提升JAVA程序的性能 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/252/2021_2022__E6_8F_90_E5_8D_87JAVA_c104_252379.htm 随着时间的推移，Java虚拟机变得越来越好，但是通过一些简单的技巧，你仍然可以明显地改进程序的性能。 简介 Java的诸多优点已经广为称道。特别是“ 一次编程，到处运行 ”的承诺使开发人员可以自由地进行跨平台应用程序的开发而不存在预处理器指令的开销。通常认为Java的弱点在于其性能方面。 在当前这种认识并不是完全正确的，有很多产品可以提高Java程序的性能并能够使其在很多应用程序中不再成为一个问题。例如，TowerJ是一种将Java字节代码转换成高度优化的本地可执行程序的后期编译器，Jrockit是一种具有自适应优化能力的服务器端的Java虚拟机。尽管如此，运用一些简单的技巧可以使你不必购买上述的这些工具也能够改进Java代码的性能。在文本中我将说明其中的一些。 本文的讨论主要基于那些高吞吐量的代码(服务器端)。鉴于主要的开销是由那些涉及到对象创建和再创建的GUI代码所引起的，对服务器端代码进行有效的性能估算指针是方法的执行时间。因此，对于所涉及到的示例代码，我记录了执行方法所需的平均时间。记录一个方法的精确执行时间并不是切实可行的，因此我对一系列方法进行计时并计算其平均值。这样做有效地模拟了那些以性能为关键的代码的执行。 每个示例都带有对字节代码操作进行解释的伪代码。所产生的实际的字节代码可以从CUJ的Web站点（www.cuj.com/code）获取。对所有字节代码的解释可以从Javasoft的站点获得。 改善字符串处理的性能 同C 一样

，Java库中定义了自己的String类型。在其外表之下，这种类型是由一个char型数组所实现的，然而使用字符串并不需要理解这一点。NULL符（’\0’）是导致很多学生在学习和使用C 的过程中受挫的祸根；使用Java则不必为此分心，程序员可以专注于应用程序本身以及创建应用程序所用的工具。但是存在着与这种省心的字符串处理方式相关的不利方面，那就是字符串的连接操作符‘ ’。 这个操作符看起来十分有用。多数需要向流写入数据的应用程序都使用‘ ’。例如：String name = new String("Joe"). System.out.println(name " is my name."). 在上面的代码段中，看起来似乎在println语句中无法作出什么修改以改善其执行的速度。然而，这个语句所产生的字节代码（在此用伪代码表示）却揭示了事实，见程序清单1。 清单1：描述由字符串连接符‘ ’所产生的字节代码操作的伪代码 create new String (STR_1) duplicate the String load the constant String "Joe" (STR_2) call String constructor store this String in the local variable array position 0 get the static out field from the Java.io.PrintStream class (OUT) create a new StringBuffer (STR_BUF_1) duplicate the StringBuffer call StringBuffer constructor store this StringBuffer in the local variable array position 1 invoke the append method on STR_BUF_1 with STR_1 as the argument load the constant String " is my name." (STR_3) invoke the append method on STR_BUF_1 with STR_3 as the argument invoke toString method on STR_BUF_1 (STR_4) invoke the println method on OUT 这段简单的代码创建了5个对象[注1]：STR_1, STR_2, STR_3, STR_4, and STR_BUF_1。 需要注意对象创建相对来讲是非常耗费资源的。必须为每个类和类的每一个超类

的所有实例变量分配堆存储空间；所有的实例变量必须被初始化；而且类的构造函数和每个超类的构造函数必须被执行。为了创造高效的代码，只限于在绝对必须的情况下进行对象的创建是十分必要的。 那么，刚才的代码是否可以用一种更高效的方法重写呢？请考虑下面的程序段： StringBuffer name = new StringBuffer("Joe"). System.out.println(name.append(" is my name.").toString()). 相应的字节代码/伪代码请参见程序清单2。 清单2：使用StringBuffer的append操作符的字节代码/伪代码 create new StringBuffer (STR_BUF_1) duplicate the StringBuffer load the constant String "Joe" (STR_1) call StringBuffer constructor store this StringBuffer in the local variable array position 1 get the static out field from the Java.io.PrintStream class (OUT) load STR_BUF_1 load the constant String " is my name." (STR_2) invoke the append method on STR_BUF_1 with STR_2 as the argument invoke toString method on STR_BUF_1 (STR_3) invoke the println method on OUT 上面的代码值创建了4个对象：STR_1, STR_2, STR_3, 和STR_BUF_1。你可能会认为减少一个对象的创建并不能起到多大作用。然而，县棉的代码创建了8个对象： String name = new String("Joe"). name =" is my". name =" name.". 而这段代码仅创建了5个： StringBuffer name = new StringBuffer("Joe"). name.append(" is my"). name.append(" name.").toString(). 第二段代码执行的速度比第一段快两倍还多[注2]。 结论：使用StringBuffer来改进字符串处理代码的性能。其目标是使新对象的创建达到最少，这可以通过使用在StringBuffer之上使用append方法来代替在String上使用连接操作符来实现。 更快的日志记录（faster logging） 在我参与开

发的每个软件项目中，都要求有一种适当的日志记录机制。在应用程序中包括日志记录功能的原因有很多。主要的原因是为了使维护更加容易。为了在发行的应用程序中实现错误报告，有必要设置开始点。在很多情况下，用户提交的报告含义不清，其描述的问题可能是由多方面因素造成的。如果有一种适当的机制能够使用户收集关于此问题的额外信息，那么解决问题的周期会大大地缩减。 并没有标准的方法来产生这种信息，通常这有赖于开发人员如何适当地建立这种机制。然而，日志记录机制的实现对应用程序的性能会造成较大的影响。我们的目标是建立一种能够输出有价值的运行时信息但同时使其对运行时性能的影响达到最小的机制。 避免运行时开销的最显而易见的办法是不将日志记录包括在发行的应用程序中；换句话说，如果执行日志记录的实际代码没有编译到应用程序中的话，那么也就不会对性能造成影响。程序清单3展示了一个定义这样一种记录机制的类。可以对其进行设置使日志记录代码从所产生的字节代码中忽略。这个类将是一个单元素（Singleton）以避免创建不必要的Logger类的实例。 