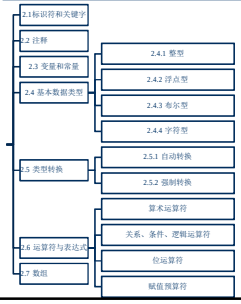
第二章 Java语言基础

基本程序设计



2.6 java中的运算符

- 算术运算符
- 关系和逻辑运算符
- 位运算符
- 赋值运算符
- 其他

2.6.1、算术运算符

类 别	运 算 符	用 法	描 述
双目运算符	+	op1 + op2	加，或字符串连接
	-	op1 – op2	减
	*	op1 * op2	乘
	/	op1 / op2	除（包括整数除和小数除）
	%	op1 % op2	求余
单目运算符	+	+ op	正值
	-	- op	负值
	+ +	+ + op , op + +	加1（不能是常量或表达式）
	--	-- op , op --	减1（不能是常量或表达式）

注意 / %运算的整数运算和小数运算

1、整数和浮点数除

- 整数除和浮点数除是有区别的：

1）整数之间做除法，则做整除运算，否则做精确除，(精度不同)

int x=2100 ; x=x/1000*1000

x=x/1000.0*1000 想想结果。

2）如果被除数是浮点数，除数就可以为0，

/ 结果是正或负数无穷:Infinity,不是浮点数则发生异常

- 5.0/0 = Infinity ； -5.0/0 = -Infinity
- 5/0 异常
- 4.5%0 =NaN //%浮点运算/0 为NaN

浮点运算注意

- (3)浮点数运算精度
- 因为浮点数有精度限制，小数部分转换成二进制有些时候并不是准确的，会使数学运算结果存在一定的误差，如：

4.0f-2.1f=1.9000001

4.0-2.1f=1.9000000953674316

现在如果主方法不用strictfp标记，就不会使用严格的浮点计算（也就是说不用截断。）

2、取模运算

- 取模运算，余数与被除数的符号相同，并向零靠近，例如：

如

5%2=? 5%-2=? , -5%2=? , -5%-2=?

5%2=1 5%-2=1, -5%2=-1, -5%-2=-1

double a=5.2, b=3.1 ; double c=a%b

double c=a%b=2.1 （不提倡浮点型取余）

3、自增++自减--：

```
public class TestDemo {
    public static void main(String[] args) {
     int i=1,j;
     ++i;     //i=2
     j=i++;   //i先执行赋值运算j=i，再自加为3
     System.out.println("j="+j); //j==2
     j=++i; //i先执行自加运算为4，再执行赋值运算j=i
     System.out.println("j="+j); //j和i的值为4
     j=(++i)+(++i)+(++i); //i先做++，再做加法运算j=5+6+7=18,i=7
     System.out.println("j="+j); //j==18
     j=(i++)+(i++)+(i++);//i先做加运算,再做++，j=7+8+9=24,i=10
     System.out.println("j=%d\n",j); //j==24
    }
}
```

4、+字符串连接功能

+还可做连接符。当“+”连接字符串和其他类型数据时，其他类型数据先转成字符串，再进行连接运算。

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

2.6.2、关系运算符

运算符	表达式	描述
>,<,>=,<=	例如：op1 > op2	大小比较
? :	a>b?a:b	If(a>b)then value is a else b
==,!=	A==true,a!=false	相等，不等
instanceof	op1 instanceof op2	对象op1是否属于类型op2

例如：

```
boolean b1=true,b2=false,r1,r2,r3;
b2<0; //错误，布尔数据不能与其他类型数据进行比较
r1=b1==b2; //两个布尔类型数据可以比较是否相等.r1=false
2<3<5   //错误，不支持数学上的连续比较
float f1=9.0000003F;
r2=f1>9; //r2=false,因为浮点数精确度原因，f1存储为9.0,
r3=（3-2.17==0.83）; //r3=false,3-2.17存储为0.8300000000000001
```

