最简单的基于FFmpeg的移动端例子：Android HelloWorld

2015年07月23日 20:09:22　阅读数：82222

==========

最简单的基于FFmpeg的移动端例子系列文章列表：

最简单的基于FFmpeg的移动端例子：Android HelloWorld

最简单的基于FFmpeg的移动端例子：Android 视频解码器

最简单的基于FFmpeg的移动端例子：Android 视频解码器-单个库版

最简单的基于FFmpeg的移动端例子：Android 推流器

最简单的基于FFmpeg的移动端例子：Android 视频转码器

最简单的基于FFmpeg的移动端例子附件：Android 自带播放器

最简单的基于FFmpeg的移动端例子附件：SDL Android HelloWorld

最简单的基于FFmpeg的移动端例子：IOS HelloWorld

最简单的基于FFmpeg的移动端例子：IOS 视频解码器

最简单的基于FFmpeg的移动端例子：IOS 推流器

最简单的基于FFmpeg的移动端例子：IOS 视频转码器

最简单的基于FFmpeg的移动端例子附件：IOS自带播放器

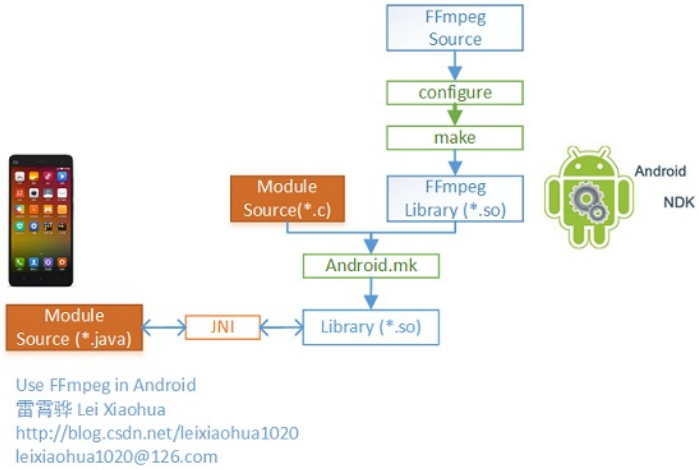
最简单的基于FFmpeg的移动端例子：Windows Phone HelloWorld

==========

从本文开始打算记录一系列FFmpeg在Android/IOS开发的示例程序。前面几篇文章记录FFmpeg安卓端开发的例子，后面几篇文章记录FFmpeg IOS端开发的例子。这些例子中FFmpeg相关的代码源自于《FFmpeg示例合集》中的程序。本文记录第一个程序：安卓平台下基于FFmpeg的HelloWorld程序。该程序的源代码源自于《 最简单的基于FFMPEG的Helloworld程序 》。

Android程序FFmpeg类库使用说明

Android应用程序使用FFmpeg类库的流程图如下所示。



上图中的流程可以分为“编译FFmpeg类库”、“编写Java端代码”、“编写C语言端代码”三个步骤。

（1）编译FFmpeg类库

a) 下载安装NDK

下载NDK之后直接解压缩就可以使用了。在Windows下使用的时候需要用到Cygwin。在这里我自己使用Linux编译类库。

b) 修改FFmpeg的configure

下载FFmpeg源代码之后，首先需要对源代码中的configure文件进行修改。由于编译出来的动态库文件名的版本号在.so之后（例如“libavcodec.so.5.100.1”），而android平台不能识别这样文件名，所以需要修改这种文件名。在configure文件中找到下面几行代码：

```
[plain]
1. SLIBNAME_WITH_MAJOR='$(SLIBNAME).$(LIBMAJOR)'
2. LIB_INSTALL_EXTRA_CMD='$$(RANLIB)"$(LIBDIR)/$(LIBNAME)"'
3. SLIB_INSTALL_NAME='$(SLIBNAME_WITH_VERSION)'
4. SLIB_INSTALL_LINKS='$(SLIBNAME_WITH_MAJOR)$(SLIBNAME)'
```

替换为下面内容就可以了：

```
[plain]
1. SLIBNAME_WITH_MAJOR='$(SLIBPREF)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)'
2. LIB_INSTALL_EXTRA_CMD='$$(RANLIB)"$(LIBDIR)/$(LIBNAME)"'
3. SLIB_INSTALL_NAME='$(SLIBNAME_WITH_MAJOR)'
4. SLIB_INSTALL_LINKS='$(SLIBNAME)'
```

c) 生成类库

按照configure、make、make　install的步骤就可以得到FFmpeg的头文件和类库文件了。其中configure的配置脚本在网上比较多。下面列举几个脚本。

