位运算的运用（Bitwise operations)

本节内容

1. 位运算符
2. 算数移位与逻辑移位
3. 位运算的应用

什么是位运算

程序中的所有数在计算机内存中都是以二进制的形式储存的。位运算说穿了，就是直接对整数在内存中的二进制位进行操作。比如，and运算本来是一个逻辑运算符，但整数与整数之间也可以进行and运算。举个例子，6的二进制是110，11的二进制是1011，那么6 and 11的结果就是2，它是二进制对应位进行逻辑运算的结果（0表示False，1表示True，空位都当0处理）：

```
110 AND 1011 --> 0010(b) --> 2(d)
```

由于位运算直接对内存数据进行操作，不需要转成十进制，因此处理速度非常快。当然有人会说，这个快了有什么用，计算6 and 11没有什么实际意义啊。本文就将告诉你，位运算到底可以干什么，有些什么经典应用，以及如何用位运算优化你的程序。

符号	描述	运算规则
&	与	两个位都为1时，结果才为1
|	或	两个位都为0时，结果才为0
^	异或	两个位相同为0，相异为1
~	取反	0变1，1变0
<<	左移	各二进位全部左移若干位，高位丢弃，低位补0
>>	右移	各二进位全部右移若干位，对无符号数，高位补0，有符号数，各编译器处理方法不一样，有的补符号位（算术右移），有的补0（逻辑右移）

XOR – 异或

> 异或：相同为0，不同为1。也可用「不进位加法」来理解。

异或操作的一些特点：

```
x ^ 0 = x
x ^ 1s = ~x // 1s = ~0
x ^ (~x) = 1s
x ^ x = 0 // interesting and important!
a ^ b = c => a ^ c = b, b ^ c = a // swap
a ^ b ^ c = a ^ (b ^ c) = (a ^ b) ^ c // associative
```

1. 将 `x` 最右边的 `n` 位清零 - `x & (~0 << n)`
2. 获取 `x` 的第 `n` 位值(0或者1) - `(x >> n) & 1`
3. 获取 `x` 的第 `n` 位的幂值 - `x & (1 << (n - 1))`
4. 仅将第 `n` 位置为 `1` - `x | (1 << n)`
5. 仅将第 `n` 位置为 `0` - `x & (~(1 << n))`
6. 将 `x` 最高位至第 `n` 位(含)清零 - `x & ((1 << n) - 1)`
7. 将第 `n` 位至第0位(含)清零 - `x & (~((1 << (n + 1)) - 1))`

编程常用的位运算操作

- X & 1 == 1 OR == 0
- X = X & (X-1) => 清零最低位的1
- X & -X => 得到最低位的1
- X & ~X => 0

实战题目

1. https://leetcode.com/problems/number-of-1-bits/

2. https://leetcode.com/problems/power-of-two/

3. https://leetcode.com/problems/counting-bits/description/

4. https://leetcode.com/problems/n-queens-ii/description/