JAVA循环谜题31:循环者的鬼魂 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_138533.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： while (i != 0) { i >>>= 1. } 回想一下，>>>=是对应于无符号右移操作符的赋值操作符。0被从左移入到由移位操作而空出来的位上，即使被移位的负数也是如此。 这个循环比前面三个循环要稍微复杂一点，因为其循环体非空。在其循环题中，i的值由它右移一位之后的值所替代。为了使移位合法，i必须是一个整数类型（byte、char、short、int或long）。无符号右移操作符把0从左边移入，因此看起来这个循环执行迭代的次数与最大的整数类型所占据的位数相同，即64次。如果你在循环的前面放置如下的声明，那么这确实就是将要发生的事情： long i = -1. // -1L has all 64 bits set 你怎样才能将它转变为一个无限循环呢？解决本谜题的关键在于>>>=是一个复合赋值操作符。（复合赋值操作符包括*=、/=、%=、 =、-=、>=、>>>=、&amp.=、^=和|=。）有关混合操作符的一个不幸的事实是，它们可能会自动地执行窄化原始类型转换[JLS 15.26.2]，这种转换把一种数字类型转换成了另一种更缺乏表示能力的类型。窄化原始类型转换可能会丢失级数的信息，或者是数值的精度[JLS 5.1.3]。 让我们更具体一些，假设你在循环的前面放置了下面的声明： short i = -1. 因为i的初始值（(short)0xffff）是非0的，所以循环体会被执行。在执行移位操作时，第一步是将i提升为int类型。所有算数操作都会对short、byte和char类型的操作数执行这样的提升。这种提升是一个拓宽原始类型转换，因此没有任

何信息会丢失。这种提升执行的是符号扩展，因此所产生的int数值是0xffffffff。然后，这个数值右移1位，但不使用符号扩展，因此产生了int数值0x7fffffff。最后，这个数值被存回到i中。为了将int数值存入short变量，Java执行的是可怕的窄化原始类型转换，它直接将高16位截掉。这样就只剩下(short)oxffff了，我们又回到了开始处。循环的第二次以及后续的迭代行为都是一样的，因此循环将永远不会终止。 如果你将i声明为一个short或byte变量，并且初始化为任何负数，那么这种行为也会发生。如果你声明i为一个char，那么你将无法得到无限循环，因为char是无符号的，所以发生在移位之前的拓宽原始类型转换不会执行符号扩展。 总之，不要在short、byte或char类型的变量之上使用复合赋值操作符。因为这样的表达式执行的是混合类型算术运算，它容易造成混乱。更糟的是，它们执行将隐式地执行会丢失信息的窄化转型，其结果是灾难性的。 对语言设计者的教训是语言不应该自动地执行窄化转换。还有一点值得好好争论的是，Java是否应该禁止在short、byte和char变量上使用复合赋值操作符。