FFmpeg类库完整功能脚本
下面这个脚本可以生成一套功能完整，体积比较大的类库。

```
cd ffmpeg

make clean

export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.8/prebuilt
export PLATFORM=$NDK/platforms/android-8/arch-arm
export PREFIX=../simplefflib
build_one(){
  ./configure --target-os=linux --prefix=$PREFIX \
--enable-cross-compile \
--enable-runtime-cpudetect \
--disable-asm \
--arch=arm \
--cc=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-gcc \
--cross-prefix=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi- \
--disable-stripping \
--nm=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-nm \
--sysroot=$PLATFORM \
--enable-gpl --enable-shared --disable-static --enable-small \
--disable-ffprobe --disable-ffplay --disable-ffmpeg --disable-ffserver --disable-debug \
--extra-cflags="-fPIC -DANDROID -D__thumb__ -mthumb -Wfatal-errors -Wno-deprecated -mfloat-abi=softfp -marm -march=armv7-a"
}

build_one

make
make install

cd ..
```

该脚本中前面几个变量“NDK”、“PREBUILT”、“PLATFORM”根据NDK路径的不同需要做相应的修改。另外需要注意64位NDK和32位NDK中某些文件夹名称也有一些区别：例如32位NDK中文件夹名称为“linux-x86”而64位NDK中文件夹名称为“linux-x86_64”。

将上述脚本拷贝至ffmpeg源代码外面，成功执行之后，会将类库和头文件生成到脚本所在目录下的“simplefflib”文件夹中。

FFmpeg类库裁剪功能后包含libx264和libfaac支持脚本

下面这个脚本可以生成一个裁剪功能后，包含libx264和libfaac支持的类库。

```
export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.6/prebuilt
export PLATFORM=$NDK/platforms/android-8/arch-arm
export PREFIX=../264fflib
build_one(){
  ./configure --target-os=linux --prefix=$PREFIX \
--enable-cross-compile \
--enable-runtime-cpudetect \
--disable-asm \
--arch=arm \
--cc=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-gcc \
--cross-prefix=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi- \
--disable-stripping \
--nm=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-nm \
--sysroot=$PLATFORM \
--enable-gpl --enable-shared --disable-static --enable-nonfree --enable-version3 --enable-small --disable-vda --disable-iconv \
--disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=libfaac \
--disable-muxers --enable-muxer=mov --enable-muxer=ipod --enable-muxer=psp --enable-muxer=mp4 --enable-muxer=avi \
--disable-decoders --enable-decoder=aac --enable-decoder=aac_latm --enable-decoder=h264 --enable-decoder=mpeg4 \
--disable-demuxers --enable-demuxer=h264 --enable-demuxer=avi --enable-demuxer=mpc --enable-demuxer=mov \
--disable-parsers --enable-parser=aac --enable-parser=ac3 --enable-parser=h264 \
--disable-protocols --enable-protocol=file \
--disable-bsfs --enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
--disable-indevs --enable-zlib \
--disable-outdevs --disable-ffprobe --disable-ffplay --disable-ffmpeg --disable-ffserver --disable-debug \
--extra-cflags="-I ../android-lib/include -fPIC -DANDROID -D__thumb__ -mthumb -Wfatal-errors -Wno-deprecated -mfloat-abi=softfp -marm -march=armv7-a" \
--extra-ldflags="-L ../android-lib/lib"

}

build_one

make
make install

cd ..
```

其中libfaac和libx264需要单独编译生成。它们编译过的类库位于“android-lib”文件夹中。libx264的编译脚本如下所示。

```
cd x264
export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.6/prebuilt
export PLATFORM=$NDK/platforms/android-8/arch-arm
export PREFIX=../android-lib
./configure --prefix=$PREFIX \
--enable-static \
--enable-pic \
--disable-asm \
--disable-cli \
--host=arm-linux \
--cross-prefix=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi- \
--sysroot=$PLATFORM
cd ..
```

