Android jni本地编程 快速入门

## 一、JNI（Java Native Interface）

### 1、什么是JNI：

JNI(Java Native Interface):java本地开发接口

JNI是一个协议，这个协议用来沟通java代码和外部的本地代码(c/c++)

外部的c/c++代码也可以调用java代码

### 2、为什么使用JNI：

效率上C/C++是本地语言，比java更高效

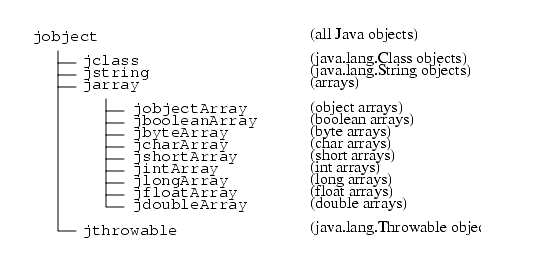
代码移植，如果之前用C语言开发过模块，可以复用已经存在的c代码

java反编译比C语言容易，一般加密算法都是用C语言编写，不容易被反编译

### 3、Java基本数据类型与C语言基本数据类型的对应



### 4、引用类型对应



### 5、堆内存和栈内存的概念

**栈内存**：系统自动分配和释放，保存全局、静态、局部变量，在站上分配内存叫静态分配，大小一般是固定的

**堆内存**：程序员手动分配(malloc/new)和释放(free/java不用手动释放，由GC回收)，在堆上分配内存叫动态分配，一般硬件内存有多大堆内存就有多大

## 二、交叉编译

### 1、交叉编译的概念

交叉编译即在一个平台，编译出另一个平台能够执行的二进制代码

主流平台有：Windows、Mac os、Linux

主流处理器：x86、arm、mips

### 2、交叉编译的原理

即在一个平台上，模拟其他平台的特性

编译的流程：源代码-->编译-->链接-->可执行程序

### 3、交叉编译的工具链

多个工具的集合，一个工具使用完后接着调用下一个工具

### 4、常见的交叉编译工具

NDK(Native Development Kit)：开发JNI必备工具，就是模拟其他平台特性类编译代码的工具

CDT(C/C++ Development Tools)：是Eclipse开发C语言的一个插件，高亮显示C语言的语法

Cygwin：一个Windows平台的Unix模拟器（可以参考之前博客：[Cygwin简介及使用](http://www.cnblogs.com/rocomp/p/4805553.html)）

### 5、NDK的目录结构（可以在Google官网下载NDK开发工具，需要FQ）

docs：帮助文档

build/tools：linux的批处理文件

platforms：编译c代码需要使用的头文件和类库

prebuilt：预编译使用的二进制可执行文件

sample：jni的使用例子

source：ndk的源码

toolchains：工具链

ndk-build.cmd：编译打包c代码的一个指令，需要配置系统环境变量

## 三、JNI的第一个例子

好了，准备知识已经完毕，下面开始我们的一个JNI例子。

1、新建一个Android项目，在根目录下创建jni文件夹，用于存放C源码。

2、在java代码中，创建一个本地方法getStringFromC本地方法没有方法体。

private native String getStringFromC();

3、在jni中创建一个C文件，定义一个函数实现本地方法，函数名必须用使用本地方法的全类名，点改为下划线。

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <jni.h>

4 //方法名必须为本地方法的全类名点改为下划线，穿入的两个参数必须这样写，

5 //第一个参数为Java虚拟机的内存地址的二级指针，用于本地方法与java虚拟机在内存中交互

6 //第二个参数为一个java对象，即是哪个对象调用了这个c方法

7 jstring Java\_com\_mwp\_jnihelloworld\_MainActivity\_getStringFromC (JNIEnv\* env,

8 jobject obj){

9 //定义一个C语言字符串

10 char\* cstr = "hello form c";

11 //返回值是java字符串，所以要将C语言的字符串转换成java的字符串

12 //在jni.h中定义了字符串转换函数的函数指针

13 //jstring (\*NewStringUTF)(JNIEnv\*, const char\*);

14 //第一种方法：很少用

15 jstring jstr1 = (\*(\*env)).NewStringUTF(env, cstr);

16 //第二种方法，推荐

17 jstring jstr2 = (\*env) -> NewStringUTF(env, cstr);

18 return jstr2;

19 }

4、在jni中创建Android.mk文件，用于配置本地方法

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

include $(CLEAR\_VARS)

#编译生成的文件的类库叫什么名字

LOCAL\_MODULE := hello

#要编译的c文件

LOCAL\_SRC\_FILES := Hello.c

include $(BUILD\_SHARED\_LIBRARY)

