最简单的基于FFmpeg的移动端例子：IOS HelloWorld

2015年07月27日 20:18:17　阅读数：29879

==========

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

==========

本文记录IOS平台下基于FFmpeg的HelloWorld程序。该示例C语言的源代码来自于《 最简单的基于FFMPEG的Helloworld程序 》。相关的概念就不再重复记录了。

IOS程序使用FFmpeg类库的说明

IOS应用程序使用FFmpeg类库的流程如下所示。

1. 编译FFmpeg类库

编译IOS的FFmpeg类库需要支持5种架构：armv7、armv7s、arm64、i386、x86_64。其中前面3个是给真机使用的，后面2个是给模拟器使用的。本文记录的FFmpeg类库还支持第三方类库libx264和libfaac，所以在编译之前还要先编译libx264和libfaac的源代码。总体说来，IOS下的类库需要编译成两个版本：thin和fat。每种架构对应一个thin版本的类库，将这些不同架构thin版本的类库合成起来之后，就形成了fat版本的类库。下面简单记录一下编译步骤。编译过程中IOS SDK版本为8.3，FFmpeg版本为2.7.1，faac和x264分别使用了最新版本的源代码。

(1) 第三方库libx264的编译

这一步用于生成支持armv7、armv7s、arm64、i386、x86_64几种架构的fat版本的libx264.a。下面这个脚本可以首先编译生成上面5种架构的thin版本的libx264.a，分成5个文件夹存储于thin-x264文件夹中；然后将这些类库合并成为1个fat版本的libx264.a，存储于fat-x264文件夹中。

build_x264.sh

```
[plain]
```

```
#!/bin/sh
# LXH,MXY
#
# directories
SOURCE="x264"
FAT="fat-x264"

SCRATCH="scratch-x264"
# must be an absolute path
THIN=`pwd`/"thin-x264"

#This is decided by your SDK version.
SDK_VERSION="8.3"

cd ./x264

#============== simulator ===============
PLATFORM="iPhoneSimulator"

#i386
ARCHS="i386"

export DEVROOT=/Applications/Xcode.app/Contents/Developer/Platforms/${PLATFORM}.platform/Developer
export SDKROOT=$DEVROOT/SDKs/${PLATFORM}${SDK_VERSION}.sdk
export CC=$DEVROOT/usr/bin/gcc
export LD=$DEVROOT/usr/bin/ld
export CXX=$DEVROOT/usr/bin/g++
export LIBTOOL=$DEVROOT/usr/bin/libtool
export HOST=i386-apple-darwin

COMMONFLAGS="-pipe -gdwarf-2 -no-cpp-precomp -isysroot ${SDKROOT} -fPIC"
export LDFLAGS="${COMMONFLAGS} -fPIC"
export CFLAGS="${COMMONFLAGS} -fvisibility=hidden"


for ARCH in $ARCHS; do

echo "Building $ARCH ......"

make clean
./configure \
--host=i386-apple-darwin \
--sysroot=$SDKROOT \
--prefix="$THIN/$ARCH" \
--extra-cflags="-arch $ARCH -miphoneos-version-min=6.0" \
--extra-ldflags="-L$SDKROOT/usr/lib/system -arch $ARCH -miphoneos-version-min=6.0" \
--enable-pic \
--enable-static \
--disable-asm \
make && make install && make clean

echo "Installed: $DEST/$ARCH"
done

#x86_64

ARCHS="x86_64"

unset DEVROOT
unset SDKROOT
unset CC
unset LD
unset CXX
unset LIBTOOL
unset HOST
unset LDFLAGS
unset CFLAGS

make clean
for ARCH in $ARCHS; do

echo "Building $ARCH ......"

./configure \
--prefix="$THIN/$ARCH" \
--enable-pic \
--enable-static \
--disable-asm \
make && make install && make clean

echo "Installed: $DEST/$ARCH"
done

#================ iphone ==================

export PLATFORM="iPhoneOS"

ARCHS="arm64 armv7 armv7s "

export DEVROOT=/Applications/Xcode.app/Contents/Developer
export SDKROOT=$DEVROOT/Platforms/${PLATFORM}.platform/Developer/SDKs/${PLATFORM}${SDK_VERSION}.sdk
```

```
#DEVPATH=/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS${SDK_VERSION}.sdk
export CC=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang
export AS=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/as
COMMONFLAGS="-pipe -gdwarf-2 -no-cpp-precomp -isysroot ${SDKROOT} -marm -fPIC"
export LDFLAGS="${COMMONFLAGS} -fPIC"
export CFLAGS="${COMMONFLAGS} -fvisibility=hidden"
export CXXFLAGS="${COMMONFLAGS} -fvisibility=hidden -fvisibility-inlines-hidden"


for ARCH in $ARCHS; do

echo "Building $ARCH ......"

./configure \
--host=arm-apple-darwin \
--sysroot=$DEVPATH \
--prefix="$THIN/$ARCH" \
--extra-cflags="-arch $ARCH" \
--extra-ldflags="-L$DEVPATH/usr/lib/system -arch $ARCH" \
--enable-pic \
--enable-static \
--disable-asm

make && make install && make clean

echo "Installed: $DEST/$ARCH"

done

cd ..

#================ fat lib ==================

ARCHS="armv7 armv7s i386 x86_64 arm64"

echo "building fat binaries..."
mkdir -p $FAT/lib
set - $ARCHS
CWD=`pwd`
cd $THIN/$1/lib
for LIB in *.a
do
cd $CWD
lipo -create `find $THIN -name $LIB` -output $FAT/lib/$LIB
done

cd $CWD
cp -rf $THIN/$1/include $FAT
```

