不以物喜，不以己悲

.发送私信.请教问题

博客分类

- 数据挖掘&机器学习(0)
- Android基础(24)
- 微信开发之 python(1)
- 数据结构与算法分析(8)
- 数据库(1)
- 设计模式(2)
- 项目管理(5)
- web(2)
- C/C++(2)
- 音频处理(1)
- 手机蓝牙(1)
- 智能家居开发之LNMP(2)
- 印记(4)

阅读排行

1. 1. 关于Android Studio 和 Gradle的那些事儿
2. 2. android.support.v4.widget.DrawerLayout 抽屉效果导航菜单
3. 3. No JVM could be found on your system解决方法
4. 4. android studio 运行代码时识别genymotion设备
5. 5. volley框架下发送和读取cookie
6. 6. 【wav音频解析】之wavread函数的C++实现
7. 7. volley JsonObjectRequest post请求
8. 8. android开发教程（3）— jni编程之采用SWIG从Java调用C/C++

最新评论

- @✌：你想去哦婆婆婆婆婆婆您破破破哦婆婆婆婆在微信我... 查看»
- @liyinggang：可是你的那个shareUtil呢，麻烦楼主能提供下代码... 查看»
- @RenKaidi：好顶赞 查看»
- @RenKaidi：好顶赞 查看»
- @刘备字玄德：楼主我把这个JsonObjectPostRequest复制了然后用... 查看»
- @刘备字玄德：支持 查看»
- @刘小米：引用来自“liz28”的评论想问一下，parseNetwor... 查看»
- @liz28：想问一下，parseNetworkResponse中的response和 ... 查看»
- @liz28：想问一下，parseNetworkResponse中的response和 ... 查看»
- @liz28：想问一下，parseNetworkResponse中的response和 ... 查看»

访客统计

- 今日访问：116
- 昨日访问：360
- 本周访问：476
- 本月访问：4722
- 所有访问：65031

空间 » 博客 » Android基础

转 android开发教程（4）— jni编程之采用 javah 从java调用C++

android java c++ javah

目录[-]

用Java调用C/C++代码

当无法用 Java 语言编写整个应用程序时，JNI 允许您使用本机代码。在下列典型情况下，您可能决定使用本机代码：

- 希望用更低级、更快的编程语言去实现对时间有严格要求的代码。
- 希望从 Java 程序访问旧代码或代码库。
- 需要标准 Java 类库中不支持的依赖于平台的特性。

须知：SWIG和javah的区别（强烈推荐）

我看了网上的关于 jni编程 的教程很多，但不尽相同，刚开始会犯迷糊。我想笔者往往忽略了一个关键点，那就是采用了什么方式决定了步骤的流程。有两种生成 jni的方式：一种是通过SWIG从C++代码生成过度的java代码；另一种是通过javah的方式从java代码自动生成过度的C++代码。两种方式下的步骤流程正好相反。

第一种方式：由于需要配置SWIG环境，有点麻烦了，所以往往大家不采用这个途径，参照我的另一篇博文 http://my.oschina.net/liusicong/blog/314162。

第二种方式：javah的方式则通过shell指令就可以完成整个流程，所以网上的教程也多数是这一类的，本文也是。

从 Java 代码调用 C/C++ 的六个步骤

从 Java 程序调用 C 或 C ++ 代码的过程由六个步骤组成。我们将在下面几页中深入讨论每个步骤，但还是先让我们迅速地浏览一下它们。

1. **编写 Java 代码**。我们将从编写 Java 类开始，这些类执行三个任务：声明将要调用的本机方法；装入包含本机代码的共享库；然后调用该本机方法。
2. **编译 Java 代码**。在使用 Java 类之前，必须成功地将它们编译成字节码。
3. **创建 C/C++ 头文件**。C/C++ 头文件将声明想要调用的本机函数说明。然后，这个头文件与 C/C++ 函数实现（请参阅步骤 4）一起来创建共享库（请参阅步骤 5）。
4. **编写 C/C++ 代码**。这一步实现 C 或 C++ 源代码文件中的函数。C/C++ 源文件必须包含步骤 3 中创建的头文件。
5. **创建共享库文件**。从步骤 4 中创建的 C 源代码文件来创建共享库文件。
6. **运行 Java 程序**。运行该代码，并查看它是否有用。我们还将讨论一些用于解决常见错误的技巧。