2.6.3 条件运算符

- Java 中唯一的三元运算符，其格式如下：

变量 = <布尔表达式> ? <表达式1> : <表达式2>

```
public class Max {
    public static void main(String args[]) {
        int x=20, y=30, max;
        max = (x>y) ? x : y;
        System.out.println(“max=”+max);
    }
}
```

输出结果：max=30

11

2.6.4 逻辑运算符

一个关系运算只能表示一个条件，如果一个问题有多个条件，这时可以用逻辑运算将多个条件连在一起。

运算符	用法	描述
&&	op1 && op2	逻辑与 （and ）
||	op1 || op2	逻辑或 （or）
!	! op1	逻辑非
^	op1 ^ op2	逻辑异或(xor) [两个值不同为true]
|	op1 | op2	逻辑或
&	op1 &op2	逻辑与

注意：

&和&&的区别在于，如果使用前者连接，那么无论任何情况， &两边的表达式都会参与计算。如果使用后者连接，当&&的左边为false，则将不会计算其右边的表达式。 “|”和“‖”的区别与 “&”和“&&”的区别同理。(*_*)

2.6.4 逻辑运算符

一个关系运算只能表示一个条件，如果一个问题有多个条件，这时可以用逻辑运算将多个条件连在一起。

分类	运算符	表达式	描述
与	&& 短路逻辑与	op1 && op2	左右操作数都为真时为真
	& 逻辑与	op1 &op2	左右操作数都为真时为真
或	|| 短路逻辑或	op1 || op2	左右语句有一则或超过一则为真时为真
	| 逻辑或	op1 | op2	左右语句有一则或超过一则为真时为真
非	! 逻辑非	! op1	取反：假时为真，真时为假
异或	^ 逻辑异或	op1 ^ op2	左右相异时为真，左右相同时为假

注意：

&和&&的区别在于，如果使用前者连接，&两边的表达式都会参与计算。如果使用后者连接，当&&的左边为false，则将不会计算其右边的表达式。

op1	op2	op1&op2	op1&&op2	op1|op2	op1||op2	!op1	op1^op2
true	true	true	true	true	true	false	false
true	false	false	false	true	true	false	true
false	true	false	false	true	true	true	true
false	false	false	false	false	false	true	false

例：分析下列程序的输出结果

```
1. boolean b1,b2,b3;
2. int a=5,b=6;
3. int i=100;
4. b1=(a>b)&&( ++i==1);     //b1=false,++i==1没有执行,i=100
5. b2=(a<b)||(i++==1); //b2=true,i++==1没有执行,i=100
6. b3=true||false&&false; //b3=true
```

优先级：逻辑非>逻辑与>逻辑异或>逻辑或。

4、位运算：逻辑位运算和移位运算

位运算是对操作数以二进制位为单位进行的操作，运算结果均为整型。位运算分为逻辑位运算和移位运算两种。

1、位运算

运算符	用 法	描 述
~	~ op	**按位取反**
&	op1 & op2	**按位与**
|	op1 | op2	**按位或**
^	op1 ^ op2	**按位异或**
>>	op1 >>op2	op1右移op2位（空出位由最高位数字填充）
>>>	op1>>>op2	op1无符号右移op2位 （空出位由0填充）
<<	op1 << op2	op1左移op2位 （空出位由0填充）

按位运算

操作数X	操作数Y	~X	X&Y	X|Y	X^Y
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

例如“25&-12”运算：

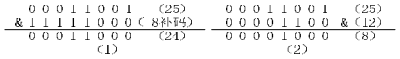
位运算的应用

1、按位反“~”应用

对二进制的每一位都进行取反操作，0变1,1变0。

2、按位与运算符&

- 对整数a的某些二进制位清0，其余位不变。需要选取合适的整数b，将对应于a要清零的位设为0，其余位设为1，b被称为掩码（mask）（1）。
- 取某个整数a中指定位的值，选取整数b，将a要取的位设为1，其余位为0 (2)



