

终于找到一篇极佳的 NDK 入门文章

张风捷特烈 鸿洋 10月24日

本文作者

作者：张风捷特烈

链接：

<https://juejin.im/post/5d984028518825095879e45d>

本文由作者授权发布。

知其然，知其所以然；方能以不变，应其万变。----张风捷特烈

0 前言

笔者看了一些NDK的项目。一些教程不是HelloWord就是直接整FFmpeg或OpenCV,可谓一个天一个地，而且目录结构和Android3.5的默认结构并不是太一致,一直没找到什么合心的文章。故写此文连接这天地，来总结一下在NDK开发之前你应知道的东西。

在此之前，先划分三类人,如果不认清自己是什么角色就去玩NDK,你会很糟心：

- user：纯粹.so链接库使用者(伸手党)
- creator：纯粹ndk开发者,创作.so链接库(创作家)
- designer：在现有的.so上自己开发.so链接库实现特定功能(程序设计师)

本文内容

1. 本文将以user、creator、designer三者的视角来看NDK
2. AndroidStudio3.5的默认目录结构
3. 有现成的C++代码，如何让Android调用它？
4. arm64-v8a、armeabi-v7a、x86、x86_64分别是干嘛的？
5. 动态链接库.so是什么鬼，如何从c/c++生成.so？

6.libs,jniLibs,jin目录到底该怎么放?如何自定义文件放置的位置?

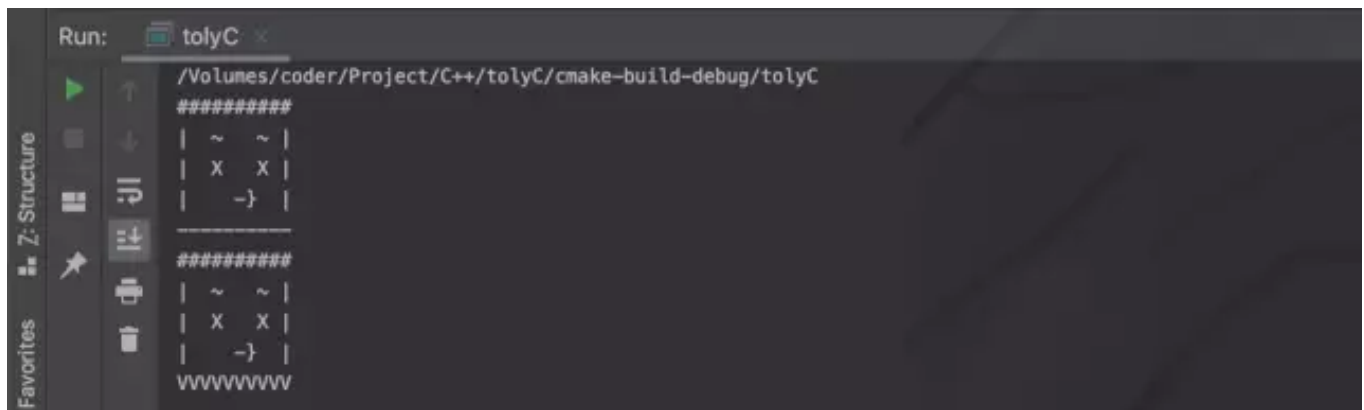
7.一些让人糟心的异常

前置知识

也许你很怕C++,就像你在新手村被3级的boss虐到心理阴影,但是你现在已经50级了,还怕曾经虐你的3级的boss吗?

建议阅读:[- C++趣玩篇1 -] 从打印开始说起,这篇对本文很重要,是简单,也很有趣。现在情况如此:上篇中C++实现了一个打印脸的类,我想在Android中使用它。

<https://juejin.im/post/5d95a605e51d45783f5aa4cd>



1 对于纯粹.so使用者(User)

