Java中的浮点数分析 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_E4_B8_AD_E7_9A_84_c97_138652.htm 浮点数分为单精度和双精度,Java中的单精度和双精度分别为float和double.你们知道float和double是怎么存储的吗? float占4个字节,double占8个字节,为了方便起见,这里就只讨论float类型. float其实和一个int型的大小是一样的,一共32位,第一位表示符号,2-9表示指数,后面23位表示小数部分.这里不多说,请参考:《Java中float的取值范围》。 这里只举一个例子,希望能抛砖引玉,就是研究一下浮点数0.1的存储形式,先运行这个程序. public class Test{ public static void main(String[] args) { int x = 0x3d800000. int i = 1 int j = 1 float f = 0.1f. int y = Float.floatToIntBits(f). float rest = f - ((float) 1) / j. while (i > 0) { j float deta = ((float) 1) / j. if (rest >= deta) { rest -= deta. x |= i. } i >>= 1. } pr(x). pr(y). } static void pr(int i) { System.out.println(Integer.toBinaryString(i)). } } 结果: 11110111001100110011001101101 11110111001100110011001101101 程序说明: int x=0x3d80000. 因为浮点表示形式为1.f*2n-127我们要表示0.1,可以知道n-127=-4,到n=123 符号为正,可知前9是 001111011,暂时不考虑后面的23位小数,所以我们先假设x=0x3d800000. int i = 1 i初始为第右起第23位为1,就是x的第10位 int j = 1 i初始为4,因为n-127为-4,这里是为了求它的倒数. float f = 0.1f. int y = Float.floatToIntBits(f). y就是它的32位表示 float rest = f - ((float) 1) / j. 这个rest表示除了1.f中的1剩下的,也就是0.f while (i > 0) { j float deta = ((float) 1) / j. if (rest >= deta) { rest -= deta. x |= i. } i >>= 1. } 这个循环来计

算23位小数部分,如果rest不小于deta,表示这个位可以置为1. 其他的不多说了,输入结果是一样的,可以说0.1这个浮点数肯定是不精确的,但是0.5可以精确的表示,想想为什么吧.