5、在jni目录下执行ndk-build.cmd指令，编译c文件

6、在java代码中加载编译后生成的so类库，调用本地方法，将项目部署到虚拟机上之后就会发现toast弹出的C代码定义的字符串，第一个例子执行成功了。

static{

//加载打包完毕的so类库

System.loadLibrary("hello");

}

7、jni打包的C语言类库默认仅支持arm架构，需要在jni目录下创建Android.mk文件添加如下代码可以支持x86架构

APP\_ABI := armeabi armeabi-v7a x86

## 四、JNI常见错误

1、find Library returned null:

CPU平台不匹配，或者在加载类库时，类库名字写错了

2、本地方法找不到：

忘记加载类库了，或者C代码中方法名写错了

## 五、javah工具与javap工具

1、javah：生成本地方法头文件

需要在C/C++模块下才能生效

在JDK1.7中，在src目录下执行javah全类名

在JDK1.6中，在bin/classes目录下执行

2、javap：打印方法签名

在C语言中调用java的方法需要用到反射，C语言的反射需要一个方法签名，使用javap能够生成方法签名，很熟练的话也可以自己写方法签名

在bin/classes目录下执行javap -s全类名

## 六、使用本地方法加密字符串的一个小例子

C语言字符串与Java中的字符串类型不同，所以需要进行字符串类型转换。

**一个重要的思想**：C语言计算字符串的长度不方便，但是java很方便，只需要调用一个length()方法就可以，所以像这种需求，那个语言有优势就用哪个语言算，算完当做参数传递给另一种语言就ok。

混合语言编程这应该是一种非常有用的思想。

Java非常容易被反编译，所以加密都是用c语言写的

#include <jni.h>

#include <string.h>

//将java字符串转换为c语言字符串（工具方法）

char\* Jstring2CStr(JNIEnv\* env, jstring jstr)

{

char\* rtn = NULL;

jclass clsstring = (\*env)->FindClass(env,"java/lang/String");

jstring strencode = (\*env)->NewStringUTF(env,"GB2312");

jmethodID mid = (\*env)->GetMethodID(env,clsstring, "getBytes", "(Ljava/lang/String;)[B");

jbyteArray barr= (jbyteArray)(\*env)->CallObjectMethod(env,jstr,mid,strencode); // String .getByte("GB2312");

jsize alen = (\*env)->GetArrayLength(env,barr);

jbyte\* ba = (\*env)->GetByteArrayElements(env,barr,JNI\_FALSE);

if(alen > 0)

{

rtn = (char\*)malloc(alen+1); //"\0"

memcpy(rtn,ba,alen);

rtn[alen]=0;

}

(\*env)->ReleaseByteArrayElements(env,barr,ba,0); //

return rtn;

}

JNIEXPORT jstring JNICALL Java\_com\_mwp\_encodeanddecode\_MainActivity\_encode

(JNIEnv \* env, jobject obj, jstring text, jint length){

char\* cstr = Jstring2CStr(env, text);

int i;

for(i = 0;i<length;i++){

\*(cstr+i) += 1; //加密算法，将字符串每个字符加1

}

return (\*env)->NewStringUTF(env,cstr);

}

JNIEXPORT jstring JNICALL Java\_com\_mwp\_encodeanddecode\_MainActivity\_decode

(JNIEnv \* env, jobject obj, jstring text, jint length){

char\* cstr = Jstring2CStr(env, text);

int i;

for(i = 0;i<length;i++){

\*(cstr+i) -= 1;

}

return (\*env)->NewStringUTF(env, cstr);

}

## 七、JNI操作一个数组（引用传递）

传递数组其实是传递一个堆内存的数组首地址的引用过去，所以实际操作的是同一块内存，当调用完方法，不需要返回值，实际上参数内容已经改变，Android中很多操作硬件的方法都是这种C语言的传引用的思路

1public class MainActivity extends Activity {

2

3 static{

4 System.loadLibrary("encode");

5 }

6 int[] array = {1,2,3,4,5};

7 @Override

8 protected void onCreate(Bundle savedInstanceState) {

9 super.onCreate(savedInstanceState);

10 setContentView(R.layout.activity\_main);

11 }

12

13 public void click(View v){

14 encodeArray(array);

15 //不需要返回值，实际操作的是同一块内存，内容已经发生了改变

16 for (int i : array) {

17 System.out.println(i);

18 }

19 }

20

21 //传递数组其实是传递一个堆内存的数组首地址的引用过去，所以实际操作的是

22 //同一块内存，当调用完方法，不需要返回值,实际上参数内容已经改变，

23 //Android中很多操作硬件的方法都是这种C语言的传引用的思路，要非常熟练

24 private native void encodeArray(int[] arr);

