**二进制**

**2进制在哪里?**

计算机内部 **只有** 2进制数据!

int i = 50; System.out.println(i);//"50" Integer.toString(i)

Java编程语言, 编程语言利用算法支持10进制, 使用用户感受上可以使用10进制!

编程语言: 人类与计算机沟通的桥梁!

显示2进制数据

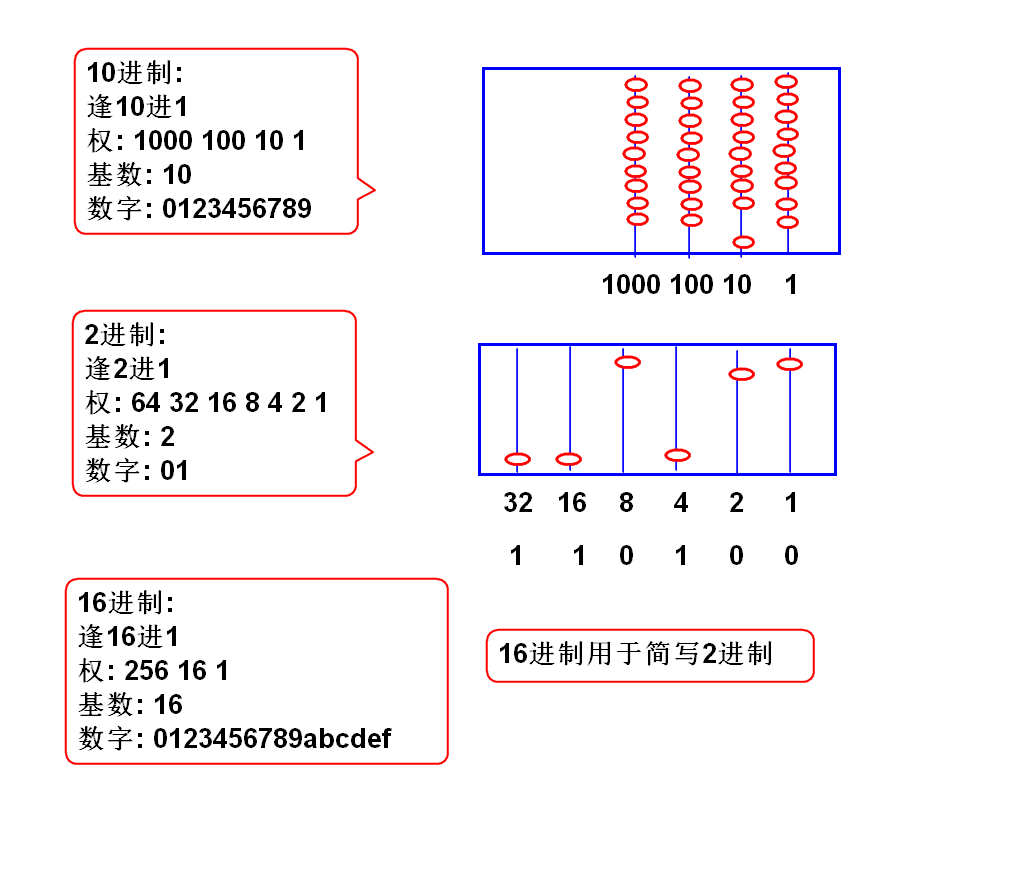
int i = 50;// Integer.parseInt(); System.out.println(Integer.toBinaryString(i));

* Java 如何接收10进制数据: Integer.parseInt();
  + 将10进制字符串转换为2进制int
* Java 如何输出10进制数据: Integer.toString();
  + 将2进制int转换为10进制字符串

Scanner in = new Scanner(System.in); int n = in.nextInt();// 底层调用了 Integer.parseInt();