(2) 第三方库libfaac的编译

这一步用于生成支持armv7、armv7s、arm64、i386、x86_64几种架构的fat版本的libfaac.a。下面这个脚本可以首先编译生成上面5种架构的thin版本的libfaac.a，分成5个文件夹存储于fat-faac中；然后将这些类库合并成为1个fat版本的libfaac.a，存储于fat-faac中。

build_faac.sh

```
[plain]
#!/bin/sh
cd ./faac
make distclean
cd ..

CONFIGURE_FLAGS="--enable-static --with-pic"

ARCHS="arm64 armv7s x86_64 i386 armv7"

# directories
SOURCE="faac"
FAT="fat-faac"

SCRATCH="scratch-faac"
# must be an absolute path
THIN=`pwd`/"thin-faac"

COMPILE="y"
LIPO="y"

if [ "$*" ]
then
if [ "$*" = "lipo" ]
then
# skip compile
COMPILE=
else
ARCHS="$*"
if [ $# -eq 1 ]
then
```

```
# skip lipo
LIPO=
fi
fi
fi

if [ "$COMPILE" ]
then
CWD=`pwd`
for ARCH in $ARCHS
do
echo "building $ARCH..."
mkdir -p "$SCRATCH/$ARCH"
cd "$SCRATCH/$ARCH"

if [ "$ARCH" = "i386" -o "$ARCH" = "x86_64" ]
then
PLATFORM="iPhoneSimulator"
CPU=
if [ "$ARCH" = "x86_64" ]
then
SIMULATOR="-mios-simulator-version-min=7.0"
HOST=
else
SIMULATOR="-mios-simulator-version-min=5.0"
HOST="--host=i386-apple-darwin"
fi
else
PLATFORM="iPhoneOS"
if [ $ARCH = "armv7s" ]
then
CPU="--cpu=swift"
else
CPU=
fi
SIMULATOR=
HOST="--host=arm-apple-darwin"
fi

XCRUN_SDK=`echo $PLATFORM | tr '[:upper:]' '[:lower:]'`
CC="xcrun -sdk $XCRUN_SDK clang -Wno-error=unused-command-line-argument-hard-error-in-future"
AS="/usr/local/bin/gas-preprocessor.pl $CC"
CFLAGS="-arch $ARCH $SIMULATOR"
CXXFLAGS="$CFLAGS"
LDFLAGS="$CFLAGS"

CC=$CC CFLAGS=$CXXFLAGS LDFLAGS=$LDFLAGS CPPFLAGS=$CXXFLAGS CXX=$CC CXXFLAGS=$CXXFLAGS  $CWD/$SOURCE/configure \
$CONFIGURE_FLAGS \
$HOST \
--prefix="$THIN/$ARCH" \
--disable-shared \
--without-mp4v2

make clean && make && make install-strip
cd $CWD
done
fi

#================ fat lib ===================

echo "building fat binaries..."
mkdir -p $FAT/lib
set - $ARCHS
CWD=`pwd`
cd $THIN/$1/lib
for LIB in *.a
do
cd $CWD
lipo -create `find $THIN -name $LIB` -output $FAT/lib/$LIB
done

cd $CWD
cp -rf $THIN/$1/include $FAT
```

