# float型和double型数据的存储方式

对于浮点类型的数据采用单精度类型（float）和双精度类 型(double)来存储，float数据占用32bit,double数据占用64bit。通常

float可以保证十进制科学计数法小数点后6位有效精度和第7位的部分精度

double可以保证十进制科学计数法小数点后15位有效精度和第16位的部分精度。

因为float和double的精度是由尾数决定的，什么是尾数呢，下面看看浮点型数据在底层是如何存储的。

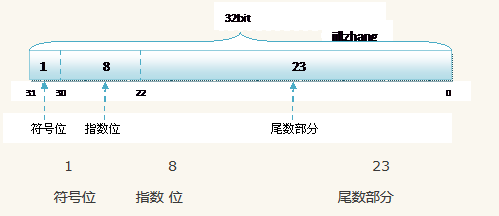
我们在声明一个变量float f= 2.25f的时候，是如何分配内存的呢？如果胡乱分配，那世界岂不是乱套了么?

其实不论是float还是double在存储方式上都是遵从IEEE（**二进制浮点数算术标准**）的规范 的，float遵从的是IEEE R32.24 ,而double 遵从的是R64.53。

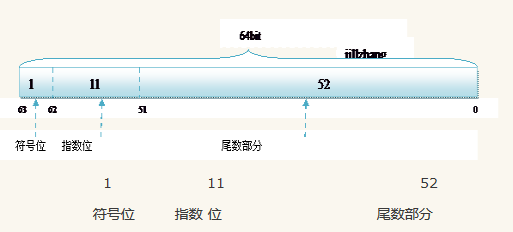
无论是单精度还是双精度在存储中都分为三个部分：

* 符号位(Sign) : 0代表正，1代表为负
* 指数位（Exponent）:用于存储科学计数法中的指数数据，并且要加上偏移量（float偏移127，double偏移量1023）
* 尾数部分（Mantissa）：尾数部分

其单精度float的存储方式如下图所示：

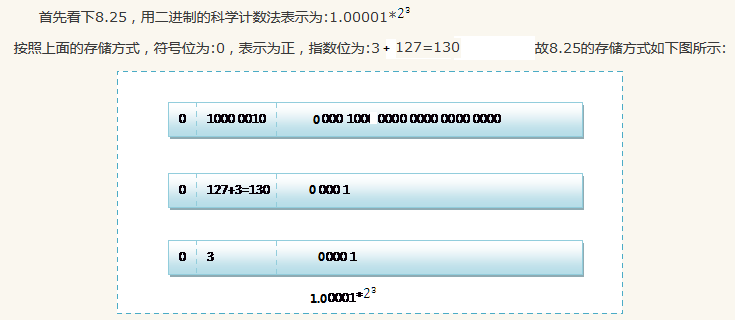


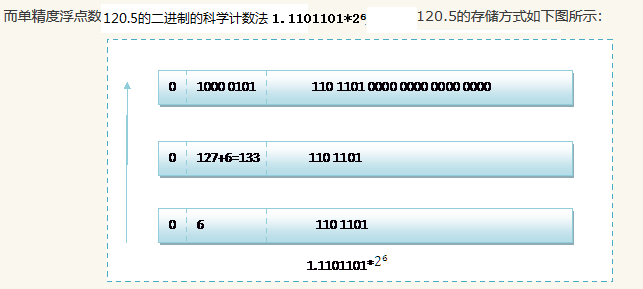
而双精度double的存储方式为:



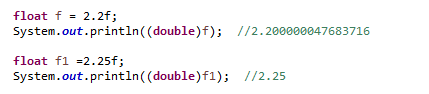
R32.24和R64.53的存储方式都是用科学计数法来存储数据的，比如8.25用十进制的科学计数法表示就为:8.25\*100,而120.5可以表示为:1.205\*102

可是计算机根本不认识十进制的数据，只认识0，1，所以在计算机存储中，首先要将上面的数更改为二进制的科学计数法表示，8.25用二进制表示可表示为1000.01，120.5用二进制表示为：1110110.1。用二进制的科学计数法表示1000.01可以表示为1.00001\*23，1110110.1可以表示为1.1101101\*26。任何一个数都的科学计数法表示都为1.XXX\*2n。尾数部分就可以表示为xxxx,第一位都是1嘛，干嘛还要表示呀？可以将小数点前面的1省略，所以23bit的尾数部分，可以表示的精度却变成了 24bit。





奇怪输出结果：



首先我们看看2.25的单精度存储方式，2.25 --> 10.01 --> 1.001\*21

符号位0，指数部分1+127 --> 10000000

尾数部分：001 0000 0000 0000 0000 0000

很简单 0 1000 0000 001 0000 0000 0000 0000 0000,

而2.25的双精度表示为:0 100 0000 0000 0010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000,这样2.25在进行强制转换的时候，数值是不会变的

而我们再看看2.2呢

发现小数部分的二进制是一个无限循环的排列 00110011001100110011...

对于单精度数据来说，尾数只能表示23bit的精度，所以2.2的float存储为:

2.2 --> 10.0011001100110011001100 -->1.00011001100110011001100\*21

符号位0，指数部分1+127 --> 1000 0000

尾数部分：00011001100110011001100

0 1000 0000 00011001100110011001100

但是这样存储方式，换算成十进制的值，却不会是2.2的

因为十进制在转换为二进制的时候可能会不准确，如2.2，而double类型的数据也存在同样的问题，所以在浮点数表示中会产生些许的误差，在单精度转换为双精度的时候，也会存在误差的问题