**什么是2进制**

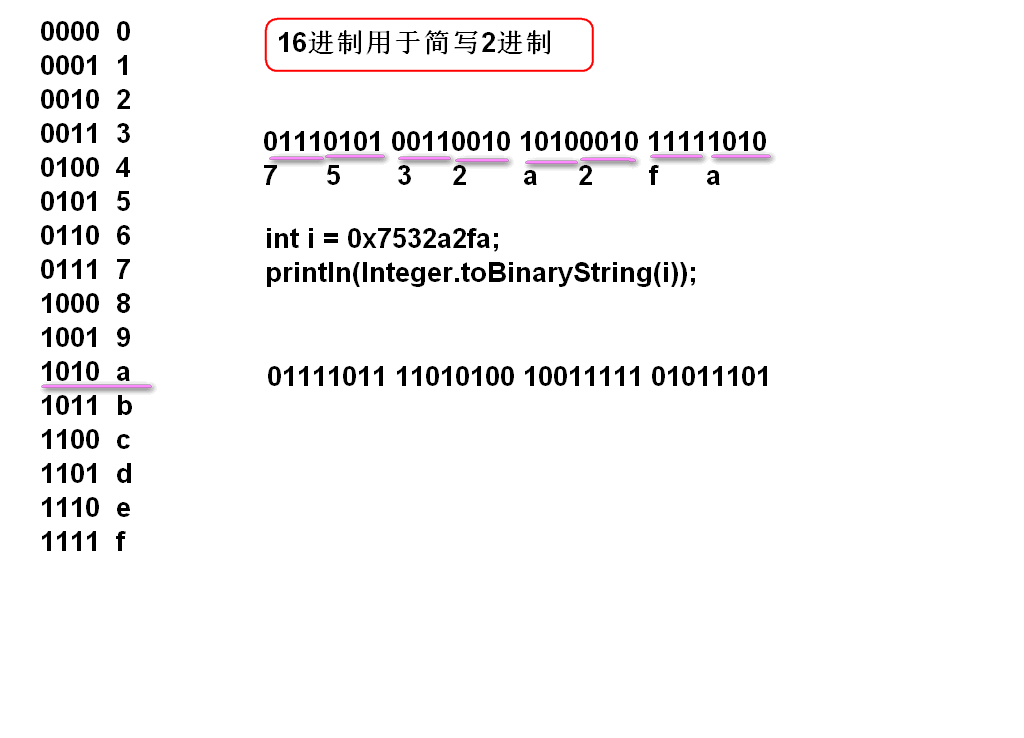
逢2进1的计数规则



**16进制**

16进制用于简写2进制

4位2进制可以简写为一位16进制数字



经典面试题:

int i = 0x32; //0011 0010 System.out.println(i); 如上代码输出结果( ) 答案: 50 int i = 0xac; System.out.println(i); 如上代码输出结果( ) 答案:

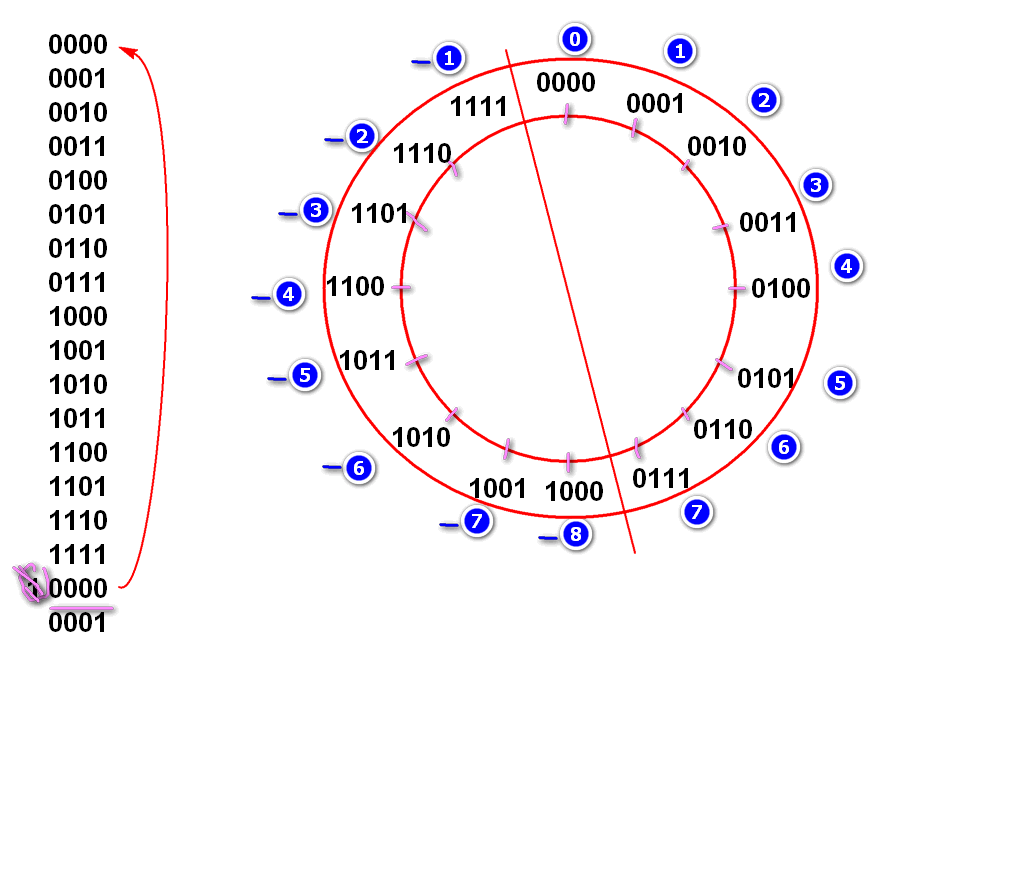
**补码**

计算机处理 **有符号数(负数)** 问题的算法.

补码算法: 4位数补码

规则:

1. 总的位数是4位数
2. 如果计算结果超过4位自动溢出舍弃



经典面试题目:

int n = 0xffffffff; System.out.println(n); 如上代码输出结果是:( ) A.2147483647 B.-2147483648 C.-2147483647 D.-1 答案: D 判断对错: 正数相加溢出结果是负数? 答案: 错! System.out.println(~8+1); 输出结果是( ) 答案: -8 System.out.println(~8); 输出结果是( ) 答案: -9 System.out.println(~-8); 输出结果是( ) 答案: 7

