

管满满

热爱生活，热爱编程，分享源于学习。

博客园 首页 新随笔 联系 订阅 管理

Android NDK开发之从Java与C互调中详解JNI使用（一）

生活

这一个礼拜过得真的是苦不堪言，上周因为打球脚踝直接扭伤，肿的想猪蹄一样，然后休息几天消肿了，可以缓慢龟速的行走了，然而五一回来上班第一天，上班鞋子还能穿上，下班脚已插不进鞋子里面了，好吧，又肿回来了，苦逼。

正文

回到正文，上篇我们已学习到了[Android NDK开发之从环境搭建到Demo级十步流](#)，主题是DNK环境搭建和Demo示例开发步骤，而今天我们要学习的是通过JNI实现Java和C之间的交互。

对于JNI的理解，上一节也已讲过，这里在回顾下：

JNI：Java Native Interface 也就是java本地接口，它是一个协议，这个协议用来沟通java代码和本地代码(c/c++)。通过这个协议，Java类的某些方法可以使用原生实现，同时让它们可以像普通的Java方法一样被调用和使用，而原生方法也可以使用Java对象，调用和使用Java方法。也就是说，使用JNI这种协议可以实现：java代码调用c/c++代码，而c/c++代码也可以调用java代码。

接下来就以一个[登录实例](#)来详细的讲解使用JNI来完成Java与C代码之间的交互。

公告

昵称：管满满
园龄：1年6个月
粉丝：13
关注：4
[+加关注](#)

<	2017年12月						>
日	一	二	三	四	五	六	
26	27	28	29	30	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
31	1	2	3	4	5	6	

Java 调用 C 本地方法

JNI使用演示

一、首先先构造登录界面

登录界面中有三个文本输入框，分别对应用户名，密码，验证码。详情如下

NDKDemo JNI使用详解

请输入用户名

请输入密码

请输入验证码

登录

二、在Java2CJNI类中添加我们登录的login本地方法，且方法中带有三个不同类型的参数

搜索

找找看

谷歌搜索

我的标签
Android(11)
5.0新特性(3)
RecyclerView(3)
源码分析(3)
自定义View(3)
view(2)
JNI(2)
NDK(2)
React Native(1)
6.0新特性(1)



```
1 package com.sanhui.ndkdemo;
2
3 /**
4  * Created by guanmanman on 2017/4/26.
5  */
6
7 public class Java2CJNI {
8
9     static {
10         System.loadLibrary("Java2C");
11     }
12
13     public native String java2C();
14
15     public native String login(String userName, String PSW, int code);
16
17 }
```

三、使用javah生成头文件，并存放在jni文件夹下，且在本地方法中实现我们的登录逻辑（具体的逻辑将会在下面的JNI使用中详细讲解）

更多

随笔档案

2017年6月 (3)

2017年5月 (3)

2017年4月 (1)

2016年12月 (4)

2016年11月 (7)

