FFMPEG 的快速入门

关于本文档：

本文档全整理自 ffmpeg 工程组网站（www.ffmpeg.com.cn）,所有内容均出自此处，各人没有做任何修改。因为工作当中需要用到 ffmpeg，在线查看不是太方便，所以利用非工作时间，整理了四份文档供方便学习之用。平时工作太忙时间有限，文档有误之处，大家见谅，也可以访问 www.ffmpeg.com.cn 查看在线文档。

目录

1 Ffmpeg 简介.....1
1.1 获取 ffmpeg 源代码.....1
1.2 组成结构.....1
1.3FFMPEG 功能.....1
1.4ffmpeg 与 ffdshow 的关系.....2
2 FFMPEG 入门的基础知识.....2
2.1 关于 frame 的一些基础知识.....3
2.1 时间戳.....4
2.3 关于编译 FFMPEG 的初级教程.....5
3 Ffmpeg 快速安装.....6
3.1FFMPEG 和 FFMPEG-PHP 的安装.....6
3.2 如何使 PHP 支持 ffmpeg（ffmpeg-php 模块的安装）.....9
4 Ffmpeg 快速命令使用.....13
4.1Ffmpeg 使用语法.....13
4.2 视频文件截图.....13
4.3 如何使用 ffmpeg 编码得到高质量的视频.....13
4.4 使用 ffmpeg 录像屏幕.....14
5 Ffmpeg 快速应用开发.....15
6 Ffmpeg 编译详解.....24
Linux.....24
开发人员注意选项.....27
7 Ffmpeg 编译 FAQ 集.....28
7.1 Configure 过程出错.....28
7.2Make 过程出错.....28
7.3 如何编译 FFServer.....28
7.4 提供 java+winwows 下使用 ffmpeg 解决视频转换思路和代码.....37
7.5 如何用 vc 顺利编译 ffmpeg.....39
7.6FFMPEG 在 windows 下编译出错.....41
7.7VC 下编译的几个小问题.....43
7.8Ffmpeg（2006/10/26-6793 版） dll lib x264 vc6sp6 编译成功.....44
7.9 关于运行 ffserver 有错误.....46
7.10 如何加入 faac 和 faad 的支持.....48
7.11ffmpeg.exe 初始化出错.....49

1 Ffmpeg 简介

FFmpeg is a complete solution to record, convert and stream audio and video. It includes libavcodec, the leading audio/video codec library. FFmpeg is developed under Linux, but it can compiled under most operating systems, including Windows.

1.1 获取 ffmpeg 源代码

1.2 组成结构

ffmpeg 项目由以下几部分组成:
ffmpeg 视频文件转换命令行工具,也支持经过实时电视卡抓取和编码成视频文件.
ffserver 基于 HTTP(RTSP 正在开发中)用于实时广播的多媒体服务器.也支持时间平移
ffplay 用 SDL 和 FFmpeg 库开发的一个简单的媒体播放器
libavcodec 一个包含了所有 FFmpeg 音视频编解码器的库.为了保证最优性能和高可复用性,大多数编解码器从头开发的.
libavformat 一个包含了所有的普通音视格式的解析器和产生器的库.

1.3FFMPEG 功能

Ffmpeg 能使用任何支持的格式和协议作为输入：
比如你可以输入 YUV 文件：ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
它将要使用如下文件： /tmp/test0.Y, /tmp/test0.U, /tmp/test0.V, /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, 等等...
你能输入原始的 YUV420P 文件：ffmpeg -i /tmp/test.yuv /tmp/out.avi
原始的 YUV420P 文件包含原始的 YUV 极性，每帧以 Y 平面开始，跟随 U 和 V 平面，它们是 Y 平面水平垂直的一半分辨率
你能输出原始的 YUV420P 文件
ffmpeg -i mydivx.avi -o hugefile.yuv
你能设置几个输入文件和输出文件
ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg 上面的命令行转换音频文件 a.wav 和原始的 YUV 视频文件 a.yuv 到 mpeg 文件 a.mpeg
你也能同时转换音频和视频
ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2 上面的命令行转换 a.wav 的采样率到 22050HZ 并编码为

mpeg 音频
你也能同时编码到几种格式并且在输入流和输出流之间建立映射
ffmpeg -i /tmp/a.wav -ab 64 /tmp/a.mp2 -ab 128 /tmp/b.mp2 -map 0:0 -map 0:0 上面的命令行转换一个 64Kbits 的 a.wav 到 128kbits 的 a.mp2 '-map file:index'在输出流的顺序上定义了那一路输入流是用于每一个输出流的，转码解密的 VOB: ffmpeg -i snatch_1.vob -f avi -vcodec mpeg4 -b 800 -g 300 -bf 2 -acodec mp3 -ab 128 snatch.avi 上面的命令行将 vob 的文件转化成 avi 文件，mpeg4 的视频和 mp3 的音频。注意命令中使用了 B 帧，所以 mpeg4 流是 divx5 兼容的。GOP 大小是 300 意味着 29.97 帧频下每 10 秒就有 INTRA 帧。该映射在音频语言的 DVD 转码时候尤其有用

1.4ffmpeg 与 ffdshow 的关系

看不少人对 ffdshow 和 ffmpeg 的关系有点搞不清楚，所以响应 Leon 老大的号召，发个帖子简单说明一下。
ffdshow is DirectShow and VFW codec for decoding/encoding many video and audio formats, including DivX and XviD movies using libavcodec, xvid and other opensourced libraries with a rich set of postprocessing filters.
上边是 sf 的 ffdshow 的简单说明。我得理解就是，ffdshow 是对一些 codec（ffmpeg, xvid， and other）的封装，封装成了 DirectShow 和 VFW 的标准组件。比如对于 xvid 来讲，ffdshow 是可以选择具体使用那个 codec 的， ffmpeg(libavcodec) or xvid。
那么封装有没有额外的成本哪？有，但对大部分应用来讲，可以忽略不计。就如 c++和 c。
先说这么多，欢迎拍砖。
写得不错，目前群里面的兄弟都在研究 ffmpeg，对 ffdshow 研究的少，因为 ffdshow 只是一件外衣，核心还是 ffmpeg，估计等一段时间就会有些朋友转向 ffdshow 的。
我再补充几句吧,vfw 和 dshow 里的 CODEC 分别是通过 fourcc 码和 guid 机制寻找的,可以在系统注册 codec 后调用,比自带编解码库形式更加统一,便于使用。此外，vfw 和 dshow 是代表了两个微软不同时期的音视频处理封装库，里面包含了音视频驱动，音视频处理的一整套方案。
原文出处，ffmpeg 工程组：ffmpeg 与 ffdshow 的关系

2 FFMPEG 入门的基础知识

2.1 关于 frame 的一些基础知识

关于 frame 的一些基本知识 只是摘抄了一部分，供初学者参考。
b.帧速率：帧速率是每秒显示的图像数。标准影片(NTSC) 是 29.97 帧第秒 (fps)，电影是每秒 24 帧 fps。欧洲标准是(PAL) 25 帧 fps。如果你对你影片的尺寸

不是太注重的话，保留默认的 Current 选项。这将会使你制作的影片的帧速率和源文件一致。不管怎样，如果你想降低带宽和 CPU 的占用，你可以选择一个低的帧速率。高的帧速率拥有高的品质的，但文件尺寸也更大。如果你选择的帧速率低于你的源文件的帧速率，一些帧将被删除。如果你选择的帧速率比你的源文件高的话，已有的帧将被重复 (不推荐，因为增加了尺寸，但品质没有提高)。如果你选择的帧速率低于你的源文件的帧速率，使用一个你当前帧速率的简分数，比如 1/2， 1/3 等等。例如，你当前的帧速率是 30 (29.97)，使用 15 或 10。但话说回来了，要最好的 H.264 品质，最好保留 Current，当前）设置。

c.关键帧：很多编码软件使用 frame differencing（帧差异）来压缩图像。帧差异其实是判断从开始帧起哪些信息发生了变化 (称为 key frame 关键帧)。关键帧包含了图像的所有信息。后来的帧仅包含改变了的信息。这取决于你用的编码软件，你可以指定你想要的关键帧如何出现。 如果你没有足够的关键帧，你的影片品质可能比较差，因为所有的帧从别的帧处产生。另一问题是，关键帧多了将导致影片更大，码率更高。 在一些编码软件中，当从一帧到下一帧有太多的内容发生改变时，那些增加的关键帧是自动插入的。 对于一般的用途，一个比较好的原则是每 5 秒设一个关键帧。如果你正在建立一个 RTSP 流文件，并且关心传输网络的可靠度，你可能要 1 到 2 秒增加一个关键帧。要让编码软件来处理关键帧的间隔，选择 Automatic。针对 H.264，我们推荐让编码软件来确定关键帧的间隔，为此你要选择 Automatic 以获得最佳品质。

e.码率：通常情况下，高码率就有高的品质，但文件也会很大。在大多数情况下，你要根据你观看的影片设置码率，例如，对于 384K 连接速度，你要限制码率为 350-360k 每秒来留一些带宽给网络传输。如果文件是下载回来后播放，那码率可以很高（高码率，然而，网速比较慢的用户将要花比较长的时间来等待播放的开始）。另外，记住在对话框中设置码率时，你要留一些空间给音频。

针对 H.264， 这里有一些常用的码率方案：

- § 画面尺寸 1920 x 1080 (真正高清)， 选择码率为 7,000-8,000 Kbps。
- § 画面尺寸 1280 x 720 (通用高清)， 选择码率为 5,000-6,000 Kbps。
- § 画面尺寸 640 x 480 (标清)， 选择码率为 1,000-2,000 Kbps。
- § 画面尺寸 320 x 240 (网络传输)， 选择码率为 300-500 Kbps。
- § 画面尺寸 176 x 144 (3G)， 10-15 fps 的内容选择码率为 50-60 Kbps， 24-30 fps 的内容选择码率为 150-200 Kbps。

提及 3G 格式，一定要记住影片的码率会被你设置的其它的压缩选项所影响，如同帧速率。因此高的帧速率，要有高的码率，如果你对码率要求不是特别严格并且你只想 QuickTime 带给你一个比较好的影片效果，你可以通过选择 Automatic 让 H.264 编码器选择一个理想的码率。 编码器会按你选择的尺寸和你用品质滑动条选择的品质来选择合适的编码。

f.优化：如果你已经输入了你自己的码率而不是自动选择码率，在 Optimized for 下拉菜单中就有你选择的传送方式的相关选项。这些选项将告诉编码器可以高于或低于你选择的的码率多少。要得到最好的品质，选择 Download。如果你想要借助 CD 或 DVD 来传送影片，在码率中选择 CD/DVD，CD/DVD 需要被进行一些限制，因此光驱要保持与观看者的电脑读与数据传送畅通 。如果你想借助 RTSP 流来传送影片，码率选择 Streaming 将是最大限制。此选项仅能用于有限制的压缩软件，如 H.264。

相关问题：

为什么会有关键帧的存在?
对应解答:
这是因为 mpeg 或者其他压缩方法（我只了解过 mpeg），为了提高压缩比，就选择某一帧作为基帧，以它为参考，后面的帧只记录改变的信息，这是一个压缩的
技巧，记录信息的改变是通过前后帧之间的图像相关性来完成的，分为（I，B，P）三种帧式，这三种帧式分别是三种不同的采用相关性的方式。这里的基帧就是
关键帧了。
有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论

2.1 时间戳

音视频同步-时间戳

媒体内容在播放时，最令人头痛的就是音视频不同步。从技术上来说，解决音视频同步问题的最佳方案就是时间戳：首先选择一个参考时钟（要求参考时钟上的
时间是线性递增的）；生成数据流时依据参考时钟上的时间给每个数据块都打上时间戳（一般包括开始时间和结束时间）；在播放时，读取数据块上的时间戳，同
时参考当前参考时钟上的时间来安排播放（如果数据块的开始时间大于当前参考时钟上的时间，则不急于播放该数据块，直到参考时钟达到数据块的开始时间；如
果数据块的开始时间小于当前参考时钟上的时间，则“尽快”播放这块数据或者索性将这块数据“丢弃”，以使播放进度追上参考时钟）。

