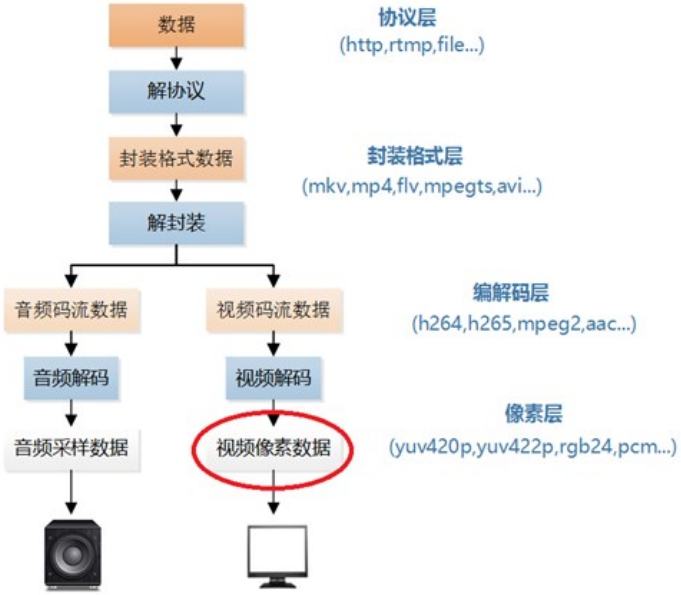
ffmpeg

FFmpeg 是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序。采用LGPL或GPL许可证。它提供了录制、转换以及流化音视频的完整解决方案。它包含了非常先进的音频/视频编解码库libavcodec，为了保证高可移植性和编解码质量，libavcodec里很多code都是从头开发的。

它主要包括：视频的采集，视频的编辑，视频的截图，视频的水印等

视频像素数据在视频播放器的解码流程中的位置如下图所示。



开发流程：

一，环境配置

所需工具

1，ndk-r8d

2，ubuntu 12.04 32位(我是用虚拟机的)

3，ffmpeg -1.2.4 (ffmpeg的版本最好与ndk对应，我试过很多版本，目前只有ndk-r8d和ffmpeg1.2.4能使)

4 ，jdk6

5，android sdk

6，eclipse

注：配置好ndk

二，编译ffmpeg.so库，也就是移植ffmepg到**android**。

编译ffmpeg.so库，也就是移植ffmepg到android

1，首先在你的工程目录下建立一个jni文件夹，然后把ffmpeg-1.2.4解压到jni目录下，然后把ffmpeg-1.2.4文件夹重命名为ffmpeg

2，再者在jni目录下建一个Android.mk文件

include $(all-subdir-makefiles) //这句话的意思是包含该目录下的所有mk文件

3、然后在jni/ffmpeg下建立Android.mk

```
1. LOCAL_PATH := $(call my-dir)
2. include $(CLEAR_VARS) //清楚所有全局变量的值 除了LOCAL_PATH
3. PATH_TO_FFMPEG_SOURCE:=$(LOCAL_PATH)/ffmpeg
4. LOCAL_C_INCLUDES += $(PATH_TO_FFMPEG_SOURCE)
5. LOCAL_WHOLE_STATIC_LIBRARIES := libavformat libavcodec libavfilter libavutil libpostproc libswscale libswresample
6. LOCAL_MODULE := ffmpeg //生成so库的名称 libffmpeg.so
7. include $(BUILD_SHARED_LIBRARY)
8. include $(call all-makefiles-under,$(LOCAL_PATH))
```

4，av.mk文件

```
1. include $(LOCAL_PATH)/../config.mak
2. OBJS :=
3. OBJS-yes :=
4. MMX-OBJS-yes :=
5. include $(LOCAL_PATH)/Makefile
6. # collect objects
7. OBJS-$(HAVE_MMX) += $(MMX-OBJS-yes)
8. OBJS += $(OBJS-yes)
9. FFNAME := lib$(NAME)
```

```
FFLIBS := $(foreach,NAME,$(FFLIBS),lib$(NAME))
FFCFLAGS  = -DHAVE_AV_CONFIG_H -Wno-sign-compare -Wno-switch -Wno-pointer-sign
FFCFLAGS += -DTARGET_CONFIG=\"config-$(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))
```

