

# 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：82036

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](#)

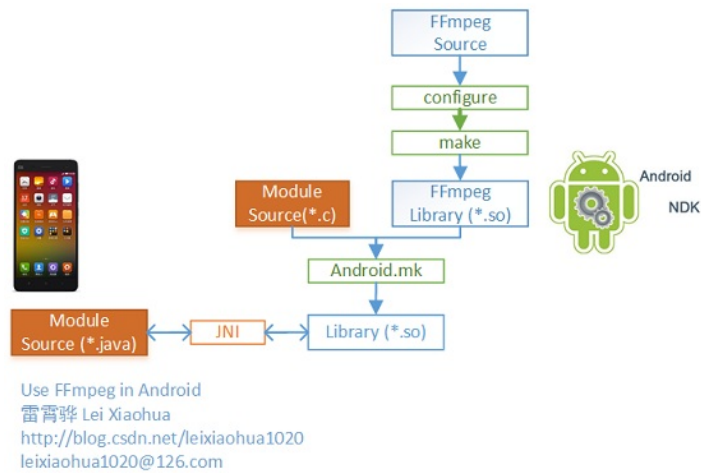
[最简单的基于FFmpeg的移动端例子：Windows Phone HelloWorld](#)

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='$(SLIBPREFIX)$(FULLNAME)-$(LIBMAJOR)$$(LIBSUF)'
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。

```
[plain]
1. cd ffmpeg
2.
3. make clean
4.
5. export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
6. export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.8/prebuilt
7. export PLATFORM=$NDK/platforms/android-8/arch-arm
8. export PREFIX=./simplefflib
9. build_one(){
10.     ./configure --target-os=linux --prefix=$PREFIX \
11.     --enable-cross-compile \
12.     --enable-runtime-cpudetect \
13.     --disable-asm \
14.     --arch=arm \
15.     --cc=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-gcc \
16.     --cross-prefix=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi- \
17.     --disable-stripping \
18.     --nm=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-nm \
19.     --sysroot=$PLATFORM \
20.     --enable-gpl --enable-shared --disable-static --enable-small \
21.     --disable-ffprobe --disable-ffplay --disable-ffmpeg --disable-ffserver --disable-debug \
22.     --extra-cflags="-fPIC -DANDROID -D__thumb__ -mthumb -Wfatal-errors -Wno-deprecated -mfloat-abi=softfp -marm -march=armv7-a"
23. }
24.
25. build_one
26.
27. make
28. make install
29.
30. 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。

```
[plain]
1. export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
2. export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.6/prebuilt
3. export PLATFORM=$NDK/platforms/android-8/arch-arm
4. export PREFIX=./264fflib
5. build_one(){
6.     ./configure --target-os=linux --prefix=$PREFIX \
7.     --enable-cross-compile \
8.     --enable-runtime-cpudetect \
9.     --disable-asm \
10.     --arch=arm \
11.     --cc=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-gcc \
12.     --cross-prefix=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi- \
13.     --disable-stripping \
14.     --nm=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-nm \
15.     --sysroot=$PLATFORM \
16.     --enable-gpl --enable-shared --disable-static --enable-nonfree --enable-version3 --enable-small --disable-vda --disable-iconv \
17.     --disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=libfaac \
18.     --disable-muxers --enable-muxer=mov --enable-muxer=ipod --enable-muxer=psp --enable-muxer=mp4 --enable-muxer=avi \
19.     --disable-decoders --enable-decoder=aac --enable-decoder=aac_latm --enable-decoder=h264 --enable-decoder=mpeg4 \
20.     --disable-demuxers --enable-demuxer=h264 --enable-demuxer=avi --enable-demuxer=mpc --enable-demuxer=mov \
21.     --disable-parsers --enable-parser=aac --enable-parser=ac3 --enable-parser=h264 \
22.     --disable-protocols --enable-protocol=file \
23.     --disable-bsfs --enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
24.     --disable-indevs --enable-zlib \
25.     --disable-outdevs --disable-ffprobe --disable-ffplay --disable-ffmpeg --disable-ffserver --disable-debug \
26.     --extra-cflags="-I ../android-lib/include -fPIC -DANDROID -D__thumb__ -mthumb -Wfatal-errors -Wno-deprecated -mfloat-abi=softfp -marm -march=armv7-a" \
27.     --extra-ldflags="-L ../android-lib/lib"
28. }
29.
30. build_one
31.
32. make
33. make install
34.
35. 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。

```
[plain]
1. cd x264
2. export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
3. export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.6/prebuilt
4. export PLATFORM=$NDK/platforms/android-8/arch-arm
5. export PREFIX=./android-lib
6. ./configure --prefix=$PREFIX \
7. --enable-static \
8. --enable-pic \
9. --disable-asm \
10. --disable-cli \
11. --host=arm-linux \
12. --cross-prefix=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi- \
13. --sysroot=$PLATFORM
14. cd ..
```

