# 构成Java的语法---语句

类是构成一个Java应用程序的基本元素,而语句则是构成一个类的基本元素,利用语句我们可以达到定义数据和操作数据,所以要想完美的自定义一个类,那么语句是我们必不可少的一个学习点。

Java中的语句分为声明和赋值语句，条件和循环语句，调用和返回语句；我们之所以每两个放在一起是有深意的，我们大致将语句分为这三块，并以此为纲来说一下，重点在流程控制语句（条件和循环语句）。

Java中的一个类就是由一系列的声明、赋值、条件、循环、调用和返回语句组成的。一般来说代码都是可以嵌套的：一个条件或循环语句中也可以嵌套条件语句或循环语句。

## 1 声明与赋值语句

声明语句将一个变量名和一个类型在编译时关联起来。Java需要我们用声明语句指定变量的名称和类型。这样我们才能清楚的指明对其的操作。***Java是一种强类型语音，因为Java编译器会检查类型的一致性***（例如，它不会允许boolean类型和浮点类型的变量相乘）。变量可以声明在第一次使用之前的任何地方，一般我们会在首次使用时声明它，变量的作用域就是定义它的地方（成员或局部，即其最近的一个大括号内），一般由相同代码段中声明之后的所有语句组成。

赋值语句（由一个表达式定义的）某个数据类型的值（可以是具体值，也可以是一个引用）和一个变量关联起来。在Java中，我们进行赋值操作，我们表达的不是数学中的等于“=”，而是将一个地址与该变量名绑定，让该变量名指向这个内存地址。

|  |
| --- |
| **public** **class** Test {  //声明语句由数据类型与变量名称构成，--这里的数据类型可以是基本数据类型，也可以是引用类型--对象  **static** **int** *i*;  /\*int ii;  ii = 89;\*///1.8不允许在局部中声明完后再赋值，要在一起操作  **public** **static** **void** main(String[] args) {  *i* = 23;  System.*out*.println(*i*);  }  } |



我们在开发中往往建议将声明和赋值放在一起操作，即声明的同时给语句赋值。

### 注意事项

* JDK1.8不允许在全局（方法外）中声明完后再赋值，要在一起操作；如int i;i = 0;不能编译通过。但是在局部（方法中）中是完全可以的，也不存在编译问题。
* 在全局中允许只声明变量而不用赋值，但是局部必须声明且初始化。这是由于Java的类加载机制决定的，在类加载后会先将成员变量初始化，然后再赋其值，但是局部变量不存在这种，所以必须初始化。
* Final修饰的成员变量必须声明且初始化。
* 在方法中声明的局部变量要注意声明周期，如下面声明是不允许的，这是由于第一个变量声明周期尚未结束，这时第二个变量声明的名字和前者相同，所以编译器提示duplicate local variable“重复的局部变量”
* 声明的局部变量如果和成员变量名称相同，那么遵循就近原则。

|  |  |
| --- | --- |
| **final** **int** i = 0;  {  **int** i = 0;  } |  |

## 2 调用与返回语句

我们对Java最大的印象只怕就在面向对象了，我们写的所有类的最大作用就是封装了方法和状态，使得有需要的类可以复用封装的方法。这里的复用是通过实例调用方法，例如我们写一个加法的运算，那么以后凡是遇到加法的运算，那么就都可以通过调用这个方法来解决了，这就是调用的思想，而有些方法是具有返回值的，返回值的语句是返回语句；举例如下：

|  |
| --- |
| **package** cn.assign.statement;  **public** **class** Test {  //声明语句由数据类型与变量名称构成，--这里的数据类型可以是基本数据类型，也可以是引用类型--对象  **static** **int** *i*;  /\*int ii;  ii = 89;\*///1.8不允许在局部中声明完后再赋值，要在一起操作  **public** **static** **void** main(String[] args) {  *i* = 23;  System.*out*.println(*i*);  *add*(7, 8);//标准调用语句  **int** in = *add*(23, 23);//声明+赋值+调用  System.*out*.println(in);  }  //简单的加法运算  **public** **static** **int** add(**int** a,**int** b){  **return** a+b;//返回语句  }  } |

通过上面我们发现语句并不固定于单类型语句的使用，而是花样百出的嵌套，组合使用，这也是高级编程语言的灵活所在。在返回语句中我们发现有一个关键字return，这里我们不多做介绍，我们放在流程语句和continue和break一起介绍。

## 3 流程控制

做任何事情都要遵循一定的原则。例如，到图书馆去借书，就必须有借书证，并且借书证不能过期，这两个条件缺一不可。程序设计也是如此，需要有流程控制语言来实现与用户的交流，并根据用户的输入决定程序要“做什么”、“怎么做”等。

流程控制语言对于任何一门编程语言来说都是至关重要的，它提供了控制程序步骤的基本手段。如果没有流程控制语句，整个程序将按照线性的顺序来执行，不能根据用户的输入来决定执行的序列。我们大略的将流程控制语句分为以下这么几类，复合语句、条件语句、选择语句、循环语句，下面我们将一一介绍。

### 3.1 复合语句

与C语言及其它语言相同，Java语言的复合语句是以整个区块为单位的语句，所以又称块语句。复合语句由开括号“{”开始，和闭括号“}”结束。

在前面的学习中已经解触到了这种复合语句。例如在定义一个类或方法时，类体就是以“{}”作为开始与结束的标记，方法体同样也是以“{}”作为标记。对于复合语句中的每个语句都是从上到下的执行。***复合语句以整个块为单位，可以用在任何一个单独语句可以用到的地方，并且在复合语句中还可以嵌套复合语句（代码块就是一种典型的复合语句）***。

在使用复合语句时要注意，复合语句为局部变量创建一个作用域，该作用域为程序的一部分，在该作用域中某个变量被创建并能够被使用，如果某个变量的作用域外使用该变量，则会发生错误。还需要注意的是，不能在嵌套的两个块中声明同名的变量（并不绝对，只要控制好变量的声明周期就可以了，注意这里是生命周期，不是作用域），如下就是完全可以的编译可以通过不会报错的。

{

{

**int** i = 0;

}

**int** i = 0;

}

【例5.1】创建CompoundStatementPractice类，进行复合语句的一些验证；

class CompoundStatementPractice{

public static void main(String[] args){

int a = 23 ;

System.out.println("a的值是:"+a);

{

int b = 24 , c = 25 ;

System.out.println("b的值是:"+b);

boolean boo ; //我们一定要注意变量的作用域,也就是它的生命周期

{

boo = b > c ;

System.out.println("复合语句中的嵌套语句:"+boo);

}

// int a = 20 ; 如果在main方法中已经声明成员变量,复合语句中再次声明相同名字的成员变量会出现错误

a = a++ + 1 ;

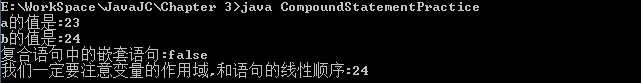
}

//System.out.println("在变量的作用域外使用变量会出现错误:"+b); 会提示找不到符号

System.out.println("我们一定要注意变量的作用域,和语句的线性顺序:"+a);

}

}



#### 3.1.1 代码块

在编程过程中我们通常会遇到如下这种形式的程序：

|  |
| --- |
| **package** democlass;  **public** **class** CodeBlock {  {  System.*out*.println("构造代码块1！");  }  **static**{  System.*out*.println("直接由static关键字进行声明的代码块就是静态代码块！");  }  //定义构造函数  **public** CodeBlock(){  System.*out*.println("只要一有实例产生，构造代码块就运行，构造代码块优先于构造函数！");  }  **static**{  System.*out*.println("静态代码块优先于构造函数执行，且不管实例化多少次对象，只执行一次！");  }  {  System.*out*.println("构造代码块2！");  }  **public** **static** **void** main(String[] args) {  {  **int** a = 6;//a属于局部变量  System.*out*.println("普通代码块："+a);  }  //System.out.println("普通代码块中的变量为局部变量，出了{}生命周期结束"+a);  **int** a = 7;  System.*out*.println("代码块之外再次声明一个与代码块中变量名称相同的变量不冲突，因为代码块决定了生命周期，如同成员变量和局部变量"+a);    **new** CodeBlock();  **new** CodeBlock();  }  } |

如上阴影部分的程序段我们称之为代码块，所谓代码块就是用{}将多行代码封装起来，形成一个独立的数据体，用于实现特定的算法。***一般来说代码块是不能单独运行的，他必须要有运行主体***，在Java中代码块有四种形式；普通代码块、构造代码块、静态代码块、同步代码块。代码块没有名称，设置代码块的目的是为了初始化类,前面我们也说过可以将代码块认为是特殊的复合语句，下面我们来一一分析一下这四种代码块;

##### 3.1.1.1 普通代码块

直接定义在方法中的代码块我们称之为普通代码块；

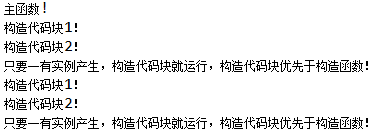
|  |
| --- |
| **public** **static** **void** main(String[] args) {  {  **int** a = 6;//a属于局部变量  System.*out*.println("普通代码块："+a);  }  //System.out.println("普通代码块中的变量为局部变量，出了{}生命周期结束"+a);  } |

  普通代码块运行顺序，按照正常的代码顺序，从上至下依次运行，在这里我们要注意一个问题，如果我们声明了一个局部变量，然后在声明的局部变量后面创建代码块，则代码块中局部变量名不能和上面局部变量名相同；这是因为方法中局部变量生命周期没有消失，再次声明会出现变量名相同的情况，和先声明代码块，再声明局部变量不同；普通代码块其实就是最普通的复合语句。

##### 3.1.1.2 构造代码块

　　将代码块直接定义在类中，则称之为构造代码块。构造代码块优先于构造方法执行，且执行多次，***只要一有实例化对象产生，就执行构造代码块中内容；***

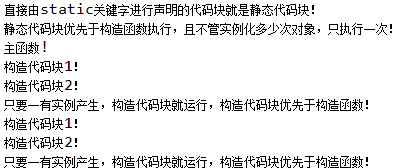
|  |
| --- |
| **package** democlass;  **public** **class** CodeBlock {  {  System.*out*.println("构造代码块1！");  }  //定义构造函数  **public** CodeBlock(){  System.*out*.println("只要一有实例产生，构造代码块就运行，构造代码块优先于构造函数！");  }  {  System.*out*.println("构造代码块2！");  }  **public** **static** **void** main(String[] args) {  System.*out*.println("主函数!");  **new** CodeBlock();  **new** CodeBlock();  }  } |