可见，避免音视频不同步现象有两个关键——一是在生成数据流时要打上正确的时间戳。如果数据块上打的时间戳本身就有问题，那么播放时再怎么调整也于
事无补。假如，视频流内容是从 0s 开始的，假设 10s 时有人开始说话，要求配上音频流，那么音频流的起始时间应该是 10s，如果时间戳从 0s 或其它时间开始打，
则这个混合的音视频流在时间同步上本身就出了问题。打时间戳时，视频流和音频流都是参考参考时钟的时间，而数据流之间不会发生参考关系；也就是说，视频
流和音频流是通过一个中立的第三方（也就是参考时钟）来实现同步的。第二个关键的地方，就是在播放时基于时间戳对数据流的控制，也就是对数据块早到或晚
到采取不同的处理方法。图 2.8 中，参考时钟时间在 0-10s 内播放视频流内容过程中，即使收到了音频流数据块也不能立即播放它，而必须等到参考时钟的时间达
到 10s 之后才可以，否则就会引起音视频不同步问题。

基于时间戳的播放过程中，仅仅对早到的或晚到的数据块进行等待或快速处理，有时候是不够的。如果想要更加主动并且有效地调节播放性能，需要引入一个
反馈机制，也就是要将当前数据流速度太快或太慢的状态反馈给“源”，让源去放慢或加快数据流的速度。熟悉 DirectShow 的读者一定知道，DirectShow 中的
质量控制（Quality Control）就是这么一个反馈机制。DirectShow 对于音视频同步的解决方案是相当出色的。但 WMF SDK 在播放时只负责将 ASF 数据流读出并
解码，而并不负责音视频内容的最终呈现，所以它也缺少这样的一个反馈机制。

为了更好地理解基于时间戳的音视频同步方案，下面举一个生活中的例子。假设你和你的一个朋友约好了今天 18:00 在沪上广场见面，然后一起吃饭，再去打
游戏。实际上，这个 18:00 就是你和你朋友保持同步的一个时间点。结果你 17:50 就到了沪上广场，那么你必须等你的朋友。10 分钟过后，你的朋友还没有到，这
时他打来电话说有事耽搁了，要晚一点才能到。你没办法，因为你已经在旁边的餐厅预订了

位置，如果不马上赶过去，预订就会被取消，于是你告诉你的朋友直接
到餐厅碰头吧，要他加快点。于是在餐厅将来的某个时间点就成为你和你朋友的又一个同步点。虽然具体时间不定（要看你朋友赶过来的速度），但这样努力的方
向是对的，你和你朋友肯定能在餐厅见到面。结果呢？你朋友终于在 18:30 赶过来了，你们最终“同步”了。吃完饭 19:30 了，你临时有事要处理一下，于是跟你
朋友再约好了 20:00 在附近的一家游戏厅碰头。你们又不同步了，但在游戏厅将来的某个时间点你们还是会再次同步的。

悟出什么道理了没有？其实，同步是一个动态的过程，是一个有人等待、有人追赶的过程。同步只是暂时的，而不同步才是常态。人们总是在同步的水平线上
振荡波动，但不会偏离这条基线太远。

作者: happydeer 来源: CSDN 日期: 2006-3-2 3:26:58
from: http://www.window07.com/dev/code/vc/2006-3-2/k56287.htm

2.3 关于编译 FFMPEG 的初级教程

首先我们要下载相关工具，这里不多说，大家按照我的地址去下载文件就好了
MINGW 下载地址：http://prdownloads.sourceforge.net/mingw/MinGW-3.1.0-1.exe?download
然后在下载 MSYS ：http://prdownloads.sf.net/mingw/MSYS-1.0.10.exe?download
好先喝点咖啡，哈哈
首先我们先安装一下 MINGW，我的目录是 c:/MINGW，默认的，然后接下来要安装 MSYS
这里要有些注意，安装目录看到别人是这么说的，C:\MinGW\bin\1.0，意思就是安装在你的 MINGW 目录下的 BIN 里面
OK，开始安装吧！注意安装完毕以后有个 DOS 界面，这里至关重要，请谨慎操作
安装好以后，我们在下载一个 LAME，我不知道是干嘛用的，反正就安装吧
下载地址：http://prdownloads.sourceforge.net/lame/lame-3.97b2.tar.gz?download
然后解压到 C:\MinGW\bin\1.0\lame-3.97
好了，启动桌面上的 MSYS，然后出入一下代码
首先进入 lame 目录，呵呵
下面步骤
1.cd c:
2.cd MinGw
3.cd bin
4 cd 1.0
5.cd lame-3.97
然后开始编译，一下是步骤
1 ./configure（根据的你电脑速度决定快慢）
2. make
3. make install
好了，你可以下载 FFMPEG 文件进行编译了，哈哈
首先下载 FFMPEG
然后解压到磁盘里面，同样使用 MSYS 进入该目录输入一下代码
./configure --enable-memalign-hack --enable-mingw32 --enable-mp3lame --extra-cflags=-I/local/include

--extra-ldflags=-
L/local/lib
执行完毕以后，在输入
make
执行完毕以后，在输入
make install
OK 了吧：）
在初级编译的过程中可能会遇到各种问题，在我们的论坛中有很多人提出了编译问题，而且大部分都得到了很好的解决
请参照我们的论坛：

3 Ffmpeg 快速安装

3.1FFMPEG 和 FFMPEG-PHP 的安装

===软件下载===! FFmpeg 官方主页： http://ffmpeg.sourceforge.net http://ffmpeg.org/
cvs -z9 -d:pserver:anonymous@mplayerhq.hu:/cvsroot/ffmpeg co ffmpeg

FFmpeg-php 官方主页 http://ffmpeg-php.sourceforge.net
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ffmpeg-php login
cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ffmpeg-php co ffmpeg-php

Lame 下载地址:http:// www.linuxpackages.net/sea ... ;name=lame&ver=

Apache v2.0.54 官方主页： http://www.apache.org
http://www.apache.org/dist/httpd/httpd-2.0.54.tar.gz

PHP v4.3.11 官方主页： http://www.php.net
http://cn.php.net/distributions/php-5.0.4.tar.gz

GD Library v2.0.33 官方主页： http://www.boutell.com/gd/
http://www.boutell.com/gd/http/gd-2.0.33.tar.gz

FreeType v2.1.10 官方主页： http://www.freetype.org
http:// savannah.nongnu.org/downl ... etype-2.1.10.tar.gz

Jpeg v6b 官方主页： http://www.ijg.org
http://www.ijg.org/files/jpegsrc.v6b.tar.gz

LibPNG v1.2.8 官方主页： http://sourceforge.net/project/showfiles.php?group_id=5624
http:// voxel.dl.sourceforge.net/ ... 1.2.8-config.tar.gz

zlib v1.2.2 官方主页： http://www.gzip.org/zlib/
http://www.zlib.net/zlib-1.2.2.tar.gz
[编辑]开始安装
解压缩
把所有源码压缩包放在一个目录中，解压缩所有 .tar.gz 和 .tar.bz2 压缩包
for i in `ls *.gz`;do tar zxvf $i; done;
for i in `ls *.bz2`;do tar jxvf $i; done;

安装
lame
lame-CVS20050706-i686-1jto.tgz 解压后
复制 lame 中./usr/include ./usr/bin ./usr/lib 三个文件夹到服务器对应地方

ffmpeg
cd ffmpeg
./configure --prefix=/usr/local --enable-memalign-hack --enable-mp3lame --enable-shared
make
make install
cd..
cp -r /usr/local/lib/libav*.so /usr/lib
注：
--prefix=/usr/local 不得修改，否则 ffmpeg-php 安装不上
运行/usr/local/bin/ffmpeg –v 测试是否安装成功

Apache2
cd httpd-2.0.54
./configure --enable-so --prefix=/server/httpd --with-config-file-path=/server/httpd/conf
make
make install
cd ..
注：
运行/server/httpd/bin/httpd –k start http://192.168.1.xxx 测试是否安装成功
测试完后要关闭/server/httpd/bin/httpd –k stop

FreeType
cd freetype-2.1.10
./configure --prefix=/usr/local/freetype
make
make install
cd ..

LibPNG
cd libpng-1.2.8

./configure --prefix=/usr/local/libpng --enable-shared --enable-static
make test
make install
cd ..

Jpeg
cd jpeg-6b
mkdir /usr/local/jpeg
mkdir /usr/local/jpeg/bin
mkdir /usr/local/jpeg/lib
mkdir /usr/local/jpeg/include
mkdir /usr/local/jpeg/man
mkdir /usr/local/jpeg/man/man1
./configure --prefix=/usr/local/jpeg --enable-shared --enable-static
make
make install
cd ..

zlib
cd zlib-1.2.2
./configure
make
make install
cd ..

GD Library
cd gd-2.0.33
./configure --prefix=/usr/local/gd --with-jpeg=/usr/local/jpeg --with-freetype=/usr/local/freetype --with-png --with-zlib
make
make install
cd ..

ffmpeg-php
按照官方的说法，可以在装 php 前安装 ffmpeg-php
但是我没成功
运行不了 phpize
因此此步在装了 php 后再进行
有点傻

PHP#####
cd php-4.3.11
./configure --prefix=/server/php --with-apxs2=/server/httpd/bin/apxs --with-gd=/usr/local/gd --enable-gd --enable-gd-native-

ttf --with-jpeg-dir=/usr/local/jpeg --with-png --with-ttf --with-zlib --with-freetype-dir=/usr/local/freetype --with-config-
file-path=/server/httpd/conf
make
make install
cp php.ini-dist /server/httpd/conf/php.ini
cd ..

ffmpeg-php
mv ffmpeg-php /path/to/php_sources/ext/ffmpeg
cd /path/to/php_sources
autoconf
./configure --prefix=/server/php --with-apxs2=/server/httpd/bin/apxs --with-gd=/usr/local/gd --enable-gd --enable-gd-native-
ttf --with-jpeg-dir=/usr/local/jpeg --with-png --with-ttf --with-zlib --with-freetype-dir=/usr/local/freetype --with-config-
file-path=/server/httpd/conf --with-ffmpeg=/user/local/include
make
make install

安装完毕
运行 /server/httpd/bin/httpd –k start
测试
<?
phpinfo();
?>

3.2 如何使 PHP 支持 ffmpeg（ffmpeg-php 模块的安装）

下载 ffmpeg 安装包地址: http://ffmpeg.mplayerhq.hu/download.html
使用该命令下载 ffmpeg: svn checkout svn://svn.mplayerhq.hu/ffmpeg/trunk ffmpeg
./configure --enable-share --prefix=/usr
make clean && make && make install

下载 ffmpeg-php 安装包地址: http://ffmpeg-php.sourceforge.net/
tar -xvf ffmpeg-php-0.4.9.tar
mv ffmpeg-php-0.4.9 php-4.3.6/ext/ffmpeg
cd php-4.3.6/ext/ffmpeg
phpize
cd /opt/software/php-4.3.6
rm configure
./buildconf –force
第二次编译 php，让它支持 ffmpeg-php
./configure --with-png-dir=/usr --with-gd --enable-gd-native-ttf --with-ttf --with-freetype --without-gdbm

--with-gettext --
with-ncurses --with-gmp --with-iconv --with-jpeg-dir=/usr --with-png --enable-ftp --enable-sockets -with-xml --with-dom --
with-zlib --enable-track-vars --with-mysql --with-apxs2=/usr/local/apache2/bin/apxs --with-ffmpeg=/user/local/include
make && make install
加粗为需要注意的地方,路径或参数不同会造成无法加载 ffmpeg.so 的错误.
错误提示如:
Warning: dl(): Not supported in multithreaded Web servers - use extension statements in your php.ini
加粗为需要注意的地方,路径或参数不同会造成无法加载 ffmpeg.so 的错误.
错误提示如:
Warning: dl(): Not supported in multithreaded Web servers - use extension statements in your php.ini
有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：
FFMPEG 和 FFMPEG-PHP 的安装
[编辑] 获取 ffmpeg 源代码

