挠痒痒

JNI编程(二) —— 让C++和Java相互调用(1)

博客分类：
- Java

编程JNIJavaC++

自己在外面偷偷的算了下，又有将近两个月没更新过blog了。趁着今天有兴致，来更新JNI编程的第二篇文章。在第一篇里，大概介绍了JNI的特点、用途和优劣。并且做一个最简单的JNI的例子，不过说实话那个例子在实际的开发中没有太大的价值，实际开发中所需要的JNI程序要远远比那个复杂。所以这一篇就来介绍下如何通过JNI实现java和C++的相互通信，来满足实际开发的需要。

所谓”通信“，其实说白了无非也就是我们所说的方法调用，在上一篇的例子里介绍了如何在Java中调用本地的DLL，其实在Java代码中，除了对本地方法标注native关键字和加上要加载动态链接库之外，JNI基本上是对上层coder透明的，上层coder调用那些本地方法的时候并不知道这个方法的方法体究竟是在哪里，这个道理就像我们用JDK所提供的API一样。所以在Java中使用JNI还是很简单的，相比之下在C++中调用java，就比前者要复杂的多了。

现在来介绍下JNI里的数据类型。在C++里，编译器会很据所处的平台来为一些基本的数据类型来分配长度，因此也就造成了平台不一致性，而这个问题在Java中则不存在，因为有JVM的缘故，所以Java中的基本数据类型在所有平台下得到的都是相同的长度，比如int的宽度永远都是32位。基于这方面的原因，java和c++的基本数据类型就需要实现一些mapping，保持一致性。下面的表可以概括：

Java类型	本地类型	JNI中定义的别名
int	long	jint
long	_int64	jlong
byte	signed char	jbyte
boolean	unsigned char	jboolean
char	unsigned short	jchar
short	short	jshort
float	float	jfloat
double	double	jdouble
Object	_jobject*	jobject

上面的表格是我在网上搜的，放上来给大家对比一下。对于每一种映射的数据类型，JNI的设计者其实已经帮我们取好了相应的别名以方便记忆。如果想了解一些更加细致的信息，可以去看一些jni.h这个头文件，各种数据类型的定义以及别名就被定义在这个文件中。

了解了JNI中的数据类型，下面就来看这次的例子。这次我们用Java来实现一个前端的market（以下就用Foreground代替）用CPP来实现一个后端factory（以下用backend代替）。我们首先还是来编写包含本地方法的java类。

Java代码

```
1.  package com.chnic.service;
2.
3.  import com.chnic.bean.Order;
4.
5.  public class Business {
6.      static{
7.          System.loadLibrary("FruitFactory");
8.      }
9.
10.     public Business(){
11.
12.     }
13.
14.     public native double getPrice(String name);
15.     public native Order getOrder(String name, int amount);
16.     public native Order getRamdomOrder();
17.     public native void analyzeOrder(Order order);
18.
19.     public void notification(){
20.         System.out.println("Got a notification.");
21.     }
22.
23.     public static void notificationByStatic(){
24.         System.out.println("Got a notification in a static method.");
25.     }
26. }
```

这个类里面包含4个本地方法，一个静态初始化块加载将要生成的dll文件。剩下的方法都是很普通的java方法，等会在backend中回调这些方法。这个类需要一个名为Order的JavaBean。

Java代码

```
1.  package com.chnic.bean;
2.
3.  public class Order {
4.
5.      private String name = "Fruit";
6.      private double price;
7.      private int amount = 30;
8.
9.      public Order(){
10.
```

```
    }

    public int getAmount() {
        return amount;
    }

    public void setAmount(int amount) {
        this.amount = amount;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

JavaBean中，我们为两个私有属性赋值，方便后面的例子演示。到此为止除了测试代码之外的Java端的代码就全部高调了，接下来进行生成.h头文件、建立C++工程的工作，在这里就一笔带过，不熟悉的朋友请回头看第一篇。在工程里我们新建一个名为Foctory的C++ source file 文件，去实现那些native方法。具体的代码如下。

Cpp代码

```
#include <iostream.h>
#include <string.h>
#include "com_chnic_service_Business.h"

jobject getInstance(JNIEnv* env, jclass obj_class);

JNIEXPORT jdouble JNICALL Java_com_chnic_service_Business_getPrice(JNIEnv* env,
                                                                   jobject obj,
                                                                   jstring name)
{
    const char* pname = env->GetStringUTFChars(name, NULL);
    cout << "Before release: "  << pname << endl;

    if (strcmp(pname, "Apple") == 0)
    {
        env->ReleaseStringUTFChars(name, pname);
        cout << "After release: " << pname << endl;
        return 1.2;
    }
    else
    {
        env->ReleaseStringUTFChars(name, pname);
        cout << "After release: " << pname << endl;
```

