

程序员书库

初学者的入门宝典，程序员的百科全书



CD-ROM

10小时多媒体视频讲解

本书特色

- ※ 起点低，即使没有任何编程经验，也能通过本书掌握Java
- ※ 避免大段理论讲解，而是通过大量实例进行讲解，有很强的实践性
- ※ 对代码进行了详细注释，阅读起来非常容易，没有任何障碍
- ※ 通过现实中的事物类比Java中的概念，使读者可以很容易理解
- ※ 重点讲解Java语言的基础知识和应用，并对一些设计模式也有所介绍
- ※ 全书提供190个实例和2个综合案例，非常实用

Java

从入门到精通

高宏静 等编著



化学工业出版社

第 3 章 控制流程语句

程序的运行是有顺序的，通常从上到下，从左到右，如果想改变程序的运行方式就要使用本章将要学习的控制流程语句。编程语言都是使用控制语句来执行程序的过程，进行程序状态的改变。Java 主要的控制语句有 3 种，选择语句、循环语句、跳转语句。本章将对这 3 种语句的各种形式进行详细的介绍，并在 3.5 节中举了两个例子作为程序控制语句的实例。

3.1 作用域

在介绍程序控制语句之前，首先要对块有所了解。在 Java 中，块是用一对花括号括起来的一系列语句。块定义了变量使用的范围，各个块之间可以嵌套，在块中声明的变量只有在当前块才有作用，在块外不能使用。使用块的时候要注意以下两点。

- ❑ 变量的作用范围。
- ❑ 两个嵌套块能不能声明相同名字的变量。

看下面两个具体的例子。

```
public void method()
{
    int n;
    {
        int k;
        n++;    //该语句是合法的
    }          //k 作用范围
    //k++;      //该语句非法
}              //n 作用范围
```

在上面的一个方法中有一个变量 `n`，方法中嵌套了一个块，块中有一个变量 `k`。其中变量 `n` 在整个方法中都可以访问，包括在其嵌套块中。变量 `k` 只能在其所在的最小块中使用，超出该块访问该变量的语句在编译的时候都会产生错误。例如下面的程序。

```
public void method()
{
    int n;
    {
        int k;
        //int n;    //该语句非法
    }
    int k;          //该语句是合法的
}
```

在该程序中，将 `int n` 注释掉，这样才能正常运行。如果去掉注释，由于 `n` 变量作用域的原因，该程序是会报错的。

3.2 条件语句

一周中，如果是周一到周五就要去上课或者上班，如果是周六或者周日就可以好好休息一下了。在 Java 中，遇到这种情况就需要使用条件语句。条件语句是指根据程序运行时产生的结果或者用户的输入条件执行相应的代码。在 Java 中有两种条件语句可以使用，分别是 if 条件语句和 switch 条件语句。使用它们可以根据条件来选择接下来要干什么。下面对这两种形式语句进行介绍。

3.2.1 if 条件语句

if 条件语句是最简单的条件语句，作为条件分支语句，它可以控制程序在两个不同的路径中执行。if 语句的一般形式如下。

```
if(条件)
{
    //语句块 1
}
else
{
    //语句块 2
}
```

条件可以是一个 boolean 值，也可以是一个 boolean 类型的变量，也可以是一个返回值为 boolean 类型的表达式。当需要必须执行该语句的时候，可以把条件设为 true，虽然这样做可能失去了其原来的功能，但是有时候确实需要这样。当条件为真或其值为真的时候执行语句块 1 的内容，否则执行语句块 2 的内容。示例如下。

```
int a=0;
int b=0;
.....
if(a<b)
    a=0
else
    b=0;
```

在上面的例子中，首先是声明了两个 int 类型的值 a 和 b，然后下面省略了对它们的操作。在条件语句中，如果 a 比 b 小，则把 a 的值设为 0；如果 b 的值大于 a，则把 b 的值设为 0。再看把一个 boolean 类型的变量作为条件来判断的例子。

```
boolean b;
.....
if(b)
{
    //语句块 1
}
else
{
    //语句块 2
}
```