This tutorial is about transcoding video fom one codec into another using FFMPEG. I got deeper into FFMPEG when I wanted to transcode into FLV (Flash Video) and it works very well. I developed the Riva FLV Encoder, a GUI for FFMPEG.
Update 26.10.2006: This tutorial is a little outdated as there were many changes in FFMPEG like the switch from CVS to Subversion and the workflow to compile FFMPEG under Windows has become more difficult. For the new tweaks check this tutorial.
Download MinGW " MSYS current releases (Window Exe Binaries MSYS-1.0.10.exe & MinGW-3.1.0-1.exe)
Install MinGW
Install MSYS
HINT: During the Postinstall be sure to set the right path to MinGW with a "/" instead of a Windows-"". If you did it wrong anyway re-install MSYS to the same directory and do the postinstall right (I missed it a few times)
Download and compile Lame
Extract Lame to your MSYS home-directory
Open MSYS and change to your lame-directory (cd ../lame-XXX)
Enter the following commands:
./configure //(takes a few minutes)
make //(lame is being comiled; takes a few minutes, too)
make install
After installing you will recognize that there are new directories and files in MSYS/local which we will use while compiling ffmpeg with mp3-support
Download Subversion Client like Tortoise SVN (http://http://tortoisesvn.tigris.org/) and install it
Check out the sourcecode from svn://svn.mplayerhq.hu/ffmpeg
Compile FFMPEG
Change the directory in MSYS to your ffmpeg-directory (cd ../ffmpeg)
Enter the command:
./configure --enable-memalign-hack --enable-mingw32 --enable-mp3lame --extra-cflags=-I/local/include --extra-ldflags=-L/local/lib

HINT: you can paste into MSYS by pressing your center mouse-button
1. "--enabled-memalign-hack" is a Windows hack. Without this option ffmpeg always crashs with the message "removing common factors from framerate" when encoding AVIs.
2. "--enable-mingw32". I see no difference without it but we compile with MinGW and it would not do a harm when ffmpeg knows this
3. "--enable-mp3lame": Enable transcoding audio with the open-source mp3-lame-codec
4. "--extra-cflags=-I/local/include --extra-ldflags=-L/local/lib": The cflags- and ldflags-parameter sets the right path to your lame-installation which you did in step 3.d.
Enter command: make (ffmpeg is being compiled; takes a few minutes) With "make install" you could now copy the ffmpeg.exe to c:Program Filesffmpeg. But there is no need to.

MSYS 是什么,他与 CYGWIN 有什么区别?
[编辑] Use FFMPEG

Copy your compiled ffmpeg.exe from your MSYS directory to the directory where you like to transcode with ffmpeg
Open the Dos-Shell and change to the directory where you copied the ffmpeg.exe
Copy a test.mpg into your directory and enter the following command:
ffmpeg -i test.mpg -ab 56 -ar 22050 -b 500 -r 15 -s 320x240 test.flv
Your first FLV should be encoded now

[编辑] Render Images from a Video

Enter command:
ffmpeg -an -y -t 0:0:0.001 -i test.flv -f image2 test%d.jpg
HINT: With -t you set the length of images to be extracted. Above we entered 1 millisecond the extract one image. If you miss this parameter all images of the video will be extracted

[编辑] ZLib Support

(e.g. for TSCC and Quicktime codecs). This should be compiled into FFMPEG. It is not an explicit compile in the configure statement. Do the following steps and after configure you should see that zlib is "on".
Download and compile ZLib
Extract the files to your mysys directory
Change the directory in MSYS to that directory
Enter command
./confugure
make
make install.
[编辑] AC3 Support

Add "--enable-a52 --enable-gpl" to your configure command

[编辑] 3GP Support

If you want to enable 3GP support you have to add the AMR audio codec. Download the TS26.104 REL-5 V5.1.0 26104-5??.zip here. Extract the codec into libavcodec/amr_float and add "--enable-amr_nb" to your configure command

[编辑] XVID Support

(thanks to garvin.thornten at datel.co.uk) Download and install the codec from www.xlib.org (see xvidcore-xxxx/doc/install). Add "--enable-xvid --enable-gpl" to your configure command. When compiling with xvid codec in MinGW or cygwin you will get a "mkstemp" error when compiling "xvidff.c". To fix this edit "libavcodec/xvidff.c" and add the following after the #includes. This will probably be fixed in a future ffmpeg release: '

```
/* Added for windows compile ---------- */
#include

int xvid_ff_2pass(void *ref, int opt, void *p1, void *p2); void xvid_correct_framerate(AVCodecContext *avctx);

int mkstemp(char* template)

char temppath[512];
if(GetTempPath(512,temppath)!=0)
{
  if(GetTempFileName(temppath,"fil",0,template)!=0)
  {
    FILE *pFile;
    pFile=fopen(template,"w+");
    if(pFile!=NULL)
    return (int)pFile;
  }
}
  return -1;
}
/* ------------------------------------------ */
```

Link about qscale removed "-f singlejpeg" as its identical to "-f mjpeg"

4 Ffmpeg 快速命令使用

4.1Ffmpeg 使用语法

ffmpeg [[options][`-i' input_file]]... {[options] output_file}...
如果没有输入文件，那么视音频捕捉（只在 Linux 下有效，因为 Linux 下把音视频设备当作文件句柄来处理）就会起作用。作为通用的规则，选项一般用于下一个特定的文件。如果你给 –b 64 选项，改选会设置下一个视频速率。对于原始输入文件，格式选项可能是需要的。缺省情况下，ffmpeg 试图尽可能的无损转换，采用与输入同样的音频视频参数来输出。

ffmpeg 转换所涉及到的选项较多，可参考 ffmpeg 选项详解。

ffmpeg 支持多种文件格式和多种音频、视频编码器，可参考 ffmepg 格式详解，（附：常见视频文件格式详解）

4.2 视频文件截图

截取一张 352x240 尺寸大小的，格式为 jpg 的图片
ffmpeg -i test.asf -y -f image2 -t 0.001 -s 352x240 a.jpg
把视频的前 3 0 帧转换成一个 Animated Gif
ffmpeg -i test.asf -vframes 30 -y -f gif a.gif
截取指定时间的缩微图
ffmpeg -i test.avi -y -f image2 -ss 8 -t 0.001 -s 350x240 test.jpg
-ss 后跟的时间单位为秒
转换文件为 3GP 格式
ffmpeg -y -i test.mpeg -bitexact -vcodec h263 -b 128 -r 15 -s 176x144
-acodec aac -ac 2 -ar 22500 -ab 24 -f 3gp test.3gp
或
ffmpeg -y -i test.wmv -ac 1 -acodec libamr_nb -ar 8000 -ab 12200 -s 176x144 -b 128 -r 15 test.3gp
[编辑] 视频格式转换

4.3 如何使用 ffmpeg 编码得到高质量的视频

ffmpeg.exe -i "D:\Video\Fearless\Fearless.avi" -target film-dvd -s 720x352
-padtop 64 -padbottom 64 -maxrate 7350000 -b 3700000 -sc_threshold 1000000000
-trellis -cgop -g 12 -bf 2 -qblur 0.3 -qcomp 0.7 -me full -dc 10 -mbd 2
-aspect 16:9 -pass 2 -passlogfile "D:\Video\ffmpegencode" -an -f mpeg2video "D:\Fearless.m2v"
转换指定格式文件到 FLV 格式
ffmpeg.exe -i test.mp3 -ab 56 -ar 22050 -b 500 -r 15 -s 320x240 f:\test.flv
ffmpeg.exe -i test.wmv -ab 56 -ar 22050 -b 500 -r 15 -s 320x240 f:\test.flv

转码解密的 VOB

ffmpeg -i snatch_1.vob -f avi -vcodec mpeg4 -b 800 -g 300 -bf 2 -acodec mp3 -ab 128 snatch.avi

上面的命令行将 vob 的文件转化成 avi 文件，mpeg4 的视频和 mp3 的音频。注意命令中使用了 B 帧，所以 mpeg4 流是 divx5 兼容的。GOP 大小是 300 意味着 29.97 帧频下每 10 秒就有 INTRA 帧。该映射在音频语言的 DVD 转码时候尤其有用。

同时编码到几种格式并且在输入流和输出流之间建立映射

ffmpeg -i /tmp/a.wav -ab 64 /tmp/a.mp2 -ab 128 /tmp/b.mp2 -map 0:0 -map 0:0

上面的命令行转换一个 64Kbits 的 a.wav 到 128kbits 的 a.mp2 '-map file:index'在输出流的顺序上定义了哪一路输入流是用于每一个输出流的。

转换文件为 3GP 格式

ffmpeg -i test.avi -y -b 20 -s sqcif -r 10 -acodec amr_wb -ab 23.85 -ac 1 -ar 16000 test.3gp

注：如果要转换为 3GP 格式，则 ffmpeg 在编译时必须加上–enable-amr_nb –enable-amr_wb，详细内容可参考：转换视频为 3GPP 格式

转换文件为 MP4 格式（支持 iPhone/iTouch）

ffmpeg -y -i input.wmv -f mp4 -async 1-s 480x320 -acodec libfaac -vcodec libxvid -qscale 7 -dts_delta_threshold 1 output.mp4

ffmpeg -y -i source_video.avi input -acodec libfaac -ab 128000 -vcodec mpeg4 -b 1200000 -mbd 2 -flags +4mv+trell -aic 2 -cmp 2 -subcmp 2 -s 320x180 -title X final_video.mp4

将一段音频与一段视频混合

ffmpeg -i son.wav -i video_origine.avi video_finale.mpg

将一段视频转换为 DVD 格式

ffmpeg -i source_video.avi -target pal-dvd -ps 2000000000 -aspect 16:9 finale_video.mpeg

注： target pal-dvd : Output format ps 2000000000 maximum size for the output file, in bits (here, 2 Gb) aspect 16:9 : Widescreen

转换一段视频为 DivX 格式

ffmpeg -i video_origine.avi -s 320x240 -vcodec msmpeg4v2 video_finale.avi

Turn X images to a video sequence

ffmpeg -f image2 -i image%d.jpg video.mpg

注： This command will transform all the images from the current directory (named image1.jpg, image2.jpg, etc...) to a video file named video.mpg.

Turn a video to X images

ffmpeg -i video.mpg image%d.jpg

注：This command will generate the files named image1.jpg, image2.jpg, ...

The following image formats are also availables : PGM, PPM, PAM, PGMYUV, JPEG, GIF, PNG, TIFF, SGI.

4.4 使用 ffmpeg 录像屏幕

ffmpeg -vcodec mpeg4 -b 1000 -r 10 -g 300 -vd x11:0,0 -s 1024x768 ~/test.avi

：其中，-vd x11:0,0 指录制所使用的偏移为 x=0 和 y=0，-s 1024×768 指录制视频的大小为 1024×768。录制的视频文件为 test.avi，将保存到用户主目录中

如果你只想录制一个应用程序窗口或者桌面上的一个固定区域，那么可以指定偏移位置和区域大小。使用 xwininfo -frame 命令可以完成查找上述参数。

重新调整视频尺寸大小
ffmpeg -vcodec mpeg4 -b 1000 -r 10 -g 300 -i ~/test.avi -s 800×600 ~/test-800-600.avi
注：ffmpeg 的屏幕录制功能只能在 Linux 环境下有效。
[编辑] 视频采集

把摄像头的实时视频录制下来，存储为文件
ffmpeg -f video4linux -s 320*240 -r 10 -i /dev/video0 test.asf
更多信息可参考 ffmpeg 工程组论坛的讨论贴：有关 ffmpeg 的视频采集
注：ffmpeg 的视频采集功能只能在 Linux 环境下使用
[编辑] 使用 ffmpeg 压制 H.264 视频

ffmpeg -threads 4 -i INPUT -r 29.97 -vcodec libx264 -s 480x272 -flags +loop -cmp +chroma -deblockalpha 0 -deblockbeta 0 -crf 24
-bt 256k -refs 1 -coder 0 -me umh -me_range 16 -subq 5 -partitions +parti4x4+parti8x8+partp8x8 -g 250 -keyint_min 25 -level 30
-qmin 10 -qmax 51 -trellis 2 -sc_threshold 40 -i_qfactor 0.71 -acodec libfaac -ab 128k -ar 48000 -ac 2 OUTPUT
注：使用该指令可以压缩出比较清晰，而且文件转小的 H.264 视频文件
使用 VHook 为视频添加水印
[编辑] 使用 ffmpeg 获取 PCM 数据

ffmpeg -i input.mpg -f s16le -ar 44100 -acodec pcm_s16le output.pcm

5 Ffmpeg 快速应用开发

从这一步开始，这里放几个简单的例子，手把手知道初学者马上进入开发状态

从这一步开始，这里放几个简单的例子，手把手知道初学者马上进入开发状态

Ffmpeg 中的 Libavformat 和 libavcodec 库是访问大多数视频文件格式的一个很好的方法。不幸的是，在开发您自己的程序时，这套库基本上没有提供什么实际的文档可以用来作为参考（至少我没有找到任何文档），并且它的例程也并没有太多的帮助。