1.目录结构

当你只是单纯的使用动态链接库.so中的已有功能,也就是传说中的伸手党。

那你与NDK只是擦肩而过,并不需要理会C/C++,也不需要创建一个NDK的项目,甚至连JNI都有现成的。

你所需要做的只是在main下新建jniLibs,经过测试,其为默认的.so成放置地,此时gradle文件你可以一字不动。

app 目录下



2. JNI接口定义

俗话说拿人家手短，吃人家嘴软。由于JNI是根据包名找到C/C++函数的,使用时必须和creator定义的接口完全一致(包括包名)。

```
---->[com.toly1994.jni_creator.Facer]--by 张风捷特烈-----
package com.toly1994.jni_creator;
public class Facer {
    public static native String getFacer( String top, String bottom, String brow, String eyes);
}
```

3. 库的使用

这个库是等会要创造的,这里先来演示。System.loadLibrary指定库名
其中库全名为libtoly_facer-lib.so, 加载时toly_facer-lib即可

这样在上一篇[- C++趣玩篇1 -] 从打印开始说起中实现的打印类就可以在Android中使用。

<https://juejin.im/post/5d95a605e51d45783f5aa4cd>



```

public class MainActivity extends AppCompatActivity {
    static { //加载类库
        System.loadLibrary("toly_facer-lib");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView=findViewById(R.id.hello);
        textView.setTextSize(30);
        //通过native接口getFacer使用类库中C++方法
        textView.setText(Facer.getFacer("-", "-", "~", "X"));
    }
}

```

OK,现在80%的人问题解决了。(手动搞笑)

2 对于纯粹ndk开发者(Creator)

如果你有现成的C++代码想要直接用在Android上，或者想要手撸个什么高效的框架，或者想要让你的源码不容易破解，那么废话不多说，就开整吧。哥敬你是条好汉。

现在你需要创建一个Native C++ 的Android项目。

这里就来实现toly_facer-lib

1.准备活动

上一篇中已经完成了C++类

头文件

```

--->[app/src/main/cpp/Facer.h]---
//
// Created by 张风捷特烈 on 2019/10/3.
//

#include <iostream>

using namespace std;

```

```

#ifndef TOLYC_FACER_H
#define TOLYC_FACER_H

class Facer {
public:
    Facer(const string &top="#", const string &bottom="#", const string &brow="~", const string &eyes=".");
    ~Facer();
public:
    string top;
    string bottom;
    string brow;
    string eyes;
public:
    void printFace() ;
    string getFace();
};

#endif //TOLYC_FACER_H

```

cpp实现文件

```

--->[app/src/main/cpp/Facer.cpp]---
//
// Created by 张风捷特烈 on 2019/10/3.
//

#include "Facer.h"

Facer::Facer(
    const string &top,
    const string &bottom,
    const string &brow,
    const string &eyes) : top(top),
                        bottom(bottom),
                        brow(brow),
                        eyes(eyes) {}

void Facer::printFace() {
    cout<< getFace() << endl;
}

Facer::~Facer() {

}

string Facer::getFace() {
    string result;
    for (int i = 0; i < 10; ++i) {
        i != 9 ? result+=top : result+=top+"\n";
    }
    result+= " | " +brow + " | " + brow + " | " +"\n";
    result+= " | " +eyes + " | " + eyes + " | " +"\n";
    result+= " |  -}  | \n";
    for (int i = 0; i < 10; ++i) {
        i != 9 ? result+=bottom : result+=bottom+"\n";
    }
    return result;
}

```

2.项目结构

新建Native C++ 的项目之后，main文件夹下会有cpp文件夹，这就是C++代码的家
如果直接将两个Facer文件拷贝进去,会飘红。因为还没有在CmakeLists中进行配置



app目录下

3.CmakeLists中的配置

```
cmake_minimum_required(VERSION 3.4.1)

add_library(toly_facer-lib SHARED
    toly_facer-lib.cpp Facer.h Facer.cpp)#直接加入文件

find_library(log-lib log)

target_link_libraries(toly_facer-lib ${log-lib})
```

当然也许你肯定懒得一个个添加，可以加载cpp文件夹下的所有.cpp和.c文件

```
cmake_minimum_required(VERSION 3.4.1)
```

```
cmake_minimum_required(VERSION 3.4.1)

#定义全局 my_source_path 变量
file(GLOB my_source_path
    ${CMAKE_SOURCE_DIR}/*.cpp
    ${CMAKE_SOURCE_DIR}/*.c)

add_library(toly_facer-lib
    SHARED ${my_source_path})
```

```
find_library(log-lib log)

target_link_libraries(toly_facer-lib ${log-lib})
```