libfaac的编译脚本如下所示。

```
cd faac
export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.6/prebuilt
export PLATFORM=$NDK/platforms/android-9/arch-arm
CFLAGS="-fpic -DANDROID -fpic -mthumb-interwork -ffunction-sections -funwind-tables -fstack-protector -fno-short-enums -D__ARM_ARCH_7__ -Wno-psabi -march=armv7 -mtune=xscale -msoft-float -mthumb -Os -fomit-frame-pointer -fno-strict-aliasing -finline-limit=64 -DANDROID -Wa,--noexecstack -MMD -MP "
#CFLAGS="-fpic -DANDROID -fpic -mthumb-interwork -D__ARM_ARCH_7__ -Wno-psabi -march=armv7-a -mtune=xscale -msoft-float -mthumb -Os -fomit-frame-pointer -fno-strict-aliasing -finline-limit=64 -DANDROID -Wa, -MMD -MP "
CROSS_COMPILE=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-
export CPPFLAGS="$CFLAGS"
export CFLAGS="$CFLAGS"
export CXXFLAGS="$CFLAGS"
export CXX="${CROSS_COMPILE}g++ --sysroot=${PLATFORM}"
export LDFLAGS="$LDFLAGS"
export CC="${CROSS_COMPILE}gcc --sysroot=${PLATFORM}"
export NM="${CROSS_COMPILE}nm"
export STRIP="${CROSS_COMPILE}strip"
export RANLIB="${CROSS_COMPILE}ranlib"
export AR="${CROSS_COMPILE}ar"

./configure \
--without-mp4v2 \
--host=arm-linux \
--enable-static \

make
make install

cp -rf /usr/local/include/faac.h ../android-lib/include
cp -rf /usr/local/include/faaccfg.h ../android-lib/include
cp -rf /usr/local/lib/libfaac.a ../android-lib/lib

cd ..
```

FFmpeg编译后生成的类库文件包含下面几个：

libavformat-56.so
libavcodec-56.so
libavfilter-5.so
libavdevice-56.so
libavutil-54.so
libpostproc-53.so
libswresample-1.so
libswscale-3.so

（2）编写Java端代码

使用Android IDE（例如Eclipse ADT）创建一个空的Android项目。也可以直接使用NDK中的hello-jni例子，该项目位于“{NDK目录}/samples/hello-jni”中。后文将会逐步改造hello-jni，使它支持FFmpeg类库的调用。
修改Android项目中“src”文件夹下的Java源代码，准备调用C语言函数。使用JNI调用C语言代码有两点需要做的步骤：

- 声明C语言函数对应的Java函数
- 声明要加载的类库

需要注意，C语言函数的声明要加上“native”关键字；加载类库的时候需要使用“System.loadLibrary()”方法。例如hello-jni例子中的Activity源代码如下所示。

```java
package com.example.hellojni;

import android.app.Activity;
import android.widget.TextView;
import android.os.Bundle;


public class HelloJni extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        /* Create a TextView and set its content.
         * the text is retrieved by calling a native
         * function.
         */
        TextView  tv = new TextView(this);
        tv.setText( stringFromJNI() );
        setContentView(tv);
    }

    /* A native method that is implemented by the
     * 'hello-jni' native library, which is packaged
     * with this application.
     */
    public native String  stringFromJNI();

    /* This is another native method declaration that is *not*
     * implemented by 'hello-jni'. This is simply to show that
     * you can declare as many native methods in your Java code
     * as you want, their implementation is searched in the
     * currently loaded native libraries only the first time
     * you call them.
     *
     * Trying to call this function will result in a
     * java.lang.UnsatisfiedLinkError exception !
     */
    public native String  unimplementedStringFromJNI();

    /* this is used to load the 'hello-jni' library on application
     * startup. The library has already been unpacked into
     * /data/data/com.example.hellojni/lib/libhello-jni.so at
     * installation time by the package manager.
     */
    static {
        System.loadLibrary("hello-jni");
    }
}
```

从源代码可以看出，该Activity加载了名称为“libhello-jni.so”的类库（Java代码中不包含前面的“lib”和后面的“.so”），并声明了stringFromJNI()方法。
在这里，为了调用FFmpeg而经过修改后的Activity加载类库部分的源代码如下所示。

```java
static {
 System.loadLibrary("avutil-54");
    System.loadLibrary("avcodec-56");
    System.loadLibrary("avformat-56");
    System.loadLibrary("avdevice-56");
    System.loadLibrary("swresample-1");
    System.loadLibrary("swscale-3");
    System.loadLibrary("postproc-53");
    System.loadLibrary("avfilter-5");
 System.loadLibrary("hello-jni");
}
```

（3）编写C语言端代码

a) 获取C语言的接口函数声明

根据Java对于C语言接口的定义，生成相应的接口函数声明。这一步需要用到JDK中的“javah”命令。例如对于hello-jni例子，首先切换到src文件夹下，输入如下命令：

```plain
javah com.example.hellojni.HelloJni
```

