

qinjuning、lets go


编程珠玑 JVM Android爱好者QQ群: 55945620

目录视图

摘要视图

RSS 订阅

个人资料



qinjuning

访问: 1416766次

积分: 9419

等级: BLOG 5

排名: 第884名

原创: 43篇

转载: 0篇

译文: 3篇

评论: 1026条


通知

本文博客欢迎转载, 请保留出处。谢谢合作

武汉Android联盟QQ群

Android爱好者QQ群: 55945620, 希望热爱武汉Android同胞能够加入。


博客专栏



Android框架浅析

文章: 10篇

阅读: 593426



Android技巧拾取

文章: 17篇

阅读: 490589

文章分类

Andoird技巧拾取 (27)

Andoird框架浅析 (12)

Java技巧拾取 (3)

Linux学习笔记 (1)

总结:难得糊涂 (3)

专家和你聊Web, 速来报名 微信开发学习路线高级篇上线 免费公开课平台正式上线啦 恭喜July新书上市

JNI学习积累之三 ---- 操作JNI函数以及复杂对象传递

分类: Java技巧拾取 2012-05-27 21:19 20156人阅读 评论(13) 收藏 举报

jni

java

class

string

list

callback

目录(?) [+]

本文原创, 转载请注明出处: <http://blog.csdn.net/qinjuning>

在掌握了JNI函数的使用和相关类型的映射后, 以及知晓何利用javah工具生成对应的jni函数以及如何生成动态链接库 (windos下就是.dll库, Linux就是.so库了, 不懂在Window下生成dll动态库的, 具体流程可看我的这篇博客: 《Android中JNI的使用之一: Java原生JNI的使用、javah指令的使用以及图解教材》)。即可掌握JNI的使用了了。

总的来说, JNI是不难的。通过前面的学习相信你应该有所了解。今天, 我们从几个简单的小例子, 来对JNI进行下实战训练。可都是些小例子, 耐心看咯。

主要操作内容, 包括如下几个部分:

1、在Native层返回一个字符串

2、从Native层返回一个int型二维数组(int a[][])

3、从Native层操作Java层的类: 读取/设置类属性

4、在Native层操作Java层的类: 读取/设置类属性、回调Java方法

5、从Native层返回一个复杂对象(即一个类咯)

6、在Java层传递复杂对象至Native层

7、从Native层返回ArrayList集合对象

广而告知, 这些操作就是简单的利用一些JNI函数即实现了。so easy。

一、在Native层返回一个字符串

Java层原型方法:

[java] view plain

01. public class

02. ...

03. public native void JNICALL getString(JNIEnv env, jobject obj);

04. ...

05. }

阅读排行

Android中View绘制流程I	(157373)
Android中Context详解 ---	(150847)
Android中获取应用程序((96211)
Android框架浅析之锁屏(I	(94181)
Android中Preference的使	(64947)
Android中滑屏初探 ---- s	(50035)
Android中将布局文件/Vie	(46719)
Android中滑屏实现 ---- 手	(46337)
Andriod中绘(画)图 ---- Car	(45347)
Android中获取正在运行	(43643)

评论排行

Android中View绘制流程I	(106)
Android中Context详解 ---	(68)
Android中滑屏实现 ---- 手	(64)
Android中获取应用程序((63)
毕业半年，点滴在心中	(62)
Android框架浅析之锁屏(I	(59)
Android中获取应用程序((52)
Android学习进阶路线导航	(47)
Android中将布局文件/Vie	(42)
Android中Preference的使	(41)

文章存档

2014年12月 (1)
2013年04月 (1)
2013年03月 (1)
2013年01月 (1)
2012年11月 (1)

