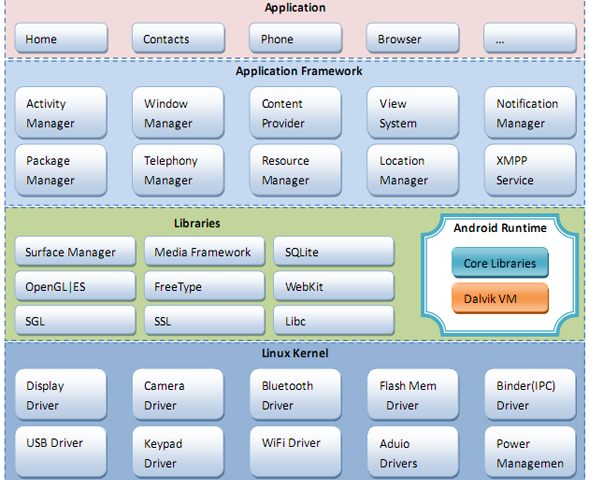
首先看一下Android平台的框架图：（网上盗用）



　　可以看到Android上层的Application和ApplicationFramework都是使用Java编写，

底层包括系统和使用众多的LIiraries都是C/C++编写的。

　　所以上层Java要调用底层的C/C++函数库必须通过Java的JNI来实现。

下面将学习Android是如何通过Jni来实现Java对C/C++函数的调用。以HelloWorld程序为例：

**第一步：**

使用Java编写HelloWorld 的Android应用程序：

[复制代码](javascript:void(0);)

package com.lucyfyr;

import android.app.Activity;

import android.os.Bundle;

import android.util.Log;

public class HelloWorld extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

　　super.onCreate(savedInstanceState);

　　setContentView(R.layout.main);

　　Log.v("dufresne", printJNI("I am HelloWorld Activity"));

}

　　static

　　{

　　　　//加载库文件

　　　　System.loadLibrary("HelloWorldJni");

　　}

　　//声明原生函数 参数为String类型 返回类型为String

　　private native String printJNI(String inputStr);

}

[复制代码](javascript:void(0);)

　　这一步我们可以使用eclipse来生成一个App；

**因为eclipse会自动为我们编译此Java文件，后面要是用到。**

**第二步：**

　　生成共享库的头文件：

　　进入到eclipse生成的Android Project中 ：/HelloWorld/bin/classes/com/lucyfyr/ 下：

　　可以看到里面后很多后缀为.class的文件，就是eclipse为我们自动编译好了的java文件，其中就有：

　　　　HelloWorld.class文件。

　　退回到classes一级目录：/HelloWorld/bin/classes/

执行如下命令：

**javah com.lucyfyr.HelloWorld**

　　生成文件:**com\_lucyfyr\_HelloWorld.h**

[复制代码](javascript:void(0);)

/\* DO NOT EDIT THIS FILE - it is machine generated \*/

#include <jni.h>

/\* Header for class com\_lucyfyr\_HelloWorld \*/

#ifndef \_Included\_com\_lucyfyr\_HelloWorld

#define \_Included\_com\_lucyfyr\_HelloWorld

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*

\* Class: com\_lucyfyr\_HelloWorld

\* Method: printJNI

\* Signature: (Ljava/lang/String;)Ljava/lang/String;

\*/

JNIEXPORT jstring JNICALL Java\_com\_lucyfyr\_HelloWorld\_printJNI

(JNIEnv \*, jobject, jstring);

#ifdef \_\_cplusplus

}

#endif

#endif

[复制代码](javascript:void(0);)

　　可以看到自动生成对应的函数：**Java\_com\_lucyfyr\_HelloWorld\_printJNI**

**Java\_ + 包名（com.lucyfyr） + 类名(HelloWorld) + 接口名(printJNI)：必须要按此JNI规范来操作；**

　　java虚拟机就可以在com.simon.HelloWorld类调用printJNI接口的时候自动找到这个C实现的Native函数调用。

　　当然函数名太长，可以在.c文件中通过函数名映射表来实现简化。

**第三步：**

实现JNI原生函数源文件：

新建com\_lucyfyr\_HelloWorld.c文件：

[复制代码](javascript:void(0);)

#include <jni.h>

#define LOG\_TAG "HelloWorld"

#include <utils/Log.h>

/\* Native interface, it will be call in java code \*/

JNIEXPORT jstring JNICALL Java\_com\_lucyfyr\_HelloWorld\_printJNI(JNIEnv \*env, jobject obj,jstring inputStr)

{

　　LOGI("dufresne Hello World From libhelloworld.so!");

　　// 从 instring 字符串取得指向字符串 UTF 编码的指针

　　const char \*str =

　　(const char \*)(\*env)->GetStringUTFChars( env,inputStr, JNI\_FALSE );

　　LOGI("dufresne--->%s",(const char \*)str);

　　// 通知虚拟机本地代码不再需要通过 str 访问 Java 字符串。

　　(\*env)->ReleaseStringUTFChars(env, inputStr, (const char \*)str );

　　return (\*env)->NewStringUTF(env, "Hello World! I am Native interface");

}

/\* This function will be call when the library first be load.

