



# Java程序设计

---

## 第3章 运算符与流程控制



# 第3章 运算符与流程控制

---

3.1 运算符

3.2 if 语句

3.3 switch多分支语句

3.4 if语句与switch语句的区别

3.5 循环语句

3.6 跳转语句



## 3.1 运算符

---

Java语言中的运算符主要包括：

- 赋值运算符
- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符



## 3.1.1 赋值运算符

赋值运算符的符号为“=”，它的作用是将数据、变量、对象赋值给相应类型的变量，例如下面的代码：

```
int i = 75; // 将数据赋值给变量
```

```
long l = i; // 将变量赋值给变量
```

```
Object object = new Object(); // 创建对象
```

赋值运算符的运算顺序为从右到左。例如在下面的代码中，首先是计算表达式“9412 + 75”的和，然后将计算结果赋值给变量result：

```
int result = 9412 + 75;
```



# 赋值运算符

如果两个变量的值相同，也可以采用下面的方式完成赋值操作：

```
int x, y; // 声明两个int型变量  
x = y = 0; // 为两个变量同时赋值
```

## 3.1.2 算术运算符

算术运算符支持整数型数据和浮点型数据的运算，当整数型数据与浮点型数据之间进行算术运算时，Java会自动完成数据类型的转换，并且计算结果为浮点型。

运算符	功能	举例	运算结果	结果类型
+	加法运算	10 + 7.5	17.5	double
-	减法运算	10 - 7.5F	2.5F	float
*	乘法运算	3 * 7	21	int
/	除法运算	21 / 3L	7L	long
%	求余运算	10 % 3	1	int



# 算术运算符

---

在进行算术运算时，有两种情况需要考虑：

- 没有小数参与运算
- 有小数参与运算。



# 没有小数参与运算

在对整数型数据或变量进行加法 (+)、减法 (-) 和乘法 (\*) 运算时, 与数学中的运算方式完全相同, 但是在整数之间进行除法 (/) 和求余 (%) 运算时需要注意几个问题。

- 注意除法运算
- 注意求余运算
- 关于0的问题





# 除法运算

在整数类型的数据和变量之间进行除法运算时，无论能否整除，运算结果都将是一个整数，而且这个整数不是通过四舍五入得到的，而是简单地去掉小数部分。

例如通过下面的代码分别计算10除以3和5除以2，最终输出的运算结果依次为3和2：

```
System.out.println(10 / 3); // 输出运算结果为3  
System.out.println(5 / 2); // 输出运算结果为2
```

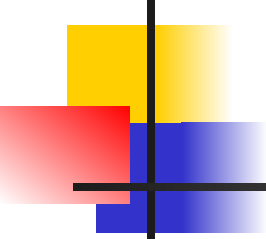


# 求余运算

在整数类型的数据和变量之间进行求余 (%) 运算时, 运算结果是数学运算中余数。

例如通过下面的代码分别计算  $10 \% 3$ 、 $10 \% 5$  和  $10 \% 7$ , 最终输出的运算结果依次为 1、0 和 3:

```
System.out.println(10 % 3); // 输出运算结果为1  
System.out.println(10 % 5); // 输出运算结果为0  
System.out.println(10 % 7); // 输出运算结果为3
```



# 关于0的问题

与数学运算一样，0可以做被除数，但是不可以做除数。当0做被除数时，无论是除法运算，还是求余运算，运算结果都为0。

例如通过下面的代码分别计算0除以6和0除以6求余数，最终输出的运算结果均为0：

```
System.out.println(0 / 6); // 输出运算结果为0  
System.out.println(0 % 6); // 输出运算结果为0
```

**注意：**如果0做除数，虽然可以编译成功，但是在运行时会抛出 `java.lang.ArithmeticException` 异常，即算术运算异常。



# 有小数参与运算

在对浮点数类型的数据或变量进行算术运算时，如果在算术表达式中含有double类型的数据或变量，则运算结果为double型，否则运算结果为float型。

在对浮点数类型数据或变量进行算术运算时，计算机的计算结果可能会在小数点后包含n位小数，这些小数在有些时候并不是精确的，计算机的计算结果会与数学运算的结果存在一定的误差，只能是尽量接近数学运算中的结果。



