

0x00 闲言碎语

最近状态很糟，写点东西静下心，捋捋思绪，就总结下之前学得JNI的一些知识点好了

0x01 一个NDK demo

新建项目工程

```
package myapplication.mask.com.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    static{
        System.loadLibrary("demo");//载入类库
    }

    public native static String FromNative();//声明Native方法

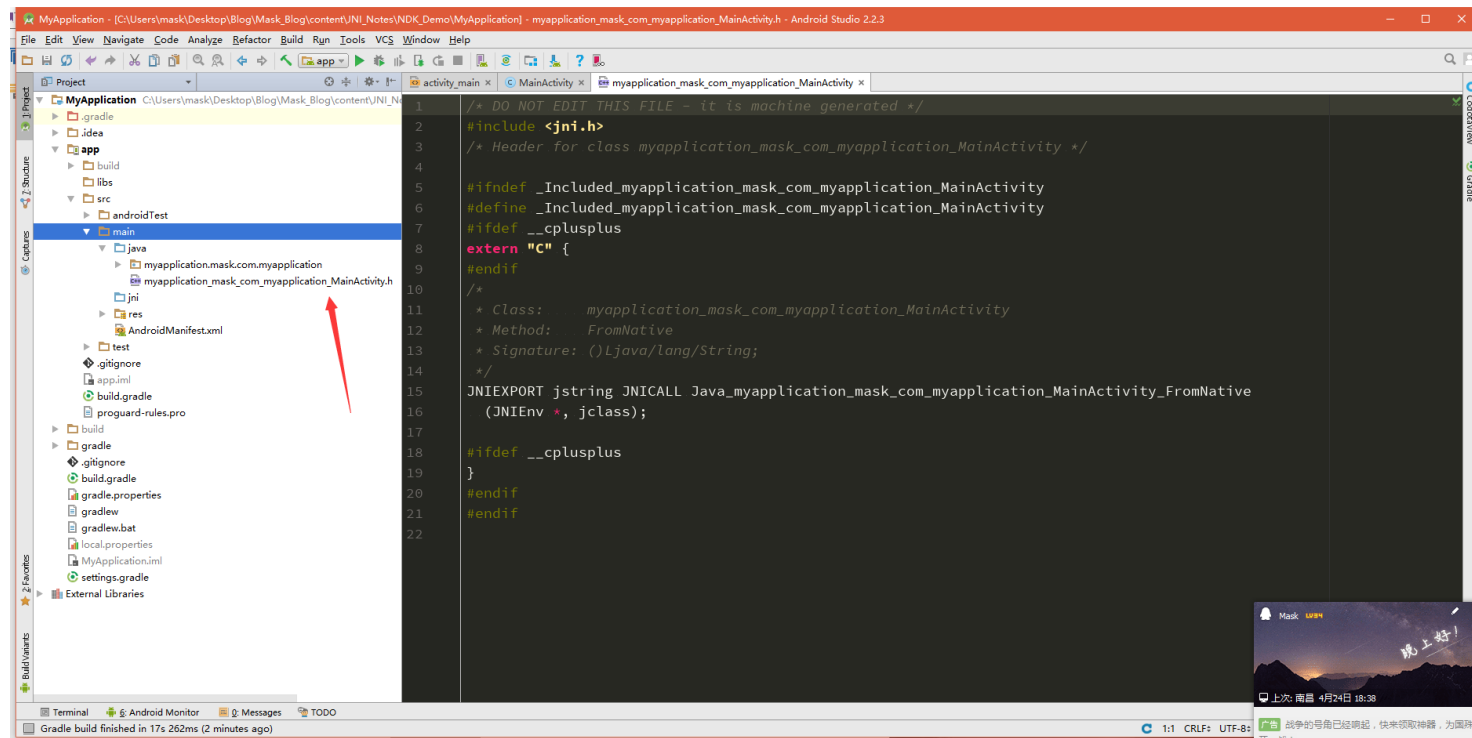
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toast.makeText(this, FromNative(), Toast.LENGTH_SHORT).show();
    }
}
```

然后Make Project

下面切换到app/src/main/java目录下，执行下面的命令生成一个JNI头文件

```
javah myapplication.mask.com.myapplication.MainActivity
```



我们来看看这个头文件都有些啥

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h> //需要导入jni.h头文件
/* Header for class myapplication_mask_com_myapplication_MainActivity */

#ifdef _Included_myapplication_mask_com_myapplication_MainActivity
#define _Included_myapplication_mask_com_myapplication_MainActivity
#endif
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class: myapplication_mask_com_myapplication_MainActivity /*类名*/
 * Method: FromNative /*方法名*/
 * Signature: ()Ljava/lang/String; /*签名信息（是不是很像smali代码?！！反正我第一反应是这个*/
 */
JNIEXPORT jstring JNICALL Java_myapplication_mask_com_myapplication_MainActivity_FromNative
    (JNIEnv *, jclass); //这个Native方法的声明

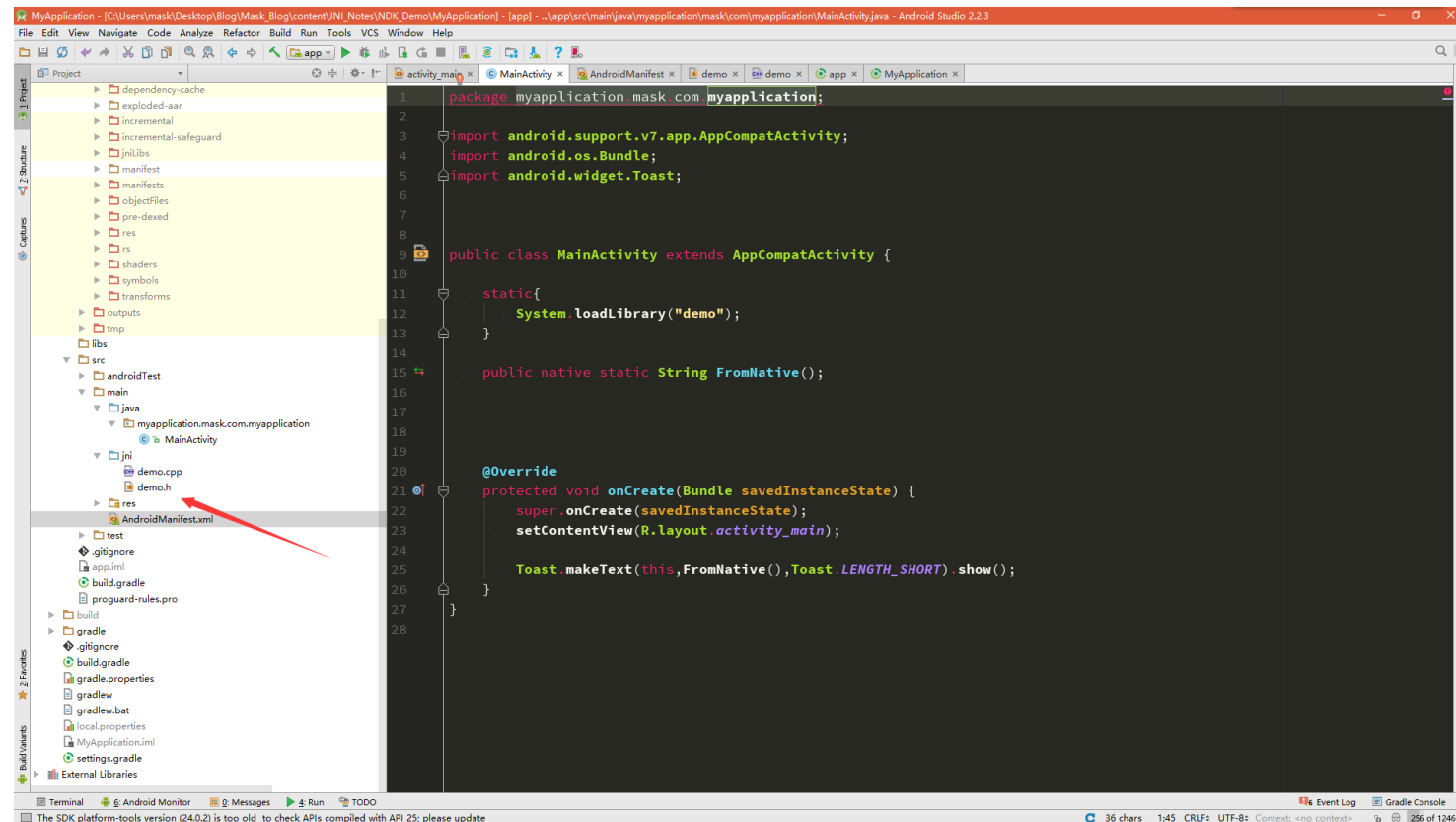
#ifdef __cplusplus
}
#endif
#endif
```

这里有个概念那就是我怎么知道这个声明的方法就是我要调用的呢？

这个是JNI规定的写法，为了方便，编译时会自动在头文件生成，包含的内容一看便知，将包名中的"."换成了"_"然后"+"类名

```
myapplication_mask_com_mypapplication_MainActivity
```

为了方便，我把这个头文件重命名为demo.h，然后新建一个JNI目录，将demo.h移动到JNI下



然后开始编译，将项目的build.gradle改成如下：

```
apply plugin: 'com.android.model.application' //此处加了model
model { //对应加上model
    android {
        compileSdkVersion 25
        buildToolsVersion "24.0.1"
        defaultConfig {
            applicationId "myapplication.mask.com.myapplication"
            minSdkVersion.apiLevel 19 //此处修改了
            targetSdkVersion.apiLevel 25//此处修改了
            versionCode 1
        }
    }
}
```

```

        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles.add(file("proguard-rules.pro"))//此处修改了
        }
    }
    ndk {
        moduleName "demo" //库名
        stl "stlport_static"//以静态链接方式使用的stlport版本的STL
        ldLibs.addAll(["log","z","android"])
        abiFilters.addAll(['armeabi','armeabi-v7a']) //abi平台
    }
    //sourceSets { main { jni.srcDirs = ['src/main/jni', 'src/main/jni/'] } }
}
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.1.0'
    testCompile 'junit:junit:4.12'
}

```

再来修改工程下的build.gradle:

```

// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle-experimental:0.8.3' //此处需要修改gradle的版本号

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

```

```
task clean(type: Delete) {  
    delete rootProject.buildDir  
}
```

其他编译方式:

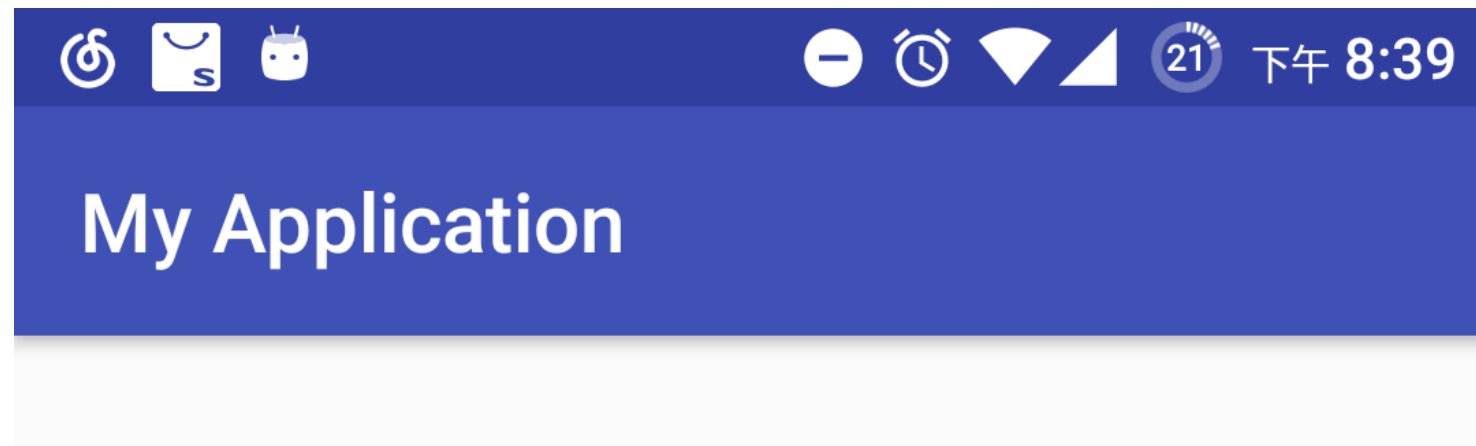
<http://www.tuicool.com/articles/3mu22ie>

http://www.jianshu.com/p/9d001d966053?utm_source=tuicool&utm_medium=referral

然后新建一个demo.cpp文件

```
//  
// Created by mask on 2017/4/25.  
//  
#include <iostream>  
#include "demo.h"  
#include <jni.h>  
  
using namespace std;  
  
