**float类型在内存中的表示**

http://blog.csdn.net/nethibernate/article/details/6120382

先说一下计算机中二进制的算法：

* 整数   
  整数的二进制算法大家应该很熟悉，就是不断的除以2取余数，然后将余数倒序排列。比如求9的二进制：   
  9/2=4 余 1   
  4/2=2 余 0   
  2/2=1 余 0   
  1/2=0 余 1   
  一直计算到商为0为止，然后将得到的余数由下到上排列，就得到了9的二进制：1001。   
  从上面的算法我们可以看到，用整数除以2，最终都能够到0。因此，整数是可以用二进制来精确表示的。

**[除以2反序取余]**

* 小数   
  小数的二进制算法和整数的大致相反，就是不断的拿小数部分乘以2取积的整数部分，然后正序排列。比如求0.9的二进制：   
  0.9\*2=1.8 取 1   
  0.8\*2=1.6 取 1   
  0.6\*2=1.2 取 1   
  0.2\*2=0.4 取 0   
  0.4\*2=0.8 取 0   
  0.8\*2=1.6 取 1   
  … …   
  如此循环下去。因此我么得到的二进制小数也是无限循环的：0.11100110011...   
  从小数的二进制算法中我们可以知道，如果想让这种算法停止，只有在小数部分是0.5的时候才可以，但是很不幸，这类的小数很少。所以大部分小数是很难用二进制来精确表示的。

**[乘以2正序取整]**

------------------------我是分割线------------------------------   
  
OK，有了上面的知识，我们进入正题：看看float类型在内存中是如何表示的。    
float类型又称为单精度浮点类型，在 [IEEE 754-2008](http://en.wikipedia.org/wiki/IEEE_754-2008)中是这样定义它的结构的：

|  |
| --- |
| S     EEEEEEEE      FFFFFFFFFFFFFFFFFFFFFFF 31   30        23    22                               0 |

float类型总共4个字节——32位：

1. 符号位  
   其中最左边的为符号位，0为正，1为负。
2. 指数  
   接下来的E是指数，一共8位，也用二进制来表示。

**[E全0 尾数附加位为0 否则尾数附加位为1]**

1. 尾数  
   最后的F是小数部分，尾数正是由这23位的小数部分+1位组成的。（这个稍后解释）。

这里我们需要多说一下指数。虽然指数也是用8位二进制来表示的，但是IEEE在定义它的时候做了些手脚，使用了偏移来计算指数。

IEEE规定，在float类型中，用来计算指数的偏移量为127。也就是说，如果你的指数实际是0，那么在内存中存的就是0+127=127的二进制。稍后我们来看这个到底如何使用。

好了，看了这么多，我们该演示一下计算机如何将一个十进制的实数转换为二进制的。就拿6.9这个数字来举例吧。-\_-||!

首先，我们按照上面说的方法，分别将整数和小数转换成对应的二进制。这样6.9的二进制表示就是110.1110011001100...。这里就看出来 了，6.9转换成二进制，小数部分是无限循环的，这在现在的计算机系统上是无法精确表示的。这是计算机在计算浮点数的时候常常不精确的原因之一。

其次，将小数点左移（或右移）到第一个有效数字之后。说的通俗些，就是把小数点移到第一个1之后。这样的话，对于上面的110.1110011001100...我们就需要把小数点左移2位，得到1.101110011001100...。

接下来的事情就有意思了。首先我们把得到的1.101110011001100..这个数，从小数点后第一位开始，数出23个来，填充到上面float内存 结构的尾数部分（就是那一堆F的地方），我们这里数出来的就是10111001100110011001100。这里又要发生一次不精确了，小数点后超出 23位的部分都将被舍弃，太惨了。

不过，这里有一个可能让大家觉得特别坑爹的事情，就是小数点前面的1也不要了。仔细看看上面的内存结构，确实没有地方存放这个1。原因是这样的：IEEE觉 得，既然我们大家都约定把小数点移动到第一个有效数字之后，那也就默认小数点前面一定有且只有一个1，所以把这个1存起来也浪费，干脆就不要了，以后大家 都这么默契的来就好。这也是为什么我上面说尾数是23位+1位的原因。

填充完尾数，该填充指数了。这个指数就是刚才我们把小数点移动的位数，左移为正，右移为负，再按照上面所说的偏移量算法，我们填充的指数应该是2+127=129。转换成8位二进制就是10000001。