##### 3.1.1.3 静态代码块

直接使用static关键字进行声明的就是静态代码块。静态块在类加载的时候执行,所以它优先于构造函数,优先于main函数，且不管多少个实例化对象产生，静态代码块只执行一次，***静态代码块的主要功能就是为静态发生初始化。在执行类时，希望先执行类的初始化动作，可以使用static定义一个静态代码块。***

|  |
| --- |
| **package** democlass;  **public** **class** CodeBlock {  {  System.*out*.println("构造代码块1！");  }  **static**{  System.*out*.println("直接由static关键字进行声明的代码块就是静态代码块！");  }  //定义构造函数  **public** CodeBlock(){  System.*out*.println("只要一有实例产生，构造代码块就运行，构造代码块优先于构造函数！");  }  **static**{  System.*out*.println("静态代码块优先于构造函数执行，且不管实例化多少次对象，只执行一次！");  }  {  System.*out*.println("构造代码块2！");  }  **public** **static** **void** main(String[] args) {  System.*out*.println("主函数!");  **new** CodeBlock();  **new** CodeBlock();  }  } |



静态代码块主要用于类的初始化。它只执行一次，并在构造函数之前执行。静态代码块的特点主要有：

* 静态代码块会在类被加载时自动执行。
* 静态代码块只能定义在类里面，不能定义在方法里面。
* 静态代码块里的变量都是局部变量，只在块内有效。
* 一个类中可以定义多个静态代码块，按顺序执行。
* 静态代码块只能访问类的静态成员，而不允许访问实例成员，因为静态代码块运行时此时实例还尚未产生。

##### 3.1.1.4 同步代码块

使用 synchronized 关键字修饰，并使用“{}”括起来的代码片段，它表示同一时间只能有一个线程进入到该方法块中，是一种多线程保护机制。如下代码所示：

synchronized (objectA) {

System.out.println(threadName + "ÒÑ¾­½øÈëÍ¬²½´úÂë¿éobjectA");

}

 　　从上面我们可以看到普通代码块的作用是限定变量的生命周期，尽早释放，节约内存，当调用其所在的方法时执行；构造代码块的作用是把多个构造方法共同的部分提取出来，共用构造代码块，每次调用构造方法时，都会优先于构造方法执行，也就是每次new一个对象时自动调用，对对象进行初始化；静态代码块的作用是对类进行一些初始化  只加载一次，所有对象共享一份，类加载的时候加载执行，只执行一次；同步代码块的作用是达到线程安全的目的，线程调用该方法时调用；我们从上面可以看出代码块的执行顺序，类加载，静态代码块>构造代码块>构造函数。我们可以通过上面的代码进行验证。但总的来说代码块只有一个具有特定功能的语句集合，它在不同的时期运行，或初始化或同步的作用。

### 3.2 条件语句

条件语句可根据不同的条件执行不同的语句。条件语句包括if条件语句和switch多分支语句（也叫选择语句）。

#### 3.2.1 if条件语句

If条件语句是一个重要的编程语句，它用于告诉程序在某个条件成立的情况下执行某段程序，而在另一种情况下执行另外的语句。

使用if条件语句，可选择是否要执行紧跟在条件之后的那个语句。关键字if之后是作为条件的“布尔表达式”，如果该表达式返回的结果是true，则执行其后的语句；若为false，则不执行if条件之后的语句。If条件语句可分为简单的if条件语句、if...else语句和if...else if多分支语句。

##### 3.2.1.1 简单的if语句

语法格式如下：

If(布尔表达式){

语句序列

}

* 布尔表达式：必要参数，表示它最后返回的结果必须是一个布尔值。它可以是一个单纯的布尔变量或常量，或者使用关系或布尔运算符的表达式。布尔表达式必须用括号括起来。
* 语句序列：可选参数。可以是一条或多条语句（如果是多条语句那么就是块语句，实际上我们可以认为条件表达式后边就是跟了衣蛾块语句），当表达式为true时执行这些语句。如果语句序列中仅含有一条语句，则可以省略条件表达语句中的大括号。但是强烈不建议这么做，因为这会使代码不清晰，在以后的代码重构中也会因疏忽造成问题，再者编译器会进行优化，实际上即便我们在有一条语句的情况下加上大括号，编译器也会帮我们去除。

【例5.2】创建IfPractice类，进行简单if条件语句的练习

class IfPractice{

public static void main(String[] args){

int a = 23 , b = 24 ;

boolean boo = true ,aoo = false ;

if(! false == boo || (5 == 6)){

System.out.println("表达式的形式可以多样但其结果必须为boolean类型");

}

if(5 == 6);{

System.out.println("如果if条件语句后即()后直接跟;则该条if简单语句语句无论是否成立都继续下面代码");

}

if(a < b){

if( a < b){

System.out.println("If语句可以像复合语句一样嵌套语句!");

}

System.out.println("简单if语句的标准形态;");

}

if(aoo) System.out.println("简单if语句序列中若只有一条语句则可以省略{}这时并无语法错误,并且if语句的作用范围也可覆盖,但不支持,为了增强阅读性");

System.out.println("if语句若省略{},那么其作用范围只是紧随其后的一条语句,其他的语句会线性执行,不会受其表达式的影响!");

if(boo){}//当简单if语句序列中没有内容时可以省略{},末尾加;,当然也可以保留,

if(a < b && boo){

System.out.println("简单if语句中的条件一定要是布尔表达式,也就是结果必须是布尔型,格式不限制");

}

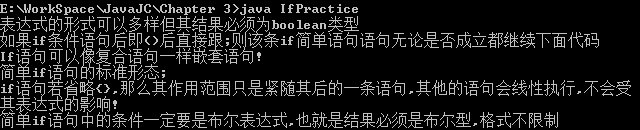
/\*if(3 = 3){

System.out.println("我们一定要确定if语句条件表达式的数据类型,如上3是int型,结果也是int型,会爆出错误以外的类型");

}\*/

}

}



|  |  |
| --- | --- |
| If语句注意点：  a,每一种格式都是单分支语句，即只会沿着一条分支运行。  b,第二种格式与三元运算符的区别：三元运算符运算完要有值出现。好处是：可以写在其他表达式中。  c,条件表达式无论写成什么样子，只看最终的结构是否是true 或者 false;  D.if语句同复合语句一样可以嵌套语句 | If语句流程运行图： |

##### 3.2.1.2 if...else语句

If...else语句是条件语句中最常用的一种形式,它会针对某种条件有选择地做出处理。通常表现为“如果满足某种条件，就进行某种处理，否则就进行另一种处理”。语法格式如下：

|  |
| --- |
| If(){  若干语句  }else{  若干语句  } |

If后面()内的表达式的值必须是boolean型的。如果表达式的值为true，则执行紧跟if语句的符合语句：如果表达式的值为false，则执行else后面的复合语句。If...else语句的执行流程如下图所示。

同简单的if条件语句一样，如果if...else语句的语句序列中只有一条语句（不包括注释），则可以省略该语句序列外面的大括号（不建议这么做）。有时为了编程的需要，else或if后面的大括号里可以没有语句。

|  |  |
| --- | --- |
| If....else语句和三元运算符的区别 | If ....else语句的执行行程图 |
| 对于if...else语句可以使用三元运算符对语句进行简化,如下面的代码 if(a > 0) b = a ; else b = -a;  可以简化成: b = a>0?a:-a ；**上面代码为求绝对值的语句**，如果a > 0，就把a的值赋给b,否则将-a赋值给b。也就是“？”前面的表达式为真，则将问号和冒号之间的表达式的计算结果赋值给变量b，否则将冒号后面的计算结果赋值给b，***使用三元运算符的好处是可以使代码简洁，并且有一个返回值***。 | 图3.4 if...else语句的执行过程图 |

在第一种简单if语句中条件不成立时if语句不能给出任何反馈,这时出现了第二种if语句，当条件表达式的结果为true则执行语句1,否则执行语句2.

**注意事项:**

* **else没有条件表达式**
* **else必须和if组合使用,else不能单独使用 会报错**
* **一个if语句只能有一个else，但可以有多个if else（条件表达式）{语句块}语句**
* **else是可以省略的,当知道条件表达式成立时,可以省略else,这时就相当于简单的if语句了**
* **if语句和else量和使用时,if如果控制多行语句就一定要加{},否则出现上面第二种情况的错误,这时情况为简单if语句+输出语句+else,,如果if控制一行else控制一行时大括号能省,这是如时条件表达式为true则进行紧挨if语句的输出,反之输出else,当if控制一行else控制多行时如果条件表达式为true则进行紧挨if语句的输出,else中输出除去紧挨else的语句 反之else中输出,所以Java中不管什么情况不要省略{},避免错误**

【例3.3】创建IfElsePractice类，并针对if....else的一些特性进行验证

class IfElsePractice{

public static void main(String[] args){

/\*else{

System.out.println("else必须和if语句连用,不能单独使用,否则会错误,有'if',但是没有'else'");

}\*/

if(5 > 6){

System.out.println("我们可以看到简单if语句条件不成立时,if语句不能出现任何反馈,那么我们怎么知道是条件不成立还是程序错误呢,这时就出现了if...else语句");

}

if(5 > 6){

System.out.println("为补充if语句而出现的if...else语句");

}else/\*(5 < 6)\*/{

System.out.println("if...else语句中else语句后不能有条件语句直接跟随{},如果跟随(),会发出错误,提示在()后加;");

}/\*else{

System.out.println("一个if语句只能和一个else语句连用,否则会错误,有'if',但是没有'else'");

}\*/

if(5 < 6){

System.out.println("当if后面只控制一条语句时,可以省略{},它会直接输出,否则不支持,因为它只会输出紧跟他的一条");

