二进制：

笔记：

1. 什么是2进制：逢2进1的计数规则。计算机中的变量/常量都是按照2进制来计算的。
    - 2进制：
        - 规则：逢2进1
        - 数字：0 1
        - 基数：2
        - 权：128 64 32 16 8 4 2 1
    - 正数：如何将2进制转换为10进制：
        - 将一个2进制每个1位置的权相加即可

        > 权：32 16 8 4 2 1
        >
        > 二进制：1 0 1 1 0 1
        >
        > 十进制：32+8+4+1=45

        ```
        /*
         1)编译器在编译时会将10进制编译为2进制，然后按照2进制来计算
           .java(45)--->编译后--->.class(101101)
         2)int类型是32位2进制，显示2进制时自动省略高位0
         3)Integer.toBinaryString()返回底层2进制数
           System.out.println()将2进制转换为10进制输出
         */

        int n = 45; //编译时会将45编译为：101101
        System.out.println(Integer.toBinaryString(n)); //101101---以2进制输出
        System.out.println(n); //45---以10进制输出

        n++; //将101101增1----101110
        System.out.println(Integer.toBinaryString(n)); //101110---以2进制输出
        System.out.println(n); //46---以10进制输出
        ```

2. 什么是16进制：逢16进1的计数规则
    - 16进制：
        - 规则：逢16进1
        - 数字：0 1 2 3 4 5 6 7 8 9 a b c d e f
        - 基数：16
        - 权：4096 256 16 1
    - 用途：因为2进制书写太麻烦，所以常常用16进制来缩写2进制
    - 如何缩写：将2进制从低位开始，每4位2进制缩为1个16进制（每4位1缩）

    > 权：8 4 2 1
    >
    > 2进制：0010 1001 0010 1101 1110 1111 0110 1000
    >
    > 16进制：2 9 2 d e f 6 8---292def68

    > 权：8 4 2 1
    >
    > 2进制：0001 1100 1010 0011 1101

> 16进制：1 c a 3 d---1ca3d

> 权：8 4 2 1
>
> 2进制：0011 1010 1011 1010 1011
>
> 16进制：3 a b a b---3abab

```
int n = 0x1ca3d; //0x为16进制字面量前缀
int m = 0b0001_1100_1010_0011_1101; //0b为2进制字面量前缀
//下划线只是为了让代码更加易读，编译器会删除这些下划线
//计算机内部没有10进制，也没有16进制，只有2进制
System.out.println(Integer.toBinaryString(n)); //以2进制输出
System.out.println(Integer.toBinaryString(m));
System.out.println(n); //以10进制输出
System.out.println(m);

/*
 （8进制平时一般不用）
 8进制：以0开头表示8进制
 1)规则：逢8进1
 2)数字：0 1 2 3 4 5 6 7
 3)基数：8
 4)权：512 64 8 1
 */
int a = 067; //以0开头表示8进制
System.out.println(a); //55(6个8加上7个1---55)

/*
 小面试题：
 int a = 068; //问：这句话正确吗?
 答：编译错误，因为0开头的表示8进制，而8进制最大的数为7
 */
```

3. 补码：
   - 计算机处理有符号数（正负数）的一种编码方式
   - 以4位2进制为例讲解补码的编码规则：
     - 计算时如果超出4位则高位自动溢出舍弃，保持4位不变
     - 将4位2进制数分一半作为负数使用
     - 最高位称为符号位，高位为1是负数，高位为0是正数
   - 规律数：
     - 0111为4位补码的最大值，规律是1个0和3个1，可以推导出：
       - 32位补码的最大值：1个0和31个1（011111111111.....）
     - 1000为4位补码的最小值，规律是1个1和3个0，可以推导出：
       - 32位补码的最小值：1个1和31个0（100000000000.....）
     - 1111为4位补码的-1，规律是4个1，可以推导出：
       - 32位补码的-1：32个1（111111111111.....）

```
int max = Integer.MAX_VALUE; //int的最大值
int min = Integer.MIN_VALUE; //int的最小值
System.out.println(Integer.toBinaryString(max)); //01111111...
System.out.println(Integer.toBinaryString(min)); //10000000...
System.out.println(Integer.toBinaryString(-1)); //1111111...
```

- 深入理解负值：
  - 记住32位2进制数的-1的编码为：32个1
  - 负数计算：用-1减去0位置对应的权

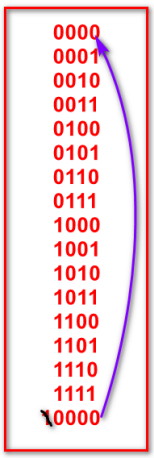
> 11111111111111111111111111111111 = -1
>
> 11111111111111111111111111111101 = -1-2 = -3
>
> 11111111111111111111111111111001 = -1-2-4 = -7
>
> 11111111111111111111111111111000 = -1-1-2-4 = -8
>
> 11111111111111111111111101101011 = -1-4-16-128 = -149
>
> 11111111111111111111111101101100 = -1-1-2-16-128 = -148

