# 计算机原码、反码、补码：

【注意：这里说的计算机，并不是指的Java语言】

【参考：https://www.cnblogs.com/zhangziqiu/archive/2011/03/30/ComputerCode.html】

## 机器数和真值：

### 机器数：

一个数在计算机中的二进制表现，叫做这个数的机器数。机器数是带符号的，在计算机中用一个数的最高位存放符号，正数为0，负数为1。

比如：十进制中的+3，如果计算机字长为8位，转换成二进制就是0000-0011；如果是-3，转换成二进制就是1000-0011。

那么，这里的0000-0011和1000-0011就是机器数。

### 2、真值：

因为第一位是符号位，所以机器数的形式值就不等于真正的数值。例如上面的有符号数 1000-0011，其最高位1代表负数，其真正数值是-3，而不是形式值（10000011按照二进制转换是128+2+1=131），所以，为区别起见，将带符号位的机器数对应的真正数值称为计算机的真值。

例：0000-0001的真值=+0000-0001=+1；1000-0001的真值=-0000-0001=-1；

## 原码、反码、补码的概念及计算：

### 原码：

1. 原码就是符号位加上真值的绝对值，即用第一位表示符号，其余位表示真值，比如如果是8位二进制：

|  |
| --- |
| [+1]原 = 0000 0001  [- 1]原 = 1000 0001  第一位是符号位 |

因为第一位是符号位，所以8位二进制的取值范围就是

|  |
| --- |
| [1111 1111 , 0111 1111], 即[-127 , 127] |

原码是人脑最容易理解和计算的表达方式

### 2、反码：

1）反码的表示法是：

【1】正数的反码是其原码本身

【2】负数的反码是在其原码的基础上，符号位不变，其余各个位取反

|  |
| --- |
| [+1] = [00000001]原 = [00000001]反  [- 1] = [10000001]原 = [11111110]反 |

可见如果一个反码表示的是负数，人脑无法直观的看出它的数值，通常需要将其转换成原码再计算。

### 3、补码：

1）补码的表示法是：

【1】正数的补码就是其原码、反码本身

【2】负数的补码是在其原码的基础上，符号位不变，其余各位取反，最后+1（即在反码的基础上+1）

|  |
| --- |
| [+1] = [00000001]原 = [00000001]反 = [00000001]补  [- 1] = [10000001]原 = [11111110]反 = [11111111]补 |

对于负数，补码表示方式也是人脑无法直观看出其数值的，通常也需要转换成原码再计算其数值。

## 为何需要原码、反码和补码：

注：在开始深入学习前, 我的学习建议是先"死记硬背"上面的原码, 反码和补码的表示方式以及计算方法。

现在我们知道了计算机可以有三种编码方式表示一个数. 对于正数因为三种编码方式的结果都相同：

|  |
| --- |
| [+1] = [00000001]原 = [00000001]反 = [00000001]补 |

所以不需要过多解释。但是对于负数：

|  |
| --- |
| [-1] = [10000001]原 = [11111110]反 = [11111111]补 |

可见原码, 反码和补码是完全不同的。既然原码才是被人脑直接识别并用于计算表示方式，为何还会有反码和补码呢?

首先, 因为人脑可以知道第一位是符号位, 在计算的时候我们会根据符号位, 选择对真值区域的加减. (真值的概念在本文最开头)。

但是对于计算机, 加减乘数已经是最基础的运算, 要设计的尽量简单. 计算机辨别"符号位"显然会让计算机的基础电路设计变得十分复杂! 于是人们想出了将符号位也参与运算的方法. 我们知道, 根据运算法则减去一个正数等于加上一个负数, 即: 1-1 = 1 + (-1) = 0 , 所以机器可以只有加法而没有减法, 这样计算机运算的设计就更简单了.

于是人们开始探索 将符号位参与运算, 并且只保留加法的方法. 首先来看原码：

|  |
| --- |
| 计算十进制的表达式：1-1=0 |
| 1 - 1 = 1 + (-1) = [00000001]原 + [10000001]原 = [10000010]原 = -2 |

如果用原码表示, 让符号位也参与计算, 显然对于减法来说, 结果是不正确的。这也就是为何计算机内部不使用原码表示一个数.