最终目录结构如下：

E:.

| Sample1.c

| Sample1.cpp

| Sample1.dll

| Sample1.exp

```
| Sample1.lib
| Sample1.obj
| test_Sample1.h
|
└─test
        Sample1.class
        Sample1.java
```

步骤 1：编写 Java 代码

我们从编写 Java 源代码文件开始，它将声明本机方法（或方法），装入包含本机代码的共享库，然后实际调用本机方法。

这里是名为 Sample1.java 的 Java 源代码文件的示例：

```
package test;
public class Sample1 {
    public native int intMethod(int n);
    public native boolean booleanMethod(boolean bool);
    public native String stringMethod(String text);
    public native int intArrayMethod(int[] intArray);
    public static void main(String[] args) {
        System.loadLibrary("Sample1");
```

```
        Sample1 sample = new Sample1();
        int square = sample.intMethod(5);
        boolean bool = sample.booleanMethod(true);
        String text =sample.stringMethod("JAVA");
        int sum = sample.intArrayMethod(new int[] { 1, 1, 2, 3, 5, 8, 13 });
        System.out.println("intMethod: " + square);
        System.out.println("booleanMethod: " + bool);
        System.out.println("stringMethod: " + text);
        System.out.println("intArrayMethod: " + sum);
    }
}
```

这段代码做了些什么？

首先，请注意对 native 关键字的使用，它只能随方法一起使用。native 关键字告诉 Java 编译器：方法是用 Java 类之外的本机代码实现的，但其声明却在 Java 中。**只能在 Java 类中声明本机方法，而不能实现它**（但是不能声明为抽象的方法，使用native关键字即可），所以本机方法不能拥有方法主体。

现在，让我们逐行研究一下代码：

- 从第 3 行到第 6 行，我们声明了四个 native 方法。
- 在第 10 行，我们装入了包含这些本机方法的实现的共享库文件。（到步骤 5 时，我们将创建该共享库文件。）
- 最终，从第 12 行到第 15 行，我们调用了本机方法。注：这个操作和调用非本机 Java 方法的操作没有差异。

步骤 2：编译 Java 代码

接下来，我们需要将 Java 代码编译成字节码。完成这一步的方法之一是使用随 SDK 一起提供的 Java 编译器 javac。用来将 Java 代码编译成字节码的命令是：

```
cd test

javac Sample1.java
```

步骤 3：创建 C/C++ 头文件

第三步是创建 C/C++ 头文件，它定义本机函数说明。完成这一步的方法之一是使用 javah.exe，它是随 SDK 一起提供的本机方法 C 存根生成器工具。这个工具被设计成用来创建头文件，该头文件为在 Java 源代码文件中所找到的每个 native 方法定义 C 风格的函数。这里使用的命令是：

cd test/..