(3) 编译armv7版本FFmpeg类库

这一步用于生成支持armv7架构的thin版本的FFmpeg类库，存储于thin-ffmpeg/armv7文件夹中。脚本如下所示。

build_ffmpeg_demo_armv7.sh

```plain
#!/bin/sh
# LXH,MXY modified

cd ffmpeg

PLATFORM="iPhoneOS"
INSTALL="thin-ffmpeg"
SDK_VERSION="8.3"

# libx264
export X264ROOT=../thin-x264/armv7
export X264LIB=$X264ROOT/lib
export X264INCLUDE=$X264ROOT/include
# libfaac
export FAACROOT=../thin-faac/armv7
export FAACLIB=$FAACROOT/lib
export FAACINCLUDE=$FAACROOT/include

export DEVROOT=/Applications/Xcode.app/Contents/Developer

export SDKROOT=$DEVROOT/Platforms/${PLATFORM}.platform/Developer/SDKs/${PLATFORM}${SDK_VERSION}.sdk
export CC=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang
export AS=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/as

COMMONFLAGS="-pipe -gdwarf-2 -no-cpp-precomp -isysroot ${SDKROOT} -marm -fPIC"
export LDFLAGS="${COMMONFLAGS} -fPIC"
export CFLAGS="${COMMONFLAGS} -fvisibility=hidden"
export CXXFLAGS="${COMMONFLAGS} -fvisibility=hidden -fvisibility-inlines-hidden"


echo "Building armv7..."

make clean
./configure \
    --cpu=cortex-a9 \
    --extra-cflags='-I$X264INCLUDE -I$FAACINCLUDE -arch armv7 -miphoneos-version-min=5.0 -mthumb' \
    --extra-ldflags='-L$X264LIB -L$FAACLIB -arch armv7 -miphoneos-version-min=5.0' \
    --enable-cross-compile \
    --arch=arm --disable-iconv\
    --target-os=darwin \
    --cc=${CC} --disable-asm\
    --sysroot=${SDKROOT} \
    --prefix=../${INSTALL}/armv7 \
--enable-gpl --enable-nonfree --enable-version3 --disable-bzlib --enable-small --disable-vda \
--disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=libfaac \
--disable-muxers --enable-muxer=flv --enable-muxer=mov --enable-muxer=ipod --enable-muxer=mpegts --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=flv --enable-demuxer=h264 --enable-demuxer=mpegts --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 --enable-protocol=rtmp --enable-protocol=rtp --enable-protocol=udp \
--disable-bsfs --enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
--disable-devices --disable-debug --disable-ffmpeg --disable-ffprobe --disable-ffplay --disable-ffserver --disable-debug

make
make install

cd ..
```