```
        return 2.1;
    }
}


JNIEXPORT jobject JNICALL Java_com_chnic_service_Business_getOrder(JNIEnv* env,
                                                                   jobject obj,
                                                                   jstring name,
                                                                   jint amount)
{
    jclass order_class = env->FindClass("com/chnic/bean/Order");
    jobject order = getInstance(env, order_class);

    jmethodID setName_method = env->GetMethodID(order_class, "setName", "(Ljava/lang/String;)V");
    env->CallVoidMethod(order, setName_method, name);

    jmethodID setAmount_method = env->GetMethodID(order_class, "setAmount", "(I)V");
    env->CallVoidMethod(order, setAmount_method, amount);

    return order;
}

JNIEXPORT jobject JNICALL Java_com_chnic_service_Business_getRamdomOrder(JNIEnv* env,
                                                                         jobject obj)
{
    jclass business_class = env->GetObjectClass(obj);
    jobject business_obj = getInstance(env, business_class);

    jmethodID notification_method = env->GetMethodID(business_class, "notification", "()V");
    env->CallVoidMethod(obj, notification_method);

    jclass order_class = env->FindClass("com/chnic/bean/Order");
    jobject order = getInstance(env, order_class);
    jfieldID amount_field = env->GetFieldID(order_class, "amount", "I");
    jint amount = env->GetIntField(order, amount_field);
    cout << "amount: " << amount << endl;
    return order;
}


JNIEXPORT void JNICALL Java_com_chnic_service_Business_analyzeOrder (JNIEnv* env,
                                                                     jclass cls,
                                                                     jobject obj)
{
    jclass order_class = env->GetObjectClass(obj);
    jmethodID getName_method = env->GetMethodID(order_class, "getName", "()Ljava/lang/String;");
    jstring name_str = static_cast<jstring>(env->CallObjectMethod(obj, getName_method));
    const char* pname = env->GetStringUTFChars(name_str, NULL);

    cout << "Name in Java_com_chnic_service_Business_analyzeOrder: " << pname << endl;
    jmethodID notification_method_static = env->GetStaticMethodID(cls, "notificationByStatic", "()V");
    env->CallStaticVoidMethod(cls, notification_method_static);

}
```

```
79.  jobject getInstance(JNIEnv* env, jclass obj_class)
80.  {
81.      jmethodID construction_id = env->GetMethodID(obj_class, "<init>", "()V");
82.      jobject obj = env->NewObject(obj_class, construction_id);
83.      return obj;
84.  }
```

可以看到，在我Java中的四个本地方法在这里全部被实现，接下来针对这四个方法来解释下，一些JNI相关的API的使用方法。先从第一个方法讲起吧：

1.getPrice(String name)

这个方法是从foreground传递一个类型为string的参数到backend，然后backend判断返回相应的价格。在cpp的代码中，我们用GetStringUTFChars这个方法来把传来的jstring变成一个UTF-8编码的char型字符串。因为jstring的实际类型是jobject，所以无法直接比较。

GetStringUTFChars方法包含两个参数，第一参数是你要处理的jstring对象，第二个参数是否需要在内存中生成一个副本对象。将jstring转换成为了一个const char*了之后，我们用string.h中带strcmp函数来比较这两个字符串，如果传来的字符串是“Apple”的话我们返回1.2。反之返回2.1。在这里还要多说一下ReleaseStringUTFChars这个函数，这个函数从字面上不难理解，就是释放内存用的。有点像cpp里的析构函数，只不过Sun帮我们已经封装好了。由于在JVM中有GC这个东东，所以多数java coder并没有写析构的习惯，不过在JNI里是必须的了，否则容易造成内存泄露。我们在这里在release之前和之后分别打出这个字符串来看一下效果。

粗略的解释完一些API之后，我们编写测试代码。

Java代码

```
1.  Business b = new Business();
2.  System.out.println(b.getPrice("Apple"));
```

运行这段测试代码，控制台上打出

```
Before release: Apple
After release: ??
1.2
```

在release之前打印出来的是我们“需要”的Apple，release之后就成了乱码了。由于传递的是Apple，所以得到1.2。测试成功。

2. getOrder(String name, int amount)

在foreground中可以通过这个方法让backend返回一个你“指定”的Order。所谓“指定”，其实也就是指

方法里的两个参数：name和amout，在cpp的代码在中，会根据传递的两个参数来构造一个Order。回到cpp的代码里。

Java代码

```
1.  jclass order_class = env->FindClass("com/chnic/bean/Order");
```

是不是觉得这句代码似曾相识？没错，这句代码很像我们java里写的Class.forName(className)反射的代码。其实在这里FindClass的作用和上面的forName是类似的。只不过在forName中要用完整的类名，但是在这里必须用"/"来代替“.”。这个方法会返回一个jclass的对象，其实也就是我们在Java中说的类对象。

Java代码

