计算机基础-位运算

每一种语言最终都会通过编译器转换成机器语言来执行，所以直接使用底层的语言就不需要便编译器的转换工作从而得到更高的执行效率，当然可读性可能会降低，这也是为什么汇编在大部分情况下有更快的速度。

计算机计算原理

####加法和乘法

举一个简单的例子来看下CPU是如何进行计算的，比如这行代码

```
int a = 35;
int b = 47;
int c = a + b;
计算两个数的和，因为在计算机中都是以二进制来进行运算，所以上面我们所给的int变量会在机器内部先转换为二进制
35:  0 0 1 0 0 0 1 1
47:  0 0 1 0 1 1 1 1
--------------------
82:  0 1 0 1 0 0 1 0


```

再来看下乘法，执行如下的代码

```

int a = 3;
int b = 2;
int c = a * b;

3:  0 0 0 0 0 0 1 1  *  2
--------------------
6:  0 0 0 0 0 1 1 0

*********************************************

int a = 3;
int b = 4;
int c = a * b;

3:  0 0 0 0 0 0 1 1  *  4
--------------------
12:  0 0 0 0 1 1 0 0

*********************************************

int a = 3;
int b = 8;
int c = a * b;

```

```
3:  0 0 0 0 0 0 1 1  *  8
--------------------
24:  0 0 0 1 1 0 0 0
```

通过以上运算可以看出当用a乘b，且如果b满足2^N的时候 就相当于把a的二进制数据向左移动N位，放到代码中 我们可以这样来写 **a << N**,所以上面**3 * 2**、**3 * 4**、**3 * 8**其实是可以写成**3<<1**、**3<<2**、**3<<3**，运算结果都是一样的。

那假如相乘的两个数都不满足2^N怎么办呢？其实这个时候编译器会将其中一个数拆分成多个满足2^N的数相加的情况，打个比方

```
int a = 15;              int a = 15
int b = 13;      =>      int b = (4 + 8 + 1)
int c = a * b;           int c = a * b
```

最后其实执行相乘运算就会变成这样 **15 * 4 + 15 * 8 + 15 * 1**，按照上文说的移位来转换为位运算就会变成**15 << 2 + 15 << 3 + 15 << 0**

####减法和除法 减法也是与加法同理只不过计算机内减法操作就是加上一个数的负数形式，且在操作系统中都是以补码的形式进行操作(因为正数的源码补码反码都与本身相同)。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. 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.
除法的话其实和乘法原理相同，不过乘法是左移而除法是右移，但是除法的计算量要比乘法大得多，其大部分的消耗都在拆分数值，和处理小数的步骤上，所以如果我们在进行生成变量的时候如果遇到多位的小数我们尽量把他换成string的形式，这也是为什么浮点运算会消耗大量的时钟周期(操作系统中每进行一个移位或者加法运算的过程所消耗的时间就是一个时钟周期，3.0GHz频率的CPU可以在一秒执行运算3.0*1024*1024*1024个时钟周期)

位运算符

含义	运算符	例子
左移	<<	0011 => 0110
右移	>>	0110 => 0011
按位或	|	0011 | 0110 =>0111
按位与	&	0011 &1001 =>0001
按位取反	~	0011 => 1100
按位异或(相同为零不同为一)	^	0011^1001 => 1010

颜色转换

上面说了iOS中经常见到的位运算的地方是在枚举中，那么颜色转换应该是除了枚举之外第二比较常用位运算的场景。打个比方设计师再给我们出设计稿的时候通常会在设计稿上按照16进制的样子给我们标色值。但是iOS中的UIColor并不支持使用十六进制的数据来初始化。所以我们需要将十六进制的色值转换为UIColor。

原理分析

UIColor中通常是用传入RGB的数值来初始化，而且每个颜色的取值范围是十进制下的0~255，而设计同学又给的是十六进制数据，所以在操作系统中需要把这两种进制的数据统一成二进制来进行计算，这就用到了位运算。这里用一个十六进制的色值来举例子比如**0xffa131**我们要转换就要先理解其组成

- 0x或者0X：十六进制的标识符，表示这个后面是个十六进制的数值，对数值本身没有任何意义
- ff 颜色中的R值,转换为二进制为 1111 1111
- a1 颜色中的G值,转换为二进制为 1010 0001
- 31 颜色中的B值,转换为二进制为 0011 0001
- 上述色彩值转换为二进制后为**1111 1111 1010 0001 0011 0001(每一位十六进制的对应4位二进制，如果位数不够记得高位补零)**

通常来讲十六进制的颜色是按照上面的RGB的顺序排列的，但是并不固定，有时候可能会在其中加A(Alpha)值，具体情况按照设计为准，本文以通用情况举例。
综上，我们只需把对应位的值转换为10进制然后/255.0f就可得到RGB色彩值，从而转换为UIColor

题目链接
https://juejin.im/post/6844903645171941384#heading-4