

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学习积累用函数大全

分类: Linux学习笔记 Java 2012-05-23 17:40 19685人阅读 评论(3) 收藏 举报

jni

outofmemoryerrr

null

string

methods

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

最近一段时间, 在工作方面比较闲, 分配的Bug不是很多, 于是好好利用这段时间就着源代码看了些许模块, 主要方式还是贼看贼看代码, 同时利用烧机的便利, 加Log观看, 基本上都能弄个脸熟。心里想着该写点什么了? 可是水平不够, 再加上包括很多真正实现地方--中间层, 基本上没看。于是乎, 也就不好卖弄了。

花了几天时间研究了下JNI, 基本上知道如何使用了。照我的观点JNI还是不难的, 难得只是我们一份尝试的心。学习过程中, 发现关于JNI函数资料真的很少, 所谓“工欲善其事, 便先利其器”, 整理出了这份资料, 希望能帮助你克服JNI学习的坎。

主要资料来源: 百度文库的《JNI常用函数》。同时对其加以了补充。

要素 : 1、 该函数大全是基于C语言方式的, 对于C++方式可以直接转换 , 例如, 对于生成一个jstring类型的方法转换分别如下:

C编程环境中使用方法为: (\*env) ->NewStringUTF(env , "123") ;  
C++编程环境中 (例如, VC下) 则是: env ->NewStringUTF( "123") ; (使用起来更简单)

2、 关于下列有些函数中: \*isCopy 的说明, 例如, 如下函数:  
const char\* GetStringUTFChars(JNIEnv\*env, jstring string, jboolean \*isCopy);

对第三个参数 jboolean \*isCopy说明如下:  
当从JNI函数GetStringUTFChars函数中返回得到字符串B时, 如果B是原始字符串 java.lang.String的一份拷贝,  
则isCopy 被赋值为JNI\_TRUE。如果B是和原始字符串指向的是JVM中的同一份数据, 则isCopy 被赋值为JNI\_FALSE。  
当isCopy 为JNI\_FALSE时, 本地代码绝不能修改字符串的内容, 否则JVM中的原始字符串也会被修改, 这会打破Java语言

阅读排行	
Android中View绘制流程	(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	(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	woshimiaoxingren: 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	清澈@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

中字符串不可变的规则。

通常，我们不必关心JVM是否会返回原始字符串的拷贝，只需要为isCopy传递NULL作为参数。

---- 以上内容来自 《JNI编程指南》

南》

一、类操作

jclass **DefineClass** (JNIEnv \*env, jobject loader, const jbyte \*buf, jsize bufLen);

功能：从原始类数据的缓冲区中加载类。

参数： env JNI 接口指针。

loader 分派给所定义的类的类加载器。

buf 包含 .class 文件数据的缓冲区。

bufLen 缓冲区长度。

返回值：返回 Java 类对象。如果出错则返回NULL。

抛出： ClassFormatError 如果类数据指定的类无效。

ClassCircularityError 如果类或接口是自身的超类或超接口。

OutOfMemoryError 如果系统内存不足。

jclass **FindClass** (JNIEnv \*env, const char \*name);

功能：该函数用于加载本地定义的类。它将搜索由CLASSPATH 环境变量为具有指定名称的类所指定的目录和 zip文件。

参数： env JNI 接口指针。

name 类全名（即包名后跟类名，之间由"/"分隔）。如果该名称以"["（数组签名字符）打头，则返回一个数组类。

返回值：返回类对象全名。如果找不到该类，则返回 NULL。

抛出： ClassFormatError 如果类数据指定的类无效。

ClassCircularityError 如果类或接口是自身的超类或超接口。

NoClassDefFoundError 如果找不到所请求的类或接口的定义。

OutOfMemoryError 如果系统内存不足。

jclass **GetObjectClass** (JNIEnv \*env, jobject obj);

功能：通过对象获取这个类。该函数比较简单，唯一注意的是对象不能为NULL，否则获取的class肯定返回也为NULL。

参数： env JNI 接口指针。

obj Java 类对象实例。

jclass **GetSuperclass** (JNIEnv \*env, jclass clazz);