(4) 编译armv7s版本FFmpeg类库

这一步用于生成支持armv7s架构的thin版本的FFmpeg类库，存储于thin-ffmpeg/armv7s文件夹中。脚本如下所示。

build_ffmpeg_demo_armv7s.sh

```plain
#!/bin/sh
# LXH,MXY modified

cd ffmpeg

PLATFORM="iPhoneOS"
INSTALL="thin-ffmpeg"
SDK_VERSION="8.3"

# libx264
export X264ROOT=../thin-x264/armv7s
export X264LIB=$X264ROOT/lib
export X264INCLUDE=$X264ROOT/include
# libfaac
export FAACROOT=../thin-faac/armv7s
export FAACLIB=$FAACROOT/lib
export FAACINCLUDE=$FAACROOT/include

export DEVROOT=/Applications/Xcode.app/Contents/Developer

export SDKROOT=$DEVROOT/Platforms/${PLATFORM}.platform/Developer/SDKs/${PLATFORM}${SDK_VERSION}.sdk
export CC=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang
export AS=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/as

COMMONFLAGS="-pipe -gdwarf-2 -no-cpp-precomp -isysroot ${SDKROOT} -marm -fPIC"
export LDFLAGS="${COMMONFLAGS} -fPIC"
export CFLAGS="${COMMONFLAGS} -fvisibility=hidden"
export CXXFLAGS="${COMMONFLAGS} -fvisibility=hidden -fvisibility-inlines-hidden"


echo "Building armv7s..."

make clean
./configure \
    --cpu=cortex-a9 \
    --extra-cflags='-I$X264INCLUDE -I$FAACINCLUDE -arch armv7s -miphoneos-version-min=5.0 -mthumb' \
    --extra-ldflags='-L$X264LIB -L$FAACLIB -arch armv7s -miphoneos-version-min=5.0' \
    --enable-cross-compile \
    --arch=arm --disable-iconv\
    --target-os=darwin \
    --cc=${CC} --disable-asm \
    --sysroot=${SDKROOT} \
    --prefix=../${INSTALL}/armv7s \
--enable-gpl --enable-nonfree --enable-version3 --disable-bzlib --enable-small --disable-vda \
 --disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=libfaac \
 --disable-muxers --enable-muxer=flv --enable-muxer=mov --enable-muxer=ipod --enable-muxer=mpegts --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=flv --enable-demuxer=h264 --enable-demuxer=avi --enable-demuxer=mpegts --enable-demuxer=mpc --enable-demuxer=mov \
 --disable-parsers --enable-parser=aac --enable-parser=ac3 --enable-parser=h264 \
 --disable-protocols --enable-protocol=file --enable-protocol=rtmp --enable-protocol=rtp --enable-protocol=udp \
 --disable-bsfs --enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
 --disable-devices --disable-debug --disable-ffmpeg --disable-ffprobe --disable-ffplay --disable-ffserver --disable-debug

make
make install

cd ..
```

(5) 编译arm64版本FFmpeg类库

这一步用于生成支持arm64架构的thin版本的FFmpeg类库，存储于thin-ffmpeg/arm64文件夹中。脚本如下所示。

build_ffmpeg_demo_arm64.sh

```plain
#!/bin/sh
# LXH,MXY modified

cd ffmpeg

PLATFORM="iPhoneOS"
INSTALL="thin-ffmpeg"
SDK_VERSION="8.3"

# libx264
export X264ROOT=../thin-x264/arm64
export X264LIB=$X264ROOT/lib
export X264INCLUDE=$X264ROOT/include
# libfaac
export FAACROOT=../thin-faac/arm64
export FAACLIB=$FAACROOT/lib
export FAACINCLUDE=$FAACROOT/include

export DEVROOT=/Applications/Xcode.app/Contents/Developer

export SDKROOT=$DEVROOT/Platforms/${PLATFORM}.platform/Developer/SDKs/${PLATFORM}${SDK_VERSION}.sdk
export CC=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang
export AS=$DEVROOT/Toolchains/XcodeDefault.xctoolchain/usr/bin/as

COMMONFLAGS="-pipe -gdwarf-2 -no-cpp-precomp -isysroot ${SDKROOT} -marm -fPIC"
export LDFLAGS="${COMMONFLAGS} -fPIC"
export CFLAGS="${COMMONFLAGS} -fvisibility=hidden"
export CXXFLAGS="${COMMONFLAGS} -fvisibility=hidden -fvisibility-inlines-hidden"


echo "Building arm64..."

make clean
./configure \
    --extra-cflags='-I$X264INCLUDE -I$FAACINCLUDE -arch arm64 -miphoneos-version-min=5.0 -mthumb' \
    --extra-ldflags='-L$X264LIB -L$FAACLIB -arch arm64 -miphoneos-version-min=5.0' \
    --enable-cross-compile \
    --arch=arm --disable-iconv \
    --target-os=darwin \
    --cc=${CC} --disable-asm \
    --sysroot=${SDKROOT} \
    --prefix=../${INSTALL}/arm64 \
--enable-gpl --enable-nonfree --enable-version3 --disable-bzlib --enable-small --disable-vda \
--disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=libfaac \
--disable-muxers --enable-muxer=flv --enable-muxer=mov --enable-muxer=ipod --enable-muxer=mpegts --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=flv --enable-demuxer=h264 --enable-demuxer=avi --enable-demuxer=mpegts --enable-demuxer=mpc --enable-demuxer=mov \
--disable-parsers --enable-parser=aac --enable-parser=ac3 --enable-parser=h264 \
--disable-protocols --enable-protocol=file --enable-protocol=rtmp --enable-protocol=rtp --enable-protocol=udp \
--disable-bsfs --enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
--disable-devices --disable-debug --disable-ffmpeg --disable-ffprobe --disable-ffplay --disable-ffserver --disable-debug

make
make install

cd ..
```

