
计算机中 能够存的数据的最小单位是“字节”



1个字节是8个bit位

1024个字节是1kb

1024kb是1M

1024M是1G

一、算数运算符

符号	作用	说明
+	加	参看小学一年级数学
-	减	参看小学一年级数学
*	乘	参看小学二年级数学，与“×”等同
/	除	参看小学二年级数学，与“÷”等同
%	取模、取余	获取的是两个数据做除法的余数

■ 注意事项：

/ 和 % 的区别：两个数据做除法，/ 取结果的商，% 取结果的 **余数**。

整数操作只能得到整数，要想得到小数，必须有浮点数参与运算。

二、类型转换

1.隐式转换

- 隐式转换的两种提升规则

- 取值范围小的,和取值范围大的进行运算,小的会先提升为大的,再进行运算

```

public class Test {
    public static void main(String[] args) {
        int a = 10;
        double b = 12.3;
        数据类型? c = a + b;
        double类型
    }
}

```

- byte short char三种类型的数据在运算的时候,都会直接先提升为int,然后再进行运算

```

public class Test {
    public static void main(String[] args) {
        byte a = 10;
        byte b = 20;
        数据类型? c = a + b;
        int 类型
    }
}

```

取值范围最大

2.隐式转换小结

- 取值范围
 - byte>short>int>long>float>double
- 什么时候转换
 - 数据类型不一样，不能进行计算，需要转换成一样的才可以计算
- 转换规则1：
 - 取值范围小的，和取值范围大的进行运算，小的会先提升为大的在进行计算
- 转换规则2
 - byte short char 三种类型的数据在运算的时候，都会直接提升为int，然后再进行运算

三、强制转换

- 如果把一个取值范围大的数值，赋值给取值范围小的变量。是不允许直接赋值的。如果一定要这么做就需要加入强制转换
- 格式：目标数据类型 变量名=(目标数据类型)被强转的数据

```
byte b1= 10;  
byte b2=20;  
byte b3=(byte)(b1+b2);
```

四、+ 加号运算符

- “字符+字符”当"+"操作中出现字符串时,这个"+"是字符串连接符,而不是算术运算符了.会将前后的数据进行拼接,并产生一个新的字符串.
- “字符+数字”时,会把字符通过ASCII码表查询到对应的数字再进行计算.

```
System.out.println(1 + 'a'); //98
```

```
System.out.println('a' + "abc"); //"aabc"
```

五、自增自减运算符

两种用法

单独使用

++ 和 -- 无论是放在变量的前边还是后边，单独写一行结果是一样的。

参与计算

1

`int a = 10;
int b = a++;`

后++
先用后加

2

`int a = 10;
int b = ++a;`

前++
先加后用

先把a变量中的值拿出来，赋值给b,然后再进行自增

先把a进行自增，然后把自增后的结果赋值给左边b

```
int x = 10;  
//后++：先用后加  
//先把x变量中的值拿出来，赋值给y，然后再进行自增，  
//赋值给y的值是自增前的。  
int y = x++; // x = 11 y = 10  
//先++：先加后用  
//先把x进行自增，然后把自增后的结果赋值给左边的变量  
//先把x自增，变成12，然后再把自增之后的12赋值给z  
int z = ++x; // x = 12 z = 12
```

六、赋值运算符

符号	作用	说明
=	赋值	int a=10, 将10赋值给变量a
+=	加后赋值	a+=b, 将a+b的值给a a=a+b
-=	减后赋值	a-=b, 将a-b的值给a a=a-b
=	乘后赋值	a=b, 将a×b的值给a
/=	除后赋值	a/=b, 将a÷b的商给a
%=	取余后赋值	a%=b, 将a÷b的余数给a

这些运算符，都隐藏了一个条件，就是强制类型转换 如： short a=1;
a+=1; ——> a=(short)a+1;

