NDK 应用及扩展

Android NDK 作用主要用来编译本地库(由 C,C++文件编译后运行于目标 CPU 的库)给 Android java 调用,同时,可以用来打包编译动态库和静态库,但不能将本地库和 java 文件打包成 APK(这是别的范畴,后面有简单实现这功能的方法).

本例子主要介绍如何使用 NDK 编译出动态库(.so)和静态库(.a),并且在编译它们的时候又调用其他的动态库或静态库,并且将其他的库打包成最终的一个库文件.so 给 java 调用. 最后简单说明其他实现编译 so 文件的方法,打包 so 到 apk 里的方法.

NDK 内部变量说明请参考<【eoe 特刊】第七期：NDK>.pdf 文件.

本例子是在 ubuntu9.04 上使用 android-ndk-1.6_r1-linux-x86 和 android-sdk-linux_x86_16.

1. 安装.
   a) Tar –jxvf android-ndk-1.6_r1-linux-x86.tar.bz2 得到 android-ndk-1.6_r1 文件夹
   b) 将 android-ndk-1.6_r1 文件夹放到你要的位置,进入 android-ndk-1.6_r1 文件夹,此时的 pwd 就是 NDK 的根目录,我定义为$NDK_ROOT,
   c) 在$NDK_ROOT 执行命令 ./build/host-setup.sh ,能成功安装 DNK 环境.(如果有问题,请按提示添加系统缺失文件)
2. 编译一个动态库和一个静态库.
   a) 在$NDK_ROOT 目录下进入 apps 文件夹,创建 jni 文件夹和 Application.mk 文件,其内容如下

```
APP_PROJECT_PATH := $(call my-dir)
APP_MODULES      := testapi
```

   testapi 是生产模块的主要名字,
   (此版本系统默认编译$ APP_PROJECT_PATH 下 jni 文件夹下的 c,c++文件)
   b) 进入 jni 文件夹,创建 add.c,add.h,Android.mk 内容分别如下:
      i. add.h:

```
#ifndef ADD_H
#define ADD_H
extern int add(int x, int y);
#endif /* ADD_H */
```

      ii. add.c

```
#include "add.h"
int add(int x, int y){
    return x + y;
}
```

      iii. Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := testapi
LOCAL_SRC_FILES := add.c
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)
```

iv. 回到$NDK_ROOT 目录,命令行输入

make APP=api

编译成功,在$NDK_ROOT/apps/api/libs/armeabi 得到动态库 libtestapi.so, 3147Btye

v. 将 Android.mk 最后一样修改成

include $(BUILD_STATIC_LIBRARY)

在$NDK_ROOT 目录,命令行输入

make APP=api

编译成功,在$NDK_ROOT/out/apps/api/得到静态库 libtestapi.a, 2572Btye

3. 利用 Eclipse 创建 android 的 java 程序通过 JNI 调用动态库 libapp.so,该 so 又调用步骤 2. 生产的库文件 libtestapi.*.

Android APK 据我了解 JNI 只能调用动态库,不能把静态库添加到 APK 里面.

a) 用 eclipse 在任意位置,创建工程 apktest,创建 test.jni 包里 jnitest 的类,jnitest.java 内容如下

```
package test.jni;
public class jnitest    {
        public native int appadd(int    x, int    y);
}
```

b) 然后进入工程的 src 目录制作 JNI 头文件

i. javac test/jni/jnitest.java

javah test.jni.jnitest

产生得到 test_jni_jnitest.h 头文件

c) 在修改 jnitest.java 成如下形式

```
package test.jni;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class jnitest extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            TextView    tv = new TextView(this);
            int             x    = 77;
            int             y    = 88;
            System.loadLibrary("apptest");
            int    z = appadd(x, y);
            tv.setText( x + "+" + y + "=" + z );
            setContentView(tv);
        }
        public native int appadd(int    x, int    y);
    }
```

其中主要参考了 apps\two-libs 下,因为这样简单让我们看起来容易理解.

没有语法错误后 Eclipse 在其工程的 bin 文件夹下自动产生 apktest.apk 13.2k

此时,这模拟器运行 apk 会报错,因为动态链接库 libapptest.so 没有在 apk 里,也没在/system/lib

d) 回到$NDK_ROOT 目录下,进入 apps 目录执行一下命令

ln -s /home/clay/workspace/apktest/ app

其中/home/clay/workspace/apktest/是 eclipse 创建工程的位置,

e) 因为 ln –s 的关系,app 目录几乎等同/home/clay/workspace/apktest/,我用 app 目录代表 /home/clay/workspace/apktest/目录.方便理解

f) 在 app 目录下创建 jni 文件夹和 Application.mk 文件,其内容如下