展开

最新评论

Android中View绘制流程以及inva woshimiaoqingren: 好人一生平安
Android中measure过程、WRAP u010588886: 好厉害的感觉
Android中Context详解 ---- 你所不 jj1234588: thanks!
Android中获取应用程序(包)的信 happy_horse: 真棒
Android中Preference的使用以及 zy000000001: 十分的到位!!!! 感谢
Android中Context详解 ---- 你所不 qq_29877767: @leehong2005:xxxactivity.this getApplicationCont...
Android中measure过程、WRAP u010236416: 非常好
毕业半年，点滴在心中 qq_18212135: 最近很迷茫，本人 现大四，无意中看到楼主博客， 可是还是不知道怎么选择。。。。 楼主还能看到吗。。。。QQ似...
Android中获取应用程序(包)的信 清溪@Cherry: 楼主辛苦了，学 习了
Android中Context详解 ---- 你所不 I5741T: 谢谢分享! 学习了

Native层该方法实现为：

```
[java] view plain copy print ?
01.  /*
02.  * Class:      com_feixun_jni_HelloJni
03.  * Method:     getAJNIString
04.  * Signature:  ()Ljava/lang/String;
05.  */
06.  //返回字符串
07.  JNIEXPORT jstring JNICALL Java_com_feixun_jni_HelloJni_getAJNIString(JNIEnv * env, jobject obj)
08.  {
09.      jstring str = env->newStringUTF("HelloJNI"); //直接使用该JNI构造一个jstring对象返回
10.      return str ;
11.  }
```

二、在Native层返回一个int型二维数组(inta[][])

Java层原型方法：

```
[java] view plain copy print ?
01.  public class HelloJni {
02.      ...
03.      //参数代表几行几列数组，形式如: int a[dimon][dimon]
04.      private native int[][] getTwoArray(int dimon) ;
05.      ...
06.  }
```

Native层该方法实现为：

```
[java] view plain copy print ?
01.  /*
02.  * Class:      com_feixun_jni_HelloJni
03.  * Method:     getTwoArray
04.  * Signature:  (I)[[I
05.  */
06.  //通过构造一个数组的数组，返回 一个二维数组的形式
07.  JNIEXPORT jobjectArray JNICALL Java_com_feixun_jni_HelloJni_getTwoArray
08.  (JNIEnv * env, jobject object, jint dimion)
09.  {
10.
11.      jclass intArrayClass = env->FindClass("[I"); //获得一维数组 的类引用，即jintArray类型
12.      //构造一个指向jintArray类一维数组的对象数组，该对象数组初始大小为dimion
13.      jobjectArray obejctIntArray = env->NewObjectArray(dimion ,intArrayClass , NULL);
14.
15.      //构建dimion个一维数组，并且将其引用赋值给obejctIntArray对象数组
16.      for( int i = 0 ; i< dimion ; i++ )
17.      {
18.          //构建jint型一维数组
19.          jintArray intArray = env->NewIntArray(dimion);
20.
21.          jint temp[10] ; //初始化一个容器，假设 dimion < 10 ;
22.          for( int j = 0 ; j < dimion ; j++)
23.          {
24.              temp[j] = i + j ; //赋值
25.          }
26.
27.          //设置jint型一维数组的值
28.          env->SetIntArrayRegion(intArray, 0 , dimion ,temp);
29.          //给object对象数组赋值，即保持对jint一维数组的引用
30.          env->SetObjectArrayElement(obejctIntArray , i ,intArray);
31.
32.          env->DeleteLocalRef(intArray); //删除局部引用
33.      }
34.
35.      return obejctIntArray; //返回该对象数组
36.  }
```

三、在Native层操作Java层的类：读取/设置类属性

Java层原型方法：

```
[java] view plain copy print ?
01. public class HelloJni {
02.     ...
03.     //在Native层读取/设置属性值
04.     public native void native_set_name() ;
05.     ...
06.
07.     private String name = "I am at Java" ; //类属性
08. }
```