/*头文件中的信息，可以直接复制过来  
 * Class: myapplication_mask_com_myapplication_MainActivity  
 * Method:FromNative  
 * Signature: ()Ljava/lang/String;  
 */  
JNIEXPORT jstring JNICALL Java_myapplication_mask_com_myapplication_MainActivity_FromNative  
    (JNIEnv *env, jclass clazz)  
    {  
    return env->NewStringUTF("Hello from JNI!");  
    }
```

然后运行:



Hello from JNI!

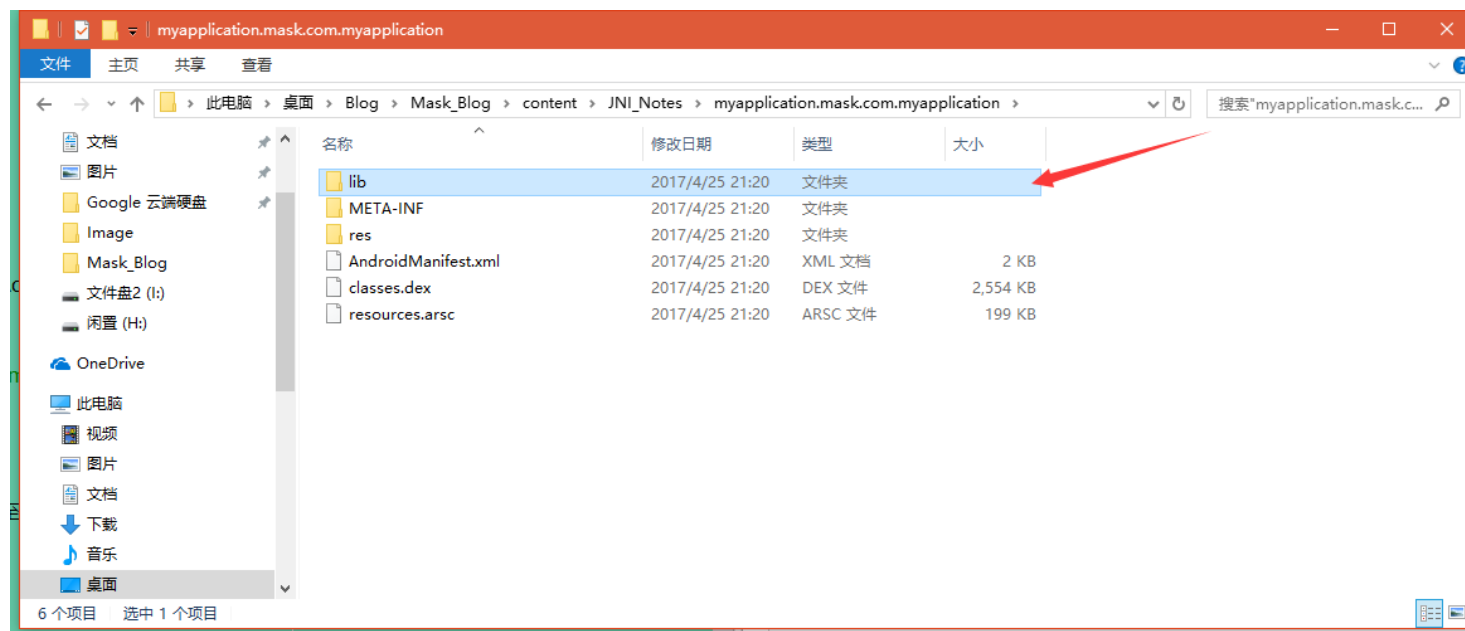
如果用c写的话, demo.c

```
//  
// Created by mask on 2017/4/25.  
//  
  
#include "demo.h"  
#include <jni.h>  
  
/*  
 * Class: myapplication_mask_com_myapplication_MainActivity  
 * Method: FromNative  
 * Signature: ()Ljava/lang/String;  
 */  
JNIEXPORT jstring JNICALL Java_myapplication_mask_com_myapplication_MainActivity_FromNative  
    (JNIEnv *env, jclass clazz)  
    {  
    return (*env)->NewStringUTF(env, "Hello from JNI!");  
    }
```

可以发现这(*env)其实是个二级指针, 而c++则是使用了一级指针, 至于分别指向了哪, 不急, 下面开始切入正题

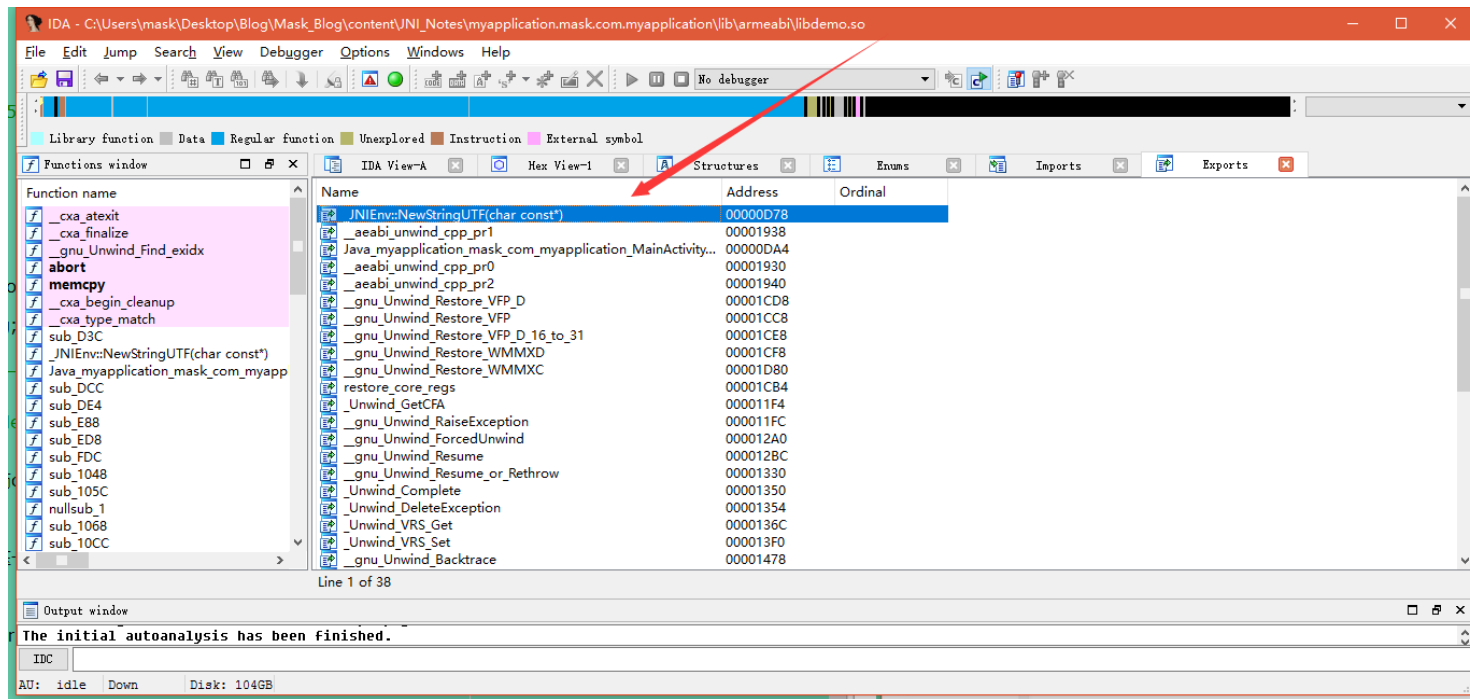
0x02 反汇编

那么可以把AS关了，把刚刚那么应用解压一下，你会发现有一个lib目录，这就是一般情况下库文件存放的地方



用IDA打开那个so文件(我一般喜欢打开armeabi下的)

这个就是我们编写的Native函数,跟进去



```
; ===== S U B R O U T I N E =====
.text:0000D78
.text:0000D78 ; Attributes: bp-based frame fpd=8
.text:0000D78
.text:0000D78 ; _DWORD __fastcall JNIEnv::NewStringUTF(JNIEnv *__hidden this, const char *)
.text:0000D78 WEAK _ZN7_JNIEnv12NewStringUTFEPKc
.text:0000D78 _ZN7_JNIEnv12NewStringUTFEPKc ; CODE XREF: Java_myapplication_mask_com_myapplication_MainActivity_FromNative+14 p
.text:0000D78
.text:0000D78 var_8 = -8
.text:0000D78 var_4 = -4
.text:0000D78
.text:0000D78 PUSH{R7,LR}
.text:0000D7A SUB SP, SP, #8
.text:0000D7C ADD R7, SP, #0
.text:0000D7E STR R0, [R7,#8+var_4]
.text:0000D80 STR R1, [R7,#8+var_8]
.text:0000D82 LDR R3, [R7,#8+var_4]
.text:0000D84 LDR R2, [R3]
.text:0000D86 MOVSR3, #0x29C
.text:0000D8A LDR R3, [R2,R3]
.text:0000D8C LDR R1, [R7,#8+var_4]
.text:0000D8E LDR R2, [R7,#8+var_8]
.text:0000D90 MOVSR0, R1
.text:0000D92 MOVSR1, R2
.text:0000D94 BLX R3
```

```
.text:00000D96 MOVSR3, R0
.text:00000D98 NOP
.text:00000D9A MOVSR0, R3
.text:00000D9C MOV SP, R7
.text:00000D9E ADD SP, SP, #8
.text:00000DA0 POP {R7,PC}
.text:00000DA0 ; End of function _JNIEnv::NewStringUTF(char const*)
```

一句一句分析：

导入函数

```
EXPORT Java_myapplication_mask_com_myapplication_MainActivity_FromNative
.text:00000DA4 Java_myapplication_mask_com_myapplication_MainActivity_FromNative
```

将R7和LR寄存器的值压入栈（什么？啥是寄存器？啥是内存栈？呐，这个都不知道的话，下面你忽略吧0.0），LR寄存器是用来保存下一条指令的

```
PUSH{R7,LR}
```

SP = SP -8

SP寄存器是用来保存栈顶元素的，而内存栈是自下到上，降序的，此处SP减去8，就是为了开辟出一段栈空间

```
SUB SP, SP, #8
```

R7 = SP + 0

此处相当于R7 = SP，将SP的值传给R7，为什么这样做呢？这其实是一种保护机制，因为SP是时刻指向栈顶的，可以看到，下面的一些操作都是以SP为基地址进行的，那么我们这里用R7来替代SP，，将SP的值保存在R7中，我们后边调用函数出栈时，在将R7的值还给SP，这样可以保证栈平衡

```
ADD R7, SP, #0
```

将R0寄存器的值赋给(R7+8-4)地址处，这是一个写操作

```
STR R0, [R7,#8+var_4]
```

将R1寄存器的值赋给(R7+8-8)地址处，这是一个写操作

```
STR R1, [R7,#8+var_8]
```

那么R0,R1寄存器的值是什么呢？在ARM中，前4个参数是用R0-R3来保存的（如果多余4个参数，剩下的则是用栈来操作）

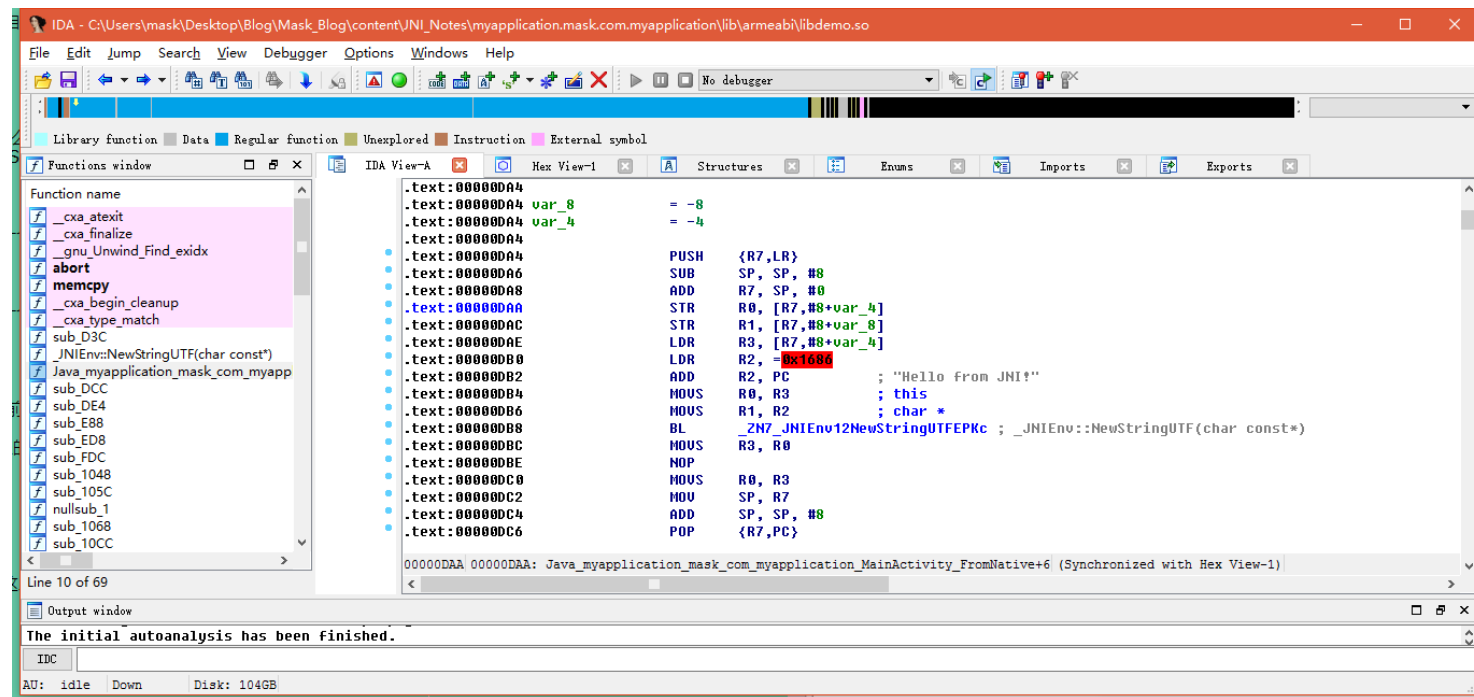
那么JNI函数的第一个参数是一个JNIEnv 类型的指针*env，第二个参数是一个jclass类型的参数clazz或者是一个jobject类型的object，呐，这个函数其实只有一个参数，那就是JNIEnv类型的指针*env,为啥没有第二个，因为压根就啥也没操作，这里就只是return了一个字符串而已。。。

将第一个参数保存到R3寄存器

```
LDR R3, [R7,#8+var_4]
```

将aHelloFromJni - 0xDB6的值赋给R2, 这里有个小tip, 你可以鼠标右键点击那个 "=", 他会自动帮你转为最终地址

```
LDR R2, =(aHelloFromJni - 0xDB6)
```



R2 = R2 + PC

PC寄存器为当前指令, 但是这里需要注意, 之前我也搞错过, ARM在执行命令时, 其实是分三步走的

- 1.取址
- 2.编译
- 3.执行

那么当我执行第一条命令时, 第二条指令在编译, 第三条指令在取址, 过程大概如下:

```
取址-----编译-----执行
               取址-----编译-----执行
                           取址-----编译-----执行
```

那么就好理解了, 此时PC的值0x00000DB6

那么R2此时的值为(aHelloFromJni - 0xDB6+0xDB6)处的值, 那么就是Hello from JNI! 后面的注释也有给出