功能：获取父类或者说超类。如果 clazz 代表类class而非类 object，则该函数返回由 clazz 所指定的类的超类。如果 clazz

指定类 object 或代表某个接口，则该函数返回NULL。

参数： env JNI 接口指针。

clazz Java 类对象。

返回值： 由 clazz 所代表的类的超类或 NULL。

jboolean **IsAssignableFrom** (JNIEnv \*env, jclass clazz1, jclass clazz2);

功能：确定 clazz1 的对象是否可安全地强制转换为clazz2。

参数： env JNI 接口指针。

clazz1 第一个类参数。

clazz2 第二个类参数。

返回值： 下列某个情况为真时返回 JNI\_TRUE：

- 1、第一及第二个类参数引用同一个 Java 类。
- 2、第一个类是第二个类的子类。
- 3、第二个类是第一个类的某个接口。

## 二、异常操作

```
jint Throw(JNIEnv *env, jthrowable obj);
```

**功能：**抛出 `java.lang.Throwable` 对象。

**参数：** env JNI 接口指针。

obj java.lang.Throwable 对象。

**返回值：** 成功时返回 0，失败时返回负数。

抛出: java.lang.Throwable 对象 obj。

```
jint ThrowNew (JNIEnv *env , jclass clazz, const char *message);
```

**功能：**利用指定类的消息（由 message 指定）构造异常对象并抛出该异常。

**参数：** env JNI 接口指针。

clazz java.lang.Throwable 的子类。

message 用于构造java.lang.Throwable对象的消息。

**返回值：**成功时返回 0，失败时返回负数。

**抛出：** 新构造的 `java.lang.Throwable` 对象。

jthrowable **ExceptionOccurred** (JNIEnv \*env);

**功能：**确定是否某个异常正被抛出。在平台相关代码调用 `ExceptionClear()` 或 Java 代码处理该异常前，异常将始终保持

抛出状态。

**参数：** env JNI 接口指针。

**返回值：** 返回正被抛出的异常对象，如果当前无异常被抛出，则返回NULL。

```
void ExceptionDescribe (JNIEnv *env);
```

**功能：**将异常及堆栈的回溯输出到系统错误报告信道（例如 `stderr`）。该例程可便利调试操作。

**参数：** env JNI 接口指针。

```
void ExceptionClear (JNIEnv *env);
```

**功能：**清除当前抛出的任何异常。如果当前无异常，则此例程不产生任何效果。

**参数：** env JNI 接口指针。

```
void FatalError (JNIEnv *env, const char *msg);
```

**功能：**抛出致命错误并且不希望虚拟机进行修复。该函数无返回值。

**参数：** env JNI 接口指针。

msg 错误消息。

////////////////////

### 三、全局及局部引用

```

jobject NewGlobalRef (JNIEnv *env, jobject obj);

```

**功能：**创建 obj 参数所引用对象的新全局引用。obj 参数既可以是全局引用，也可以是局部引用。全局引用通过调用

DeleteGlobalRef() 来显式撤消。

**参数：**env JNI 接口指针。

obj 全局或局部引用。

**返回值：** 返回全局引用。如果系统内存不足则返回 NULL。

void **DeleteGlobalRef** (JNIEnv \*env, jobject globalRef);

**功能**：删除 globalRef 所指向的全局引用。

**参数**：env JNI 接口指针。

globalRef 全局引用。

void **DeleteLocalRef** (JNIEnv \*env, jobject localRef);

**功能**：删除 localRef 所指向的局部引用。

**参数**：env JNI 接口指针。

localRef 局部引用。

////////////////////////////////////

#### 四、对象操作

jobject **AllocObject** (JNIEnv \*env, jclass clazz);