libfaac的编译脚本如下所示。

```
[plain]
1. cd faac
2. export NDK=/home/leixiaohua1020/cdtworkspace/android-ndk-r9d
3. export PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.6/prebuilt
4. export PLATFORM=$NDK/platforms/android-9/arch-arm
5. 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 "
6. #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,--noexecstack -MMD -MP "
7. CROSS_COMPILE=$PREBUILT/linux-x86_64/bin/arm-linux-androideabi-
8. export CPPFLAGS="$CFLAGS"
9. export CFLAGS="$CFLAGS"
10. export CXXFLAGS="$CFLAGS"
11. export CXX="{CROSS_COMPILE}g++ --sysroot=${PLATFORM}"
12. export LDFLAGS="$LDFLAGS"
13. export CC="{CROSS_COMPILE}gcc --sysroot=${PLATFORM}"
14. export NM="{CROSS_COMPILE}nm"
15. export STRIP="{CROSS_COMPILE}strip"
16. export RANLIB="{CROSS_COMPILE}ranlib"
17. export AR="{CROSS_COMPILE}ar"
18.
19. ./configure \
20. --without-mp4v2 \
21. --host=arm-linux \
22. --enable-static \
23.
24. make
25. make install
26.
27. cp -rf /usr/local/include/faac.h ../android-lib/include
28. cp -rf /usr/local/include/faaccfg.h ../android-lib/include
29. cp -rf /usr/local/lib/libfaac.a ../android-lib/lib
30.
31. 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。

```

1. package com.example.hellojni;
2.
3. import android.app.Activity;
4. import android.widget.TextView;
5. import android.os.Bundle;
6.
7.
8. public class HelloJni extends Activity
9. {
10.     /** Called when the activity is first created. */
11.     @Override
12.     public void onCreate(Bundle savedInstanceState)
13.     {
14.         super.onCreate(savedInstanceState);
15.
16.         /* Create a TextView and set its content.
17.          * the text is retrieved by calling a native
18.          * function.
19.          */
20.         TextView tv = new TextView(this);
21.         tv.setText( stringFromJNI() );
22.         setContentView(tv);
23.     }
24.
25.     /* A native method that is implemented by the
26.      * 'hello-jni' native library, which is packaged
27.      * with this application.
28.      */
29.     public native String stringFromJNI();
30.
31.     /* This is another native method declaration that is *not*
32.      * implemented by 'hello-jni'. This is simply to show that
33.      * you can declare as many native methods in your Java code
34.      * as you want, their implementation is searched in the
35.      * currently loaded native libraries only the first time
36.      * you call them.
37.      *
38.      * Trying to call this function will result in a
39.      * java.lang.UnsatisfiedLinkError exception !
40.      */
41.     public native String unimplementedStringFromJNI();
42.
43.     /* this is used to load the 'hello-jni' library on application
44.      * startup. The library has already been unpacked into
45.      * /data/data/com.example.hellojni/lib/libhello-jni.so at
46.      * installation time by the package manager.
47.      */
48.     static {
49.         System.loadLibrary("hello-jni");
50.     }
51. }

```

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。

```

1. static {
2.     System.loadLibrary("avutil-54");
3.     System.loadLibrary("avcodec-56");
4.     System.loadLibrary("avformat-56");
5.     System.loadLibrary("avdevice-56");
6.     System.loadLibrary("swresample-1");
7.     System.loadLibrary("swscale-3");
8.     System.loadLibrary("postproc-53");
9.     System.loadLibrary("avfilter-5");
10.    System.loadLibrary("hello-jni");
11. }

```

### (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：