```
ADD R2, PC
```

将R3的值赋给R0，并且会影响标志位

```
MOVS R0, R3
```

将R2的值赋给R1，并且会影响标志位

```
MOVS R1, R2
```

上面两步其实是为子函数做准备，将*env和Hello from JNI!进行传参，和之前说的一样，是用R0,R1保存前两个参数

BL进行函数跳转，根据注释可以知道，调用的是_JNIEnv::NewStringUTF(char const*)函数，有兴趣可以跟过去分析，过程都差不多

```
BL _ZN7_JNIEnv12NewStringUTFPKc ; _JNIEnv::NewStringUTF(char const*)
```

NOP为空操作，就是啥也不干

```
MOVS R3, R0
```

```
NOP
```

```
MOVS R0, R3
```

这里将R7的值赋给SP，和我们之前说的一样

```
MOV SP, R7
```

这里SP = SP+8，回收调之前开辟的空间

```
ADD SP, SP, #8
```

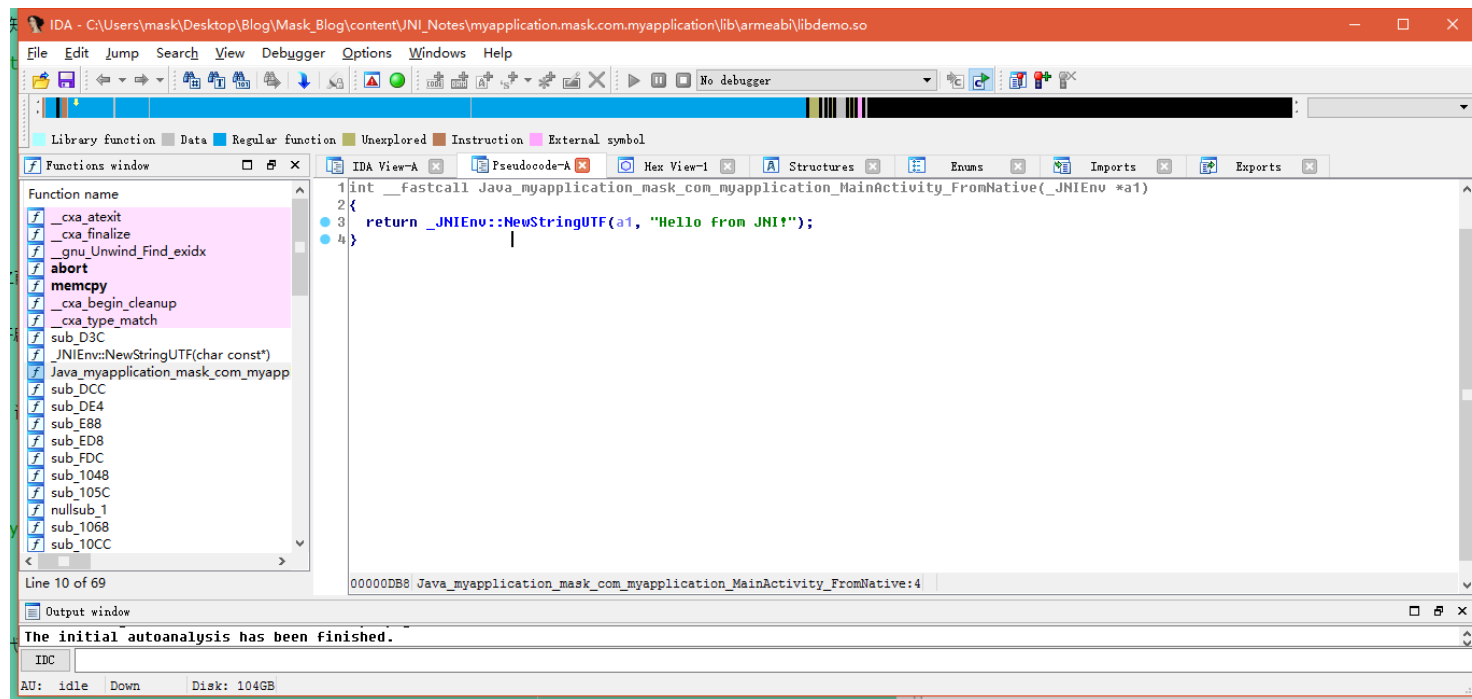
出栈，此时LR的值保存到PC中，说明结束上述操作，开始下一条指令

```
POP {R7,PC}
```

函数结束

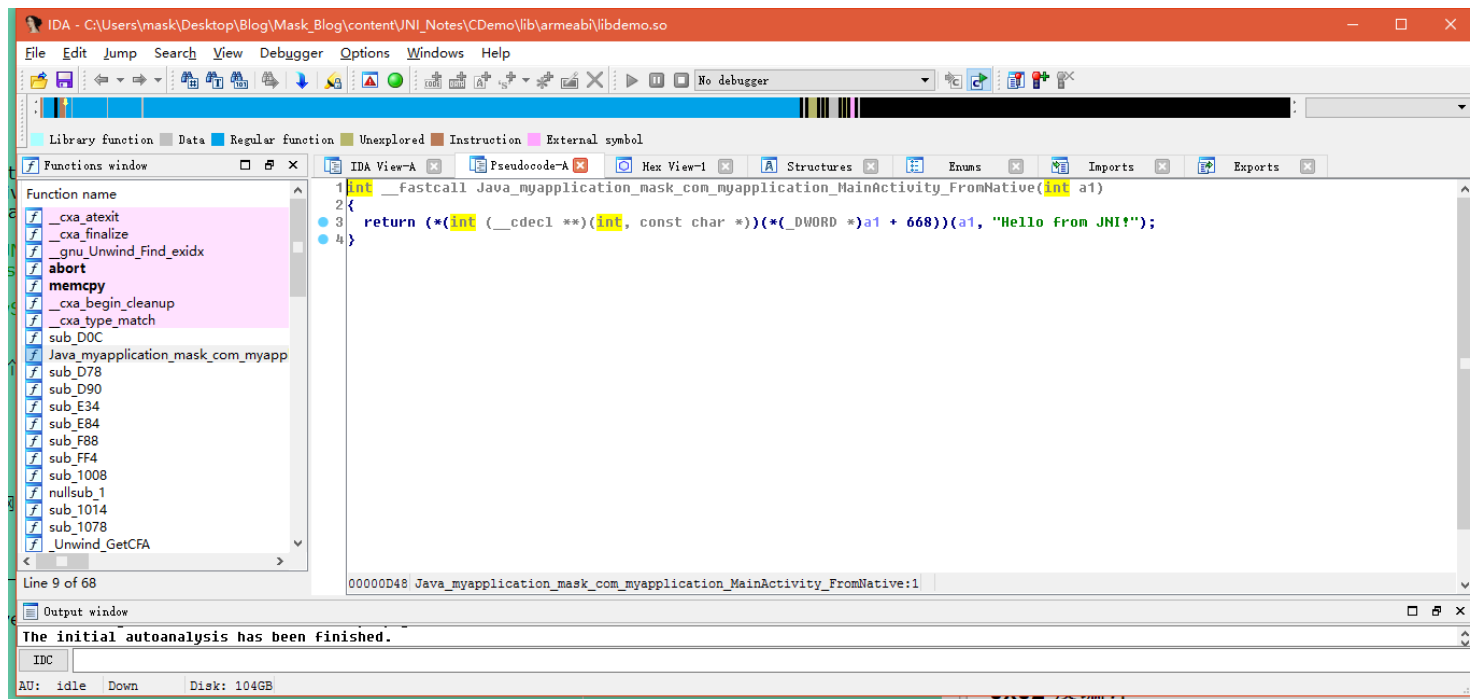
```
; End of function Java_myapplication_mask_com_myapplication_MainActivity_FromNative
```

汇编分析完了，我们再来看看伪代码，可以更好的理解刚刚的操作，F5大法好哇



哗擦，居然识别出来了。。。。。。还想演示一遍修复参数呢！！！！

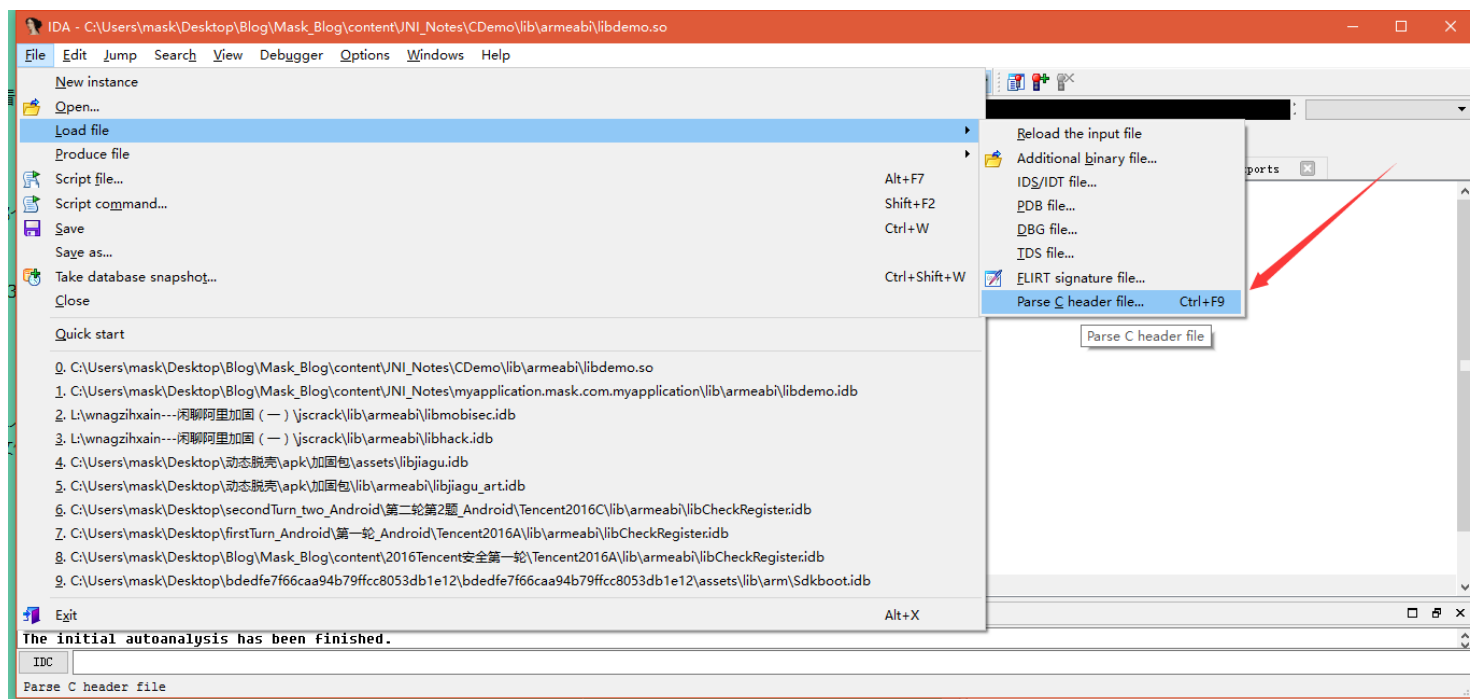
待我翻下刚刚C版本写的那个



呐，稳，没识别出来23333

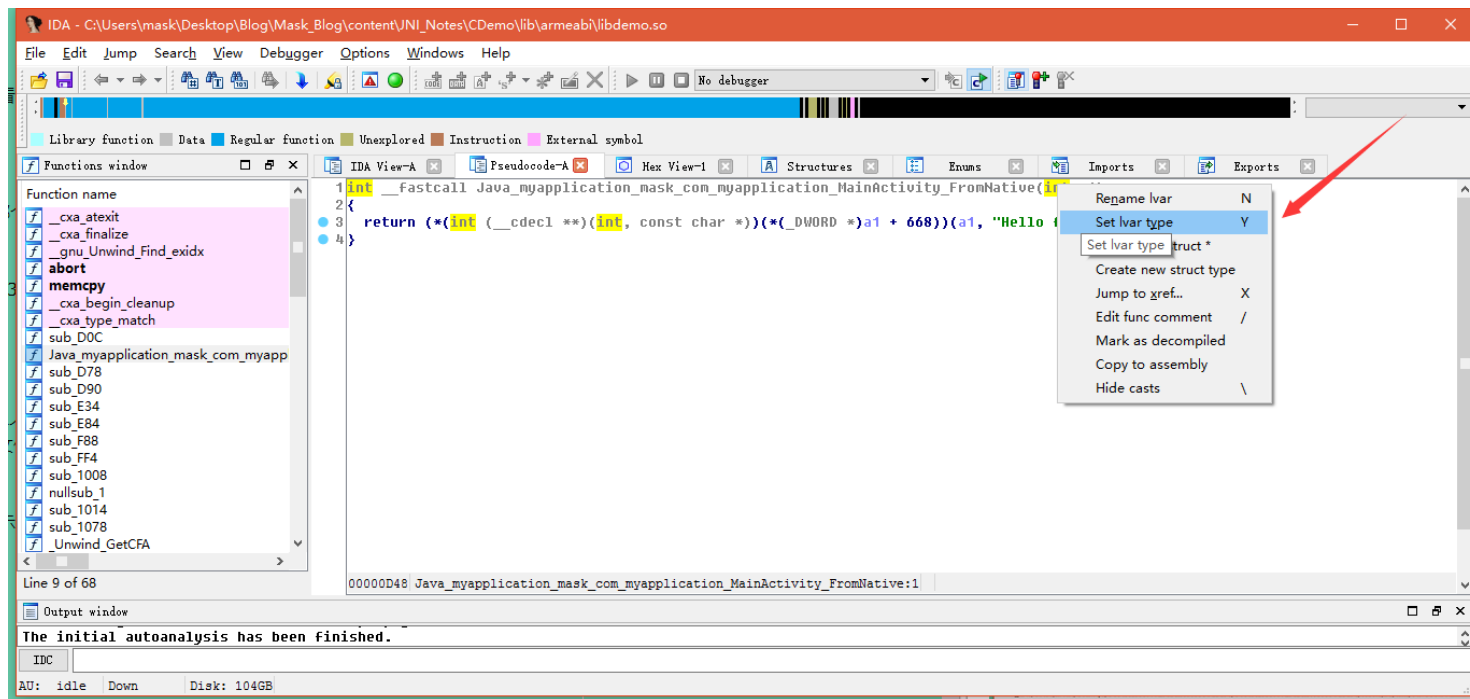
开始修复

之前说过，JNI函数的第一个参数一般默认是JNIEnv的结构体指针，那么这里我们要让IDA能够识别JNI结构体，我们需要导入一个JNI.h头文件（我的IDA好像自带），这个文件是啥，待会再说，这个IDA.h你可以去网上下载，其实NDK就自带这个

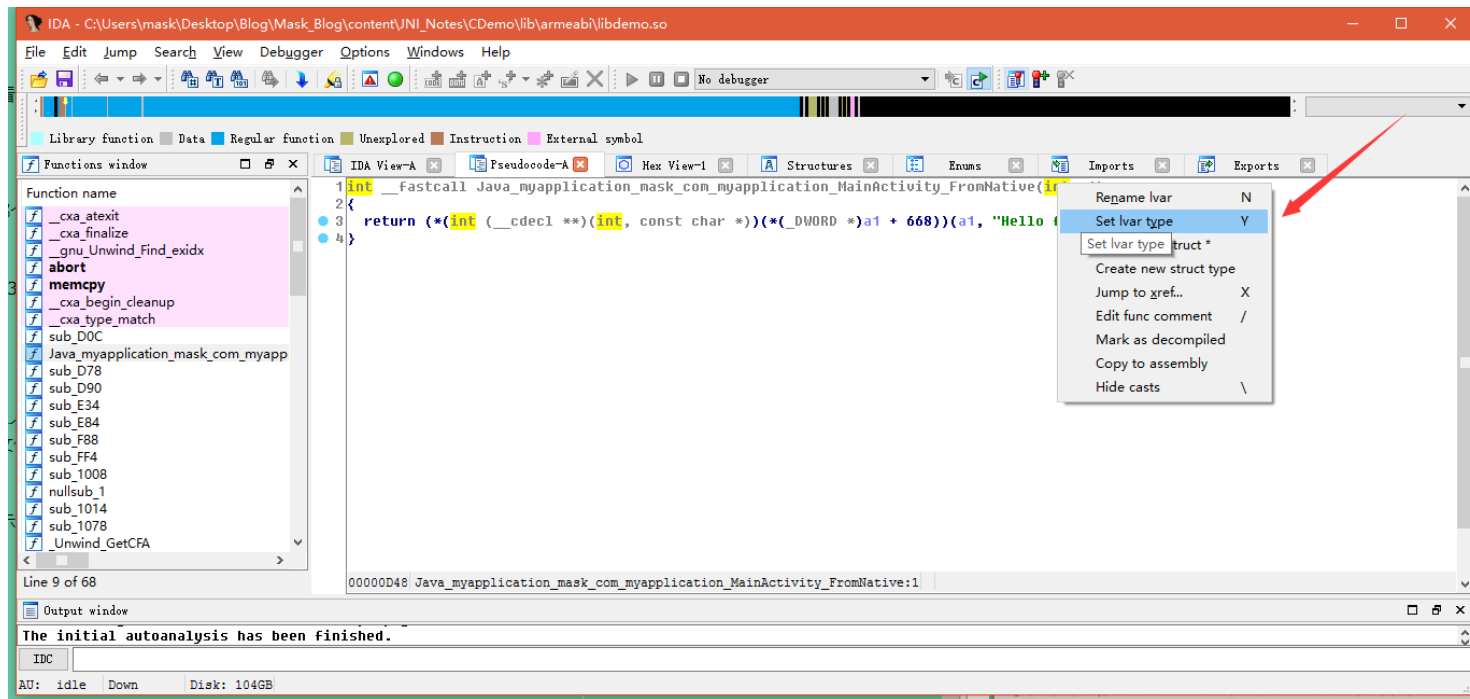


导入该文件，成功会有提示，提示啥我忘了，反正会告诉你导入成功了

然后



修复参数类型，输入JNIEnv*



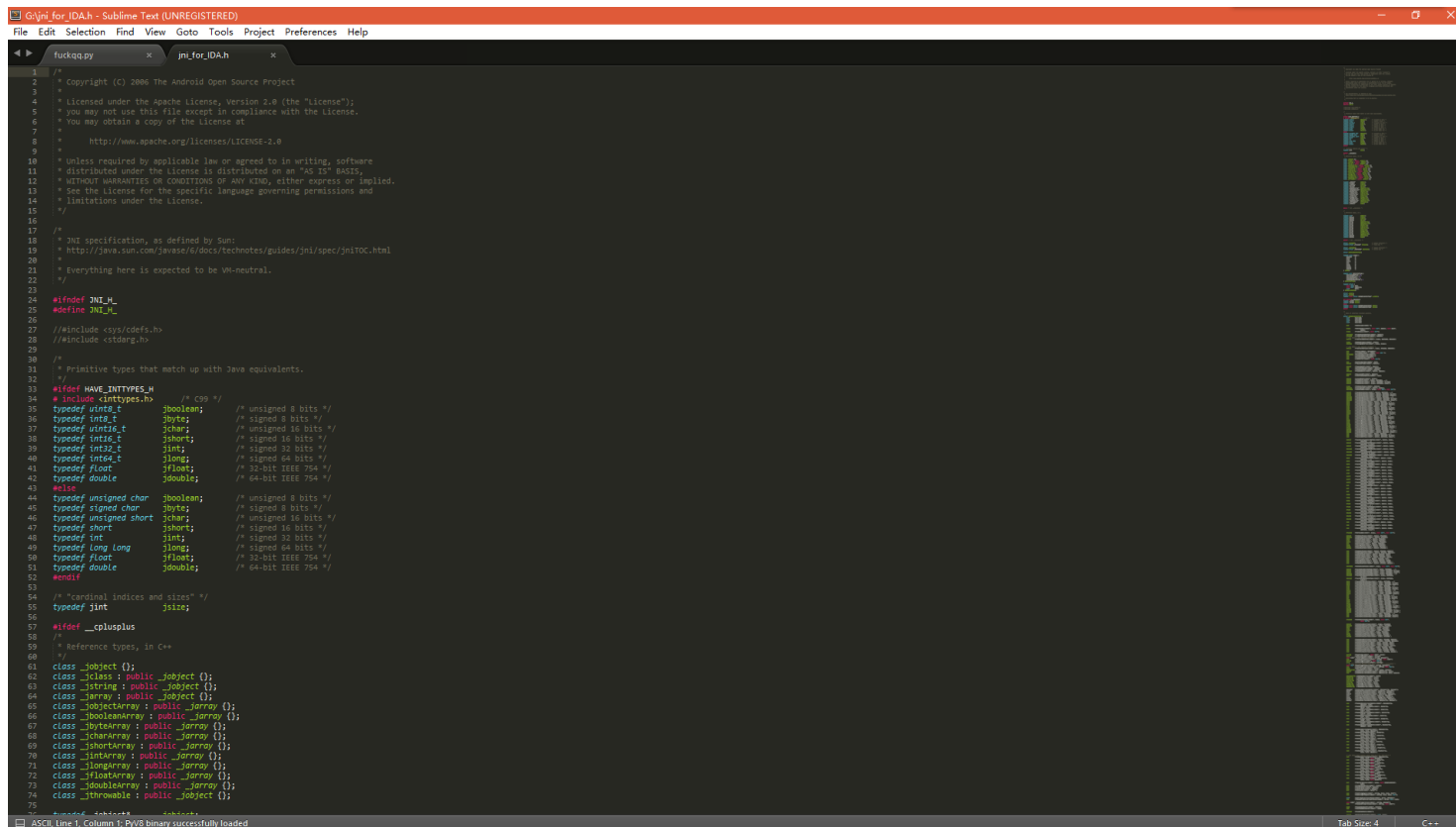
我这里函数都给我识别出来了。。。, 其实应该还有一步, 如果你的函数没有正确识别的话, 右键你的函数名, 会显示Force call type, 点击这个就能正确识别了

这里还有几个疑问没解决

- 1.JNIEnv* env这个指针到底指向哪?
- 2.JNI.h是干啥的?
- 3.第二个参数类型何时是jclass何时是jobject

0x03 JNI.h

先来看看这个JNI.h是啥



这里其实是一些数据类型转换，比如JAVA中的