# 有小数参与运算

如果被除数为浮点型数据或变量，无论是除法运算，还是求余运算，0都可以做除数。如果是除法运算，当被除数是正数时，运算结果为Infinity，表示无穷大，当被除数是负数时，运算结果为-Infinity，表示无穷小；如果是求余运算，运算结果为NaN，表示非数字。

例如下面的代码：

```
System.out.println(7.5 / 0); // 输出的运算结果为Infinity
```

```
System.out.println(-7.5 / 0); // 输出的运算结果为-Infinity
```

```
System.out.println(7.5 % 0); // 输出的运算结果为NaN
```

```
System.out.println(-7.5 % 0); // 输出的运算结果为NaN
```



## 3.1.3 关系运算符

关系运算符用于比较大小，运算结果为boolean型，当关系表达式成立时，运算结果为true，否则运算结果为false。

运算符	功能	举例	结果	可运算数据类型
>	大于	'a' > 'b'	false	整数、浮点数、字符
<	小于	2 < 3.0	true	整数、浮点数、字符
==	等于	'X' == 88	true	所有数据类型
!=	不等于	true != true	false	所有数据类型
>=	大于或等于	6.6 >= 8.8	false	整数、浮点数、字符
<=	小于或等于	'M' <= 88	true	整数、浮点数、字符

要注意关系运算符“==”和赋值运算符“=”的区别!





## 3.1.4 逻辑运算符

逻辑运算符用于对boolean型数据进行运算，运算结果仍为boolean型。Java中的逻辑运算符包括：

- ! (取反)
- ^ (异或)
- & (与)
- | (或)
- && (简洁与)
- || (简洁或)

下面将依次介绍各个运算符的用法和特点。



# 取反运算符 “!”

运算符 “!” 用于对逻辑值进行取反运算，当逻辑值为true时，经过取反运算后运算结果为false，否则当逻辑值为false时，经过取反运算后运算结果则为true，

例如下面的代码：

```
System.out.println(!true); // 输出结果为false  
System.out.println(!false); // 输出结果为true
```





## 异或运算符 “^”

运算符 “^” 用于对逻辑值进行异或运算，当运算符的两侧同时为true或false时，运算结果为false，否则运算结果为true。

例如下面的代码：

```
System.out.println(true ^ true); // 输出的运算结果为false  
System.out.println(true ^ false); // 输出的运算结果为true  
System.out.println(false ^ true); // 输出的运算结果为true  
System.out.println(false ^ false); // 输出的运算结果为false
```



## 运算符 “&&” 和 “&”

运算符 “&&” 和 “&” 均用于逻辑与运算，当运算符的两侧同时为true时，运算结果为true，否则运算结果均为false。

例如下面的代码：

```
System.out.println(true & true); // 输出结果为true
System.out.println(true & false); // 输出结果为false
System.out.println(false & true); // 输出结果为false
System.out.println(false & false); // 输出结果为false
System.out.println(true && true); // 输出结果为true
System.out.println(true && false); // 输出结果为false
System.out.println(false && true); // 输出结果为false
System.out.println(false && false); // 输出结果为false
```





# 运算符“&&”和“&”的区别

运算符“&&”为简洁与运算符，运算符“&”为非简洁与运算符，它们的区别如下：

- 运算符“&&”只有在其左侧为true时，才运算其右侧的逻辑表达式，否则直接返回运算结果false。
- 运算符“&”无论其左侧为true或false，都要运算其右侧的逻辑表达式，最后才返回运算结果。



# 运算符 “||” 和 “|”

运算符 “||” 和 “|” 均用于逻辑或运算，当运算符的两侧同时为 false 时，运算结果为 false，否则运算结果均为 true，例如下面的代码：

```
System.out.println(true | true) ; // 输出的运算结果为 true
System.out.println(true | false) ; // 输出的运算结果为 true
System.out.println(false | true) ; // 输出的运算结果为 true
System.out.println(false | false) ; // 输出的运算结果为 false
System.out.println(true || true) ; // 输出的运算结果为 true
System.out.println(true || false) ; // 输出的运算结果为 true
System.out.println(false || true) ; // 输出的运算结果为 true
System.out.println(false || false) ; // 输出的运算结果为 false
```





