

# 1.本地方法

“A native method is a Java method whose implementation is provided by non-java code.”（本地方法是一个非Java的方法，它的具体实现是非Java代码的实现）

简单地讲，一个Native Method是一个Java调用非Java代码的接囗

一个Native Method是这样一个Java方法：该方法的实现由非Java语言实现，比如C。

这个特征并非Java所特有，很多其它的编程语言都有这一机制，比如在C++中，你可以用extern 告知C++编译器去调用一个C的函数。

在定义一个native method时，并不提供实现体（有些像定义一个Java interface），因为其实现体是由非java语言在外面实现的。

本地接口的作用是融合不同的编程语言为Java所用，它的初衷是融合C/C++程序。

native方法举例：

|  |
| --- |
| 1.**Object 类的 getClass() 方法**  public final **native** Class<?> getClass();  **Thread 类的 start() 方法**  **public synchronized void start() {**  **/\*\***  **\* This method is not invoked for the main method thread or "system"**  **\* group threads created/set up by the VM. Any new functionality added**  **\* to this method in the future may have to also be added to the VM.**  **\***  **\* A zero status value corresponds to state "NEW".**  **\*/**  **if (threadStatus != 0)**  **throw new IllegalThreadStateException();**  **/\* Notify the group that this thread is about to be started**  **\* so that it can be added to the group's list of threads**  **\* and the group's unstarted count can be decremented. \*/**  **group.add(this);**  **boolean started = false;**  **try {**  **start0();**  **started = true;**  **} finally {**  **try {**  **if (!started) {**  **group.threadStartFailed(this);**  **}**  **} catch (Throwable ignore) {**  **/\* do nothing. If start0 threw a Throwable then**  **it will be passed up the call stack \*/**  **}**  **}**  **}**  **private native void start0();** |

* 标识符native可以与其它java标识符连用，但是abstract除外
* 为什么要使用 Native Method？

Java使用起来非常方便，然而有些层次的任务用Java实现起来不容易，或者我们对程序的效率很在意时，问题就来了。

* 与Java环境的交互

有时Java应用需要与Java外面的硬件环境交互，这是本地方法存在的主要原因。你可以想想Java需要与一些底层系统，如操作系统或某些硬件交换信息时的情况。

本地方法正是这样一种交流机制：它为我们提供了一个非常简洁的接口，而且我们无需去了解Java应用之外的繁琐的细节。

* 与操作系统的交互

JVM支持着Java语言本身和运行时库，它是Java程序赖以生存的平台，它由一个解释器（解释字节码）和一些连接到本地代码的库组成。

然而不管怎样，它毕竟不是一个完整的系统，它经常依赖于一底层系统的支持。这些底层系统常常是强大的操作系统。

通过使用本地方法，我们得以用Java实现了jre的与底层系统的交互，甚至JVM的一些部分就是用C写的。

还有，如果我们要使用一些Java语言本身没有提供封装的操作系统的特性时，我们也需要使用本地方法。

* Sun’s Java

Sun的解释器是用C实现的，这使得它能像一些普通的C一样与外部交互。jre大部分是用Java实现的，它也通过一些本地方法与外界交互。

例如：类java.lang.Thread的setPriority()方法是用Java实现的，但是它实现调用的是该类里的本地方法setPriority0()。这个本地方法是用C实现的，并被植入JVM内部在Windows 95的平台上，这个本地方法最终将调用Win32 setpriority() API。

这是一个本地方法的具体实现由JVM直接提供，更多的情况是本地方法由外部的动态链接库（external dynamic link library）提供，然后被JVM调用。

* 本地方法的现状

目前该方法使用的越来越少了，除非是与硬件有关的应用，比如通过Java程序驱动打印机或者Java系统管理生产设备，在企业级应用中已经比较少见。因为现在的异构领域间的通信很发达，比如可以使用Socket通信，也可以使用Web Service等等，不多做介绍。

# 2.本地方法栈

1.Java虚拟机栈用于管理Java方法的调用，而本地方法栈用于管理本地方法（一般非Java实现的方法）的调用

2.本地方法栈，也是线程私有的。

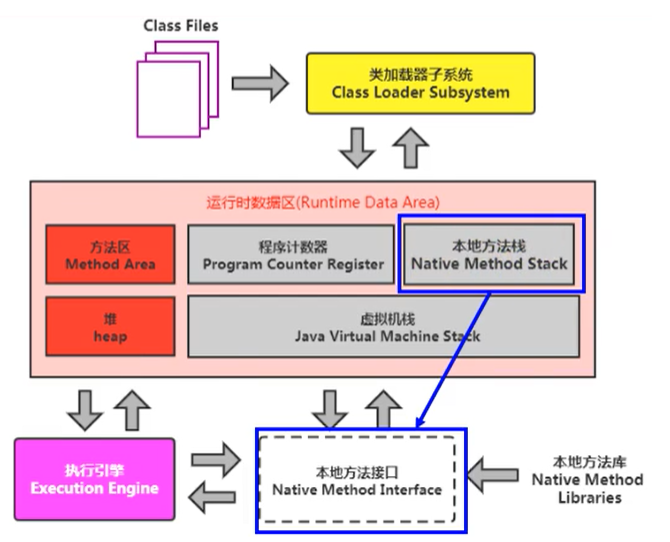
3.允许被实现成固定或者是可动态拓展的内存大小。（和Java虚拟机栈在内存溢出方面情况是相同的）

* 如果线程请求分配的栈容量超过本地方法栈允许的最大容量，Java虚拟机将会抛出一个StackOverFlowError异常。
* 如果本地方法栈可以动态扩展，并且在尝试扩展的时候无法申请到足够的内存，或者在创建新的线程时没有足够的内存去创建对应的本地方法栈，那么java虚拟机将会抛出一个OutOfMemoryError异常。

4.本地方法是使用C语言实现的

5.它的具体做法是Native Method Stack中登记native方法，在Execution Engine执行时加载本地方法库。

6.当某个线程调用一个本地方法时，它就进入了一个全新的并且不再受虚拟机限制的世界。它和虚拟机拥有同样的权限，本地方法可以通过本地方法接口来 访问虚拟机内部的运行时数据区



它甚至可以直接使用本地处理器中的寄存器

直接从本地内存的堆中分配任意数量的内存

7.并不是所有的JVM都支持本地方法。因为Java虚拟机规范并没有明确要求本地方法栈的使用语言、具体实现方式、数据结构等。如果JVM产品不打算支持native方法，也可以无需实现本地方法栈。

* 注意事项

1. 当某个线程调用一个本地方法时，它就进入了一个全新的并且不再受虚拟机限制的世界。它和虚拟机拥有同样的权限。

本地方法可以通过本地方法接口来访问虚拟机内部的运行时数据区

它甚至可以直接使用本地处理器中的寄存器

直接从本地内存的堆中分配任意数量的内存

1. 并不是所有的JVM都支持本地方法。因为Java虚拟机规范并没有明确要求本地方法栈的使用语言、具体实现方式、数据结构等。如果JVM产品不打算支持native方法，也可以无需实现本地方法栈。
2. 在Hotspot JVM中，直接将本地方法栈和虚拟机栈合二为一。