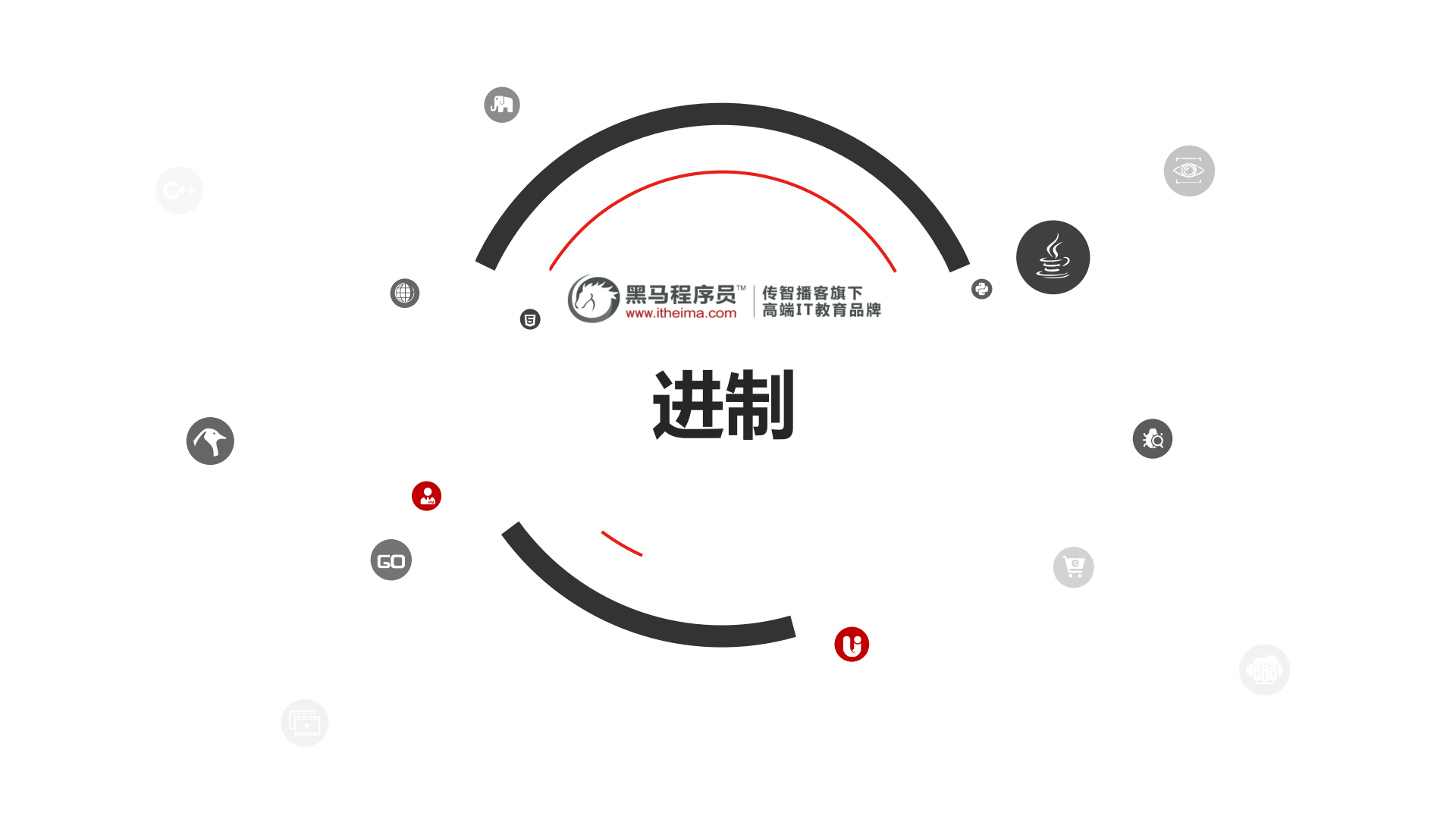

黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
进制

目录 Contents

进制介绍

进制：指进位制，是人们规定的一种进位方式

表示某一位置上的数，运算时是逢X进一位。

十进制是逢十进一，二进制就是逢二进一，八进制是逢八进一...

常见进制：二进制，八进制，十进制，十六进制

进制

为什么要学习进制?