\* You can do some init in the libray. return which version jni it support.

\*/

jint JNI\_OnLoad(JavaVM\* vm, void\* reserved)

{

　　void \*venv;

　　LOGI("dufresne----->JNI\_OnLoad!");

　　if ((\*vm)->GetEnv(vm, (void\*\*)&venv, JNI\_VERSION\_1\_4) != JNI\_OK) {

　　　　LOGE("dufresne--->ERROR: GetEnv failed");

　　　　return -1;

　　}  
　　return JNI\_VERSION\_1\_4;

}

[复制代码](javascript:void(0);)

**OnLoadJava\_com\_lucyfyr\_HelloWorld\_printJNI**

函数里面做一些log输出 注意JNI中的log输出的不同。

*JNI\_OnLoad函数JNI规范定义的，当共享库第一次被加载的时候会被回调，*

*这个函数里面可以进行一些初始化工作，比如注册函数映射表，缓存一些变量等，*

*最后返回当前环境所支持的JNI环境。本例只是简单的返回当前JNI环境。*

**第四步：**

编译生成so库

　　编译com\_lucyfyr\_HelloWorld.c成so库可以和app一起编译，也可以都单独编译。

在当前目录下建立jni文件夹：HelloWorld/jni/

　　下建立Android.mk ，并将com\_lucyfyr\_HelloWorld.c和 com\_lucyfyr\_HelloWorld.h 拷贝到进去

编写编译生成so库的Android.mk文件：

[复制代码](javascript:void(0);)

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

# 一个完整模块编译

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES:=com\_lucyfyr\_HelloWorld.c

LOCAL\_C\_INCLUDES := $(JNI\_H\_INCLUDE)

LOCAL\_MODULE := libHelloWorldJni

LOCAL\_SHARED\_LIBRARIES := libutils

LOCAL\_PRELINK\_MODULE := false

LOCAL\_MODULE\_TAGS :=optional

include $(BUILD\_SHARED\_LIBRARY)

[复制代码](javascript:void(0);)

系统变量解析：

　　LOCAL\_PATH - 编译时的目录  
　　$(call 目录，目录….) 目录引入操作符  
　　　　如该目录下有个文件夹名称 src，则可以这样写 $(call src)，那么就会得到 src 目录的完整路径

　　include $(CLEAR\_VARS) -清除之前的一些系统变量  
　　LOCAL\_MODULE － 编译生成的目标对象  
　　LOCAL\_SRC\_FILES － 编译的源文件  
　　LOCAL\_C\_INCLUDES － 需要包含的头文件目录  
　　LOCAL\_SHARED\_LIBRARIES － 链接时需要的外部库  
　　LOCAL\_PRELINK\_MODULE － 是否需要prelink处理   
　　include$（BUILD\_SHARED\_LIBRARY） － 指明要编译成动态库

　　android.mk编译模块添加具体方法参考：<http://blog.csdn.net/yili_xie/article/details/4906865>

编译此模块：输入编译命令

**./makeMtk mm packages/apps/HelloWorld/jni/**

　　上面是我的工程根目录编译命令。具体编译方式根据自己系统要求执行。

编译输出： libHelloWorldJni.so (system/lib中视具体而定)

　　此时库文件编译好了可以使用，如果在eclipse中模拟器上使用，需要将 libHelloWorldJni.so导入到system/lib 下，

　　或者对应app的data/data/com.lucyfyr/lib/下；

但是在导入system/lib 下的时候，却始终提示out of memory。于是网上说使用emulator去搞，

　　但是此命令使用起来却有问题，始终没有解决。但是可以导入到data/data/com.lucyfyr/lib/下使用。但是我这里直接在真机上弄的。

看一下HelloWorld中Android.mk文件的配置

　　其中存在：

　　　　include $(LOCAL\_PATH)/jni/Android.mk 表示编译库文件

　　　　LOCAL\_JNI\_SHARED\_LIBRARIES := libHelloWorldJni 表示app依赖库，打包的时候会一起打包。

**第五步：**

验证执行

　　将编译好的apk安装到手机上

　　使用adb push到手机上去需要自己去导入库文件libHelloWorldJni.so到data/data/com.lucyfyr/lib/

　　使用adb install方式安装则会自动导入。

启动HelloWorld ：输入命令 **adb logcat |grep dufresne**

输出log如下：

　　I/HelloWorld(28500): dufresne Hello World From libhelloworld.so!

　　I/HelloWorld(28500): dufresne--->I am HelloWorld Activity

　　V/dufresne(28500): Hello World! I am Native interface

符合调用打印顺序正确。

以上通过一个简单的例子学习了Android如何编写编译C库文件，以及如何使用库文件。

对于JNI的使用其中还涉及到其他方面的一些知识：C++接口如何调用，函数名注册表怎么回事，

参数类型如何匹配，JNI如何调用Java中的方法java<--->JNI等。