(6) 编译i386版本FFmpeg类库

这一步用于生成支持i386架构的thin版本的FFmpeg类库，存储于thin-ffmpeg/i386文件夹中。脚本如下所示。

build_ffmpeg_demo_i386.sh

```plain
#!/bin/sh
# LXH,MXY modified

cd ffmpeg

PLATFORM="iPhoneSimulator"
INSTALL="thin-ffmpeg"
SDK_VERSION="8.3"

# libx264
export X264ROOT=../thin-x264/i386
export X264LIB=$X264ROOT/lib
export X264INCLUDE=$X264ROOT/include
# libfaac
export FAACROOT=../thin-faac/i386
export FAACLIB=$FAACROOT/lib
export FAACINCLUDE=$FAACROOT/include

export DEVROOT=/Applications/Xcode.app/Contents/Developer/Platforms/${PLATFORM}.platform/Developer
export SDKROOT=$DEVROOT/SDKs/${PLATFORM}${SDK_VERSION}.sdk
export CC=$DEVROOT/usr/bin/gcc
export LD=$DEVROOT/usr/bin/ld

export CXX=$DEVROOT/usr/bin/g++

export LIBTOOL=$DEVROOT/usr/bin/libtool

COMMONFLAGS="-pipe -gdwarf-2 -no-cpp-precomp -isysroot ${SDKROOT} -fPIC"
export LDFLAGS="${COMMONFLAGS} -fPIC"
export CFLAGS="${COMMONFLAGS} -fvisibility=hidden"


echo "Building i386..."
make clean
./configure \
    --cpu=i386 \
    --extra-cflags='-I$X264INCLUDE -I$FAACINCLUDE -arch i386 -miphoneos-version-min=5.0' \
    --extra-ldflags='-L$X264LIB -L$FAACLIB -arch i386 -miphoneos-version-min=5.0' \
    --enable-cross-compile \
    --arch=i386 --disable-iconv \
    --target-os=darwin \
    --cc=${CC} \
    --sysroot=${SDKROOT} \
--prefix=../${INSTALL}/i386 \
--enable-gpl --enable-nonfree --enable-version3 --disable-bzlib --enable-small --disable-vda \
--disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=libfaac \
--disable-muxers --enable-muxer=flv --enable-muxer=mov --enable-muxer=mpegts --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=flv --enable-demuxer=h264 --enable-demuxer=mpegts --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 --enable-protocol=rtmp --enable-protocol=rtp --enable-protocol=udp \
--disable-bsfs --enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
--disable-devices --disable-debug --disable-ffmpeg --disable-ffprobe --disable-ffplay --disable-ffserver --disable-debug

make
make install

cd ..
```