```
APP_PROJECT_PATH := $(call my-dir)
APP_MODULES      := apptest
```

其中 apptest 要和 java 里 System.loadLibrary("apptest");一致

g) 在上一步的 jni 文件夹里创建 appcall.c 和 Android.mk 文件,内容如下

i. Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE       := apptest
LOCAL_SRC_FILES := callapp.c
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)
```

ii. appcall.c 根据 test_jni_jnitest.h

```
#include <jni.h>
JNIEXPORT jint JNICALL Java_test_jni_jnitest_appadd
  (JNIEnv *env, jobject obj, jint x, jint y){
    return x + y;
}
```

为检查 JNI,我们先直接在 libapptest.so 实现,待验证后再修改 call 别的库文件.

h) 回到$NDK_ROOT 目录,执行 make APP=app

得到 $NDK_ROOT /apps/app/libs/armeabi/libapptest.so 2146Byte

如果这时候,用 adb push /apps/app/libs/armeabi/libapptest.so /system/lib,然后从模拟菜单里运行刚才 apk 程序(名字也叫 apk,建工程时候定下来的),此时可以运行,而且显示”77+88=165”

i) 此时回到 Eclipse 从菜单选择 project->clean,然后 eclipse 会自动(如果没设 auto 需手动)重新编译该工程,这时候你会发现工程里 libs/armeabi/libapptest.so 出现,而且,bin 目录下的 apk 变成了 14.7k,用 zip 或 rar(windows 下)你会发现你的 apk 文件里比之前多了 lib 文件夹,其子文件夹 armeabi 里有 libapptest.so 文件.用 adb shell 进入到虚拟机的/system/lib 目录,删除之前 libapptest.so,可以重启模拟器.

证明系统没有 libapptest.so.后,运行后边编译的 apk 文件,这时候 apk 能正确运行,没有报错,而/system/lib 目录下仍是没有 libapptest.so,这说明:我们这时候的 apk 把 libapptest.so 打包进去,而且系统能解出 libapptest.so 并加载.

但是,这只是 eclipse 通过 ADT 完成把 so 打包到 apk 里的.如果用源代码开发包 mm 命令方式产生的 apk(不是 NDK)似乎不行(之后再讨论,或许 makefile 没定义好). NDK 绝对没有做这打包的动作.

4. 利用 NDK 打印调试

a) 在 3.基础上,修改 android.mk 连接 liblog.so 成为以下

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

```
include $(CLEAR_VARS)
LOCAL_MODULE    := apptest
LOCAL_SRC_FILES := callapp.c
LOCAL_LDLIBS := -llog
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)
```

b) 然后修改 callapp.c,添加打印

```
#include <jni.h>
#include <android/log.h>
#define LOGD(...) __android_log_print(ANDROID_LOG_INFO, "Logtest", __VA_ARGS__);
JNIEXPORT jint JNICALL Java_test_jni_jnitest_appadd
  (JNIEnv *env, jobject obj, jint x, jint y){
    LOGD("TEST %d+%d=%d",x,y,x+y);
__android_log_print(ANDROID_LOG_INFO, "Logtest","================");
    return x + y;
}
```

c) 在$NDK_ROOT 下输入 make APP=app,重新编译动态库,eclipse clean rebuilt,运行,在 eclipse 的 logcat 可以看到以下:

```
I/Logtest (  375): TEST 77+88=165
I/Logtest (  375): ================
```

//注意,如果不用 NDK,只有 SDK 开发,可以直接#include <utils/Log.h>后用 LOGD(),类似 c 的 printf,,另外,此时即使你 include 了<stdio.h>,c 的 printf 仍是没有打印输出的

5. 为动态库 libapptest.so 链接别的静态库 libXX.a

a) 修改前,libapptest.so 为 2829Byte,libtestapi.a 为 2572Byte.

b) 把 api 工程的 add.h 复制到$NDK_ROOT/apps/app/jni/下,修改 callapp.c 如下:

```
#include <jni.h>
#include <android/log.h>
#include "add.h"
#define LOGD(...) __android_log_print(ANDROID_LOG_INFO, "Logtest", __VA_ARGS__);
JNIEXPORT jint JNICALL Java_test_jni_jnitest_appadd
  (JNIEnv *env, jobject obj, jint x, jint y){
    LOGD("TEST %d+%d=%d",x,y,x+y);
__android_log_print(ANDROID_LOG_INFO, "Logtest","================");
    return add(x,y);
}
```