javah –classpath . test.Sample1

在 Sample1.java 上运行 javah.exe 的结果，生成头文件test_Sample1.h，内容如下：

/* DO NOT EDIT THIS FILE - it ismachine generated */

#include <jni.h>

/* Header for class test_Sample1 */

#ifndef _Included_test_Sample1

#define _Included_test_Sample1

```
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     test_Sample1
 * Method:    intMethod
 * Signature: (I)I
 */
JNIEXPORT jint JNICALLJava_test_Sample1_intMethod
 (JNIEnv *, jobject, jint);


/*
 * Class:     test_Sample1
 * Method:    booleanMethod
 * Signature: (Z)Z
 */
JNIEXPORT jboolean JNICALLJava_test_Sample1_booleanMethod
 (JNIEnv *, jobject, jboolean);
```

```
/*
 * Class:     test_Sample1
 * Method:    stringMethod
 * Signature:(Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALLJava_test_Sample1_stringMethod
 (JNIEnv *, jobject, jstring);

/*
 * Class:     test_Sample1
 * Method:    intArrayMethod
 * Signature: ([I)I
 */
JNIEXPORT jint JNICALLJava_test_Sample1_intArrayMethod
 (JNIEnv *, jobject, jintArray);

#ifdef __cplusplus
```

```
}

#endif

#endif
```

关于 C/C++ 头文件

正如您可能已经注意到的那样，Sample1.h 中的 C/C++ 函数说明和 Sample1.java 中的 Java native 方法声明有很大差异。JNIEXPORT 和 JNICALL 是用于导出函数的、依赖于编译器的指示符。返回类型是映射到 Java 类型的 C/C++ 类型。附录 A：JNI 类型中完整地说明了这些类型。

除了 Java 声明中的一般参数以外，所有这些函数的参数表中都有一个指向 JNIEnv 和 jobject 的指针。指向 JNIEnv 的指针实际上是一个指向函数指针表的指针。正如将要在步骤 4 中看到的，这些函数提供各种用来在 C 和 C++ 中操作 Java 数据的能力。

jobject 参数引用当前对象。因此，如果 C 或 C++ 代码需要引用 Java 函数，则这个 jobject 充当引用或指针，返回调用的 Java 对象。函数名本身是由前缀“Java_”加全限定类名，再加下划线和方法名构成的。

JNI类型

JNI 使用几种映射到 Java 类型的本机定义的 C 类型。这些类型可以分成两类：原始类型和伪类（pseudo-classes）。在 C 中，伪类作为结构实现，而在 C++ 中它们是真正的类。

Java 原始类型直接映射到 C 依赖于平台的类型，如下所示：

C 类型 jarray 表示通用数组。在 C 中，所有的数组类型实际上只是 jobject 的同义类型。但是，在 C++ 中，所有的数组类型都继承了 jarray，jarray 又依次继承了 jobject。下列表显示了 Java 数组类型是如何映射到 JNI C 数组类型的。

这里是一棵对象树，它显示了 JNI 伪类是如何相关的。

步骤 4：编写 C/C++ 代码

当谈到编写 C/C++ 函数实现时，有一点需要牢记：说明必须和 Sample1.h 的函数声明完全一样。我们将研究用于 C 实现和 C++ 实现的完整代码，然后讨论两者之间的差异。

C函数实现

以下是 Sample1.c，它是用 C 编写的实现：

```
#include "test_Sample1.h"
#include <string.h>

JNIEXPORT jint JNICALLJava_test_Sample1_intMethod
   (JNIEnv *env, jobject obj, jint num) {
    return num * num;
  }
JNIEXPORT jboolean JNICALLJava_test_Sample1_booleanMethod
  (JNIEnv *env, jobject obj, jboolean boolean) {
   return!boolean;
}

JNIEXPORT jstring JNICALLJava_test_Sample1_stringMethod
  (JNIEnv *env, jobject obj, jstring string) {
    const char *str = (*env)->GetStringUTFChars(env, string, 0);
     charcap[128];
```

```
    strcpy(cap, str);
    (*env)->ReleaseStringUTFChars(env, string, str);
    return (*env)->NewStringUTF(env, strupr(cap));
}

JNIEXPORT jint JNICALL Java_test_Sample1_intArrayMethod
  (JNIEnv *env, jobject obj, jintArray array) {
     inti, sum = 0;
    jsize len = (*env)->GetArrayLength(env, array);
     jint*body = (*env)->GetIntArrayElements(env, array, 0);
     for(i=0; i<len; i++)
    {   sum += body[i];
     }
    (*env)->ReleaseIntArrayElements(env, array, body, 0);
    return sum;
}

void main(){}
```