```

1. 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。

```

1.  /* DO NOT EDIT THIS FILE - it is machine generated */
2.  #include <jni.h>
3.  /* Header for class com_example_hellojni_HelloJni */
4.
5.  #ifndef _Included_com_example_hellojni_HelloJni
6.  #define _Included_com_example_hellojni_HelloJni
7.  #ifdef __cplusplus
8.  extern "C" {
9.  #endif
10. /*
11.  * Class:      com_example_hellojni_HelloJni
12.  * Method:     stringFromJNI
13.  * Signature:  ()Ljava/lang/String;
14.  */
15. JNIEXPORT jstring JNICALL Java_com_example_hellojni_HelloJni_stringFromJNI
16. (JNIEnv *, jobject);
17.
18. /*
19.  * Class:      com_example_hellojni_HelloJni
20.  * Method:     unimplementedStringFromJNI
21.  * Signature:  ()Ljava/lang/String;
22.  */
23. JNIEXPORT jstring JNICALL Java_com_example_hellojni_HelloJni_unimplementedStringFromJNI
24. (JNIEnv *, jobject);
25.
26. #ifdef __cplusplus
27. }
28. #endif
29. #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：

```

1.  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]
1. #include <string.h>
2. #include <jni.h>
3.
4. /* This is a trivial JNI example where we use a native method
5.  * to return a new VM String. See the corresponding Java source
6.  * file located at:
7.  *
8.  *   apps/samples/hello-jni/project/src/com/example/hellojni/HelloJni.java
9.  */
10. jstring
11. Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
12.                                                    jobject this )
13. {
14.     #if defined( __arm__ )
15.         #if defined( __ARM_ARCH_7A__ )
16.             #if defined( __ARM_NEON__ )
17.                 #define ABI "armeabi-v7a/NEON"
18.             #else
19.                 #define ABI "armeabi-v7a"
20.             #endif
21.         #else
22.             #define ABI "armeabi"
23.         #endif
24.     #elif defined( __i386__ )
25.         #define ABI "x86"
26.     #elif defined( __mips__ )
27.         #define ABI "mips"
28.     #else
29.         #define ABI "unknown"
30.     #endif
31.
32.     return (*env)->NewStringUTF(env, "Hello from JNI ! Compiled with ABI " ABI ".");
33. }
```

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]
1. #include <string.h>
2. #include <jni.h>
3. #include "libavcodec/avcodec.h"
4.
5. jstring
6. Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
7.                                                    jobject this )
8. {
9.     char info[10000] = { 0 };
10.    sprintf(info, "%s\n", avcodec_configuration());
11.    return (*env)->NewStringUTF(env, info);
12. }
```

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]
1. LOCAL_PATH := $(call my-dir)
2.
3. include $(CLEAR_VARS)
4.
5. LOCAL_MODULE     := hello-jni
6. LOCAL_SRC_FILES := hello-jni.c
7.
8. 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]
1.  LOCAL_PATH := $(call my-dir)
2.
3.  # Ffmpeg library
4.  include $(CLEAR_VARS)
5.  LOCAL_MODULE := avcodec
6.  LOCAL_SRC_FILES := libavcodec-56.so
7.  include $(PREBUILT_SHARED_LIBRARY)
8.
9.  include $(CLEAR_VARS)
10. LOCAL_MODULE := avdevice
11. LOCAL_SRC_FILES := libavdevice-56.so
12. include $(PREBUILT_SHARED_LIBRARY)
13.
14. include $(CLEAR_VARS)
15. LOCAL_MODULE := avfilter
16. LOCAL_SRC_FILES := libavfilter-5.so
17. include $(PREBUILT_SHARED_LIBRARY)
18.
19. include $(CLEAR_VARS)
20. LOCAL_MODULE := avformat
21. LOCAL_SRC_FILES := libavformat-56.so
22. include $(PREBUILT_SHARED_LIBRARY)
23.
24. include $(CLEAR_VARS)
25. LOCAL_MODULE := avutil
26. LOCAL_SRC_FILES := libavutil-54.so
27. include $(PREBUILT_SHARED_LIBRARY)
28.
29. include $(CLEAR_VARS)
30. LOCAL_MODULE := postproc
31. LOCAL_SRC_FILES := libpostproc-53.so
32. include $(PREBUILT_SHARED_LIBRARY)
33.
34. include $(CLEAR_VARS)
35. LOCAL_MODULE := swresample
36. LOCAL_SRC_FILES := libswresample-1.so
37. include $(PREBUILT_SHARED_LIBRARY)
38.
39. include $(CLEAR_VARS)
40. LOCAL_MODULE := swscale
41. LOCAL_SRC_FILES := libswscale-3.so
42. include $(PREBUILT_SHARED_LIBRARY)
43.
44. # Program
45. include $(CLEAR_VARS)
46. LOCAL_MODULE := hello-jni
47. LOCAL_SRC_FILES := hello-jni.c
48. LOCAL_C_INCLUDES += $(LOCAL_PATH)/include
49. LOCAL_LDLIBS := -llog -lz
50. LOCAL_SHARED_LIBRARIES := avcodec avdevice avfilter avformat avutil postproc swresample swscale
51. 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：