就可以在当前目录下生成一个头文件“com_example_hellojni_HelloJni.h”，该头文件内容如下所示。

```cpp
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_example_hellojni_HelloJni */

#ifndef _Included_com_example_hellojni_HelloJni
#define _Included_com_example_hellojni_HelloJni
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_example_hellojni_HelloJni
 * Method:    stringFromJNI
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_example_hellojni_HelloJni_stringFromJNI
  (JNIEnv *, jobject);

/*
 * Class:     com_example_hellojni_HelloJni
 * Method:    unimplementedStringFromJNI
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_example_hellojni_HelloJni_unimplementedStringFromJNI
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

从源代码可以看出，JNI调用的C语言函数是有固定格式的，即：

Java_{包名}_{包名}..._{类名}(JNIEnv *,...)

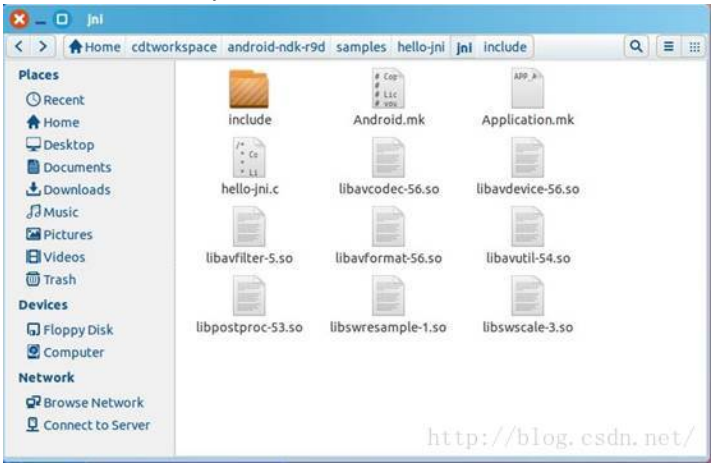
对于HelloJni类中的stringFromJNI方法，其C语言版本的函数声明为：

```cpp
JNIEXPORT jstring JNICALL Java_com_example_hellojni_HelloJni_stringFromJNI (JNIEnv *, jobject)
```

PS：这个头文件只是一个参考，对于JNI来说并不是必须的。也可以根据命名规则直接编写C语言函数。

b) 编写C语言接口函数代码

在Android项目根目录下新建“jni”文件夹，用于存储C语言源代码以及相关的开发资源。将编译生成的FFmpeg的类库（.so文件）和头文件（.h文件）拷贝到这个目录下，然后新建一个C语言文件，就可以开始编写相应的逻辑了。此时jni文件夹目录结构如下图所示。



C语言文件用于实现上文中通过“javah”命令生成的头文件的函数。对于hello-jni例子，其C语言文件内容如下所示。

```cpp
#include <string.h>
#include <jni.h>

/* This is a trivial JNI example where we use a native method
 * to return a new VM String. See the corresponding Java source
 * file located at:
 *
 *   apps/samples/hello-jni/project/src/com/example/hellojni/HelloJni.java
 */
jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
                                                  jobject thiz )
{
#if defined(__arm__)
  #if defined(__ARM_ARCH_7A__)
    #if defined(__ARM_NEON__)
      #define ABI "armeabi-v7a/NEON"
    #else
      #define ABI "armeabi-v7a"
    #endif
  #else
   #define ABI "armeabi"
  #endif
#elif defined(__i386__)
   #define ABI "x86"
#elif defined(__mips__)
   #define ABI "mips"
#else
   #define ABI "unknown"
#endif

    return (*env)->NewStringUTF(env, "Hello from JNI !  Compiled with ABI " ABI ".");
}
```

可以看出，Java_com_example_hellojni_HelloJni_stringFromJNI()根据宏定义判定了系统类型并且返回了一个字符串。在这里要注意，C语言中的char[]是不能直接对应为Java中的String类型的（即jstring）。char[]转换为String需要通过JNIEnv的NewStringUTF()函数。
为了调用FFmpeg而经过修改后的Java_com_example_hellojni_HelloJni_stringFromJNI()的源代码如下所示。

```cpp
#include <string.h>
#include <jni.h>
#include "libavcodec/avcodec.h"

jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
                                                  jobject thiz )
{
    char info[10000] = { 0 };
    sprintf(info, "%s\n", avcodec_configuration());
    return (*env)->NewStringUTF(env, info);
}
```

可以看出该函数调用了libavcodec的avcodec_configuration()方法，用于获取FFmpeg的配置信息。

c) 编写Android.mk

完成C语言程序的编写后，就可以开始编写Android的makefile文件Android.mk了。hello-jni例子中的Android.mk内容如下：