```
#ifndef HAVE_INTTYPES_H
# include <inttypes.h> /* C99 */

typedef uint8_t jboolean; /* unsigned 8 bits */
typedef int8_t jbyte; /* signed 8 bits */
typedef uint16_t jchar; /* unsigned 16 bits */
typedef int16_t jshort; /* signed 16 bits */
typedef int32_t jint; /* signed 32 bits */
typedef int64_t jlong; /* signed 64 bits */
typedef float jfloat; /* 32-bit IEEE 754 */
typedef double jdouble; /* 64-bit IEEE 754 */
#else
typedef unsigned char jboolean; /* unsigned 8 bits */
typedef signed char jbyte; /* signed 8 bits */
typedef unsigned short jchar; /* unsigned 16 bits */
typedef short jshort; /* signed 16 bits */
typedef int jint; /* signed 32 bits */
typedef long long jlong; /* signed 64 bits */
typedef float jfloat; /* 32-bit IEEE 754 */
typedef double jdouble; /* 64-bit IEEE 754 */
#endif
```

这里是C和C++中的一些差别

```
#ifndef __cplusplus
/*
 * Reference types, in C++
 */
class _jobject {};
class _jclass : public _jobject {};
class _jstring : public _jobject {};
class _jarray : public _jobject {};
class _jobjectArray : public _jarray {};
class _jbooleanArray : public _jarray {};
class _jbyteArray : public _jarray {};
class _jcharArray : public _jarray {};
class _jshortArray : public _jarray {};
class _jintArray : public _jarray {};
class _jlongArray : public _jarray {};
class _jfloatArray : public _jarray {};
class _jdoubleArray : public _jarray {};
class _jthrowable : public _jobject {};

typedef _jobject*    jobject;
typedef _jclass*jclass;
typedef _jstring*    jstring;
typedef _jarray*jarray;
typedef _jobjectArray* jobjectArray;
typedef _jbooleanArray* jbooleanArray;
typedef _jbyteArray*jbyteArray;
typedef _jcharArray*jcharArray;
typedef _jshortArray* jshortArray;
typedef _jintArray* jintArray;
typedef _jlongArray*jlongArray;
typedef _jfloatArray* jfloatArray;
typedef _jdoubleArray* jdoubleArray;
typedef _jthrowable*jthrowable;
typedef _jobject*    jweak;

#else /* not __cplusplus */

/*
 * Reference types, in C.
 */
typedef void*    jobject;
typedef jobject jclass;
typedef jobject jstring;
typedef jobject jarray;
typedef jarray jobjectArray;
typedef jarray jbooleanArray;
```

```

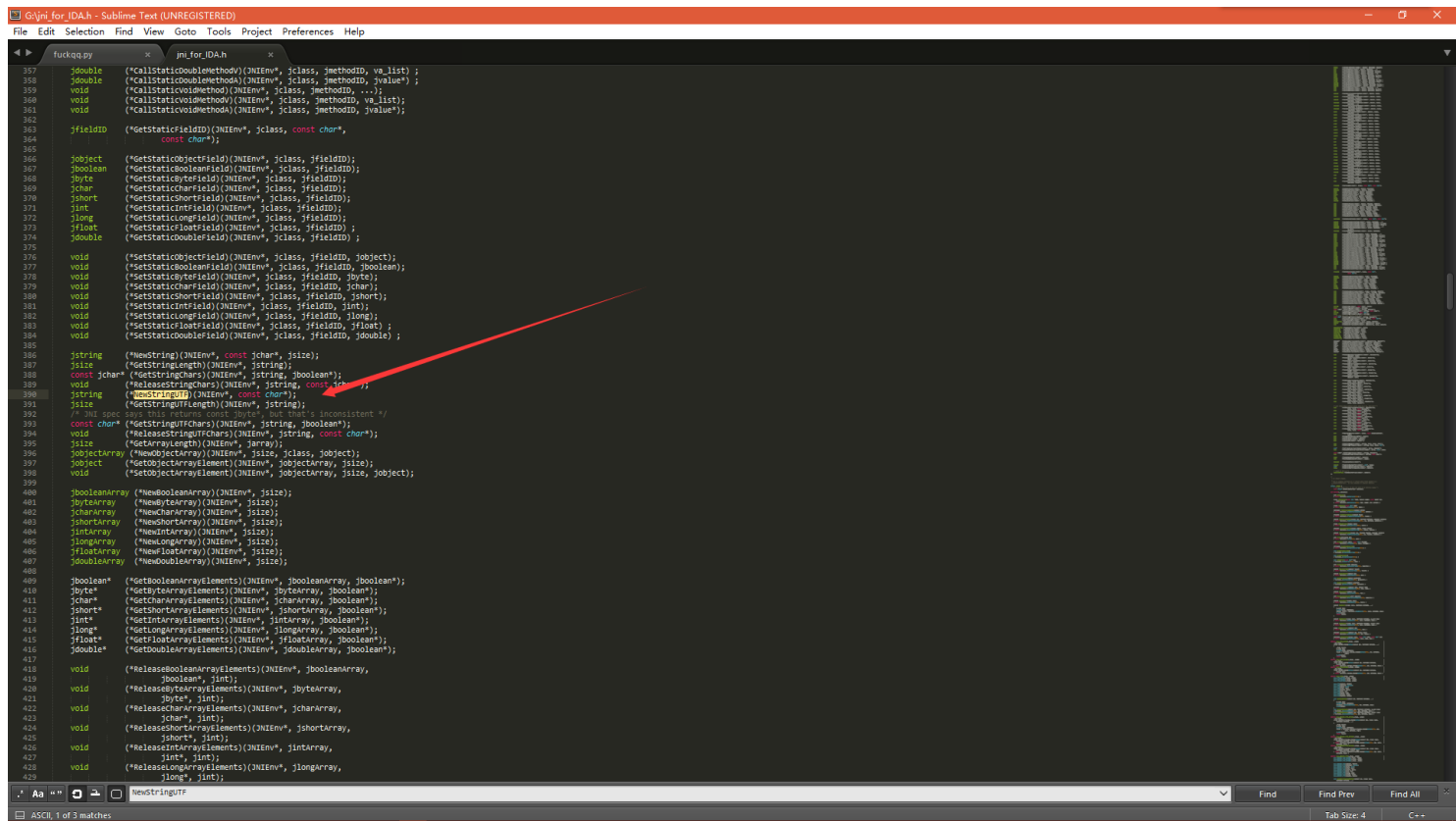
typedef jarray  jbyteArray;
typedef jarray  jcharArray;
typedef jarray  jshortArray;
typedef jarray  jintArray;
typedef jarray  jlongArray;
typedef jarray  jfloatArray;
typedef jarray  jdoubleArray;
typedef jobject jthrowable;
typedef jobject jweak;

```

简单看看就好，下面来看看关键