Native层该方法实现为：

```
[java] view plain copy print ?
01. /*
02.  * Class:      com_feixun_jni_HelloJni
03.  * Method:     native_set_name
04.  * Signature:  ()V
05.  */
06. //在Native层操作Java对象，读取/设置属性等
07. JNIEXPORT void JNICALL Java_com_feixun_jni_HelloJni_native_1set_1name
08. (JNIEnv *env , jobject obj ) //obj代表执行此JNI操作的类实例引用
09. {
10.     //获得jfieldID 以及 该字段的初始值
11.     jfieldID nameFieldId ;
12.
13.     jclass cls = env->GetObjectClass(obj); //获得Java层该对象实例的类引用，即HelloJNI类引用
14.
15.     nameFieldId = env->GetFieldID(cls , "name" , "Ljava/lang/String;"); //获得属性句柄
16.
17.     if(nameFieldId == NULL)
18.     {
19.         cout << " 没有得到name 的句柄Id \n;" ;
20.     }
21.     jstring javaNameStr = (jstring)env->GetObjectField(obj , nameFieldId); // 获得该属性的值
22.     const char * c_javaName = env->GetStringUTFChars(javaNameStr , NULL); //转换为 char *类型
23.     string str_name = c_javaName ;
24.     cout << "the name from java is " << str_name << endl ; //输出显示
25.     env->ReleaseStringUTFChars(javaNameStr , c_javaName); //释放局部引用
26.
27.     //构造一个jString对象
28.     char * c_ptr_name = "I come from Native" ;
29.
30.     jstring cName = env->NewStringUTF(c_ptr_name); //构造一个jstring对象
31.
32.     env->SetObjectField(obj , nameFieldId , cName); // 设置该字段的值
33. }
```

四、在Native层操作Java层的类：回调Java方法

Java层原型方法：

```
[java] view plain copy print ?
01. public class HelloJni {
02.     ...
03.     //Native层回调的方法实现
04.     public void callback(String fromNative){
05.         System.out.println(" I was invoked by native method ##### " + fromNative);
06.     };
07.     public native void doCallBack(); //Native层会调用callback()方法
08.     ...
09.
10.     // main函数
11.     public static void main(String[] args)
12.     {
13.         new HelloJni().doCallBack();
14.     }
15. }
```

Native层该方法实现为：

```
[java] view plain copy print ?
01.  /*
02.   * Class:      com_feixun_jni_HelloJni
03.   * Method:     doCallBack
04.   * Signature: ()V
05.   */
06.  //Native层回调Java类方法
07.  JNIEXPORT void JNICALL Java_com_feixun_jni_HelloJni_doCallBack
08.  (JNIEnv * env , jobject obj)
09.  {
10.      //回调Java中的方法
11.
12.      jclass cls = env->GetObjectClass(obj); //获得Java类实例
13.      jmethodID callbackID = env->GetMethodID(cls , "callback" , "(Ljava/lang/String;)V") ; //或
      得该回调方法句柄
14.
15.      if(callbackID == NULL)
16.      {
17.          cout << "getMethodId is failed \n" << endl ;
18.      }
19.
20.      jstring native_desc = env->NewStringUTF(" I am Native");
21.
22.      env->CallVoidMethod(obj , callbackID , native_desc); //回调该方法，并且传递参数值
23.  }
```

接下来，我们会操作复杂对象，也就是Java层的类，包括从Native层返回一个类以及传递一个类到Native层去，这儿我们使用的类非常简单，如下：

Student.java类

```
[java] view plain copy print ?
01.  package com.feixun.jni;
02.
03.  public class Student
04.  {
05.      private int age ;
06.      private String name ;
07.      //构造函数，什么都不做
08.      public Student(){ }
09.
10.      public Student(int age ,String name){
11.          this.age = age ;
12.          this.name = name ;
13.      }
14.
15.      public int getAge() {
16.          return age;
17.      }
18.      public void setAge(int age) {
19.          this.age = age;
20.      }
21.      public String getName() {
22.          return name;
23.      }
24.      public void setName(String name){
25.          this.name = name;
26.      }
27.
28.      public String toString(){
29.          return "name --- >" + name + " age --->" + age ;
30.      }
31.  }
```