这种情况意味着，当我在最近某个项目中需要用到 libavformat/libavcodec 库时，需要作很多试验来搞清楚怎样使用它们。这里是我所学习的——希望我做的这些能够帮助一些人，以免他们重蹈我的覆辙，作同样的试验，遇到同样的错误。你还可以从这里下载一个 demo 程序。我将要公开的这部分代码需要 0.4.8 版本的 ffmpeg 库中的 libavformat/libavcodec 的支持（我正在写最新版本）。如果您发现以后的版本与我写的程序不能兼容，请告知我。

在这个文档里，我仅仅涉及到如何从文件中读入视频流；音频流使用几乎同样的方法可以工作的很好，不过，我并没有实际使用过它们，所以，我没于办法提供任何示例代码。

或许您会觉得奇怪，为什么需要两个库文件 libavformat 和 libavcodec ：许多视频文件格式（AVI 就是一个最好的例子）实际上并没有明确指出应该使用哪种编码来解析音频和视频数据；

它们只是定义了音频流和视频流（或者，有可能是多个音频视频流）如何被绑定在一个文件里面。这就是为什么有时候，当你打开了一个 AVI 文件时，你只能听到声音，却不能看到图象——因为你的系统没有安装合适的视频解码器。所以， libavformat 用来处理解析视频文件并将包含在其中的流分离出来， 而 libavcodec 则处理原始音频和视频流的解码。

打开视频文件： 首先第一件事情——让我们来看看怎样打开一个视频文件并从中得到流。我们要做的第一件事情就是初始化 libavformat/libavcodec:

av_register_all(); 这一步注册库中含有的所有可用的文件格式和编码器，这样当打开一个文件时，它们才能够自动选择相应的文件格式和编码器。要注意你只需调用一次 av_register_all()，所以，尽可能的在你的初始代码中使用它。如果你愿意，你可以仅仅注册个人的文件格式和编码，不过，通常你不得不这么做却没有什么原因。

下一步，打开文件： AVFormatContext *pFormatCtx; const char *filename="myvideo.mpg"; // 打开视频文件 if(av_open_input_file(&pFormatCtx, filename, NULL, 0, NULL)!=0)

```
    handle_error(); // 不能打开此文件
```

最后三个参数描述了文件格式，缓冲区大小（size）和格式参数；我们通过简单地指明 NULL 或 0 告诉 libavformat 去自动探测文件格式并且使用默认的缓冲区大小。请在你的程序中用合适的出错处理函数替换掉 handle_error()。 下一步，我们需要取出包含在文件中的流信息： // 取出流信息 if(av_find_stream_info(pFormatCtx)<0)

```
    handle_error(); // 不能够找到流信息
```

这一步会用有效的信息把 AVFormatContext 的流域（streams field）填满。作为一个可调试的诊断，我们会将这些信息全盘输出到标准错误输出中，不过你在一个应用程序的产品中并不用这么做： dump_format(pFormatCtx, 0, filename, false);

就像在引言中提到的那样，我们仅仅处理视频流，而不是音频流。为了让这件事情更容易理解，我们只简单使用我们发现的第一种视频流：

int i, videoStream; AVCodecContext *pCodecCtx; // 寻找第一个视频流 videoStream=-1; for(i=0; i<pFormatCtx->nb_streams; i++)

```
    if(pFormatCtx->streams->codec.codec_type==CODEC_TYPE_VIDEO)
    {
        videoStream=i;
        break;
    }
```

if(videoStream==-1)

```
    handle_error(); // Didn't find a video stream
```

// 得到视频流编码上下文的指针 pCodecCtx=&pFormatCtx->streams[videoStream]->codec;

好了，我们已经得到了一个指向视频流的称之为上下文的指针。但是我们仍然需要找到真正的编码器打开它。

AVCodec *pCodec;

// 寻找视频流的解码器 pCodec=avcodec_find_decoder(pCodecCtx->codec_id);

if(pCodec==NULL)

```
    handle_error(); // 找不到解码器
```

// 通知解码器我们能够处理截断的 bit 流——ie， // bit 流帧边界可以在包中

if(pCodec->capabilities & CODEC_CAP_TRUNCATED)

```
    pCodecCtx->flags|=CODEC_FLAG_TRUNCATED;
```

// 打开解码器 if(avcodec_open(pCodecCtx, pCodec)<0)

```
    handle_error(); // 打不开解码器
```

（那么什么是“截断 bit 流”？好的，就像一会我们看到的，视频流中的数据是被分割放入包中的。因为每个视频帧的数据的大小是可变的，那么两帧之间的边界就不一定刚好是包的边界。这里，我们告知解码器我们可以处理 bit 流。）

存储在 AVCodecContext 结构中的一个重要的信息就是视频帧速率。为了允许非整数的帧速率（比如 NTSC 的 29.97 帧），速率以分数的形式存储，分子在 pCodecCtx->frame_rate，分母在 pCodecCtx->frame_rate_base 中。在用不同的视频文件测试库时，我注意到一些编码器（很显然 ASF）似乎并不能正确的给予赋值（ frame_rate_base 用 1 代替 1000）。下面给出修复补丁：

// 加入这句话来纠正某些编码器产生的帧速错误 if(pCodecCtx->frame_rate>1000 && pCodecCtx->frame_rate_base==1)

```
    pCodecCtx->frame_rate_base=1000;
```

注意即使将来这个 bug 解决了，留下这几句话也并没有什么坏处。视频不可能拥有超过 1000fps 的帧速。

只剩下一件事情要做了：给视频帧分配空间以便存储解码后的图片：

AVFrame *pFrame;

pFrame=avcodec_alloc_frame();

就这样，现在我们开始解码这些视频。

解码视频帧 就像我前面提到过的，视频文件包含数个音频和视频流，并且他们各个独自被分开存储在固定大小的包里。我们要做的就是使用 libavformat 依次读取这些包，过滤掉所有那些视频流中我们不感兴趣的部分，并把它们交给 libavcodec 进行解码处理。在做这件事情时，我们要注意这样一个事实，两帧之间的边界也可以在包的中间部分。 听起来很复杂？幸运的是，我们在一个例程中封装了整个过程，它仅仅返回下一帧：

bool GetNextFrame(AVFormatContext *pFormatCtx, AVCodecContext *pCodecCtx,

```
    int videoStream, AVFrame *pFrame)
```

{

```
    static AVPacket packet;
    static int      bytesRemaining=0;
    static uint8_t  *rawData;
    static bool     fFirstTime=true;
    Int bytesDecoded;
    Int frameFinished;
```

// 我们第一次调用时，将 packet.data 设置为 NULL 指明它不用释放了

```
    if(fFirstTime)
    {
        fFirstTime=false;
        packet.data=NULL;
    }
```

// 解码直到成功解码完整的一帧

```
    while(true)
    {
         //  除非解码完毕，否则一直在当前包中工作
        while(bytesRemaining > 0)
```

```
        {
        //  解码下一块数据
            bytesDecoded=avcodec_decode_video(pCodecCtx, pFrame,
                &frameFinished, rawData, bytesRemaining);
                // 出错了？
            if(bytesDecoded < 0)
            {
                fprintf(stderr, "Error while decoding frame\n");
                return false;
            }
            bytesRemaining-=bytesDecoded;
            rawData+=bytesDecoded;
                // 我们完成当前帧了吗？接着我们返回
            if(frameFinished)
                return true;
        }
        // 读取下一包，跳过所有不属于这个流的包
        do
        {
            // 释放旧的包
            if(packet.data!=NULL)
                av_free_packet(&packet);
            // 读取新的包
            if(av_read_packet(pFormatCtx, &packet)<0)
                goto loop_exit;
        } while(packet.stream_index!=videoStream);
        bytesRemaining=packet.size;
        rawData=packet.data;
    }
```

loop_exit:

```
        // 解码最后一帧的余下部分
    bytesDecoded=avcodec_decode_video(pCodecCtx, pFrame,
&frameFinished,
        rawData, bytesRemaining);
        // 释放最后一个包
    if(packet.data!=NULL)
        av_free_packet(&packet);
    return frameFinished!=0;
```

}

现在，我们要做的就是在一个循环中，调用 GetNextFrame () 直到它返回 false。还有一处需要注意：大多数编码器返回 YUV 420 格式的图片（一个亮度和两个色度通道，色度通道只占亮度通道空间分辨率的一半（译者注：此句原句为 the chrominance channels samples at half the spatial resolution of the luminance channel））。看你打算如何对视频数据处理，或许你打算将它转换至 RGB 格式。（注意，尽管，如果你只是打算显示视频数据，那大可不必要这么做；查看一下 X11 的 Xvideo 扩展，它可以在硬件层进行 YUV 到 RGB 转换。）幸运的是，libavcodec 提供给我们了一个转换例程 img_convert ，它可以像转换其他图象进行 YUV 和 RGB 之间的转换。这样解码视频的循环就变成这样：

```
while(GetNextFrame(pFormatCtx, pCodecCtx, videoStream, pFrame)) {
    img_convert((AVPicture *)pFrameRGB, PIX_FMT_RGB24,
(AVPicture*)pFrame,
        pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);
    // 处理视频帧（存盘等等）
    DoSomethingWithTheImage(pFrameRGB);
}
```

RGB 图象 pFrameRGB （AVFrame *类型）的空间分配如下：

```
AVFrame *pFrameRGB; int numBytes; uint8_t *buffer;
// 分配一个 AVFrame 结构的空间 pFrameRGB=avcodec_alloc_frame();
if(pFrameRGB==NULL)
    handle_error();
// 确认所需缓冲区大小并且分配缓冲区空间 numBytes=avpicture_get_size(PIX_FMT_RGB24,
pCodecCtx->width,
    pCodecCtx->height);
buffer=new uint8_t[numBytes];
// 在 pFrameRGB 中给图象位面赋予合适的缓冲区 avpicture_fill((AVPicture *)pFrameRGB,
buffer, PIX_FMT_RGB24,
    pCodecCtx->width, pCodecCtx->height);
```

清除 好了，我们已经处理了我们的视频，现在需要做的就是清除我们自己的东西：// 释放 RGB 图象 delete [] buffer; av_free(pFrameRGB);

// 释放 YUV 帧 av_free(pFrame);

// 关闭解码器（codec） avcodec_close(pCodecCtx);

// 关闭视频文件 av_close_input_file(pFormatCtx);

完成！ 更新（2005 年 4 月 26 号）：有个读者提出：在 Kanotix （一个 Debian 的发行版）上面编译本例程，或者直接在 Debian 上面编译，头文件中 avcodec.h 和 avformat.h 需要加上前缀“ffmpeg”，就像这样：

1. include <ffmpeg/avcodec.h>

2. include <ffmpeg/avformat.h>

同样的， libdts 库在编译程序时也要像下面这样加入进来：

g++ -o avcodec_sample.0.4.9 avcodec_sample.0.4.9.cpp -lavformat -lavcodec -ldts -lz

几个月前，我写了一篇有关使用 ffmpeg 下 libavformat 和 libavcodec 库的文章。从那以来，我收到过一些评论，并且新的 ffmpeg 预发行版(0.4.9-pre1) 最近也要出来了，增加了对在视频文件中定位的支持，新的文件格式，和简单的读取视频帧的接口。这些改变不久就会应用到 CVS 中，不过这次是我第一次在发行版中看到它们。（顺便感谢 Silviu Minut 共享长时间学习 CVS 版的 ffmpeg 的成果－－他的有关 ffmpeg 的信息和 demo 程序在这里。）