}else{

System.out.println("当else后面只控制一条语句时,可以省略{},它会直接输出,否则不支持,因为它只会输出紧跟他的一条");

}

/\*if(5 > 6)

System.out.println("{}省略"); 当if...else...语句中，if后边控制多条语句时，{}不可

以省略，否则会报错“有if没有else”

System.out.println("{}省略");

else{

System.out.println("{}省略时,");

}\*/

//嵌套语句在if...else语句中的应用

if(true){

if(false){

System.out.println("嵌套语句在if...else中的使用");

}else{

System.out.println("嵌套语句在if...else中的使用");

}

}else{

System.out.println("嵌套语句在if...else中的使用");

}

/\*三元运算符和if...else语句的区别,

**If--else语句与三元运算符的区别:三元运算符运算完要有值出现。好处是：可以写在其他表达式中,而if----else语句则没有这种限制,但是可以写在其他方法中,不能写在表达式中**\*/

//求 a的绝对值和b的和

int a = 7 , b = 3 , sum ;

if(a > 0){

System.out.println("您所求的两个数的和为:"+(a+b));

}else

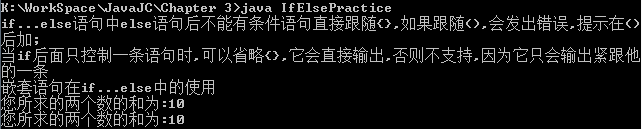
System.out.println("您所求的数的绝对值为:"+(-a+b));

sum = (a > 0 ? a : (-a))+b;

System.out.println("您所求的两个数的和为:"+sum);

}

}



面试题：问下面一段代码是否会正常输出?

if(5 > 6)

System.out.println("{}省略”);

System.out.println("{}省略");

else{

System.out.println("{}省略时,");

}

答:不会,代码会出现错误,当代码判断条件语句不成立时,会自动跳过紧跟他的一条语句,这是它的作用范围,然后编译下一条代码,这时因为没有发现else,会将他认为是简单if语句,再往下执行时发现了else语句,这是编译会将其作为不合法代码跳出”错误:有’if’没有’else’”

##### 3.2.1.3 if...else if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，如果满足另一种情况则执行另一种处理”。语法格式如下：

|  |  |  |
| --- | --- | --- |
| if（条件表达式1）{  语句序列1  }else if（条件表达式2）{  语句序列2  }  ....  else if(条件表达式n){  语句序列n  } | 条件表达式1~n:必要参数。可以由多个表达式组成，但最后返回的结果一定要是boolean类型；  语句序列：可以是一条或者多条语句，当条件表达式1的值为true时，执行语句序列1;当条件表达式2的值为true时,执行语句序列2,依次类推。当省略任意一组语句序列时，可以保留其外面的大括号,也可以将大括号替换为”;”  if...else if多分支语句的执行过程如下图所示: |  |

**注意事项: A.else if必须和if组合使用，else if语句末尾可以是一条单独的else语句**

**B.else不能有条件表达式**

**C.我们if语句第三种格式,任何一个条件表达式成立,if语句直接结束.不在往下进行,这是短路语法**

【例3.4】创建IfElseIfPractice类，进行对IfElseIf语句的特性进行一些验证

class IfElseIfPractice{

public static void main(String[] args){

if(false){

System.out.println("1");

}else if(false){

System.out.println("2");

}else if(true){//当发现if条件语句为true时,只执行这条语句,其他的就不再执行,

System.out.println("3");//跳出这个if..else..if语句,执行其他程序

}else if(true){

System.out.println("4");

}else{

System.out.println("5");

}

}

}



##### 3.2.1.4 if条件语句练习

【例3.5】学生考试成绩出来了请对其进行划分等级,要求90--100 A等 80--89 B等 70--79 C等 60--69 D等 60分以下 叫家长,谈人生

分析：我们初看到这道题目时，要注意这么几点

* 考试成绩大于等于0小于等于100,所以我们要考虑输入错误的情况!
* 由于成绩是一个变量，所以这里我们需要一个变量来代表这个成绩
* 等级对于成绩来说是一个区间，并且这是一个条件语句，所以我们需要应用到比较运算符，和逻辑运算符。
* 由于我们在这里需要比较很多区间，所以我们需要考虑用哪种语句，在这里很明显选择第三种if语句

class GradePractice{

public static void main(String[] args){

int score = 178 ;

if(score <= 100 && score >= 90){

System.out.println("您的成绩是:"+score+",相对应的成绩等级为:A等");

}else if(score < 90 && score >= 80){

System.out.println("您的成绩是:"+score+",相对应的成绩等级为:B等");

}else if(score < 80 && score >= 70){

System.out.println("您的成绩是:"+score+",相对应的成绩等级为:C等");

}else if(score < 70 && score >= 60){

System.out.println("您的成绩是:"+score+",相对应的成绩等级为:D等");

}else if(score >=0 && score < 60){

System.out.println("您的成绩是:"+score+",叫家长,谈人生!");

}else{

System.out.println("成绩输入错误请检查后重新输入!");

}

}

}



【例3.6】给出月份,判断它所处的季节 3-5 春季 6-8 夏季 9-11 秋季 12-2 冬季

分析：我们首先来看需求

* 首先我们需要定义一个变量来代表月份，我们用这个变量来在区间中比较；
* 第二这个变量要符合我们的要求，即大于等于1，小于等于12，所以我们要考虑输入错误的情况；
* 第三我们要选择合适的语句，在这里我们有很多种方法，下面一一介绍

方法一：一个个比较,使用简单if语句。

class SeasonPractice1{

public static void main(String[] args){

int month = 9 ; //声明一个代表月份的变量

if( month == 1){

System.out.println("这个月份属于冬季!");

}

......

if( month == 3){

System.out.println("这个月份属于春季!");

}

......

if( month == 6){

System.out.println("这个月份属于夏季!");

}

......

if( month == 9){

System.out.println("这个月份属于秋季!");

}

......

if( month == 12){

System.out.println("这个月份属于冬季!");

}else{

System.out.println("您输入的数据储存在问题,请检查后重新输入!");

}

}

}



方法2：使用逻辑运算符加上if语句。

class SeasonPractice2{

public static void main(String[] args){

int month = 2 ;

if(month == 12 || month == 1 || month == 2){

System.out.println("这个月份属于冬季!");

//我们在这里使用if...else.if语句,需要明白语句的一个特性当有else.if语句为真时即跳出语句不再进行if.else语句,进行其他语句

}else if(month == 3 || month == 4 || month == 5){

System.out.println("这个月份属于冬季!");

}else if(month == 6 || month == 7 || month == 8){

System.out.println("这个月份属于冬季!");

}else if(month == 9 || month == 10 || month == 11){

System.out.println("这个月份属于冬季!");

}else {

System.out.println("对不起,您输入的数据存在问题,请检查后重新输入!");

}

}

}



方法三：使用比较运算符,逻辑运算符和if语句；

class SeasonPractice3{

public static void main(String[] args){

int month = 17 ;

if(month == 1 || month == 2 || month == 12){

System.out.println("这个月份属于冬季!");

}else if(month >= 3 && month <= 5){

System.out.println("这个月份属于春季!");

}else if(month >= 6 && month <= 8){

System.out.println("这个月份属于夏季!");

}else if(month >= 9 && month <= 11){

System.out.println("这个月份属于秋季!");

}else{

System.out.print("对不起,您输入的数据错误,请检查后重新输入!");

}

}

}



方法四：精简代码，从上面的三种方法我们可以看出代码的相同性有这么几点，（变量的判断比较，结果的输出，逻辑运算符的运用），我们将这相同的几点进行精简如下！

class SeasonLastPractice{

public static void main(String[] args){

int month = 7 ;

String str ;

if(month == 1 || month == 2 || month == 12){

str = "这个月份属于冬季!";

}else if(month >= 3 && month <= 5){

str = "这个月份属于春季!";

}else if(month >= 6 && month <= 8){

str = "这个月份属于夏季!";

}else if(month >= 9 && month <= 11){

str = "这个月份属于秋季!";

}else{

str = "对不起,您输入的数据错误,请检查后重新输入!";

}

System.out.print(str);

}

}

【例3.7】比较三个数的大小(if语句的嵌套使用)

class ComparePractice{

public static void main(String[] args){

/\*三个数的比较我们要考虑这些情况,当甲和乙比较后,我们还要考虑已和丙的比较等,所以我们这里就用到了嵌套语句\*/

int a = 2 , b = 5 , c = 4 ;

if(a > b){

if(b > c){

System.out.println("这三个数的大小为"+"a > b > c");

}else if(a > c){

System.out.println("这三个数的大小为"+"a > c > b");

}else{

System.out.println("这三个数的大小为"+"c > a > b");

}

}else{

if(b > c){

if(a > c){

System.out.println("这三个数的大小为"+"b > a > c");

}else{

System.out.println("这三个数的大小为"+"b > c > a");

}

}

}

}

}



【例3.8】求三个数中的最大值；

class MaxPractice{

public static void main(String[] args){

int a = 2 , b = 8 , c = 6 , max = 0;

//如果只是求最大值,那么我们只用拿出大的那个值同最后一个值比较

if(a > b){

if(a > c){

max = a ;

}else{

max = c ;

}

}else{

if(b > c){

max = b ;

}else{

max = c ;

}

}

System.out.print("三个数的最大值为:"+max);

}

}



### 3.3 选择语句

在编程中一个常见的问题就是检测一个变量是否符合某个条件,如果不匹配，再用另一个值来检测它，依次类推。当然，这种问题使用if条件语句也可以完成。

|  |
| --- |
| if（grade == “A”）{  System.out.println(“真棒”);  }  if(grade == “b”){  System.out.println(“做的不错”);  } |

这个程序显得比较笨重,程序员需要测试不同的值来给出输出语句。在Java语言中，当需要对某个表达式的多个值进行检测时,可以使用switch语句,所以这里可以用switch语句将动作组织起来，就能以一个较简单明了的方式来实现“多选一”的选择。语法格式如下：

|  |
| --- |
| switch（表达式）{  case 常量值1：  语句块 1  [break;]  .......  case 常量值n：  语句块 n  [break;]  default:  语句块 n + 1;  [break;]  } |