积分与排名

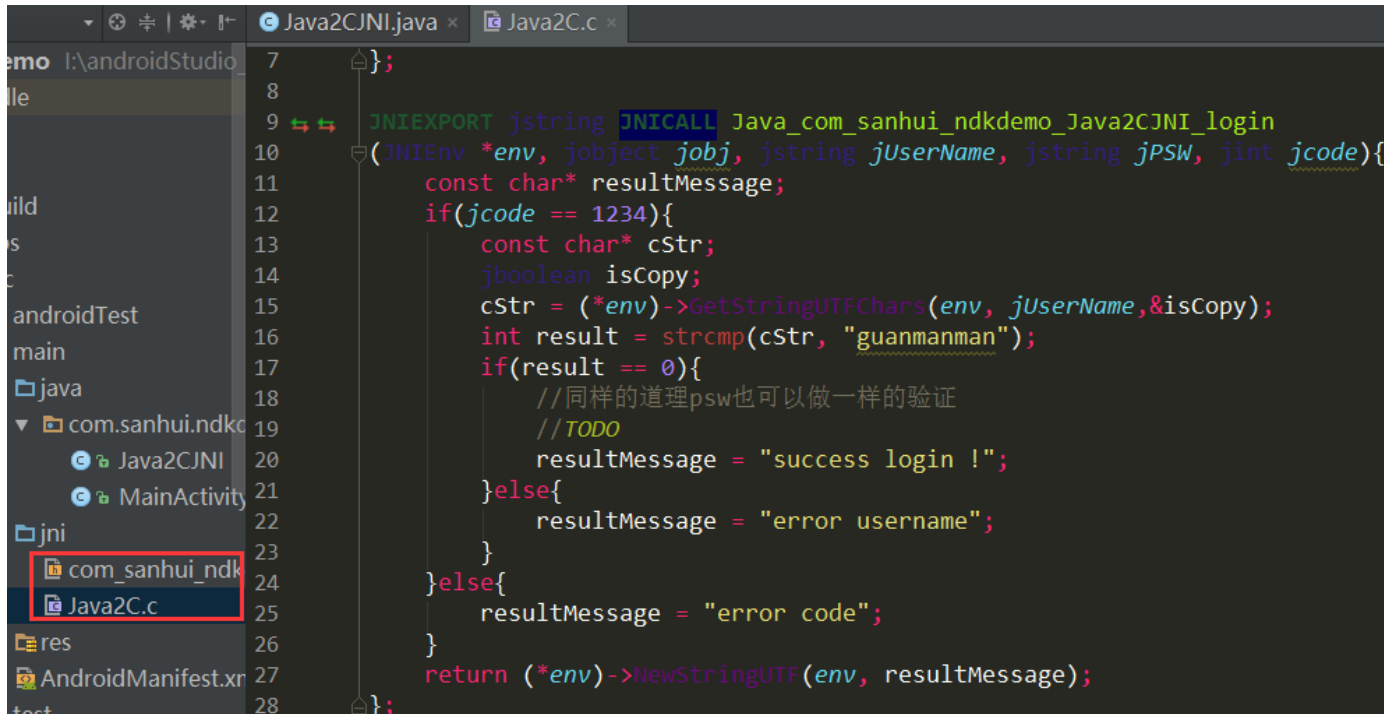
积分 - 22785

排名 - 15535

阅读排行榜

1. Android 图片加载框架Glide4.0源码完全解析（一）(4752)

2. Android6.0运行时权限管理(4088)



```
7  };
8
9  JNIEXPORT jstring JNICALL Java_com_sanhui_ndkdemo_Java2CJNI_login
10  (JNIEnv *env, jobject obj, jstring jUserName, jstring jPSW, jint jcode){
11      const char* resultMessage;
12      if(jcode == 1234){
13          const char* cStr;
14          jboolean isCopy;
15          cStr = (*env)->GetStringUTFChars(env, jUserName,&isCopy);
16          int result = strcmp(cStr, "guanmanman");
17          if(result == 0){
18              //同样的道理psw也可以做一样的验证
19              //TODO
20              resultMessage = "success login !";
21          }else{
22              resultMessage = "error username";
23          }
24      }else{
25          resultMessage = "error code";
26      }
27      return (*env)->NewStringUTF(env, resultMessage);
28  }
```

四、在我们的MainActivity中调用本地方法

3. Android 网络框架之Retrofit2使用详解及从源码中解析原理(2926)

4. Android 图片加载框架Glide4.0源码完全解析 (二) (2448)

5. Android 5.X新特性之为RecyclerView添加下拉刷新和上拉加载及SwipeRefreshLayout实现原理(2195)

评论排行榜

1. 2016点滴生活：收获与展望(8)