在这篇文章里，我仅仅会描述一下以前的版本(0.4.8)和最新版本之间的区别，所以，如果你是采用新的 libavformat / libavcodec ，我建议你读前面的文章。

首先，说说有关编译新发行版吧。用我的编译器（SuSE 上的 gcc 3.3.1），在编译源文件 ffv1.c 时会报一个编译器内部的错误。我怀疑这是个精简版的 gcc－－我在编译 OpenCV 时也遇到了同样的事情－－但是不论如何，一个快速的解决方法就是在编译此文件时不要加优化参数。最简单的方法就是作一个 make，当编译时遇到编译器错误，进入 libavcodec 子目录（因为这也是 ffv1.c 所在之处），在你的终端中使用 gcc 命令去编译 ffv1.c，粘贴，编辑删除编译器开关（译者注：就是参数）"-O3"，然后使用那个命令运行 gcc。然后，你可以变回 ffmpeg 主目录并且重新运行 make，这次应该可以编译了。

都有哪些更新？ 有那些更新呢？从一个程序员的角度来看，最大的变化就是尽可能的简化了从视频文件中读取个人的视频帧的操作。在 ffmpeg 0.4.8 和其早期版本中，在从一个视频文件中的包中用例程 av_read_packet()来读取数据时，一个视频帧的信息通常可以包含在几个包里，而另情况更为复杂的是，实际上两帧之间的边界还可以存在于两个包之间。幸亏 ffmpeg 0.4.9 引入了新的叫做 av_read_frame()的例程,它可以从一个简单的包里返回一个视频帧包含的所有数据。使用 av_read_packet()读取视频数据的老办法仍然支持，但是不赞成使用－－我说：摆脱它是可喜的。

这里让我们来看看如何使用新的 API 来读取视频数据。在我原来的文章中（与 0.4.8 API 相关），主要的解码循环就像下面这样：

```
while(GetNextFrame(pFormatCtx, pCodecCtx, videoStream, pFrame)){
    img_convert((AVPicture *)pFrameRGB, PIX_FMT_RGB24,
(AVPicture*)pFrame,
        pCodecCtx->pix_fmt, pCodecCtx->width, pCodecCtx->height);
    // 处理视频帧（存盘等等）
    DoSomethingWithTheImage(pFrameRGB);
}
```

GetNextFrame() 是个有帮助的例程，它可以处理这样一个过程，这个过程汇编一个完整的视频帧所需要的所有的包。新的 API 简化了我们在主循环中实际直接读取和解码数据的操作：

```
while(av_read_frame(pFormatCtx, &packet)>=0){
    // 这是视频流中的一个包吗？
    if(packet.stream_index==videoStream)
    {
        // 解码视频流
        avcodec_decode_video(pCodecCtx, pFrame, &frameFinished,
            packet.data, packet.size);
        // 我们得到一帧了吗？
        if(frameFinished)
        {
            // 把原始图像转换成 RGB
            img_convert((AVPicture *)pFrameRGB, PIX_FMT_RGB24,
                (AVPicture*)pFrame, pCodecCtx->pix_fmt,
pCodecCtx->width,
                pCodecCtx->height);
            // 处理视频帧（存盘等等）
            DoSomethingWithTheImage(pFrameRGB);
        }
    }
    // 释放用 av_read_frame 分配空间的包
    av_free_packet(&packet);
}
```

看第一眼，似乎看上去变得更为复杂了。但那仅仅是因为这块代码做的都是要隐藏在 GetNextFrame()例程中实现的（检查包是否属于视频流，解码帧并释放包）。总的说来，因为我们能够完全排除 GetNextFrame ()，事情变得更简单了。 我已经更新了 demo 程序使用最新的 API。简单比较一下行数（老版本 222 行 Vs 新版本 169 行）显示出新的 API 大大的简化了这件事情。

0.4.9 的另一个重要的更新是能够在视频文件中定位一个时间戳。它通过函数 av_seek_frame() 来实现，此函数有三个参数：一个指向 AVFormatContext 的指针，一个流索引和定位时间戳。此函数在给定时间戳以前会去定位第一个关键帧。所有这些都来自于文档。我并没有对 av_seek_frame()进行测试，所以这里我并不能够给出任何示例代码。如果你成功的使用 av_seek_frame() ，我很高兴听到这个消息。

捕获视频(Video4Linux and IEEE1394) Toru Tamaki 发给我了一些使用 libavformat / libavcodec 库从 Video4Linux 或者 IEEE1394 视频设备源中抓捕视频帧的样例代码。对 Video4Linux,调用 av_open_input_file() 函数应该修改如下： AVFormatParameters formatParams; AVInputFormat *iformat;

formatParams.device = "/dev/video0"; formatParams.channel = 0; formatParams.standard = "ntsc"; formatParams.width = 640; formatParams.height = 480; formatParams.frame_rate = 29; formatParams.frame_rate_base = 1; filename = ""; iformat = av_find_input_format("video4linux");

av_open_input_file(&ffmpegFormatContext,

```
                filename, iformat, 0, &formatParams);
```

For IEEE1394, call av_open_input_file() like this:

AVFormatParameters formatParams; AVInputFormat *iformat;

formatParams.device = "/dev/dv1394"; filename = ""; iformat = av_find_input_format("dv1394");

av_open_input_file(&ffmpegFormatContext,

```
                filename, iformat, 0, &formatParams);
```

继续。。。 如果我碰巧遇到了一些有关 libavformat / libavcodec 的有趣的信息，我计划在这里公布。所以，如果你有任何的评论，请通过这篇文章顶部给出的地址联系我。 标准弃权：我没有责任去纠正这些代码的功能和这篇文章中涉及的技术。

注：本文为译文，与原文相比，遗漏了获取媒体文件信息相关部分，完整的信息请参考原文： http://www.inb.uni-luebeck.de/~boehme/using_libavcodec.html ，感谢 wotobo 在论坛：http://bbs.chinavideo.org/viewthread.php?tid=3312&extra=page%3D1 指正该问题。

6 Ffmpeg 编译详解

针对 ffmpeg 无论是 win 还是 linux 下编译都会出现很多问题，因此这里设立编译选项，力图构建一个标准化编译过程，使初学者快速消除对应用 ffmpeg 的恐惧感，至于其中遇到的问题，则放到下面一个选项 FAQ 里

Linux

查看 configure 帮助

cd ffmpeg

./configure --help 或着生一个文本文件好以参照./configure --help > ffmpegcfg.txt

选项列表

Usage: configure [options] Options: [defaults in brackets after descriptions]

Standard options:

```
  --help                   print this message
  --log[=FILE|yes|no]      log tests and output to FILE [config.err]
  --prefix=PREFIX          install in PREFIX [/usr/local]
  --libdir=DIR             install libs in DIR [PREFIX/lib]
  --shlibdir=DIR           install shared libs in DIR [PREFIX/lib]
  --incdir=DIR             install includes in DIR
[PREFIX/include/ffmpeg]
  --mandir=DIR             install man page in DIR [PREFIX/man]
  --enable-static          build static libraries [default=yes]
  --disable-static         do not build static libraries [default=no]
  --enable-shared          build shared libraries [default=no]
  --disable-shared         do not build shared libraries [default=yes]
  --enable-gpl             allow use of GPL code, the resulting libav*
                           and ffmpeg will be under GPL [default=no]
  --enable-pp              enable GPLed postprocessing support
[default=no]
  --enable-swscaler        software scaler support [default=no]
  --enable-beosthreads     use BeOS threads [default=no]
  --enable-pthreads        use pthreads [default=no]
```

```
  --enable-w32threads      use Win32 threads [default=no]
  --enable-x11grab         enable X11 grabbing [default=no]
```

External library support:

```
  --enable-sunmlib         use Sun medialib [default=no]
  --enable-dc1394          enable IIDC-1394 grabbing using libdc1394
                           and libraw1394 [default=no]
  --enable-liba52          enable GPLed liba52 support [default=no]
  --enable-liba52bin       open liba52.so.0 at runtime [default=no]
  --enable-avisynth        allow reading AVISynth script files
[default=no]
  --enable-libamr-nb       enable libamr-nb floating point audio codec
  --enable-libamr-wb       enable libamr-wb floating point audio codec
  --enable-libfaac         enable FAAC support via libfaac [default=no]
  --enable-libfaad         enable FAAD support via libfaad [default=no]
  --enable-libfaadbin      open libfaad.so.0 at runtime [default=no]
  --enable-libgsm          enable GSM support via libgsm [default=no]
  --enable-libmp3lame      enable MP3 encoding via libmp3lame
[default=no]
  --enable-libnut          enable NUT (de)muxing via libnut,
                           native demuxer exists [default=no]
  --enable-libogg          enable Ogg muxing via libogg [default=no]
  --enable-libtheora       enable Theora encoding via libtheora
[default=no]
  --enable-libvorbis       enable Vorbis en/decoding via libvorbis,
                           native implementations exist [default=no]
  --enable-libx264         enable H.264 encoding via x264 [default=no]
  --enable-libxvid         enable Xvid encoding via xvidcore,
                           native MPEG-4/Xvid encoder exists
[default=no]
```

Advanced options (experts only):

```
  --source-path=PATH       path to source code [/root/ffmpeg]
  --cross-prefix=PREFIX    use PREFIX for compilation tools []
  --cross-compile          assume a cross-compiler is used
  --target-os=OS           compiler targets OS [linux]
  --cc=CC                  use C compiler CC [gcc]
  --make=MAKE              use specified make [make]
  --extra-cflags=ECFLAGS   add ECFLAGS to CFLAGS []
  --extra-ldflags=ELDFLAGS add ELDFLAGS to LDFLAGS []
```

```
  --extra-libs=ELIBS       add ELIBS []
  --build-suffix=SUFFIX    suffix for application specific build []
  --arch=ARCH              select architecture  [i686]
  --cpu=CPU                selects the minimum cpu required (affects
                           instruction selection, may crash on older
CPUs)
  --enable-powerpc-perf    enable performance report on PPC
                           (requires enabling PMC)
  --disable-mmx            disable MMX usage
  --disable-armv5te        disable armv5te usage
  --disable-armv6          disable armv6 usage
  --disable-iwmmxt         disable iwmmxt usage
  --disable-altivec        disable AltiVec usage
  --disable-audio-oss      disable OSS audio support [default=no]
  --disable-audio-beos     disable BeOS audio support [default=no]
  --disable-v4l            disable video4linux grabbing [default=no]
  --disable-v4l2           disable video4linux2 grabbing [default=no]
  --disable-bktr           disable bktr video grabbing [default=no]
  --disable-dv1394         disable DV1394 grabbing [default=no]
  --disable-network        disable network support [default=no]
  --disable-ipv6           disable ipv6 support [default=no]
  --disable-zlib           disable zlib [default=no]
  --disable-vhook          disable video hooking support
  --disable-debug          disable debugging symbols
  --disable-mpegaudio-hp   faster (but less accurate)
                           MPEG audio decoding [default=no]
  --disable-ffmpeg         disable ffmpeg build
  --disable-ffserver       disable ffserver build
  --disable-ffplay         disable ffplay build
  --enable-small           optimize for size instead of speed
  --enable-memalign-hack   emulate memalign, interferes with memory
debuggers
  --disable-encoder=NAME   disables encoder NAME
  --enable-encoder=NAME    enables encoder NAME
  --disable-decoder=NAME   disables decoder NAME
  --enable-decoder=NAME    enables decoder NAME
  --disable-encoders       disables all encoders
  --disable-decoders       disables all decoders
  --disable-muxer=NAME     disables muxer NAME
  --enable-muxer=NAME      enables muxer NAME
  --disable-muxers         disables all muxers
  --disable-demuxer=NAME   disables demuxer NAME
  --enable-demuxer=NAME    enables demuxer NAME
```

```
  --disable-demuxers       disables all demuxers
  --enable-parser=NAME     enables parser NAME
  --disable-parser=NAME    disables parser NAME
  --disable-parsers        disables all parsers
  --enable-bsf=NAME        enables bitstream filter NAME
  --disable-bsf=NAME       disables bitstream filter NAME
  --disable-bsfs           disables all bitstream filters
  --enable-protocol=NAME   enables protocol NAME
  --disable-protocol=NAME  disables protocol NAME
  --disable-protocols      disables all protocols
  --list-decoders          show all available decoders
  --list-encoders          show all available encoders
  --list-muxers            show all available muxers
  --list-demuxers          show all available demuxers
  --list-parsers           show all available parsers
  --list-protocols         show all available protocols
  --list-bsfs              show all available bitstream filters
```

Developer options (useful when working on FFmpeg itself):

```
  --enable-gprof           enable profiling with gprof []
  --disable-opts           disable compiler optimizations
  --enable-extra-warnings  enable more compiler warnings
  --disable-strip          disable stripping of executables and shared
libraries
```

NOTE: Object files are built at the place where configure is launched.

