Java高级谜题87：紧张的关系 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_E9_AB_98_E7_BA_A7_c97_138699.htm 在数学中，等号（＝）定义了一种真实的数之间的等价关系（equivalence relation）。这种等价关系将一个集合分成许多等价类（equivalence class），每个等价类由所有相互相等的值组成。其他的等价关系包括有所有三角形集合上的“ 全等 ”关系和所有书的集合上的“ 有相同页数 ”的关系等。事实上，关系 ～ 是一种等价关系，当且仅当它是自反的、传递的和对称的。这些性质定义如下：自反性：对于所有x，x ～ x。也就是说，每个值与其自身存在关系 ～ 。 传递性：如果x ～ y 并且y ～ z，那么x ～ z。也就是说，如果第一个值与第二个值存在关系 ～，并且第二个值与第三个值存在关系 ～ ，那么第一个值与第三个值也存在关系 ～ 。 对称性：如果x ～ y，那么y ～ x。也就是说，如果第一个值和第二个值存在关系 ～ ，那么第二个值与第一个值也存在关系 ～ 。 如果你看了谜题29，便可以知道操作符 == 不是自反的，因为表达式(Double.NaN == Double.NaN ）值为false，表达式（ Float.NaN == Float.NaN ）也是如此。但是操作符 == 是否还违反了对称性和传递性呢？事实上它并不违反对称性：对于所有x和y的值，（ x == y ）意味着（ y == x ）。传递性则完全是另一回事。 谜题35为操作符 == 作用于原始类型的数值时不符合传递性的原因提供了线索。当比较两个原始类型数值时，操作符 == 首先进行二进制数据类型提升（binary numeric promotion）[JLS 5.6.2]。这会导致这两个数值中有一个会进行拓宽原始类型转换（widening primitive

conversion）。大部分拓宽原始类型转换是不会有问题的，但有三个值得注意的异常情况：将int或long值转换成float值，或long值转换成double值时，均会导致精度丢失。这种精度丢失可以证明 == 操作符的不可传递性。 实现这种不可传递性的窍门就是利用上述三种数值比较中的两种去丢失精度，然后就可以得到与事实相反的结果。可以这样构造例子：选择两个较大的但不相同的 long型数值赋给x和z，将一个与前面两个long型数值相近的double型数值赋给y。下面的程序就是其代码，它打印的结果是true true false，这显然证明了操作符 == 作用于原始类型时具有不可传递性。 public class Transitive { public static void main(String[] args) throws Exception { long x = Long.MAX_VALUE. double y = (double) Long.MAX_VALUE. long z = Long.MAX_VALUE - 1. System.out.print((x == y)“ “). // Imprecise! System.out.print((y == z)“ “). // Imprecise! System.out.println(x == z). // Precise! } } 本谜题的教训是：要警惕到float和double类型的拓宽原始类型转换所造成的损失。它们是悄无声息的，但却是致命的。它们会违反你的直觉，并且可以造成非常微妙的错误（见谜题34）。更一般地说，要警惕那些混合类型的运算（谜题5、8、24和31）。本谜题给语言设计者的教训和谜题34一样：悄无声息的精度损失把程序员们搞糊涂了。 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