1. 下载源码

Exiv2官网地址：<http://www.exiv2.org/download.html>, 下载最新的V0.26版本

Expat 官网地址: <https://libexpat.github.io/>, 下载最新的V2.2.2，注意不要下载win版本。

二、编译环境

1. Android NDK

2. Ubuntu 16.04 LTS

三、参考资料

1. <https://stackoverflow.com/questions/30647594/ndk-build-outputs-error-adding-symbols-file-in-wrong-format>，该链接具有非常好的参考价值，但不支持build-xmp

2. standalone toolchain: https://developer.android.com/ndk/guides/standalone\_toolchain.html

四、编译

1.创建独立交叉编工具链

./make-standalone-toolchain.sh --platform=android-21 --install-dir=~/my-android-toolchain --ndk-dir='/home/hzb/Android/Sdk/ndk-bundle/' --toolchain=arm-linux-androideabi-4.9

创建独立工具链的好处是会自动将所有文件都copy到指定目录下，在使用过程中会更好的link相应的头文件和库文件。

Platform最小要使用android-21，尝试使用19，会在编译exiv2的时候报crwimage.cpp中某个方法找不到（不记得是哪个方法了，没有再试）。

2. 编译Expat库

Xmpsdk即使依赖expat做xml解析。

为了能够正常编辑支持xmp的exiv2，一定要先生成对应的expat库。又是一个要编译的东西。头大。没办法，继续搞。Expat的下载地址见文章开头。

Expat是纯c写的，所以相对于exiv2，编译要简单很多，而且expat是android默认内置的动态库，可以到/system/libs或者libs32中查找。当时从手机里面拷贝出来，放到对应路径，也能编译成功。这是下策，但是真有这么干的，见：<https://stackoverflow.com/questions/16960523/how-to-use-internal-expat-xml-parser-in-android>。

编译BuildExpat.sh的脚本如下：

#!/bin/bash

export INSTALL\_DIR="`pwd`/jni/expat"

export NDK=/home/hzb/Android/Sdk/ndk-bundle

export SYSROOT=/home/hzb/android\_toolchain\_21/sysroot

export TOOLCHAIN=/home/hzb/android\_toolchain\_21

export CROSS\_COMPILE="arm-linux-androideabi"

export PATH="$TOOLCHAIN/bin:$PATH"

export ARCH="armeabi-v7a"

export CC="${CROSS\_COMPILE}-gcc --sysroot=$SYSROOT"

export LD="${CROSS\_COMPILE}-ld --sysroot=$SYSROOT"

export CFLAGS="-D\_\_ANDROID\_\_ -D\_\_ARM\_ARCH\_7\_\_ -D\_\_ARM\_ARCH\_7T\_\_ -D\_\_ARM\_ARCH\_7E\_\_ -D\_\_ARM\_ARCH\_7TE\_\_ -mandroid --sysroot $SYSROOT -march=armv7-a -mfloat-abi=softfp -mthumb -O2"

export LDFLAGS='-s'

mkdir -p $INSTALL\_DIR

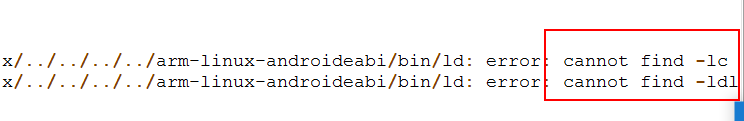
./configure --host=arm-linux-androideabi --prefix=$INSTALL\_DIR

make

make install

exit 0

以上脚本中注意红色字部分，--sysroot表示系统库的所在路径，该路径下通常包含usr/include, usr/lib两个文件夹。如果不加红字部分，在执行./buildexpat.sh时可能会提示c编译器执行失败（某些网上资料说也可能会默认使用独立编译链，所以会自动找到该文件夹，不会报错）。打开config.log，会发现以下错误：