# 运算符“||”和“|”的区别

---

运算符“||”为简洁或运算符，运算符“|”为非简洁或运算符，它们的区别如下：

- 运算符“||”只有在其左侧为false时，才运算其右侧的逻辑表达式，否则直接返回运算结果true。
- 运算符“|”无论其左侧为true或false，都要运算其右侧的逻辑表达式，最后才返回运算结果。





## 3.1.5 位运算符

---

位运算是对操作数以二进制位为单位进行的操作和运算，运算结果均为整数型。

位运算符又分为逻辑位运算符和移位运算符两种。



# 逻辑位运算符

---

逻辑位运算符包括：

- “~” （按位取反）
- “&” （按位与）
- “|” （按位或）
- “^” （按位异或）

它们用来对操作数进行按位运算，运算规则如下表所示。

# 逻辑位运算符

操作数x	操作数y	$\sim x$	$x \& y$	$x   y$	$x \wedge y$
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

按位取反运算是将二进制位中的0修改为1，1修改为0；在进行按位与运算时，只有当两个二进制位都为1时，结果才为1；在进行按位或运算时，只要有一个二进制位为1，结果就为1；在进行按位异或运算时，当两个二进制位同时为0或1时，结果为0，否则结果为1。







# 移位运算符

---

移位运算符包括：

- “<<” （左移，低位添0补齐）
- “>>” （右移，高位添符号位）
- “>>>” （无符号右移，高位添0补齐）

它们用来对操作数进行移位运算。

## 3.1.6 对象运算符 (instanceof)

对象运算符用来判断对象是否为某一类型，运算结果为boolean型，如果是则返回true，否则返回false，对象运算符的关键字为“instanceof”，它的用法为：

对象标识符 instanceof 类型标识符

例如：

```
java.util.Date date = new java.util.Date();  
System.out.println(date instanceof java.util.Date);           // 结果为true  
System.out.println(date instanceof java.sql.Date);             // 结果为false
```

## 3.1.7 其他运算符

Java中除了前面介绍的几类运算符外，还有一些不属于上述类别的运算符，如下表所示。

运算符	说 明	运算结果类型
++	一元运算符，自动递增	与操作元的类型相同
--	一元运算符，自动递减	与操作元的类型相同
?:	三元运算符，根据“?”左侧的逻辑值，决定返回“:”两侧中的一个值，类似if…else流程控制语句	与返回值的类型相同
[]	用于声明、建立或访问数组的元素	数组类型
.	访问类的成员或对象的实例成员	若访问的是成员变量，则类型与该成员变量相同；若访问的是方法，则类型与该方法的返回值相同





# 自动递增、递减运算符

与C、C++类似，Java语言也提供了自动递增与递减运算符，其作用是自动将变量值加1或减1。

它们既可以放在操作元的前面，也可以放在操作元的后面，根据运算符位置的不同，最终得到的结果也是不同的。

放在操作元前面的自动递增、递减运算符，会先将变量的值加1，然后再使该变量参与表达式的运算



# 自动递增、递减运算符

放在操作元后面的递增、递减运算符，会先使变量参与表达式的运算，然后再将该变量加1。

例如：

```
int num1=3;
```

```
int num2=3;
```

```
int a=2+(++num1);    //先将变量num1加1，然后再执行“2+4”
```

```
int b=2+(num2++);    //先执行“2+3”，然后再将变量num2加1
```



# 三元运算符 “?:”

三元运算符 “?:” 的应用形式如下:

逻辑表达式 ? 表达式1 : 表达式2

三元运算符 “?:” 的运算规则为:

若逻辑表达式的值为true, 则整个表达式的值为表达式1的值, 否则为表达式2的值。

例如:

```
int store=12;
```

```
System.out.println(store<=5?"库存不足! ":"库存量: "+store);
```

这段代码的输出结果为 “库存量: 12”



## 3.1.8 运算符的优先级别 及结合性

当在一个表达式中存在多个运算符进行混合运算时，会根据运算符的优先级别来决定运算顺序，优先级最高的是括号“()”，它的使用与数学运算中的括号一样，只是用来指定括号内的表达式要优先处理。