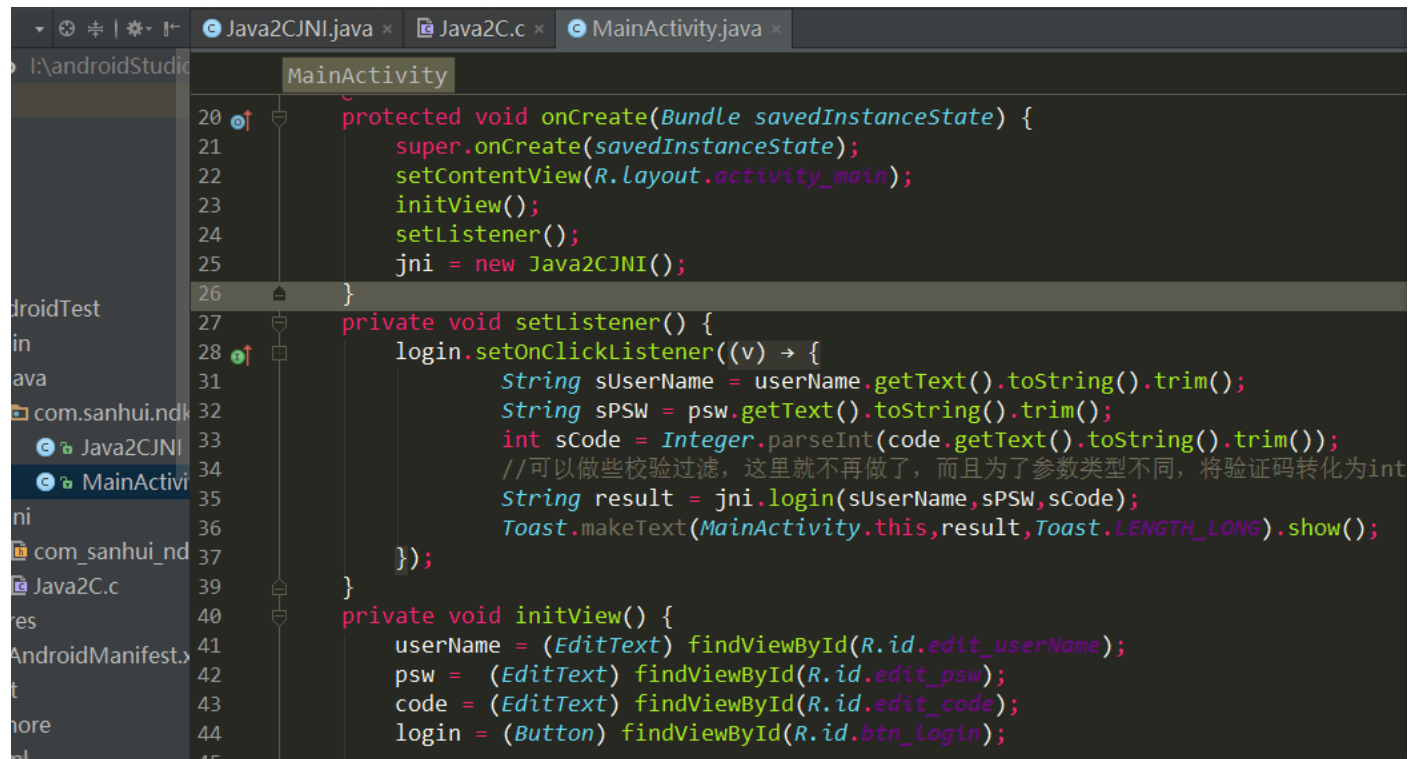
2. React Native环境配置之Windows版本搭建(1)

3. Android 自定义控件之继承ViewGroup创建新容器(1)

4. Android 设计模式实战之关于封装计费代码库的策略模式详谈(1)

5. Android NDK开发之C调用Java及原生代码断点调试 (二) (1)

推荐排行榜



```
1  Java2CJNI.java x  Java2C.c x  MainActivity.java x
2  I:\androidStu...
3  MainActivity
4
5  20  protected void onCreate(Bundle savedInstanceState) {
6
7  21      super.onCreate(savedInstanceState);
8
9  22      setContentView(R.layout.activity_main);
10
11  23      initView();
12
13  24      setListener();
14
15  25      jni = new Java2CJNI();
16
17  26  }
18
19  27  private void setListener() {
20
21  28      login.setOnClickListener((v) -> {
22
23  31          String sUserName = userName.getText().toString().trim();
24
25  32          String sPSW = psw.getText().toString().trim();
26
27  33          int sCode = Integer.parseInt(code.getText().toString().trim());
28          //可以做些校验过滤，这里就不再做了，而且为了参数类型不同，将验证码转化为int
29
30  34          String result = jni.login(sUserName, sPSW, sCode);
31
32  35          Toast.makeText(MainActivity.this, result, Toast.LENGTH_LONG).show();
33
34  36      });
35
36  37  }
37
38  39  }
39
40  40  private void initView() {
41
42  41      userName = (EditText) findViewById(R.id.edit_userName);
43
44  42      psw = (EditText) findViewById(R.id.edit_psw);
45
46  43      code = (EditText) findViewById(R.id.edit_code);
47
48  44      login = (Button) findViewById(R.id.btn_login);
49
50  45  }
```