开发人员注意选项

make 之后会生成以下文件

- ffmpeg, ffplay, ffserver(不带调试信息)
- ffmpeg_g, ffplay_g(带调试信息)
  - xxx_g 文件可以用 gdb(ddd)来调试
  - 具体我也不是很清楚,以前我经常用 ffplay 来调试,我另加了 configer 选项,现在忘了.请知道者补充.

7 Ffmpeg 编译 FAQ 集

7.1 Configure 过程出错

我想要做的是交叉编译 ffmpeg 我先交叉编译了 yasm 后有交叉编译了 x264 它们都交叉编译安装在了/opt/ffmpeg/目录下。下面就是交叉编译 ffmpeg 我的 configure 如下：./configure --prefix=/opt/ffmpeg/ --enable-libx264 --enable-gpl --cross-compile --cross-prefix=/usr/local/arm/3.4.1/bin/arm-linux- --cc=gcc --disable-ffserver --enable-ffplay --disable-encoders --arch=armv4l --disable-opts --disable-mmx 提示：ERROR：x264 not found. 可是 x264 已经装过了啊。我看了 config.err 提示说 x264.h： no such file or directory. 我进到 /opt/ffmpeg/include/ 发现有 x264.h 这个文件啊！不知道为何提示说找不到 x264?? 有谁做过请给些指导！感激。

参考： ./configure --xxxx, --extra-cflags=-I/opt/ffmpeg/include --extra-ldflags=-L/opt/ffmpeg/lib

7.2Make 过程出错

/usr/ffmpeg/ffplay.c:2451: undefined reference to `XOpenDisplay' /usr/ffmpeg/ffplay.c:2455: undefined reference to `XCloseDisplay'

Makefile needs option -lX11

编译 ffplay.c 是需要 SDL 库的！

编译最新的 ffmpeg 添加 pThread 库支持时出错的解决方案

7.3 如何编译 FFServer

我知道怎样编译 ffmpeg，但是编译以后我根本看不见 ffserver 我的编译命令是：

```
./configure --enable-memalign-hack --enable-mingw32
make
```

但是编译后只能看见这三个.EXE 文件

```
ffmpeg.exe,ffmpeg_g.exe,out_example.exe
```

请问一下，是不是要编译 ffserver，还需要另外设置其它的参数啊？

默认的情况下是打开的，只有加了

```
--disable-ffserver --disable-network --disable-protocols
--disable-muxers
```

等选项后才会不编译 ffserver

那为什么我在默认的情况下，编译以后得不到 ffserver 呢？

the simplest answer is cygwin,not mingw FFserver is OK on cygwin

if you keep on with mingw32 try the following patch(dont based on CVS today)--posted by a guy on FFmpeg maillist:

```
--- ffmpeg_orig/configure        2006-03-28 17:36:07.000000000 -0600
+++ ffmpeg/configure        2006-03-31 08:37:14.000000000 -0600
@@ -922,7 +922,8 @@
    dv1394="no"
    dc1394="no"
    ffserver="no"
-    network="no"
+    #network="no"
+    extralibs="$extralibs -lws2_32"
if test "$mingwce" = "yes"; then
    protocols="no"
fi
diff -Naur ffmpeg_orig/libavformat/http.c ffmpeg/libavformat/http.c
--- ffmpeg_orig/libavformat/http.c        2006-01-12
16:43:23.000000000 -0600
+++ ffmpeg/libavformat/http.c        2006-03-31 13:13:56.000000000
-0600
@@ -19,14 +19,16 @@
#include "avformat.h"
#include <unistd.h>
#include <sys/types.h>
+#ifndef __MINGW32__
#include <sys/socket.h>
#include <netinet/in.h>
```

```
-#ifndef __BEOS__
-# include <arpa/inet.h>
-#else
+#include <netdb.h>
+#endif
+#ifdef __BEOS__
# include "barpainet.h"
+#elif !defined(__MINGW32__)
+# include <arpa/inet.h>
#endif
-#include <netdb.h>


/* XXX: POST protocol is not completly implemented because ffmpeg use
diff -Naur ffmpeg_orig/libavformat/os_support.c
ffmpeg/libavformat/os_support.c
--- ffmpeg_orig/libavformat/os_support.c        2006-01-22
18:57:59.000000000 -0600
+++ ffmpeg/libavformat/os_support.c        2006-03-31
09:40:29.000000000 -0600
@@ -32,6 +32,9 @@
#include <sys/time.h>
#endif
#include <time.h>
+#ifdef __MINGW32__
+#include <winsock.h>
+#endif

/**
 * gets the current time in micro seconds.
@@ -65,3 +68,30 @@
}
#endif /* !defined(HAVE_LOCALTIME_R) */
#endif /* !defined(CONFIG_WINCE) */
+
+#ifdef __MINGW32__
+int init_winsock()
+{
+WSADATA wsaData;
+WORD wVersionRequested=MAKEWORD(1,1);
+int Win32isStupid;
+
+  Win32isStupid=WSAStartup(wVersionRequested, &wsaData);
```

```
+  if (Win32isStupid) return -1;
+
+  return 0;
+}
+
+int inet_aton(const char *hostname, struct in_addr *sin_addr)
+{
+    sin_addr->s_addr=inet_addr(hostname);
+    if (sin_addr->s_addr == INADDR_NONE) {
+      return 0;
+    }
+
+    return -1;
+}
+#endif
+
+
+
diff -Naur ffmpeg_orig/libavformat/os_support.h
ffmpeg/libavformat/os_support.h
--- ffmpeg_orig/libavformat/os_support.h        2004-04-24
06:51:38.000000000 -0500
+++ ffmpeg/libavformat/os_support.h        2006-03-31
13:10:17.000000000 -0600
@@ -13,7 +13,18 @@
#ifdef __MINGW32__
__declspec(dllimport) void __stdcall Sleep(unsigned long
dwMilliseconds);
// #  include <windows.h>
+#define GUID microsuck_GUID
+#include <winsock.h>
+#undef GUID
#  define usleep(t)    Sleep((t) / 1000)
+#  define sleep(t)     Sleep((t) * 1000)
+typedef int socklen_t;
+#define O_NONBLOCK FIONBIO
+#define fcntl(fd,b,c) { u_long arg=1L; \
+                        ioctlsocket(fd, c, &arg); }
+// #define EINPROGRESS WSAEINPROGRESS
+#define EINPROGRESS 0
+int init_winsock();
#endif
```

```
#ifdef __BEOS__
diff -Naur ffmpeg_orig/libavformat/rtp.c ffmpeg/libavformat/rtp.c
--- ffmpeg_orig/libavformat/rtp.c        2006-03-30 10:44:32.000000000
-0600
+++ ffmpeg/libavformat/rtp.c        2006-03-31 13:14:03.000000000
-0600
@@ -22,14 +22,16 @@

#include <unistd.h>
#include <sys/types.h>
+#ifndef __MINGW32__
#include <sys/socket.h>
#include <netinet/in.h>
-#ifndef __BEOS__
-# include <arpa/inet.h>
-#else
+#include <netdb.h>
+#endif
+#ifdef __BEOS__
# include "barpainet.h"
+#elif !defined(__MINGW32__)
+# include <arpa/inet.h>
#endif
-#include <netdb.h>

//#define DEBUG

diff -Naur ffmpeg_orig/libavformat/rtpproto.c
ffmpeg/libavformat/rtpproto.c
--- ffmpeg_orig/libavformat/rtpproto.c        2006-01-12
16:43:25.000000000 -0600
+++ ffmpeg/libavformat/rtpproto.c        2006-03-31 13:14:08.000000000
-0600
@@ -21,14 +21,16 @@
#include <unistd.h>
#include <stdarg.h>
#include <sys/types.h>
+#ifndef __MINGW32__
#include <sys/socket.h>
#include <netinet/in.h>
-#ifndef __BEOS__
-# include <arpa/inet.h>
-#else
```

```
+#include <netdb.h>
+#endif
+#ifdef __BEOS__
# include "barpainet.h"
+#elif !defined(__MINGW32__)
+# include <arpa/inet.h>
#endif
-#include <netdb.h>
#include <fcntl.h>

#define RTP_TX_BUF_SIZE  (64 * 1024)
diff -Naur ffmpeg_orig/libavformat/rtsp.c ffmpeg/libavformat/rtsp.c
--- ffmpeg_orig/libavformat/rtsp.c        2006-03-10
18:22:21.000000000 -0600
+++ ffmpeg/libavformat/rtsp.c        2006-03-31 13:14:12.000000000
-0600
@@ -20,12 +20,14 @@

#include <unistd.h> /* for select() prototype */
#include <sys/time.h>
+#ifndef __MINGW32__
#include <netinet/in.h>
#include <sys/socket.h>
-#ifndef __BEOS__
-# include <arpa/inet.h>
-#else
+#endif
+#ifdef __BEOS__
# include "barpainet.h"
+#elif !defined(__MINGW32__)
+# include <arpa/inet.h>
#endif

//#define DEBUG
diff -Naur ffmpeg_orig/libavformat/tcp.c ffmpeg/libavformat/tcp.c
--- ffmpeg_orig/libavformat/tcp.c        2006-02-02 07:07:30.000000000
-0600
+++ ffmpeg/libavformat/tcp.c        2006-03-31 13:06:43.000000000
-0600
@@ -19,17 +19,19 @@
#include "avformat.h"
#include <unistd.h>
#include <sys/types.h>
```

```
+#ifndef __MINGW32__
#include <sys/socket.h>
#include <netinet/in.h>
+#include <netdb.h>
+#endif
#if defined(__BEOS__) || defined(__INNOTEK_LIBC__)
typedef int socklen_t;
#endif
-#ifndef __BEOS__
-# include <arpa/inet.h>
-#else
+#ifdef __BEOS__
# include "barpainet.h"
+#elif !defined (__MINGW32__)
+# include <arpa/inet.h>
#endif
-#include <netdb.h>
#include <sys/time.h>
#include <fcntl.h>

@@ -77,6 +79,10 @@
    if (port <= 0 || port >= 65536)
        goto fail;

+#ifdef __MINGW32__
+    init_winsock();
+#endif
+
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(port);
    if (resolve_host(&dest_addr.sin_addr, hostname) < 0)
@@ -147,11 +153,7 @@
        tv.tv_usec = 100 * 1000;
        ret = select(fd_max + 1, &rfds, NULL, NULL, &tv);
        if (ret > 0 && FD_ISSET(s->fd, &rfds)) {
-#ifdef __BEOS__
            len = recv(s->fd, buf, size, 0);
-#else
-            len = read(s->fd, buf, size);
-#endif
            if (len < 0) {
                if (errno != EINTR && errno != EAGAIN)
#ifdef __BEOS__
```

```
@@ -184,11 +186,7 @@
        tv.tv_usec = 100 * 1000;
        ret = select(fd_max + 1, NULL, &wfds, NULL, &tv);
        if (ret > 0 && FD_ISSET(s->fd, &wfds)) {
-#ifdef __BEOS__
            len = send(s->fd, buf, size, 0);
-#else
-            len = write(s->fd, buf, size);
-#endif
            if (len < 0) {
                if (errno != EINTR && errno != EAGAIN) {
#ifdef __BEOS__
@@ -211,7 +209,7 @@
static int tcp_close(URLContext *h)
{
    TCPContext *s = h->priv_data;
-#ifdef CONFIG_BEOS_NETSERVER
+#if defined(CONFIG_BEOS_NETSERVER) || defined(__MINGW32__)
    closesocket(s->fd);
#else
    close(s->fd);
diff -Naur ffmpeg_orig/libavformat/udp.c ffmpeg/libavformat/udp.c
--- ffmpeg_orig/libavformat/udp.c        2006-01-12 16:43:25.000000000
-0600
+++ ffmpeg/libavformat/udp.c        2006-03-31 13:06:54.000000000
-0600
@@ -19,14 +19,16 @@
#include "avformat.h"
#include <unistd.h>
#include <sys/types.h>
+#ifndef __MINGW32__
#include <sys/socket.h>
+#include <netdb.h>
#include <netinet/in.h>
-#ifndef __BEOS__
-# include <arpa/inet.h>
-#else
+#endif
+#ifdef __BEOS__
# include "barpainet.h"
+#elif !defined(__MINGW32__)
+# include <arpa/inet.h>
#endif
```

```
-#include <netdb.h>

#ifndef IPV6_ADD_MEMBERSHIP
#define IPV6_ADD_MEMBERSHIP IPV6_JOIN_GROUP
@@ -211,7 +213,7 @@

 fail:
     if (udp_fd >= 0)
-#ifdef CONFIG_BEOS_NETSERVER
+#if defined(CONFIG_BEOS_NETSERVER) || defined(__MINGW32__)
         closesocket(udp_fd);
 #else
         close(udp_fd);
@@ -357,6 +359,11 @@
     getsockname(udp_fd, (struct sockaddr *)&my_addr1, &len);
     s->local_port = ntohs(my_addr1.sin_port);

+#ifdef __MINGW32__
+    tmp=65536;   /* 64k UDP buffer size.  Should this be bigger? */
+    setsockopt(udp_fd, SOL_SOCKET, SO_RCVBUF, &tmp, sizeof(tmp));
+#endif
+
 #ifndef CONFIG_BEOS_NETSERVER
     if (s->is_multicast) {
         if (h->flags & URL_WRONLY) {
@@ -411,7 +418,7 @@
     return 0;
 fail:
     if (udp_fd >= 0)
-#ifdef CONFIG_BEOS_NETSERVER
+#if defined(CONFIG_BEOS_NETSERVER) || defined(__MINGW32__)
         closesocket(udp_fd);
 #else
         close(udp_fd);
@@ -471,7 +478,7 @@
 {
     UDPContext *s = h->priv_data;

-#ifndef CONFIG_BEOS_NETSERVER
+#if !defined(CONFIG_BEOS_NETSERVER) && !defined(__MINGW32__)
 #ifndef CONFIG_IPV6
     if (s->is_multicast && !(h->flags & URL_WRONLY)) {
         if (setsockopt(s->udp_fd, IPPROTO_IP, IP_DROP_MEMBERSHIP,
```