4、同样在jni/ffmpeg目录下建立config.sh文件　PREBUILT 和PLATFORM 要根据自己的ndk实际路径来配置。

```
PREBUILT=/home/yy/java/ndk8/android-ndk-r8d/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86
PLATFORM=/home/yy/java/ndk8/android-ndk-r8d/platforms/android-14/arch-arm

./configure --target-os=linux \
--arch=arm \
--enable-version3 \
--enable-gpl \
--enable-nonfree \
--enable-shared \
--enable-stripping \
--enable-ffmpeg \
--disable-ffplay \
--disable-ffserver \
--disable-ffprobe \
--enable-decoders \
--disable-symver \
--enable-encoders \
--enable-muxers \
--disable-devices \
--enable-protocols \
--enable-protocol=file \
--enable-avfilter \
--enable-network \
--disable-avdevice \
--disable-asm \
--enable-cross-compile \
--cc=$PREBUILT/bin/arm-linux-androideabi-gcc \
--cross-prefix=$PREBUILT/bin/arm-linux-androideabi- \
--strip=$PREBUILT/bin/arm-linux-androideabi-strip \
--extra-cflags="-fPIC -DANDROID" \
--extra-ldflags="-Wl,-T,$PREBUILT/arm-linux-androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,-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" \

sed -i 's/HAVE_LRINT 0/HAVE_LRINT 1/g' config.h
sed -i 's/HAVE_LRINTF 0/HAVE_LRINTF 1/g' config.h
sed -i 's/HAVE_ROUND 0/HAVE_ROUND 1/g' config.h
sed -i 's/HAVE_ROUNDF 0/HAVE_ROUNDF 1/g' config.h
sed -i 's/HAVE_TRUNC 0/HAVE_TRUNC 1/g' config.h
sed -i 's/HAVE_TRUNCF 0/HAVE_TRUNCF 1/g' config.h
sed -i 's/HAVE_CBRT 0/HAVE_CBRT 1/g' config.h
```

```
sed -i 's/HAVE_CBRTF 0/HAVE_CBRTF 1/g' config.h
sed -i 's/HAVE_ISINF 0/HAVE_ISINF 1/g' config.h
sed -i 's/HAVE_ISNAN 0/HAVE_ISNAN 1/g' config.h
sed -i 's/HAVE_SINF 0/HAVE_SINF 1/g' config.h
sed -i 's/HAVE_RINT 0/HAVE_RINT 1/g' config.h
```

5,在jni/ffmpeg/libavformat下添加Android,mk内容如下：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include $(LOCAL_PATH)/../av.mk
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)
```

6、在jni/ffmpeg/libavcodec下添加Android,mk内容如下：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include $(LOCAL_PATH)/../av.mk
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)
```

7,在jni/ffmpeg/libavutil libavfilter libpostproc libswscale libswresample 下添加Android,mk内容如下：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include $(LOCAL_PATH)/../av.mk
LOCAL_SRC_FILES := $(FFFILES)
LOCAL_C_INCLUDES :=       \
$(LOCAL_PATH)       \
$(LOCAL_PATH)/..
LOCAL_CFLAGS += $(FFCFLAGS)
LOCAL_STATIC_LIBRARIES := $(FFLIBS)
LOCAL_MODULE := $(FFNAME)
include $(BUILD_STATIC_LIBRARY)
```

8、**运行config.sh**

添加运行权限　chmod +x config.sh

执行config.sh

9、这样就会在ffmpeg目录下生成config.h，config.log,config.mak文件

然后修改jni/ffmpeg/config.h下的

#define avrestrict restrict为#define restrict

把config.log中的restrict的关键字删掉

10、删除 libavformat libavcodec libavutil libpostproc libswscale libswresample 目录下Makefile下的

include $(SUBDIR)../config.mak

删除libavcodec libavutil libswresample目录下Makefile下的 log2_tab.o \

11，把 ffmpeg/libavutil/time.h更名为avtime.h,
同时修改下面文件中的引用libavutil/time.h为libavutil/avtime.h
ffmpeg/libavformat/avformat.h:211
ffmpeg/libavformat/avio.c:25
ffmpeg/libavformat/hls.c:33
ffmpeg/libavformat/hlsproto.c:29
ffmpeg/libavformat/mux.c:39:30
ffmpeg/libavformat/utils.c:40
ffmpeg/libavutil/time.c:36
注:上面需要修改avtime.h文件引用的部分文件。根据版本，环境不同可能还会出现其他的文件引用time.h,如果当你编译的时候说找不到time.h，你就可以根据日志显示的文件逐个修改。很好解决的。

12、**最后重要的事情就是编译so库了**
回到project的目录下，如我的项目结构是是ffmpegPro/jni/ffmpeg，那么就需要退回到ffmpegPro目录下，执行
$NDK/ndk-build
如顺利就会在ffmpegPro目录下生成一个libs/armeabi的文件夹，里面会有一个libffmpeg.so的文件

13、编译过程中遇到的问题
问题1，编译的时候会在libavfilter文件的某个文件出现找不到time.h文件。time.h　No such file or directory

解决办法：参照第10点。

问题2，出现/home/yy/Java/ndkr8/android-ndk-r8d/build/core/build-binary.mk:41: * target file `clean' has both : and :: entries. Stop.**

解决办法：找到build-binary.mk提示错误那一行，把那一行注释掉。

问题3:会出现语法不对的，如 error: expected ';', ',' or ')' before 'vi' ，原因是因为不认restrict这个关键字

解决办法：参照第8点，把restrict的关键字删掉
三，**改编ffmpeg接口，供jni调用**
1，把编译好的ffmpeg.so文件复制到android-ndk-r8d/platforms/android-14/arch-arm/usr/lib目录下，注：android-14就是对应你的config.sh文件配置中的PLATFORM=/home/yy/java/ndk8/android-ndk-r8d/platforms/android-14/arch-arm的android-14
2，在jni目录下建立一个Android.mk文件（把之前的Android.mk文件删掉，或者重命名），内容如下：