**功能**：分配新 Java 对象而不调用该对象的任何构造函数。返回该对象的引用。clazz 参数务必不要引用数组类。

**参数**：env JNI 接口指针。

clazz Java 类对象。

**返回值**：返回 Java 对象。如果无法构造该对象，则返回 NULL。

**抛出**：InstantiationException：如果该类为一个接口或抽象类。

OutOfMemoryError：如果系统内存不足。

jobject **NewObject** (JNIEnv \*env, jclass clazz, jmethodID methodID, ...); //参数附加在函数后面

jobject **NewObjectA** (JNIEnv \*env, jclass clazz, jmethodID methodID, jvalue \*args); //参数以指针形式附加

jobject **NewObjectV** (JNIEnv \*env, jclass clazz, jmethodID methodID, va\_list args); //参数以"链表"形式附加

**功能**：构造新 Java 对象。方法 ID 指示应调用的构造函数方法。**注意：该 ID 特指该类 class 的构造函数 ID，必须通过调用**

**GetMethodID()** 获得，且调用时的方法名必须为 <init>，而返回类型必须为 void (V)。clazz 参数务必不要引用数组类。

**参数**：env JNI 接口指针。

clazz Java 类对象。

methodID 构造函数的方法 ID。

NewObject 的其它参数：传给构造函数的参数，可以为空。

NewObjectA 的其它参数：args：传给构造函数的参数数组。

NewObjectV 的其它参数：args：传给构造函数的参数 va\_list。

**返回值**：返回 Java 对象，如果无法构造该对象，则返回 NULL。

**抛出**：InstantiationException 如果该类为接口或抽象类。

OutOfMemoryError 如果系统内存不足。

构造函数抛出的任何异常。

jclass **GetObjectClass** (JNIEnv \*env, jobject obj);

**功能**：返回对象的类。

**参数**：env JNI 接口指针。

obj Java 对象（不能为 NULL）。

**返回值**：返回 Java 类对象。

jboolean **IsInstanceOf** (JNIEnv \*env, jobject obj, jclass clazz);

**功能：**测试对象是否为某个类的实例。

**参数：** env JNI 接口指针。

obj Java 对象。

clazz Java 类对象。

**返回值：**如果可将 obj 强制转换为 clazz，则返回 JNI\_TRUE。否则返回 JNI\_FALSE。NULL 对象可强制转换为任何类。

jboolean **IsSameObject** (JNIEnv \*env, jobjectref1, jobject ref2);

**功能：**测试两个引用是否引用同一 Java 对象。

**参数：** env JNI 接口指针。

ref1 Java 对象。

ref2 Java 对象。

**返回值：**如果 ref1 和 ref2 引用同一 Java 对象或均为 NULL，则返回 JNI\_TRUE。否则返回 JNI\_FALSE。

////////////////////////////////////

## 五、字符串操作

jstring **NewString** (JNIEnv \*env, const jchar \*unicodeChars, jsize len);

**功能：**利用 Unicode 字符数组构造新的 java.lang.String 对象。

**参数：** env: JNI 接口指针。

unicodeChars: 指向 Unicode 字符串的指针。

len: Unicode 字符串的长度。

**返回值：**Java 字符串对象。如果无法构造该字符串，则为 NULL。

**抛出：** OutOfMemoryError: 如果系统内存不足。

jsize **GetStringLength** (JNIEnv \*env, jstring string);

**功能：**返回 Java 字符串的长度（Unicode 字符数）。

**参数：** env: JNI 接口指针。

string: Java 字符串对象。

**返回值：**Java 字符串的长度。

const jchar \* **GetStringChars** (JNIEnv\*env, jstring string, jboolean \*isCopy);

**功能：**返回指向字符串的 Unicode 字符数组的指针。该指针在调用 ReleaseStringchars() 前一直有效。

如果 isCopy 非空，则在复制完成后将 \*isCopy 设为 JNI\_TRUE。如果没有复制，则设为 JNI\_FALSE。

**参数：** env: JNI 接口指针。

string: Java 字符串对象。

isCopy: 指向布尔值的指针。

**返回值：**指向 Unicode 字符串的指针，如果操作失败，则返回 NULL。

void **ReleaseStringChars** (JNIEnv \*env, jstring string, const jchar \*chars);