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

有关编译 FFServer 的讨论

7.4 提供 java+winwows 下使用 ffmpeg 解决视频转换思路和代码

提供一个在 java 开发环境下使用 ffmepg 开发视频共享网站上传视频转换技术的思路

如果你想在 java 中使用的话,你必须先采用 VC++等开发工具写一个 COM 或是 OCX 组件，通过这个组件调用 ffmpeg 提供的动态库，
这个组件是可以满足你的需求的，然后在 java 中你再把这个组件封装一下。
如果你想把 ffmpeg.exe 这个程序封装为一个 DLL，或是 COM 的话，难度很大，特别是你现在对 VC++不熟悉！
　　建议：你编写一个独立的转换系统，支持多线程、批处理……，可独立运行的，然后你的 java 程序通过一定的方式（如通过数
据库交换)向前面说的独立转换系统提交转换任务,转换系统接收到新的转换后,开始转换，同时写入转换进度到数据库中，这时
，你的 java 程序也可以从数据库中获取实时的转换进度了，转换系统转换完成之后，把结果也写入数据库，这样，你的 java 程序
也就知道转换完成了。这样做的好处是，可以实现分布式处理、多服务器并行处理、系统的负载均衡处理等，优势还是很明显的。
　　可以很明确说,大型一点的视频网站,都不是用户上传一个文件就马上在 WEB 进程中转换的，而是有一套如我上面所说的转换
系统，而且转换不一定是在 WEB 发布的服务器上进行的，有专门的转换服务器。

基本思路如下：

看了 admin 说的...的确很对...
我现在做的那个项目就有你这样的问题...
我也是直接写了一个类...用 asp.net 调用外部程序(ffmpeg.exe),转换,然后获取返回的信息...来写到数据库...
而这个类是用一条新的进程来执行的...网站主线程不需要等待...
而且这个类还会自动扫描数据库...只要后用户上传后,调用了这个类...他就会扫描一遍数据库,发现没转换的,循环转换...
这样转换和网站就可以脱离开了...
而我也明白你的意思...你是想做成一个 DLL 或者 COM...让网站主线程直接调用...把信息实时反馈给用户...

但这是不太可能的...因为网站不是本地程序...不可能跟外壳(网站就是指客户端页面,本地程序就是指窗体)实时交互...就算
用 ajax 也不行...因为后台处理有一个并发的问题...
我公司一台服务器,用最新架构的至强 3.0CPU...也最多能并行运行 10 个左右的 FFMPEG...超过了就会卡机...
所以 WEB 的后台转换必须是用队列来实现...一次转一个...这是最理想的...当然...这又要用到线程阻塞之类的东西...这就没
办法帮你了...呵呵...

代码详解:

我是用 C#写的...代码有啊...这论坛就有...
http://bbs.chinavideo.org/viewthread.php?tid=2326&extra=page%3D1

详细思路如下:

思路也不难...用户上传完毕...把上传后的文件名和路径等信息插到数据库...并标上未转换的标记(怎么实现见人见志)...
然后调用转换类(或者方法)...扫描数据库...做个 SELECT 就行了...得出数据集...
这要看你的思路了...
1.可以把全部未转换的 select 出来,然后用 for(XXX)来循环这个数据集
2.用 select top 1 加 while 来扫描...
第一种方法就安全一点,但在这个类(或者方法)执行(数据集取出来后)过程,再有用户上传,也不会被加入转换队列,需要等当前转
换队列完成后,再有用户上传,才会被加到转换队列...
第二种方法就比较危险一点,毕竟是用 while,只要中间逻辑出现任何错误,就有可能出现死循环...但是用 while 和 select top 1
的话,转换队列是非常灵活的...是取一条,转一个...也就是说,只要这个 while 没完成(就是说数据库中还有没被转换的,包括转换
开始后才增加的数据),他都会调出来加到队列转换...相对实时一点...
上面说的是数据库扫描的思路...
下面说说转换的安全性...
视频转换占用 CPU 严重人尽皆知,这个就没啥好说的...所以我推荐不并发...每次只转换 1 个视频...
那么你用开发语言(JAVA 也好,C#也好)调用外部程序(ffmpeg.exe)的时候...记得...一定要等待进程结束...不然的话...等着
死机吧...我试过做一个 while+select top1 的转换队列系统...忘记了加上等待进程结束的语句...结果 FFMPEG 那个线程被运
行了上百条...但其实我数据库只有 1 条未转换记录...但是只要没转换完成,他就不会把数据库的记录改写(改成已转换),所以
WHILE 不断的建立新线程来执行...

调用外部程序,系统会分配一条独立的线程序,这里只要加上等待线程结束的语句就好了...
另外,调用整个类(或者方法),也必须开一条单独的线程序,而且只能运行 1 个这样的线程序...
举个例...你的网站肯定不会只有 1 个别用户访问...那么的话...这个类(或者方法)有可能被调用多次...那么如果不限制执行这
个类(或者方法)的数量,那么就会产生多个类(或者方法)的镜像...同时执行...同时扫描数据库,同时开 1 条 FFMPEG 进程来转换同
1 份文件...最后的结果也是消耗无谓的资源...导致服务器挂掉...这也很好的说明 web 站和本地程序开发的很大不同...一个并发
量问题...
所以呢...这整个类(或者方法)你也要做保护装置...你可以用 1 个全局变量来保存一个状态...例如这个全局变量为 status...那
么每当这个类(或者方法)开始执行,就把 status 改成 1...然后在这个类(或者方法)外部再加一个判断:
```
if(status == 1) return;
```
这样就是一个简单的保护了...
复杂一点的方法呢...就是用一条线程来执行这个类(或者方法)...用阻塞来防止被多次运行这个类(或者方法)...至于实现的
办法...每种语言都不同...你自己找找方法吧...
这是我将要使用在我项目上的思路...呵呵...
最后,告诫一句...程序员...只有想不到,没有做不到...多多发散自己的思维...对你以后工作会有很大帮助...

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

有关 java+winwows 下使用 ffmpeg 解决视频转换技术的讨论

7.5 如何用 vc 顺利编译 ffmpeg

如何用 vc 顺利编译 ffmpeg

这是编译问题,而不是技术问题
C 语言熟练的话，在 VC 下修改完成编译问题不是很大。
最主要是把那些不符合规范的语句修改好。举两个例子：
1） c->time_base= (AVRational){1,25}; 修改为：
```
    c->time_base.num = 1;
   c->time_base.den = 25;
```
2) static PixFmtInfo pix_fmt_info[PIX_FMT_NB] = { ... } 里面的写法也不对。
...

VC 下编译根据各人的技术积累经验，难度不是很大，基本上
都是体力活。C 很熟的人把全部修改过来两到三天足够了。

ffmpeg 中除了包含 c 语言部分还有部分的汇编语言，所以 ffmpeg 在 VC 下顺利编译需要的外部编译器

- 1.GCC

我很久以前编译该项目的时候所使用的工具。
GCC 很好找，因为它是免费的工具。
GCC 主要用来编译 ffmpeg 的汇编代码。
ffmpeg 中的汇编代码使用 GCC 编译时的编译选项
gcc -c -O3 -march=i586 -mtune=i686 -fomit-frame-pointer -finline
-finline-functions
- DHAVE_AV_CONFIG_H -pipe -mno-cygwin -mdll -I. -I.. $(InputPath) -o
$(IntDir)\$(InputName).obj
-Ilibavutil

- 2.Intel C++

intel C++是 intel 开发的。它支持许多 C 语言的国际标准，
ffmpeg 项目中用的很多编码标准 c99 标准，而 Intel C++支持这些标准。

但是全部编译是会遇到一些问题的

这里是 ffmpeg 文档里关于 vc 编译器的说明
FFmpeg will not compile under Visual C++ -- and it has too many
dependencies on the GCC compiler
to make a port viable. However, if you want to use the FFmpeg libraries
in your own applications
, you can still compile those applications using Visual C++. An important
restriction to this
is that you have to use the dynamically linked versions of the FFmpeg
libraries (i.e. the DLLs),
and you have to make sure that Visual-C++-compatible import libraries are
created during the
FFmpeg build process.
意思就是说有些依赖 gcc 和汇编，vc 下比较困难，但是没必要在 vc 下编译整个库，只需要在 vc 下编译工程文件，然后调用 ffmpeg
的库就可以了。