■ 注意事项： 扩展的赋值运算符隐含了强制类型转换

七、关系运算符

(关系运算符/比较运算符) 的分类

符号	说明
==	a==b, 判断a和b的值是否相等, 成立为true, 不成立为false
!=	a!=b, 判断a和b的值是否不相等, 成立为true, 不成立为false
>	a>b, 判断a是否大于b, 成立为true, 不成立为false
>=	a>=b, 判断a是否大于等于b, 成立为true, 不成立为false
<	a<b, 判断a是否小于b, 成立为true, 不成立为false
<=	a<=b, 判断a是否小于等于b, 成立为true, 不成立为false

■ 注意事项: 关系运算符的结果都是boolean类型, 要么是true, 要么是false。
千万不要把“==”误写成“=”。

八、逻辑运算符

符号	作用	说明	
&	逻辑与 (且)	并且, 两边都为真, 结果才是真	两边都要满足
	逻辑或	或者, 两边都为假, 结果才是假	两边满足一个
^	逻辑异或	相同为 false, 不同为 true	
!	逻辑非	取反	

九、短路逻辑运算符

符号	作用	说明
&&	短路与	结果和&相同，但是有短路效果
	短路或	结果和 相同，但是有短路效果

■ 注意事项：

◆ &|，无论左边 true false，右边都要执行。

&& ||，如果左边能确定整个表达式的结果，**右边不执行**。

&&: 左边为false，右边不管是真是假，整个表达式的结果一定是false。

||: 左边为true，右边不管是真是假，整个表达式的结果一定是true。

这两种情况下，右边不执行，提高了效率。

◆ 最常用的逻辑运算符：**&&**，**||**，**!**

十、三元运算符

- 条件（三元）运算符是 Java 唯一使用三个操作数的运算符：一个条件后跟一个问号（?），如果条件为[真值]，则执行冒号（:）前的表达式；若条件为[假值]，则执行最后的表达式。该运算符经常当作 [if...else] 语句的简捷形式来使用。

(三元运算符/三元表达式) 格式

■ 格式：关系表达式 **?** 表达式1 **:** 表达式2;

■ 范例：求两个数的较大值。

```
int max = a > b ? a : b; 把三元运算符的结果赋值给一个变量
System.out.println(a > b ? a : b);
```

十一、运算符的优先级

优先级	运算符
1	. () {}
2	!、~、++、--
3	*、/、%
4	+、-
5	<<、>>、>>>
6	<、<=、>、>=、instanceof
7	==、!=
8	&
9	^
10	
11	&&
12	
13	?:
14	=、+=、-=、*=、/=、%=、&=、

- 只用记住一点“小括号()”优先于所有，想要先算谁，就用小括号将其括上

十二、原码，反码，补码

- 计算机中，最小的存储单元是“一个字节”它占8个bit位。范围从1000 0000 ~ 0111 1111 (-128~127)
-

十进制数字	原码	反码	补码
+0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	0000 0000
-1	1000 0001	1111 1110	1111 1111
-2	1000 0010	1111 1101	1111 1110
-3	1000 0011	1111 1100	1111 1101
-4	1000 0100	1111 1011	1111 1100
...
-126	1111 1110	1000 0001	1000 0010
-127	1111 1111	1000 0000	1000 0001
-128	无	无	1000 0000