原因就是expat编译过程中依赖libc.so和libdl.so，因为没有加入sysroot，所以找不到对应的库。一开始未加sysroot选项时，我尝试直接使用.configure --host=arm-linux-androideabi --prefix=$INSTALL\_DIR执行，然后执行make是没有问题的，但是编译时使用的是gcc，然后默认的去找编译系统的/usr/下的路径。后来去到ndk-bundle目录下查找这两个类，都在对应的platform下找到了，所以后面加上了sysroot。

再次执行./buildExpat.sh，编译成功，最后的文件可以在jni/expat/下找到，包括bin、include、lib、share四个文件夹，其中include和lib是编译exiv2所需要的文件。Include文件中包含expat.h、expat\_config.h、expat\_external.h三个头文件，lib中主要使用libexpat.a, libexpat.so。由于编译exiv2我们编译的是静态库，所以下面编译exiv2时使用的是libexpat.a。可以将include和lib两个文件夹拷贝到一个其它文件夹下，其中lib下只有libexpat.a，接着将这个文件夹路径在buildExiv2.sh中设置为—with-expat=”${yourExpatPath}”。我这里用到的就是/home/hzb/work/expat

3. 编译Exiv2库

先看看编译脚本buildExiv2.sh如何写:

#!/bin/bash

INSTALL\_DIR="`pwd`/jni/exiv2"

export NDK=/home/hzb/Android/Sdk/ndk-bundle

export SYSROOT=/home/hzb/android\_toolchain\_21/sysroot

export TOOLCHAIN=/home/hzb/android\_toolchain\_21

export PATH=$TOOLCHAIN/bin:$PATH

export ARCH="armeabi-v7a"

export CROSS\_COMPILE="arm-linux-androideabi"

export CC="${CROSS\_COMPILE}-gcc --sysroot=$SYSROOT"

export CXX="${CROSS\_COMPILE}-g++ --sysroot=$SYSROOT"

export CFLAGS="-D\_\_ANDROID\_\_ -D\_\_ARM\_ARCH\_7\_\_ -D\_\_ARM\_ARCH\_7T\_\_ -D\_\_ARM\_ARCH\_7E\_\_ -D\_\_ARM\_ARCH\_7TE\_\_ -mandroid --sysroot $SYSROOT -march=armv7-a -mfloat-abi=softfp -O2"

export CXXFLAGS="$CFLAGS"

export LDFLAGS="-Wl,-rpath-link=$SYSROOT/usr/lib/ -L$SYROOT/usr/lib/ -L${NDK}/sources/cxx-stl/gnu-libstdc++/libs/armeabi-v7a"

export LIBS="-lstdc++ -lsupc++"

mkdir -p $INSTALL\_DIR

./configure --disable-shared --disable-nls --host=arm-linux-androideabi --with-expat=/home/hzb/work/expat --prefix=$INSTALL\_DIR