Switch语句中表达式的值必须是整型或字符型,常量值1~常量值n必须也是整型或字符型。Switch语句首先计算表达式的值，如果表达式的值和某个case后面的变量值相同，则执行该case语句后的若干个语句直到遇到break语句为止。此时如果该case语句中没有break语句，将继续执行后面case中的若干个语句，直到遇到break语句为止。若没有一个常量的值与表达式的值相同，则执行default后面的语句。default语句为可选的，如果它不存在，而且switch语句中表达式的值不与任何case的常量值相同，switch则不做任何处理。

Switch语句注意项：

* **同一个switch语句，case的常量值必须互不相同。**
* **当我们看到switch的时候,这就是一个选择语句**
* **表达式:是一个常量或者变量,byte,short,int,char,所以case取值其实与表达式的类型一致**
* **jdk1.5以后支持枚举(1.5新特性), jdk1.7以后支持String(1.7新特性)**
* **.case:表示的就是一个选项,取值就是选项的值**
* **执行语句:功能代码.**
* **break:switch语句的结束符.**
* **default:默认值,相当于if语句的else**

【例3.9】根据指定的日期（1~7），输出相应的星期；

**package** cn.yorick.switchStatement;

/\*\*

\*

\* **@author** YorickYou

\* **@Date** 2019年3月9日

\* **@version** 1.8

\* GitHub https://github.com/YorickYou/Java-SE.git

\* core switch语句简单练习

\*/

**public** **class** SwitchWeekPractice {

**public** **static** **void** main(String[] args){

**int** day = 9 ;

**switch**(day){

**default**: //若没有一个常量的值与表达式的值相同则默认执行default,

System.***out***.println("请检验您的输入数字是否正确!");

//如果没有发现break,则继续执行下面的case语句直到发现break

**case** 1:

System.***out***.println("根据您输入的数字可以确定是:星期一");

**break**;

/\*default: 一个switch语句中只能有一个default,否则会提示标签重复

System.out.println("请检验您的输入数字是否正确!");\*/

**case** 2:

System.***out***.println("根据您输入的数字可以确定是:星期二");

**break**;

**case** 3:

System.***out***.println("根据您输入的数字可以确定是:星期三");

**break**;

**case** 4:

System.***out***.println("根据您输入的数字可以确定是:星期四");

**break**;

**case** 5:

System.***out***.println("根据您输入的数字可以确定是:星期五");

**break**;

**case** 6:

System.***out***.println("根据您输入的数字可以确定是:星期六");

**break**;

**case** 7:

System.***out***.println("根据您输入的数字可以确定是:星期日");

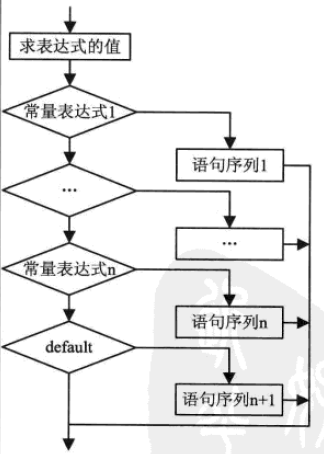
**break**;

}

}

}

Switch语句的执行过程：



在switch语句中,case语句后常量表达式的值可以是正数,但局部可以是实数,例如下面的代码就是不合法的。

Case 1.1：

【例3.10】根据输入的城市名字，查询出城市所在的省；

**package** cn.yorick.switchStatement;

/\*\*

\* **@author** YorickYou

\* **@Date** 2019年3月9日

\* **@version** 1.8

\* GitHub https://github.com/YorickYou/Java-SE.git

\* core 根据给定的城市名字,查询所属省份

\*/

**public** **class** ProvincePractice {

**public** **static** **void** main(String[] args){

String str = "洛阳";

**switch**(str){

**case** "石家庄":

System.***out***.println("这是河北省会;");

**break**;

**case** "济南":

System.***out***.println("这是山东省会;");

**break**;

**case** "郑州":

System.***out***.println("这是河南省会;");

**break**;

**case** "洛阳":

System.***out***.println("这是河南地级市;");

**break**;

**default**:

System.***out***.println("对不起您所查找的城市在数据库中没有发现,请检查后输入!");

}

}

}



【例3.11】根据给定的月份，判断所属的季节；冬季（12,1,2）春季（3，4，5）夏季（6，7，8）秋季（9,10,11）

**package** cn.yorick.switchStatement;

/\*\*

\* **@author** YorickYou

\* **@Date** 2019年3月9日

\* **@version** 1.8

\* GitHub https://github.com/YorickYou/Java-SE.git

\* core 根据给定的月份，判断所属的季节

\*/

**public** **class** SwitchMonthPractice {

**public** **static** **void** main(String[] args){

**int** month = 9 ;

String str = "";

**switch**(month){

//运用Switch的省略,将break适当省略,将共性放在一处源代码

**case** 1:

str = "冬季";

**case** 2:

str = "冬季";

**case** 12:

str = "冬季";

**break**;

**case** 3:

str = "春季";

**case** 4:

str = "春季";

**case** 5:

str = "春季";

**break**;

**case** 6:

str = "夏季";

**case** 7:

str = "夏季";

**case** 8:

str = "夏季";

**break**;

**case** 9:

str = "秋季";

**case** 10:

str = "秋季";

**case** 11:

str = "秋季";

**break**;

**default**:

System.***out***.println("请检查您所输入的数字是否在范围中!");

}

System.***out***.print("根据您所输入的月份判断季节为:"+str);

}

}



#### 3.3.1 注意：

##### 3.3.1.1 总是在switch语句里加上default

Switch…作为最滑稽的表达式之一，我不知道是该心存敬畏还是默默哭泣。不管怎样，我们既然无法摆脱 switch ，在必要的时候我们最好能够正确使用它，例如：

// Bad

switch (value) {

    case 1: foo(); break;

    case 2: bar(); break;

}

// Good

switch (value) {

    case 1: foo(); break;

    case 2: bar(); break;

    default:

        throw new ThreadDeath("That'll teach them");

}

因为在当 value=3 被引入到软件中的时候，default 就能发挥作用，使其正常运行！别和我提 enum 类型，因为这对 enums 也一样适用。

##### 3.3.1.2 用大括号隔开 switch 的每一个 case 块

事实上，switch是最坑爹的语句，任何喝醉了或是赌输了的人都可以在某种语言中使用它。看看下面这个例子：

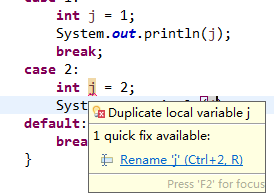
// Bad, doesn't compile

switch (value) {

    case 1: int j = 1; break;

    case 2: int j = 2; break;

}



// Good

switch (value) {

    case 1: {

        final int j = 1;

        break;

    }

    case 2: {

        final int j = 2;

        break;

    }

    // Remember:

    default:

        throw new ThreadDeath("That'll teach them");

}

在switch语句中，为所有的case都只定义了一个作用域。事实上，这些case不是真正意义上的语句，他们更像是标签，而switch就是指向这些标签的goto语句。事实上，你甚至可以把case语句和 [惊人的FORTRAN77项声明](http://docs.oracle.com/cd/E19957-01/805-4939/6j4m0vn99/index.html" \t "C:/Users/Administrator/Desktop/_blank)类比，对于FORTRAN，它的神秘已经超越了它的功能。

这意味着变量final int j 可以被任何case访问，不论我们是否有break。看起来并不是很直观。我们可以通过添加简单的花括号为每一个case创建一个新的嵌套的作用域，当然不要忘了在每个 case 的语句块最后加 break。

#### 3.3.2 问答:

switch case语句中必须得有default语句么

不是必须的。 switch case 语句是一个条件选择语句，找到相同的case值做为入口，执行后面的程序；若所有的case都不满足，则找default入口；若未找到则退出整个switch语句。 所以default只是一个备用的入口，有没有都无所谓，但是为了程序更完整、更安全，建议写上default。

在SWITCH语句中可以使用return关键字代替break关键字吗？

可以，理论上[switch函数](https://www.baidu.com/s?wd=switch%E5%87%BD%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "C:/Users/Administrator/Desktop/_blank)中每个case语句最后都要加个break（原因是为了只执行一个符合条件的case语句 然后就跳出switch）执行break后 跳出switch 继续执行switch后的语句 如果用return代替 那么就是跳出整个函数。  
 用return代替后结果要看***使用switch语句的函数***是什么情况

* 如果是void [main函数](https://www.baidu.com/s?wd=main%E5%87%BD%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "C:/Users/Administrator/Desktop/_blank) return 直接结束程序
* 如果是int main 之类的 return 后需要加返回值 同样直接结束函数
* 如果是被调用的函数，return后面要接相应的返回值 然后跳出被调用的函数，继续运行主函数

如下我们给出一个实例代码：

**package** cn.yorick.switchStatement;

**import** java.util.Scanner;

**public** **class** TestSwitch {

**public** **static** **void** main(String[] args){

Scanner input = **new** Scanner(System.***in***);

System.***out***.print("Enter an integer: ");

**int** number = input.nextInt();

System.***out***.println(*getReturn*(number));

}

**public** **static** String getReturn(**int** number){

String set0 = "Yes";

String set1 = "No";

String set2 = "Cancel";

**switch**(number){

**case** 0 : **return** set0;

**case** 1 : **return** set1;

**case** 2 : **return** set2;

**default**: **return** "Not Find";

}

}

}



在SWITCH语句中可以使return关键字和break关键字并存吗？

这是不可以的，return有跳出整个函数的语义，而break是跳出一个语句块的语义，它们两个在一起是语义冲突的，我们在下面这个代码中看一下。

|  |  |
| --- | --- |
| **case** 0 : **return** set0;**break**;  编译不能通过会提示不能达到的代码 |  |
| **case** 0 : **break**;**return** set0;、  和上面一样提示“编译不能通过会提示不能达到的代码” |  |