4.设计JNI的native接口方法和C++实现

此方法所属类名、包名对user都至关重要。对于creator随意啦，就是任性

```
---->[src/main/java/com/toly1994/jni_creator/Facer.java]----
package com.toly1994.jni_creator;

public class Facer {
    public static native String getFacer( String top, String bottom, String brow, String eyes);
}
```

C++与Java的相互作用,就是Java进行输入,经C++转化将有价值的东西传给Java端

```
---->[src/main/cpp/toly_facer-lib.cpp]----
#include <jni.h>
#include <string>
#include "Facer.h"

extern "C"
JNIEXPORT jstring JNICALL
Java_com_toly1994_jni_1creator_Facer_getFacer(JNIEnv *env, jclass clazz, jstring top,
                                              jstring bottom, jstring brow, jstring eyes) {
    Facer facer(//使用 env->GetStringUTFChars将jstring转化为string
                env->GetStringUTFChars(top, 0),
                env->GetStringUTFChars(bottom, 0),
                env->GetStringUTFChars(brow, 0),
                env->GetStringUTFChars(eyes, 0)
    );

    return env->NewStringUTF(facer.getFace().c_str());
}
```

基本上流程就是这样。

3 扫盲科普

1.arm64-v8a、armeabi-v7a、x86、x86_64

arm 架构注重的是续航能力:大部分的移动设备

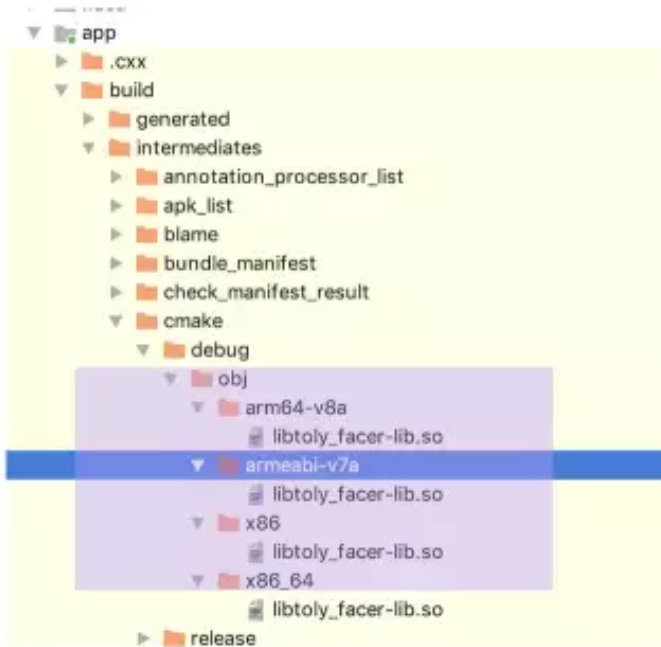
x86 架构注重的是性能:大部分的台式机和笔记本电脑

arm64-v8a :第8代、64位ARM处理器

armeabi-v7a :第7代及以上的 ARM处理器

x86 : x86 架构的 CPU (Intel 的 CPU)
x86_64:x86 架构的64位 CPU (Intel 的 CPU)

默认会编译出四种.so文件

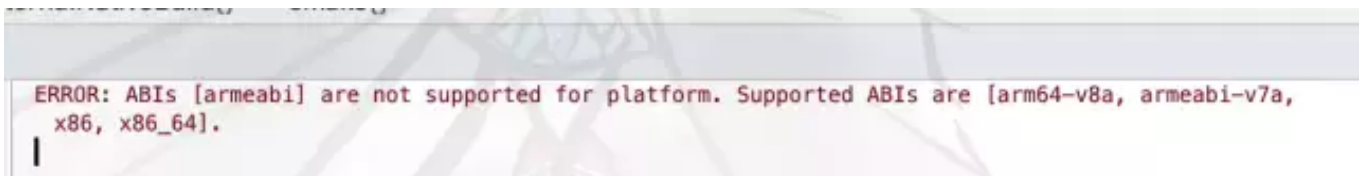


默认情况会编译出四种.so

2.配置输出的.so架构类型

可以通过app下的build.gradle来指定编译的.so类型

注意只有这四种类中，以前很多项目中存在abiFilters 'armeabi'但现在会崩



```
android {  
    defaultConfig {  
        externalNativeBuild {  
            cmake {  
                abiFilters 'armeabi-v7a', 'arm64-v8a'  
            }  
        }  
    }  
}
```