例如：

```
int num=8*(4+6); // num为80
```

## 3.1.8 运算符的优先级别 及结合性

对于处在同一层级的运算符，则按照它们的结合性，即“先左后右”还是“先右后左”的顺序来执行。

Java中除赋值运算符的结合性为“先右后左”外，其他所有运算符的结合性都是“先左后右”。

关于运算符优先级的顺序，如下表所示。



优先级	说明	运算符											
最高	括号	()											
	正负号	+			-								
	一元运算符	++			--			!			~		
	乘除运算	*			/			%					
	加减运算	+			-								
	移位运算	<<			>>			>>>					
	比较大小	<			>			<=			>=		
	比较是否相等	==			!=								
	按位与运算	&											
	按位异或运算	^											
	按位或运算												
	逻辑与运算	&&											
	逻辑或运算												
	三元运算符	?:											
最低	赋值及复合赋值	=	*=	/=	%=	+=	-=	>>=	>>>=	<<<=	&=	^=	=





## 3.2 if 语句

---

if语句可分为以下3种形式:

- (1) 简单的if条件语句
- (2) if-else条件语句
- (3) if-else if多分支条件语句



## 3.2.1 简单的if条件语句

简单的if条件语句就是对某种条件做出相应的处理。通常表现为“如果满足某种情况，那么就进行某种处理”。它的一般形式为：

```
if(表达式) {  
    语句序列  
}
```

例如：如果今天下雨，我们就不出去玩。

条件语句为：

```
if(今天下雨) {  
    我们就不出去玩  
}
```



# 简单的if条件语句

表达式是必要参数。其值可以由多个表达式组成，但是其最后结果一定是boolean类型，也就是其结果只能是true或false。

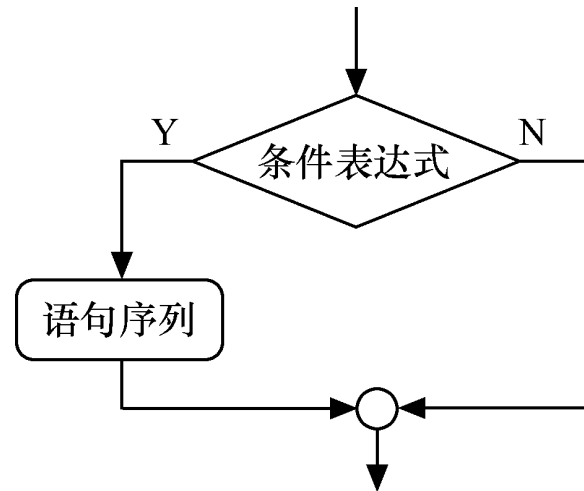
语句序列是可选参数。包含一条或多条语句，当表达式的值为true时执行这些语句。如果该语句只有一条语句，大括号也可以省略不写。下面的代码都是正确的。

```
if(今天下雨);
```

```
if(今天下雨)
```

```
    我们就不出去玩;
```

# 简单的if条件语句



如图所示，if条件语句，在条件表达式的结果为true时，将执行语句序列。



## 3.2.2 if...else条件语句

if...else条件语句也是条件语句的一种最常用的形式。else是可选的。通常表现为“如果满足某种条件，就做某种处理，否则做另一种处理”。它的一般形式为：

```
if(表达式) {  
    语句序列1  
}  
else {  
    语句序列2  
}
```



# if...else 条件语句

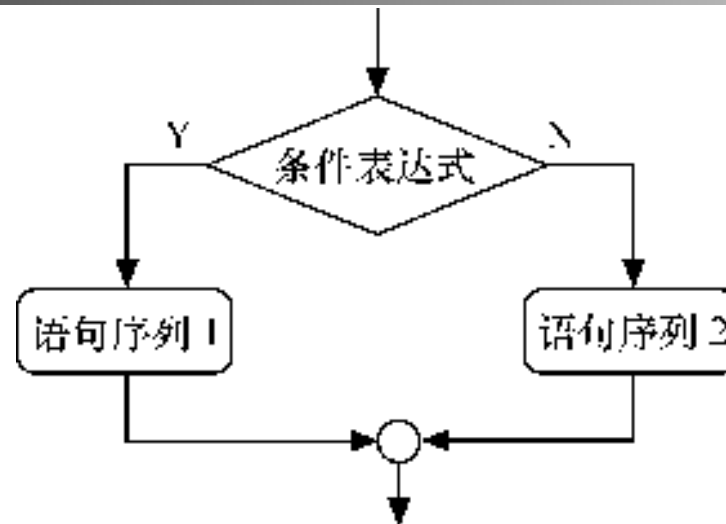
语句序列1是可选参数。由一条或多条语句组成，当表达式的值为true时执行这些语句。