```plain
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)
```

编译FFmpeg示例程序的时候由于用到了libavcodec等相关的库，所以将Android.mk文件修改如下：

```
[plain]
LOCAL_PATH := $(call my-dir)

# FFmpeg library
include $(CLEAR_VARS)
LOCAL_MODULE := avcodec
LOCAL_SRC_FILES := libavcodec-56.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avdevice
LOCAL_SRC_FILES := libavdevice-56.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avfilter
LOCAL_SRC_FILES := libavfilter-5.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avformat
LOCAL_SRC_FILES := libavformat-56.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avutil
LOCAL_SRC_FILES := libavutil-54.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := postproc
LOCAL_SRC_FILES := libpostproc-53.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := swresample
LOCAL_SRC_FILES := libswresample-1.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := swscale
LOCAL_SRC_FILES := libswscale-3.so
include $(PREBUILT_SHARED_LIBRARY)

# Program
include $(CLEAR_VARS)
LOCAL_MODULE := hello-jni
LOCAL_SRC_FILES := hello-jni.c
LOCAL_C_INCLUDES += $(LOCAL_PATH)/include
LOCAL_LDLIBS := -llog -lz
LOCAL_SHARED_LIBRARIES := avcodec avdevice avfilter avformat avutil postproc swresample swscale
include $(BUILD_SHARED_LIBRARY)
```

d) 编写Application.mk（可选）

Application.mk中的APP_ABI设定了编译后库文件支持的指令集，默认使用“armeabi”。在hello-jni例子中，APP_ABI取值为“all”。由于我们编译的FFmpeg并不在像x86这样的平台下运行，所以不需要“all”，把它修改为“armeabi”或者删除就可以了（对于hello-jni这个例子，不做这一步的话会在编译x86平台类库的时候报错，但并不影响后面的测试运行）。

e) 运行ndk-build

编写完Android的Makefile文件之后，就可以运行ndk-build编译生成可以通过JNI调用的类库了。ndk-build本身是一个脚本，位于NDK根目录下。切换到Android程序目录中，直接执行该脚本就可以了。
ndk-build成功后，会在根目录下的“libs/armeabi”目录中生成相关的库文件。hello-jni例子中，会生成以下库文件：

libavformat-56.so
libavcodec-56.so
libavfilter-5.so
libavdevice-56.so
libavutil-54.so
libpostproc-53.so
libswresample-1.so
libswscale-3.so
libhello-jni.so

接下来就可以在Android手机或者虚拟机上对整个Android工程进行测试了。

f) 程序运行结果

程序最终的运行结果截图如下所示。

从图中可以看出，程序中打印出了FFmpeg的配置信息。

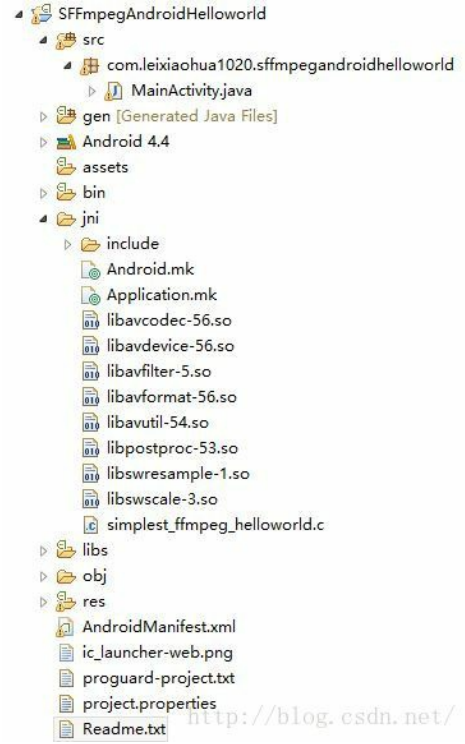
FFmpeg Helloworld

本文记录的FFmpeg Helloworld程序C语言的源代码来自于《最简单的基于FFMPEG的Helloworld程序》。改程序会输出FFmpeg类库下列信息：

Protocol: FFmpeg类库支持的协议
AVFormat: FFmpeg类库支持的封装格式
AVCodec: FFmpeg类库支持的编解码器
AVFilter: FFmpeg类库支持的滤镜
Configure: FFmpeg类库的配置信息

源代码

项目的目录结构如图所示。Java源代码位于src目录，而C代码位于jni目录。



Android程序Java端代码位于src\com\leixiaohua1020\sffmpegandroidhelloworld\MainActivity.java，如下所示。

[java]