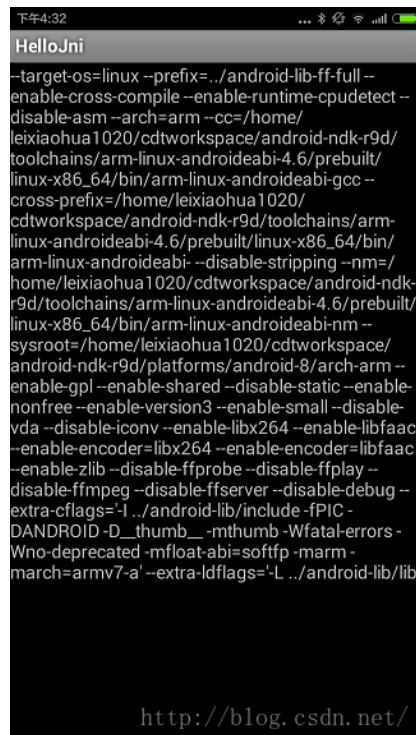
```
libavformat-56.so
libavcodec-56.so
libavfilter-5.so
libavdevice-56.so
libavutil-54.so
libpostproc-53.so
libswresample-1.so
libswscale-3.so
libhello-jni.so
```

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。

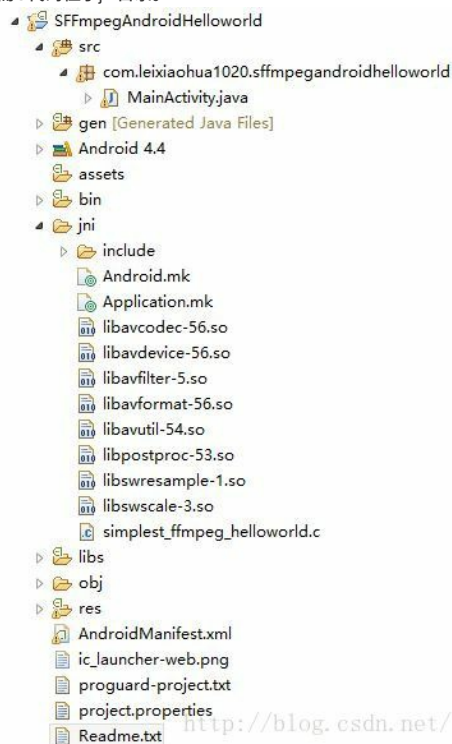
## FFmpeg Helloworld

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。

```
1.  /**
2.   * 最简单的基于FFmpeg的HelloWorld例子-安卓
3.   * Simplest FFmpeg Android HelloWorld
4.   *
5.   * 雷霄骅 Lei Xiaohua
6.   * leixiaohua1020@126.com
7.   * 中国传媒大学/数字电视技术
8.   * Communication University of China / Digital TV Technology
9.   * http://blog.csdn.net/leixiaohua1020
10.  *
11.  *
12.  * 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：
13.  * Protocol: FFmpeg类库支持的协议
14.  * AVFormat: FFmpeg类库支持的封装格式
15.  * AVCodec: FFmpeg类库支持的编解码器
16.  * AVFilter: FFmpeg类库支持的滤镜
17.  * Configure: FFmpeg类库的配置信息
18.  *
19.  * This is the simplest program based on FFmpeg in Android. It can show following
20.  * informations about FFmpeg library:
21.  * Protocol: Protocols supported by FFmpeg.
22.  * AVFormat: Container format supported by FFmpeg.
23.  * AVCodec: Encoder/Decoder supported by FFmpeg.
24.  * AVFilter: Filters supported by FFmpeg.
25.  * Configure: configure information of FFmpeg.
26.  *
27.  */
28.
29.
30. package com.leixiaohua1020.sffmpegandroidhelloworld;
31.
32.
33. import android.os.Bundle;
34. import android.app.Activity;
35. import android.text.method.ScrollingMovementMethod;
36. import android.util.Log;
37. import android.view.Menu;
38. import android.view.View;
39. import android.view.View.OnClickListener;
40. import android.widget.Button;
41. import android.widget.TextView;
42.
43. public class MainActivity extends Activity {
44.
45.     @Override
46.     protected void onCreate(Bundle savedInstanceState) {
47.         super.onCreate(savedInstanceState);
48.         setContentView(R.layout.activity_main);
49.
50.         final TextView libinfoText = (TextView) findViewById(R.id.text_libinfo);
51.         libinfoText.setMovementMethod(ScrollingMovementMethod.getInstance());
52.
53.         libinfoText.setText(configurationinfo());
54.
55.         Button configurationButton = (Button) this.findViewById(R.id.button_configuration);
56.         Button urlprotocolButton = (Button) this.findViewById(R.id.button_urlprotocol);