**功能：**通知虚拟机平台相关代码无需再访问 chars。参数 chars 是一个指针，可通过 GetStringChars() 从 string 获得。

**参数：** env: JNI 接口指针。

string: Java 字符串对象。

chars: 指向 Unicode 字符串的指针。

jstring **NewStringUTF** (JNIEnv \*env, const char \*bytes);

**功能：**利用 UTF-8 字符数组构造新 java.lang.String 对象。

**参数：** env: JNI 接口指针。如果无法构造该字符串，则为 NULL。

bytes: 指向 UTF-8 字符串的指针。

**返回值:** Java 字符串对象。如果无法构造该字符串, 则为NULL。

**抛出:** OutOfMemoryError: 如果系统内存不足。

jsize **GetStringUTFLength** (JNIEnv \*env, jstring string);

**功能:** 以字节为单位返回字符串的 UTF-8 长度。

**参数:** env: JNI 接口指针。

string: Java 字符串对象。

**返回值:** 返回字符串的 UTF-8

const char\* **GetStringUTFChars** (JNIEnv\*env, jstring string, jboolean \*isCopy);

**功能:** 返回指向字符串的 UTF-8 字符数组的指针。该数组在被ReleaseStringUTFChars() 释放前将一直有效。 如果 isCopy

不是 NULL, \*isCopy 在复制完成后即被设为 JNI\_TRUE。如果未复制, 则设为 JNI\_FALSE。

**参数:** env: JNI 接口指针。

string: Java 字符串对象。

isCopy: 指向布尔值的指针。

**返回值:** 指向 UTF-8 字符串的指针。如果操作失败, 则为 NULL。

void **ReleaseStringUTFChars** (JNIEnv \*env, jstring string, const char \*utf);

**功能:** 通知虚拟机平台相关代码无需再访问 utf。utf 参数是一个指针, 可利用 GetStringUTFChars() 获得。

**参数:** env: JNI 接口指针。

string: Java 字符串对象。

utf: 指向 UTF-8 字符串的指针。

////////////////////////////////////

## 六、数组操作

jsize **GetArrayLength** (JNIEnv \*env, jarray array);

**功能:** 返回数组中的元素数。

**参数:** env: JNI 接口指针。

array: Java 数组对象。

**返回值:** 数组的长度。

jarray **NewObjectArray** (JNIEnv \*env, jsize length, jclass elementClass, jobject initialElement);

**功能:** 构造新的数组, 它将保存类 elementClass 中的对象。所有元素初始值均设为 initialElement。

**参数:** env: JNI 接口指针。

length: 数组大小。

elementClass: 数组元素类。

initialElement: 初始值。 可以为NULL 。

**返回值:** Java 数组对象。如果无法构造数组, 则为 NULL。

**抛出:** OutOfMemoryError: 如果系统内存不足。

**说明:** 使用该函数时, 为了便于易操作性, 我们一般可以用 jobjectArray 数组类型或得返回值, 例如:

```
jobjectArray objArray = env->NewObjectArray ();
```

```
//操作该对象
```

```
env->GetObjectArrayElement (objArray, 0);//获得该object数组在索引0处的值.(可以强制转换类型).
```



jobject **GetObjectArrayElement** (JNIEnv \*env, jobjectArray array, jsize index);

**功能：** 返回 Object 数组的元素。

**参数：** env: JNI 接口指针。  
array: Java 数组。  
index: 数组下标。

**返回值：** Java 对象。

**抛出：** ArrayIndexOutOfBoundsException: 如果 index 不是数组中的有效下标。

void **SetObjectArrayElement** (JNIEnv \*env, jobjectArray array, jsize index, jobject value);

**功能：** 设置 Object 数组的元素。

**参数：** env: JNI 接口指针。  
array: Java 数组。  
index: 数组下标。  
value: 新值。

**抛出：** ArrayIndexOutOfBoundsException: 如果 index 不是数组中的有效下标。

ArrayStoreException: 如果 value 的类不是数组元素类的子类。

### **New<PrimitiveType>Array**方法类型

NativeType **New<PrimitiveType>Array** (JNIEnv \*env, ArrayType array, jboolean\*isCopy);