(7) 编译x86_64版本FFmpeg类库

这一步用于生成支持x86_64架构的thin版本的FFmpeg类库，存储于thin-ffmpeg/x86_64文件夹中。脚本如下所示。

build_ffmpeg_demo_x86_64.sh

```plain
#!/bin/sh
# LXH,MXY modified

cd ./ffmpeg

INSTALL="thin-ffmpeg"

# libx264
export X264ROOT=../thin-x264/x86_64
export X264LIB=$X264ROOT/lib
export X264INCLUDE=$X264ROOT/include
# libfaac
export FAACROOT=../thin-faac/x86_64
export FAACLIB=$FAACROOT/lib
export FAACINCLUDE=$FAACROOT/include

unset DEVROOT
unset SDKROOT
unset CC
unset LD
unset CXX
unset LIBTOOL
unset HOST
unset LDFLAGS
unset CFLAGS

echo "Building x86_64..."

make clean
./configure \
    --extra-cflags='-I$X264INCLUDE -I$FAACINCLUDE' \
    --extra-ldflags='-L$X264LIB -L$FAACLIB' \
    --disable-iconv \
    --disable-asm \
--prefix=../${INSTALL}/x86_64 \
--enable-gpl --enable-nonfree --enable-version3 --disable-bzlib --enable-small --disable-vda \
--disable-encoders --enable-libx264 --enable-libfaac --enable-encoder=libx264 --enable-encoder=mpeg2video --enable-encoder=libfaac \
--disable-muxers --enable-muxer=flv --enable-muxer=mov --enable-muxer=ipod --enable-muxer=mpegts --enable-muxer=psp --enable-muxer=mp4 --enable-muxer=avi \
--disable-decoders --enable-decoder=aac --enable-decoder=mpeg2video --enable-decoder=aac_latm --enable-decoder=h264 --enable-decoder=mpeg4 \
--disable-demuxers --enable-demuxer=flv --enable-demuxer=h264 --enable-demuxer=avi --enable-demuxer=mpegts --enable-demuxer=mpc --enable-demuxer=mov \
--disable-parsers --enable-parser=aac --enable-parser=ac3 --enable-parser=h264 \
--disable-protocols --enable-protocol=file --enable-protocol=rtmp --enable-protocol=rtp --enable-protocol=udp \
--enable-bsf=aac_adtstoasc --enable-bsf=h264_mp4toannexb \
--disable-devices --disable-debug --disable-ffmpeg --disable-ffprobe --disable-ffplay --disable-ffserver --disable-debug

make
make install
cd ..
```

(8) 打包类库

这一步用于将上述步骤中生成的5个版本的FFmpeg打包生成fat版本的FFmpeg类库。这一步骤执行完毕后，将thin-ffmpeg中几个thin版本的类库合并为一个fat版本的类库，并存储于fat-ffmpeg文件夹中。脚本如下所示。

build_ffmpeg_fat.sh

```plain
#!/bin/sh


# directories
THIN=`pwd`/"thin-ffmpeg"
FAT=`pwd`/"fat-ffmpeg"
CWD=`pwd`
# must be an absolute path


ARCHS="armv7s i386 armv7 arm64 x86_64"


echo "building fat binaries..."

mkdir -p $FAT/lib
set - $ARCHS
cd thin-ffmpeg/$1/lib
for LIB in *.a
do
cd $CWD
lipo -create `find $THIN -name $LIB` -output $FAT/lib/$LIB
done

cd $CWD
cp -rf $THIN/$1/include $FAT
```

生成完fat版本的类库后，可以在命令行使用lipo命令查看类库的架构，如下所示。

```plain
lipo -info libavcodec.a
```

2. 编写IOS程序

编写包含FFmpeg类库支持的IOS程序分成两步：配置Xcode环境，编写C语言代码。

(1) 配置Xcode环境

下面以Xcode的IOS中的Single View Application为例，记录一下配置步骤：

(a)

拷贝头文件所在的include文件夹和fat版本的FFmpeg类库(包括libavformat.a，libavcodec.a，libavutil.a，libavdevice.a，libavfilter.a，libpostproc.a， libswresample.a， libswscale.a；以及第三方fat版本类库libx264.a， libfaac.a）至项目文件夹。并将它们添加至项目中。

(b)

项目属性->Build Settings中配置以下3个选项。
Linking->Other Linker Flags中添加下面内容：

```plain
-lavformat
-lavcodec
-lavutil
-lavdevice
-lavfilter
-lpostproc
-lswresample
-lswscale
-lx264
-lfaac
```

