JAVA循环谜题33:循环者遇到了狼人 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_138525.htm 请提供一个对i的声明，将下面的循环转变为一个无限循环。这个循环不需要使用任何5.0版的特性： while (i != 0 amp. i == -i) { } 这仍然是一个循环。在布尔表达式(i != 0 amp. i == -i)中，一元减号操作符作用于i，这意味着它的类型必须是数字型的：一元减号操作符作用于一个非数字型操作数是非法的。因此，我们要寻找一个非0的数字型数值，它等于它自己的负值。NaN不能满足这个属性，因为它不等于任何数值，因此，i必须表示一个实际的数字。肯定没有任何数字满足这样的属性吗？ 嗯，没有任何实数具有这种属性，但是没有任何一种Java数值类型能够对实数进行完美建模。浮点数值是用一个符号位、一个被通俗地称为尾数（mantissa）的有效数字以及一个指数来表示的。除了0之外，没有任何浮点数等于其符号位反转之后的值，因此i的类型必然是整数型的。 有符号的整数类型使用的是2的补码算术运算：为了对一个数值取其负值，你要反转其每一位，然后加1，从而得到结果[JLS 15.15.4]。2的补码算术运算的一个很大的优势是，0具有唯一的表示形式。如果你要对int数值0取负值，你将得到0xffffffff 1，它仍然是0。 但是，这也有一个相应的不利之处，总共存在偶数个int数值准确地说有232个其中一个用来表示0，这样就剩些奇数个int数值来表示正整数和负整数，这意味着正的和负的int数值的数量必然不相等。这暗示着至少有一个int数值，其负值不能正确地表示成为一个int数值。 事实上，恰恰就有一个这样的int数值，它就

是Integer.MIN_VALUE，即-231。他的十六进制表示是0x80000000。其符号位为1，其余所有的位都是0。如果我们对这个值取负值，那么我们将得到0x7fffffff 1，也就是0x80000000，即Integer.MIN_VALUE！换句话说，Integer.MIN_VALUE是它自己的负值，Long.MIN_VALUE也是一样。对这两个值取负值将会产生溢出，但是Java在整数计算中忽略了溢出。其结果已经阐述清楚了，即使它们并不总是你所期望的。 下面的声明将使得布尔表达式(i != 0 amp. i == -i)的计算结果为true，从而使循环无限环绕下去： int i = Integer.MIN_VALUE. 下面这个也可以： long i = Long.MIN_VALUE. 如果你对取模运算很熟悉，那么很有必要指出，这个谜题也可以用代数方法解决。Java的int算术运算是实际的算术运算对232取模的运算，因此本谜题需要一个对这种线性全等的非0解决方案： i ≡ -i(mod 232) 将i加到恒等式的两边，我们可以得到： 2i ≡ 0(mod 32) 对这种全等的非0解决方案就是 i = 231。尽管这个值不能表示成为一个int，但是它是和-231全等的，即与Integer.MIN_VALUE全等。 总之，Java使用2的补码的算术运算，它是非对称的。对于每一种有符号的整数类型（int、long、byte和short），负的数值总是比正的数值多一个，这个多出来的值总是这种类型所能表示的最小数值。对Integer.MIN_VALUE取负值得到的还是它没有改变过的值，Long.MIN_VALUE也是如此。对Short.MIN_VALUE取负值并将所产生的int数值转型回short，返回的同样是最初的值（Short.MIN_VALUE）。对Byte.MIN_VALUE来说，也会产生相似的结果。更一般地讲，千万要当心溢出：就像狼人一样，它是个杀手。 对语言设计者的教训与谜题26中的教

训一样。应该对某种溢出不会悄悄发生的整数算术运算形式提供语言级的支持。