c) 修改$NDK_ROOT/apps/app/jni/Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := apptest
LOCAL_SRC_FILES := callapp.c
LOCAL_STATIC_LIBRARIES := libtestapi
LOCAL_LDLIBS := -llog
```

LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

d) 把$NDK_ROOT/out/api/libtestapi.a 复制到$NDK_ROOT/out/apps/app/,然后 make APP=app,编译成功.$NDK_ROOT/apps/app/libs/armeabi/libapptest.so 3012Byte
如果把 a 文件放到 jni 文件里编译,会提示找不到 No rule to make target `out/apps/app//libtestapi.a, 我认为, LOCAL_STATIC_LIBRARIES, LOCAL_SHARED_LIBRARIES 默认也是要自己编译出来

e) Eclipse 把工程 clean rebuilt,run, 显示和打印都是正确的. 表是 libtestapi.a 的确是编进到 libapptest.so 文件里了

6. 为动态库 libapptest.so 连接别的动态库 libXX.so

a) 第 4 修改后,libapptest.so 为 2829Byte,libtestapi.so 为 3147Byte.(之前的记录)

b) 维持第 5 修改的 c 文件,

c) 修改$NDK_ROOT/apps/app/jni/Android.mk 为

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := apptest
LOCAL_SRC_FILES := callapp.c
LOCAL_SHARED_LIBRARIES := libtestapi
LOCAL_LDLIBS := -llog
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

d) 同理,在$NDK_ROOT, cp out/apps/api/libtestapi.so out/apps/app/,make APP=app –B 可以通过, 但是此时,eclipse clean 后 rebuilt,run 出现错误.因为动态库在编译的时候只做连接,并不包含到最终目标,这和静态库不一样.同时,你也会发现 $NDK_ROOT/apps/app/libs/armeabi/下只有 libapptest.so,没有 libtestapi.so.请注意,这时候编译得到到 libapptest.so 是 2913Byte,和之前的不调用 add()函数几乎差不多.

e) 这时候,我们把 libtestapi.so 复制到$NDK_ROOT/apps/app/libs/armeabi/,eclipse clean&rebuilt,run,结果仍是出错.虽然 bin 目录下的 apktest.apk 的包 libs/armeabi/里面有 libtestapi.so 和 libapptest.so

f) 如果这时候,把 libtestapi.so 复制到 $NDK_ROOT/apps/app/asset,eclipse clean&rebuilt,run,结果仍是出错.虽然 bin 目录下的 apktest.apk 包的

g) 即使我在 jnitest.java 里,添加了 System.loadLibrary("apptest"); System.loadLibrary("testapi");重复 e 和 f,仍旧失败

h) 如果这时候,我们用 adb remount(让模拟器系统可以改写),再 adb push out/apps/app/libtestapi.so /system/lib,之后,运行 eclipse 修改的 apk,发现可以正常正确的运行.

由此,我认为 NDK 能编译,但是,不能把 so 打包,同时也不能把编译中间的 so 文件(libtestapi.so)打包,但是 NKD 却能以某种方式(我不知道)告诉 APK 打包的程序,所以 eclipse 能把 ibs/armeabi/下 libapptest.so 和 libtestapi.so 打包,但是系统跑起来却不能自动加载(能加载 libapptest.so,不能加载 libtestapi.so)

7. 为动态库 libapptest.so 连接别的动态库 libXX.so,方法二.

a) 第 4 修改后,libapptest.so 为 2829Byte,libtestapi.so 为 3147Byte.(之前的记录)

b) 维持第 5 修改的 c 文件,

c) 用 adb shell 进入到/system/lib 里删除第 6 加进去的 libtestapi.so

d) 修改$NDK_ROOT/apps/app/jni/Android.mk 为

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE       := apptest
LOCAL_SRC_FILES := callapp.c
LOCAL_LDLIBS :=    -ldl -llog
LOCAL_PRELINK_MODULE := false
#LOCAL_ALLOW_UNDEFINED_SYMBOLS = true
include $(BUILD_SHARED_LIBRARY)
```

e) 这时候 make APP=app, 提示找 add 的原型，这时候 ，我们把 LOCAL_ALLOW_UNDEFINED_SYMBOLS = true 打开,这时候,能编译通过得到 2.82k 的 libapptest.so,但是却出现正确的,,其实是刚好有一个有个 lib 的函数是 add, 如果我们把我们 add.c 和 add.h 里的 add 函数改成,add22，重新编译 api 和 app,运行就出错了.但是,这时候需要我们手工把 libtestapi.so 放到/system/lib 里,仍旧不能运行, 所以怀疑,该 flag 是从全部的 so 里搜索(改成 add22 后,重复之前所以步骤都是可以得到按之前的说明的结果)