经过对 `boolean` 类型的变量 `b` 的一系列操作，如果 `b` 的值为 `true` 则执行语句块 1 中的语句；如果 `b` 的值为 `false`，则执行语句块 2 中的语句。还有一种比较复杂的形式是把一个方法作为条件，前提是该方法能得到一个布尔值。示例如下。

```
if(b())
{
    //语句块 1
}
else
{
    //语句块 2
}
public boolean b()
{
    //方法体
}
```

在上面这一段程序中声明了一个方法，简单地说方法就是一系列操作的集合，方法可以有返回值，也可以仅仅是执行这些操作。上面的程序中声明的方法 `b` 就是一个可以得到一个布尔值为返回值的结果，也就是说执行该方法就能得到一个 `true` 值或是一个 `false` 值。上面的程序把执行方法 `b` 作为其条件，当方法 `b` 执行完后会得到一个值，如果 `b` 执行的结果为 `true`，就执行语句块 1；如果 `b` 执行的结果为 `false`，就执行语句块 2。

3.2.2 嵌套 if 条件语句

当某一情况下，无法使用一次判断选择结果时，就要使用到嵌套形式多次判断。`if` 条件语句可以嵌套使用，有一个原则是 `else` 语句总是和其最近的 `if` 语句相搭配，当然前提是这两个部分必须在一个块中。使用格式如下。

```
if(条件 1)
{
    //语句块 1
    if(条件 2)
    {
        //语句块 2
    }
    else
    {
        //语句块 3
    }
}
else
{
    //语句块 4
}
```

当条件 1 值为 `true` 的时候，会执行其下面紧跟着的花括号内的语句块，即语句块 1；如果条件 1 的值为 `false`，就会直接执行语句块 4。执行语句块 1 的时候会先判断条件 2，然后根据条件 2 的真假值情况选择执行语句块 2 还是语句块 3。

当条件有多个运行结果的时候，上面的两种形式就不能满足要求了，可以使用 `if-else` 阶

梯的形式来进行多个条件选择。格式如下。

```
if(条件 1)
{
    //语句块 1
}
else if(条件 2)
{
    //语句块 2
}
else if(条件 3)
{
    //语句块 3
}
else if(条件 4)
{
    //语句块 4
}
else
    语句块 5
```

上面的程序执行过程是首先判断条件 1 的值，如果为 `true` 的话，执行语句块 1，跳过下面的各个语句块。如果为 `false` 的话，执行条件 2 的判断，如果条件 2 的值为 `true` 的话，就会执行语句块 2，跳过下面的语句……依此类推，如果所有的四个条件都不能满足就执行语句块 5 的内容。下面的程序就是使用了这种结构进行成绩判断的，代码如下。

```
public class Demo1
{
    public static void main(String[ ] args)
    {
        //用 k 表示成绩
        int k=87;
        //用 str 存放成绩评价
        String str=null;
        if(k<0|k>100)
            str="成绩不合法";
        else if(k<60)
            str="成绩不及格";
        else if(60<k&k<75)
            str="成绩合格";
        else if(k>=75&k<85)
            str="成绩良好";
        else
            str="成绩优秀";
        System.out.println("分数: "+k+str);
    }
}
```

程序首先声明了一个 `int` 型的变量 `k` 来存放成绩，`String` 类型的变量 `str` 是用来存放对其评价的，然后通过 `if-else` 阶梯的形式来判断成绩是优秀、良好、及格还是其他，最后把成绩评定打印出来。程序的运行结果如下。

```
分数: 87 成绩优秀
```

3.2.3 switch 条件语句

上面的示例使用了 if-else 阶梯的形式进行多路分支语句的处理，但这样处理的过程太过复杂，Java 提供了一种简单的形式，若用 switch 语句来处理会使过程非常简单，格式如下。

```
switch(表达式)
{
case value1:
    //程序语句
    break;
case value2:
    //程序语句
    break;
case value3:
    //程序语句
    break;
case value4:
    //程序语句
    break;
.....
default:
    //程序语句
}
```

其中表达式必须是 byte、short、int 或者是 char 类型。在 case 后边的 value 值必须是跟表达式类型一致的类型或者是可以兼容的类型，不能出现重复的 value 值。

switch 语句的执行过程是这样的，首先它计算表达式的值，然后根据值来匹配每个 case，找到匹配的话就执行该 case 的程序语句；如果没有匹配的 case 值，就执行 default 的语句块。