设要取整数的第3位和第4位可让该数与00001100进行与运算，最后结果可能是0（3,4位为0）、4（3位为1，四位为0）,8（3位为0，四位为1）和12（3,4位都为1）。

位运算的应用

- 3 按位或运算符|

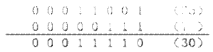
按位或可以用来把整数a的某些特定的位置1，其余位不变，方法是将掩码对应于a要置1的那些位设为1，其余设为0，

```
  0 0 0 1 1 0 0 1    (25)
| 0 0 0 0 0 1 1 1    (7 )
  0 0 0 1 1 1 1 1    (31)
```

位运算的应用

4、^运算：

- **按位异或可以实现对整数的某些二进制位取反，其余位不变。**



- **特性：如果对一个值自身做异或运算，返回0；一个数异或0，返回这个值本身。即：**

  a^a=0　　//任何数异或自己都等于0

  0^a =a　　//任何数异或0都等于它本身

位运算的技巧（针对，信息安全系的同学）

□ ^运算：

对于安全服务来讲，可用在加密和奇偶校验上。

a^b=c　　//加密

c^b=a^b^b=a^(b^b)=a^0=a,　即

c^b=a　　//解密

移位运算符

运算符	表达式	描述
<< （有符号左移）	op1<<op2	操作数 op1 的二进位顺序往左移动 op2 位。空出的低位用 0 填充，移出的高位舍弃不要。
>> （有符号右移）	op1>>op2	操作数 op1 的二进位顺序往右移动 op2 位。空出的高位用原来的最高位（符号位）填充，移出的低位舍弃。
>>> （无符号右移）	op1>>>op2	操作数 op1 的二进位顺序往右移动 op2 位。空出的高位用 0 填充，移出的低位舍弃。

位左移运算<<

每左移1位，相当于操作数乘以2，左移n位，相当于乘以2的n次方。

位左移运算的表达式形式：

操作数<<移动的位数

例如“-12<<3”运算将-12的各位顺序往左移动3位：

<<3	1	1	1	1	0	1	0	0	（-12 补码）
	1	0	1	0	0	0	0	0	（-96）

结果为-96，相当于$-12*2^3$。

位右移运算>>

每右移一位，相当于操作数除以2，右移n位，相当于除以2的n次方

表达式形式：操作数>>移动的位数

如图：4>>2=1。

>>2	0	0	0	0	0	1	0	0	（4）
	0	0	0	0	0	0	0	1	（1）

位右移运算可以使结果的符号与原操作数的符号相同，又称为算术右移。

无符号位右移运算>>>

对于正整数，每右移1位都相当于将操作数除以2，右移n位相当于除以2的n次方，但对于负数，无符号右移的结果与原数差距较大。位右移运算的表达式形式：

操作数>>>移动的位数

例如：

4>>>2=1;

-4>>>2=1073741822;

移位运算的注意事项

- 移位运算符适用数据类型有byte ,short ,char,int ,long
- 对低于int型的操作数系统将先自动转换为int型再移位
- 对于int型整数移位a>>b,系统系统先将b对32取模（低5位），得到的结果才是真正移位的位数，对long型对64取模（低6位）。
  如：a>>33 和a>>1结果一样，a>>32的结果还是a原来的数字。
- 移位不会改变变量本身的值，如a>>1；在一行语句中单独存在，毫无意义。

- x>>1的结果和x/2的结果是一样的，x<<2 和x*4的结果也是一样的。
- 总之，一个整数左移n位（如果没有越界），就是等于这个数乘以2的n次方，一个整数右移n位（如果没有越界），就是等于这个数除以2的n次方。
- 请思考：如何用移位运算实现求2的x次方。答案 y=1<<x .