Search Paths->Header Search Paths添加下面内容

```plain
$(PROJECT_DIR)/include
```

Search Paths->Library Search Paths添加下面内容

```plain
$(PROJECT_DIR)
```

其它的一些配置。这些配置随着FFmpeg版本的不同而有略微的不同（在某些情况下也可能不需要配置）。我目前使用的2.7.1版本的FFmpeg需要配置下面的选项。
项目属性->General->Linked Frameworks and Libraries中添加两个类库：AVFoundation.framework和libz.dylib。

(2) 编写C语言代码

做好上面配置后，就可以在项目中编写代码测试一下FFmpeg是否正确配置了。由于IOS使用的Objective-C是兼容C语言的，所以可以直接写C语言代码调用FFmpeg。

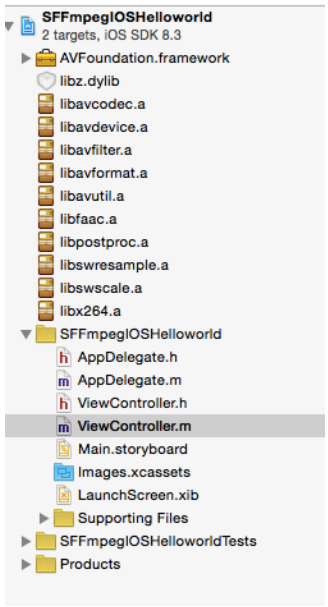
可以在ViewController.m中的viewDidLoad()函数中添加一行printf()代码打印FFmpeg类库的版本信息，如下所示。

```objc
#import "ViewController.h"
#include <libavcodec/avcodec.h>

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    printf("%s",avcodec_configuration());
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

如果类库编译无误，启动IOS程序的时候会在控制台打印版本信息。

源代码

项目的目录结构如图所示。



C代码位于ViewController.m文件中，内容如下所示。

```objc
/**
 * 最简单的基于FFmpeg的HelloWorld程序 － IOS
 * Simplest FFmpeg Helloworld － IOS
 *
 * 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
 *
 * 本程序可以获得FFmpeg类库相关的信息。
 * This software can get information about FFmpeg libraries.
 *
 */

#import "ViewController.h"
#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libavfilter/avfilter.h>

@interface ViewController ()

@end
```

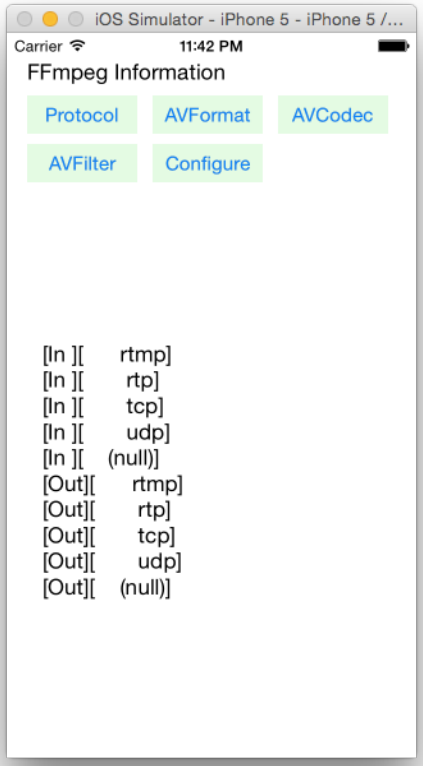
```
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    av_register_all();
    char info[10000] = { 0 };
    printf("%s\n", avcodec_configuration());
    sprintf(info, "%s\n", avcodec_configuration());
    NSString * info_ns = [NSString stringWithFormat:@"%s", info];
    self.content.text=info_ns;
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)clickProtocolButton:(id)sender {
    //Alert
    /*
    UIAlertView *alter = [[UIAlertView alloc] initWithTitle:@"Title" message:@"This is content" delegate:nil cancelButtonTitle:@"Clos
otherButtonTitles:nil];
    [alter show];
     */
    char info[40000]={0};
    av_register_all();

    struct URLProtocol *pup = NULL;
    //Input
    struct URLProtocol **p_temp = &pup;
    avio_enum_protocols((voidvoid **)p_temp, 0);
    while ((*p_temp) != NULL){
        sprintf(info, "%s[In ][%10s]\n", info, avio_enum_protocols((voidvoid **)p_temp, 0));
    }
    pup = NULL;
    //Output
    avio_enum_protocols((voidvoid **)p_temp, 1);
    while ((*p_temp) != NULL){
        sprintf(info, "%s[Out][%10s]\n", info, avio_enum_protocols((voidvoid **)p_temp, 1));
    }
    //printf("%s", info);
    NSString * info_ns = [NSString stringWithFormat:@"%s", info];
    self.content.text=info_ns;
}