C++ 函数实现

以下是 Sample1.cpp (C++ 实现)

```
#include "test_Sample1.h"
#include <string.h>

JNIEXPORT jint JNICALL Java_Sample1_intMethod
  (JNIEnv*env, jobject obj, jint num) {
   return num* num;
}

JNIEXPORT jboolean JNICALLJava_Sample1_booleanMethod
   (JNIEnv *env,jobject obj, jboolean boolean) {
   return!boolean;
}

JNIEXPORT jstring JNICALL Java_Sample1_stringMethod
   (JNIEnv*env, jobject obj, jstring string) {
     constchar *str = env->GetStringUTFChars(string, 0);
```

```
    charcap[128];
    strcpy(cap, str);
    env->ReleaseStringUTFChars(string, str);
    returnenv->NewStringUTF(strupr(cap));
}

JNIEXPORT jint JNICALL Java_Sample1_intArrayMethod
  (JNIEnv*env, jobject obj, jintArray array) {
    int i,sum = 0;
    jsizelen = env->GetArrayLength(array);
    jint*body = env->GetIntArrayElements(array, 0);
    for(i=0; i<len; i++)
    {   sum += body[i];
    }
    env->ReleaseIntArrayElements(array, body, 0);
    returnsum;
}
```

```
void main(){}
```

C 和 C++ 函数实现的比较

唯一的差异在于用来访问 JNI 函数的方法。在 C 中，JNI 函数调用由“(*env)->”作前缀，目的是为了取出函数指针所引用的值。在 C++ 中，JNIEnv 类拥有处理函数指针查找的内联成员函数。下面将说明这个细微的差异，其中，这两行代码访问同一函数，但每种语言都有各自的语法。

C 语法：jsize len = (*env)->GetArrayLength(env,array);

C++ 语法：jsize len =env->GetArrayLength(array);

步骤 5：创建共享库文件

接下来，我们创建包含本机代码的共享库文件。大多数 C 和 C++ 编译器除了可以创建机器代码可执行文件以外，也可以创建共享库文件。用来创建共享库文件的命令取决于您使用的编译器。下面是在 Windows 和 linux 系统上执行的命令。

Windows：使用visual studio commandprompt工具cl.exe

```
cl -I"C:\Program Files\Java\jdk1.6.0_23\include"-I"C:\Program Files\Java\jdk1.6.0_23\include\win32" -LD Sample1.c-FeSample1.dll
```

Linux：使用gcc工具

```
gcc -c -fPIC -I/usr/java/jdk1.6.0_22/include/-I/usr/java/jdk1.6.0_22/include/linux/ Sample1.c

gcc -shared -fPIC -o libSample1.so Sample1.o
```

步骤 6：运行 Java 程序

最后一步是运行 Java 程序，并确保代码正确工作。因为必须在 Java 虚拟机中执行所有 Java 代码，所以需要使用 Java 运行时环境。完

成这一步的方法之一是使用 java，它是随 SDK 一起提供的 Java 解释器。所使用的命令是：

java -cp . test.Sample1

输出：

intMethod: 25

booleanMethod: false

stringMethod: JAVA

intArrayMethod: 33

故障排除

当使用 JNI 从 Java 程序访问本机代码时，您会遇到许多问题。您会遇到的三个最常见的错误是：

- **无法找到动态链接**。它所产生的错误消息是：java.lang.UnsatisfiedLinkError。这通常指无法找到共享库，或者无法找到共享库内特定的本机方法。
- **无法找到共享库文件**。当用 System.loadLibrary(String libname) 方法（参数是文件名）装入库文件时，请确保文件名拼写正确以及没有指定扩展名。还有，确保库文件的位置在类路径中，从而确保 JVM 可以访问该库文件。
- **无法找到具有指定说明的方法**。确保您的 C/C++ 函数实现拥有与头文件中的函数说明相同的说明。