```

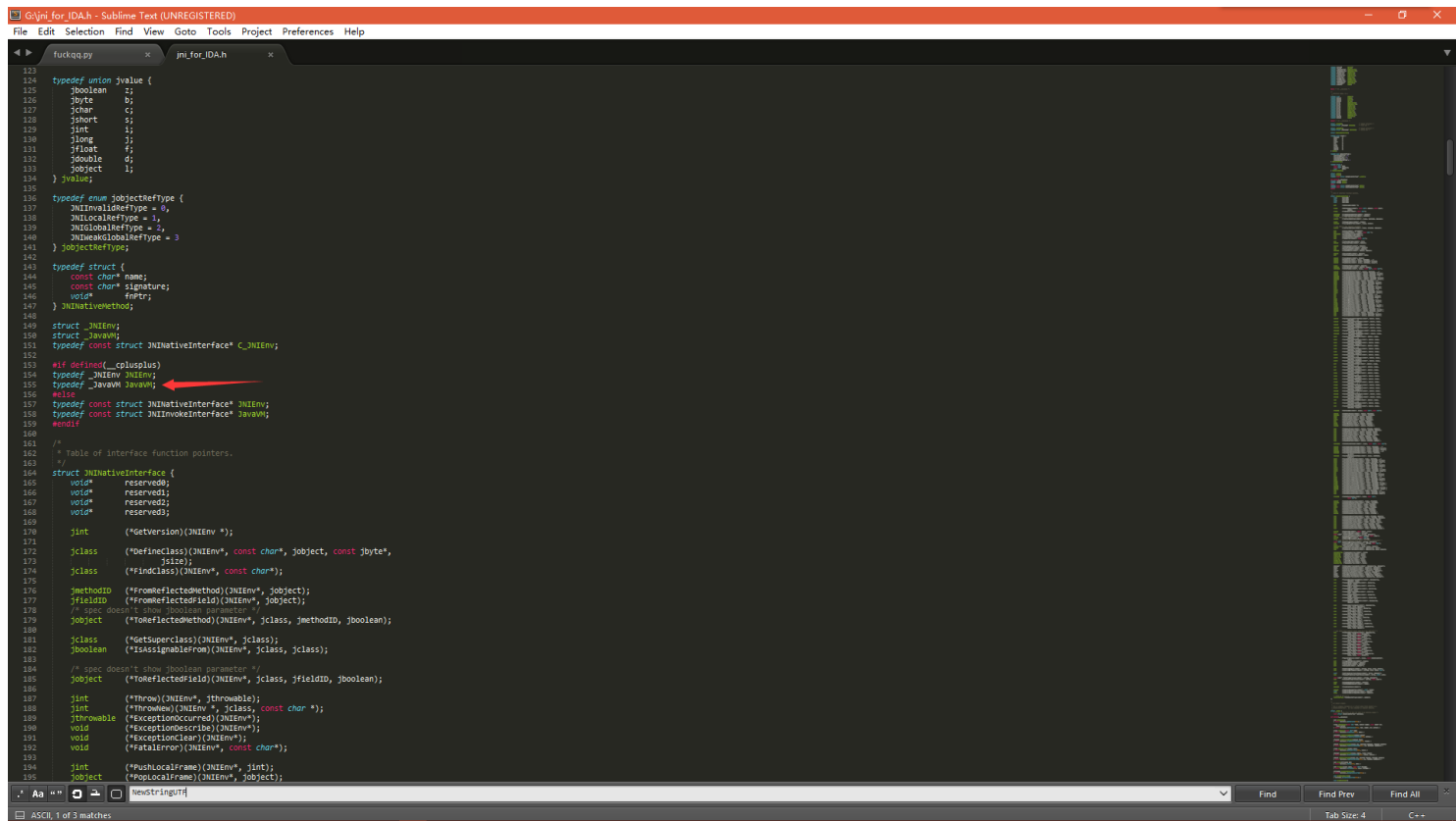
112 typedef jobject JNICALL jweak;
113
114 #endif /* not __cplusplus */
115
116 struct _jfieldID; /* opaque structure */
117 typedef struct _jfieldID* jfieldID; /* field ids */
118
119 struct _jmethodID; /* opaque structure */
120 typedef struct _jmethodID* jmethodID; /* method ids */
121
122 struct JNIInvokeInterface;
123
124 typedef union jvalue {
125     jboolean b;
126     jbyte b2;
127     jchar c;
128     jshort s;
129     jint i;
130     jlong j;
131     jfloat f;
132     jdouble d;
133     jobject l;
134 } jvalue;
135
136 typedef enum jobjectRefType {
137     JNIInvalidRefType = 0,
138     JNILocalRefType = 1,
139     JNIGlobalRefType = 2,
140     JNIWeakGlobalRefType = 3
141 } jobjectRefType;
142
143 typedef struct {
144     const char* name;
145     const char* signature;
146     void* fptr;
147 } JNINativeMethod;
148
149 struct _JNIEnv;
150 struct _JavaVM;
151 typedef const struct JNIInvokeInterface* C_JNIEnv;
152
153 #if defined(__cplusplus)
154 typedef _JNIEnv JNIEnv;
155 typedef _JavaVM JavaVM;
156 #else
157 typedef const struct JNIInvokeInterface* JNIEnv;
158 typedef const struct JNIInvokeInterface* JavaVM;
159 #endif
160
161 /*
162  * Table of interface function pointers.
163  */
164 struct JNIInvokeInterface {
165     void* reserved0;
166     void* reserved1;
167     void* reserved2;
168     void* reserved3;
169
170     jint (*GetVersion)(JNIEnv *);
171
172     jclass (*DefineClass)(JNIEnv*, const char*, jobject, const jbyte*,
173                          jint);
174     jclass (*FindClass)(JNIEnv*, const char*);
175
176     jmethodID (*FromReflectedMethod)(JNIEnv*, jobject);
177     jfieldID (*FromReflectedField)(JNIEnv*, jobject);
178     /* spec doesn't show boolean parameter */
179     jobject (*ToReflectedMethod)(JNIEnv*, jclass, jmethodID, jboolean);
180
181     jclass (*GetSuperclass)(JNIEnv*, jclass);
182     jboolean (*IsAssignableFrom)(JNIEnv*, jclass, jclass);
183
184     /* spec doesn't show boolean parameter */
185     jobject (*ToReflectedField)(JNIEnv*, jclass, jfieldID, jboolean);
186
187     /*
188      * (Throwable)GetEnv & throwables.
189      */
189 };
190
191 #endif
192
193 #endif
194
195 #endif
196
197 #endif
198
199 #endif
200
201 #endif
202
203 #endif
204
205 #endif
206
207 #endif
208
209 #endif
210
211 #endif
212
213 #endif
214
215 #endif
216
217 #endif
218
219 #endif
220
221 #endif
222
223 #endif
224
225 #endif
226
227 #endif
228
229 #endif
230
231 #endif
232
233 #endif
234
235 #endif
236
237 #endif
238
239 #endif
240
241 #endif
242
243 #endif
244
245 #endif
246
247 #endif
248
249 #endif
250
251 #endif
252
253 #endif
254
255 #endif
256
257 #endif
258
259 #endif
260
261 #endif
262
263 #endif
264
265 #endif
266
267 #endif
268
269 #endif
270
271 #endif
272
273 #endif
274
275 #endif
276
277 #endif
278
279 #endif
280
281 #endif
282
283 #endif
284
285 #endif
286
287 #endif
288
289 #endif
290
291 #endif
292
293 #endif
294
295 #endif
296
297 #endif
298
299 #endif
300
301 #endif
302
303 #endif
304
305 #endif
306
307 #endif
308
309 #endif
310
311 #endif
312
313 #endif
314
315 #endif
316
317 #endif
318
319 #endif
320
321 #endif
322
323 #endif
324
325 #endif
326
327 #endif
328
329 #endif
330
331 #endif
332
333 #endif
334
335 #endif
336
337 #endif
338
339 #endif
340
341 #endif
342
343 #endif
344
345 #endif
346
347 #endif
348
349 #endif
350
351 #endif
352
353 #endif
354
355 #endif
356
357 #endif
358
359 #endif
360
361 #endif
362
363 #endif
364
365 #endif
366
367 #endif
368
369 #endif
370
371 #endif
372
373 #endif
374
375 #endif
376
377 #endif
378
379 #endif
380
381 #endif
382
383 #endif
384
385 #endif
386
387 #endif
388
389 #endif
390
391 #endif
392
393 #endif
394
395 #endif
396
397 #endif
398
399 #endif
400
401 #endif
402
403 #endif
404
405 #endif
406
407 #endif
408
409 #endif
410
411 #endif
412
413 #endif
414
415 #endif
416
417 #endif
418
419 #endif
420
421 #endif
422
423 #endif
424
425 #endif
426
427 #endif
428
429 #endif
430
431 #endif
432
433 #endif
434
435 #endif
436
437 #endif
438
439 #endif
440
441 #endif
442
443 #endif
444
445 #endif
446
447 #endif
448
449 #endif
450
451 #endif
452
453 #endif
454
455 #endif
456
457 #endif
458
459 #endif
460
461 #endif
462
463 #endif
464
465 #endif
466
467 #endif
468
469 #endif
470
471 #endif
472
473 #endif
474
475 #endif
476
477 #endif
478
479 #endif
480
481 #endif
482
483 #endif
484
485 #endif
486
487 #endif
488
489 #endif
490
491 #endif
492
493 #endif
494
495 #endif
496
497 #endif
498
499 #endif
500
501 #endif
502
503 #endif
504
505 #endif
506
507 #endif
508
509 #endif
510
511 #endif
512
513 #endif
514
515 #endif
516
517 #endif
518
519 #endif
520
521 #endif
522
523 #endif
524
525 #endif
526
527 #endif
528
529 #endif
530
531 #endif
532
533 #endif
534
535 #endif
536
537 #endif
538
539 #endif
540
541 #endif
542
543 #endif
544
545 #endif
546
547 #endif
548
549 #endif
550
551 #endif
552
553 #endif
554
555 #endif
556
557 #endif
558
559 #endif
560
561 #endif
562
563 #endif
564
565 #endif
566
567 #endif
568
569 #endif
570
571 #endif
572
573 #endif
574
575 #endif
576
577 #endif
578
579 #endif
580
581 #endif
582
583 #endif
584
585 #endif
586
587 #endif
588
589 #endif
590
591 #endif
592
593 #endif
594
595 #endif
596
597 #endif
598
599 #endif
600
601 #endif
602
603 #endif
604
605 #endif
606
607 #endif
608
609 #endif
610
611 #endif
612
613 #endif
614
615 #endif
616
617 #endif
618
619 #endif
620
621 #endif
622
623 #endif
624
625 #endif
626
627 #endif
628
629 #endif
630
631 #endif
632
633 #endif
634
635 #endif
636
637 #endif
638
639 #endif
640
641 #endif
642
643 #endif
644
645 #endif
646
647 #endif
648
649 #endif
650
651 #endif
652
653 #endif
654
655 #endif
656
657 #endif
658
659 #endif
660
661 #endif
662
663 #endif
664
665 #endif
666
667 #endif
668
669 #endif
670
671 #endif
672
673 #endif
674
675 #endif
676
677 #endif
678
679 #endif
680
681 #endif
682
683 #endif
684
685 #endif
686
687 #endif
688
689 #endif
690
691 #endif
692
693 #endif
694
695 #endif
696
697 #endif
698
699 #endif
700
701 #endif
702
703 #endif
704
705 #endif
706
707 #endif
708
709 #endif
710
711 #endif
712
713 #endif
714
715 #endif
716
717 #endif
718
719 #endif
720
721 #endif
722
723 #endif
724
725 #endif
726
727 #endif
728
729 #endif
730
731 #endif
732
733 #endif
734
735 #endif
736
737 #endif
738
739 #endif
740
741 #endif
742
743 #endif
744
745 #endif
746
747 #endif
748
749 #endif
750
751 #endif
752
753 #endif
754
755 #endif
756
757 #endif
758
759 #endif
760
761 #endif
762
763 #endif
764
765 #endif
766
767 #endif
768
769 #endif
770
771 #endif
772
773 #endif
774
775 #endif
776
777 #endif
778
779 #endif
780
781 #endif
782
783 #endif
784
785 #endif
786
787 #endif
788
789 #endif
790
791 #endif
792
793 #endif
794
795 #endif
796
797 #endif
798
799 #endif
800
801 #endif
802
803 #endif
804
805 #endif
806
807 #endif
808
809 #endif
810
811 #endif
812
813 #endif
814
815 #endif
816
817 #endif
818
819 #endif
820
821 #endif
822
823 #endif
824
825 #endif
826
827 #endif
828
829 #endif
830
831 #endif
832
833 #endif
834
835 #endif
836
837 #endif
838
839 #endif
840
841 #endif
842
843 #endif
844
845 #endif
846
847 #endif
848
849 #endif
850
851 #endif
852
853 #endif
854
855 #endif
856
857 #endif
858
859 #endif
860
861 #endif
862
863 #endif
864
865 #endif
866
867 #endif
868
869 #endif
870
871 #endif
872
873 #endif
874
875 #endif
876
877 #endif
878
879 #endif
880
881 #endif
882
883 #endif
884
885 #endif
886
887 #endif
888
889 #endif
890
891 #endif
892
893 #endif
894
895 #endif
896
897 #endif
898
899 #endif
900
901 #endif
902
903 #endif
904
905 #endif
906
907 #endif
908
909 #endif
910
911 #endif
912
913 #endif
914
915 #endif
916
917 #endif
918
919 #endif
920
921 #endif
922
923 #endif
924
925 #endif
926
927 #endif
928
929 #endif
930
931 #endif
932
933 #endif
934
935 #endif
936
937 #endif
938
939 #endif
940
941 #endif
942
943 #endif
944
945 #endif
946
947 #endif
948
949 #endif
950
951 #endif
952
953 #endif
954
955 #endif
956
957 #endif
958
959 #endif
960
961 #endif
962
963 #endif
964
965 #endif
966
967 #endif
968
969 #endif
970
971 #endif
972
973 #endif
974
975 #endif
976
977 #endif
978
979 #endif
980
981 #endif
982
983 #endif
984
985 #endif
986
987 #endif
988
989 #endif
990
991 #endif
992
993 #endif
994
995 #endif
996
997 #endif
998
999 #endif
1000
1001 #endif
1002
1003 #endif
1004
1005 #endif
1006
1007 #endif
1008
1009 #endif
1010
1011 #endif
1012
1013 #endif
1014
1015 #endif
1016
1017 #endif
1018
1019 #endif
1020
1021 #endif
1022
1023 #endif
1024
1025 #endif
1026
1027 #endif
1028
1029 #endif
1030
1031 #endif
1032
1033 #endif
1034
1035 #endif
1036
1037 #endif
1038
1039 #endif
1040
1041 #endif
1042
1043 #endif
1044
1045 #endif
1046
1047 #endif
1048
1049 #endif
1050
1051 #endif
1052
1053 #endif
1054
1055 #endif
1056
1057 #endif
1058
1059 #endif
1060
1061 #endif
1062
1063 #endif
1064
1065 #endif
1066
1067 #endif
1068
1069 #endif
1070
1071 #endif
1072
1073 #endif
1074
1075 #endif
1076
1077 #endif
1078
1079 #endif
1080
1081 #endif
1082
1083 #endif
1084
1085 #endif
1086
1087 #endif
1088
1089 #endif
1090
1091 #endif
1092
1093 #endif
1094
1095 #endif
1096
1097 #endif
1098
1099 #endif
1100
1101 #endif
1102
1103 #endif
1104
1105 #endif
1106
1107 #endif
1108
1109 #endif
1110
1111 #endif
1112
1113 #endif
1114
1115 #endif
1116
1117 #endif
1118
1119 #endif
1120
1121 #endif
1122
1123 #endif
1124
1125 #endif
1126