五、来看执行结果

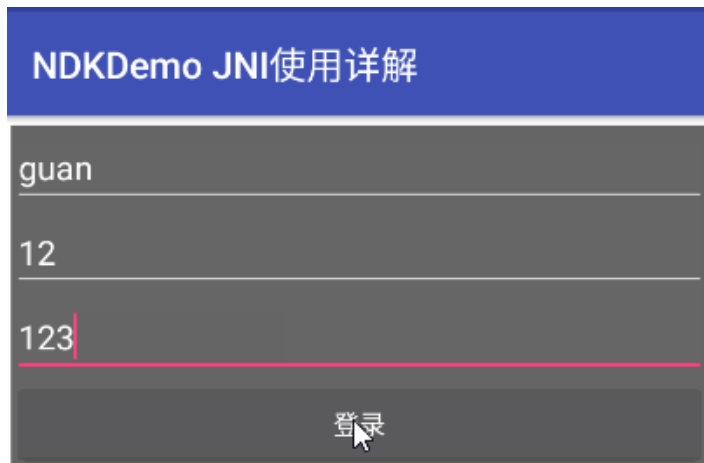
1. 2016点滴生活：收获与展望(7)

2. Android NDK开发之C调用Java及原生代码断点调试（二）(3)

3. Android NDK开发之从环境搭建到Demo级十步流(2)

4. Android 网络框架之Retrofit2使用详解及从源码中解析原理(1)

5. Android 设计模式实战之关于封装计费代码库的策略模式详谈(1)



上述的执行过程及结果，逻辑是在C代码中完成的，主要逻辑非常的简单，就是判断验证码和用户名是否相同，不同返回错误信息，相同则是登录成功。

上面所完成的是我们接下来要讲解JNI的基础前提，所以并没有详细的讲解，如果伙伴们有疑问或是不明白的地方请移步到[Android NDK开发之从环境搭建到Demo级十步流](#) 来先学习了先基础的知识，因为本次讲解是在上一节的基础上进行的。

OK，现在我们来详细的看下在Java2C.c中我们是怎么使用JNI来完成调用Java调用C的。

```
JNIEXPORT jstring JNICALL Java_com_sanhui_ndkdemo_Java2CJNI_login
(JNIEnv *env, jobject obj, jstring jUserName, jstring jPSW, jint jcode){
    const char* resultMessage;
```

```
if(jcode == 1234){
    const char* cStr;
    jboolean isCopy;
    cStr = (*env)->GetStringUTFChars(env, jUserName,&isCopy);
    int result = strcmp(cStr, "guanmanman");
    if(result == 0){
        //同样的道理psw也可以做一样的验证
        //TODO
        resultMessage = "success login !";
    }else{
        resultMessage = "error username";
    }
}
}
}
return (*env)->NewStringUTF(env, resultMessage);
};
```

JNI方法声明

我们知道通过javah会为我们生成这样的头文件方法（补全后）：

```
JNIEXPORT jstring JNICALL Java_com_sanhui_ndkdemo_Java2CJNI_login
(JNIEnv *env, jobject jobj, jstring jUserName, jstring jPSW, jint jcode){}
```

这里做下解释：

- ①：我们知道在调用本地方法时会返回一个String类型的返回值，jstring 就是JNI对应Java的一个返回类型。
- ②：通过Java_ + 包名（com.sanhui.ndkdemo）+ 类名(Java2CJNI) + 方法名(login)的形式生成在原生代码中所识别的方法，当java虚拟机调用com.sanhui.ndkdemo.Java2CJNI.login的方法时会自动查找到这个C实现的Native函数并调用。
- ③：参数列表中包含五个参数，其中三个是通过java调用传递的，这个不多说，JNIEnv 是指向可用JNI函数表的接口指针，也就是说通过它可以调用JNI所封装的函数进行处理逻辑业务，jobject 则是Java类中的对象引用，这里指的是Java2CJNI类的实例，也就是说，当调用本地方法时，JNI将会自动的获取当前类的实例，以方便原生代码使用。