语句序列2也是可选参数。包含一条或多条语句，当表达式的值为false时执行这些语句。例如：如果指定年为闰年，二月份为29天，否则二月份为28天。条件语句为：

```
if(今年是闰年){  
    二月份为29天  
}  
else{  
    二月份为28天  
}
```



# if...else 条件语句



如图所示，if...else语句在表达式的值为true时，执行语句序列1，否则，执行语句序列2。



### 3.2.3 if...else if多分支语句

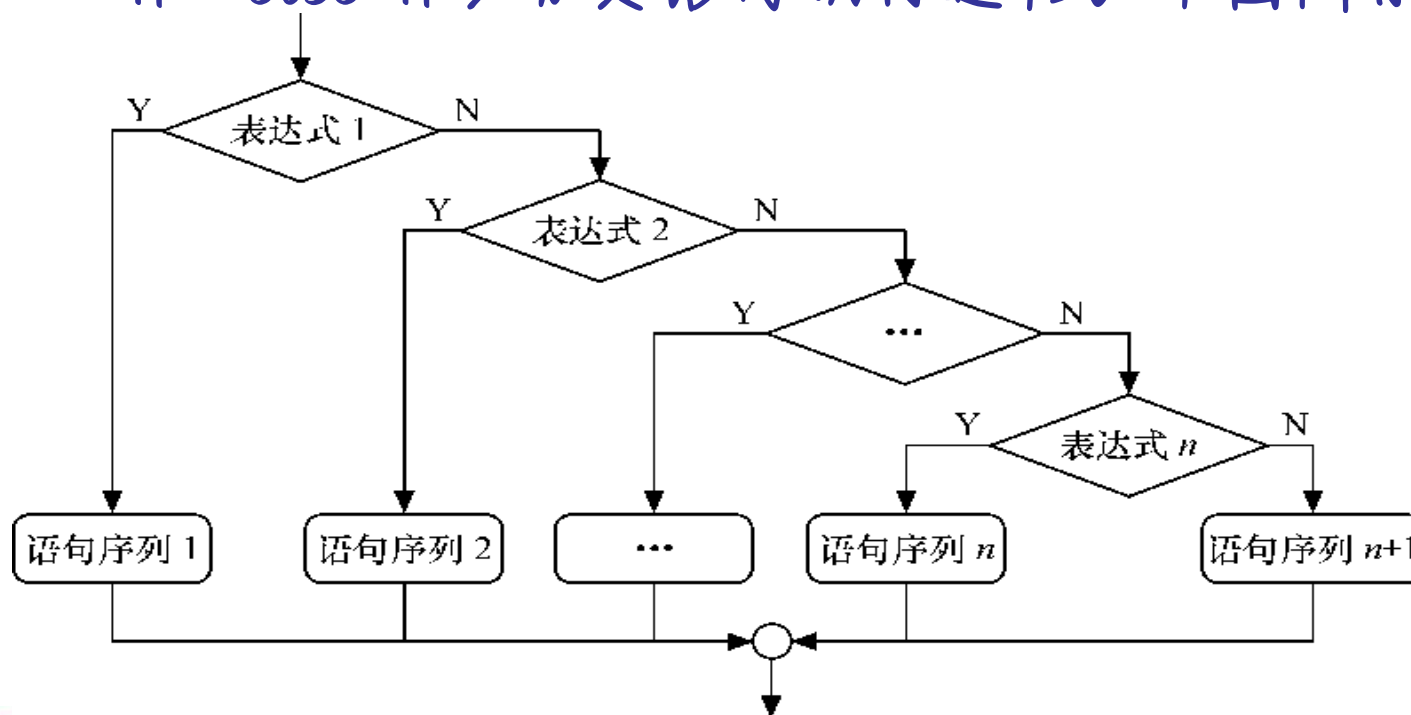
if...else if多分支语句用于针对某一事件的多种情况进行处理。通常表现为“如果满足某种条件，就进行某种处理，否则如果满足另一种条件才执行另一种处理”。它的一般形式为：

```
if(表达式1) {  
    语句序列1  
}else if(表达式2) {  
    语句序列2  
}else {  
    语句序列n  
}
```