```
1. include $(CLEAR_VARS)
2. PATH_TO_FFMPEG_SOURCE:=$(LOCAL_PATH)/ffmpeg
3. LOCAL_C_INCLUDES += $(PATH_TO_FFMPEG_SOURCE)
4. LOCAL_LDLIBS := -lffmpeg -llog -ljnigraphics -lz -ldl -lgcc  //这些就是所要关联的库了，刚才把ffmpeg.so复制到android-ndk-r8d/platforms/android-14/arch-arm/usr/lib目录下的原因就是为了这个
5. LOCAL_MODULE    := ffmpeg-jni
6. LOCAL_SRC_FILES := ffmpeg-jni.c ffmpeg/cmdutils.h ffmpeg/cmdutils.c ffmpeg/ffmpeg.h ffmpeg/ffmpeg_opt.c ffmpeg/ffmpeg_filter.c  //必须把这几个文件编译进去，不然会很多undefinded的。。
7. include $(BUILD_SHARED_LIBRARY)
```

3，同样在jni目录下建立一个ffmpeg-jni.c的文件，内容为ffmpeg.c文件的内容，只不过main函数改名为video_merge函数，然后在该.c文件创建一个jni接口，函数如下

```
1. jstring
2. Java_com_example_ffmpegpro_MainActivity_stringFromJNI( JNIEnv* env,
3.                                           jobject thiz )
4. {
5.     //LOGI("goto function video_gen()");
6.
7.         char const *str;
8.         int a=10;
9.         char *arg[10];
10.        arg[0]="ffmpeg";
11.        arg[1]="-i";
12.     arg[2]="/sdcard/movie/video2.avi";
```

```
    arg[3]="-i";
    arg[4]="/sdcard/movie/222.mp3";
    arg[5]="-vcodec";
    arg[6]="copy";
    arg[7]="-acodec";
    arg[8] = "copy";
    arg[9]="/sdcard/movie/output.avi";


    __android_log_print(ANDROID_LOG_INFO, "JNIMsg","============");
    __android_log_print(ANDROID_LOG_INFO, "filePath",arg[2]);

    int ret = video_merge(a,arg);

    str="Using FFMPEG doing your job";
    return (*env)->NewStringUTF(env,str);
}
```

4，接下来就是编译ffmepg-jni.c了。
回到ffmepgPro的目录下，执行$NDK/ndk-build　这样就在libs/armeabi的文件夹，里面会有一个libffmpeg-jni.so的文件
这样就可以在你的andorid项目里用jni使用这个功能了。

5、遇到的问题列表如下：

问题1：在编译ffmpeg-jni.c的时候，遇到很多undefinded。原因是在Android.mk文件按没有把下面的文件编译进去
ffmpeg/cmdutils.h ffmpeg/cmdutils.c ffmpeg/ffmpeg.h ffmpeg/ffmpeg_opt.c ffmpeg/ffmpeg_filter.c（上面我已经加进去了）

问题2：在getutime函数中说没有定义struct rusage**数据结构**。storage size of 'rusage' isn't known

解决办法：在头文件找到

```
#if HAVE_SYS_RESOURCE_H
#include <sys/time.h>
#include <sys/types.h>
#include <sys/resource.h>
#elif HAVE_GETPROCESSTIMES
```

把#include <sys/resource.h> #include <sys/time.h> 放在if语句外面就可以了。顺便在cmdutils.c文件中，也把这两个头文件引进来。。

问题3：找不到version.h文件。version.h　No such file or directory

解决办法：运行version.sh文件生成version.h　如：./version.sh . version.h