```java
/**
 * 最简单的基于FFmpeg的Helloworld例子-安卓
 * Simplest FFmpeg Android Helloworld
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 *
 * 本程序是移植FFmpeg到安卓平台的最简单程序。它可以打印出FFmpeg类库的下列信息：
 *  Protocol:  FFmpeg类库支持的协议
 *  AVFormat:  FFmpeg类库支持的封装格式
 *  AVCodec:   FFmpeg类库支持的编解码器
 *  AVFilter:  FFmpeg类库支持的滤镜
 *  Configure: FFmpeg类库的配置信息
 *
 * This is the simplest program based on FFmpeg in Android. It can show following
 * informations about FFmpeg library:
 *  Protocol:  Protocols supported by FFmpeg.
 *  AVFormat:  Container format supported by FFmpeg.
 *  AVCodec:   Encoder/Decoder supported by FFmpeg.
 *  AVFilter:  Filters supported by FFmpeg.
 *  Configure: configure information of FFmpeg.
 *
 */


package com.leixiaohua1020.sffmpegandroidhelloworld;


import android.os.Bundle;
import android.app.Activity;
import android.text.method.ScrollingMovementMethod;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final TextView libinfoText = (TextView) findViewById(R.id.text_libinfo);
        libinfoText.setMovementMethod(ScrollingMovementMethod.getInstance());

        libinfoText.setText(configurationinfo());

        Button configurationButton = (Button) this.findViewById(R.id.button_configuration);
        Button urlprotocolButton = (Button) this.findViewById(R.id.button_urlprotocol);
        Button avformatButton = (Button) this.findViewById(R.id.button_avformat);
        Button avcodecButton = (Button) this.findViewById(R.id.button_avcodec);
        Button avfilterButton = (Button) this.findViewById(R.id.button_avfilter);

        urlprotocolButton.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0){
                libinfoText.setText(urlprotocolinfo());
            }
        });

        avformatButton.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0){
                libinfoText.setText(avformatinfo());
            }
        });

        avcodecButton.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0){
                libinfoText.setText(avcodecinfo());
            }
        });

        avfilterButton.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0){
                libinfoText.setText(avfilterinfo());
            }
        });

        configurationButton.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0){
                libinfoText.setText(configurationinfo());
            }
        });

```

```
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    //JNI
    public native String urlprotocolinfo();
    public native String avformatinfo();
    public native String avcodecinfo();
    public native String avfilterinfo();
    public native String configurationinfo();

    static{
        System.loadLibrary("avutil-54");
        System.loadLibrary("swresample-1");
        System.loadLibrary("avcodec-56");
        System.loadLibrary("avformat-56");
        System.loadLibrary("swscale-3");
        System.loadLibrary("postproc-53");
        System.loadLibrary("avfilter-5");
        System.loadLibrary("avdevice-56");
        System.loadLibrary("sffhelloworld");
    }

}
```

C语言端源代码位于jni/simplest_ffmpeg_helloworld.c，如下所示。

```cpp
/**
 * 最简单的基于FFmpeg的Helloworld例子-安卓
 * Simplest FFmpeg Android Helloworld
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 *
 * 本程序是移植FFmpeg到安卓平台的最简单程序。它可以打印出FFmpeg类库的下列信息：
 * Protocol:  FFmpeg类库支持的协议
 * AVFormat:  FFmpeg类库支持的封装格式
 * AVCodec:   FFmpeg类库支持的编解码器
 * AVFilter:  FFmpeg类库支持的滤镜
 * Configure: FFmpeg类库的配置信息
 *
 * This is the simplest program based on FFmpeg in Android. It can show following
 * informations about FFmpeg library:
 * Protocol:  Protocols supported by FFmpeg.
 * AVFormat:  Container format supported by FFmpeg.
 * AVCodec:   Encoder/Decoder supported by FFmpeg.
 * AVFilter:  Filters supported by FFmpeg.
 * Configure: configure information of FFmpeg.
 */

#include <stdio.h>

#include "libavcodec/avcodec.h"
#include "libavformat/avformat.h"
#include "libavfilter/avfilter.h"

//Log
#ifdef ANDROID
#include <jni.h>
#include <android/log.h>
#define LOGE(format, ...)  __android_log_print(ANDROID_LOG_ERROR, "(>_<)", format, ##__VA_ARGS__)
#else
#define LOGE(format, ...)  printf("(>_<) " format "\n", ##__VA_ARGS__)
#endif