从 C/C++ 程序调用 Java 代码

JNI 允许您从本机代码内调用 Java 类方法。要做到这一点，通常必须使用 Invocation API 在本机代码内创建和初始化一个 JVM。下列是您可能决定从 C/C++ 代码调用Java 代码的典型情况：

- 希望实现的这部分代码是平台无关的，它将用于跨多种平台使用的功能。
- 需要在本机应用程序中访问用 Java 语言编写的代码或代码库。

- 希望从本机代码利用标准 Java 类库。

目录结构：

E:.

readme.txt

Sample2.c

Sample2.class

Sample2.cpp

Sample2.exe

Sample2.java

Sample2.obj

从C/C++ 程序调用 Java 代码的四个步骤

1. **编写 Java 代码**。这个步骤包含编写一个或多个 Java 类，这些类实现（或调用其它方法实现）您想要访问的功能。
2. **编译 Java 代码**。在能够使用这些 Java 类之前，必须成功地将它们编译成字节码。
3. **编写 C/C++ 代码**。这个代码将创建和实例化 JVM，并调用正确的 Java 方法。
4. **运行本机 C/C++ 应用程序**。将运行应用程序以查看它是否正常工作。我们还将讨论一些用于处理常见错误的技巧。

步骤 1：编写Java 代码

我们从编写一个或多个 Java 源代码文件开始，这些文件将实现我们想要本机 C/C++ 代码使用的功能。

下面显示了一个 Java 代码示例Sample2.java：

```
1. publicclass Sample2
 2. {
 3.  public static int intMethod(int n) {
 4.       return n*n;
 5.   }
 6.
 7.  public static boolean booleanMethod(boolean bool) {
 8.        return !bool;
 9.   }
10. }
```

注：Sample2.java 实现了两个 static Java 方法：intMethod(intn) 和 booleanMethod(boolean bool)（分别在第 3 行和第 7 行）。static方法是一种不需要与对象实例关联的类方法。调用 static方法要更容易些，因为不必实例化对象来调用它们。

步骤 2：编译Java 代码

接下来，我们将 Java 代码编译成字节码。完成这一步的方法之一是使用随SDK 一起提供的Java 编译器 javac。使用的命令是：

javacSample2.java

步骤 3：编写 C/C++ 代码

即使是在本机应用程序中运行，所有 Java 字节码也必须在 JVM 中执行。因此 C/C++ 应用程序必须包含用来创建和初始化 JVM 的调

用。为了方便我们，SDK 包含了作为共享库文件（jvm.dll 或 jvm.so）的 JVM，这个库文件可以嵌入到本机应用程序中。

让我们先从浏览一下 C 和 C++ 应用程序的整个代码开始，然后对两者进行比较。

带有嵌入式 JVM的 C 应用程序

Sample2.c 是一个带有嵌入式 JVM 的简单的 C 应用程序：

```
1. #include <jni.h>
2.
3. #ifdef _WIN32
4. #define PATH_SEPARATOR ';'
5. #else
6. #define PATH_SEPARATOR ':'
7. #endif
8.
9. int main()
10. {
11.    JavaVMOption options[1];
12.    JNIEnv *env;
13.    JavaVM *jvm;
```

```
14.  JavaVMInitArgs vm_args;
15.  long status;
16.  jclass cls;
17.  jmethodID mid;
18.  jint square;
19.  jboolean not;
20.
21.  options[0].optionString =  "-Djava.class.path=.";
22.  memset(&vm_args, 0,  sizeof(vm_args));
23.  vm_args.version =  JNI_VERSION_1_2;
24.  vm_args.nOptions = 1;
25.  vm_args.options = options;
26.  status =  JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
27.
28.  if (status != JNI_ERR)
29.  {
30.    cls =  (*env)->FindClass(env, "Sample2");
31.    if(cls !=0)
```

```
32.    { mid =  (*env)->GetStaticMethodID(env, cls, "intMethod",  "(I)I");
33.      if(mid !=0)
34.      { square =  (*env)->CallStaticIntMethod(env, cls, mid, 5);
35.                          printf("Result  of intMethod: %d\n", square);
36.      }
37.
38.      mid = (*env)->GetStaticMethodID(env,  cls, "booleanMethod", "(Z)Z");
39.      if(mid !=0)
40.      { not =  (*env)->CallStaticBooleanMethod(env, cls, mid, 1);
41.        printf("Result of  booleanMethod: %d\n", not);
42.      }
43.    }
44.
45.    (*jvm)->DestroyJavaVM(jvm);
46.    return 0;
47/  }
48.  else
49.  return -1;
```

```
50. }
```