执行完该 case 的语句块后，使用 break 语句跳出 switch 语句，如果没有 break 语句的话，程序会执行下一个 case 的语句块，直到碰到 break 语句为止。下面是一个将数字的汉字表达形式找出来的程序，使用 switch 结构的语句如下。

```
public class Demo2
{
    public static void main(String[] args)
    {
        int k=5;
        String str="k="+k+"的汉字形式是：";
        //switch 语句的使用
        switch (k) {
            case 1:
                str+="一";
                break;
            case 2:
                str+="二";
                break;
            case 3:
                str+="三";
                break;
            case 4:
```

```

        str+="四";
        break;
    case 5:
        str+="五";
        break;
    case 6:
        str+="六";
        break;
    case 7:
        str+="七";
        break;
    case 8:
        str+="八";
        break;
    case 9:
        str+="九";
        break;
    case 0:
        str+="零";
        break;
    default:
        str="数字超出 10";
        break;
    }
    System.out.println(str);
}

```

程序的功能是根据数字来判断其汉字表达形式，如果数字大于 10 的话就表示非法，执行 default 语句。程序的运行结果如下。

k=5 的汉字形式是：五

break 语句在 switch 中是十分重要的，一定不能省略。下面的例子是一个不正确使用 switch 语句的格式，它没有添加 break 语句，程序如下。

```

public class Demo3
{
    public static void main(String[] args)
    {
        int n=2;
        switch (n) {
            //没有 break 语句的 switch 语句，注意它的执行结果
            case 0:
                System.out.println("case0 执行");
            case 1:
                System.out.println("case1 执行");
            case 2:
                System.out.println("case2 执行");
            case 3:
                System.out.println("case3 执行");
            case 4:
                System.out.println("case4 执行");
            case 5:
                System.out.println("case5 执行");
            case 6:

```

```

        System.out.println("case6 执行");
        break;
    default:
        System.out.println("default 执行");
    }
}
}

```

程序执行的时候，首先根据 `n` 的值选择 `case 2` 来执行，但是由于一直没有 `break` 语句，程序无法跳出 `switch` 语句，会一直执行直到碰到 `case 6` 中的 `break` 语句跳出 `switch`，`default` 语句就不再执行了。程序的运行结果如下。

```

case2 执行
case3 执行
case4 执行
case5 执行
case6 执行

```

这是一个非常需要注意的地方，面试中经常会考到这个地方。

3.3 循环语句

在程序语言中，循环语句是指需要重复执行的一组语句，直到遇到让循环终止的条件为止。Java 中常用的循环有 3 种形式，`for`、`while` 和 `do-while` 循环。本节会对各种形式的循环语句进行介绍。

3.3.1 while 循环语句

各个循环语句之间的区别是不大的，但是也有本质的区别。先来看一下 `while` 循环语句，`while` 循环语句是 Java 最基本的循环语句，格式如下。

```

while(条件)
{
    //循环体
}

```

当条件为真的时候会一直执行循环体的内容，直到条件的值为假为止。其中条件可以是 `boolean` 值、`boolean` 变量、表达式，也可以是一个能获得布尔类型结果的方法。如果条件为假，则会跳过循环体执行下面的语句。下面是一个简单的示例。

```

public class Demo4
{
    public static void main(String[] args)
    {
        //定义一个 int 型变量
        int n=10;
        //使用 while 循环，条件是 n>0
        while (n>0)
        {

```



```

        System.out.println("n="+n);
        //把 n 的值减 1
        n--;
    }
}

```

在程序中有一个循环，当 n 的值大于零的时候，它会执行循环体的内容，把当前 n 的值打印出来，并对它进行自减操作。程序的运行结果如下。

```

n=10
n=9
n=8
n=7
n=6
n=5
n=4
n=3
n=2
n=1

```

在该循环中，每执行一次循环， n 的值都将减 1，当 n 的值为 0 时，“ $(n>0)$ ”这个表达式的结果就为 `false`，这时就不再执行循环，从而出现上面的结果。有时候在程序中需要一些语句一直执行，可以把条件的值直接设为 `true`，格式如下。

```

while(true)
{
    //循环体
}