JNI 数据类型

原生数据类型同Java一样，都包含两类，基本数据类型与引用类型。

JNI 基础数据类型

在JNI当中，基本数据类型可以直接与C/C++的相对应的基本数据类型映射，所以我们可以直接拿来使用并不需要转换。

我们先来看看java、JNI、C之间的基本类型的映射关系：

java类型	JNI类型	C类型
Boolean	jboolean	unsigned char
Byte	jbyte	char
Char	jchar	unsigned short
Short	jshort	short
Int	jint	int
Long	jlong	long long
Float	jfloat	float
Double	jdouble	double

通过上面讲解和表格我们现在可以明白为什么我们通过本地方法传递进来的整形验证码，使用JNI技术转化为jint类型时可以直接的进行如下判断：

```
if(jcode == 1234){
}
```

JNI 引用类型-字符串

引用类型我们并不能直接的使用，它不是以原生数据类型形式展现，而是需要通过JNI提供的一组相关的API把引用类型提供给原生代码使用。

java类型	JNI类型
java.lang.Class	jclass
java.lang.Throwable	jthrowable
java.lang.String	jstring
byte[]	jbyteArray
boolean[]	jbooleanArray
char[]	jcharArray

java类型	JNI类型
short[]	jshortArray
int[]	jintArray
long[]	jlongArray
float[]	jfloatArray
double[]	jdoubleArray

上面也说过引用类型不能够直接被原生代码使用，它需要通过JNI提供的API来提供给原生使用，例如：

```
const char* cStr;  
jboolean isCopy;  
cStr = (*env)->GetStringUTFChars(env, jUserName,&isCopy);  
int result = strcmp(cStr, "guanmanman");
```

上面的代码中，GetStringUTFChars就是JNI所提供的API，通过该方法可以把java中所传递的String字符串转化为原生代码所能识别的字符串，然后在进行下面的比较操作。strcmp方法是C语言中所提供的比较字符串的方法。

说到这里，我们在来详细的看下JNI针对字符串的操作：

①：创建字符串

可以在原生代码中使用NewString函数构建Unicode编码风格的字符串实例，假如你使用的是utf-8的编码格式，可以使用NewStringUTF函数来构建，如：

```
jstring jString = (*env)->NewString(env,"I am from C");  
jstring jString = (*env)->NewStringUTF(env,"I am from C");
```

②：把Java字符串转换成C字符串

Java字符串String是属于引用类型的，它不能够直接被原生代码使用，为了在原生代码中使用Java传递过来的String串，我们需要借用JNI API，根据编码的不同，分别可以用GetStringChars和GetStringUTFChars函数进行转换，如我们上面的实例：

```
const char* cStr = (*env)->GetStringUTFChars(env, jUserName,&isCopy);
```

isCopy参数是可选的，它是jboolean类型的，它的调用者明确返回的C字符串地址是指向副本还是指向堆中的对象。

JNI 引用类型-数组

在引用类型中还有一种比较重要的类型，那就是数组，JNI中也提供了相应的API来处理和使用数组。

①：创建数组

JNI通过New<?>Array函数在原生代码中创建数组，<?>类型可以是byte，short，int等类型，如：

```
jintArray intArray = (*env)->NewIntArray(env,5);
```

NewIntArray数组中应该给出明确的数组长度，例如，5就是intArray的数组长度。

②：对数组的操作

如以上介绍，数组属于引用类型，通过Java传递过来的数组我们是无法直接进行操作的，但是根据JNI提供的API我们可以很顺利的进行数组的操作，如：

```
//获取数组长度
//jsize GetArrayLength(this, array);
//获取数组元素
//jint* GetIntArrayElements(this, array, isCopy);
```

不仅仅可以对数组进行直接的操作，而且还可以对数组的副本进行操作，如

```
void (*GetBooleanArrayRegion)(JNIEnv*, jbooleanArray,
                               jsize, jsize, jboolean*);
void (*GetByteArrayRegion)(JNIEnv*, jbyteArray,
                             jsize, jsize, jbyte*);
void (*GetCharArrayRegion)(JNIEnv*, jcharArray,
                             jsize, jsize, jchar*);
void (*GetShortArrayRegion)(JNIEnv*, jshortArray,
                              jsize, jsize, jshort*);
void (*GetIntArrayRegion)(JNIEnv*, jintArray,
                            jsize, jsize, jint*);
void (*GetLongArrayRegion)(JNIEnv*, jlongArray,
                             jsize, jsize, jlong*);
void (*GetFloatArrayRegion)(JNIEnv*, jfloatArray,
                              jsize, jsize, jfloat*);
void (*GetDoubleArrayRegion)(JNIEnv*, jdoubleArray,
                               jsize, jsize, jdouble*);
```