make

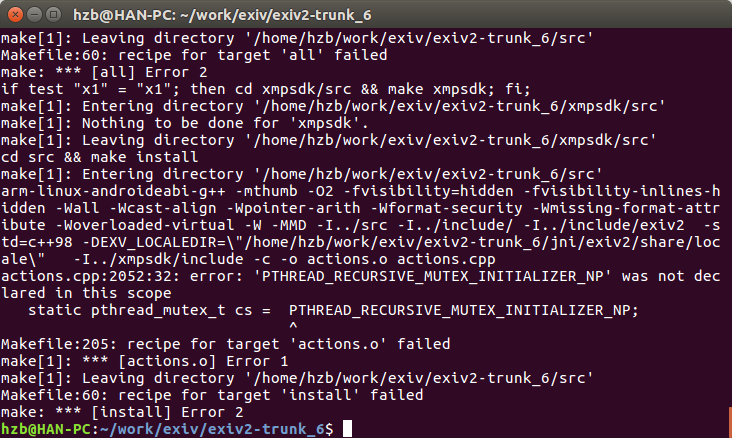
make install

exit 0

这是buildExiv2的脚本。标红字的部分即是expat的设置。

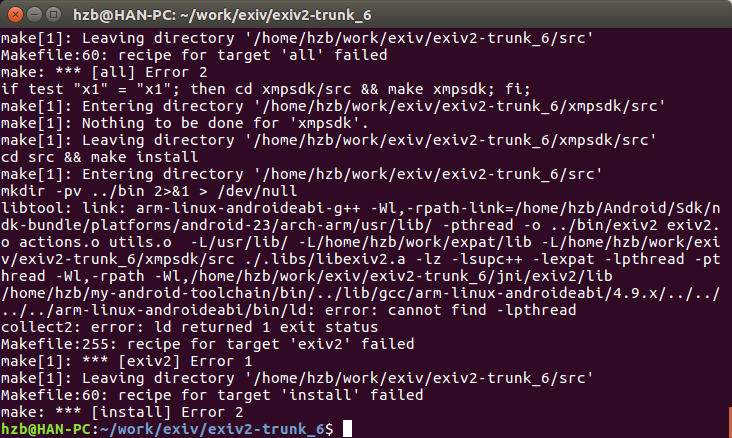
4.执行./buildExiv2

执行上面脚本./buildExiv2后，会出现：



进入actions.cpp查看出错的代码，然后发现该变量应该定义到pthread.h。搜索ndk中出现的pthread.h，没有找到define的地方，只能怀疑在arm中没有定义。没定义就算了，就使用android里面基础的mutex。只需要将static pthread\_mutex\_t cs = PTHREAD\_RECURSIVE\_MUTEX\_INITIALIZER\_NP，改成static pthread\_mutex\_t cs。就可以编译通过了。

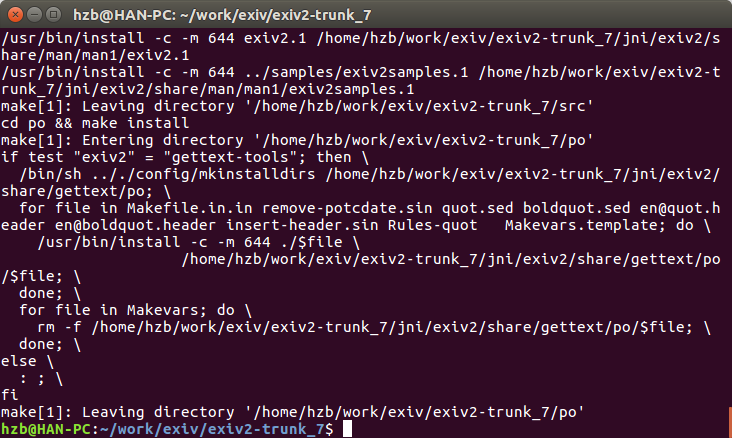
继续编译，结果还是出错：



找不到lpthread，经查，android中不支持lpthread，建议使用pthread。那好吧，就该吧。怎么改？

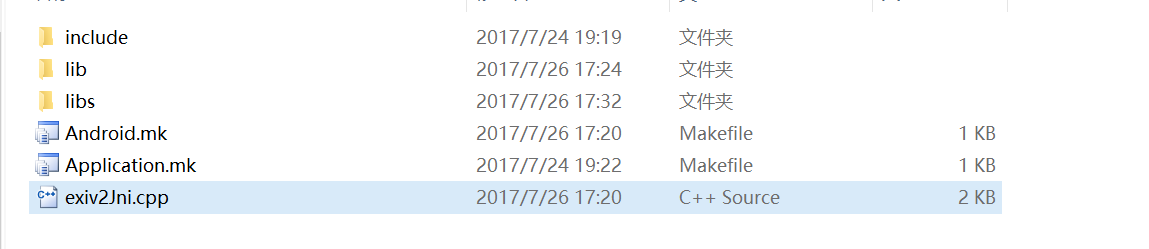
进入到exiv2的根目录，执行命令 find . –name “\*” | xargs grep “lpthread”，在结果中会显示./config/config.ac, .configure,清掉所有的lpthread，改成pthread。重新编译，这次改过了吧。我这边出现了，依然会显示上面cannot find –lpthread的错误。Why？那就继续执行find命令吧，发现在两个新的文件中又出现了-lpthread（产生的原因有可能是前面两个configure相关文件没有清零干净lpthread），这两个文件分别是src/.libs/libexiv2.la和xmpsdk/src/.libs/libxmpsdk.la。la文件是用来关联依赖包的，xmpsdk依赖于expat，exiv2依赖于xmpsdk，继续删掉lpthread吧。出现该原因是在初始清理lpthread时没有清理干净，要么重新做清理，也可以直接修改la文件。