这样清一下项目，再编译出来的只有'armeabi-v7a', 'arm64-v8a'

此时运行到模拟器上，会发现找不到类库，则说明模拟器去X86的。运行到真机无误，则说明真机是arm的

3..so文件是什么？

如果说.dll估计你会说：哦，好像见过。

其实.so和.dll并没有本质的区别，它们都是一个C++实现的功能团。

只不过.so是用在linux上的，.dll是用在Windows上的。

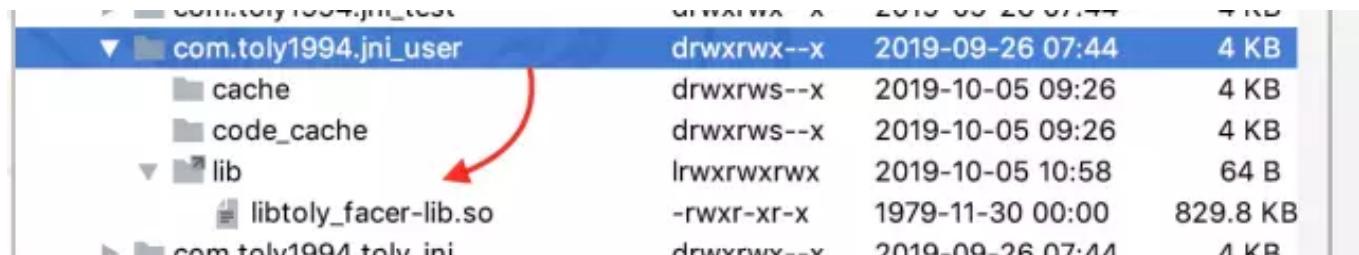
如今操作系统三足鼎立，当然少不了MacOS,类似的在MacOS中有.dylib文件。

它们都是 C++ 的动态链接库(Dynamic Link Library)

而Android作为Linux的一员，C++ 编译出的.so便是顺理成章。

那如何将C++编译成.so库?这便是NDK在做的事，也是上面在做的事。

打包时gradle会将对应的.so包打到apk里,然后.so就能在linux里愉快的玩耍了。



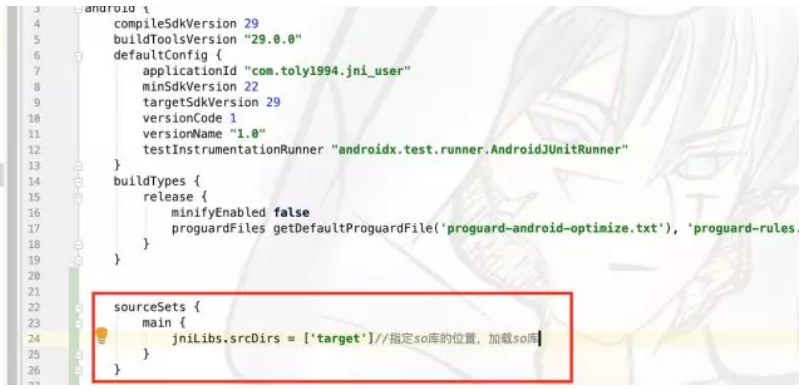
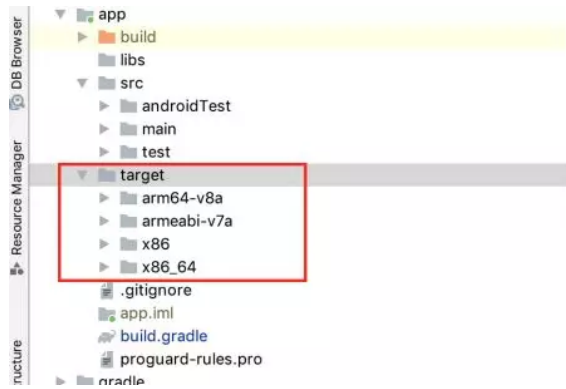
com.toly1994.jni_test	drwxrwx--x	2019-09-26 07:44	4 KB
cache	drwxrws--x	2019-10-05 09:26	4 KB
code_cache	drwxrws--x	2019-10-05 09:26	4 KB
lib	lrwxrwxrwx	2019-10-05 10:58	64 B
libtoly_facer-lib.so	-rwxr-xr-x	1979-11-30 00:00	829.8 KB
com.toly1994.toly_jni	drwxrwx--x	2019-09-26 07:44	4 KB