- (IBAction)clickAVFormatButton:(id)sender {
    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;
    }
    //printf("%s", info);
    NSString * info_ns = [NSString stringWithFormat:@"%s", info];
    self.content.text=info_ns;
}


- (IBAction)clickAVCodecButton:(id)sender {

    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;
    }
    //printf("%s", info);
    NSString * info_ns = [NSString stringWithFormat:@"%s", info];
    self.content.text=info_ns;
}

- (IBAction)clickAVFilterButton:(id)sender {
    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);
    }
    //printf("%s", info);
    NSString * info_ns = [NSString stringWithFormat:@"%s", info];
    self.content.text=info_ns;
}

- (IBAction)clickConfigurationButton:(id)sender {
    char info[10000] = { 0 };
    av_register_all();

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

    //printf("%s", info);
    //self.content.text=@"Lei Xiaohua";
    NSString * info_ns = [NSString stringWithFormat:@"%s", info];
    self.content.text=info_ns;
}


@end
```

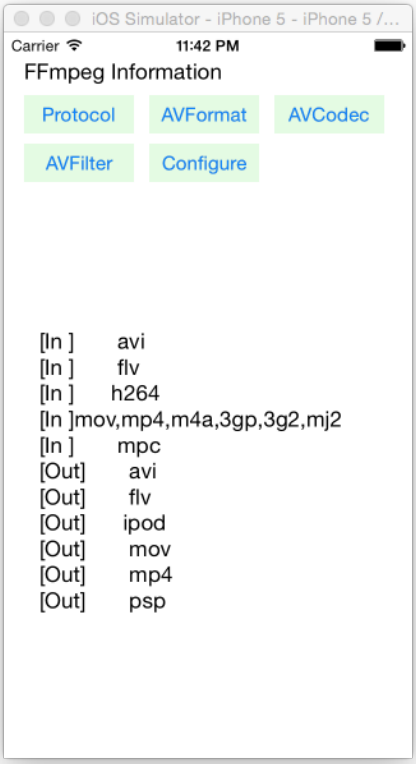
运行结果

App在手机上运行后的结果如下图所示。

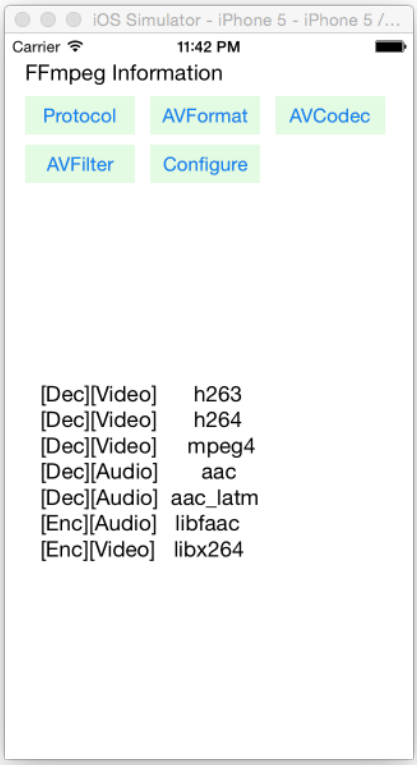
单击不同的按钮，可以得到类库不同方面的信息。单击“Protocol”按钮内容如下所示。



单击“AVFormat”按钮内容如下所示。



单击“AVCodec”按钮内容如下所示。



单击“Configure”按钮即为程序开始运行时候的内容。

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 IOS 编译 arm

个人分类： FFMPEG IOS多媒体

所属专栏： FFmpeg