原因：计算机数据在底层运算的时候，都是以二进制形式

也有数据是以八进制、十进制、或者十六进制进行存储或运算，了解不同的进制，便于我们对数据的运算过程理解的更加深刻。

进制

十进制

运算规则：逢十进一，借一当十。

```
  13
+  17
------
  20
```

```
②
2被借走了1，还剩下1，1落下去

①
20
-  7   0减7不够，向前借1，当做10进行运算
       10-7=3
------
13
```

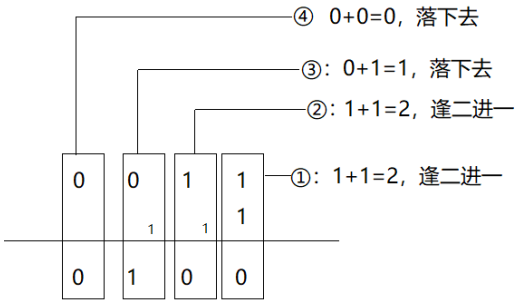
进制

二进制

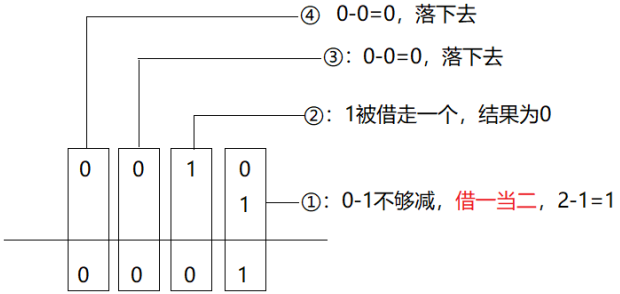
介绍：二进制数据是用0和1两个数码来表示。例如：0101000

进位规则是“逢二进一”，借位规则是“借一当二”。

二进制计算：0011 + 1



二进制计算：0010 - 1



八进制和十六进制

- **八进制介绍**：采用0，1，2，3，4，5，6，7八个数字，逢八进1

- **十六进制介绍**：用数字0到9和字母A到F（或a~f）表示，其中:A~F表示10~15，这些称作十六进制。
【0】【1】【2】【3】【4】【5】【6】【7】【8】【9】【a】【b】【c】【d】【e】【f】

进制

不同进制的书写格式

- 十进制：Java中，数值默认都是10进制，不需要加任何修饰。
- 二进制：数值前面以0b开头，b大小写都可以。
- 八进制：数值前面以0开头。
- 十六进制：数值前面以0x开头，x大小写都可以。

注意：以上内容是jdk7版本之后才被支持。

目录 Contents

进制

进制转换

- 二进制到十进制的转换
- 公式：系数 * 基数的权次幂 相加
  - 系数：每一【位】上的数
  - 基数：几进制，就是几
  - 权：从数值的右侧，以0开始，逐个+1增加

例如：0b100

拆解：0b为二进制标识

系数：1 0 0

基数：2（因为当前数值是2进制）

权 ：从右侧开始，以0为编号，逐个加1

0 ---- 0

0 ---- 1

1 ---- 2

套入公式：系数*基数的权次幂相加

0 * 2 ^ 0 = 0

0 * 2 ^ 1 = 0

1 * 2 ^ 2 = 4

结果：0 + 0 + 4 = 4

进制

进制转换

- 十六进制到十进制的转换
- 公式：系数 * 基数的权次幂 相加
  - 系数：每一【位】上的数
  - 基数：几进制，就是几
  - 权：从数值的右侧，以0开始，逐个+1增加

例如：0x100

拆解：0x为十六进制标识

系数：1　0　0

基数：16（因为当前数值是16进制）

权：从右侧开始，以0为编号，逐个加1

0 ---- 0

0 ---- 1

1 ---- 2

套入公式：系数*基数的权次幂相加

0 * 16 ^ 0 = 0

0 * 16 ^ 1 = 0

1 * 16 ^ 2 = 256

结果：0 + 0 + 256 = 256

进制

进制转换

- 总结：任意进制到十进制的转换
- 公式：系数 * 基数的权次幂 相加
  - 系数：每一【位】上的数
  - 基数：几进制，就是几
  - 权：从数值的右侧，以0开始，逐个+1增加