4.如何自定义资源文件位置

个人建议习惯优于配置,用默认挺好的。如果你是非常有个性的...也可以在gradle里进行制定

虽然你也许不会用到，但是看一下,看到要认得，不至一脸蒙圈。

对于使用者，可以随意指定盛放.so的文件夹，需要在app下的build.gradle配置



```

android {
    ...
    sourceSets {
        main {
            jniLibs.srcDirs = ['target']//指定so库的位置·加载so库
        }
    }
}

```

对于创造者，也可以使用jni.srcDirs来指定C++代码盛放的位置

```

sourceSets.main{
    jni.srcDirs = ["src/main/cpp"]
}

```

4 对于程序设计师 (Designer)

俗话说难的不是重写，而是对烂代码的重构，有时候修改比创作更难。

已有的.so文件但功能上又需要定制，于是第三类就诞生了，也是最头疼的。

其实FFmpeg和OpenCV等都是这第三类，用已存在事物去构建新事物，便是设计。

1.项目结构

算法和核心代码已经实现，我们需要做的是结合业务进行接口封装及方法调用
 这里我就用OpenCV的使用来进行演示: 你需要创建的是Native C++项目
 (Opencv下载什么的，不废话了，详见:OpenCV专题1 - AndroidStudio的JNI工程及引用OpenCV)



app文件夹下

C++创作区

定义JNI接口

已有的.so文件

Designer目的:

- 1.强化封装, 输出.so
- 2.直接应用于项目中

2.你的角色

这时, 你是设计者, 兼具创造者和使用者两重角色。但比纯粹的创造要简单, 比纯粹的使用要难。

这时可以通过CmakeLists去链接到OpenCV的.so文件, 这样你就可以使用OpenCV的头文件进行功能实现

```
cmake_minimum_required(VERSION 3.4.1)

include_directories(include)#引入include文件夹

#定义全局 my_source_path 变量
file(GLOB my_source_path ${CMAKE_SOURCE_DIR}/*.cpp ${CMAKE_SOURCE_DIR}/*.c)

add_library(toly_cv SHARED ${my_source_path})

#添加动态链接库
add_library(lib_opencv SHARED IMPORTED)
set_target_properties(lib_opencv PROPERTIES IMPORTED_LOCATION
    ${CMAKE_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libopencv_java4.so) #so文件位置

## 在ndk中查找log库 取别名log-lib
find_library(log-lib log)
# 在ndk中查找jnigraphics库 取别名jnigraphics-lib jnigraphics
find_library(jnigraphics jnigraphics)

target_link_libraries(
    toly_cv
    lib_opencv
    jnigraphics
    log
)
```

你可以定义一个JNI接口来暴露你在C++层实现的方法，再打包成.so供他人使用
这便是开源的魅力，比如下面的灰色图像,使用者可以拿着打出的.so包通过TolyCV来使用



```
cmake_minimum_required(VERSION 3.4.1)
```

```
---->[com.toly1994.jni_designer.TolyCV]----
```

```
public class TolyCV {
    public static native Bitmap grayBitmap(Bitmap bitmap, Bitmap.Config argb8888);
}
```

```
---->[src/main/cpp/toly_cv.cpp]----
```

```
#include <jni.h>
#include <string>
#include "bitmap_utils.h"
#include <opencv2/imgproc/types_c.h>
```

```
extern "C"
```

```
JNIEXPORT jobject JNICALL
```

```
Java_com_toly1994_jni_1designer_TolyCV_grayBitmap(JNIEnv *env, jclass clazz, jobject bitmap, jobject argb8888) {
```

```
    Mat srcMat;
```

```
    Mat dstMat;
```

```
    bitmap2Mat(env, bitmap, &srcMat);
```

```
    cvtColor(srcMat, dstMat, CV_BGR2GRAY); //将图片的像素信息灰度化盛放在dstMat
```

```
    return createBitmap(env, dstMat, argb8888); //使用dstMat创建一个Bitmap对象
```

```
}
```