25}

1 #include <jni.h>

2 /\*

3 \* Class: com\_mwp\_jniarray\_MainActivity

4 \* Method: encodeArray

5 \* Signature: ([I)V

6 \*/

7JNIEXPORT void JNICALL Java\_com\_mwp\_jniarray\_MainActivity\_encodeArray

8 (JNIEnv \* env, jobject obj, jintArray arr){

9 //拿到整型数组的长度以及第0个元素的地址

10 //jsize (\*GetArrayLength)(JNIEnv\*, jarray);

11 int length = (\*env)->GetArrayLength(env, arr);

12 // jint\* (\*GetIntArrayElements)(JNIEnv\*, jintArray, jboolean\*);

13 int\* arrp = (\*env)->GetIntArrayElements(env, arr, 0);

14 int i;

15 for(i = 0;i<length;i++){

16 \*(arrp + i) += 10; //将数组中的每个元素加10

17 }

18}

## 八、偷用美图秀秀的C语言本地类库加深JNI的理解

项目中不需要有c代码，只需要有一个编译过后的类库供Java调用就可以了。

将美图秀秀的apk文件解压缩，将lib目录下C类库导入自己的项目，反编译美图秀秀的apk文件，将其本地方法类JNI.java复制到自己的项目根据本地方法名和参数猜函数的作用及如何使用，下例调用了美图的一个LOMO美化效果：

1public class MainActivity extends Activity {

2

3 static{

4 //加载美图秀秀的类库

5 System.loadLibrary("mtimage-jni");

6 }

7 private ImageView iv;

8 private Bitmap bitmap;

9 @Override

10 protected void onCreate(Bundle savedInstanceState) {

11 super.onCreate(savedInstanceState);

12 setContentView(R.layout.activity\_main);

13

14 iv = (ImageView) findViewById(R.id.iv);

15

16 bitmap = BitmapFactory.decodeFile("sdcard/aneiyi.jpg");

17 iv.setImageBitmap(bitmap);

18 }

19

20 public void click(View v){

21

22 int width = bitmap.getWidth();

23 int height = bitmap.getHeight();

24

25 //用于保存所有像素信息的数组

26 int[] pixels = new int[width\*height];

27 //获取图片的像素颜色信息，保存至pixels

28 bitmap.getPixels(pixels, 0, width, 0, 0, width, height);

29

30 JNI jni = new JNI();

31 //调用美图秀秀本地库中的美图方法，靠猜

32 //arg0:保存了所有像素颜色信息的数组

33 //arg1:图片的宽

34 //arg2:图片的高

35 //此方法是通过改变pixels的像素颜色值来实现美化效果，传递一个数组参数是不需要返回值的

36 jni.StyleLomoB(pixels, width, height);

37

38 Bitmap bmNew = Bitmap.createBitmap(pixels, width, height, bitmap.getConfig());

39 iv.setImageBitmap(bmNew);

40 }

41}

## 九、在C语言中调用java方法（反射）

1、有时需要在C语言中调用java的方法，如刷新UI显示加载资源进度：在本地方法C语言代码中打印Android的Logcat日志输出，Google已经帮我们封装好了方法，只需要调用一下就可以。如果要输出中文的话，必须将C语言的文件编码改成utf-8,否则乱码。

在C语言中调用java的方法需要用到反射，C语言的反射需要一个方法签名，使用javap能够生成方法签名，很熟练的话也可以自己写方法签名。

在bin/classes目录下执行javap –s全类名

1public class MainActivity extends Activity {

2 static{

3 System.loadLibrary("hello");

4 }

5

6 @Override

7 protected void onCreate(Bundle savedInstanceState) {

8 super.onCreate(savedInstanceState);

9 setContentView(R.layout.activity\_main);

10 }

11

12 public void click(View v){

13 cLog();

14 }

15

16 public native void cLog();

17

18 public void show(String message){

19 Builder builder = new Builder(this);

20 builder.setTitle("标题");

21 builder.setMessage(message);

22 builder.show();

23 }

24

25 }

#include <jni.h>

#include <android/log.h>

#define LOG\_TAG "System.out"

#define LOGD(...) \_\_android\_log\_print(ANDROID\_LOG\_DEBUG, LOG\_TAG, \_\_VA\_ARGS\_\_)

#define LOGI(...) \_\_android\_log\_print(ANDROID\_LOG\_INFO, LOG\_TAG, \_\_VA\_ARGS\_\_)