### 3.4 循环语句

循环语句就是满足一定条件的情况下反复执行某一操作，直到条件不再满足为止，这里一定要注意后一句，如果缺失那么就是死循环。在Java中提供了3中常用的循环语句,分别是while循环语句、do...while循环语句和for循环语句。下面分别对这3中循环语句进行介绍。

#### 3.4.1 while循环

While循环语句也称为条件判断语句，它的循环方式为利用一个条件来控制是否要继续反复执行这个语句。语法格式如下：

|  |  |
| --- | --- |
| While（条件表达式）{  执行语句  } |  |

当条件表达式的返回值为真时，则执行“{}”中的语句，当执行完“{}”中的语句后，重新判断条件表达式的返回值，直到表达式返回的结果为假时，退出循环。While循环语句的执行过程如上图所示

【例3.12】将1到100的所有奇数都打印出来，并对他们进行求和运算；

分析：首先我们要明白这是一个累加的运算，我们在代码中涉及到的是两个变量，奇数和累加的和；对于奇数，我们对他的判断应该是除2取余为1,下面我们通过代码来分析；

class OddPractice{

public static void main(String[] args){

int i = 0 ,sum = 0; //定义一个代表1到100的变量和奇数和的变量

while(i <= 100){

if(i%2==1){

System.out.print(i+"、");

sum += i;

}

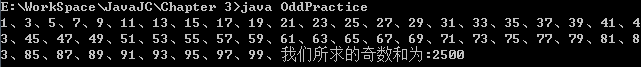
i++;//省略的话死循环,因为i值不变while后边的条件表达式永远为真

}

System.out.println("我们所求的奇数和为:"+sum);

}

}



注意：有时候我们大意会将while表达式的括号后加“；”，例如：

While（x == 5）;

System.out.println(“x的值为5”);

这时程序会认为要执行的是一条空语句,而进入死循环,Java编译器又不会报错,可能会浪费很多时间去调试,应该要注意这个问题。

#### 3.4.2 do...while循环

do...while循环语句与while循环语句类似,它们之间的区别是while循环语句为先判断条件是否成立再执行循环体,而do...while循环语句则先执行一次循环后,再判断条件是否成立。也就是说do...while循环语句中大括号中的程序段至少要被执行一次。语法格式如下：

|  |  |  |
| --- | --- | --- |
| Do{  执行语句  }while(条件表达式); | 与while语句的一个明显区别是do...while循环语句在结尾处多了一个分号(;)。根据do...while循环语句的语法特点总结出do...while循环语句的执行流程如左所示: |  |

【例3.13】比较while和do...while;

class CompareWhilePractice{

public static void main(String[] args){

int a = 100 , b = 100 ;

//鲜明的比较while和do...while的区别

while(a == 60){

System.out.println("Ok"+a);

a--;

}

do{

System.out.println("Ok"+b);

b--;

}while(b == 60);

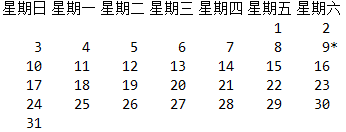
}

}



【例3.14】打印当前月的月历

|  |
| --- |
| **package** cn.yorick.gregorianCalendar;  **import** java.text.DateFormatSymbols;  **import** java.util.Calendar;  **import** java.util.GregorianCalendar;  /\*\*  \* **@author** YorickYou  \* **@Date** 2019年3月9日  \* **@version** 1.8  \* GitHub https://github.com/YorickYou/Java-SE.git  \* core 使用GregorianCalendar打印当前月的月历  \*/  **public** **class** GregorianCalendarTest {  **public** **static** **void** main(String[] args) {  GregorianCalendar date = **new** GregorianCalendar(); //获取当前时间的日历系统对象  **int** month = date.get(Calendar.***MONTH***); //获取当前日期的月份  **int** today = date.get(Calendar.***DAY\_OF\_MONTH***); //获取当前时间的当前月天数  String weekdays[] = **new** DateFormatSymbols().getShortWeekdays();  **for** (**int** i = 1; i < 8; i++) {  System.***out***.print(weekdays[i] + " ");  }  System.***out***.println();  //设置日期为当前月的第一天  date.set(Calendar.***DAY\_OF\_MONTH***, 1);  //获取当前月第一天对应的星期  **int** value = date.get(Calendar.***DAY\_OF\_WEEK***);  //打印日历的第一行  **int** cnt = 1;  **while** (weekdays[cnt] != weekdays[value]) {  System.***out***.print(" ");  ++cnt;  }  **do** {  **int** day = date.get(Calendar.***DAY\_OF\_MONTH***);  System.***out***.printf("%5s",day);  **if**(day == today) {  System.***out***.print("\*");  }**else** {  System.***out***.print(" ");  }  **if**(weekdays[date.get(Calendar.***DAY\_OF\_WEEK***)] == weekdays[7]) {  System.***out***.println();  }  date.add(Calendar.***DAY\_OF\_MONTH***, 1);//日期增加1  } **while** (date.get(Calendar.***MONTH***) == month);  }  } |



#### 3.4.3 for循环

for循环语句是Java程序设计中使用最频繁的循环语句之一。一个for循环可以用来重复执行某条语句，知道某个条件得到满足。在Java5以后新增了foreach语法更使for循环如鱼得水,不同于while循环，for循环语句是支持迭代的一种通用结构，利用每次迭代之后更新的计数器或类似的变量来控制迭代次数。下面针对这两种for循环进行详细探讨。

##### 3.4.3.1 for语句

|  |  |  |
| --- | --- | --- |
| 语法格式如下:  For(表达式1;表达式2;表达式3){  语句序列  }  For语句的第一部分通常用于对计数器初始化；第二部分给出每次新一轮循环执行前要检查的循环条件；第三部分只是如何更新计数器；  同C++一样，尽管Java允许在for循环的各个部分放置任何表达式，但有一条不成文的规则：for语句的3个部分应该对同一个计数器变量进行初始化、检查和更新。若不遵守规则，编写的循环通常晦涩难懂。 | 表达式1:初始化表达式,负责完成变量的初始化,可以在for循环前面进行,不一定要放在()中进行;  表达式2:循环条件表达式,值为boolean型的表达式,指定循环条件，需要注意：在循环中检测两个浮点数是否相等要格外小心，因为它们往往超出我们的想象，也是我们最容易忽略的地方。  表达式3:循环后操作表达式,负责修整变量,改变循环条件  在执行for循环时,首先执行表达式1,完成某一变量的初始化工作;下一步判断表达式2的值,若表达式2的值为true,则进入循环体;在执行完循环体后紧接着计算表达式3,这部分通常是增加或减少循环控制变量的一个表达式.这样一轮循环就结束了。第二轮循环从计算表达式2开始，若表达式2返回true，则继续循环，否则跳出整个for语句。For循环语句的执行过程如左所示： |  |

小结：

* 关键字:for：for循环本质上只是while循环的一种简化形式，它们作用相同，但也不能完全划等号；所有的while循环都可以改为for循环但不是所有的for循环都可以改写为while循环；如下代码：就很难改写为while,这里更适合使用for循环

Int sum = 0;

For(int i = 0 ; i < 4 ; i++){

If(i % 3 == 0){

Continue;

}

}

* 固定格式:for(初始化表达式(3);循环条件表达式(1);循环后的操作表达式(4))(2){执行语句(5)}
* 运行顺序: 循环条件为true时 3--1--5--4--1--5

循环条件为false时 3--1 当循环条件为false时,循环终止

注意事项:

1、变量作用域问题：

如果后边要使用这个变量，需要定义在for循环前

如果后边不使用这个变量，定义在for循环中和定义在循环前都可，但建议定义在循环中，一是便于理解，而是控制变量生命周期

2、第一个表达式可以放置在循环体前，可以为空

3、第三个表达式可以放置在循环体中，可以为空

4、第二个表达式可以为空，不报错，但是为死循环。除非特殊业务或算法需求，否则不会出现该种情况

【例3.14】计算1到100相加的和；

class SumPractice{

public static void main(String[] args){

int sum = 0;

for(int i = 1;i<101;i++){

sum += i;

}

System.out.println("1到100的累计相加的和为:"+sum);

}

}



注意：在编程时，有时会使用for循环的特殊语法格式来实现无限循环，语法格式为：

for（；；）{

......

}

对于这种无限循环可以通过break语句跳出循环，例如：

For（；；）{

If(i<20){

System.out.println(“无限循环”);

}

break；

}

##### 3.4.3.2 foreach语句

foreach语句是for语句的特殊简化版本，foreach语句并不能完全取代for语句，然而任何foreach语句都可以改为for语句版本。foreach并不是一个关键字，习惯上将这种特殊的for语句格式称之为foreach语句。foreach语句在遍历数组等方面为程序员提供了很大方便（在数组中我们将会看到大量应用）。foreach语句也叫增强for循环，增强for循环是for的一种新用法！用来循环遍历**数组**和实现Iterable接口的**集合**。***编译器在编译时会将加强for转换为普通for，***增强for的语法格式如下：

for（元素变量x:遍历对象 obj）{

引用了x的Java语句；

}

x的值从数组的第一个元素一直到最后一个元素！foreach语句中的元素变量x，不必对其进行初始化,仅仅是声明了一个变量。下面通过一个小例子来说明；

【例3.15】创建一个一维数组，对其进行遍历，通过两种for循环；

public class Repetition{

public static void main(String args[]){

int arr[] = {1,2,3,4,5};

System.out.print("数组中的元素为：");

/\*标准for循环

for(int i = 0;i<5;i++){

System.out.print(arr[i]+",");

}\*/

for(int x:arr){

System.out.print(x+",");

//foreach语句，元素变量x为遍历数组的元素值

}

}

}



增强for的冒号左边是定义变量，右边必须是数组或实现Iterable接口的集合类型。例如上例中循环遍历的arr这个int数组，增强for内部会依次把arr中的元素赋给变量i。

增强for

\* 1. 只是语法简单了一点点！

\* 2. 它没有老for增大！由于增强for没有使用索引,所以没有办法来对特定索引的元素进行操作。

特点：

\* 1. 只能遍历数组和实现Iterable接口的集合...