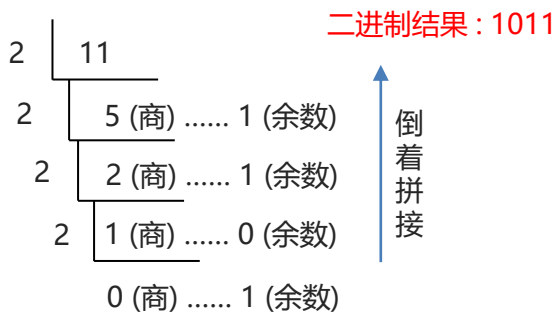
进制转换

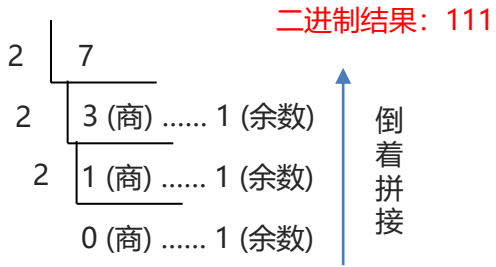
- 十进制到二进制的转换
- 公式：除基取余

  使用源数据，不断的除以基数（几进制，基数就是几）得到余数，直到商为0，再将余数倒着拼起来即可。

需求：将十进制数字11，转换为2进制。

实现方式：源数据为11，使用11不断的除以**基数**，也就是2，直到商为0。





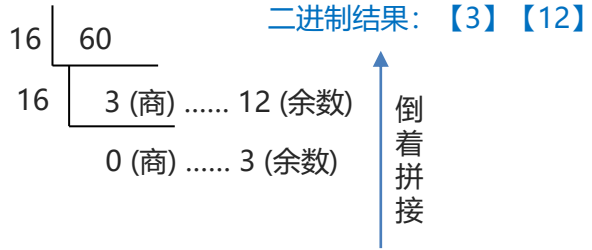
进制转换

- 十进制到十六进制的转换
- 公式：除基取余

  使用源数据，不断的除以基数（几进制，基数就是几）得到余数，直到商为0，再将余数倒着拼起来即可。

需求：将十进制数字60，转换为16进制。

实现方式：源数据为60，使用60不断的除以基数，也就是16，直到商为0。



注意：十六进制中，12使用C进行表示
结果：3C

进制转换

- 结论：十进制到任意进制的转换

- 公式：除基取余

  使用源数据，不断的除以基数（几进制，基数就是几）得到余数，直到商为0，再将余数倒着拼起来即可。

快速进制转换法

- 8421码：

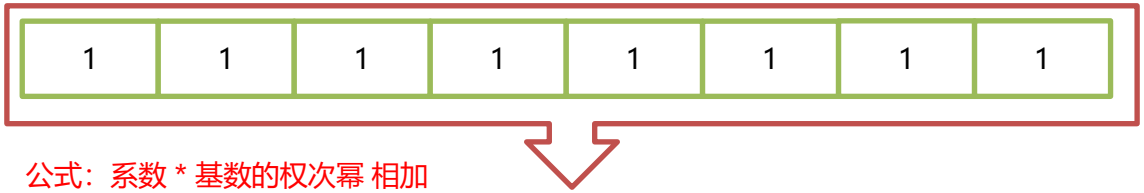
8421码又称BCD码，是BCD代码中最常用的一种

BCD： (Binary-Coded Decimal) 二进制码十进制数

在这种编码方式中，每一位二进制值的1都是代表一个固定数值，把每一位的1代表的十进制数加起来得到的结果就是它所代表的十进制数。

进制

二进制快速转十进制



可将2进制数值直接套入其中，0所对应的值不取，1所对应的值取出并相加

例如：二进制0b1101　　1　　1　　0　　1

8 + 4 + 1 = 13

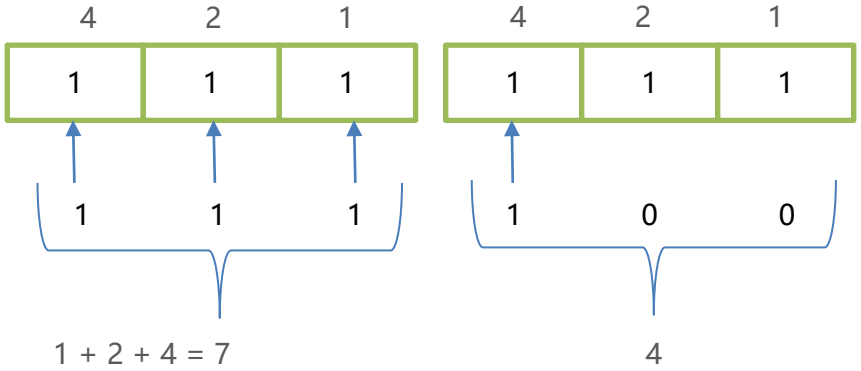
二进制0b1101，转10进制后，结果为13

二进制快速转八进制

八进制：将三个二进制位看为一组，再进行转换

原因：八进制逢八进一，三个二进制位最多可以表示111，也就是数值7，如果出现第四位，就超范围了

需求：将60的二进制0b111100转换为八进制

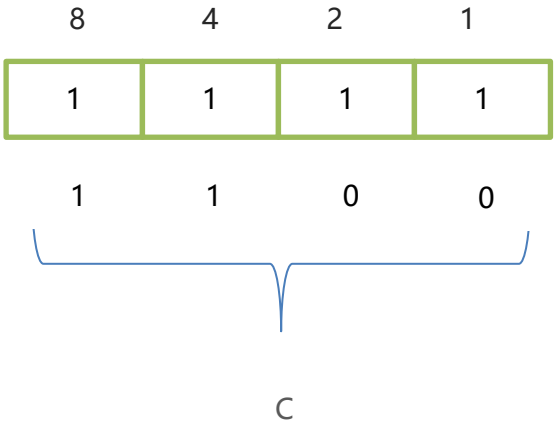


二进制快速转十六进制

十六进制：将四个二进制位看为一组，再进行转换

原因：十六进制逢十六进一，四个二进制位最多可以表示1111，也就是数值15，如果出现第五位，就超范围了

需求：将60的二进制0b111100转换为十六进制



目录 Contents

原码反码补码

- 为什么要学习原码反码补码?

```
public class Test {
	public static void main(String[] args) {