# if...else if 多分支语句

语句序列1在表达式1的值为true时被执行，语句序列2在表达式2的值为true时被执行，语句序列n在表达式1的值为false，表达式2的值也为false时被执行。

if...else if 多分支语句执行过程如下图所示。





# if...else if 多分支语句

例如：如果今天是星期一，上数学课；如果今天是星期二，上语文课；否则上自习。

条件语句为：

```
if(今天是星期一){  
    上数学课  
}else if(今天是星期二){  
    上语文课  
}else{  
    上自习  
}
```

## 3.2.4 if语句的嵌套

if语句的嵌套就是在if语句中又包含一个或多个if语句。这样的语句一般都用在比较复杂的分支语句中。它的一般形式为右侧的语句格式。

在嵌套的语句中最好不要省略大括号。以提高代码的可读性。

```
if(表达式1) {  
    if(表达式2) {  
        语句序列1  
    }else{  
        语句序列2  
    }  
}else{  
    if(表达式3) {  
        语句序列3  
    }else{  
        语句序列4  
    }  
}
```





## 3.3 switch多分支语句

switch语句是多分支的开关语句。根据表达式的值来执行输出的语句。这样的语句一般用于多条件多值的分支语句中。右侧是它的语法格式：

break用于结束switch语句。

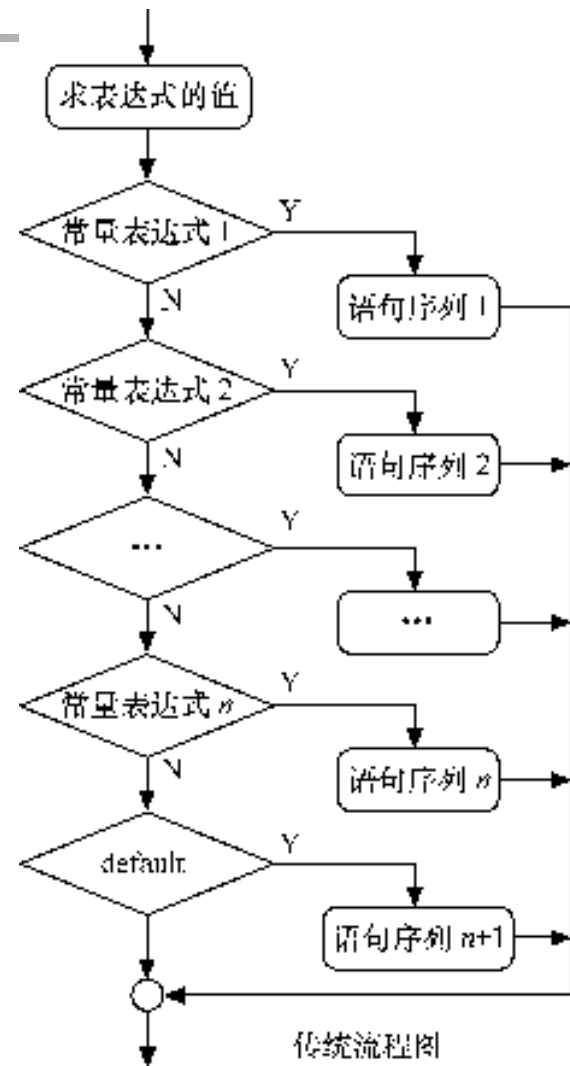
```
switch(表达式) {  
    case 常量表达式1: 语句序列1  
        [break;]  
    case 常量表达式2: 语句序列2  
        [break;]  
    ... ...  
    case 常量表达式n: 语句序列n  
        [break;]  
    default: 语句序列n+1  
        [break;]  
}
```