\* 2. 在遍历时不能使用下标

\* 3. 遍历只能从头到尾

语法：

\* 1. for(元素类型 变量名 : 集合或数组对象)

\* 2. 每次循环，获取一个数组中的元素给左边的变量了。

增强for到底可以遍历什么：

\* 1. 可以遍历数组和任何的实现了Iterable接口的集合

\* 2. 增强for的底层是普通for循环，有些地方会说底层是Iterator实际上这不完全正确，下面我们通过一个代码来看一下：

|  |  |
| --- | --- |
| **package** cn.yorick.forStatement;  **import** java.util.ArrayList;  **import** java.util.List;  **public** **class** ForeachTest {  **public** **static** **void** main(String[] args) {  String[] strArr = {"A","B","C","D","E"};  List<String> list = **new** ArrayList<>();  list.add("A");  list.add("B");  list.add("C");  list.add("D");  list.add("E");  **for** (String string : list) {  System.***out***.print(string);  }  System.***out***.println("\r\n---------------");  **for** (String string : strArr) {  System.***out***.print(string);  }  }  }  从右边我们可以看到foreach的底层其实都转换成了普通for循环，数组的遍历和我们使用普通for循环一样，而实现了Iterator接口的集合，则会使用Iterator进行遍历，这也就是为什么foreach支持数组和实现Iterator的集合了，这是底层机制决定的。 | 反编译  package cn.yorick.forStatement;  import java.io.PrintStream;  import java.util.\*;  public class ForeachTest  {  public ForeachTest()  {  }  public static void main(String args[])  {  String strArr[] = {  "A", "B", "C", "D", "E"  };  List list = new ArrayList();  list.add("A");  list.add("B");  list.add("C");  list.add("D");  list.add("E");  String string;  for(Iterator iterator = list.iterator(); iterator.hasNext(); System.out.print(string))  string = (String)iterator.next();  System.out.println("\r\n---------------");  String args1[];  int j = (args1 = strArr).length;  for(int i = 0; i < j; i++)  {  String string = args1[i];  System.out.print(string);  }  }  } |

#### 3.4.4 循环语句小结

为了完成重复的事情，我们通常要使用循环来完成需求

3.1 while循环 3.2 do while 3.3 for循环

While语句格式 do{执行语句} for(初始化表达式;循环

条件表达式;循

While(条件表达式){ while(条件表达式); 环后的操作表达式){

执行语句 } 执行语句 }

for条件相对复杂，但是最常用,while/do while 简单易懂--具有明确的判断条件时使用，Do while循环较少使用；

循环需要的内容：

1、循环条件----循环条件表达式 如果没有循环条件就会一直循环下去----存在死循环现象

2、循环体：执行语句----循环执行的代码语句

3、循环变量定义并初始化-----初始化表达式

4、循环变量的变化情况----循环后的操作表达式

***如果已经可以判断出循环次数的话，使用for循环比较方便，如果不知道循环次数使用while循环比较合适。***

#### 3.4.5 循环语句的练习

【例3.15】将1到100中的偶数打印出来，并且对他们进行求和；

class EvenPractice{

public static void main(String[] args){

int sum = 0;

for(int i=1;i<101;i++){

if(i%2==0){

System.out.print(i+"、");

sum += i;

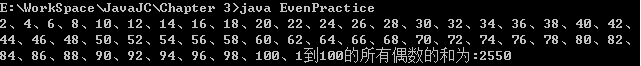
}

}

System.out.println("1到100的所有偶数的和为:"+sum);

}

}



对比【例3.12】while语法计算奇数和

【例3.16】珠峰高度8848m , 一张纸厚度为0.1mm ,判断将一张纸对折多少次可以达到珠峰高度，但不超过珠峰高度,并输出打印对折次数,以及对折后的纸的厚度!分别使用For循环和While循环

class PaperPractice{

public static void main(String[] args){

int x = 88480000 , y = 1 ;//定义珠峰高度和纸的厚度

int n = 0 ;//用n来表示折叠的次数

/\*for(;y<=x;n++){ //for循环的运行顺序: 没有折叠时n=0,纸的厚度小于珠峰高度,纸的厚度乘2加倍,折叠次数加1,此时代表折叠了一次

y = y\*2; //折叠到倒数第2次时,纸的厚度小于珠峰高度,此时继续循环,纸的厚度超出珠峰高度,次数也会比实际需求大1，所以实际上只需折叠26次，三次循环都是如此

}\*/

/\*while(y<=x){

n++; //在原有折叠次数上再折叠一次

y = y \* 2 ; //折叠一次后纸的厚度翻倍

}\*/

do{

y=y\*2;

n++;

}while(y<=x);

System.out.println("折叠"+n+"次后纸的厚度为:"+y);

System.out.println("折叠的倒数第二次纸的厚度是:"+y/2);

}

}



分析：当我们折叠第一次时，n为1，厚度翻倍，当折叠到倒数第二次时，折叠后，此时纸的厚度仍小于珠峰高度，仍然满足循环条件，此时会再循环一次，但循环后纸的厚度超过珠峰高度，循环停止，所以如果我们求纸折叠多少次才能达到珠峰高度时，并不超过时，应该为最后得到的折叠次数减去1。另外观察我们也可以发现对于不知道循环次数的循环我们使用while循环比较容易理解！

【例3.17】给出一个整数，计算出它的阶乘；使用for和while循环

分析：阶乘：[正整数](http://baike.baidu.com/view/464125.htm" \t "http://baike.baidu.com/_blank)阶乘指从 1 乘以 2 乘以 3 乘以 4 一直乘到所要求的数

class FactorialPractice{

public static void main(String[] args){

int x = 12 ;//定义一个我们要阶乘的数

int p = 1;//定义一个阶乘后的结果,由于是从1开始,所以我们将它定义为1

int y=1;//定义一个乘法的开始数字

/\*for(;y<=x;y++){

p = p\*y;

}\*/

while(y<=x){

p = p\*y;

y++;

}

System.out.println("我们所求的阶乘是:"+p);

}

}



【例3.18】规则图形的打印，打印一个给定行数和给定列数的规则图形；

class FigurePractice{

public static void main(String[] args){

//打印一个给定行数给定列数的规则图形(行高9,列数12)

int line = 9 , list = 12;

for(int x=1;x<=line;x++){

//行的循环里嵌套了列的循环换

for(int y=1;y<=list;y++){

System.out.print("\* ");

}

//列尾我们要注意换行

System.out.println("");

}

//打印一个直角三角形

for(int x=0;x<5;x++){

for(int q=0;q<=x;q++){

System.out.print("\* ");

}

System.out.println("");

}

//打印一个倒立的正三角,实际上就是打印一个空格三角形和另一个三角形

for(int y=5;y>0;y--){//定义正三角形的边长,正三角形的边长和它的行数相同

for(int s=5;s>y;s--){//打印空格的循环和打印\*的循环要注意先后,打印空格实际上就是打印一个直角三角形

System.out.print(" ");

}

for(int z=y;z>0;z--){

System.out.print("\* ");

}

System.out.println("");

}

//另一种方法

int last = 5;

for(int y=0;y<last;y++){

for(int q=0;q<y;q++){

System.out.print(" ");

}

for(int z=y;z<last;z++){

System.out.print("\* ");

}

System.out.println("");

}

//打印一个由数字构成的正三角形

for(int i=1;i<10;i++){

for(int j=1;j<10-i;j++){//最多的前面有八个空格

System.out.print(" ");

}

for(int j=i;j>0;j--){//数字的打印我们分两部分,第一部分从i倒序到1

System.out.print(j);

}

for(int j=1;j<i;j++){//第二部分从2打印当i

System.out.print(j+1);

}

System.out.println("");

}

}

}

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

【例3.19】打印一个正九九乘法表和一个倒立的九九乘法表

class TablePractice{

public static void main(String[] args){

for(int x=1;x<10;x++){ //我们要注意这和大三角形的思想是一致的,但是我们要打出来的是一些字符串的链接,

for(int y=1;y<=x;y++){//并且这些链接中包含两个变量

System.out.print(y+"\*"+x+"="+x\*y+" ");

}

System.out.println("");

}

for(int x=9;x>0;x--){

for(int y=1;y<=x;y++){

System.out.print(y+"\*"+x+"="+x\*y+" ");

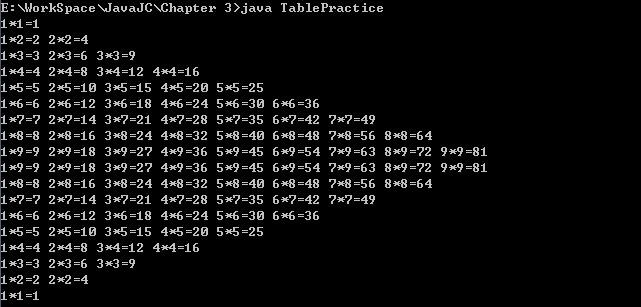
}

System.out.println("");

}

}

}



【例3.20】有一对兔子,从第三个月开始产小兔子,每个月产一对,小兔子在第三个月也开始生小兔子,假设无意外情况,20个月后有多少只小兔子

分析:发现规律,做如下计算:

第一个月有兔子1对,

第二个月有兔子1对,

第三个月有兔子2对,

第四个月有兔子3对,

第五个月有兔子5对,

第六个月有兔子8对......从上面我们可以看到有这么个规律“从第三个月开始腿子的对数是前两个月的兔子对数之和”，由此我们可以对所求兔子对数进行一个规律运算；

变量分析：通过上面的规律我们可以得出对于兔子对数我们需要3个变量，即第一个月兔子对数M1，第二个月兔子对数M2，第三个月兔子对数M3，M3=M1+M2；除此之外我们需要一个月份的变量，者用于循环；

class RabbitPractice{

public static void main(String[] args){

int M1 = 1 ,M2 = 1 ,M3 = 2 ;

//通过规律我们从第三个月开始按照规律差计算

for(int month=3;month<=20;month++){

M3 = M1 + M2;//通过前两个月兔子对数计算出第三个与兔子对数

M1 = M2; //此时向下循环的话,最早的第二个月变成了现在的第一个月,

M2 = M3; //第三个月变成了第二个月,再下次循环中的第三个月就是前两个月之和

}

System.out.print("20个月后我们共有兔子"+M3+"对;");