57.         Button avformatButton = (Button) this.findViewById(R.id.button_avformat);
58.         Button avcodecButton = (Button) this.findViewById(R.id.button_avcodec);
59.         Button avfilterButton = (Button) this.findViewById(R.id.button_avfilter);
60.
61.         urlprotocolButton.setOnClickListener(new OnClickListener() {
62.             public void onClick(View arg0){
63.                 libinfoText.setText(urlprotocolinfo());
64.             }
65.         });
66.
67.         avformatButton.setOnClickListener(new OnClickListener() {
68.             public void onClick(View arg0){
69.                 libinfoText.setText(avformatinfo());
70.             }
71.         });
72.
73.         avcodecButton.setOnClickListener(new OnClickListener() {
74.             public void onClick(View arg0){
75.                 libinfoText.setText(avcodecinfo());
76.             }
77.         });
78.
79.         avfilterButton.setOnClickListener(new OnClickListener() {
80.             public void onClick(View arg0){
81.                 libinfoText.setText(avfilterinfo());
82.             }
83.         });
84.
85.         configurationButton.setOnClickListener(new OnClickListener() {
86.             public void onClick(View arg0){
87.                 libinfoText.setText(configurationinfo());
88.             }
89.         });
90.     }
91. }
```

```

91.     }
92.
93.     @Override
94.     public boolean onCreateOptionsMenu(Menu menu) {
95.         // Inflate the menu; this adds items to the action bar if it is present.
96.         getMenuInflater().inflate(R.menu.main, menu);
97.         return true;
98.     }
99.
100.    //JNI
101.    public native String urlprotocolinfo();
102.    public native String avformatinfo();
103.    public native String avcodecinfo();
104.    public native String avfilterinfo();
105.    public native String configurationinfo();
106.
107.    static{
108.        System.loadLibrary("avutil-54");
109.        System.loadLibrary("swresample-1");
110.        System.loadLibrary("avcodec-56");
111.        System.loadLibrary("avformat-56");
112.        System.loadLibrary("swscale-3");
113.        System.loadLibrary("postproc-53");
114.        System.loadLibrary("avfilter-5");
115.        System.loadLibrary("avdevice-56");
116.        System.loadLibrary("sffmpeghelloworld");
117.    }
118.
119. }

