浮点数

一、概述

Java 虚拟机的浮点数支持符合 IEEE0-754 1985 浮点标准，该标准定义了 32 位和 64 位浮点数的格式及这些浮点数的运算. 在 java 虚拟机中，浮点运算基于 32 位 float 类型和 64 位 double 类型进行. 每一个执行 float 类型运算的操作码,都会有一个与之对应的操作码，该操作码在 double 类型上实现同样的功能.

浮点数由符号,尾数,基数和指数四部分组成. 符号位的值要么是 1,要么是-1,尾数永远是一个正数，它确定浮点数的有效位数. 指数指与尾数,符号相乘的基数的幂的值，幂值可以为正,也可以为负. 可以用以下公式计算: 符号位与尾数相乘，然后再乘以基数的指数次幂,即得到所指的浮点数.

符号 * 尾数 * 基数的指数次幂

二、规范化指数

因为同一个浮点数可以表示为多个不同的尾数,基数和指数的组合,所以浮点数可以有多种表示形式. 例如，数字-5 可以被表示为以下所列的几种形式，这几种形式都是以 10 为基数.

符号	尾数	基数^指数
-1	50	10^-1
-1	5	10^0
-1	0.5	10^1
-1	0.05	10^2

对于每一个浮点数来说，都会有一种被称为"规范化"的表示形式. 如果一个浮点数的尾数满足下面所列的关系式,则称这个浮点数为规范化的浮点数.

1/基数 <= 尾数 < 1

在以 10 为基数的浮点数中，尾数的小数点位置在第一个不为 0 的数字的左边. 因此, -5 的规范化浮点数表示为: -1 * 0.5 * 10^1. 换句话说，一个规范化浮点数的尾数，它的小数点左边的数字一定为 0，紧接小数点右边的数字一定不为 0. 不符合这条规则的浮点数被称为非规范化的浮点数. 需要注意的是，因为 0 在小数点右边没有不为 0 的数字，因此数字 0 没有规范化的表示. 在处理数字 0 的时候经常会发出这样的感叹" 为什么要规范化??"

Java 虚拟机中浮点数使用 2 为基数，因此，它们可以表示为如下形式:

符号 * 尾数 * 2 的指数次幂

Java 虚拟机中浮点数的尾数使用二进制来表示. 规范化尾数的二进制小数点(基于二进制与十进制中作用相同)在最高有效非 0 数字的左边. 因为二进制系统只有两个数字 -- 0 和 1，所以，对于规范化尾数来说，最高有效的数字就是 1.

对于 float 类型或者 double 类型来说，符号位是最高有效位. 尾数在 float 类型中占最低有效位的 23 位，在 double 类型中占最低有效位的 52 位. 指数位于符

号和尾数之间，它在 float 类型中占 8 位，在 double 类型中占 11 位. float 类型的格式如下所示，符号位表示为 s，指数位表示为 e，尾数位表示为 m

31 30~23 22~0

s eeeeeeee mmmmmmmmmmmmmmmmmmmmmmm

符号位为 0，表示正数；符号位为 1，表示负数. 尾数通常表示为二进制正数. 它并不是二进制补码数. 如果符号位为 1，浮点数的值为负，但是尾数仍然表示为正数，但是如果要表示浮点数的值,就需要乘上-1.

指数位的解释有三种方式. 指数位全为 1，表示该数为乘法或者减法所产生的特殊值之一 -- 无穷大或者非数字(NaN Not a Number)，NaN 是某种特殊操作，诸如 0 除以 0 的结果，指数位均为 0，表示该数是一个非规范化浮点数. 其他类型的指数位均表示该数为一个规范化的浮点数.

尾数区域包含一位附加精度位，它不同于尾数中的其他位的作用. float 类型的尾数只占据 23 位，但它有 24 位精度.double 类型的尾数占据 52 位,但它有 53 位精度. 由于 Java 虚拟机的浮点数的指数指明该数是否是规范化的，尾数中的最高有效位是可以预知的,因此没有被划入位数的范围. 如果指数位全为 0，那么该数为非规范化的浮点数，可知尾数的最高有效位为 0. 否则，浮点数是规范化的，那么尾数的最高有效位为 1.

Java 虚拟机在任何浮点数操作中均不抛出异常. 特殊值(例如正无穷大,负无穷大或者 NaN)作为可疑操作(比如除 0)的结果返回. 指数位全为 1，表示特殊的浮点值. 指数位全为 1 而且尾数全为 0,表示无穷大. 无穷大的符号由符号位表示出来.指数位全为 1,尾数位不全为 0，表示该数为 NaN. Java 虚拟机总是为 NaN 产生同样的尾数:除了尾数中最高有效位为 1 外,其余各位均为 0. float 类型的这些特殊值如下所示:

值	浮点位(符号 指数 尾数)
+无穷	0 11111111 00000000000000000000000
-无穷	1 11111111 00000000000000000000000
NaN	1 11111111 10000000000000000000000

如果指数位既不全为 0,又不全为 1，该数即为规范化的浮点数. 可以通过把指数位看作是一个正数，然后从这个正数中减去一个偏移量的方法来确定 2 的幂指数. 对于 float 类型，偏移量为 127. 对于 double 类型，偏移量为 1023. 例如，一个浮点数的指数区域为 00000001，那么它的幂指数可以通过如下步骤得到：把指数区域看作是一个正整数(1)，然后减去偏移量(127)，然后就可以得到 2 的幂指数为 1-127=-126. 这个值是 float 类型的的 2 的幂指数的最小值. 另外,指数区域为 11111110 的 2 的幂指数为(254-127),即为 127. 127 是 float 类型的最大的 2 的幂指数.

指数位全为 0 表示尾数非规范化，这意味着未指明的最重要位的值为 0,而不是 1. 此时 2 的幂指数与规范化尾数的 2 的幂指数最小值相等. 对于 float 类型，该值为-126. 与 2 的-126 次幂相乘的规范化尾数，其指数区域为 00000001;而与 2 的-126 次幂相乘的非规范化尾数,其指数区域为 00000000.

指数范围底端的非规范化浮点数允许渐进的下溢. 如果这个最小的指数用来描述规范化数,那么一些较大的数将会下溢至 0. 换句话说，指定了非规范化数最

小指数使得描述一些更小的数成为可能. 这些更小的数与那些规范化数相比较, 它们的精度较低, 但是这样可以有效地解决指数一旦达到规范化最小值,就会下溢至 0 的问题.

三、其他：

所有浮点数取余操作都不会导致异常抛出. 任何值除以 0 的取余操作都会得出 NaN 的结果. 关于无穷大, 0, NaN 和有限值之间的多种组合的取余结果如下所示:

a	b	a/b	a%b
有限值	+-0.0	+-无限值	NaN
有限值	+=无限值	+-0	a
+-0.0	+-0.0	NaN	NaN
+-无限值	有限值	+-无限值	NaN
+-无限值	+无限值	NaN	NaN

四、参考(Float 中的关于浮点数的处理)：

演示了从 4 个字节的数据中还原一个 float

```
float bytesToFloat(byte[] bytes) {
    // 用一个联合,实现浮点数 float 和 int 的共用体
    union {
        int i;
        float f;
    } u;

    /* do conversion */

    int ival =      ((bytes[srcpos + 0] & 0xFF) << 24) +
                    ((bytes[srcpos + 1] & 0xFF) << 16) +
                    ((bytes[srcpos + 2] & 0xFF) << 8) +
                    ((bytes[srcpos + 3] & 0xFF) << 0);
    u.i = ival;
    return u.f;
}
```