五、在Native层返回一个复杂对象(即一个类略)

Java层的方法对应为：

```
[java] view plain copy print ?
01. public class HelloJni {
02.     ...
03.     //在Native层返回一个Student对象
04.     public native Student nativeGetStudentInfo() ;
05.     ...
06. }
```

Native层该方法实现为：

```
[java] view plain copy print ?
01. /*
02.  * Class:      com_feixun_jni_HelloJni
03.  * Method:     nativeGetStudentInfo
04.  * Signature:  ()Lcom/feixun/jni/Student;
05.  */
06. //返回一个复杂对象
07. JNIEXPORT jobject JNICALL Java_com_feixun_jni_HelloJni_nativeGetStudentInfo
08. (JNIEnv * env, jobject obl)
09. {
10.     //关于包描述符，这儿可以是 com/feixun/jni/Student 或者是 Lcom/feixun/jni/Student;
11.     // 这两种类型 都可以获得class引用
12.     jclass stucls = env->FindClass("com/feixun/jni/Student"); //或得Student类引用
13.
14.     //获得得该类型的构造函数 函数名为 <init> 返回类型必须为 void 即 V
15.     jmethodID constrocMID = env->GetMethodID(stucls,"<init>","(Ljava/lang/String;)V");
16.
17.     jstring str = env->NewStringUTF(" come from Native ");
18.
19.     jobject stu_obj = env->NewObject(stucls,constrocMID,11,str); //构造一个对象，调用该类的构造
    函数，并且传递参数
20.
21.
22.     return stu_obj ;
23. }
```

六、从Java层传递复杂对象至Native层

Java层的方法对应为：

```
[java] view plain copy print ?
01. public class HelloJni {
02.     ...
03.     //在Native层打印Student的信息
04.     public native void printStuInfoAtNative(Student stu);
05.     ...
06. }
```

Native层该方法实现为：

```
[java] view plain copy print ?
01. /*
02.  * Class:      com_feixun_jni_HelloJni
03.  * Method:     printStuInfoAtNative
04.  * Signature:  (Lcom/feixun/jni/Student;)V
05.  */
06. //在Native层输出Student的信息
07. JNIEXPORT void JNICALL Java_com_feixun_jni_HelloJni_printStuInfoAtNative
08. (JNIEnv * env, jobject obj, jobject obj_stu) //第二个类实例引用代表Student类，即我们传递下来的
    对象
09. {
10.
11.     jclass stu_cls = env->GetObjectClass(obj_stu); //或得Student类引用
12.
13.     if(stu_cls == NULL)
14.     {
15.         cout << "GetObjectClass failed \n" ;
16.     }
17.     //下面这些函数操作，我们都见过的。O(n_n)O~
18.     jfieldID ageFieldID = env->GetFieldID(stucls,"age","I"); //获得得Student类的属性id
19.     jfieldID nameFieldID = env->GetFieldID(stucls,"name","Ljava/lang/String;"); // 获得属性
```

```

20. ID
21.     jint age = env->GetIntField(objstu , ageFieldID); //获得属性值
22.     jstring name = (jstring)env->GetObjectField(objstu , nameFieldID); //获得属性值
23.
24.     const char * c_name = env->GetStringUTFChars(name ,NULL); //转换成 char *
25.
26.     string str_name = c_name ;
27.     env->ReleaseStringUTFChars(name,c_name); //释放引用
28.
29.     cout << " at Native age is :" << age << " # name is " << str_name << endl ;
30. }

```

七、最后加个难度，即在Native层返回集合对象(留这儿，以后也好找点)

Java层的对应方法为：

```

[java] view plain copy print ?
01. public class HelloJni {
02.     ...
03.     //在Native层返回ArrayList集合
04.     public native ArrayList<Student> native_getListStudents();
05.     ...
06. }