f) 如果这时候我们只打开把 Android.mk 修改成如下

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE       := apptest
LOCAL_SRC_FILES := callapp.c
LOCAL_LDLIBS := -ltestapi -ldl -llog
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)
```

LOCAL_LDLIBS := -ltestapi -ldl –llog//有时候可以分成几行来写,但分开写,有时候编译不通过.(I don't know why)

这是仿照我们之前添加 log 打印的方式一样,把我们的动态库当成 NDK 的库,所以,我把 libtestapi.so 放到 NDK, cp apps/api/libs/armeabi/libtestapi.so build/platforms/android-4/arch-arm/usr/lib/,

在这之前,我们 make APP=app -B 提示找不到 libtestapi.so,把 libtestapi.so 放到 NDK 的 lib 后,编译可以通过.如果没通过,需要把 libtestapi.so 的 mode 变成 777。

这时候,eclipse 重新编译工程运行,提示错误,但是,我们用 adb 把 libtestapi.so push 弄到/system/lib 后,重新运行,就 ok 了

关于 NDK 的到此差不多告一段落.

由于 android 的 makefile 的一大堆变量 刚开始就把人吓坏了.至今也没理解多少,这一切都是不断在网上查,自己摸索,不断看他的列子和其他人的建议,尝试的结果,其中反反复复好多次.

==========

其实通过 JNI 的方法来实现还有两种方式,这些都是我同事 rock 帮忙弄的,借花献佛,简单介绍一下.

1. linux 命令方式,
   a) 用 gcc xxx.c -shared -o libXXX.so,生成 动态链接库.
   b) gcc xxx2.c -L . -l XXX -o APP,把 xxx2c 文件链接动态库 libXXX.so,形成 APP 可执行

文件//注意,这个文件时在 linux 下运行的,是 host

c) 因为我的目标板(target board)是 arm 核心的,所以使用/usr/local/arm/4.3.2/bin/ 交叉编译器,模仿以上两步(自己写 makefile),注意头文件时 arm 的头文件就可以了,这样 so 和 APP 可以在 arm 板上 run 了,可以使用 gdb 等工具调试.

2. Android 原代码 SDK 开发.需要有完整的 code,并且会找到系统的编译器和头文件,把编译出来的中间 so 文件放到特定位置.详细参考<Android 平台怎样使用第三方动态库>
   So 文件放的位置主要根据编译时候提示在哪里找不到,就放到哪里(因为 android 默认要完全有原代码开发,所以它默认我们的 apk 用的 so 需要有原文件,vendor 下的例外)
3. 另外,很幸运,我们目标是 ARM 核心,有以上两步和 NDK 得到 so 文件几乎可以混用.SDK 开发得到的 so 文件和 NDK 得到的 so 文件,分别 push 我目标板子和模拟器,都可以跑起来.而且三种方法得到的 so,apk 都可以在我的 ARM 目标板可以跑起来.并且,NDK 得到的 libapptest.so 可以调用已经 push 到/system/lib 里 arm 编译器编译出来的 libtestapi.so 文件.但是,arm 编译器编出来的 libtestapi.so 不能放到 NDK 或 SDK 里混合来编译出新的 libapptest.so, 提示" libthunder_api.so has EABI version 0, but target out/target/product/generic/obj/SHARED_LIBRARIES/libthunder_app_intermediates/LINKED/libthunder_app.so has EABI version 4",也就是说,NDK 和 SDK 里会检测版本.幸运的是我用的 NDK 和 SDK 都是 1.6 的,而 arm 工具链编出来没有版本(EABI version 0)
   另外,才傻傻发现,SDK 里已经有 NDK 了…

另外,如果我们用 SDK 开发,我们使用 mm 得到的 APK 里是没有打包 so 文件的,这时候测试,只能手工把 so 文件 push 到系统里.而且,你会发现,如打包了 so 文件的 apk 文件用 adb install 以后,在/system/lib 目录,是找不到这 so 文件的.我曾经在/data/app 或/data/system 目录里见过,但是现在仍是没发现,不知道为什么?如果有谁知道请分享一下.

虽然 mm 命令方式生成的 apk 里没 so 文件,但是整个 android 系统 built 通过以后,建成 img 文件的时候,so 文件是会被放到/system/lib 目录下的,例如 lib3G 等等.

我在网上查到,ant 的开发方式可以把 so 打包到 APK 里,但是 SDK 开发用 mm 目前我没有试出来,曾经看过别人写的 android.mk 里把 so 当成 source 文件去编译 APK,但是我用 NDK 的时候却不成功,但是根据 linux c 的感觉,把 so 当成 source(或中间文件) 一起编译打包理论是可以的..

==========

这些事熬了半个通宵写了一部分边印证得出来的,平时上班的时间也尝试了很久很久,所以如此艰难弄了这说不定 bug 很多的半成品,虚荣一下,留个名.

CLAY_PAN

2010July30