```
1.  jmethodID construction_id = env->GetMethodID(obj_class, "<init>", "()V");
2.  jobject obj = env->NewObject(obj_class, construction_id);
```

拿到"类对象"了之后，按照Java RTTI的逻辑我们接下来就要唤醒那个类对象的构造函数了。在JNI中，包括构造函数在内的所有方法都被看成Method。每个method都有一个特定的ID，我们通过GetMethodID这个方法就可以拿到我们想要的某一个java 方法的ID。GetMethodID需要传三个参数，第一个是很显然jclass，第二个参数是java方法名，也就是你想取的method ID的那个方法的方法名（有些绕口 ），第三个参数是方法签名。

在这里有必要单独来讲一讲这个方法签名，为什么要用这个东东呢？我们知道，在Java里方法是可以被重载的，比如我一个类里有public void a(int arg)和public void a(String arg)这两个方法，在这里用方法名来区分方法显然就是行不通的了。方法签名包括两部分：参数类型和返回值类型；具体的格式：(参数1类型签名 参数2类型签名)返回值类型签名。下面是java类型和年名类型的对照的一个表

Java类型	对应的签名
boolean	Z
byte	B
char	C
shrot	S
int	I
long	L
float	F
double	D
void	V
Object	L用/分割包的完整类名; Ljava/lang/String;
Array	[签名 [I [Ljava/lang/String;

其实除了自己对照手写之外，JDK也提供了一个很好用的生成签名的工具javap,cmd进入控制台到你要生成签名的那个类的目录下。在这里用Order类打比方，敲入： javap -s -private Order。 所有方法签名都会被输出，关于javap的一些参数可以在控制台下面输入 javap -help查看。（做coder的 毕竟还是要认几个单词的）

啰嗦了一大堆，还是回到我们刚刚的getMethodID这个方法上。因为是调用构造函数，JNI规定调用构造函数的时候传递的方法名应该为<init> ,通过javap查看 我们要的那个无参的构造函数的方法签是()V。得到方法签名，最后我们调用NewObject方法来生成一个新的对象。

拿到了对象，之后我们开始为对象jobject填充数值，还是首先拿到setXXX方法的Method ID，之后调用Call<Type>Method来调用java方法。这里的<Type>所指的是方法的返回类型，我们刚刚调用的是set方法的返回值是void，因此这里的方法也就是CallVoidMethod，这个方法的参数除了前两个要传入jobject和jmethodID之外还要传入要调用的那个方法的参数，而且要顺序必须一致，这点和Java的反射一模一样，在这里就不多解释。（看到这一步是不是对java 反射又有了自己新的理解？）

终于介绍完了第二个方法，下来就是测试代码测试。

Java代码

```
1.  Order o = b.getOrder("Watermelom", 100);
2.  System.out.println("java: " + o.getName());
3.  System.out.println("java: " + o.getAmount());
```

控制台打出

```
java: Watermelom
java: 100
```

就此，我们完成了第二个方法的测试。

鉴于篇幅的关系，第三个 第四个方法的解释放到下一篇里解释。突然发现这篇写的貌似有点长了，再写下去的话臭鸡蛋和番茄就飘过来了 ， 具体的代码也会在下一篇里上传。

分享到：
JNI编程(二) —— 让C++和Java相互调用 ... | shell script II

- 2008-08-14 17:44
- 浏览 21187
- 评论(3)
- 相关推荐

评论

3 楼 ostrichmyself 2010-07-27
总结得不错~
2 楼 C_J 2009-05-13

```
jstring name_str = static_cast<jstring>(env->CallObjectMethod(obj, getName_method));
```

你这里不需要做内存释放么？
1 楼 甜菜侯爵 2008-08-22
这么好的文章没人发掘么？

赞一个

发表评论

您还没有登录，请您登录后再发表评论

chnic

- 浏览: 98326 次
- 性别:
- 来自: 北京
- 我现在离线

最近访客 更多访客>>

liuxinglanyue

weixinwei

qq243142247

bety

文章分类

- 全部博客 (21)
- Java (6)
- Webservice (5)
- DB & SQL (0)
- AJAX & JAVASCRIPT (1)
- Application Server (1)
- UNIX & Linux (2)
- RCP (5)

社区版块

- 我的资讯 (0)
- 我的论坛 (1)
- 我的问答 (0)

存档分类

- 2015-04 (6)
- 2014-05 (1)
- 2008-09 (1)
- 更多存档...

最新评论

- WLLT： 我也是 估计少包
  利用AXIS开发Webservice(四) —— 如何抛出自定义异常
- kilery_019： zgl217 写道发布的时候 tomcat服务器报错：- Un ...
  利用AXIS开发Webservice(三) —— 如何传递JavaBean和你的对象
- zgl217： 发布的时候 tomcat服务器报错：- Unable to d ...
  利用AXIS开发Webservice(三) —— 如何传递JavaBean和你的对象
- lxdhq1011： 你好，我想问一下android调用webservice，如何调 ...
  利用AXIS开发Webservice(一) —— 如何发布自己的webservice
- zhongaili520： 调用第三方提供的dll库，怎么实现
  JNI编程(一) —— 编写一个最简单的JNI程序