# switch多分支语句

switch语句中表达式的值必须是整型或字符型。即int、short、byte和char型。Switch会根据表达式的值，执行符合常量表达式的语句序列。

当表达式的值没有匹配的常量表达式时，则执行default定义的语句序列，即“语句序列n+1”。

default是可选参数，如果没有该参数，并且所有常量值与表达式的值不匹配，那么switch语句就不会进行任何操作。



该语句执行流程如右图所示





## 3.4 if语句和switch语句的区别

if语句和switch语句可以从使用的效率上来进行区别，也可以从实用性角度去区分。

如果从使用的效率上进行区分，在对同一个变量的不同值作条件判断时，使用switch语句的效率相对更高一些，尤其是判断的分支越多越明显。

如果从语句的实用性的角度去区分，那switch语句肯定不如if语句。if语句是应用最广泛和最实用的语句。



## 3.5 循环语句

循环语句就是重复执行某段程序代码，直到满足特定条件为止。在Java语言中循环语句有以下3种形式：

- for循环语句
- while循环语句
- do-while循环语句





## 3.5.1 for循环语句

for语句是最常用的循环语句，一般用在循环次数已知的情况下。它的一般形式为：

```
for (初始化语句;循环条件;迭代语句) {  
    语句序列  
}
```

初始化语句用于初始化循环体变量。

循环条件用于判断是否继续执行循环体。其只能是true或false。

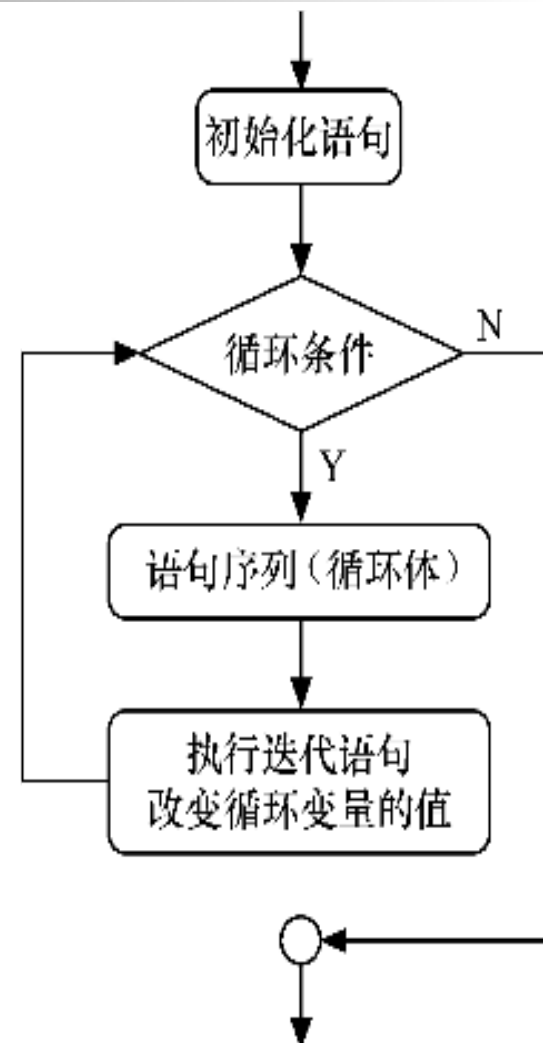
迭代语句用于改变循环条件的语句。

语句序列称为循环体，当循环条件的结果为true时，将重复执行。

# for 循环语句

for 循环语句的流程首先执行初始化语句，然后判断循环条件，当循环条件为 true 时，就执行一次循环体，最后执行迭代语句，改变循环变量的值。这样就结束了一轮的循环。接下来进行下一次循环（不包括初始化语句），直到循环条件的值为 false 时，才结束循环。

for 循环语句执行过程如图所示。



## 3.5.2 while循环语句

while语句是用一个表达式来控制循环的语句。

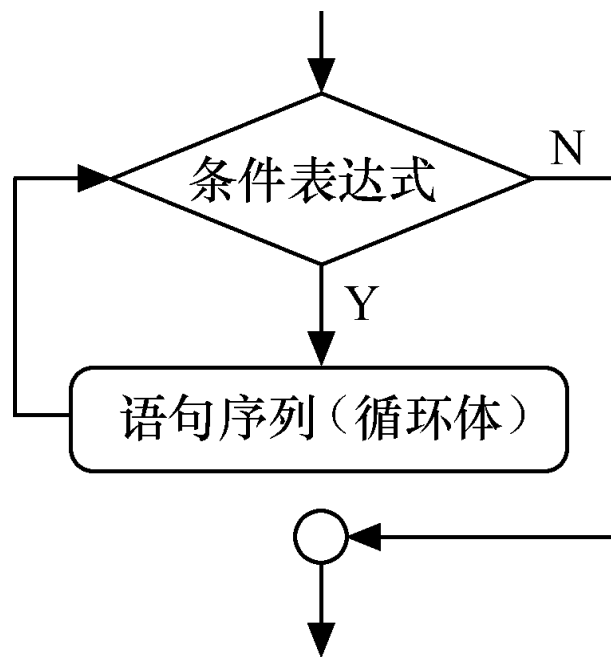
它的一般形式为：

```
while(表达式) {  
    语句序列  
}
```