//FIX
struct URLProtocol;
/**
 * com.leixiaohua1020.sffmpegandroidhelloworld.MainActivity.urlprotocolinfo()
 * Protocol Support Information
 */
JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegandroidhelloworld_MainActivity_urlprotocolinfo(JNIEnv *env, jobject obj){

    char info[40000]={0};
    av_register_all();

    struct URLProtocol *pup = NULL;
```

```
    //Input
    struct URLProtocol **p_temp = &pup;
    avio_enum_protocols((void **)p_temp, 0);
    while ((*p_temp) != NULL){
        sprintf(info, "%s[In ][%10s]\n", info, avio_enum_protocols((void **)p_temp, 0));
    }
    pup = NULL;
    //Output
    avio_enum_protocols((void **)p_temp, 1);
    while ((*p_temp) != NULL){
        sprintf(info, "%s[Out][%10s]\n", info, avio_enum_protocols((void **)p_temp, 1));
    }

    //LOGE("%s", info);
    return (*env)->NewStringUTF(env, info);
}

/**
 * com.leixiaohua1020.sffmpegandroidhelloworld.MainActivity.avformatinfo()
 * AVFormat Support Information
 */
JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegandroidhelloworld_MainActivity_avformatinfo(JNIEnv *env, jobject obj){

    char info[40000] = { 0 };

    av_register_all();

    AVInputFormat *if_temp = av_iformat_next(NULL);
    AVOutputFormat *of_temp = av_oformat_next(NULL);
    //Input
    while(if_temp!=NULL){
        sprintf(info, "%s[In ][%10s]\n", info, if_temp->name);
        if_temp=if_temp->next;
    }
    //Output
    while (of_temp != NULL){
        sprintf(info, "%s[Out][%10s]\n", info, of_temp->name);
        of_temp = of_temp->next;
    }
    //LOGE("%s", info);
    return (*env)->NewStringUTF(env, info);
}

/**
 * com.leixiaohua1020.sffmpegandroidhelloworld.MainActivity.avcodecinfo()
 * AVCodec Support Information
 */
JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegandroidhelloworld_MainActivity_avcodecinfo(JNIEnv *env, jobject obj)
{
    char info[40000] = { 0 };

    av_register_all();

    AVCodec *c_temp = av_codec_next(NULL);

    while(c_temp!=NULL){
        if (c_temp->decode!=NULL){
            sprintf(info, "%s[Dec]", info);
        }
        else{
            sprintf(info, "%s[Enc]", info);
        }
        switch (c_temp->type){
        case AVMEDIA_TYPE_VIDEO:
            sprintf(info, "%s[Video]", info);
            break;
        case AVMEDIA_TYPE_AUDIO:
            sprintf(info, "%s[Audio]", info);
            break;
        default:
            sprintf(info, "%s[Other]", info);
            break;
        }
        sprintf(info, "%s[%10s]\n", info, c_temp->name);


        c_temp=c_temp->next;
    }
    //LOGE("%s", info);

    return (*env)->NewStringUTF(env, info);
}

/**
 * com.leixiaohua1020.sffmpegandroidhelloworld.MainActivity.avfilterinfo()
 * AVFilter Support Information
 */
JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegandroidhelloworld_MainActivity_avfilterinfo(JNIEnv *env, jobject obj)
{
    char info[40000] = { 0 };
    avfilter_register_all();
```

```
    AVFilter *f_temp = (AVFilter *)avfilter_next(NULL);
    while (f_temp != NULL){
        sprintf(info, "%s[%10s]\n", info, f_temp->name);
    }
    //LOGE("%s", info);

    return (*env)->NewStringUTF(env, info);
}

/**
 * com.leixiaohua1020.sffmpegandroidhelloworld.MainActivity.urlprotocolinfo()
 * Protocol Support Information
 */
JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegandroidhelloworld_MainActivity_configurationinfo(JNIEnv *env, jobject obj)
{
    char info[10000] = { 0 };
    av_register_all();

    sprintf(info, "%s\n", avcodec_configuration());

    //LOGE("%s", info);
    return (*env)->NewStringUTF(env, info);
}
```

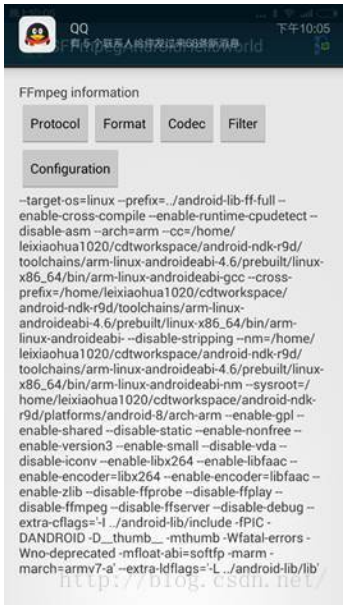
Android.mk文件位于jni/Android.mk，如下所示。

