链滴

详解 & 0xff 的作用

作者：zhaozhizheng

原文链接：https://ld246.com/article/1608639435180

来源网站：链滴

每次遇到与或位移等运算总会懵圈一次，百度明白了之后就忘了，理解不够深刻，不够明确，工中也很少写，所以记下

为什么要加上“& 0xFF”？
拆分理解下
0xFF 是 16 进制的表达方式，F 是 15；十进制为：255，二进制为：1111 1111
&运算符：如果 2 个 bit 都是 1，则得 1，否则得 0
然后开始百度……
最后一路百度到计算机的原理之：原码、补码和反码，先简单讲下这三个词的意思吧！
我们已经知道计算机中，所有数据最终都是使用二进制数表达。
我们也已经学会如何将一个 10 进制数如何转换为二进制数。
不过，我们仍然没有学习一个负数如何用二进制表达。
比如，假设有一 int 类型的数，值为 5，那么，我们知道它在计算机中表示为：
00000000 00000000 00000000 00000101
5 转换成二制是 101，不过 int 类型的数占用 4 字节（32 位），所以前面填了一堆 0。
现在想知道，-5 在计算机中如何表示？
在计算机中，负数以其正值的补码形式表达。
什么叫补码呢？这得从原码，反码说起。
原码：一个整数，按照绝对值大小转换成的二进制数，称为原码。
比如 00000000 00000000 00000000 00000101 是 5 的 原码。
反码：将二进制数按位取反，所得的新二进制数称为原二进制数的反码。
取反操作指：原为 1，得 0；原为 0，得 1。（1 变 0; 0 变 1）
比如：将 00000000 00000000 00000000 00000101 每一位取反，得 11111111 11111111 1111111 11111010。
称：11111111 11111111 11111111 11111010 是 00000000 00000000 00000000 00000101 的反码。
反码是相互的，所以也可称：
11111111 11111111 11111111 11111010 和 00000000 00000000 00000000 00000101 互反码。
补码：反码加 1 称为补码。
也就是说，要得到一个数的补码，先得到反码，然后将反码加上 1，所得数称为补码。
比如：00000000 00000000 00000000 00000101 的反码是：11111111 11111111 11111111 11111010。
那么，补码为：
11111111 11111111 11111111 11111010 + 1 = 11111111 11111111 11111111 1111101
所以，-5 在计算机中表达为：11111111 11111111 11111111 11111011。转换为十六进制：0FFFFFFFB。
再举一例，我们来看整数-1 在计算机中如何表示。
假设这也是一个 int 类型，那么：

先取 1 的原码：00000000 00000000 00000000 00000001
得反码： 11111111 11111111 11111111 11111110
得补码： 11111111 11111111 11111111 11111111

可见， - 1 在计算机里用二进制表达就是全 1。16 进制为：0xFFFFFFFF。

上面这么多蛋疼的操作仅仅是因为：在计算机中，负数以其正值的补码形式表达。
有的人可能会问：那为什么在计算机中，负数以其正值的补码形式表达？
为什么负数以其正值的补码形式表达：说到补码，就不得不引人另一个概念——模数。模数从屋意义上讲是某种计量器的容量。这里我们经常举的一个例子就是钟表，其模数为 12，即每到 12 就重从 0 开始，数学上叫取模或求余(mod)，java、C#和 C++ 里用 % 表示求余操作。例如：
14%12=2
如果此时的正确时间为 6 点，而你的手表指向的是 8 点，如何把表调准呢？有两种方法：一把表逆时拨两个小时；二是把表顺时针拨 10 个小时，即
8-2=6
(8+10)%12=6
也就是说在此模数系统里面有
8-2=8+10
这是因为 2 跟 10 对模数 12 互为补数。因此有一下结论：在模数系统中，A-B 或 A+(-B)等价于 A+[B补]，即
8-2/8+(-2)=8+10
我们把 10 叫做-2 在模 12 下的补码。这样用补码来表示负数就可以将加减法统一成加法来运算，简了运算的复杂程度。
采用补码进行运算有两个好处，一个就是刚才所说的统一加减法；二就是可以让符号位作为数值直接加运算，而最后仍然可以得到正确的结果符号，符号位无需再单独处理。此外，补码与原码相互转换其运算过程是相同的，不需要额外的硬件电路。
到这里估计大家都能大概了解原码、补码和反码了，我们回到一开始的问题。
data[1] = (byte)(deY & 0xFF);
外部传进来一个参数 func，这个参数有可能是负数的，例如传进来一个“-12”，“-12”二进为：
0000 1100 取反： 1111 0011 补码加 1： 1111 0100
byte –> int 就是由 8 位变 32 位 高 24 位全部补 1： 1111 1111 1111 1111 1111 1111 1111 010 ;
0xFF 的二进制表示就是：1111 1111，高 24 位补 0：0000 0000 0000 0000 0000 0000 1111 1111
-12 的补码与 0xFF 进行与（&）操作 最后就是:0000 0000 0000 0000 0000 0000 1111 000
最终保持“-12”取反码，补码加 1 的一致性。
byte 类型的数字要&0xff 再赋值给 int 类型，其本质原因就是想保持二进制补码的一致性
当 byte 要转化为 int 的时候，高的 24 位必然会补 1，这样，其二进制补码其实已经不一致了，amp;0xff 可以将高的 24 位置为 0，低 8 位保持原样。这样做的目的就是为了保证二进制数据的一致。