通过Get<?>ArrayRegion函数将给定的java数组复制成C数组，然后我们就可以在原生代码中针对C数组进行操作。

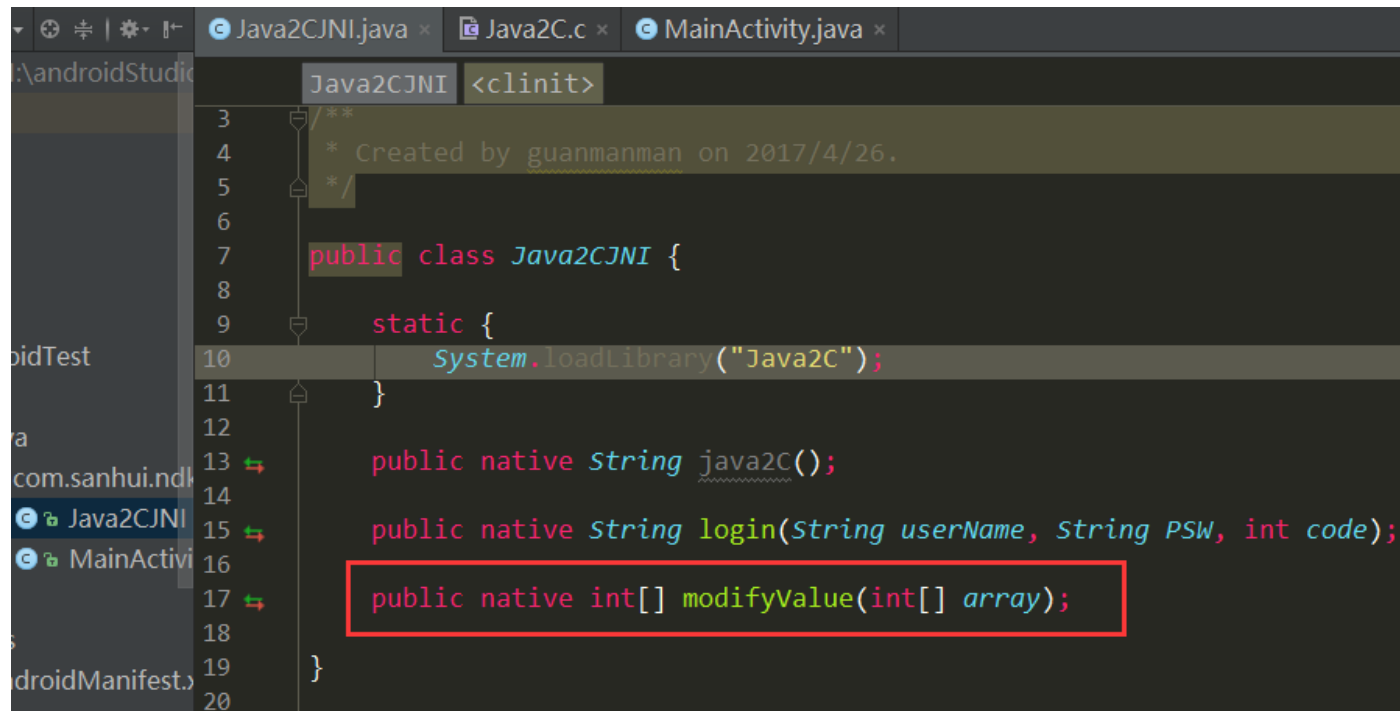
操作完成之前我们还可以通过以下函数把数组的副本给还原到原来的Java数组中，

```
/* spec shows these without const; some jni.h do, some don't */
void      (*SetBooleanArrayRegion)(JNIEnv*, jbooleanArray,
                                   jsize, jsize, const jboolean*);
void      (*SetByteArrayRegion)(JNIEnv*, jbyteArray,
                                 jsize, jsize, const jbyte*);
void      (*SetCharArrayRegion)(JNIEnv*, jcharArray,
                                jsize, jsize, const jchar*);
void      (*SetShortArrayRegion)(JNIEnv*, jshortArray,
                                 jsize, jsize, const jshort*);
void      (*SetIntArrayRegion)(JNIEnv*, jintArray,
                               jsize, jsize, const jint*);
void      (*SetLongArrayRegion)(JNIEnv*, jlongArray,
                                jsize, jsize, const jlong*);
void      (*SetFloatArrayRegion)(JNIEnv*, jfloatArray,
                                 jsize, jsize, const jfloat*);
void      (*SetDoubleArrayRegion)(JNIEnv*, jdoubleArray,
                                  jsize, jsize, const jdouble*);
```