5 让人糟心的异常

笔者也并非一路畅通无阻，走的坑也挺多,下面几个坑来给你说说

1.ninja: error: 巴拉巴拉... missing and no known rule to make it

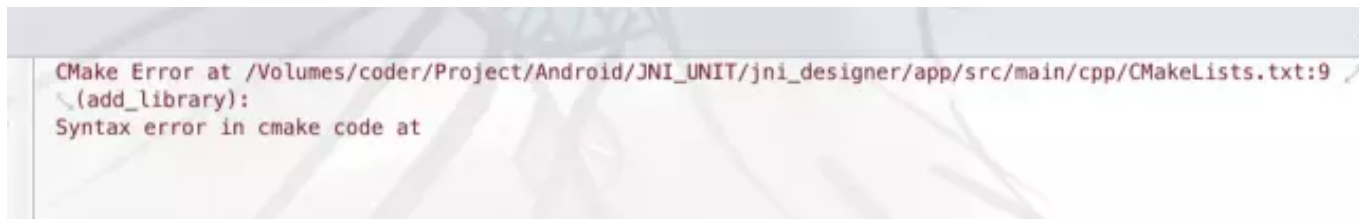


```
Build command failed.
Error while executing process /Volumes/coder/SDK/AndroidAll/sdk/cmake/3.10.2.4988404/bin/ninja with
arguments {-C /Volumes/coder/Project/Android/JNI_UNIT/jni_designer/app/.cxx/cmake/debug/armeabi-v7a
toly_cv}
ninja: Entering directory `/Volumes/coder/Project/Android/JNI_UNIT/jni_designer/app/
.cxx/cmake/debug/armeabi-v7a'

ninja: error: '/Volumes/coder/Project/Android/JNI_UNIT/jni_designer/app/src/main/jniLibs/armeabi-v7a
/libopencv_java4.so', needed by '/Volumes/coder/Project/Android/JNI_UNIT/jni_designer/app/build
/intermediates/cmake/debug/obj/armeabi-v7a/libtoly_cv.so', missing and no known rule to make it
```

仔细排查CmakeLists,可能是.so文件的路径不对

2.CMake Error at 巴拉巴拉... (add_library):



```
CMake Error at /Volumes/coder/Project/Android/JNI_UNIT/jni_designer/app/src/main/cpp/CMakeLists.txt:9
(add_library):
Syntax error in cmake code at
```

