JAVA循环谜题27:变幻莫测的i值 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_JAVA_E5_BE_AA_E7_8E_AF_c97_138539.htm 与谜题26中的程序一样，下面的程序也包含了一个记录在终止前有多少次迭代的循环。与那个程序不同的是，这个程序使用的是左移操作符（public class Shifty { public static void main(String[] args) { int i = 0. while (-1 i . System.out.println(i). } } 常量-1是所有32位都被置位的int数值（0xffffffff）。左移操作符将0移入到由移位所空出的右边的最低位，因此表达式（-1 问题在于（-1 这条规则作用于全部的三个移位操作符：>和>>>。移位长度总是介于0到31之间，如果左操作数是long类型的，则介于0到63之间。这个长度是对32取余的，如果左操作数是long类型的，则对64取余。如果试图对一个int数值移位32位，或者是对一个long数值移位64位，都只能返回这个数值自身的值。没有任何移位长度可以让一个int数值丢弃其所有的32位，或者是让一个long数值丢弃其所有的64位。 幸运的是，有一个非常容易的方式能够订正该问题。我们不是让-1重复地移位不同的移位长度，而是将前一次移位操作的结果保存起来，并且让它在每一次迭代时都向左再移1位。下面这个版本的程序就可以打印出我们所期望的32： public class Shifty { public static void main(String[] args) { int distance = 0. for (int val = -1. val != 0. val distance . System.out.println(distance). } } 这个订正过的程序说明了一条普遍的原则：如果可能的话，移位长度应该是常量。如果移位长度紧盯着你不放，那么你让其值超过31，或者如果左操作数是long类型的，让其值超过63的可能性就会大大

降低。当然，你并不可能总是可以使用常量的移位长度。当你必须使用一个非常量的移位长度时，请确保你的程序可以应付这种容易产生问题的情况，或者压根就不会碰到这种情况。 前面提到的移位操作符的行为还有另外一个令人震惊的结果。很多程序员都希望具有负的移位长度的右移操作符可以起到左移操作符的作用，反之亦然。但是情况并非如此。右移操作符总是起到右移的作用，而左移操作符也总是起到左移的作用。负的移位长度通过只保留低5位而剔除其他位的方式被转换成了正的移位长度如果左操作数是long类型的，则保留低6位。因此，如果要将一个int数值左移，其移位长度为-1，那么移位的效果是它被左移了31位。 总之，移位长度是对32取余的，或者如果左操作数是long类型的，则对64取余。因此，使用任何移位操作符和移位长度，都不可能将一个数值的所有位全部移走。同时，我们也不可能用右移操作符来执行左移操作，反之亦然。如果可能的话，请使用常量的移位长度，如果移位长度不能设为常量，那么就要千万当心。 语言设计者可能应该考虑将移位长度限制在从0到以位为单位的类型尺寸的范围内，并且修改移位长度为类型尺寸时的语义，让其返回0。尽管这可以避免在本谜题中所展示的混乱情况，但是它可能会带来负面的执行结果，因为Java的移位操作符的语义正是许多处理器上的移位指令的语义。 