继续编译，恭喜我，这次成功了。



5. 编译libxmp.so。

LibExiv2.a现在有了，离使用又近了一步，但还没有完成，需要生成一个jvm能够使用的so库。开始写jni吧。先看看目录结构：



其中include中存放的是exiv2和expat的头文件，可以从之前编译好的路径中找到include文件夹，复制里面所有的文件到这个目录即可。lib下放置的是libExpat.a和libExiv2.a两个静态库。

接下来，再看看如何写exiv2Jni.cpp:

#include <jni.h>

#include <string>

#include "exiv2/exiv2.hpp"

#include <iostream>

using namespace std;

extern "C"

JNIEXPORT void JNICALL

Java\_com\_xxxx\_yyyyy\_zzzzz\_ExivJni\_setImageGPano(JNIEnv \*env, jobject instance, jstring path\_, jint width, jint height) {

const char \*path = env->GetStringUTFChars(path\_, 0);

Exiv2::XmpData xmpData;

xmpData["Xmp.GPano.UsePanoramaViewer"] = true;

xmpData["Xmp.GPano.ProjectionType"] = "equirectangular";

xmpData["Xmp.GPano.CroppedAreaImageWidthPixels"] = width;

xmpData["Xmp.GPano.CroppedAreaImageHeightPixels"] = height;

xmpData["Xmp.GPano.CroppedAreaLeftPixels"] = 0;

xmpData["Xmp.GPano.CroppedAreaTopPixels"] = 0;

xmpData["Xmp.GPano.FullPanoWidthPixels"] = width;

xmpData["Xmp.GPano.FullPanoHeightPixels"] = height;

Exiv2::Image::AutoPtr image = Exiv2::ImageFactory::open(path);

image->setXmpData(xmpData);

image->writeMetadata();

cout << path << endl;

env->ReleaseStringUTFChars(path\_, path);

}

如何使用xmp，可以参考exiv2中自带的sample代码，这里面需要注意的是红字部分：extern "C"。因为这个库是c++方式编译，不加这句会导致jvm找不到对应的native method。

接下来看看Android.mk.

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

#static library info

include $(CLEAR\_VARS)

LOCAL\_MODULE := exiv2

LOCAL\_SRC\_FILES := lib/libexiv2.a

LOCAL\_EXPORT\_C\_INCLUDES := include/

LOCAL\_EXPORT\_LDLIBS := -Llib -lz -lexpat

include $(PREBUILT\_STATIC\_LIBRARY)

#wrapper info

include $(CLEAR\_VARS)

LOCAL\_C\_INCLUDES += $(LOCAL\_PATH)/include/

LOCAL\_MODULE := xmp

LOCAL\_SRC\_FILES := exiv2Jni.cpp

LOCAL\_STATIC\_LIBRARIES := exiv2

include $(BUILD\_SHARED\_LIBRARY)

可千万别忘了expat，否则会报libexiv2.a中某些方法无法找到的错误。

Application.mk的内容：

NDK\_TOOLCHAIN\_VERSION=4.9

APP\_ABI := armeabi-v7a

APP\_CPPFLAGS += -fexceptions

APP\_STL := gnustl\_static

APP\_CPPFLAGS += -frtti

APP\_PLATFORM := android-21

APP\_BUILD\_SCRIPT := Android.mk

开始编译：ndk-build NDK\_APPLICATION\_PATH=. NDK\_APPLICATION\_MK=./Application.mk

这一步基本上没有什么问题，编译完毕后，会在libs/armebi-v7a中看到libxmp.so。

6. 编写java层

这个简单，就不写了。