```
[plain]
# Android.mk for FFmpeg
#
# Lei Xiaohua 雷霄骅
# leixiaohua1020@126.com
# http://blog.csdn.net/leixiaohua1020
#

LOCAL_PATH := $(call my-dir)

# FFmpeg library
include $(CLEAR_VARS)
LOCAL_MODULE := avcodec
LOCAL_SRC_FILES := libavcodec-56.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avdevice
LOCAL_SRC_FILES := libavdevice-56.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avfilter
LOCAL_SRC_FILES := libavfilter-5.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avformat
LOCAL_SRC_FILES := libavformat-56.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := avutil
LOCAL_SRC_FILES := libavutil-54.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := postproc
LOCAL_SRC_FILES := libpostproc-53.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := swresample
LOCAL_SRC_FILES := libswresample-1.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := swscale
LOCAL_SRC_FILES := libswscale-3.so
include $(PREBUILT_SHARED_LIBRARY)

# Program
include $(CLEAR_VARS)
LOCAL_MODULE := sffhelloworld
LOCAL_SRC_FILES :=simplest_ffmpeg_helloworld.c
LOCAL_C_INCLUDES += $(LOCAL_PATH)/include
LOCAL_LDLIBS := -llog -lz
LOCAL_SHARED_LIBRARIES := avcodec avdevice avfilter avformat avutil postproc swresample swscale
include $(BUILD_SHARED_LIBRARY)
```

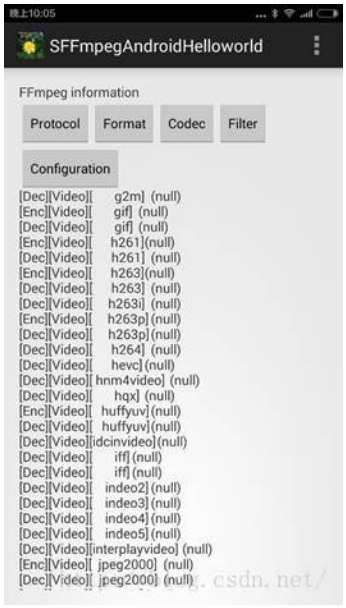
其它部分源代码暂不详细例举。

运行结果

App在手机上运行后的结果如下图所示。



单击不同的按钮，可以查看FFmpeg类库相关的信息。



下载

simplest ffmpeg mobile

项目主页
Github： https://github.com/leixiaohua1020/simplest_ffmpeg_mobile

开源中国： https://git.oschina.net/leixiaohua1020/simplest_ffmpeg_mobile

SourceForge： https://sourceforge.net/projects/simplestffmpegmobile/

CSDN工程下载地址： http://download.csdn.net/detail/leixiaohua1020/8924391

本解决方案包含了使用FFmpeg在移动端处理多媒体的各种例子：

[Android]
simplest_android_player: 基于安卓接口的视频播放器

simplest_ffmpeg_android_helloworld: 安卓平台下基于FFmpeg的HelloWorld程序
simplest_ffmpeg_android_decoder: 安卓平台下最简单的基于FFmpeg的视频解码器
simplest_ffmpeg_android_decoder_onelib: 安卓平台下最简单的基于FFmpeg的视频解码器-单库版
simplest_ffmpeg_android_streamer: 安卓平台下最简单的基于FFmpeg的推流器
simplest_ffmpeg_android_transcoder: 安卓平台下移植的FFmpeg命令行工具
simplest_sdl_android_helloworld: 移植SDL到安卓平台的最简单程序
[IOS]
simplest_ios_player: 基于IOS接口的视频播放器
simplest_ffmpeg_ios_helloworld: IOS平台下基于FFmpeg的HelloWorld程序
simplest_ffmpeg_ios_decoder: IOS平台下最简单的基于FFmpeg的视频解码器
simplest_ffmpeg_ios_streamer: IOS平台下最简单的基于FFmpeg的推流器
simplest_ffmpeg_ios_transcoder: IOS平台下移植的ffmpeg.c命令行工具
simplest_sdl_ios_helloworld: 移植SDL到IOS平台的最简单程序

文章标签： ffmpeg Android IOS 视频 编解码
个人分类： FFMPEG Android多媒体
所属专栏： FFmpeg