JNIEXPORT void JNICALL Java\_com\_mwp\_ccalljava2\_MainActivity\_cLog (JNIEnv \* env, jobject obj){

//打印log输出

LOGD("我是C语言打印的debug日志");

LOGI("我是C语言打印的info日志");

//通过反射来调用java的方法，需要知道方法签名，使用javap得到方法签名

//在bin/classes目录下执行javap -s 全类名

//得到类的字节码对象

//jclass (\*FindClass)(JNIEnv\*, const char\*);

jclass clazz = (\*env)->FindClass(env, "com/mwp/ccalljava2/MainActivity");

//jmethodID (\*GetMethodID)(JNIEnv\*, jclass, const char\*, const char\*);

jmethodID methodID = (\*env)->GetMethodID(env, clazz, "show", "(Ljava/lang/String;)V");

//void (\*CallVoidMethod)(JNIEnv\*, jobject, jmethodID, ...);

(\*env)->CallVoidMethod(env,obj,methodID, (\*env)->NewStringUTF(env, "这是弹窗的内容"));

}

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

include $(CLEAR\_VARS)

LOCAL\_LDLIBS += -llog

LOCAL\_MODULE := hello

LOCAL\_SRC\_FILES := log.c

include $(BUILD\_SHARED\_LIBRARY)

## 十、模拟监测压力传感器

传感器的原理是使用敏感电阻如（光敏电阻，热敏电阻）等监测电流电压的变化，Android程序只需要处理传感器传递的数据，并将其显示在界面上就可以。

下面模拟一个压力传感器来练习JNI编程：

1public class MainActivity extends Activity {

2 static{

3 System.loadLibrary("monitor");

4 }

5 private MyProgressBar mpb;

6 @Override

7 protected void onCreate(Bundle savedInstanceState) {

8 super.onCreate(savedInstanceState);

9 setContentView(R.layout.activity\_main);

10

11 mpb = (MyProgressBar) findViewById(R.id.mpb);

12 mpb.setMax(100);

13 }

14

15 public void start(View v){

16 new Thread(){

17 public void run() {

18 startMonitor();

19 };

20 }.start();

21 }

22

23 public void stop(View v){

24 stopMonitor();

25 }

26

27 public native void startMonitor();

28 public native void stopMonitor();

29

30 //供本地方法调用刷新UI

31 public void show(int pressure){

32 mpb.setPressure(pressure);

33 }

34 }

#include <jni.h>

#include <stdio.h>

#include <stdlib.h>

//模拟压力传感其传递数据

int getPressure(){

return rand()%101;

}

//用于控制循环的开关

int monitor;

JNIEXPORT void JNICALL Java\_com\_mwp\_monitor\_MainActivity\_startMonitor

(JNIEnv \* env, jobject obj){

monitor = 1;

int pressure;

jclass clazz;

jmethodID methodid;

while(monitor){

//本地方法获取传感器数据

pressure= getPressure();

//使用反射调用java方法刷新界面显示

//jclass (\*FindClass)(JNIEnv\*, const char\*);

clazz= (\*env)->FindClass(env, "com/mwp/monitor/MainActivity");

//jmethodID (\*GetMethodID)(JNIEnv\*, jclass, const char\*, const char\*);

methodid= (\*env)->GetMethodID(env, clazz, "show","(I)V");

// void (\*CallVoidMethod)(JNIEnv\*, jobject, jmethodID, ...);

(\*env)->CallVoidMethod(env, obj, methodid, pressure);

sleep(1);

}

}

JNIEXPORT void JNICALL Java\_com\_mwp\_monitor\_MainActivity\_stopMonitor

(JNIEnv \* env, jobject obj){

//结束循环

monitor = 0;

}

## 十一、使用C++代码实现本地方法

1、把c文件后缀名换成cpp。

2、Android.mk文件中的hello.c也要换成hello.cpp。

3、c++使用的环境变量结构体中，访问了c使用的结构体的函数指针，函数名全部都是一样的，只是参数去掉了结构体指针。

4、访问函数指针时，把env前面的\*号去掉，因为此时env已经是一级指针。

5、clean，清除之前编译的残留文件。

6、把声明函数的h文件放入jni文件夹中，include该h文件。

#include <jni.h>

#include "com\_mwp\_cplusplus\_MainActivity.h"

JNIEXPORT jstring JNICALL Java\_com\_mwp\_cplusplus\_MainActivity\_helloC

(JNIEnv \* env, jobject obj){

char\* cstr = "hello from c";

//return (\*env)->NewStringUTF(env, cstr);

return env->NewStringUTF(cstr);

}