**2进制运算符**

~ 取反(非) & 与运算 | 或运算 >>> 逻辑右移动运算 >> 数学右移动运算 << 左移动运算

& 与运算

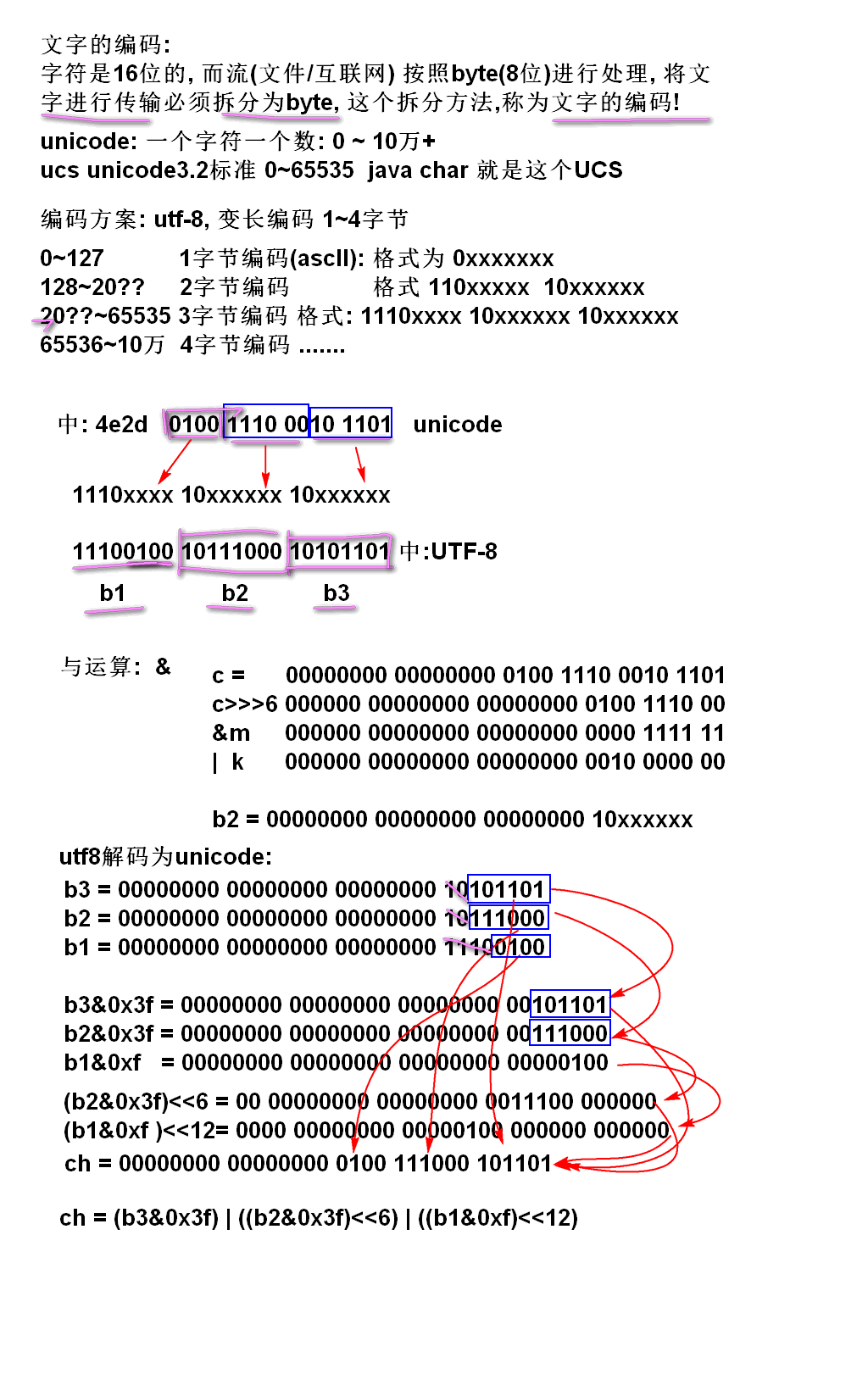
0 & 0 = 0 0 & 1 = 0 1 & 0 = 0 1 & 1 = 1 n = 01001001 01110111 10101010 01010101 m = 00000000 00000000 00000000 00111111 &--------------------------------------- k = 00000000 00000000 00000000 00010101 int n = '中'; int m = 0x3f; int k = n&m; println(Integer.toBinaryString(n)); println(Integer.toBinaryString(m)); println(Integer.toBinaryString(k));

| 或运算(逻辑+)

0 | 0 = 0 0 | 1 = 1 1 | 0 = 1 1 | 1 = 1 n = 00000000 00000000 00000000 00101101 m = 00000000 00000000 00000000 10000000 | ---------------------------------------- k = 00000000 00000000 00000000 10101101 int n = 0x2d; int m = 0x80; int k = n | m; println(Integer.toBinaryString(n)); println(Integer.toBinaryString(m)); println(Integer.toBinaryString(k));

>>> 逻辑右移动运算

n = 00100010 01111010 10101010 10111011 m = n>>>1 000100010 01111010 10101010 1011101 k = n>>>2 0000100010 01111010 10101010 101110 int n = 0x227aaabb; int m = n>>>1; int k = n>>>2; int j = n>>>6; //按照2进制输出



