1. 在ubuntu系统下搭建ffmpeg的编译环境

首先要获取ffmpeg的源代码，可以在github上下载，也可以到<http://ffmpeg.org/releases/ffmpeg-0.8.7.tar.gz>

在解压压缩包之前，需要安装一些SDL库相关的工具，不然ffplay播放会有问题，安装的工具主要有：

sudo apt-get install libx11-dev; sudo apt-get install xorg-dev。

安装完成之后，解压ffmpeg的源码，执行：

cd ffmpeg-0.8.7;

./configure –enable-memalign-hack –disable-debug

make

编译完成之后，可以看到目录下会生成ffmpeg, ffmpeg\_g, ffplay, ffplay\_g, ffserver, ffserver\_g,ffprobe, ffprobe\_g这八个可执行程序。

检测编译出的可执行程序是否能用，可以用ffplay播放一个本地视频文件：

./ffplay ~/test.mpg

这样ffmpeg在linux系统上的编译就OK了。

1. 在android系统下编译ffmpeg

Android编译是依赖于Android.mk文件，因此在android源码中编译ffmpeg，也需要给他添加相应的Android.mk文件。

将ffmpeg的源码拷贝到external目录下：

cp ~/ffmpeg ~/android4.2/external

在ffmpeg的根目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

Include $(CLEAR\_VARS)

#不能写成LOCAL\_STATIC\_LIBRARIES

LOCAL\_WHOLE\_STATIC\_LIBRARIES := libavformat libavcodec libavutil libpostproc libswscale

LOCAL\_MODULE := libffmpeg

include $(BUILD\_SHARED\_LIBRARY)

include $(call all-makefiles-under, $(LOCAL\_PATH))

在ffmpeg根目录下添加av.mk文件，用来配置编译libffmpeg所依赖的静态库，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(LOCAL\_PATH)/../config.mak

OBJS :=

OBJS-yes :=

MMX-OBJS-yse :=

include $(LOCAL\_PATH)/Makefile

OBJS-$(HAVE\_MMX) += $(MMX-OBJS-yes)

OBJS += $(OBJS-yes)

FFNAME := lib$(NAME)

FFLIBS := $(foreach, NAME, $(FFLIBS), lib$(NAME))

FFCFLAGS := -DHAVE\_AV\_CONFIG\_H –Wno-sign-compare –Wno-pointer-sign

FFCFLAGS += -DTARGET\_CONFIG=\”config-$(TARGET\_ARCH).h\”

ALL\_S\_FILES := $(wildcard $(LOCAL\_PATH)/$(TARGET\_ARCH)/\*.S)

ALL\_S\_FILES := $(addprefix $(TARGET\_ARCH)/,$(notdir $(ALL\_S\_FILES)))

ifneq ($(ALL\_S\_FILES))

ALL\_S\_OBJS := $(patsubst %.S,%.o,$(ALL\_S\_FILES))

C\_OBJS := $(filter-out $(ALL\_S\_OBJS),$(OBJS))

S\_OBJS := $(filter $(ALL\_S\_OBJS), $(OBJS))

else

C\_OBJS := $(OBJS)

S\_OBJS :=

endif

C\_FILES := $(patsubst, %.o,%.c,$(C\_OBJS))

S\_FILES := $(patsubst %.o,%.S,$(S\_OBJS))

FFFILES := $(sort $(S\_FILES)) $(sort $(C\_FILES))

在ffmpeg根目录下添加config.sh文件，内容如下：

#!/bin/bash

PREBUILT=~/mdback/prebuilt/linux-x86/toolchain/arm-linux-androideabi-4.4.x

PLATFORM=~/mdback/prebuilt/ndk/android-ndk-r6/platforms/android-9/arch-arm

./configure --target-os=linux \

--arch=arm \

--enable-cross-compile \

--cc=$PREBUILT/bin/arm-linux-androideabi-gcc \

--cross-prefix=$PREBUILT/bin/arm-linux-androideabi- \

--nm=$PREBUILT/bin/arm-linux-androideabi-nm \

--extra-cflags=”-fPIC -DANDROID” \

--enable-static \

--disable-shared \

--disable-asm \

--disable-yasm \

--prefix=~/mdback/external/ffmpeg0.6 \

--extra-ldflags=”-WI,-T,$PREBUILT/arm-linux-androideabi/lib/ldscripts/armelf\_linux\_eabi.x –WI,\

-rpath-link=$PLATFORM/usr/lib \

-L$PLATFORM/usr/lib –nostdlib \

$PREBUILT/lib/gcc/arm-linux-androideabi/4.4.3/crtbegin.o \

$PREBUILT/lib/gcc/arm-linux-androideabi/4.4.3/crtend.o –lc –lm -ldl”

config.sh完成之后，修改一下config.sh的权限:

chmod a+x config.sh

执行完成之后，需要删除libavformat, libavcodec, libavutil, libpostproc, libswscale, libavfilter 目录下的Makefile文件，将下面的内容从Makefile文件中删除：

include $(SUBDIR)../config.mak和include $(SUBDIR)../subdir.mak

还需要修改config.mak， 将!HAVE\_LRINT＝yes 改为HAVE\_LRINT=yes

现在需要的是编译libffmpeg库所依赖的静态库，libavformat, libavfilter, libavutil, libpostproc, libswscale，在指定的目录下添加相应的Android.mk文件，实现编译静态库。