带有嵌入式 JVM的 C++ 应用程序

Sample2.cpp 是一个带有嵌入式 JVM 的 C++ 应用程序：

```
 1. #include <jni.h>
 2. #include  <string.h>
 3. #ifdef _WIN32
 4. #define PATH_SEPARATOR ';'
 5. #else
 6. #define PATH_SEPARATOR ':'
 7. #endif
 8.
 9. int main()
10. {
11.     JavaVMOption options[1];
12.     JNIEnv *env;
13.     JavaVM *jvm;
14.     JavaVMInitArgs vm_args;
```

```
15.     long status;
16.     jclass cls;
17.     jmethodID mid;
18.     jint square;
19.     jboolean not;
20.
21.     options[0].optionString =  "-Djava.class.path=.";
22.     memset(&vm_args, 0, sizeof(vm_args));
23.     vm_args.version = JNI_VERSION_1_2;
24.     vm_args.nOptions = 1;
25.     vm_args.options = options;
26.     status = JNI_CreateJavaVM(&jvm,  (void**)&env, &vm_args);
27.
28.     if (status != JNI_ERR)
29. {
30.       cls =  env->FindClass("Sample2");
31.       if(cls !=0)
32.   {   mid = env->GetStaticMethodID(cls,  "intMethod", "(I)I");
```

```
33.             if(mid !=0)
34.             {   square = env->CallStaticIntMethod(cls, mid, 5);
35.                 printf("Result of intMethod:  %d\n", square);
36.             }
37.
38.             mid = env->GetStaticMethodID(cls,  "booleanMethod", "(Z)Z")
39.             if(mid !=0)
40.             {   not = env->CallStaticBooleanMethod(cls, mid, 1);
41.                 printf("Result of booleanMethod:  %d\n", not);
42.             }
43.     }
44.
45.         jvm->DestroyJavaVM();
46.    return 0;
47.   }
48.       else
49.         return -1;
50. }
```

C 和 C++ 实现的比较

C 和C++ 代码几乎相同；唯一的差异在于用来访问 JNI 函数的方法。在 C 中，为了取出函数指针所引用的值，JNI 函数调用前要加一个 (*env)-> 前缀。在 C++ 中，JNIEnv类拥有处理函数指针查找的内联成员函数。因此，虽然这两行代码访问同一函数，但每种语言都有各自的语法，如下所示。

C 语法： cls = (*env)->FindClass(env, "Sample2");

C++ 语法： cls = env->FindClass("Sample2");

C 语法： mid = (*env)->GetStaticMethodID(env, cls, "intMethod", "(I)I");

C++ 语法： mid = env->GetStaticMethodID(cls, "intMethod", "(I)I");

C 语法： square = env->CallStaticIntMethod(cls, mid, 5);

C++ 语法： square = (*env)->CallStaticIntMethod(env, cls, mid, 5);

C 语法： (*jvm)->DestroyJavaVM(jvm);

C++ 语法： jvm->DestroyJavaVM();

对 C 应用程序更深入的研究

我们刚才编写了许多代码，但它们都做些什么呢？ 在执行步骤 4 之前，让我们更深入地研究一下 C 应用程序的代码。我们将先浏览一些必要的步骤，包括准备本机应用程序以处理 Java 代码、将 JVM 嵌入本机应用程序，然后从该应用程序内找到并调用 Java 方法。