1127 #endif
1128
1129 #endif
1130
1131 #endif
1132
1133 #endif
1134
1135 #endif
1136
1137 #endif
1138
1139 #endif
1140
1141 #endif
1142
1143 #endif
1144
1145 #endif
1146
1147 #endif
1148
1149 #endif
1150
1151 #endif
1152
1153 #endif
1154
1155 #endif
1156
1157 #endif
1158
1159 #endif
1160
1161 #endif
1162
1163 #endif
1164
1165 #endif
1166
1167 #endif
1168
1169 #endif
1170
1171 #endif
1172
1173 #endif
1174
1175 #endif
1176
1177 #endif
1178
1179 #endif
1180
1181 #endif
1182
1183 #endif
1184
1185 #endif
1186
1187 #endif
1188
1189 #endif
1190
1191 #endif
1192
1193 #endif
1194
1195 #endif
1196
1197 #endif
1198
1199 #endif
1200
1201 #endif
1202
1203 #endif
1204
1205 #endif
1206
1207 #endif
1208
1209 #endif
1210
1211 #endif
1212
1213 #endif
1214
1215 #endif
1216
1217 #endif
1218
1219 #endif
1220
1221 #endif
1222
1223 #endif
1224
1225 #endif
1226
1227 #endif
1228
1229 #endif
1230
1231 #endif
1232
1233 #endif
1234
1235 #endif
1236
1237 #endif
1238
1239 #endif
1240
1241 #endif
1242
1243 #endif
1244
1245 #endif
1246
1247 #endif
1248
1249 #endif
1250
1251 #endif
1252
1253 #endif
1254
1255 #endif
1256
1257 #endif
1258
1259 #endif
1260
1261 #endif
1262
1263 #endif
1264
1265 #endif
1266
1267 #endif
1268
1269 #endif
1270
1271 #endif
1272
1273 #endif
1274
1275 #endif
1276
1277 #endif
1278
1279 #endif
1280
1281 #endif
1282
1283 #endif
1284
1285 #endif
1286
1287 #endif
1288
1289 #endif
1290
1291 #endif
1292
1293 #endif
1294
1295 #endif
1296
1297 #endif
1298
1299 #endif
1300
1301 #endif
1302
1303 #endif
1304
1305 #endif
1306
1307 #endif
1308
1309 #endif
1310
1311 #endif
1312
1313 #endif
1314
1315 #endif
1316
1317 #endif
1318
1319 #endif
1320
1321 #endif
1322
1323 #endif
1324
1325 #endif
1326
1327 #endif
1328
1329 #endif
1330
1331 #endif
1332
1333 #endif
1334
1335 #endif
1336
1337 #endif
1338
1339 #endif
1340
1341 #endif
1342
1343 #endif
1344
1345 #endif
1346
1347 #endif
1348
1349 #endif
1350
1351 #endif
1352
1353 #endif
1354
1355 #endif
1356
1357 #endif
1358
1359 #endif
1360
1361 #endif
1362
1363 #endif
1364
1365 #endif
1366
1367 #endif
1368
1369 #endif
1370
1371 #endif
1372
1373 #endif
1374
1375 #endif
1376
1377 #endif
1378
1379 #endif
1380
1381 #endif
1382
1383 #endif
1384
1385 #endif
1386
1387 #endif
1388
1389 #endif
1390
1391 #endif
1392
1393 #endif
1394
1395 #endif
1396
1397 #endif
1398
1399 #endif
1400
1401 #endif
1402
1403 #endif
1404
1405 #endif
1406
1407 #endif
1408
1409 #endif
1410
1411 #endif
1412
1413 #endif
1414
1415 #endif
1416
1417 #endif
1418
1419 #endif
1420
1421 #endif
1422
1423 #endif
1424
1425 #endif
1426
1427 #endif
1428
1429 #endif
1430
1431 #endif
1432
1433 #endif
1434
1435 #endif
1436
1437 #endif
1438
1439 #endif
1440
1441 #endif
1442
1443 #endif
1444
1445 #endif
1446
1447 #endif
1448
1449 #endif
1450
1451 #endif
1452
1453 #endif
1454
1455 #endif
1456
1457 #endif
1458
1459 #endif
1460
1461 #endif
1462
1463 #endif
1464
1465 #endif
1466
1467 #endif
1468
1469 #endif
1470
1471 #endif
1472
1473 #endif
1474
1475 #endif
1476
1477 #endif
1478
1479 #endif
1480
1481 #endif
1482
1483 #endif
1484
1485 #endif
1486
1487 #endif
1488
1489 #endif
1490
1491 #endif
1492
1493 #endif
1494
1495 #endif
1496
1497 #endif
1498
1499 #endif
1500
1501 #endif
1502
1503 #endif
1504
1505 #endif
1506
1507 #endif
1508
1509 #endif
1510
1511 #endif
1512
1513 #endif
1514
1515 #endif
1516
1517 #endif
1518
1519 #endif
1520
1521 #endif
1522
1523 #endif
1524
1525 #endif
1526
1527 #endif
1528
1529 #endif
1530
1531 #endif
1532
1533 #endif
1534
1535 #endif
1536
1537 #endif
1538
1539 #endif
1540
1541 #endif
1542
1543 #endif
1544
1545 #endif
1546
1547 #endif
1548
1549 #endif
1550
1551 #endif
1552
1553 #endif
1554
1555 #endif
1556
1557 #endif
1558
1559 #endif
1560
1561 #endif
1562
1563 #endif
1564
1565 #endif
1566
1567 #endif
1568
1569 #endif
1570
1571 #endif
1572
1573 #endif
1574
1575 #endif
1576
1577 #endif
1578
1579 #endif
1580
1581 #endif
1582
1583 #endif
1584
1585 #endif
1586
1587 #endif
1588
1589 #endif
1590
1591 #endif
1592
1593 #endif
1594
1595 #endif
1596
1597 #endif
1598
1599 #endif
1600
1601 #endif
1602
1603 #endif
1604
1605 #endif
1606
1607 #endif
1608
1609 #endif
1610
1611 #endif
1612
1613 #endif
1614
1615 #endif
1616
1617 #endif
1618
1619 #endif
1620
1621 #endif
1622
1623 #endif
1624
1625 #endif
1626
1627 #endif
1628
1629 #endif
1630
1631 #endif
1632
1633 #endif
1634
1635 #endif
1636
1637 #endif
1638
1639 #endif
1640
1641 #endif
1642
1643 #endif
1644
1645 #endif
1646
1647 #endif
1648
1649 #endif
1650
1651 #endif
1652
1653 #endif
1654
1655 #endif
1656
1657 #endif
1658
1659 #endif
1660
1661 #endif
1662
1663 #endif
1664
1665 #endif
1666
1667 #endif
1668
1669 #endif
1670
1671 #endif
1672
1673 #endif
1674
1675 #endif
1676
1677 #endif
1678
1679 #endif
1680
1681 #endif
1682
1683 #endif
1684
1685 #endif
1686
1687 #endif
1688
1689 #endif
1690
1691 #endif
1692
1693 #endif
1694
1695 #endif
1696
1697 #endif
1698
1699 #endif
1700
1701 #endif
1702
1703 #endif
1704
1705 #endif
1706
1707 #endif
1708
1709 #endif
1710
1711 #endif
1712
1713 #endif
1714
1715 #endif
1716
1717 #endif
1718
1719 #endif
1720
1721 #endif
1722
1723 #endif
1724
1725 #endif
1726
1727 #endif
1728
1729 #endif
1730
1731 #endif
1732
1733 #endif
1734
1735 #endif
1736
1737 #endif
1738
1739 #endif
1740
1741 #endif
1742
1743 #endif
1744
1745 #endif
1746
1747 #endif
1748
1749 #endif
1750
1751 #endif
1752
1753 #endif
1754
1755 #endif
1756
1757 #endif
1758
1759 #endif
1760
1761 #endif
1762
1763 #endif
1764
1765 #endif
1766
1767 #endif
1768
1769 #endif
1770
1771 #endif
1772
1773 #endif
1774
1775 #endif
1776
1777 #endif
1778
1779 #endif
1780
1781 #endif
1782
1783 #endif
1784
1785 #endif
1786
1787 #endif
1788
1789 #endif
1790
1791 #endif
1792
1793 #endif
1794
1795 #endif
1796
1797 #endif
1798
1799 #endif
1800
1801 #endif
1802
1803 #endif
1804
1805 #endif
1806
1807 #endif
1808
1809 #endif
1810
1811 #endif
1812
1813 #endif
1814
1815 #endif
1816
1817 #endif
1818
1819 #endif
1820
1821 #endif
1822
1823 #endif
1824
1825 #endif
1826
1827 #endif
1828
1829 #endif
1830
1831 #endif
1832
1833 #endif
1834
1835 #endif
1836
1837 #endif
1838
1839 #endif
1840
1841 #endif
1842
1843 #endif
1844
1845 #endif
1846
1847 #endif
1848
1849 #endif
1850
1851 #endif
1852
1853 #endif
1854
1855 #endif
1856
1857 #endif
1858
1859 #endif
1860
1861 #endif
1862
1863 #endif
1864
1865 #endif
1866
1867 #endif
1868
1869 #endif
1870
1871 #endif
1872
1873 #endif
1874
1875 #endif
1876
1877 #endif
1878
1879 #endif
1880
1881 #endif
1882
1883 #endif
1884
1885 #endif
1886
1887 #endif
1888
1889 #endif
1890
1891 #endif
1892
1893 #endif
1894
1895 #endif
1896
1897 #endif
1898
1899 #endif
1900
1901 #endif
1902
1903 #endif
1904
1905 #endif
1906
1907 #endif
1908
1909 #endif
1910
1911 #endif
1912
1913 #endif
1914
1915 #endif
1916
1917 #endif
1918
1919 #endif
1920
1921 #endif
1922
1923 #endif
1924
1925 #endif
1926
1927 #endif
1928
1929 #endif
1930
1931 #endif
1932
1933 #endif
1934
1935 #endif
1936
1937 #endif
1938
1939 #endif
1940
1941 #endif
1942
1943 #endif
1944
1945 #endif
1946
1947 #endif
1948
1949 #endif
1950
1951 #endif
1952
1953 #endif
1954
1955 #endif
1956
1957 #endif
1958
1959 #endif
1960
1961 #endif
1962
1963 #endif
1964
1965 #endif
1966
1967 #endif
1968
1969 #endif
1970
1971 #endif
1972
1973 #endif
1974
1975 #endif
1976
1977 #endif
1978
1979 #endif
1980
1981 #endif
1982
1983 #endif
1984
1985 #endif
1986
1987 #endif
1988
1989 #endif
1990
1991 #endif
1992
1993 #endif
1994
1995 #endif
1996
1997 #endif
1998
1999 #endif
2000
2001 #endif
2002
2003 #endif
2004
2005 #endif
2006
2007 #endif
2008
2009 #endif
2010
2011 #endif
2012
2013 #endif
2014
2015 #endif
2016
2017 #endif
2018
2019 #endif
2020
2021 #endif
2022
2023 #endif
2024
2025 #endif
2026
2027 #endif
2028
2029 #endif
2030
2031 #endif
2032
2033 #endif
2034
2035 #endif
2036
2037 #endif
2038
2039 #endif
2040
2041 #endif
2042
2043 #endif
2044
2045 #endif
2046
2047 #endif
2048
2049 #endif
2050
2051 #endif
2052
2053 #endif
2054
2055 #endif
2056
2057 #endif
2058
2059 #endif
2060
2061 #endif
2062
2063 #endif
2064
2065 #endif
2066
2067 #endif
2068
2069 #endif
2070
2071 #endif
2072
2073 #endif
2074
2075 #endif
2076
2077 #endif
2078
2079 #endif
2080
2081 #endif
2082
2083 #endif
2084
2085 #endif
2086
2087 #endif
2088
2089 #endif
2090
2091 #endif
2092
2093 #endif
2094
2095 #endif
2096
2097 #endif
2098
2099 #endif
2100
2101 #endif
2102
2103 #endif
2104
2105 #endif
2106
2107 #endif
2108
2109 #endif
2110
2111 #endif
2112
2113 #endif
2114
2115 #endif
2116
2117 #endif
2118
2119 #endif
2120
2121 #endif
2122
2123 #endif
2124
2125 #endif
2126
2127 #endif
2128
2129 #endif
2130
2131 #endif
2132
2133 #endif
2134
2135 #endif
2136
2137 #endif
2138
2139 #endif
2140
2141 #endif
2142
2143 #endif
2144
2145 #endif
2146
2147 #endif
2148
2149 #endif
2150
2151 #endif
2152
2153 #endif
2154
2155 #endif
2156
2157 #endif
2158
2159 #endif
2160
2161 #endif
2162
2163 #endif
2164
2165 #endif
2166
2167 #endif
2168
2169 #endif
2170
2171 #endif
2172
2173 #endif
2174
2175 #endif
2176
2177 #endif
2178
2179 #endif
2180
2181 #endif
2182
2183 #endif
2184
2185 #endif
2186
2187 #endif
2188
2189 #endif
2190
2191 #endif
2192
2193 #endif
2194
2195 #endif
2196
2197 #endif
2198
2199 #endif
2200
2201 #endif
2202
2203 #endif
2204
2205 #endif
2206
2207 #endif
2208
2209 #endif
2210
2211 #endif
2212
2213 #endif
2214
2215 #endif
2216
2217 #endif
2218
2219 #endif
2220
2221 #endif
22
```