如果出现这个问题：(ubuntu 10.04 64位系统)

make: ～/dev/android/ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/arm-linux-androideabi-gcc: Command not found  
当然。。其实文件是存在地。。。大体原因是因为我装的是X64的无帮突，而linux版的ndk貌似是32位地  
找了半天，找到一个解决办法

step1:  
sudo apt-get install libc6-dev-i386  
step2:  
sudo apt-get install ia32-libs

1. 编译libavcodec的静态库

在libavcodec目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := $(FFFILES)

LOCAL\_C\_INCLUDES := \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/..

LOCAL\_CFLAGS += $(FFCFLAGS)

LOCAL\_LDLIBS := -lz

LOCAL\_STATIC\_LIBRARIES := $(FFLIBS)

LOCAL\_MODULE := $(FFNAME)

include $(BUILD\_STATIC\_LIBRARY)

在libavfilter目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := $(FFFILES)

LOCAL\_C\_INCLUDES := \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/..

LOCAL\_CFLAGS += $(FFCFLAGS)

LOCAL\_LDLIBS := -lz

LOCAL\_STATIC\_LIBRARIES := $(FFLIBS)

LOCAL\_MODULE := $(FFNAME)

include $(BUILD\_STATIC\_LIBRARY)

在libavformat目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := $(FFFILES)

LOCAL\_C\_INCLUDES := \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/..

LOCAL\_CFLAGS += $(FFCFLAGS)

LOCAL\_CFLAGS += -include “string.h” –Dipv6mr\_interface=ipv6mr\_ifindex

LOCAL\_LDLIBS := -lz

LOCAL\_STATIC\_LIBRARIES := $(FFLIBS)

LOCAL\_MODULE := $(FFNAME)

include $(BUILD\_STATIC\_LIBRARY)

在libavutil目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := $(FFFILES)

LOCAL\_C\_INCLUDES := \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/..

LOCAL\_CFLAGS += $(FFCFLAGS)

LOCAL\_LDLIBS := -lz

LOCAL\_STATIC\_LIBRARIES := $(FFLIBS)

LOCAL\_MODULE := $(FFNAME)

include $(BUILD\_STATIC\_LIBRARY)

在libpostproc目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := $(FFFILES)

LOCAL\_C\_INCLUDES := \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/..

LOCAL\_CFLAGS += $(FFCFLAGS)

LOCAL\_LDLIBS := -lz

LOCAL\_STATIC\_LIBRARIES := $(FFLIBS)

LOCAL\_MODULE := $(FFNAME)

include $(BUILD\_STATIC\_LIBRARY)

在libswscale目录下添加Android.mk文件，内容如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := $(FFFILES)

LOCAL\_C\_INCLUDES := \

$(LOCAL\_PATH) \

$(LOCAL\_PATH)/..

LOCAL\_CFLAGS += $(FFCFLAGS)

LOCAL\_LDLIBS := -lz

LOCAL\_STATIC\_LIBRARIES := $(FFLIBS)

LOCAL\_MODULE := $(FFNAME)

include $(BUILD\_STATIC\_LIBRARY)

在修改完以上内容之后，再ffmpeg根目录下执行：

./config.sh

执行完成后，会自动生成config.h和config.mak这两个文件，我们需要手动去修改config.h这个文件，修改如下：

将#define restrict restrict修改为#define restrict

将#define HAVE\_PTHREADS 0修改为#define HAVE\_PTHREADS１

保存之后，回到ffmpeg根目录下，执行：

mm

1. ffmpeg框架代码

ffmpeg是一个集成录制、转换、音视频编码解码功能为一体的开源解决方案，支持MPEG、DivX、MPEG4、AC3、DV、FLV等多种编码，AVI、MPEG、OGG、Matroska、ASF等多种解码。

Ffmpeg依赖的库结构主要有：

(1)、libavformat：用于各种音视频封装格式的生成和解析，包括获取解码所需要的信息以及生成编解码上下文结构和读取音视频帧等功能。

(2)、libavcodec：用于各种类型声音/图像编解码

(3)、libavutil：包含一些公共的工具函数

(4)、libswscale：用于视频场景比例缩放，色彩映射转换

(5)、libavfilter：用于后期效果处理，比如添加水印等

(6)、libavdevice：用于视频源的获取

编译出现:

1.

external/media/ffmpeg-0.8.7/libavutil/libm.h:62: error: static declaration of 'lrint' follows non-static declaration

external/media/ffmpeg-0.8.7/libavutil/libm.h:69: error: static declaration of 'lrintf' follows non-static declaration

external/media/ffmpeg-0.8.7/libavutil/libm.h:76: error: static declaration of 'round' follows non-static declaration

external/media/ffmpeg-0.8.7/libavutil/libm.h:83: error: static declaration of 'roundf' follows non-static declaration

external/media/ffmpeg-0.8.7/libavutil/libm.h:90: error: static declaration of 'trunc' follows non-static declaration

external/media/ffmpeg-0.8.7/libavutil/libm.h:97: error: static declaration of 'truncf' follows non-static declaration

错误，

解决方法

打开  config.h 文件，查找  LRINT，将 0，改成1，示例

#define HAVE\_LRINT 0

改成

#define HAVE\_LRINT 1

分别查询 LRINT，ROUND，TRUNCF，进行相应的修改

2.