**说明：** 用于构造新基本类型数组对象的一系列操作。下表说明了特定的基本类型数组构造函数。用户应把

New<PrimitiveType>Array 替换为某个实际的基本类型数组构造函数例程名（见下表），然后将 ArrayType 替换为

该例程相应的数组类型。

**参数：** env : JNI 接口指针。  
length: 数组长度。

**返回值：** Java 数组。如果无法构造该数组，则为 NULL。

<b>New&lt;PrimitiveType&gt;Array 方法组</b>	<b>数组类型</b>
NewBooleanArray()	jbooleanArray
NewByteArray()	jbyteArray
NewCharArray()	jcharArray
NewShortArray()	jshortArray
NewIntArray()	jintArray
NewLongArray()	jlongArray
NewFloatArray()	jfloatArray
NewDoubleArray()	jdoubleArray

### **Get<PrimitiveType>ArrayElements** 方法类型

NativeType \***Get<PrimitiveType>ArrayElements** (JNIEnv \*env, ArrayType array, jboolean\*isCopy);

**说明：** 一组返回基本类型数组体的函数。结果在调用相应的 Release<PrimitiveType>ArrayElements()函数前将一直有效。

由于返回的数组可能是 Java 数组的副本，因此对返回数组的更改不必在基本类型数组中反映出来，直到调用了

Release<PrimitiveType>ArrayElements()。如果 isCopy 不是 NULL，\*isCopy 在复制完成后即被设为 JNI\_TRUE。如果

未复制，则设为 JNI\_FALSE。

**使用说明：**

将 `Get<PrimitiveType>ArrayElements` 替换为表中某个实际的基本类型元素访问器例程名。

将 `ArrayType` 替换为对应的数组类型。

将 `NativeType` 替换为该例程对应的本地类型。

**参数：** `env`：JNI 接口指针。

`array`：Java 字符串对象。

`isCopy`：指向布尔值的指针。

**返回值：** 返回指向数组元素的指针，如果操作失败，则为 `NULL`。

不管布尔数组在 Java 虚拟机中如何表示，`GetBooleanArrayElements()` 将始终返回一个 `jbooleans` 类型的指针，其中每一字节代表一个元素（开包表示）。内存中将确保所有其它类型。

<code>Get&lt;PrimitiveType&gt;ArrayElements</code> 例程	数组类型	本地类型
<code>GetBooleanArrayElements()</code>	<code>jbooleanArray</code>	<code>jboolean</code>
<code>GetByteArrayElements()</code>	<code>jbyteArray</code>	<code>jbyte</code>
<code>GetCharArrayElements()</code>	<code>jcharArray</code>	<code>jchar</code>
<code>GetShortArrayElements()</code>	<code>jshortArray</code>	<code>jshort</code>
<code>GetIntArrayElements()</code>	<code>jintArray</code>	<code>jint</code>
<code>GetLongArrayElements()</code>	<code>jlongArray</code>	<code>jlong</code>
<code>GetFloatArrayElements()</code>	<code>jfloatArray</code>	<code>jfloat</code>
<code>GetDoubleArrayElements()</code>	<code>jdoubleArray</code>	<code>jdouble</code>

**`Release<PrimitiveType>ArrayElements` 方法类型**

`void Release<PrimitiveType>ArrayElements (JNIEnv *env, ArrayType array, NativeType *elems, jint mode);`

**功能：**通知虚拟机平台相关代码无需再访问 `elems` 的一组函数。`elems` 参数是一个通过使用对应的

`Get<PrimitiveType>ArrayElements()` 函数由 `array` 导出的指针。必要时，该函数将把对 `elems` 的修改复制回基本

类型数组。`mode` 参数将提供有关如何释放数组缓冲区的信息。如果 `elems` 不是 `array` 中数组元素的副本，`mode` 将无效。

否则，`mode` 将具有下表所述的功能：

模式	动作
0	复制回内容并释放 <code>elems</code> 缓冲区
<code>JNI_COMMIT</code>	复制回内容但不释放 <code>elems</code> 缓冲区
<code>JNI_ABORT</code>	释放缓冲区但不复制回变化