}

}



【例3.21】要求一个五位数个位=万位 十位=千位 个位+十位+万位+千位=百位

分析：我们从上面题目中分析出来当给定我们一个五位数时我们应该求出这个五位数的个十百千万位，然后进行运算，以一段代码为例：

int a = 98765;

System.out.println("这个数的个位是"+a%10);//整数%10,得到个位数

System.out.println("这个数的十位数是"+a/10%10);//除以10得到的结果,十位数变成了个位数,按照上面求出所需结果

System.out.println("这个数的百位数是"+a/10/10%10);

System.out.println("这个数的千位数是"+a/10/10/10%10);

System.out.println("这个数的万位数是"+a/10/10/10/10%10);

从上面我们得出了个十百千位的算法，那么我们就可以在循环中进行判断，然后输出结果；我们在循环中最先开始比较的是最小的五位数直到最大的五位数；

class NumberPractice{

public static void main(String[] args){

int ud,td,hd,thd,md;//声明五个变量代表个十百千位

//System.out.print();

for(int num=10000;num<100000;num++){

ud = num%10;

td = num/10%10;

hd = num/10/10%10;

thd = num/10/10/10%10;

md = num/10/10/10/10%10;

//判断满足条件的数字

if(ud==md&&td==thd&&ud+md+td+thd==hd){

System.out.print(num+" ");

}

}

}

}



【例3.22】水仙花问题；

1.什么是水仙花问题-----水仙花数是指一个 n 位数 ( n≥3 )，它的每个位上的数字的 n 次幂之和等于它本身。

2.假定是一个三位数num,那么num%10得出的结果就是个位值g,num/10%10得出的结果就是十位值s,num/10/10%10得出的结果就是百位值b,根据需求,g\*g\*g+s\*s\*s+b\*b\*b==num

3.要了解什么是水仙花数，这样才能快速选择是用什么方式、方法，甚至算法来解决问题，在这里我们来计算一下4位数的水仙花问题；

class DaffodilPractice{

public static void main(String[] args){

int ud,td,hd,thd;//声明个十百千位

for(int num=1000;num<10000;num++){

ud=num%10;

td=num/10%10;

hd=num/10/10%10;

thd=num/10/10/10%10;

if(ud\*ud\*ud\*ud+td\*td\*td\*td+hd\*hd\*hd\*hd+thd\*thd\*thd\*thd==num){

System.out.print(num+" ");

}

}

}

}



【例3.23】打印一个空心菱形和一个实心菱形

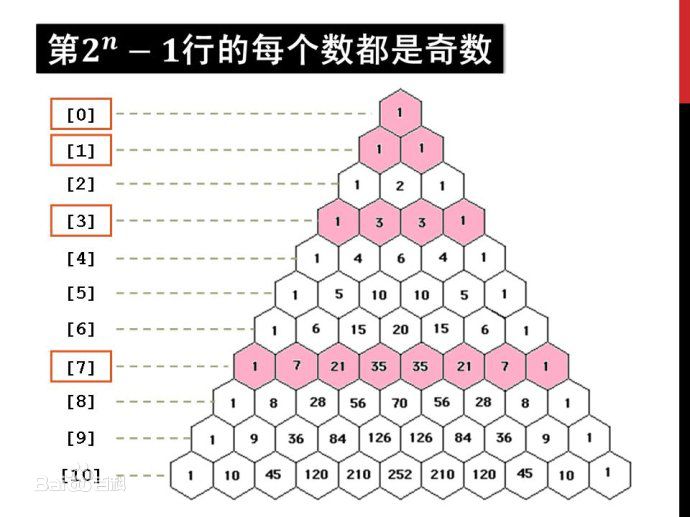
分析:打印菱形我们知道,无非就是打印三角,那么打印空心菱形呢?我们是不是可以按照同样的思路来分析呢?我们可以将菱形分为打印直三角空格,打印星号,打印正三角空格,打印星号;

下面通过代码来实现;

|  |  |  |
| --- | --- | --- |
| **package** cn.yorick.forStatement;  **public** **class** Diamond {  **public** **static** **void** main(String[] args) {  *fun1*(9);  }  **public** **static** **void** fun1(**int** lineNumber){  /\*根据行数判断循环次数,如果行数是奇数那么循环此数就是column/2+1,  \* 如果是偶数,那么空心菱形只能打印奇数,要做出提示\*/  **int** half = lineNumber / 2;  **int** moreHalf = lineNumber / 2 + 1;  //行数是奇数,进行打印  **if**(lineNumber % 2 == 1){  System.***out***.println("空心菱形：");  **for**(**int** k = 1;k <= moreHalf;k++){  **for**(**int** i = 1;i <= (moreHalf - k);i++){  System.***out***.print(" ");  }  System.***out***.print("\*");  **for**(**int** i = 1;i <= ( (k - 2) \* 2 + 1);i++){  System.***out***.print(" ");  }  **if**(k != 1){  System.***out***.print("\*");  }  System.***out***.println();  }  **for**(**int** k = half;k >=1;k--){  **for**(**int** i = 1;i <= (moreHalf - k);i++){  System.***out***.print(" ");  }  System.***out***.print("\*");  **for**(**int** i = 1;i <= ( (k - 2) \* 2 + 1);i++){  System.***out***.print(" ");  }**if**(k != 1){  System.***out***.print("\*");  }  System.***out***.println();  }  System.***out***.println("实心菱形：");  **for**(**int** i = 1;i <= lineNumber;i++){  **if**(i <= lineNumber/2 + 1){  **for**(**int** k = 1;k <= lineNumber/2 + 1- i;k++){  System.***out***.print(" ");  }  **for**(**int** k = 1;k <= i;k++){  System.***out***.print("\* ");  }  System.***out***.println();  }**else**{  **for**(**int** k =1;k <= (i -(lineNumber/2 + 1));k++){  System.***out***.print(" ");  }  **for**(**int** k = 1;k <= (2 \*(lineNumber/2 + 1) - i);k++){  System.***out***.print("\* ");  }  System.***out***.println();  }  }  }**else** {  System.***out***.println("对不起,空心菱形行数应该为大于0的奇数,请确认输出!");  }  }  } | Image | 分析:  1.分析可得菱形的行数和列数必须为奇数;  2.分析可得不管是实心菱形还是空心菱形,那么打印\*的行数和列数是相同的;  3.打印菱形时,可以分为两大部分,打印菱形的上半部分(moreHalf = linenumber/2+1)和下半部分(half = linenumber/2),而上半部分又可分为一个空格的直三角形和菱形  4.打印空格直三角形时,第一行需要打印的空格数目是moreHalf-1,然后打印一个\*;第二行需要打印moreHalf-2个空格,然后打印一个\*,然后打印一个空格,然后再打印一个\*;  打印直角三角空格代码如下:  //k=1,2.....moreHalf  For(int i = 0;i<moreHalf-k;i++){打印}  //正三角形空格,k=1时不打印,2打印1个,3打印3个,4打印5,5打印7..   1. ------0 (k-2)\*2+1 2. ------1 3. ------3 4. ------5 5. ------7 6. ------9   **for**(**int** i = 1;i <= ( (k - 2) \* 2 + 1);i++){打印}  //打印\*  //换行,这样一行就结束了,循环结束,那么上半部分也就打印完成了  //下半部分的打印,思想同上半部分相同,只需将k倒置即可 |

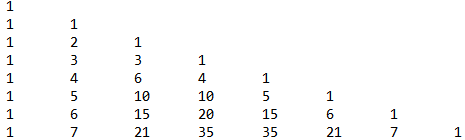
【例3.24】打印杨辉三角；分析:在代码实现之前我们先来看一下什么是杨辉三角,

杨辉三角，是[二项式系数](https://baike.baidu.com/item/%E4%BA%8C%E9%A1%B9%E5%BC%8F%E7%B3%BB%E6%95%B0 /t https://baike.baidu.com/item/%E6%9D%A8%E8%BE%89%E4%B8%89%E8%A7%92/_blank)在三角形中的一种几何排列。在欧洲，这个表叫做[帕斯卡三角形](https://baike.baidu.com/item/%E5%B8%95%E6%96%AF%E5%8D%A1%E4%B8%89%E8%A7%92%E5%BD%A2 /t https://baike.baidu.com/item/%E6%9D%A8%E8%BE%89%E4%B8%89%E8%A7%92/_blank)。它把二项式系数图形化，把组合数内在的一些代数性质直观地从图形中体现出来，是一种离散型的数与形的优美结合。实现结果如下图所示:



分析:杨辉三角的规则是:每一行的第一个值为1,每一行的最后一个值为1,当前数=上一行的数+上一行的数的前一个数;明白了规则我们就来分析一下,我们看杨辉三角是不是很像一个二维数组呢?数组的长度为列数,二维数组中角标0对应的是一个长度为1的数组,二维数组角标1对应的是一个长度为2的数组;根据这个我们下面来通过代码实现;

|  |
| --- |
| **package** cn.yorick.forStatement;  **public** **class** YangHui {  **public** **static** **void** main(String[] args) {  *fun1*(8);  }  /\*\*  \* 杨辉三角:  \* 1.每行的头一个数是1;  \* 2.每行的最后一个数是1;  \* 3.当前数是上一行的数+其前一个数  \*/  **public** **static** **void** fun1(**int** column){  /\*创建二维数组,来作为杨慧三角中的元素的容器,二维数组的长度是杨慧三角的列数,  \* 二维数组中的一维数组的长度是当前行的元素个数+空格数\*/  **int**[][] intArr = **new** **int**[column][column];  **for** (**int** i = 0; i < intArr.length; i++) {  //动态开辟空间,每一行的空间大小和行号是相同的  intArr[i] = **new** **int**[i+1];  }  //为数组填充元素  **for** (**int** i = 0; i < intArr.length; i++) {  //每一行第一个是1,最后一个也是1  intArr[i][0] = 1;  intArr[i][i] = 1;  //注意杨慧三角的运算是从第三行开始的  **for** (**int** j = 1; j < i; j++) {  //当前数是上一行的数+其前一个数,实际上只有到了第三行才会运算  intArr[i][j] = intArr[i-1][j]+intArr[i-1][j-1];  }  }  //循环过滤  **for** (**int** j = 0; j < intArr.length; j++) {  **for** (**int** j2 = 0; j2 < intArr[j].length; j2++) {  System.***out***.print(intArr[j][j2]+"\t");  }  //打印完一行换行  System.***out***.println();  }  }  } |