**移位运算的数学意义**

移动小数点运算:

124238. 小数点向右移动 1242380. 小数点向右移动 1次, 数值\*10 12423800. 小数点向右移动 2次, 数值\*10\*10 假设小数点位置不变 124238. 数字向左移动 1242380. 数字向左移动 1次, 数值\*10 12423800. 数字向左移动 2次, 数值\*10\*10

2进制数字移动, 与10进制数字移动规律相同: 2进制数值向左移动一次, 数值\*2

n = 00000000 00000000 00000000 00110010. = 50 n<<1 = 0000000 00000000 00000000 001100100. = 100 n<<2 = 000000 00000000 00000000 0011001000. = 200

案例:

int n = 50; int m = n<<1; println(m); //100

>> 数学右移位计算:

1. 相当于将原数据进行除法, 结果向小方向取整数.
2. >> 数学移位: 正数高位补0, 负数高位补1, 有数学意义
3. >>> 逻辑移位: 高位补0, 负时候没有数学意义

案例:

n = 11111111 11111111 11111111 11110111 -9 n>>1 = 111111111 11111111 11111111 1111011 -5 n>>>1= 011111111 11111111 11111111 1111011 很大

案例:

...

一般情况下单纯的将数字向右移动, 请使用 >>>

面试题:

优化 n\*8 的计算为 ( ) 答案: n<<3 优化 n\*16 的计算为 ( ) 答案: n<<4

**作业**

1. 实现utf-8编码算法:

public byte[] utf8(char ch){}

1. 实现utf-8编码算法:

public byte[] utf8(String ch){}

必须使用String的解码算法建议编码是否正确

# 反射

## 什么是反射

反射是Java提供的动态执行机制, 可以动态加载类, 动态创建对象, 动态访问属性, 动态调用方法..

静态执行: Java代码经过编译以后就确定的执行次序, 称为静态执行次序

Foo foo = new Foo(); foo.test();

动态执行: 在运行期间才确定创建那个类的对象, 执行那个方法!

Java 反射API可以实现动态执行!

案例: 执行某个类中全部的以test为开头的方法. (必须使用使用反射实现)

反射:

1. 反射是Java提供的API, 接受API的提供的功能!
2. 是Java底层的执行机制.

## 反射功能

### 动态加载类

Class cls = Class.forName(类名)

作用: 将类名对应的类加载到方法区, 如果类名错误则抛出异常.

### 动态创建对象

Object obj = cls.newInstance();

特点:

* 动态创建对象
* 可以创建任何对象
* cls对应的类必须有无参数构造器!!!
* 如果没有无参数构造器则抛出异常!
* 反射API利用Constructer API支持有参数构造器, 略...

### 反射可以查询类中的方法

可以返回类中声明的全部方法信息

Method[] methods = cls.getDeclaredMethods(); for(Method m:methods){ System.out.println(m);//输出方法信息 } Declared 声明的 Method 方法

### 动态执行方法

执行方法:

1. 必须有对象
2. 找到对象对应的类型方法信息
   * 方法信息在类上查找

案例:

Method m = cls.getDeclaredMethod(方法名);//在类信息上查找一个方法信息 //m 代表 cls类上的一个方法信息 Object obj=cls.newInstance();//动态创建对象 //动态调用(invoke)方法 Object val = m.invoke(obj);

### 反射用途

1. Eclipse 的快捷菜单使用了反射, 利用反射发现了类的属性和方法
2. Spring 利用了反射
   * 动态加载类
   * 动态创建Bean
   * 动态注入属性, 包括私有属性注入
   * 动态解析注解
3. MyBatis 利用了反射
   * 查询时候, 动态了将查询结果利于反射注入到Bean返回
4. JUnit 使用了反射
5. 注解的解析使用了反射
6. Servlet调用使用了反射