

### 1. Android\_jni 与 so and java

Android 程序分为 java 和 native 两层，java 层是 java 代码编译成的 dex 文件，native 层是 c++ 代码编译成的 so(动态链接库)文件。两者通过 jni(java native interface)进行链接。相比于 java 层，native 层的安全性更高，隐蔽性更好，在一些情况下效率也更高，当前的加密和检测一般都在 native 层中进行。

### 2. Android studio 的原生 jni 支持

Android studio 支持 jni 代码编程，可以直接创建编译 so 库，需修改 gradle 文件来开启 jni 支持。

修改步骤：

1) 修改 ./gradle/wrapper/gradle-wrapper.properties

2) 修改 build.gradle

buildscript{

dependencies{

classpath 'com.android.tools.build:gradle-experimental:0.6.0-beta5' //使用实验性的 gradle

}

}

3) 修改 build.gradle(一个示例)

```
apply plugin: 'com.android.model.application'
```

```
model {
```

```
    android {
```

```
        compileSdkVersion 23
```

```
        buildToolsVersion "23.0.2"
```

```
        defaultConfig {
```

```
            applicationId "f8.hellonative"
```

```
            minSdkVersion.apiLevel 15
```

```
            targetSdkVersion.apiLevel 23
```

```
            versionCode 1
```

```
            versionName "1.0"
```

```
            buildConfigFields {
```

```
                create() {
```

```
defaultConfig {  
    applicationId "f8.hellonative"  
    minSdkVersion.apiLevel 15  
    targetSdkVersion.apiLevel 23  
    versionCode 1  
    versionName "1.0"  
  
    buildConfigFields {  
        create() {  
            type "int"  
            name "VALUE"  
            value "1"  
        }  
    }  
}
```

```
}  
buildTypes {  
    release {  
        minifyEnabled false  
        proguardFiles.add(file("proguard-rules.pro"))  
    }  
}  
  
ndk {  
    moduleName "hello"  
    ldLibs.add("log")  
}  
}
```

```

        ndk {
            moduleName "hello"
            ldLibs.add("log")
        }
    }

    dependencies {
        compile fileTree(dir: 'libs', include: ['*.jar'])
        //testCompile 'junit:junit:4.12'
        compile 'com.android.support:appcompat-v7:23.1.1'
        compile 'com.android.support:design:23.1.1'
    }
}

```

如果 java 包无法编译通过，报告版本错误的，可以添加下面这段

```

tasks.withType(JavaCompile){ //指定编译 JDK 版本
    sourceCompatibility = JavaVersion.VERSION_1_7
    targetCompatibility = JavaVersion.VERSION_1_7
}

```

### 3. Native 函数与注册

So 库中通常具有 Jni\_onLoad 函数，是 so 加载时候同时启动的，执行一些初始化操作，可以不进行重写。

```

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM *vm, void* reserved) {
    JNIEnv* env = NULL;
    jint result = -1;

    if (vm->GetEnv((void**)&env, JNI_VERSION_1_6) != JNI_OK) {
        LOGE("This jni version is not supported");
        return -1;
    }
    if (registerNativeMethods(env, gClassName, gMethods,
                             sizeof(gMethods) / sizeof(gMethods[0])) == JNI_FALSE)
    {

```

```

        LOGE("Unable to register native methods!!!");
        return -1;
    }

    LOGE("So load success");
    return JNI_VERSION_1_6;
}

```

Native 函数声明

例如函数 `public static pHello();`

Java 层声明为 `public native static String pHello();`

静态注册方法:

Native 层 则 为 `JNIEXPORT jstring JNICALL`

`Java_f8_hellonative_MainActivity_Hello1(JNIENV *env, jclass type)`

So 中的名字为类名+函数名的组合, 并且自带两个参数:

`JNIENV* env`

`jclass`(static 方法时)或者 `jobject`(普通方法时)

动态注册方法:

调用 `RegisterNatives` 函数

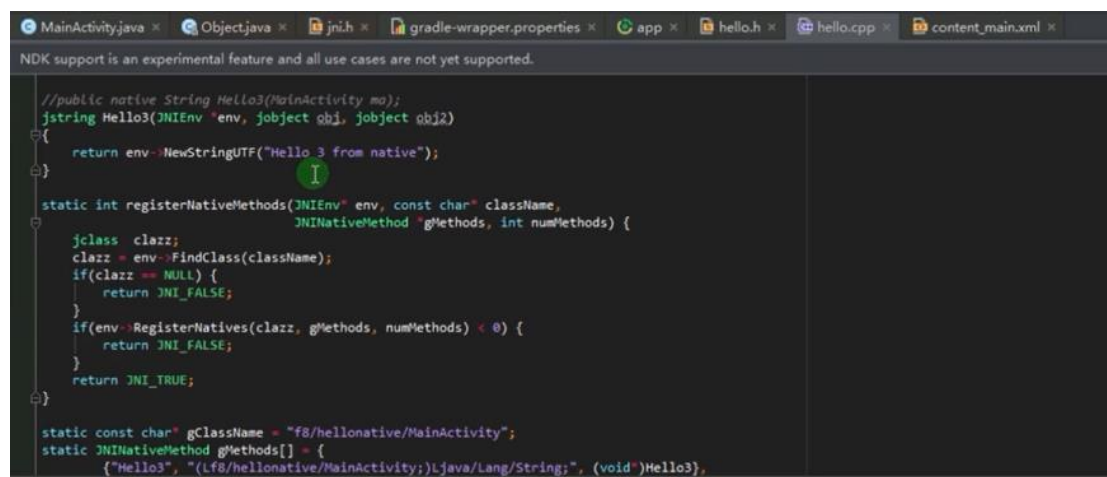
```
jint RegisterNatives(jclass clazz, const JNINativeMethod* methods,
                    jint nMethods)
```

可以使用下面的函数注册

```
static int registerNativeMethods(JNIEnv* env, const char* className,
                                JNINativeMethod *gMethods, int numMethods) {
    jclass clazz;
    clazz = env->FindClass(className);
    if(clazz == NULL) {
        return JNI_FALSE;
    }
    if(env->RegisterNativeMethods(clazz, gMethods, numMethods) < 0) {
        return JNI_FALSE;
    }
    return JNI_TRUE;
}
```

```
static const char* gClassName = "f8/dexshell/MainActivity";
static JNINativeMethod gMethods[] = {
    {"HelloWorld", "()Ljava/lang/String;", (void*)HelloWorldL},
};
```

#### 4. Demo



Ljava/Lang/String 改为 Ljava/lang/String

```
MainActivity.java × Object.java × jni.h × gradle-wrapper.properties × app × hello.h × hello.cpp × content_main.xml ×
NDK support is an experimental feature and all use cases are not yet supported.

static const char* gClassName = "f8/hellonative/MainActivity";
static JNINativeMethod gMethods[] = {
    {"Hello3", "(Lf8/hellonative/MainActivity;)Ljava/lang/String;", (void*)Hello3},
};

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* vm, void* reserved) {
    JNIEnv* env = NULL;
    jint result = -1;

    if (vm->GetEnv((void**) &env, JNI_VERSION_1_6) != JNI_OK) {
        return -1;
    }
    if (registerNativeMethods(env, gClassName, gMethods,
        sizeof(gMethods) / sizeof(gMethods[0])) != JNI_FALSE) {
        return -1;
    }

    return JNI_VERSION_1_6;
}
```

```
ndk {
    moduleName "hello"
    ldlibs.add("log");
}
```

```
public class MainActivity extends AppCompatActivity {
    static {
        System.loadLibrary("hello");
    }
}
```

```
MainActivity.java × Object.java × jni.h × gradle-wrapper.properties × app × hello.h × hello.cpp × content_main.xml ×

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}

public native String Hello3(MainActivity ma);
}
```