通过以上的函数我们就完成了在原生代码中对Java数组的操作。

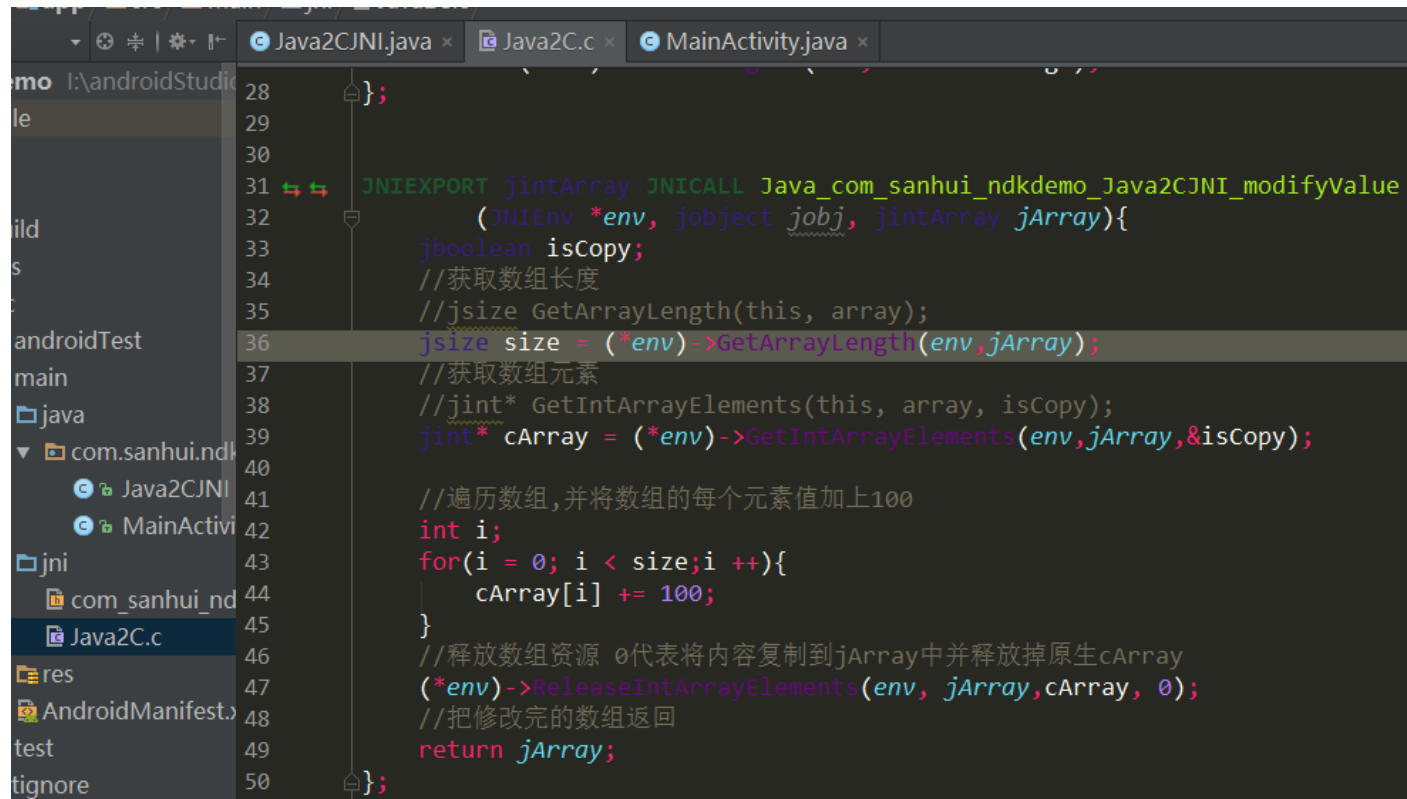
③：数组操作实例

首先，在Java2CJNI类中添加一个本地方法modifyValue，接受一个int类型的数组，完成后重新通过javah生成.h头文件，如：



```
Java2CJNI <clinit>
3  /**
4   * Created by guanmanman on 2017/4/26.
5   */
6
7  public class Java2CJNI {
8
9      static {
10         System.loadLibrary("Java2C");
11     }
12
13     public native String java2C();
14
15     public native String login(String userName, String PSW, int code);
16
17     public native int[] modifyValue(int[] array);
18
19 }
20
```

然后在Java2C.c中完成对它的逻辑操作：



```
28     }  
29  
30  
31     JNIEXPORT jintArray JNICALL Java_com_sanhui_ndkdemo_Java2CJNI_modifyValue  
32     (JNIEnv *env, jobject obj, jintArray jArray){  
33         jboolean isCopy;  
34         //获取数组长度  
35         //jsize GetArrayLength(this, array);  
36         jsize size = (*env)->GetArrayLength(env, jArray);  
37         //获取数组元素  
38         //jint* GetIntArrayElements(this, array, isCopy);  
39         jint* cArray = (*env)->GetIntArrayElements(env, jArray, &isCopy);  
40  
41         //遍历数组,并将数组的每个元素值加上100  
42         int i;  
43         for(i = 0; i < size; i++){  
44             cArray[i] += 100;  
45         }  
46         //释放数组资源 0代表将内容复制到jArray中并释放掉原生cArray  