包括 jni.h 文件

我们从 C 应用程序中所包括的 jni.h C 头文件开始，如下面的代码样本中所示：

```
#include <jni.h>
```

jni.h文件包含在 C 代码中所需要的 JNI 的所有类型和函数定义。

声明变量

接下来，声明所有希望在程序中使用的变量。JavaVMOption options[] 具有用于 JVM 的各种选项设置。当声明变量时，确保所声明的 JavaVMOption options[] 数组足够大，以便能容纳您希望使用的所有选项。在本例中，我们使用的唯一选项就是类路径选项。因为在本示例中，我们所有的文件都在同一目录中，所以将类路径设置成当前目录。可以设置类路径，使它指向任何您希望使用的目录结构。

以下代码声明了用于 Sample2.c 的变量：

```
JavaVMOption options[1];

JNIEnv *env;

JavaVM *jvm;

JavaVMInitArgs vm_args;
```

注：

- JNIEnv *env 表示 JNI 执行环境。
- JavaVM jvm 是指向 JVM 的指针。我们主要使用这个指针来创建、初始化和销毁 JVM。
- JavaVMInitArgs vm_args 表示可以用来初始化 JVM 的各种 JVM 参数。

设置初始化参数

JavaVMInitArgs 结构表示用于 JVM 的初始化参数。在执行 Java 代码之前，可以使用这些参数来定制运行时环境。正如您所见，这些选项是一个参数而 Java 版本是另一个参数。按如下所示设置了这些参数：

```
vm_args.version = JNI_VERSION_1_2;

vm_args.nOptions = 1;

vm_args.options = options;
```

设置类路径

接下来，为 JVM 设置类路径，以使它能找到所需要的 Java 类。在这个特定示例中，因为 Sample2.class 和Sample2.exe 都位于同一目

录中，所以将类路径设置成当前目录。我们用来为 Sample2.c 设置类路径的代码如下所示：

```
options[0].optionString = "-Djava.class.path=.";
// same text as command-line options for the java.exe JVM
```

为 vm_args 留出内存

在可以使用 vm_args 之前，必需为它留出一些内存。一旦设置了内存，就可以设置版本和选项参数了，如下所示：

```
    memset(&vm_args, 0, sizeof(vm_args));  // set aside enough memory for vm_args
    vm_args.version =  JNI_VERSION_1_2;          // version of  Java platform
    vm_args.nOptions = 1;                        // same as size of  options[1]
    vm_args.options = options;
```

创建 JVM

处理完所有设置之后，现在就准备创建 JVM 了。先从调用方法开始：

```
JNI_CreateJavaVM(JavaVM **jvm, void** env, JavaVMInitArgs **vm_args)
```

如果成功，则这个方法返回零，否则，如果无法创建 JVM，则返回JNI_ERR。

查找并装入 Java 类

一旦创建了 JVM 之后，就可以准备开始在本机应用程序中运行 Java 代码。首先，需要使用FindClass() 函数查找并装入 Java 类，如下所示：

```
cls = (*env)->FindClass(env, "Sample2");
```

cls 变量存储执行FindClass() 函数后的结果。如果找到该类，则 cls 变量表示该Java 类的句柄。如果不能找到该类，则 cls 将为零。

查找 Java 方法

接下来，我们希望用 GetStaticMethodID() 函数在该类中查找某个方法。我们希望查找方法 intMethod，它接收一个 int 参数并返回一个 int。以下是查找 intMethod 的代码：

```
mid = (*env)->GetStaticMethodID(env, cls, "intMethod", "(I)I");
```

mid 变量存储执行 GetStaticMethodID() 函数后的结果。如果找到了该方法，则 mid 变量表示该方法的句柄。如果不能找到该方法，则 mid 将为零。

请记住，在本示例中，我们正在调用 static Java 方法。 那就是我们使用 GetStaticMethodID() 函数的原因。GetMethodID() 函数与 GetStaticMethodID() 函数的功能一样，但它用来查找实例方法。

