 阻碍Java获得广泛应用的一个主要因素是Java程序的运行效率。Java是介于解释型和编译型之间的一种语言，同样的程序，如果用编译型语言C来实现，其运行速度一般要比Java快一倍以上。Java具有平台无关性，这使人们在开发企业级应用的时候总是把它作为主要候选方案之一，但是性能方面的因素又大大削弱了它的竞争力。为此，提高Java的性能就显得十分重要。 问题的提出 Sun公司及Java的支持者们为提高Java的运行速度已经做出了许多努力，其中大多数集中在程序设计的方法和模式选择方面。由于算法和设计模式的优化是通用的，对Java有效的优化算法和设计模式，对其他编译语言也基本同样适用，因此不能从根本上改变Java程序与编译型语言在执行效率方面的差异。 JIT(Just In Time，及时编译)技术是个比较好的思想。它的基本原理是：首先通过Java编译器把Java源代码编译成平台无关的二进制字节码。然后在Java程序真正执行之前，系统通过JIT编译器把Java的字节码编译为本地化机器码。最后，系统执行本地化机器码，节省了对字节码进行解释的时间。这样做的优点是大大提高了Java程序的性能，缩短了加载程序的时间；同时，由于编译的结果并不在程序运行间保存，因此也节约了存储空间。缺点是由于JIT编译器对所有的代码都想优化，因此同样也占用了很多时间。 动态优化技术是提高Java性能的另一个尝试。该技术试图通过把Java源程序直接编译成机器码，以充分利用Java动态编译和静态编译技术来提

高Java的性能。该方法把输入的Java源码或字节码转换为经过高度优化的可执行代码和动态库 (Windows中的. dll文件或Unix中的. so文件)。该技术能大大提高程序的性能，但却破坏了Java的可移植性。 JNI技术 实际上，有一种通常为我们忽视的技术可以在很大程度上解决这个难题，那就是JNI(Java Native Interface, Java本地化方法)。主张采用纯Java的人们通常反对本地化代码的使用，他们认为在Java程序执行的过程中调用C/C 程序会影响程序的可移植性和安全性。还有一些人认为JNI只是对过去混合编程技术的简单扩展，其实际目的是为了充分利用大量原有的C程序库。 其实，我们不必拘泥于严格的平台独立性限制，因为采用JNI技术只是针对一些严重影响Java性能的代码段，该部分可能只占源程序的极少部分，所以几乎可以不考虑该部分代码在主流平台之间移植的工作量。同时，也不必过分担心类型匹配问题，我们完全可以控制代码不出现这种错误。此外，也不必担心安全控制问题，因为Java安全模型已扩展为允许非系统类加载和调用本地方法。根据Java规范，从JDK 1. 2开始，FindClass将设法找到与当前的本地方法关联的类加载器。如果平台相关代码属于一个系统类，则无需涉及任何类加载器. 否则，将调用适当的类加载器来加载和链接已命名的类。换句话说，如果在Java程序中直接调用C/C 语言产生的机器码，该部分代码的安全性就由Java虚拟机控制。 JNI实现步骤 编写JNI代码的大致流程如下图所示：JNI实现流程图 1. 首先编写需要JNI功能的Java类源文件。其中，需要JNI实现的方法应当用native关键字声明。在该类中，用System. loadLibrary()方法加载需要的动态链接库。关键代码如下： //Compute.java …… public class Compute { public

native double comp (double [] params). …… static { // 调用动态链接库 System. loadLibrary(“ mathlib ”). } …… } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com