### 3.5 break、continue和循环标号&&return

Java语句提供了3中跳转语句，分别是break语句、continue语句和return语句。下面对这三种跳转语句进行详细介绍。

#### 3.5.1 break

break语句我们应该都不会陌生,在介绍switch语句时已经应用过了。在switch语句中，break语句用于终止下面case语句的比较。实际上，break语句还可以应用在for、while和do...while循环语句中，用于强行退出循环，也就是忽略循环体中任何其他语句和循环条件的限制。

break关键字它的作用在于跳出当前循环(即break所在的循环,如果存在for(){for(){break;}},那么break只是跳出内存循环,外层循环继续;)，应用范围为循环语句，选择语句。

【例5.25】计算1到100的所有偶数的和，要求当和大于250时跳出，并输出这个和；

class SumDemoPractice{

public static void main(String[] args){

int sum = 0 , i=1;

for(;i<101;i++){

if(i%2==0){

sum+=i;

}

//当和大于250时，break语句跳出循环,

if(sum>250){

break;

}

}

//最后一次的相加和已经大于了250,在下面的输出中会体现出来

System.out.print("从1到"+i+"的偶数相加的和为:"+sum);

}

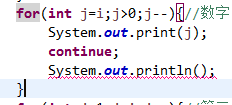
}



#### 3.5.2 continue

continue语句只能应用在for、while和do...while循环语句中，用于让程序直接跳过其后面的语句进行下一次循环。

continue关键字作用是跳出当前循环进行下次循环，整个循环不结束,这个语句离开应用范围，存在是没有意义的。这两个语句单独存在下面都不可以有语句，因为执行不到。



continue语句是结束本次循环执行下次循环，break是结束循环体进行循环后面的语句。break只能用于循环和switch语句，continue使用于循环，普通函数这两个关键字不适用。

【例5.25】打印出1到20中的所有奇数，但是13被认为是一个不吉祥的数，不打印

class SumDemo1Practice{

public static void main(String[] args){

int sum = 0 , i = 0;

while(i<=20){

i++;

if(i%2==1){

if(i==13)

continue;

System.out.print(i+" ");

}

}

}

}



#### 3.5.3 循环标号

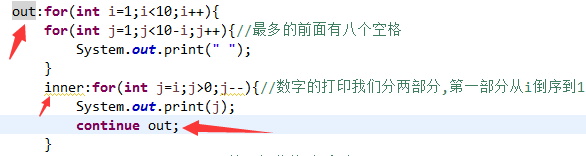
Java语言与C++有很多相似之处,这大概是由于其脱身与C++,又升华自己的缘故吧,Java的控制流程语句同C++一样,只是没有C++的goto语句，但Java同样提供了类似的功能，那就是循环标号；

break和continue跳转语句它的作用范围仅在当前循环，那么如果多层循环中我们想break跳出指定循环该怎么操作呢？这时就引出了标号的概念；标号是为循环起一个名字，或者叫做为循环做一个标记，可以通过break 标记达到直接跳出该标记循环的目的。标号的出现，可以让这两个语句(break和continue)作用于指定的范围。下面我们举例说明；

【例3.26】打印一个矩形\*；

|  |  |
| --- | --- |
| class RectanglePractice{  public static void main(String[] args){  int a = 4 , b = 5 ;  for(int i=1;i<=a;i++){  for(int j=1;j<=b;j++){  System.out.print("\*");  }  System.out.println("");  }  }  } | 打印一个矩形\*，当出现一行里超过第三个\*时，打印结束；  class RectanglePractice{  public static void main(String[] args){  int a = 4 , b = 5 ;  out:for(int i=1;i<=a;i++){  inner:for(int j=1;j<=b;j++){  System.out.print("\*");  if(j==3){  //当j==3时,它打印的是第三颗\*,此时跳出循环,但他跳出的只是内循环,  //我们如何跳出外循环呢,这时就用到了标号,为循环标号,跳出标号循环  break out;  }  }  System.out.println("");  }  }  } |
|  |  |

continue同样也是适用于标号的,如下所示:



#### 3.5.4 return

return关键字有两个功能：1是结束函数，2是返回函数的运行结果。有返回值的由return和返回内容构成返回语句，如果只有值，那么就仅仅只是停止该函数。

return 语句可以使其从当前方法中退出，返回到调用该方法的语句处，继续程序的执行。返回语句有两种格式。

* return expression: 返回一个值给调用该方法的语句，返回值的数据类型必须与方法声明中的返回值类型一致。可以使用强制类型转换来使类型一致。
* return: 当方法说明中用 void 声明返回类型为空时，应使用这种格式不返回任何值。 此时return是可以省略的。

     return语句通常用在一个方法体的最后，以退出该方法并返回一个值。在Java语言中，return语句用在一个方法体的中间时，会产生编译错误【return语句后面不能有代码】，因为这样会有一些语句执行不到。但可以通过把return语句嵌入某些语句（如 if…else,for语句等）来使程序在未执行完方法中的所有语句时退出，例如：

void method(int iVar){

if( iVar>0 ){

return; 怪蜀黍

}else{

System.out.println(iVar);

}

}

Java中的return语句使用总结   
1、return语句的作用：a、返回一个值，这个值可以是任意类型。b、使程序返回到调用该函数的地方（即终止函数运行）  
2、java中对于一个函数，不论有没有返回值类型，都可以带有return 语句。但是区别在于，return语句是否可以返回一个值（这取决于该函数的返回值类型）。  
　　a、如果函数有返回值类型（即返回值类型不为void ），则必须带有返回相应类型值的return 语句。  
　　b、如果函数没有返回值（即返回值类型为void ），则函数中return 语句（如果存在return语句！）的后面不能加任何的变量。（该情况下的函数中也可以没有return 语句，但是如果有的话，return 只能作为中断函数使用。）  
例如：

1. 有返回值

public int getAge(){

return age; //返回int 类型的变量age的值

}

2、无返回值 //函数不带return 语句

public void putAge(){

System.out.println(age);

}

3、中断函数

//函数无返回值，但是带有return语句public void put(int a) {

if (a > 0)

return; //return语句后不带返回值，作用是退出该程序的运行

else

System.out.println("fasfsa");

}

##### 注意:return和finally

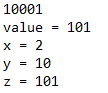
我们知道finally和try..catch语句连在一起使用,意思是不管是否有异常,finally都会执行,那么考虑一下,如果return和finally同时存在会出现什么情况,以下面代码为例来分析一下:

|  |  |
| --- | --- |
| **public** **int** finallyNotWork() {  **int** temp = 10000;  **try** {  **throw** **new** Exception();  } **catch** (Exception e) {  **return** ++temp;  }**finally** {  temp = 99999;  }  } | 左侧方法最后的返回结果是10001,而不是99999,字节码忠实地给出了答案,为什么会这样,如下: |

我们分析finally的脾气秉性,是为了避免用错,而不是深入的分析为什么JVM不支持这样的赋值方式。finally代码块的职责不在于对变量进行赋值等操作，而是清理资源、释放连接、关闭管道流等操作，此时如果有异常也要做try..catch。

相对在finally代码块中赋值，更加危险的做法是在finally块中使用return操作，这样的代码会使返回值变得非常不可控，警示代码如下：

|  |  |
| --- | --- |
| **public** **int** finallyReturn() {  **try** {  **return** ++*x*;  } **catch** (Exception e) {  **return** ++*y*;  }**finally** {  **return** ++*z*;  }  } | 如下的执行结果说明：   1. 最后return的动作是由finally代码块中的return ++z完成的，所以方法返回的结果是101； 2. 语句return ++x中的++x被成功执行，所以运行结果是x=2。 3. 如果有异常抛出，那运行结果将会是y=11.，而不是x=1 |



finally代码块中使用return语句，使返回值的判断变得复杂，所以避免返回值不可控，我们不要在finally代码块中使用return语句。

### 3.6 Dead Code

不知道有没有前辈注意过，当你编写一段“废话式的代码时”会给出一个Dead Code警告，点击警告，那么你所写的废物代码会被编译器消除，那么如果你不理睬这个警告呢？编译后会是什么样的呢？下面我们写点代码，来查看一下编译后的结果，这里使用反编译工具jd-gui.exe。代码如下：

|  |  |
| --- | --- |
| **public** **class** Test {  **public** **static** **void** main(String[] args) {  **int** a = 7;  **int** b = 9;  *test1*();  *test2*(a, b);  }  //废话代码--条件语句中，已经可以确定条件真假的会被认为是死代码（Dead Code），也叫无效代码  **public** **static** **void** test1(){  **if**(5>6){  System.*out*.println("jdk1.7输出废话代码！！");  }**else** {  System.*out*.println("都不用编译，你自己都知道要输出这段代码！");  }  }  //条件不确定的不是死代码  **public** **static** **void** test2(**int** a,**int** b){  **if**(a>b){  System.*out*.println("条件语句块中只有一句代码！");  }**else** **if** (a == b) {  System.*out*.println("条件语句块中只有两句代码！");  System.*out*.println("条件语句块中只有两句代码！");  }**else** {  System.*out*.println("会输出这句吗？");  }  }  } | 反编译  import java.io.PrintStream;  public class Test  {  public static void main(String[] args)  {  int a = 7;  int b = 9;  test1();  test2(a, b);  }    public static void test1()  {  System.out.println("都不用编译，你自己都知道要输出这段代码！");  }    public static void test2(int a, int b)  {  if (a > b)  {  System.out.println("条件语句块中只有一句代码！");  }  else if (a == b)  {  System.out.println("条件语句块中只有两句代码！");  System.out.println("条件语句块中只有两句代码！");  }  else  {  System.out.println("会输出这句吗？");  }  }  } |

上面我们知道了编译器在编译后会自动将死代码优化为已知的结果，这是代码的编译器优化.