		// byte的取值范围：-128~127
		byte b = (byte) 130;
		System.out.println(b);
	}
}
```

- 运行结果

```
E:\>java Test
-126
```

原码反码补码

- 原码反码补码介绍

注意：计算机中的数据，都是以二进制补码的形式在运算，而补码则是通过反码和原码推算出来的。

- 原码（可直观看出数据大小）

  就是二进制定点表示法，即最高位为符号位，【0】表示正，【1】表示负，其余位表示数值的大小。

  通过一个字节表示+7和-7，代码：byte b1 = 7； byte b2 = -7;

  一个字节等于8个比特位，也就是8个二进制位

  0(符号位)　　0000111

  1(符号位)　　0000111

- 反码

  正数的反码与其原码相同；负数的反码是对其原码逐位取反，但符号位除外。

- 补码（数据以该状态进行运算）

  正数的补码与其原码相同；负数的补码是在其反码的末位加1。

原码反码补码

- 原码反码补码介绍

  正数的原反补都是相同的

  负数的【反码】，是根据【原码】取反(0变1，1变0)得到的 **(符号位不变)**

  负数的【补码】，是根据【反码】的末尾+1，得到的

- 求-7的补码

  原码： 1(符号位) 0000111 → 符号位不变，0变1，1变0

  反码： 1(符号位) 1111000 → 反码的末尾+1，求补码

  +1

  补码： 1(符号位) 1111001

问题：根据原码能慢慢推导补码，根据补码能否反向推导原码？

原码反码补码

- 原码反码补码介绍

```
byte b = (byte) 130;
System.out.println(b);
```

① 整数130：默认为int，int占用4个字节，也就是4组8个二进制位

```
00000000 00000000 00000000 10000010
```

② 强转到byte：4个字节，强制转换为1个字节，就是砍掉前3组8位

```
10000010
```

③ 根据运算后的补码，反向推原码

```
补码 -> 反码： 末尾-1
        10000010
             -1
        ----------
        10000001

反码 -> 原码： 符号位不变，其余数值逐位取反
        11111110
```

④ 使用8421码开始计算

二进制原码	1	1	1	1	1	1	1	0
8142码	-	64	32	16	8	4	2	1

2 + 4 + 8 + 16 + 32 + 64 = 126

拼上符号位即是 -126

目录 Contents

位运算符

位运算符介绍

- 思考：System.out.println(6 & 2); 结果是多少?