最后，根据这个数的正负来填充符号位。我们这里是正数，所以填0。这样6.9的在内存中的存储结果就出来了：

|  |
| --- |
| 0  10000001  10111001100110011001100 |

总结一下，实数转二进制float类型的方法：

A. 分别将实数的整数和小数转换为二进制  
B. 左移或者右移小数点到第一个有效数字之后  
C. 从小数点后第一位开始数出23位填充到尾数部分   
D. 把小数点移动的位数，左移为正，右移为负，加上偏移量127，将所得的和转换为二进制填充到指数部分  
E. 根据实数的正负来填充符号位，0为正，1为负

如果需要把float的二进制转换回十进制的实数，只要将上面的步骤倒着来一边就行了。

**------------------------我是分割线------------------------------**

**需要注意的东西：**

1. 23位尾数填充的问题  
   虽然在IEEE754标准中我没有找到相应的描述，但是在实际处理的时候，截取23位尾数需要对第24位进行零舍一入的操作，至少在Java虚拟机中是这么做的。有兴趣的可以试试0.7f-0.6f。
2. 运算时向右对阶操作的舍入问题  
   这个也是在实际操作时遇到的问题。到目前为止我还无法确定向右对阶操作是否也进行了零舍一入的操作。有兴趣的可以试试9.6f-6.9f。
3. 指数全零问题  
   全部为零的指数说明当前所表示的是一个特殊的float数字。全零的float类型分为两种情况：    
   * 尾数全零。此时代表当前float数为0。根据符号位，分为+0和-0。这两个在JVM上相等的。   
     这里需要解释一下。因为IEEE的默认1的问题，所以float类型没有办法表示0，因此只能在已有的规定上做一些强制性的规则来表示0，也就有了上面的这个全零的说法。
   * 尾数不全为零。此时说明当前的float数是一个非规格化的数。
4. 指数全一问题
5. 指数全部为一也说明这个float数是一个不寻常的数字。它也分为两种情况：    
   * 尾数全零。此时根据符号位的不同，分为正无穷（+infinity）和负无穷（-infinity）。注意，这两个东西在JVM中是不相等的。
   * 尾数不全为零。此时表示此float数纯粹不是一个数（NaN，Not a Number）。这个NaN也分为QNaN（Quiet NaN）和SNaN（Signalling NaN）。至于这两个NaN有什么区别，下面这段话倒是说明了，但是我没有这方面的知识，所以不敢妄加翻译，只好把原文放在这里：  
     A QNaN is a NaN with the most significant fraction bit set. QNaN's propagate freely through most arithmetic operations. These values pop out of an operation when the result is not mathematically defined.  
     An SNaN is a NaN with the most significant fraction bit clear. It is used to signal an exception when used in operations. SNaN's can be handy to assign to uninitialized variables to trap premature usage.   
     Semantically, QNaN's denote indeterminate operations, while SNaN's denote invalid operations.   
     最后一句话说的明白，QNaN就是一个不确定操作的结果，而SNaN纯粹就是一个非法的操作结果。

------------------------我是分割线-----------------------------

OK，废话了这么多，我觉得对float类型也大致有个了解了。float明白了以后，double类型也就好说了，基本和上面一样，只是指数和尾数的位数不一样而已。

参考：

[IEEE Standard 754 Floating Point Numbers: http://steve.hollasch.net/cgindex/coding/ieeefloat.htm](http://en.wikipedia.org/wiki/IEEE_754-1985)

[IEEE 754-1985: http://en.wikipedia.org/wiki/IEEE\_754-1985](http://en.wikipedia.org/wiki/IEEE_754-2008)

[IEEE 754-2008: http://en.wikipedia.org/wiki/IEEE\_754-2008](http://en.wikipedia.org/wiki/IEEE_754-2008)

Java 理论与实践: 您的小数点到哪里去了？: http://www.ibm.com/developerworks/cn/java/j-jtp0114/

**补充作业**

1. 有关补码，简要阐述补码的好处。并计算给出 -99, -105, 205 整数的补码

2. 有关浮点数，根据IEEE745，计算11000001000100000000000000000000的单精度浮点的值，并给出计算过程。

3.写一个Java程序，将100.2转成IEEE745 二进制表示 ，给出程序和结果。