```

Native层该方法实现为：

```

[java] view plain copy print ?
01. /*
02.  * Class:      com_feixun_jni_HelloJni
03.  * Method:     native_getListStudents
04.  * Signature:  ()Ljava/util/ArrayList;
05.  */ //获得集合类型的数组
06. JNIEXPORT jobject JNICALL Java_com_feixun_jni_HelloJni_native_getListStudents
07. (JNIEnv * env, jobject obj)
08. {
09.     jclass list_cls = env->FindClass("Ljava/util/ArrayList;"); //获得ArrayList类引用
10.
11.     if(list_cls == NULL)
12.     {
13.         cout << "listcls is null \n" ;
14.     }
15.     jmethodID list_costruct = env->GetMethodID(list_cls , "<init>","()V"); //获得得构造函数Id
16.
17.     jobject list_obj = env->NewObject(list_cls , list_costruct); //创建一个ArrayList集合对象
18.     //或得ArrayList类中的 add()方法ID, 其方法原型为: boolean add(Object object) ;
19.     jmethodID list_add = env->GetMethodID(list_cls,"add","(Ljava/lang/Object;)Z");
20.
21.     jclass stu_cls = env->FindClass("Lcom/feixun/jni/Student;"); //获得Student类引用
22.     //获得该类型的构造函数 函数名为 <init> 返回类型必须为 void 即 V
23.     jmethodID stu_costruct = env->GetMethodID(stu_cls , "<init>","(Ljava/lang/String;)V");
24.
25.     for(int i = 0 ; i < 3 ; i++)
26.     {
27.         jstring str = env->NewStringUTF("Native");
28.         //通过调用该对象的构造函数来new 一个 Student实例
29.         jobject stu_obj = env->NewObject(stucls , stu_costruct , 10,str); //构造一个对象
30.
31.         env->CallBooleanMethod(list_obj , list_add , stu_obj); //执行ArrayList类实例的add方法,
           添加一个stu对象
32.     }
33.
34.     return list_obj ;
35. }

```

最后，如何调用这些JNI函数，大家都懂的，直接调用即可，我就不再贴代码了，免得罗嗦。

OK，本次JNI的学习就算告一段落了。下一步该认真仔细学习下Android中JNI的使用了。哎，怎么学的东西又那么多呢？ - -

版权声明：本文为博主原创文章，未经博主允许不得转载。

上一篇

JNI学习积累之二 ---- 数据类型映射、域描述符说明

下一篇

Android中蓝牙的基本使用----BluetoothAdapter类简介

顶

0

踩

0

主题推荐

函数

对象

操作

jni

微软

猜你在找

- Android底层技术：Java层系统服务(Android Service)
- JNI学习四本地方法创建java对象以及对字符串的操作
- ArcGIS for JavaScript
- JNI头文件详解三对象操作
- JavaScript for Qt Quick(QML)
- JNI接口学习二注册函数与操作简单数据类型
- [oeasy]教你玩转java编程-我的世界mc编程入门
- JNI攻略一操作对象的构造方法
- Android必备的Java基础知识(三)
- 利用JNI进行对象操作



查看评论

13楼 工程师WWW 2015-09-15 17:31发表

```
public class HelloJni {  
...  
public native void getAJNString();  
...  
}
```

这个返回类型明显是String吧，代码都是手写不编译的？

12楼 gordge 2015-07-13 10:19发表

写得很好，看了受益匪浅

11楼 ls70868670 2015-03-21 22:12发表

这也能贴上来？
public native void getAJNString();
原型是void，返回是jsting，这不是误认子弟么，楼主你看过代码么？

10楼 MythLove520 2014-12-12 15:45发表

哥们，你好多写法我实际验证都不太对啊

9楼 芝麻李李 2014-07-09 16:51发表

第5个，传递复杂参数时，Student.java和HelloJni.java分别应该放在那里呢？两个放到相同文件夹下，用javac命令编译HelloJni.java时会提示找不到符号，还望博主解惑

8楼 lannan840651 2014-06-24 16:08发表

正好今天工作要用JNI传复杂对象，看了楼主的文档受益匪浅，感谢楼主分享！