

按键信息	控制信息	按键信息	控制信息
2 和上侧	上	1 和缩小键	焦距远
8 和上侧	下	3 和放大键	焦距近
4 和左侧	左	以上按键抬起	停止现有动作
6 和右侧	右		

目前本文中流媒体播放器只提供以上六种控制功能，该模块根据上层出发的按键信息设

置相应位为 1，计算字节的值，形成七个字节文本发送至外部设备，当接收到上层按键停止的消息后，统一发送{0xff,0x01,0x00,0x00,0x00,0x00,0x01,}停止命令。

4.3 解码及显示层

解码层主要应用 FFmpeg 移植到 Android 平台的代码作为播放器的解码模块，该部分代码支持包括 avi、3gp、MPEG-4 等 90 多种解码格式及文件格式，并且经过修剪优化后的 FFmpeg 代码效率和效能都得到了很大的提高。

显示层本文主要应用开源的 SDL 函数库实现，SDL(Simple DirectMedia Layer)是一个跨平台的，免费的开源软件。该软件应用 C 语言开发，对外提供多种平台上图像、声音和其它输入设备的简单接口。经常用于游戏和其他多媒体应用的开发，该开源软件可以运行于多种操作系统上，其中包括 Linux、PSP、Windows、Mac OS X 等。同时 SDL 还具有视频，音频，线程，定时器，事件等功能。

5 总结

本文介绍了基于 Android 平台的流媒体播放器的分层设计结构及其各层的详细设计，该播放器的解码库源自经过剪切优化的 FFmpeg 源代码，并且本文中的播放器提供了对外部摄像头的控制功能，是其应用范围更为广泛，播放器播放本地文件及流媒体文件的效果图分别如图 3、图 4 所示：

图 3 本地文件播放效果图

图 4 网络文件播放效果图

本文虽然完成了带有控制功能的流媒体播放器的原型功能实现，但还有很多例如 QoS、代码优化的问题需要进一步的研究。

6 致谢

感谢我的导师对初稿中的错误修改以及提出的宝贵意见。

参考文献

[1]. 余志龙等 ,Google Android SDK开发范例大全余志龙，人民邮电出版社 2009 年 07 月.

[2]. H Schulzrmne, Rao Lanphier, Real Time Streaming Protocol(RTSP) RFC2326.April 1998

[3]. Wang Y, Zhu Q.R, and Shaw L. M aximallys moothim agere covery in transformcoding.IEEE Trans Communnication.1993.1544-1551.

Design of Streaming Media Player based on Android

LIU Jiebin , Song Maoqiang

（*Department of software engineering, Beijing university of posts and telecommunications, Beijing* 100876)

Abstract： With the development of mobile communications and mutil-media technology, mobile streaming media service has hug e m arket po tential, proposes a d esign base d on the l ayering ar chitecture which will be used in streaming play er combining with the decoding process in FFmpe g, and propose a fast pruning method to reduce the fil es of FFm peg and a opt imization to im pove the ef ficiency of FFmpeg, After that, FFmpeg source code is moved to Android platform. The media pl ayer can play not only local files and streaming media files, but also can supports the fuction of control external camera.

Key word: Andorid; FFmpeg; PELCO-D protocol