```

C语言源代码位于jni/simplest\_ffmpeg\_helloworld.c，如下所示。

```

1.  /**
2.   * 最简单的基于FFmpeg的HelloWorld例子-安卓
3.   * Simplest FFmpeg Android HelloWorld
4.   *
5.   * 雷霄骅 Lei Xiaohua
6.   * leixiaohua1020@126.com
7.   * 中国传媒大学/数字电视技术
8.   * Communication University of China / Digital TV Technology
9.   * http://blog.csdn.net/leixiaohua1020
10.  *
11.  *
12.  * 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：
13.  * Protocol:   FFmpeg类库支持的协议
14.  * AVFormat:   FFmpeg类库支持的封装格式
15.  * AVCodec:    FFmpeg类库支持的编解码器
16.  * AVFilter:   FFmpeg类库支持的滤镜
17.  * Configure:  FFmpeg类库的配置信息
18.  *
19.  * This is the simplest program based on FFmpeg in Android. It can show following
20.  * informations about FFmpeg library:
21.  * Protocol:   Protocols supported by FFmpeg.
22.  * AVFormat:   Container format supported by FFmpeg.
23.  * AVCodec:    Encoder/Decoder supported by FFmpeg.
24.  * AVFilter:   Filters supported by FFmpeg.
25.  * Configure:  configure information of FFmpeg.
26.  */
27.
28. #include <stdio.h>
29.
30. #include "libavcodec/avcodec.h"
31. #include "libavformat/avformat.h"
32. #include "libavfilter/avfilter.h"
33.
34. //Log
35. #ifdef ANDROID
36. #include <jni.h>
37. #include <android/log.h>
38. #define LOGE(format, ...) __android_log_print(ANDROID_LOG_ERROR, ">_<", format, ## __VA_ARGS__)
39. #else
40. #define LOGE(format, ...) printf(">_< " format "\n", ## __VA_ARGS__)
41. #endif
42.
43.
44. //FIX
45. struct URLProtocol;
46. /**
47.  * com.leixiaohua1020.sffmpeghelloworld.MainActivity.urlprotocolinfo()
48.  * Protocol Support Information
49.  */
50. JNIEXPORT jstring Java_com_leixiaohua1020_sffmpeghelloworld_MainActivity_urlprotocolinfo(JNIEnv *env, jobject obj){
51.
52.     char info[40000]={0};
53.     av_register_all();
54.
55.     struct URLProtocol *pup = NULL;

```

```

56. //Input
57. struct URLProtocol **p_temp = &pup;
58. avio_enum_protocols((void **)p_temp, 0);
59. while ((*p_temp) != NULL){
60.     sprintf(info, "%s[In ][%10s]\n", info, avio_enum_protocols((void **)p_temp, 0));
61. }
62. pup = NULL;
63. //Output
64. avio_enum_protocols((void **)p_temp, 1);
65. while ((*p_temp) != NULL){
66.     sprintf(info, "%s[Out][%10s]\n", info, avio_enum_protocols((void **)p_temp, 1));
67. }
68.
69. //LOGE("%s", info);
70. return (*env)->NewStringUTF(env, info);
71. }
72.
73. /**
74.  * com.leixiaohua1020.sffmpegeandroidhelloworld.MainActivity.avformatinfo()
75.  * AVFormat Support Information
76.  */
77. JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegeandroidhelloworld_MainActivity_avformatinfo(JNIEnv *env, jobject obj){
78.
79.     char info[40000] = { 0 };
80.
81.     av_register_all();
82.
83.     AVInputFormat *if_temp = av_iformat_next(NULL);
84.     AVOutputFormat *of_temp = av_oformat_next(NULL);
85.     //Input
86.     while(if_temp!=NULL){
87.         sprintf(info, "%s[In ][%10s]\n", info, if_temp->name);
88.         if_temp=if_temp->next;
89.     }
90.     //Output
91.     while (of_temp != NULL){
92.         sprintf(info, "%s[Out][%10s]\n", info, of_temp->name);
93.         of_temp = of_temp->next;
94.     }
95.     //LOGE("%s", info);
96.     return (*env)->NewStringUTF(env, info);
97. }
98.
99. /**
100.  * com.leixiaohua1020.sffmpegeandroidhelloworld.MainActivity.avcodecinfo()
101.  * AVCodec Support Information
102.  */
103. JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegeandroidhelloworld_MainActivity_avcodecinfo(JNIEnv *env, jobject obj)
104. {
105.     char info[40000] = { 0 };
106.
107.     av_register_all();
108.
109.     AVCodec *c_temp = av_codec_next(NULL);
110.
111.     while(c_temp!=NULL){
112.         if (c_temp->decode!=NULL){
113.             sprintf(info, "%s[Dec]", info);
114.         }
115.         else{
116.             sprintf(info, "%s[Enc]", info);
117.         }
118.         switch (c_temp->type){
119.             case AVMEDIA_TYPE_VIDEO:
120.                 sprintf(info, "%s[Video]", info);
121.                 break;
122.             case AVMEDIA_TYPE_AUDIO:
123.                 sprintf(info, "%s[Audio]", info);
124.                 break;
125.             default:
126.                 sprintf(info, "%s[Other]", info);
127.                 break;
128.         }
129.         sprintf(info, "%s[%10s]\n", info, c_temp->name);
130.
131.
132.         c_temp=c_temp->next;
133.     }
134.     //LOGE("%s", info);
135.
136.     return (*env)->NewStringUTF(env, info);
137. }
138.
139. /**
140.  * com.leixiaohua1020.sffmpegeandroidhelloworld.MainActivity.avfilterinfo()
141.  * AVFilter Support Information
142.  */
143. JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegeandroidhelloworld_MainActivity_avfilterinfo(JNIEnv *env, jobject obj)
144. {
145.     char info[40000] = { 0 };
146.     avfilter_register_all();