47         (*env)->ReleaseIntArrayElements(env, jArray, cArray, 0);  
48         //把修改完的数组返回  
49         return jArray;  
50     }  
};
```

代码比较简单，就是通过JNI API获取数组的长度、元素，然后遍历数组针对每个元素进行加上100，完成后把C数组还原给Java数组，代码注释的已很清楚的体现。

最后在MainActivity中调用modifyValue方法，来看下执行过程和结果。

NDKDemo JNI使用详解

请输入用户名

请输入密码

请输入验证码

登录

原数组: `int[] array = {1,2,3,4,5,6,7,8}`

改变后的数组值: `{ 101,102,103,104,105,106,107,108, }`

修改数组值

OK，看了下篇幅，没想到这么长了，原本打算把Java调C和C调Java一起讲解下的，看来C调Java只能放到下一篇进行讲解了。

下篇主题预告：原生C访问Java成员、调用Java方法的JNI使用，以及使用gradle-experimental来调试原生代码。

那么，今天就先讲解到这里吧。请关注微信公众号。谢谢

更多资讯请关注微信平台，有博客更新会及时通知。爱学习爱技术。



标签: JNI, NDK, Android

好文要顶

关注我

收藏该文



管满满

关注 - 4

粉丝 - 13

+加关注

0

0

« 上一篇: Android NDK开发之从环境搭建到Demo级十步流

» 下一篇: Android NDK开发之C调用Java及原生代码断点调试（二）

posted @ 2017-05-05 09:20 管满满 阅读(1204) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】腾讯云免费实验室，1小时搭建人工智能应用

【新闻】H3 BPM体验平台全面上线



最新IT新闻:

- 铁路12306 App迎3.0版更新：全新界面
 - 若福克斯资产出售 可能会让Hulu成Netflix强大对手
 - 三星260亿美元的豪赌：想垄断DRAM和NAND闪存市场
 - 苹果还是没有放弃？屏下指纹识别专利被曝光
 - Google员工和Google.org在2017年向非营利组织捐赠了2.6亿美元
- » 更多新闻...



最新知识库文章:

- 以操作系统的角度述说线程与进程
- 软件测试转型之路
- 门内门外看招聘

- 大道至简，职场上做人做事做管理
- 关于编程，你的练习是不是有效的？
- » [更多知识库文章...](#)

Copyright ©2017 管满满