/home/chf0001/android-ndk/android-ndk-r5/samples/ffmpeg/obj/local/armeabi/libavutil.a(inverse.o):(.rodata+0x0): multiple definition of `ff\_inverse'

/home/chf0001/android-ndk/android-ndk-r5/samples/ffmpeg/obj/local/armeabi/libavcodec.a(inverse.o):(.rodata+0x0): first defined here

解决：

查看后发现：

在目录libavcodec下有文件inverse.c，内容为：

#include "libavutil/inverse.c"

而libavutil下的inverse.c中定义了变量ff\_inverse，所以在libavutil.a与libavcodec.a中都存在变量ff\_inverse的定义，当这二个文件一起链接成libffmpeg.so时，就出现了上面的问题。

了解了原因，解决办法就很简单，直接修改libavcodec/Makefile，把下面一行删掉：

OBJS-$(!CONFIG\_SMALL) += inverse.o

如果想直接利用.a库生成ffmpeg可执行程序，Android.mk如下：

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := cmdutils.c \

ffmpeg.c

LOCAL\_C\_INCLUDES := $(LOCAL\_PATH) \

$(LOCAL\_PATH)/libavformat \

$(LOCAL\_PATH)/libavcodec \

$(LOCAL\_PATH)/libavdevice \

$(LOCAL\_PATH)/libavfilter \

$(LOCAL\_PATH)/libavutil \

$(LOCAL\_PATH)/libpostproc \

$(LOCAL\_PATH)/libswscale

LOCAL\_MODULE\_TAGS = optional

LOCAL\_STATIC\_LIBRARIES := \

libavformat \

libavcodec \

libavdevice \

libavfilter \

libavutil \

libpostproc \

libswscale

LOCAL\_PRELINK\_MODULE := false

LOCAL\_MODULE:=ffmpeg  
include $(BUILD\_EXECUTABLE)

利用ffmpeg.so生成ffmpeg可执行程序

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := cmdutils.c \

ffmpeg.c

LOCAL\_C\_INCLUDES := $(LOCAL\_PATH) \

$(LOCAL\_PATH)/libavcodec \

$(LOCAL\_PATH)/libavdevice \

$(LOCAL\_PATH)/libavfilter \

$(LOCAL\_PATH)/libavformat \

$(LOCAL\_PATH)/libavutil \

$(LOCAL\_PATH)/libpostproc \

$(LOCAL\_PATH)/libswscale

LOCAL\_MODULE\_TAGS = optional

LOCAL\_SHARED\_LIBRARIES := libcutils \

libutils \

libffmpeg

LOCAL\_PRELINK\_MODULE := false

LOCAL\_MODULE := ffmpeg

include $(BUILD\_EXECUTABLE)

注意：在生成ffmpeg.so 的android.mk里必须LOCAL\_WHOLE\_STATIC\_LIBRARIES := libavformat libavcodec libavutil libpostproc libswscale

而不是

LOCAL\_STATIC\_LIBRARIES := libavformat libavcodec libavutil libpostproc libswscale

因为LOCAL\_STATIC\_LIBRARIES会删除掉一些无用的symbol

而LOCAL\_WHOLE\_STATIC\_LIBRARIES会把所有的symbol都加入到so库中。

下面是两个选项的解释：

***From the description below,if we use LOCAL\_WHOLE\_STATIC\_LIBRARIES to  
generate a shared library,all the symbols in static lib will be  
included in the shared library.***

***But in my test,the generated shared library is only about 4K which  
don't have all the symbols of static library, and there's unresolved  
symbol when another app link with the shared library.***

***Can someone help me?***

***LOCAL\_WHOLE\_STATIC\_LIBRARIES  
These are the static libraries that you want to include in your module  
without allowing the linker to remove dead code from them. This is  
mostly useful if you want to add a static library to a shared library  
and have the static library's content exposed from the shared  
library.***

***LOCAL\_WHOLE\_STATIC\_LIBRARIES := libsqlite3\_android***

***LOCAL\_STATIC\_LIBRARIES  
These are the static libraries that you want to include in your  
module. Mostly, we use shared libraries, but there are a couple of  
places, like executables in sbin and host executables where we use  
static libraries instead.***

***LOCAL\_STATIC\_LIBRARIES := libutils libtinyxml***