表达式用于判断是否执行循环，它的值只能是true或false。当循环开始时，首先会执行表达式，如果表达式的值为true，则会执行语句序列，也就是循环体。当到达循环体的末尾时，会再次检测表达式，直到表达式的值为false，结束循环。

# while循环语句

while语句执行过程如图所示。





### 3.5.3 do...while循环语句

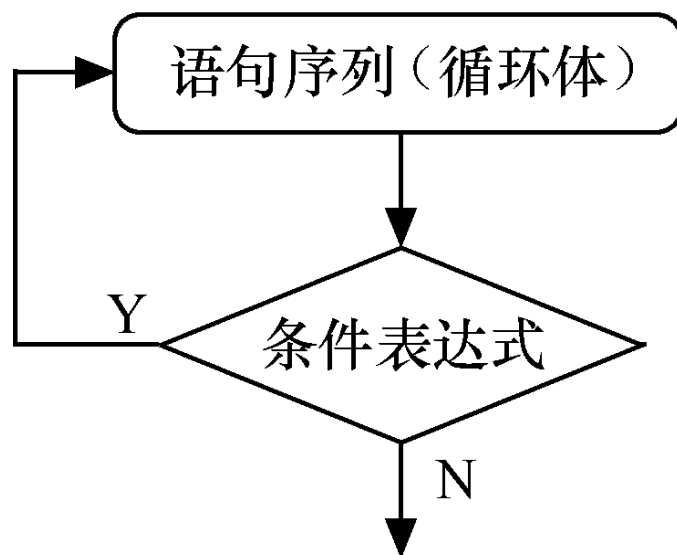
do..while循环语句称为后测试循环语句，它利用一个条件来控制是否要继续重复执行这个语句。它的一般形式为：

```
do{  
    语句序列  
}while(表达式);
```

do…while循环语句的执行过程与while循环语句有所区别。do…while循环至少被执行一次，它先执行循环体的语句序列，然后再判断是否继续执行。

# do...while循环语句

do...while循环执行语句如图所示。





## 3.5.4 循环的嵌套

循环的嵌套就是在一个循环体内又包含另一个完整的循环结构，而在这个完整的循环体内还可以嵌套其他的循环结构。循环嵌套很复杂，在for语句、while语句和do...while语句中都可以嵌套。常用的嵌套循环包括：

- for循环语句的嵌套
- while循环语句嵌套
- do...while循环语句嵌套
- for循环语句与while循环语句嵌套
- while循环语句与for循环语句嵌套
- do...while循环语句与for循环语句嵌套



## 3.6 跳转语句

---

Java语言中支持的跳转语句包括：

- break跳转语句
- continue跳转语句
- return跳转语句。





## 3.6.1 break跳转语句

---

break语句可以终止循环或其他控制结构。它在for，while或do…while循环中，用于强行终止循环。

只要执行到break语句，就会终止循环体的执行。break不仅在循环语句里适用，在switch多分支语句里也适用。



## 3.6.2 continue跳转语句

continue语句应用在for, while和do...while等循环语句中, 如果在某次循环体的执行中执行了continue语句, 那么本次循环就结束, 即不再执行本次循环中continue语句后面的语句, 而进行下一次循环。



### 3.6.3 return 跳转语句

return 语句可以从一个方法返回，并把控制权交给调用它的语句。return 语句通常被放在方法的最后，用于退出当前方法并返回一个值。它的语法格式为：

```
return [表达式];
```

表达式是可选参数，表示要返回的值。它的数据类型必须同方法声明中的返回值类型一致。

例如：编写返回a和b两数相加之和的方法可以使用如下代码：

```
public int set(int a,int b) {  
    return sum=a+b;  
}
```

如果方法没有返回值，可以省略return关键字的表达式，使方法结束。代码如下：

```
public void set(int a,int b) {  
    sum=a+b;  
    return;  
}
```