⚠ 赋值运算符号的注意事项: **前面提到byte类型在参加运算的时候就将精度提高到int类型，但是在赋值运算符不会看下列**

```
1
2
3  public class assignOperate {
4      //main
5      public static void main(String[] args) {
6          byte a = 3;
7          a += 3; //这里的不会将精度提高到int，它会经过类型转换变为 a = (byte)(a+3)
8          a = a+3; //这里就会报错
9          a++; //这里也是会变为(byte)(a+1)
10     }
11 }
```

三元运算符

基本语法：条件表达式?表达式1:表达式2

1.如果条件表达式为`true`，那么就会运算后的结果为表达式`1`的结果

2.如果条件表达式为`false`，那么就会运算后的结果为表达式`2`的结果

```
1  public class assignOperate {
2      //main
3      public static void main(String[] args) {
4          ....
5          //三元运算符
6          /*
7          1.如果条件表达式为`true`，那么就会运算后的结果为表达式`1`的结果
8
9          2.如果条件表达式为`false`，那么就会运算后的结果为表达式`2`的结果
10         */
11         int a = 10;
12         int b = 30;
13         int result = a>b?a--:b--;
14         System.out.println(result); //执行b--,先赋值30
15         System.out.println(b); //执行b--,后减等于29
16     }
17 }
```

⚠Java运算符的注意细节

- 表达式1和表达式2要为可以赋给接受变量的类型或者可以自动转换的类型

```
1          //表达式1和表达式2要为可以赋给接受变量的类型或者可以自动转换的类型
2          int result2 = a>b?1.1:1.2; //报错，精度不匹配1.2和1.1都是double类型
```

用三元运算求三个数中的最大值

```
1
2
3  //三元运算符
4  public class TemaryOperate {
5      //main
6      public static void main(String[] args){
7          int a = 10;
8          int b = 30;
9          int c = 40;
10         //求三个的最大值
11         int temp = a>b?a:b;
12         int result = temp>c?temp:c;
13         System.out.println("最大值为:"+result);
14         //一条语句
15         int max = (a>b?a:b)>c?(a>b?a:b):c;
16         System.out.println("最大值为:"+max);;
17     }
18 }
```

位运算

`>>`算术右移 `<<`算术左移，`>>>`无符号右移，`&`按位与,`|`按位或,`^`按位异或，`~`按位取反

运算符	功能
|	两位有一个为1，结果为1，否则为0
&	两位全为1，结果为1，否则为0
^	两位一个为0,一个为1，结果为1，否则为0
~	0->1 1->0
>>	低位溢出,符号位不变,并用符号位补溢出的高位
<<	符号位不变,低位补 0
>>>	逻辑右移也叫无符号右移,运算规则是: 低位溢出，高位补 0

原码、反码、补码

- 二进制的最高位是符号位: 0 为正，1为负
- 正数的原码，反码，补码都一样
- 负数的反码=
- 负数的补码=反码+1,负数的反码=负数的补码-1
- 0发反码补码都是0
- java中没有无符号数
- 在计算机运行的时候都是以**补码**的方式来运行的
- 看运行结构的时候，要看它的原码

测试题

```
1
2  //位运算
3
4  public class BitOperate {
5      //main
6      public static void main(String[] args){
7          //两个都是正数直接拿补码进行&运算
8          System.out.println(2&3);
9          //同理直接拿补码进行|运算
10         System.out.println(2|3);
11
12         /*
13         先求2的补码:      1101     求出来的补码是个负数
14         负数需要推出原码，退出原码就要求出反码，1101已经是个补码了，需要-1
15         -1后的反码:      1100
16         推出源码(符号位不变其他位取反): 1011 = -3
17
18         */
19         System.out.println(~2);
20         /*
21         -2的原码为:    1010
22         -2的反码为:    1101
23         +1求补码:      1110
24         在进行~运算:   0001 = 1
25         */
26         System.out.println(~-2);
27
28         /*
29         1的原码:    0001
30         算术右移动后:00 00 = 0   //后面两个00为低位补的0
31         */
32         System.out.println(1>>2);
33
34         /*
35         1的原码:    0001
36         算术左移动后:01 00 = 4   //后面两个00为低位补的0
```

```
        */
        System.out.println(1<<2);
        /*
        8的原码:   1000
        算术左移动后:00 10 = 2  //前面面两个00为高位补的0
        */
        System.out.println(8>>>2);

        //快速算
        System.out.println(15 >> 2 ); // 15 / 2 / 2 = 3
        System.out.println(15 << 2 ); // 15 * 2 * 2 = 60

    }
}
```

运算符号的优先级

	. () {} ; ,
R—>L	++ -- ~ !(data type)
L—>R	* / %
L—>R	+ -
L—>R	<< >> >>> 位移
L—>R	< > <= >= instanceof
L—>R	== !=
L—>R	&
L—>R	^
L—>R	|
L—>R	&&
L—>R	||
L—>R	? :
R—>L	= *= /= %=
	+= -= <<= >>=
	>>>= &= ^= |=