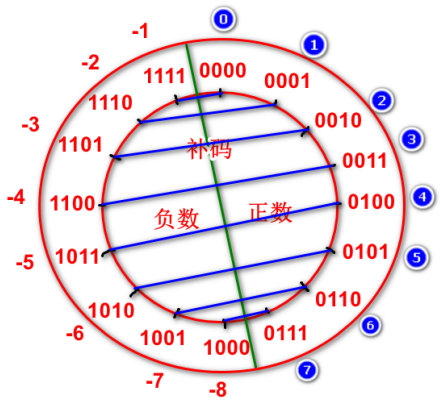
```
int n = -7;
System.out.println(Integer.toBinaryString(n)); //以2进制输出
```

- 互补对称现象：-n=~n+1------按位取反再加1

> -7 = 11111111111111111111111111111001 = -1-2-4 = -1
>
> ~-7 = 00000000000000000000000000000110 = 2+4 = 6
>
> ~-7+1 = 00000000000000000000000000000111 = 1+2+4 = 7
>
> 5 = 00000000000000000000000000000101 = 1+4 = 5
>
> ~5 = 11111111111111111111111111111010 = -1-1-4 = -6
>
> ~5+1 = 11111111111111111111111111111011 = -1-4 = -5
>
> 12 = 00000000000000000000000000001100 = 4+8 = 12
>
> ~12 = 11111111111111111111111111110011 = -1-4-8 = -13
>
> ~12+1 = 11111111111111111111111111110100 = -1-1-2-8 = -12

```
int m = -7;
int o = ~m + 1;
System.out.println(o); //7
int i = 12;
int j = ~i + 1;
System.out.println(j); //-12
```





4. 位运算：
   - 取反：~

- 0变1，1变0

- 与运算：&
  - 运算规则：逻辑乘法，见0则0

> 0 & 0 ---> 0
>
> 0 & 1 ---> 0
>
> 1 & 0 ---> 0
>
> 1 & 1 ---> 1

```
/*
          1   7    9   d    5   d    9   e
n =     00010111 10011101 01011101 10011110
m =     00000000 00000000 00000000 11111111 0xff 8位掩码
k = n&m 00000000 00000000 00000000 10011110
 */

int n = 0x179d5d9e;
int m = 0xff; //8位掩码
int k = n&m;
System.out.println(Integer.toBinaryString(n));
System.out.println(Integer.toBinaryString(m));
System.out.println(Integer.toBinaryString(k));
//如上运算的意义：k中存储的是n的最后8位，这种运算叫做掩码运算
//一般从低位开始1的个数称为掩码的位数
//想要哪几位，就把掩码的那几位设计为1，其他位设计为0
```

- 或运算：|
  - 运算规则：逻辑加法，见1则1

> 0 & 0 ---> 0
>
> 0 & 1 ---> 1
>
> 1 & 0 ---> 1
>
> 1 & 1 ---> 1

```
/*
n =     00000000 00000000 00000000 11011101 0xdd
m =     00000000 00000000 10011101 00000000 0x9d00
k = n|m 00000000 00000000 10011101 11011101 0x9ddd
 */

int n = 0xdd;
int m = 0x9d00;
int k = n | m; //将n和m错位合并
System.out.println(Integer.toBinaryString(n));
System.out.println(Integer.toBinaryString(m));
System.out.println(Integer.toBinaryString(k));
//如上运算的意义：两个数的错位合并
```

- 右移位运算：>>>（无符号）

  待补充：>>（有符号右移）
  - 运算规则：将2进制数整体向右移动，低位自动溢出舍弃，高位补0

```
/*
            6    7    d    7    8    f    6    d
n =       0110 0111 1101 0111 1000 1111 0110 1101
m = n>>>1 0011 0011 1110 1011 1100 0111 1011 0110
k = n>>>2 0001 1001 1111 0101 1110 0011 1101 1011
g = n>>>8 0000 0000 0110 0111 1101 0111 1000 1111
 */

int n = 0x67d78f6d;
int m = n>>>1;
int k = n>>>2;
int g = n>>>8;
System.out.println(Integer.toBinaryString(n));
System.out.println(Integer.toBinaryString(m));
System.out.println(Integer.toBinaryString(k));
System.out.println(Integer.toBinaryString(g));
```

- 左移位运算：<<
  - 运算规则：将2进制数整体向左移动，高位自动溢出舍弃，低位补0

```
            6    7  d   7   8   f   6   d
n =        01100111110101111000111101101101
m = n<<1   11001111101011110001111011011010
k = n<<2   10011111010111100011110110110100
g = n<<8   11010111100011110110110100000000

int n = 0x67d78f6d;
int m = n<<1;
int k = n<<2;
int g = n<<8;
System.out.println(Integer.toBinaryString(n));
System.out.println(Integer.toBinaryString(m));
System.out.println(Integer.toBinaryString(k));
System.out.println(Integer.toBinaryString(g));
```

- 移位运算的数学意义：

```
int n = 5;
int m = n<<1; //int m = n*2;
int k = n<<2;
int q = n<<3;
System.out.println(m); //10
System.out.println(k); //20
System.out.println(q); //40
/*
   权: 64  32  16  8  4  2  1
   n:              0  1  0  1 = 5
   m:           0  1  0  1    = 8+2 = 10
   k:        0  1  0  1       = 16+4 = 20
   q:  0  1  0  1             = 32+8 = 40
 */
```

补充：

1. 二进制数看正负的时候，一定先把32位补全：

    - 补全后最高位为1表示负的，最高位为0表示正的
2. 计算正数：将一个2进制数每个1位置的权相加即可

   计算负数：用-1减去0位置对应的权