多数情况下，编程人员将把“0”传给 `mode` 参数以确保固定的数组和复制的数组保持一致。其它选项可以使编程人员进一步控制内存管理，但使用时务必慎重。

**使用说明：**

将 `ArrayType` 替换为对应的数组类型。

将 `NativeType` 替换为该例程对应的本地类型。

**参数：** `env`：JNI 接口指针。

`array`：Java 数组对象。

`elems`：指向数组元素的指针。

`mode`：释放模式。

<code>Release&lt;PrimitiveType&gt;ArrayElements</code> 方法组	数组类型	本地类型
<code>ReleaseBooleanArrayElements()</code>	<code>jbooleanArray</code>	<code>jboolean</code>
<code>ReleaseByteArrayElements()</code>	<code>jbyteArray</code>	<code>jbyte</code>



ReleaseCharArrayElements()	jcharArray	jchar
ReleaseShortArrayElements()	jshortArray	jshort
ReleaseIntArrayElements()	jintArray	jint
ReleaseLongArrayElements()	jlongArray	jlong
ReleaseFloatArrayElements()	jfloatArray	jfloat
ReleaseDoubleArrayElements()	jdoubleArray	jdouble

Get<PrimitiveType>ArrayRegion 方法类型

void **Get<PrimitiveType>ArrayRegion** (JNIEnv \*env, ArrayType array, jsize start, jsize len, NativeType \*buf);

**功能：** 将基本类型数组某一区域复制到缓冲区中的一组函数。

**使用说明：**

将 Get<PrimitiveType>ArrayRegion 替换为下表的某个实际基本类型元素访问器例程名。

将 ArrayType 替换为对应的数组类型。

将 NativeType 替换为该例程对应的本地类型。

**参数：** env： JNI 接口指针。

array： Java 指针。

start： 起始下标。

len： 要复制的元素数。

buf： 目的缓冲区。

**抛出：** ArrayIndexOutOfBoundsException： 如果区域中的某个下标无效。

方法族如下：

Get<PrimitiveType>ArrayRegion方法	数组类型	本地类型
GetBooleanArrayRegion()	jbooleanArray	jboolean
GetByteArrayRegion()	jbyteArray	jbyte
GetCharArrayRegion()	jcharArray	jchar
GetShortArrayRegion()	jshortArray	jshort
GetIntArrayRegion()	jintArray	jint
GetLongArrayRegion()	jlongArray	jlong
GetFloatArrayRegion()	jfloatArray	jfloat
GetDoubleArrayRegion()	jdoubleArray	jdouble

Set<PrimitiveType>ArrayRegion 方法类型

void **Set<PrimitiveType>ArrayRegion** (JNIEnv \*env, ArrayType array, jsize start, jsize len, NativeType \*buf);

**功能：** 将基本类型数组的某一区域从缓冲区中复制回来的一组函数。

**使用说明：** 将 Set<PrimitiveType>ArrayRegion 替换为表中的实际基本类型元素访问器例程名。

将 ArrayType 替换为对应的数组类型。

将 NativeType 替换为该例程对应的本地类型。

**参数：** env： JNI 接口指针。

array： Java 数组。

start： 起始下标。

len： 要复制的元素数。

buf： 源缓冲区。

**抛出：** ArrayIndexOutOfBoundsException： 如果区域中的某个下标无效。

Set<PrimitiveType>ArrayRegion 方法族	数组类型	本地类型
SetBooleanArrayRegion()	jbooleanArray	jboolean

////////////////////////////////////

## 1、实例属性的访问

OutOfMemoryError: 如果系统内存不足。

**参数：** env: JNI 接口指针。  
 obj: Java 对象（不能为 NULL）。  
 fieldID: 有效的域 ID。  
 value: 域的新值。

方法族 如下:

Set<type>Field 方法族	本地类型
SetObjectField()	jobject
SetBooleanField()	jboolean
SetByteField()	jbyte
SetCharField()	jchar
SetShortField()	jshort
SetIntField()	jint
SetLongField()	jlong
SetFloatField()	jfloat
SetDoubleField()	jdouble