例如：

ffdshow 项目中的 ffmpeg 工程可以用 vc 编译通过，如果是 vc6.0,需要手工改一些代码，vc2003 或是 vc2005 都没有问题。

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

有关如何用 vc 顺利编译 ffmpeg 的讨论

7.6FFMPEG 在 windows 下编译出错

FFMPEG 在 windows 下编译出错 错误如下：

用的是 Mingw＋Msys
因为想编译出 dll，按照 ffmpeg 的文档
Here are the step-by-step instructions for building the FFmpeg libraries so they can be used with Visual C++:
Install Visual C++ (if you haven't done so already).
Install MinGW and MSYS as described above.
Add a call to `vcvars32.bat' (which sets up the environment variables for the Visual C++ tools) as the first line of `msys.bat'. The standard location for `vcvars32.bat' is `C:\Program Files\Microsoft Visual Studio 8\VC\bin\vcvars32.bat', and the standard location for `msys.bat' is `C:\msys\1.0\msys.bat'. If this corresponds to your setup, add the following line as the first line of `msys.bat': call "C:\Program Files\Microsoft Visual Studio 8\VC\bin\vcvars32.bat"
Start the MSYS shell (file `msys.bat') and type link.exe. If you get a help message with the command line options of link.exe, this means your environment variables are set up correctly, the Microsoft linker is on the path and will be used by FFmpeg to create Visual-C++-compatible import libraries.
Extract the current version of FFmpeg and change to the FFmpeg directory.
Type the command ./configure --enable-shared --disable-static --enable-memalign-hack to configure and, if that didn't produce any errors, type make to build FFmpeg.
The subdirectories `libavformat', `libavcodec', and `libavutil' should now contain the files `avformat.dll', `avformat.lib', `avcodec.dll', `avcodec.lib', `avutil.dll', and `avutil.lib', respectively. Copy the three DLLs to your System32 directory (typically `C:\Windows\System32').
但是 make 之后出现如下错误

```
C:/DOCUME~1/Cale/LOCALS~1/Temp/ccKCbaaa.o(.text+0x3d2): In function
`main':
f:/Codec/ffmpeg_lh/ffmpeg/qt-faststart.c:152: undefined reference to
`fseeko64'
C:/DOCUME~1/Cale/LOCALS~1/Temp/ccKCbaaa.o(.text+0x516):f:/Codec/ffmpe
g_lh/ffmpeg/qt-faststart.c:164: undefined reference to
`fseeko64'
C:/DOCUME~1/Cale/LOCALS~1/Temp/ccKCbaaa.o(.text+0x51e):f:/Codec/ffmpe
g_lh/ffmpeg/qt-faststart.c:165: undefined reference to
`ftello64'
C:/DOCUME~1/Cale/LOCALS~1/Temp/ccKCbaaa.o(.text+0x6e5):f:/Codec/ffmpe
g_lh/ffmpeg/qt-faststart.c:247: undefined reference to
`fseeko64'
C:/DOCUME~1/Cale/LOCALS~1/Temp/ccKCbaaa.o(.text+0xdc7):f:/Codec/ffmpe
g_lh/ffmpeg/qt-faststart.c:133: undefined reference to
`fseeko64'
C:/DOCUME~1/Cale/LOCALS~1/Temp/ccKCbaaa.o(.text+0xdec):f:/Codec/ffmpe
g_lh/ffmpeg/qt-faststart.c:140: undefined reference to
`ftello64'
make: *** [qt-faststart.exe] Error 1
```

在 configure 的时候加入--enable-mingw32 也是同样的错误，换了一台机器还是同样错误，使用 4 月的 ffmpeg 版本和 6 月 14 日的版本还是同样错误。

解答如下：

应该还是环境的问题。如果不使用 qt-faststart 的话，最简单的办法，就是把这些 64 位的函数手工改成 32 位平台下的对应函数就行了。
换句话说就是没有大于 4g 的 quicktime 文件需要转换，用在 32 位的机器下面就可以采用以下方法：
例如：
出错是在 qt-faststart.c 里

```
#ifdef __MINGW32__
#define fseeko(x,y,z)  fseeko64(x,y,z)
#define ftello(x)      ftello64(x)
#endif
```

只需要把#define fseek(x,y,z) fseeko64(x,y,z)改成：#define fseek(x,y,z) fseek(x,y,z)

采用旧的编译环境编译最新的 ffmpeg 源代码时，会出现如下错误提示：

```
utils.o:utils.c:(.text+0x4949): undefined reference to `gettimeofday'
```

解决方案：

You need to update your mingw runtime and probably your w32api to the latest versions. 3.13 and 3.10 respectively as of writing.

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

有关的 FFMPEG 在 windows 下编译出错的讨论

7.7VC 下编译的几个小问题

问题一：

下面的 c 语句，如何修改，可以使之在 Vc 下编译通过？
return 0.0/0.0;

对应解答：

return INT64_MAX;
或者 return 0.0;

问题二：

下面的 c 语句，如何修改，可以使之在 Vc 下编译通过？
#define tprintf(...) av_log(NULL, AV_LOG_DEBUG, __VA_ARGS__)
中间的三个小点不支持。。。也就是不支持变长参数

对应解答：

```
#ifdef __GNUC__
    #define tprintf(...) {}
#else
    static void tprintf(char *msg,...){}
#endif
```

问题三：

句法修改的问题

```
AVInputFormat mpegts_demuxer =
    {
    "mpegts",
    "MPEG2 transport stream format",
    sizeof(MpegTSContext),
    mpegts_probe,
    mpegts_read_header,
    mpegts_read_packet,
    mpegts_read_close,
    read_seek,
    mpegts_get_pcr,
    .flags = AVFMT_SHOW_IDS,
    };
```

对应解答：

把该结构的其他值，都添上了，然后就直接

```
{
 .....
 ....
 .....
 AVFMT_SHOW_IDS
}
```

应该可以了。

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

有关 VC 下编译的几个小问题的讨论

7.8Ffmpeg（2006/10/26-6793 版） dll lib x264 vc6sp6 编译成功

基本编译

- 下载最新的 ffmpeg 源代码

(1) 下载并安装 Subversion (http://subversion.tigris.org/)

(2) 运行 svn co svn://svn.mplayerhq.hu/ffmpeg/trunk ffmpeg

- 下载安装最新的 mingw 和 msys

(1) mingw 通过在 http://prdownloads.sf.net/mingw/MinGW-5.0.3.exe?download 下载 mingw installer 安装,我安装
的时候选择了 candidate
(2) msys 在 http://www.mingw.org/下载安装
安装过后要填入 mingw 的路径,比如 d:/mingw

- 下载安装最新的 bash

(1) 在 http://www.mingw.org/下载最新的二进制版本
(2) 解压至 msys 的安装目录下

- 在 msys.bat 中加入下面语句:

```
call "C:\Program Files\Microsoft Visual Studio\VC98\Bin\VCVARS32.BAT"
```

- 在 msys 中编译 ffmpeg

(1) ./configure --enable-shared --disable-static --enable-memalign-hack --prefix=d:/ffmpeg
(2) make
(3) make install

加入 x264 支持

- 安装 nasm

- 编译 x264

```
./configure --prefix=d:/mingw
make
make install
```

- 编译 ffmpeg

```
./configure --enable-shared --disable-static --enable-memalign-hack
--enable-x264
--enable-gpl -- prefix=d:/ffmpeg
make
make install
```

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

ffmpeg（2006/10/26-6793 版） dll lib x264 vc6sp6 编译成功

7.9 关于运行 ffserver 有错误

错误如下：

checkout 了一个 ffmpeg 的包
编译了，也安装上了。
按照 doc 中的说明执行
ffserver -f doc/ffserver.conf
的时候显示
[hui@dh trunk]$ ./ffserver -f doc/ffserver.conf
Codecs do not match for stream 2
ffserver started.
这应该是有错误吧，后来看到有一个 tests 文件夹，cd，然后
[hui@dh trunk]$ ../ffserver -f test.conf
ffserver started.
然后就没“not”之类的提示了，估计是可以了，这个 doc/ffserver.conf 也是有问题的/？？？？？？？？？？？？？？？？？就发布了？？？？？？？？？？？？？？？？

2.3 How do I make it work?First, build the kit. It *really* helps to have installed LAME first. Then when you run the ffserver ./configure, make sure that you have the --enable-mp3lame flag turned on. LAME is important as it allows for streaming audio to Windows Media Player. Don't ask why the other audio types do not work. As a simple test, just run the following two command lines (assuming that you have a V4L video capture card): ./ffserver -f doc/ffserver.conf &./ffmpeg

http://localhost:8090/feed1.ffmAt this point you should be able to go to your Windows machine and fire up Windows Media
Player (WMP). Go to Open URL and enter
http://<linuxbox>:8090/test.asfYou should (after a short delay) see video and
hear audio. WARNING: trying to stream test1.mpg doesn't work with WMP as it tries to

尤其是上面的文档，执行完./ffserver -f doc/ffserver.conf &后先是刚才的那个错误提示，由于多次未成功现在很是恼火(偶得气量不大)，有问
题？？？？？？？？不可能吧！！可是提示有错误呀？？？！！！不管它，继续执行 ./ffmpeg http://localhost:8090/feed1.ffm 又有了新的提示：
[hui@dh trunk]$ ./ffmpeg http://localhost:8090/feed1.ffm
FFmpeg version SVN-r9022, Copyright (c) 2000-2007 Fabrice Bellard, et al.
 configuration: --prefix=../installdir
 libavutil version: 49.4.0
 libavcodec version: 51.40.4
 libavformat version: 51.12.1
 built on May 14 2007 16:38:55, gcc: 3.2.2 20030222 (Red Hat Linux 3.2.2-5)
Could not read stream parameters from 'http://localhost:8090/feed1.ffm'

相关讨论结果如下：

ffserver 启动之后就会监听 8090 端口，这个端口号应该在 conf 文件中可配置。
ffserver 启动后会创建一临时文件 feed1.ffm，文件大小大概是 4096B，里面只有头信息，是 ffserver 根据 conf 文件的配置生成的。
ffmpeg http://localhost:8090/feed1.ffm，它在做什么呢？大概是这样的，与 ffserver 建立 http 连接，然后从 ffserver 获取音频和视频的配置信息，然后尝试用这些信息初始化设备和 codec，所以看不到 ffmpeg 加一大堆参数启动。
ffmpeg 启动工作后，会把编码后的数据通过 udp 发给 ffserver，ffserver 接收到数据后循环的写入 feed1.ffm 文件，应该是做缓冲。一边写，一边监听网络上的
动静。
ffmpeg 的 main（）入口是有参数的，就是 argv 数组地址和参数个数
而在 ffserver 中 有一个 startchd()函数,其中有一个启动子进程的函数 execvp（），正是通过这个函数，ffserver 把 ffserver.conf 中的一些有关媒体的配置信息通过 argv 数组传递给了 ffmpeg，然后 ffmpeg 就进行数据采集或者格式转换的工作，转换之后的结果以 feed1.ffm 文件存放在/tmp 目录下，由 ffserver 读取并监听网络，这就是 ffserver 多媒体服务器。

在解析参数的时候，调用 opt_output_file，这里面要创建输出文件，对于 filename 为"http://localhost:8090/feed1.ffm"来说，调用 url_fopen 是建立了和 ffserver 的连接。
再看视频输出的地方，do_video_out，需要把 encodec 后的数据写入文件（av_interleaved_write_frame)，这个地方调用了 s->oformat->write_packet()，对于 s->filename 为"http://localhost:8090/feed1.ffm"来说，这里就是利用先前的连接把数据传给 ffserver，以 feed1.ffm 文件的格式。

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

[运行 ffserver 有错误]

7.10 如何加入 faac 和 faad 的支持

- 提出问题：

我最近想加入 faac 和 faad 的支持,但是 make 的时候总是说找不到 faac 的文件.我用的是 cvs 下来的最新版 ffmpeg.请问各位应该去哪找支持 faac 的库文件?

- 具体步骤：

先下到 faac 的源代码
在这里可以找到所有的 AAC 编码和解码器资源：
http://rarewares.org/aac.html
这个是开源的网站。代码很多
http://sourceforge.net/
然后./configure
然后将编译好的 libfaac.a 拷贝到 ffmpeg 的目录下(libavutil),这个库在 faac 的目录为：faac/libfaac/.libs
然后将 faac.h,faaccfg.h 拷贝到 ffmpeg 的目录下(libavutil)
设定 ffmpeg 支持 aac --enable-faac
这样就可以让 ffmpeg 支持 aac 了

有关该问题的讨论帖可参考 ffmpeg 工程组论坛中的相关讨论：

faac

7.11ffmpeg.exe 初始化出错

如果你编译的 ffmpeg.exe 运行时，出现“应用程序正常初始化(0xc0000005)失败”（The application failed to initialize properly (0x0000005). Click on OK to terminate the application），而且是 shared link 方式，通常的原因是 GCC 的版本问题，更换一个 GCC 的版本应该可以解决该问题。

下载一份 gcc-core-4.2.1-sjlj-2.tar.gz，然后解压缩
进入 mingw\bin 目录，通常是：C:\msys\mingw\bin\ 然后重命名下述文件:
c++-sjlj.exe to c++.exe
cpp-sjlj.exe to cpp.exe
g++-sjlj.exe to g++.exe
gcc-sjlj.exe to gcc.exe
然后再重新编译，通常可以解决该问题。

详细的原因分析，可以参考这个讨论帖：ffmpeg application failed initialize