5、赋值运算符

表达式	备注
a=b	赋一个值
a=b=1	same as a=(b=1)
a op=b eg a+=b	a=a op b 被允许的运算符 +,-, *, /, %, &, ^, |, <<,>>, >>>

其他运算符

.	分量运算符，用于对象属性或方法的引用
[ ]	下标运算符，应用于数组
instanceof	对象 instanceof 类名：判断某对象是否是一个类或其子类的实例。
new	对象实例化运算符，实例化一个对象，即为对象分配内存

1. 表达式及运算符的优先级

优先级	运算符	结合性
1	() [] .	从左向右
2	! ~ +(正) -（负） ++ --	从右向左
3	new	从左向右
4	/ * %	从左向右
5	+ -	从左向右
6	>> >>> <<	从左向右
7	< > <= >= instanceof	从左向右
8	==、!=	从左向右
9	&	从左向右
10	^	从左向右
11	|	从左向右
12	&&	从左向右
13	||	从左向右
14	? :	从右向左
15	= += -= *= /= %= ^= &= |= <<= >>= >>>=	从右向左

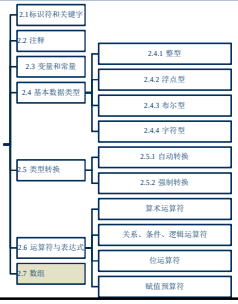
优先级总结

1. 括号( )的优先级最高
2. 优先级相同的情况下要考虑结合性。
   - 大多数运算符结合性为从左至右
   - 赋值运算符的结合性为从右至左

30

第二章 Java语言基础

基本程序设计



2.7 数组

数组是相同类型的数据元素按顺序组成的有序集，元素类型即可以是基本数据类型，也可以是对象类型。元素在数组中的相对位置由下标来指定。

- **声明数组：**
  类型　　数组名[ ]；
  类型[ ]　　数组名；
  例如： int list[ ]; 或者 int[ ] list;
- **创建数组　new**
  数组名=new 类型[长度]
  例如：list=new int[3];
- **声明与创建合而为一：**
  int arr[]=new int[10];
  int[] arr=new int[10] ;

2.7 数组

初始化

（1）用new为数组分配内存空间后，系统将为每个数组元素赋默认初值，
整型数的初始值为0，
浮点型数为0.0，
字符型初值为 ‘\u0000’，
布尔型初值为false，
引用类型的初值为null。
（2）声明的同时为各元素指定初值，其语法为：
类型[] 数组名={初值1，初值2，…}
例如： int a[ ]={100,200,300,400,500}//表示长度为5的整型数组。

2.7 数组

数组的使用

（1）访问数组元素
通过下标定位数组中的任一元素，下标从0开始递增计数。其表示方式为：
数组名[下标]
例如： a[1]=200 //对数组下标进行越界检查，以保证安全性。
（2）访问数组整体
数组名是指向数组对象的引用，可以通过修改引用来指向不同的数组对象。
例如：int a[ ]={100,200,300,400,500};
a=new int[10] //指向新的数组对象
int b=a; // b和a指向同一个数组对象。
（3） 数组字段Length
例如： a.length// a.length=5

二维数组

二维数组可以视为是将一维数组作为其元素的一维数组。

- **1、二维数组的定义**

（1）、 int intm[][] (int[][] intm)=new int[2][3];

（2）、 int a[ ][ ] = { {1,4,5}, {2,3,6}};

二维数组

- （3）、从最高维开始，分别为每一维分配空间，如：

int b[ ][ ] = new int[2] [ ];
b[0] = new int[3]；b[1] = new int[5]；

b[0][0]	b[0][1]	b[0][2]		
b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[1][4]

在使用new分配内存时，多维数组至少要给出最高维的大小。如果出现“int a2[ ][ ]=new int[ ][ ]”，编译器提示“缺少数组维”的错误。

2. 数组的使用

（1）访问数组元素，其引用格式为：

数组名[下标1][下标2] // 每一维的下标取值都从0开始。

（2） 访问数组长度length

可以求二维数组的长度，也可以分别求每一行的长度。如：

例如：

```
int a[][]={{2},{1,5},{3,4}};
System.out.println(a.length);     //打印3
System.out.println(a[0].length);  //打印1
System.out.println(a[2].length);  //打印2
```