2、静态属性的访问 :也存在相同的方法,

```
jfieldID GetStaticFieldID (JNIEnv *env, jclass clazz, const char *name, const char *sig);
NativeType GetStatic<type>Field (JNIEnv*env, jclass classzz, jfieldID fieldID);
void SetStatic<type>Field (JNIEnv *env, jclass classzz, jfieldID fieldID, NativeType
value);
```

它们与实例属性的唯一区别在于第二个参数 **jclass classzz** 代表的是类引用，而不是类实例。

3、调用实例方法

```
jmethodID GetMethodID(JNIEnv *env, jclass clazz, const char *name, const char *sig);
```

**功能：** 返回类或接口实例（非静态）方法的方法 ID。方法可在某个 clazz 的超类中定义，也可从 clazz 继承。该方法由其名称

和签名决定。GetMethodID() 可使未初始化的类初始化。要获得构造函数的方法 ID，应将 <init> 作为方法名，同时将

void (V) 作为返回类型。

**参数：** env: JNI 接口指针。  
 clazz: Java 类对象。  
 name: 方法名。  
 sig: 方法的签名。

**返回值：** 方法 ID，如果找不到指定的方法，则为 NULL。

**抛出：** NoSuchMethodError: 如果找不到指定方法。

ExceptionInInitializerError: 如果由于异常而导致类初始化程序失败。

OutOfMemoryError: 如果系统内存不足。

Call<type>Method 例程 、 Call<type>MethodA 例程 、 Call<type>MethodV 例程

```
NativeType Call<type>Method (JNIEnv*en v, jobject obj, jmethodID methodID, ...); //参数附加在函数后面,
```

```
NativeType Call<type>MethodA (JNIEnv *env, jobject obj, jmethodID methodID, jvalue *args); //参数以指针形式附加
```

```
NativeType Call<type>MethodV (JNIEnv *env, jobject obj, jmethodID methodID, va_list args); //参数以"链表"形式附加
```

**说明：** 这三个操作的方法用于从本地方法调用Java 实例方法。它们的差别仅在于向其所调用的方法传递参数时所用的机制。

这三个操作将根据所指定的方法 ID 调用 Java 对象的实例（非静态）方法。参数 methodID 必须



**参数：** env： JNI 接口指针。  
      clazz： Java 类对象。  
      methods： 类中本地方法和具体实现方法的映射指针。  
      nMethods： 类中的本地方法数。

**返回值：** 成功时返回 "0"；失败时返回负数。

**抛出：** NoSuchMethodError： 如果找不到指定的方法或方法不是本地方法。

jint **UnregisterNatives** (JNIEnv \*env, jclass clazz);

**功能：** 取消注册类的本地方法。类将返回到链接或注册了本地方法函数前的状态。    该函数不应在常规平台相关代码中使用。

相反，它可以为某些程序提供一种重新加载和重新链接本地库的途径。

**参数：** env： JNI 接口指针。  
      clazz： Java 类对象。

**返回值：** 成功时返回“0”；失败时返回负数。

////////////////////////////////////

其实，JNI方面的书籍还是比较少的，建议大家看看 [《JNI编程指南》](#)，算的上个入门书籍吧，指望你能耐心点看。

下一篇，我会继续整理JNI的基础知识，包括类型映射、域签名等；

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 [Android自定义锁屏实现----仿正点闹钟滑屏解锁](#)  
下一篇 [JNI学习积累之二 ---- 数据类型映射、域描述符说明](#)

顶

11

踩

0

主题推荐      函数      jni      微软

猜你在找

- |                                |                                      |
|--------------------------------|--------------------------------------|
| 大数据编程语言：Java基础                 | 基于Android的ELF PLTGOT符号重定向过程及ELF      |
| 微信公众平台深度开发Java版 v2.0（第一季）完整版   | Javah生成JNI头文件出现找不到类的错误               |
| Java基础核心技术：面向对象编程(day05-day07) | javablibrarypath在哪                   |
| 深入浅出Java的反射                    | ndk编译iconv                           |
| 微信公众平台深度开发 (Java版)             | Android NDK编译之undefined reference to |

