# JNI 访问 Java 中非静态方法

讲解使用javap 命令

# JNI访问Java中的静态方法

在这之前，先回顾一下在Java中调用一个方法时在JVM中的实现原理，有助于下面讲解本地代码调用Java方法实现的机制。写过Java的童鞋都知道，调用一个类的静态方法，直接通过 **类名.方法** 就可以调用。这也太简单了，有什么好讲的呢。。。但在这个调用过程中，JVM是帮我们做了很多工作的。当我们在运行一个Java程序时，JVM会先将程序运行时所要用到所有**相关的class**文件加载到JVM中，并采用按需加载的方式加载，也就是说某个类只有在被用到的时候才会被加载，这样设计的目的也是为了提高程序的性能和节约内存。所以我们在用类名调用一个静态方法之前，JVM首先会判断该类是否已经加载，如果没有被**ClassLoader**加载到JVM中，JVM会从classpath路径下查找该类，如果找到了，会将其加载到JVM中，然后才是调用该类的静态方法。如果没有找到，JVM会抛出java.lang.ClassNotFoundException异常，提示找不到这个类。ClassLoader是JVM加载class字节码文件的一种机制，不太了解的童鞋，请移步阅读[《深入分析Java ClassLoader原理》](http://blog.csdn.net/xyang81/article/details/7292380)一文。其实在JNI开发当中，本地代码也是按照上面的流程来访问类的静态方法或实例方法的

讲解使用的公式。

# JNI访问Java中构造方法

讲解为什么要使用构造方法。

# JNI访问Java调用父类方法

如果一个方法被定义在父类中，在子类中被覆盖，也可以调用父类中的这个实例方法。JNI 提供了一系列函数CallNonvirtualXXXMethod来支持调用各种返回值类型的实例方法。

# JNI 中文字符串乱码

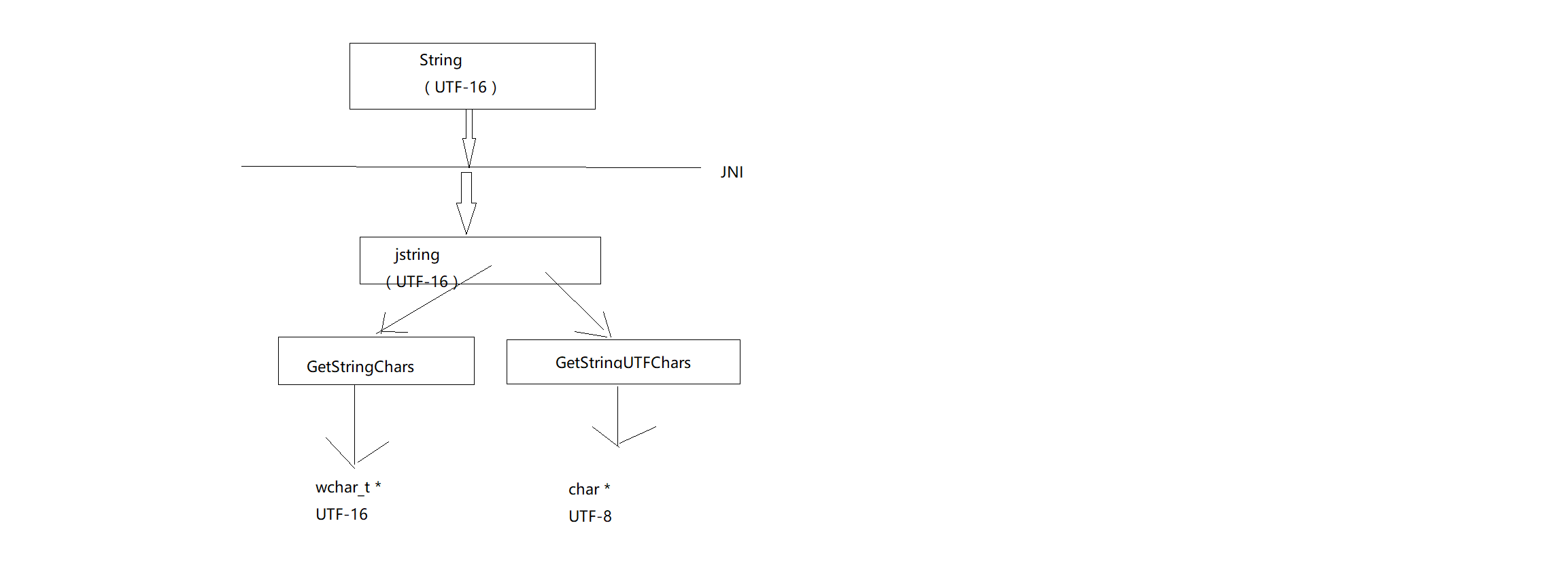
1. JNI打印java中的中文，JNI中的中文提供给Java 使用

中文字符串乱码：

前提基础知识

1. Java 内部是使用16bit的unicode编码（utf-16）来表示字符串的，无论英文还是中文都是2字节
2. Jni内部是使用utf8 编码来表示字符串的，utf8是变长编码的unicode，一般ascii 字符是字节，中文是3字节
3. c/C++ 使用的是原始数据，ascii就是一个字节，中文一般是GB2312编码，用2个字节表示一个汉字。

在我们的系统中，java string 转换从C sting 的流程如下：



为什么英文的String 没有乱码，而中文的String 有乱码？

因为如果java 的string 不含中文字符，只有标准的ascii 码，那么我们使用GetStringUTFChars或者 NewStringUTF就可以搞定了，因为这种情况下，UTF-8 编码和ascii编码是一致的，不需要转换。

然后如果字符串中有中文字符，那么在C /C++ 部分进行编码转换就是一个必须的了。我们需要将UTF8/16 编码转换成GB2312；然后GB2312 转成UTF-8/16。

在window系统上

举例说明

在linux 系统上

**iconv**库是一个免费的独立的编码转换库，支持很多平台，多种编码（事实上，它几乎可以处理我们所使用的所有字符编码），而且它的行为不受任何外部环境的影响

另外一种方式：利用java的特点

在java String 中有一个方法来转中文到对应的编码方式的函数。

**public** String(**byte** bytes[], String charsetName)

要访问这个函数我们在C里面有几个前提条件

1. jmethodID
2. byte数组
3. 字符编码

第三个参数的说明：

**GetStringUTFChars(env, j\_str, &isCopy)**

**isCopy：**取值JNI\_TRUE和JNI\_FALSE，如果值为JNI\_TRUE，表示返回JVM内部源字符串的一份拷贝，并为新产生的字符串分配内存空间。如果值为JNI\_FALSE，表示返回JVM内部源字符串的指针，意味着可以通过指针修改源字符串的内容，不推荐这么做，因为这样做就打破了Java字符串不能修改的规定。但我们在开发当中，并不关心这个值是多少，通常情况下这个参数填NULL即可。

# JNI访问java中的数组