为了解决原码做减法的问题, 出现了反码：

|  |
| --- |
| 计算十进制的表达式: 1-1=0 |
| 1 - 1 = 1 + (-1) = [0000 0001]原 + [1000 0001]原= [0000 0001]反 + [1111 1110]反 = [1111 1111]反 = [1000 0000]原 = -0 |

发现用反码计算减法, 结果的真值部分是正确的. 而唯一的问题其实就出现在"0"这个特殊的数值上. 虽然人们理解上+0和-0是一样的, 但是0带符号是没有任何意义的. 而且会有[0000 0000]原和[1000 0000]原两个编码表示0.

于是补码的出现, 解决了0的符号以及两个编码的问题:

|  |
| --- |
| 计算十进制的表达式: 1-1=0 |
| 1-1 = 1 + (-1) = [0000 0001]原 + [1000 0001]原 = [0000 0001]补 + [1111 1111]补 = [0000 0000]补=[0000 0000]原 |

这样0用[0000 0000]表示, 而以前出现问题的-0则不存在了.而且可以用[1000 0000]表示-128：

|  |
| --- |
| (-1) + (-127) = [1000 0001]原 + [1111 1111]原 = [1111 1111]补 + [1000 0001]补 = [1000 0000]补 |

-1-127的结果应该是-128, 在用补码运算的结果中, [1000 0000]补 就是-128. 但是注意因为实际上是使用以前的-0的补码来表示-128, 所以-128并没有原码和反码表示.(对-128的补码表示[1000 0000]补算出来的原码是[0000 0000]原, 这是不正确的)

使用补码, 不仅仅修复了0的符号以及存在两个编码的问题, 而且还能够多表示一个最低数. 这就是为什么8位二进制, 使用原码或反码表示的范围为[-127, +127], 而使用补码表示的范围为[-128, 127].

因为机器使用补码, 所以对于编程中常用到的32位int类型, 可以表示范围是: [-231, 231-1] 因为第一位表示的是符号位.而使用补码表示时又可以多保存一个最小值.

## 误区：

### 1、反码与取反：

反码与取反是不同的：

1. 正数的反码就是本身，而正数的取反是每一位都变成相反的

|  |
| --- |
| 十进制60（32位）的二进制是：  0000-0000 0000-0000 0000-0000 0011-1100 |
| 反码是： 0000-0000 0000-0000 0000-0000 0011-1100 |
| 取反是： 1111-1111 1111-1111 1111-1111 1100-0011  取反结果是：-61 |

1. 负数的反码是最高位（符号位）不变，其余各位取反：

|  |
| --- |
| 十进制-61（32位）的二进制是：  1111-1111 1111-1111 1111-1111 1100-0011 |
| 反码是： 1000-0000 0000-0000 0000-0000 0011-1100 |
| 取反是： 0000-0000 0000-0000 0000-0000 0011-1100  取反结果是：60 |

## 五、计算：

### 1、当二进制的最高位为1 的时候，怎么求出它的十进制呢？

1）步骤：

【1】原二进制数的取反，即：将0变成1，将1变成0

【2】取反后的二进制再加上1【注意是取反，并不是反码】

【3】此时的二进制表示的就是一个正数，换算成十进制，再加上负号就是原二进制数的十进制表现。

2）举例：

|  |
| --- |
| 1111-1111 1111-1111 1111-1111 1001-1100 |

【1】原二进制取反：

|  |
| --- |
| 0000-0000 0000-0000 0000-0000 0110-0011 |

【2】取反后的二进制加1

|  |
| --- |
| 0000-0000 0000-0000 0000-0000 0110-0100 |

【3】计算这个二进制的十进制表示： 64+32+4=100

【4】100前面加一个负号，就是-100，这就是原二进制的十进制表示数了

即： 1111-1111 1111-1111 1111-1111 1001-1100(2) = -100(10)

### 2、int a = -120，怎么求出它的二进制表示？

1）第一种方法：

【1】120的二进制表示：（64+32+16+8）

|  |
| --- |
| 0000-0000 0000-0000 0000-0000 0111-1000 |

【2】120的二进制取反：

|  |
| --- |
| 1111-1111 1111-1111 1111-1111 1000-0111 |

【3】取反后的二进制加1

|  |
| --- |
| 1111-1111 1111-1111 1111-1111 1000-1000 |

所以 -120(10) = 1111-1111 1111-1111 1111-1111 1000-1000(2)