```

```

147.     AVFilter *f_temp = (AVFilter *)avfilter_next(NULL);
148.     while (f_temp != NULL){
149.         sprintf(info, "%s[%10s]\n", info, f_temp->name);
150.     }
151.     //LOGE("%s", info);
152.
153.     return (*env)->NewStringUTF(env, info);
154. }
155.
156. /**
157.  * com.leixiaohua1020.sffmpegandroidhelloworld.MainActivity.urlprotocolinfo()
158.  * Protocol Support Information
159.  */
160. JNIEXPORT jstring Java_com_leixiaohua1020_sffmpegandroidhelloworld_MainActivity_configurationinfo(JNIEnv *env, jobject obj)
161. {
162.     char info[10000] = { 0 };
163.     av_register_all();
164.
165.     sprintf(info, "%s\n", avcodec_configuration());
166.
167.     //LOGE("%s", info);
168.     return (*env)->NewStringUTF(env, info);
169. }

```

Android.mk文件位于jni/Android.mk，如下所示。

```

[plain]
1.  # Android.mk for FFmpeg
2.  #
3.  # Lei Xiaohua 雷霄骅
4.  # leixiaohua1020@126.com
5.  # http://blog.csdn.net/leixiaohua1020
6.  #
7.
8.  LOCAL_PATH := $(call my-dir)
9.
10. # FFmpeg library
11. include $(CLEAR_VARS)
12. LOCAL_MODULE := avcodec
13. LOCAL_SRC_FILES := libavcodec-56.so
14. include $(PREBUILT_SHARED_LIBRARY)
15.
16. include $(CLEAR_VARS)
17. LOCAL_MODULE := avdevice
18. LOCAL_SRC_FILES := libavdevice-56.so
19. include $(PREBUILT_SHARED_LIBRARY)
20.
21. include $(CLEAR_VARS)
22. LOCAL_MODULE := avfilter
23. LOCAL_SRC_FILES := libavfilter-5.so
24. include $(PREBUILT_SHARED_LIBRARY)
25.
26. include $(CLEAR_VARS)
27. LOCAL_MODULE := avformat
28. LOCAL_SRC_FILES := libavformat-56.so
29. include $(PREBUILT_SHARED_LIBRARY)
30.
31. include $(CLEAR_VARS)
32. LOCAL_MODULE := avutil
33. LOCAL_SRC_FILES := libavutil-54.so
34. include $(PREBUILT_SHARED_LIBRARY)
35.
36. include $(CLEAR_VARS)
37. LOCAL_MODULE := postproc
38. LOCAL_SRC_FILES := libpostproc-53.so
39. include $(PREBUILT_SHARED_LIBRARY)
40.
41. include $(CLEAR_VARS)
42. LOCAL_MODULE := swresample
43. LOCAL_SRC_FILES := libswresample-1.so
44. include $(PREBUILT_SHARED_LIBRARY)
45.
46. include $(CLEAR_VARS)
47. LOCAL_MODULE := swscale
48. LOCAL_SRC_FILES := libswscale-3.so
49. include $(PREBUILT_SHARED_LIBRARY)
50.
51. # Program
52. include $(CLEAR_VARS)
53. LOCAL_MODULE := sffhelloworld
54. LOCAL_SRC_FILES :=simplest_ffmpeg_helloworld.c
55. LOCAL_C_INCLUDES += $(LOCAL_PATH)/include
56. LOCAL_LDLIBS := -llog -lz
57. LOCAL_SHARED_LIBRARIES := avcodec avdevice avfilter avformat avutil postproc swresample swscale
58. include $(BUILD_SHARED_LIBRARY)

```

其它部分源代码暂不详细列举。

## 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](https://github.com/leixiaohua1020/simplest_ffmpeg_mobile)

开源中国：[https://git.oschina.net/leixiaohua1020/simplest\\_ffmpeg\\_mobile](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

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com