06 原码 反码 补码

- **原码**：十进制的二进制表现形式，最左边是符号位，0为正，1为负
  - 比特：一个0/1为一个bit（比特位）
  - 字节：8个比特为一个字节。字节是计算机中最小的存储单元。
- **反码**：为了解决原码不能计算负数的问题而出现的。
  - 计算规则：
    - 正数的反码不变
    - 负数的反码在原码的基础上，符号位不变。数值取反，0变1，1变0

十进制数字	原码	反码	补码
+0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	0000 0000
-1	1000 0001	1111 1110	1111 1111
-2	1000 0010	1111 1101	1111 1110
-3	1000 0011	1111 1100	1111 1101
-4	1000 0100	1111 1011	1111 1100
-5	1000 0101	1111 1010	1111 1011
-6	1000 0110	1111 1001	1111 1010
-7	1000 0111	1000 1000	1111 1001

- **补码**：为了解决负数计算时跨0的问题而出现。
  - 计算规则
    - 正数的补码不变
    - 负数的补码在反码的基础+1，且补码还能多记录一个特殊值-128，但是-128在一个字节下，没有原码和反码。

十进制数字	原码	反码	补码
+0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	0000 0000
-1	1000 0001	1111 1110	1111 1111
-2	1000 0010	1111 1101	1111 1110
-3	1000 0011	1111 1100	1111 1101
-4	1000 0100	1111 1011	1111 1100
...	...	...	...
-126	1111 1110	1000 0001	1000 0010
-127	1111 1111	1000 0000	1000 0001
-128	无	无	1000 0000

  - **计算机中的存储和计算都是以补码的形式进行的**
  - 所以一个字节为(-128~+127)
- **运算符相关**

运算符	含义	运算规则
&	逻辑与	0为false 1为true
|	逻辑或	0为false 1为true
<<	左移	向左移动，低位补0
>>	右移	向右移动，高位补0或1
>>>	无符号右移	向右移动，高位补0

运算符	含义	运算规则
&	逻辑与	0为false 1为true

```
public class Test {
    public static void main(String[] args) {
        int a = 200;
        int b = 10;
        System.out.println(a & b);
    }
}
```

```
  0000 0000 0000 0000 0000 0000 1100 1000
& 0000 0000 0000 0000 0000 0000 0000 1010
  0000 0000 0000 0000 0000 0000 0000 1000
```

运算符	含义	运算规则
&	逻辑与	0为false 1为true
|	逻辑或	0为false 1为true
<<	左移	向左移动，低位补0

```
public class Test {
    public static void main(String[] args) {
        int a = 200;
        System.out.println(a << 2);
    }
}
```

```
0000 0000 0000 0000 0000 0000 1100 100000
```

800

- 左移一次就是乘2，右移一次就是除2