仔细排查CmakeLists,可能是你的C++代码文件路径不对

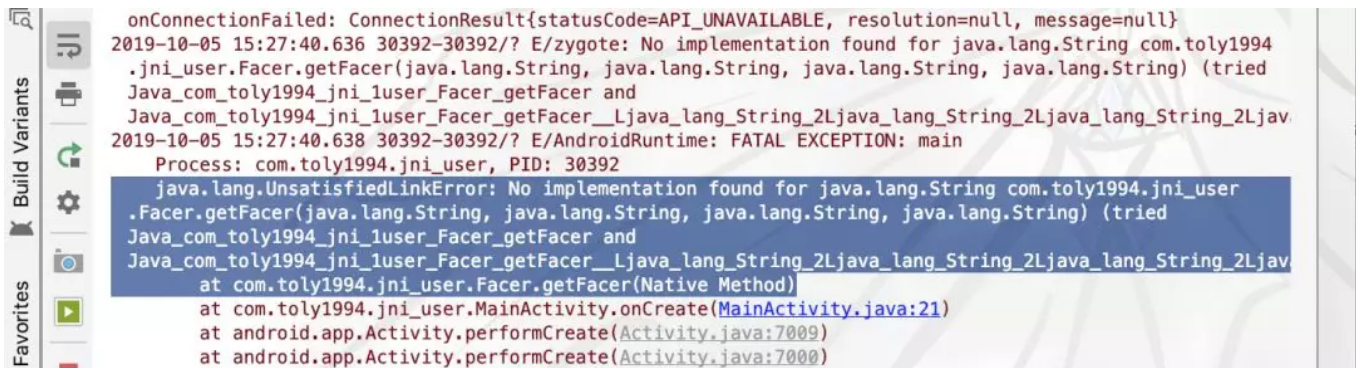
3.java.lang.UnsatisfiedLinkError: 巴拉巴拉... "XXX.so"



```
2019-10-05 15:24:04.258 30116-30116/com.toly1994.jni_designer E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.toly1994.jni_designer, PID: 30116
java.lang.UnsatisfiedLinkError: dalvik.system.PathClassLoader[DexPathList[[zip file "/data/app/com.toly1994.jni_designer-Gsxbo5THxYKA25Nf0QupbQ==/base.apk!/lib/x86, /system/lib, /vendor/lib/...]]'
at java.lang.Runtime.loadLibrary0(Runtime.java:1011)
at java.lang.System.loadLibrary(System.java:1657)
at com.toly1994.jni_designer.MainActivity.<clinit>(MainActivity.java:14)
at java.lang.Class.newInstance(Native Method)
at android.app.Instrumentation.newActivity(Instrumentation.java:1174)
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2669)
at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2856)
at android.app.ActivityThread.-wrap11(Unknown Source:0)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1500)
```

说明你的库加载异常, 看看你的库名有没有写对

4. java.lang.UnsatisfiedLinkError: No implementation found for java.lang.String 巴拉巴拉...



```
onConnectionFailed: ConnectionResult{statusCode=API_UNAVAILABLE, resolution=null, message=null}
2019-10-05 15:27:40.636 30392-30392/? E/zygote: No implementation found for java.lang.String com.toly1994
.jni_user.Facer.getFacer(java.lang.String, java.lang.String, java.lang.String, java.lang.String) (tried
Java_com_toly1994_jni_luser_Facer_getFacer and
Java_com_toly1994_jni_luser_Facer_getFacer__Ljava_lang_String_2Ljava_lang_String_2Ljava_lang_String_2Ljav
2019-10-05 15:27:40.638 30392-30392/? E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.toly1994.jni_user, PID: 30392
java.lang.UnsatisfiedLinkError: No implementation found for java.lang.String com.toly1994.jni_user
.Facer.getFacer(java.lang.String, java.lang.String, java.lang.String, java.lang.String) (tried
Java_com_toly1994_jni_luser_Facer_getFacer and
Java_com_toly1994_jni_luser_Facer_getFacer__Ljava_lang_String_2Ljava_lang_String_2Ljava_lang_String_2Ljav
at com.toly1994.jni_user.Facer.getFacer(Native Method)
at com.toly1994.jni_user.MainActivity.onCreate(MainActivity.java:21)
at android.app.Activity.performCreate(Activity.java:7009)
at android.app.Activity.performCreate(Activity.java:7000)
```

说明你的JNI接口和.so比匹配，自行匹配放到相应包名下

待续...

所以，在决心奋战NDK的时候，先认清自己是什么角色,才好分类。

Creator太过遥远，我就想做个安安静静的Designer。

我一直在找一篇这样的文章,但是没找到。所以自己写了一篇,希望对你有所帮助。

我是张风捷特烈,如果有什么想要交流的,欢迎留言。

推荐阅读：

[基于Jetpack的全系列加实战 app 教程](#)

[面试官：今日头条启动很快，你觉得可能是做了哪些优化？](#)

[都9102年了，Android 冷启动优化除了老三样还有哪些新招？](#)



扫一扫 关注我的公众号

如果你想要跟大家分享你的文章，欢迎投稿~

┐(^ 0 ^)┌明天见!

文章已于2019-10-24修改

[阅读原文](#)