位运算符

位运算符介绍

- 位运算符指的是二进制位的运算，先将十进制数转成二进制后再进行运算。
- 在二进制位运算中，1表示true，0表示false。

位运算符

位运算符介绍

符号	计算方式
&	遇到0（false）则0（false），两边同时为1（true），结果才是1（true）
|	遇到1（true）则1（true），两边都是0（false），结果才是0（false）
^	相同为false，不同为true
~	取反，二进制位全部取反，0变1，1变0，包括符号位
<<	有符号左移运算，左边符号位丢弃，右边补齐0
>>	有符号右移运算，根据符号位，补齐左边
>>>	无符号右移，无论最符号位是0还是1，都补0

位运算符

异或运算的特点

- 一个数，被另外一个数，异或两次，该数本身不变。

位运算符

案例：数据交换

需求：已知两个整数变量a = 10，b = 20，使用程序实现这两个变量的数据交换
最终输出a = 20，b = 10;

思路：

① 定义一个三方变量temp，将a原本记录的值，交给temp记录 （a的值，不会丢了）
② 使用 a 变量记录 b 的值，（第一步交换完毕，b的值也丢不了了）
③ 使用 b 变量记录 temp的值，也就是a原本的值 （交换完毕）
④ 输出 a 和 b 变量即可

位运算符

案例：数据交换

需求：已知两个整数变量a = 10，b = 20，使用程序时间这两个变量的数据交换

最终输出a = 20，b = 10;

不允许使用三方变量

```
public static void main(String[] args) {
    int a = 10;
    int b = 20;
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;
    System.out.println(a);
    System.out.println(b);
}
```

```
a = a ^ b;    // a = 10 ^ 20;
b = a ^ b;    // b = 10 ^ 20 ^ 20;
a = a ^ b;    // a = 10 ^ 20 ^ 10;
```

案例：反转

案例：反转

需求：已知一个数组 **arr = {19, 28, 37, 46, 50};** 用程序实现把数组中的元素值交换，
交换后的数组 **arr = {50, 46, 37, 28, 19};** 并在控制台输出交换后的数组元素。

数据	19	28	37	46	50
索引	0	1	2	3	4

反转后

数据	50	46	37	28	19
索引	0	1	2	3	4

案例：反转

案例：反转

数据	19	28	37	46	50
索引	0	1	2	3	4

数据	50	46	37	28	19
索引	0	1	2	3	4

案例：反转

案例：反转

数据	19	28	37	46	50
索引	0	1	2	3	4

```
int temp = arr[0]
arr[0] = arr[4]
arr[4] = temp;
```

案例：反转

案例：反转

数据	50	28	37	46	19
索引	0	1	2	3	4

```
int temp = arr[1]
arr[1] = arr[3]
arr[3] = temp;
```

案例：反转

案例：反转

数据	50	28	37	46	19
索引	0	1	2	3	4

```
int temp = arr[1]
arr[1] = arr[3]
arr[3] = temp;
```

案例：反转

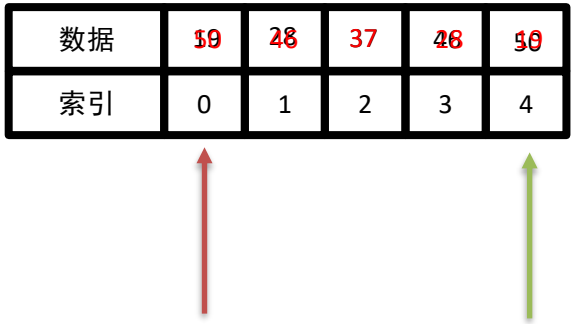
案例：反转

分析：交换的代码不止一次，可以使用循环进行

```
for (          初始化语句;          条件判断语句;          条件控制语句;  ){


                        循环体语句;


}
```



int start = 0 , end = arr.length - 1;

```
int temp = arr[0]
arr[0] = arr[4]
arr[4] = temp;
```

start++ , end --

传智播客旗下高端IT教育品牌