```
1377 double (*CallStaticDoubleMethod)(JNIEnv*, jclass, methodID, va_list);
1378 double (*CallStaticDoubleMethod)(JNIEnv*, jclass, methodID, jvalue);
1379 void (*CallStaticVoidMethod)(JNIEnv*, jclass, methodID, ...);
1380 void (*CallStaticVoidMethod)(JNIEnv*, jclass, methodID, va_list);
1381 void (*CallStaticVoidMethod)(JNIEnv*, jclass, methodID, jvalue);
1382
1383 jfieldID (*GetStaticFieldID)(JNIEnv*, jclass, const char*,
1384 const char*);
1385
1386 jobject (*GetStaticObjectField)(JNIEnv*, jclass, jfieldID);
1387 boolean (*GetStaticBooleanField)(JNIEnv*, jclass, jfieldID);
1388 jbyte (*GetStaticByteField)(JNIEnv*, jclass, jfieldID);
1389 jchar (*GetStaticCharField)(JNIEnv*, jclass, jfieldID);
1390 jshort (*GetStaticShortField)(JNIEnv*, jclass, jfieldID);
1391 jint (*GetStaticIntField)(JNIEnv*, jclass, jfieldID);
1392 jlong (*GetStaticLongField)(JNIEnv*, jclass, jfieldID);
1393 jfloat (*GetStaticFloatField)(JNIEnv*, jclass, jfieldID);
1394 jdouble (*GetStaticDoubleField)(JNIEnv*, jclass, jfieldID);
1395
1396 void (*SetStaticObjectField)(JNIEnv*, jclass, jfieldID, jobject);
1397 void (*SetStaticBooleanField)(JNIEnv*, jclass, jfieldID, boolean);
1398 void (*SetStaticByteField)(JNIEnv*, jclass, jfieldID, jbyte);
1399 void (*SetStaticCharField)(JNIEnv*, jclass, jfieldID, jchar);
1400 void (*SetStaticShortField)(JNIEnv*, jclass, jfieldID, jshort);
1401 void (*SetStaticIntField)(JNIEnv*, jclass, jfieldID, jint);
1402 void (*SetStaticLongField)(JNIEnv*, jclass, jfieldID, jlong);
1403 void (*SetStaticFloatField)(JNIEnv*, jclass, jfieldID, jfloat);
1404 void (*SetStaticDoubleField)(JNIEnv*, jclass, jfieldID, jdouble);
1405
1406 jstring (*NewString)(JNIEnv*, const jchar*, jsize);
1407 jsize (*GetStringLength)(JNIEnv*, jstring);
1408 const jchar* (*GetStringChars)(JNIEnv*, jstring, boolean*);
1409 void (*ReleaseStringChars)(JNIEnv*, jstring, const jchar*);
1410 jstring (*NewStringUTF)(JNIEnv*, const char*);
1411 jsize (*GetStringUTFLength)(JNIEnv*, jstring);
1412 /* JNI spec says this returns const jbyte, but that's inconsistent */
1413 const char* (*GetStringUTFChars)(JNIEnv*, jstring, boolean*);
1414 void (*ReleaseStringUTFChars)(JNIEnv*, jstring, const char*);
1415 jsize (*GetArrayLength)(JNIEnv*, jarray);
1416 jobjectArray (*NewObjectArray)(JNIEnv*, jsize, jclass, jobject);
1417 jobject (*GetObjectArrayElement)(JNIEnv*, jobjectArray, jsize);
1418 void (*SetObjectArrayElement)(JNIEnv*, jobjectArray, jsize, jobject);
1419
1420 jbooleanArray (*NewBooleanArray)(JNIEnv*, jsize);
1421 jbyteArray (*NewByteArray)(JNIEnv*, jsize);
1422 jcharArray (*NewCharArray)(JNIEnv*, jsize);
1423 jshortArray (*NewShortArray)(JNIEnv*, jsize);
1424 jintArray (*NewIntArray)(JNIEnv*, jsize);
1425 jlongArray (*NewLongArray)(JNIEnv*, jsize);
1426 jfloatArray (*NewFloatArray)(JNIEnv*, jsize);
1427 jdoubleArray (*NewDoubleArray)(JNIEnv*, jsize);
1428
1429 jboolean* (*GetBooleanArrayElements)(JNIEnv*, jbooleanArray, boolean*);
1430 jbyte* (*GetByteArrayElements)(JNIEnv*, jbyteArray, boolean*);
1431 jchar* (*GetCharArrayElements)(JNIEnv*, jcharArray, boolean*);
1432 jshort* (*GetShortArrayElements)(JNIEnv*, jshortArray, boolean*);
1433 jint* (*GetIntArrayElements)(JNIEnv*, jintArray, boolean*);
1434 jlong* (*GetLongArrayElements)(JNIEnv*, jlongArray, boolean*);
1435 jfloat* (*GetFloatArrayElements)(JNIEnv*, jfloatArray, boolean*);
1436 jdouble* (*GetDoubleArrayElements)(JNIEnv*, jdoubleArray, boolean*);
1437
1438 void (*ReleaseBooleanArrayElements)(JNIEnv*, jbooleanArray,
1439 jboolean*, jint);
1440 void (*ReleaseByteArrayElements)(JNIEnv*, jbyteArray,
1441 jbyte*, jint);
1442 void (*ReleaseCharArrayElements)(JNIEnv*, jcharArray,
1443 jchar*, jint);
1444 void (*ReleaseShortArrayElements)(JNIEnv*, jshortArray,
1445 jshort*, jint);
1446 void (*ReleaseIntArrayElements)(JNIEnv*, jintArray,
1447 jint*, jint);
1448 void (*ReleaseLongArrayElements)(JNIEnv*, jlongArray,
1449 jlong*, jint);
```

呐，参数和我们刚刚在汇编分析时描述的一样。那么刚刚的第一个第二个问题算是解决了。但是细心的朋友肯定注意到了一个小细节

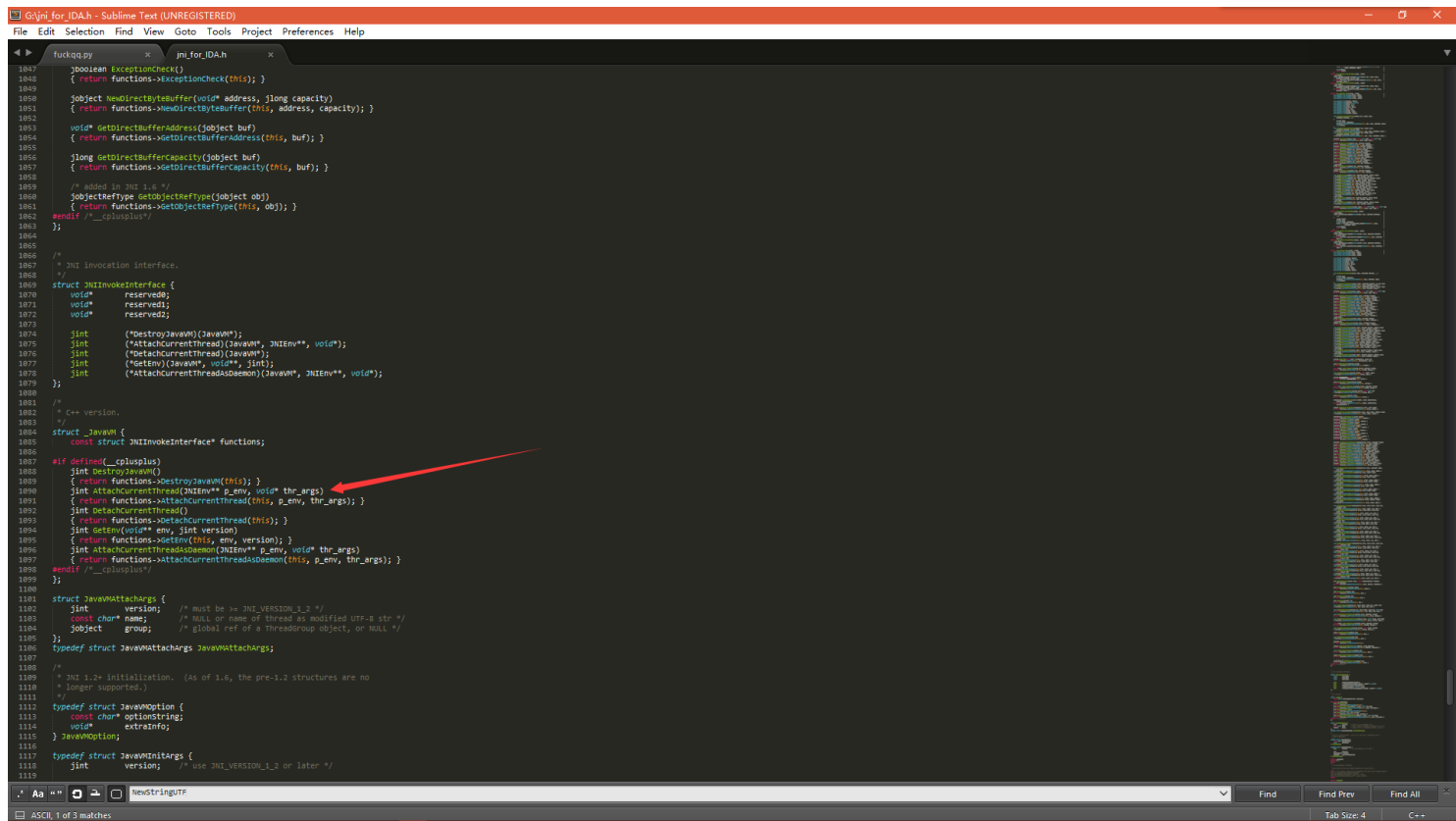
这个和JNIEnv在一块的另一个结构体是他娘的啥？凭这打破砂锅问到底的精神，我说



```
123 typedef union jvalue {
124     jboolean z;
125     jbyte b;
126     jchar c;
127     jshort s;
128     jint i;
129     jlong j;
130     jfloat f;
131     jdouble d;
132     jobject l;
133 } jvalue;
134
135 typedef enum jobjectType {
136     JNIInvalidType = 0,
137     JNIObjectType = 1,
138     JNIWeakGlobalType = 2,
139     JNIWeakGlobalType = 3
140 } jobjectType;
141
142 typedef struct {
143     const char* name;
144     const char* signature;
145     void* fptr;
146 } JNINativeMethod;
147
148 struct JNIEnv;
149 struct JavaVM;
150 typedef const struct JNINativeInterface* C_JNIEnv;
151
152 #if defined(__cplusplus)
153 typedef JNIEnv* JNIEnv;
154 typedef JavaVM* JavaVM;
155 #else
156 typedef const struct JNINativeInterface* JNIEnv;
157 typedef const struct JNIInvokeInterface* JavaVM;
158 #endif
159
160 /*
161  * Table of interface function pointers.
162  */
163 struct JNINativeInterface {
164     void* reserved1;
165     void* reserved2;
166     void* reserved3;
167     void* reserved4;
168     void* reserved5;
169
170     jint (*GetVersion)(JNIEnv *);
171
172     jclass (*DefineClass)(JNIEnv*, const char*, jobject, const jbyte*,
173                          jsize);
174     jclass (*FindClass)(JNIEnv*, const char*);
175
176     jmethodID (*FromReflectedMethod)(JNIEnv*, jobject);
177     jfieldID (*FromReflectedField)(JNIEnv*, jobject);
178     /* spec doesn't show boolean parameter */
179     jobject (*ToReflectedMethod)(JNIEnv*, jclass, jmethodID, jboolean);
180
181     jclass (*GetSuperclass)(JNIEnv*, jclass);
182     jboolean (*IsAssignableFrom)(JNIEnv*, jclass, jclass);
183
184     /* spec doesn't show boolean parameter */
185     jobject (*ToReflectedField)(JNIEnv*, jclass, jfieldID, jboolean);
186
187     jint (*Throw)(JNIEnv*, jthrowable);
188     jint (*ThrowNew)(JNIEnv*, jclass, const char *);
189     jthrowable (*ExceptionOccurred)(JNIEnv*);
190     void (*ExceptionDescribe)(JNIEnv*);
191     void (*ExceptionClear)(JNIEnv*);
192     void (*FatalError)(JNIEnv*, const char*);
193
194     jint (*PushLocalFrame)(JNIEnv*, jint);
195     jobject (*PopLocalFrame)(JNIEnv*, jobject);
196 }
```

这个JavaVM其实是虚拟机在JNI层的一个代表，一个进程中只有一个JavaVM,他是进程级的，那么相对的，JNIEnv其实是线程级的。那么他们必然是有关系的，有啥关系？你等着

看到这个JavaVm结构体了么？ 这里有个函数



通过调用这个函数我们就可以获得这个线程的JNIEnv结构体，干嘛要获得？等你要的时候你就晓得了