- 计算机中，数字的存储，以及运算都是以补码的形式来操作的。

原码

十进制数据的二进制表现形式，最左边是符号位，0为正，1为负。

原码的弊端

利用原码进行计算的时候，如果是正数完全没有问题。

但是如果是负数计算，结果就出错，实际运算的方向，跟正确的运算方向是**相反**的。

反码出现的目的

为了解决原码不能计算负数的问题而出现的。

反码的计算规则

正数的反码不变，负数的反码在原码的基础上，符号位不变。数值取反，0变1，1变0。

反码的弊端

负数运算的时候，如果结果不跨0，是没有任何问题的，但是如果结果跨0，跟实际结果会有1的偏差。

补码出现的目的

为了解决负数计算时跨0的问题而出现的。

补码的计算规则

正数的补码不变，负数的补码在反码的基础上+1。

另外补码还能多记录一个特殊的值-128，该数据在1个字节下，没有原码和反码。

补码的注意事项

计算机中的存储和计算都是以补码的形式进行的。

理解了原码，反码，补码，的基本概念，就可以深入了解以下内容了。

- 1.理解同一个数字在不同数据类型下到底有什么区别呢？bit位
 - 就是在前面的位置“补0”
 - 基本数据类型

byte类型的10	1个字节	0000 1010	1个字节，8个byte位
short类型的10	2个字节	0000 00000000 1010	2个字节，16个byte位
int类型的10	4个字节	0000 0000 0000 0000 0000 0000 0000 1010	
long类型的10	8个字节	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1010	

• 2.理解隐式转换

隐式转换

o

```
public class Test {  
    public static void main(String[] args) {  
        byte a = 10; // 0000 1010  
        int b = a; // 0000 0000 0000 0000 0000 0000 0000 1010  
        System.out.println(b); 前面的字节都补零  
    }  
}
```

• 3.理解强制转换

强制转换

o

```
public class Test {  
    public static void main(String[] args) {  
        int a = 300; // 0000 0000 0000 0000 0000 0001 0010 1100  
        byte b = (byte)a; // 0010 1100 去掉前面的字节  
        System.out.println(b); // 44 所以强转会丢失精度  
    }  
}
```

• 4.理解数字之间使用“运算符”

其他的运算符

o

运算符	含义	运算规则
&	逻辑与	0为false 1为true
	逻辑或	0为false 1为true
<<	左移	向左移动，低位补0
>>	右移	向右移动，高位补0或1
>>>	无符号右移	向右移动，高位补0

运算符	含义	运算规则
&	逻辑与	0为false 1为true

```
public class Test {
    public static void main(String[] args) {
        int a = 200;
        int b = 10;
        System.out.println(a & b);
    }
}
```

0000 0000 0000 0000 0000 0000 1100 1000

& 0000 0000 0000 0000 0000 0000 0000 1010

0000 0000 0000 0000 0000 0000 0000 1000

	逻辑或	0为false 1为true
--	-----	----------------

```
public class Test {
    public static void main(String[] args) {
        int a = 200;
        int b = 10;
        System.out.println(a | b);
    }
}
```

0000 0000 0000 0000 0000 0000 1100 1000

| 0000 0000 0000 0000 0000 0000 0000 1010

0000 0000 0000 0000 0000 0000 1100 1010

<<	左移	向左移动，低位补0
----	----	-----------

```
public class Test {
    public static void main(String[] args) {
        int a = 200;
        System.out.println(a << 2);
    }
}
```

0000 0000 0000 0000 0000 0000 1100 100000

左移两次: 200*4=800

800

小规则: 左移一次就 "X2"

>>	右移	向右移动，高位补0或1
----	----	-------------

```
public class Test {
    public static void main(String[] args) {
        int a = 200;
        System.out.println(a >> 2);
    }
}
```

000000 0000 0000 0000 0000 0000 1100 1000

右移两次: 200/4=50

50

高位符号位: 原来是正数补0, 原来是负数补1

小规则: 右移一次就 "÷2"

>>>	无符号右移	向右移动，高位补0
-----	-------	-----------

无符号右移: 表式 最高位符号位只补0, 即只有正数

© 版权声明

版权声明

1. 本网站名称: **AMA**
2. **AMA**提供的资源仅供您个人用于非商业性目的。
3. 本站文章部分内容可能来源于网络, 仅供大家学习与参考, 如有侵权, 请联系我进行删除处理。

4. 本站一切资源不代表本站立场，并不代表本站赞同其观点和对其真实性负责。
 5. 本站一律禁止以任何方式发布或转载任何违法的相关信息，访客发现请举报
 6. 本站资源大多存储在云盘，如发现链接失效，请联系我，我会第一时间更新。
 7. 本站强烈打击盗版/破解等有损他人权益和违法作为，请支持正版！
-