如果正在调用构造器，则方法的名称为“<init>”。 要了解更多关于调用构造器的知识，请参阅错误处理。 要了解更多关于用来指定参数类型的代码以及关于如何将 JNI 类型映射到 Java 原始类型的知识，请参阅附录 。

调用 Java 方法

最后，我们调用 Java 方法，如下所示：

```
square = (*env)->CallStaticIntMethod(env, cls, mid, 5);
```

CallStaticIntMethod() 方法接受 cls（表示类）、mid（表示方法）以及用于该方法一个或多个参数。在本例中参数是 int 5。

您还会遇到 CallStaticXXXMethod() 和 CallXXXMethod() 之类的方法。这些方法分别调用静态方法和成员方法，用方法的返回类型（例如，Object、Boolean、Byte、Char、Int、Long 等等）代替变量XXX。

步骤 4：运行应用程序

现在准备运行这个 C 应用程序，并确保代码正常工作。当运行 Sample2.exe 时，应该可以得到如下结果：

windows：

```
// 使用visual studio command  prompt工具cl.exe编译生成Sample2.exe执行文件

cl -Ic:"C:\Program  Files\Java\jdk1.6.0_23\include"  -Ic:"C:\Program Files\Java\jdk1.6.0_23\include\win32" Sample2.c
 "C:\Program Files\Java\jdk1.6.0_23\lib\jvm.lib"


// 运行

E:\Sample2>Sample2.exe

Result of intMethod: 25

Result of booleanMethod:  0
```

linux：

export LD_LIBRARY_PATH=.:/usr/java/jdk1.6.0_22/jre/lib/amd64/server:/usr/java/jdk1.6.0_22/jre/lib:/usr/java/jdk1.6.0_22/jre/bin

gcc -I/usr/java/jdk1.6.0_22/include/-I/usr/java/jdk1.6.0_22/include/linux/-L/usr/java/jdk1.6.0_22/jre/lib/amd64/server/ -ljvm Sample2.c -o Sample2.run

./ Sample2

故障排除

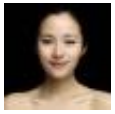
JNI 的 Invocation API 有点麻烦，因为它是用 C 语言定义的，而 C 语言基本上不支持面向对象编程。结果是，它很容易遇到问题。下面是一份检查表，它可能有助于您避免一些较常见的错误。

- 请总是确保正确设置了引用。例如，当使用 JNI_CreateJavaVM() 方法创建 JVM 时，确保它返回零。还请确保，在使用 FindClass() 和 GetMethodID() 方法之前，它们的引用设置不是零。
- 请检查方法名是否拼写正确以及是否适当地转换了方法说明。还请确保，对静态方法使用了 CallStaticXXXMethod() 以及对成员方法使用了 CallXXXMethod()。
- 确保使用任何 Java 类所需的特殊的参数或选项来初始化 JVM。例如，如果 Java 类需要大量内存，则可能需要增加堆的最大大小选项。
- 请总是确保正确设置了类路径。使用嵌入式 JVM 的本机应用程序必须能够找到 jvm.dll 或 jvm.so 共享库。
-

原文地址：http://www.cnblogs.com/BloodAndBone/archive/2010/12/22/1913882.html

- « 上一篇
- 下一篇 »

评论1

- 
1楼：刘小米 发表于 2014-09-22 14:37 回复此评论
步骤三中编写C/C++，通常需要在VS环境下编辑一个与头文件同名的.cpp文件，然后拷贝到你的工程目录下就可以了。

插入： 表情 开源软件

发表评论

关闭插入表情

关闭相关文章阅读

- 2014/09/16android开发教程（3）— jni编程之采...
- 2012/10/31java 调用c/c++
- 2013/07/08android c++ 通信
- 2014/04/25java与c++
- 2012/04/09C++之父评论C++与Java