```
jint AttachCurrentThread(JNIEnv** p_env, void* thr_args)
```

那么再来看看第三个问题，什么时候是jclass，什么时候是object？

这个其实很好理解，在Java层，如果你声明的是一个static函数，那么他就是jclass，如果不是static函数，那就是jobject，很好理解的

0x04 函数注册

在JNI注册函数有两种注册方法，一种是静态注册，就是刚才演示的那种，在Java层声明，在JNI定义，下面来讲第二种，动态注册

将动态注册前，先看看Java层是怎样找到JNI层中对应的函数的，之前说过JNI库中默认使

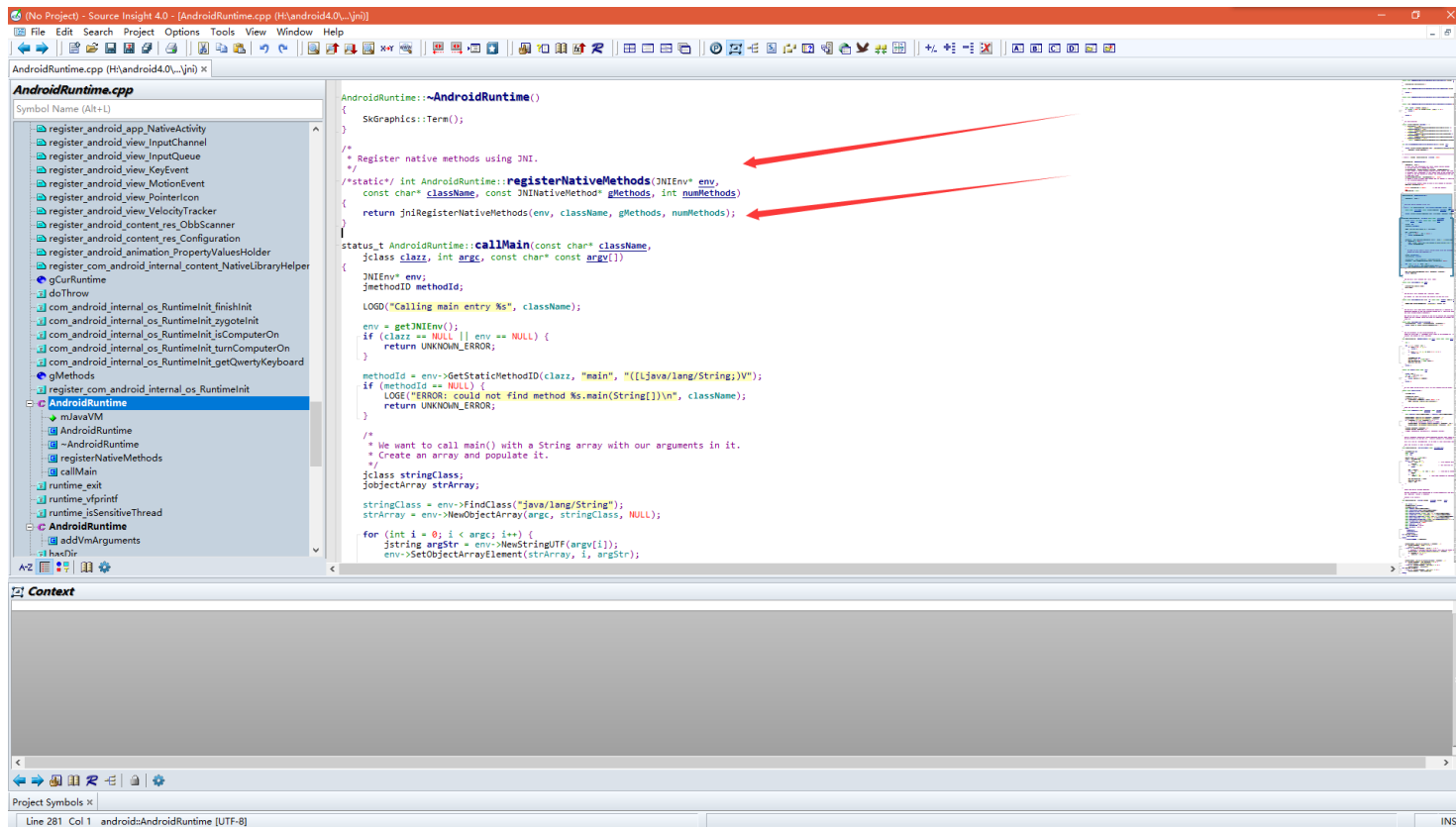
用Java_myapplication_mask_com_myapplication_MainActivity_FromNative这种格式，那么可以理解为JNI会为库和java层建立某种联系，注册函数其实就是建立这种联系，然后进行查找，作用类似指针，其实你就可以理解为指针，那么JNI中是如何建立这种关系的呢？

JNI.h中有这样一个结构体

```
111 typedef jobject      jobject;
112 typedef jobject      jobject;
113
114 #endif /* not __cplusplus */
115
116 struct _jfieldID;
117 typedef struct _jfieldID* jfieldID; /* opaque structure */
118
119 struct _jmethodID;
120 typedef struct _jmethodID* jmethodID; /* opaque structure */
121
122 struct JNIInvokeInterface;
123
124 typedef union jvalue {
125     jboolean    i;
126     jbyte       b;
127     jchar       c;
128     jshort      s;
129     jint        i;
130     jlong       l;
131     jfloat      f;
132     jdouble     d;
133     jobject     o;
134 } jvalue;
135
136 typedef enum jobjectType {
137     JNIInvalidType = 0,
138     JNIObjectType = 1,
139     JNIBooleanType = 2,
140     JNIStringType = 3
141 } jobjectType;
142
143 typedef struct {
144     const char* name;
145     const char* signature;
146     void*      fnPtr;
147 } JNIInvokeMethod;
148
149 struct JNIEnv;
150 struct Javaw;
151 typedef const struct JNIInvokeInterface* C_JNIEnv;
152
153 #if defined(__cplusplus)
154 typedef _JNIEnv JNIEnv;
155 typedef _Javaw Javaw;
156 #else
157 typedef const struct JNIInvokeInterface* JNIEnv;
158 typedef const struct JNIInvokeInterface* Javaw;
159 #endif
160
161 /*
162  * Table of interface function pointers.
163  */
164 struct JNIInvokeInterface {
165     void*      reserved0;
166     void*      reserved1;
167     void*      reserved2;
168     void*      reserved3;
169
170     jint       (*GetVersion)(JNIEnv *);
171     jclass     (*DefineClass)(JNIEnv*, const char*, jobject, const jbyte*,
172                               jsize);
173     jclass     (*FindClass)(JNIEnv*, const char*);
174
175     jmethodID  (*FromReflectedMethod)(JNIEnv*, jobject);
176     jfieldID   (*FromReflectedField)(JNIEnv*, jobject);
177     /* spec doesn't show jboolean parameter */
178     jobject    (*ToReflectedMethod)(JNIEnv*, jclass, jmethodID, jboolean);
179     jclass     (*GetSuperclass)(JNIEnv*, jclass);
180     jboolean   (*IsAssignableFrom)(JNIEnv*, jclass, jclass);
181
182     /*
183      * ...
184      */
185 }
```

```
typedef struct {
    const char* name; //函数名
    const char* signature; //签名信息
    void*      fnPtr; //这个就是函数对应的指针和上面说的吻合，果然是个指针
} JNIInvokeMethod;
```

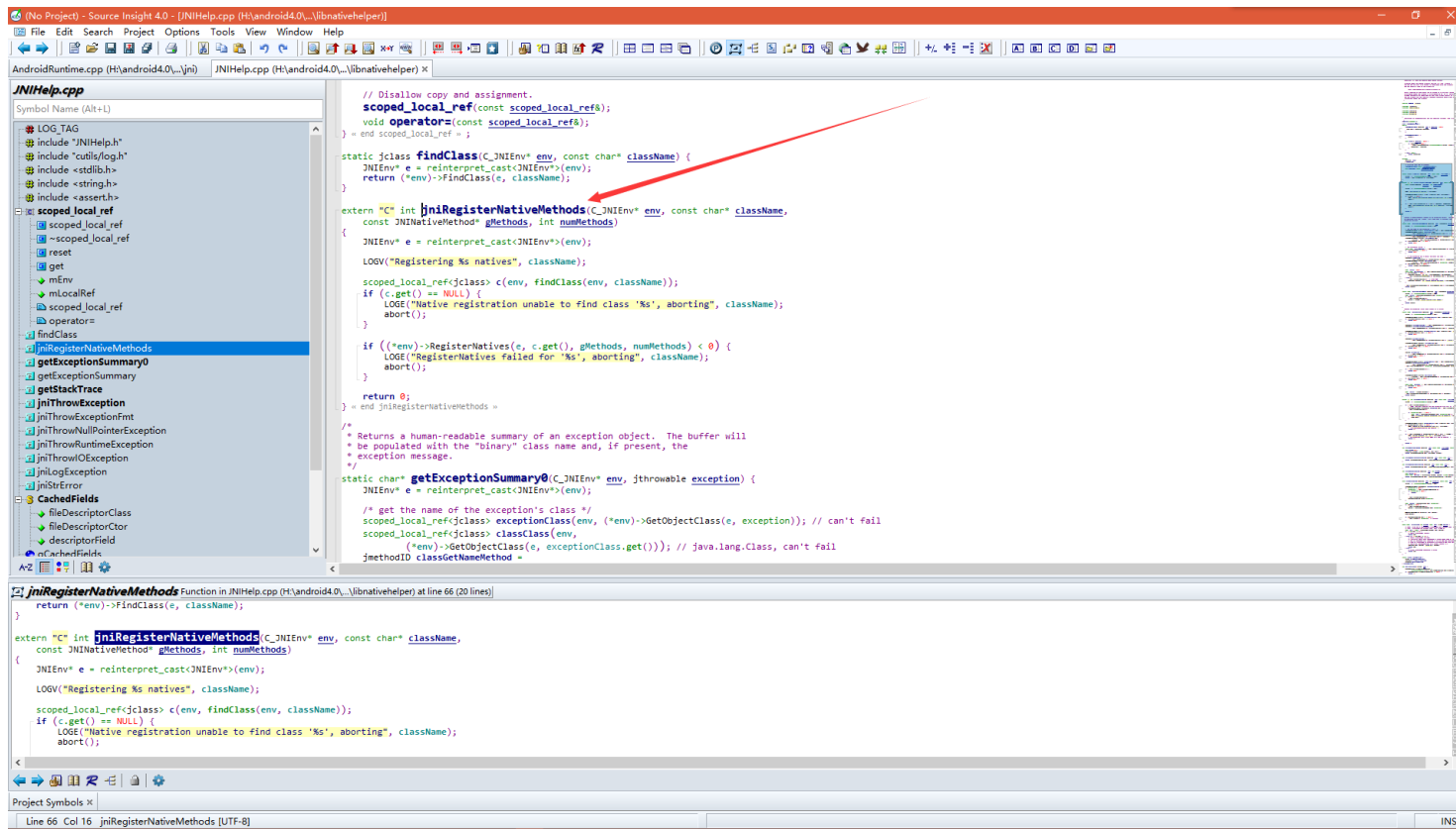
呐，找到这个关系了，怎么去注册呢？这就要分析源码了，源码位置在\frameworks\base\core\jni\AndroidRunTime.cpp,感兴趣可以看看



```
/*  
 * Register native methods using JNI.  
 */  
/*static*/ int AndroidRuntime::registerNativeMethods(JNIEnv* env,  
const char* className, const JNINativeMethod* gMethods, int numMethods)  
{  
    return jniRegisterNativeMethods(env, className, gMethods, numMethods);  
}
```

可以看到这里回调了一个函数，行，我说

诺，在这呢dalvik\libnativehelper\JNIHelp.c



```
static jclass findClass(C_JNIEnv* env, const char* className) {
    JNIEnv* e = reinterpret_cast<JNIEnv*>(env);
    return (*env)->FindClass(e, className);
}

extern "C" int jniRegisterNativeMethods(C_JNIEnv* env, const char* className,
    const JNINativeMethod* gMethods, int numMethods)
{
    JNIEnv* e = reinterpret_cast<JNIEnv*>(env);

    LOGV("Registering %s natives", className);

    scoped_local_ref<jclass> c(env, findClass(env, className));
    if (c.get() == NULL) {
        LOGE("Native registration unable to find class '%s', aborting", className);
        abort();
    }

    if ((*env)->RegisterNatives(e, c.get(), gMethods, numMethods) < 0) {
        LOGE("RegisterNatives failed for '%s', aborting", className);
    }
}
```

```
abort();  
}  
  
return 0;  
}
```

绕了半天。。。又回到JNI去了23333

那么我们动态注册的时候，只需要完成下面操作即可

```
jclass clazz = (*env) -> FindClass(env,className);  
(*env) ->RegisterNatives (env,clazz,gmethods,numMethods);
```

那么注册在哪操作呢??? 呐，需要在一个JNI_Onload()函数中，这个函数的第一个参数就是JavaVM* vm（动态注册必须实现这个函数，静态注册则不需要，但是这个函数通常可以用来初始化一些操作）

0x05 结语

这些只是新手自己的一些学习笔记，捋了捋思绪后稍微平静了些，那么也不早了

睡觉！！！！！！