```

在该程序中，循环体会不停地执行。有些读者可能会认为这样会造成无限循环，其实是不会的。在后边有的程序中会使用这种情况。

3.3.2 do-while 循环语句

`while` 语句虽然可以很好地进行循环操作，但它也是有缺陷的。如果控制 `while` 循环的条件为假的话，循环体就不会执行循环体的内容，但是有时候需要循环体至少执行一次，即使表达式为假的话也执行，这时就需要在循环末尾给出测试条件。`Java` 提供了另一种形式的循环，`do-while` 循环，它的一般格式如下。

```

do
{
    //循环体
}
while(条件)

```

`do-while` 循环首先会执行循环体，然后计算条件，如果该条件为真的话就继续执行循环体，否则就终止循环。下面用 `do-while` 循环的形式重写上一节的小程序，程序的具体实现如下。

```

public class Demo5
{
    public static void main(String[ ] args)
    {

```

```

int n=10;
//do-while 语句的使用
do {
    System.out.println("n="+n);
    n--;
} while (n>0);
}

```

程序首先执行一遍循环体，打印出 *n* 的值，然后对 *n* 进行自减操作，最后根据 *n* 值判断是否继续进行循环操作。程序的运行结果如下。

```

n=10
n=9
n=8
n=7
n=6
n=5
n=4
n=3
n=2
n=1

```

do-while 循环在处理简单菜单的时候很有用，菜单会被至少打印出来一次，然后根据后边的选择看是否会继续使用菜单。看下面的示例程序。

```

import java.io.IOException;
public class Demo6
{
    public static void main(String[] args) throws IOException
    {
        char n=0; //定义一个字符变量
        do { //使用 do-while 循环语句
            //打印出菜单
            System.out.println("1:选择 1");
            System.out.println("2:选择 2");
            System.out.println("3:选择 3");
            System.out.println("4:选择 4");
            System.out.println("5:选择 5");
            System.out.println("请输入选择: ");
            n=(char)System.in.read(); //将输入的内容转换为字符类型
            switch (n) {
                case '1': //判断用户输入的内容
                    System.out.println("选择 1");
                    break;
                case '2':
                    System.out.println("选择 2");
                    break;
                case '3':
                    System.out.println("选择 3");
                    break;
                case '4':
                    System.out.println("选择 4");
                    break;
                case '5':
                    System.out.println("选择 5");
            }
        }
    }
}

```

```

        break;
    default:
        System.out.println("输入非法");
        break;
    }

    } while (n<'1' || n>'5');           //循环的条件
}

```

在程序中首先把菜单打印出来，然后根据程序选择打印出结果。在程序中使用 `System.in.read()` 读取用户的输入，这属于后边才要讲解的内容，但是这里需要暂且先用一下。程序的运行结果如下。

```

1:选择 1
2:选择 2
3:选择 3
4:选择 4
5:选择 5
请输入选择:
3
选择 3

```

该程序不管 `while` 后面的表达式是否为 `true`，都将最少执行一次方法体。

3.3.3 for 循环语句

有时在使用 `while` 循环和 `do-while` 循环时会感觉到其功能不够强大。Java 中还提供了 `for` 循环来增强循环语句的功能。`for` 循环的一般格式如下。

```

for(初始化;条件;迭代运算)
{
    //循环体
}

```

当执行 `for` 循环，第一次先执行循环的初始化，通过它设置循环控制变量值，接下来计算条件，条件必须是一个布尔表达式，如果为真的话，就继续执行循环，否则跳出循环。然后执行的是迭代运算，通常情况下迭代运算是一个表达式，可以增加或者减小循环控制变量。最后再根据计算结果判断是否执行循环体，如此往复直到条件为假为止。下面使用 `for` 循环来计算 1 到 100 各个整数的和，程序的具体实现如下。

```

public class Demo7
//本程序用于计算 1 到 100 各个整数的和
{
    public static void main(String[ ] args)
    {
        //循环控制变量
        int n;
        //和
        int sum=0;
        //利用 for 循环求和
        for(n=100;n>0;n--)

```

```

        {
            sum+=n;
        }
        System.out.println("1 到 100 各个整数的和:"+sum);
    }
}

```

程序的运行结果如下。

1 到 100 各个整数的和:5050

一般情况下，程序控制变量只需要在程序控制的时候使用，没有必要在循环外声明它，可以把它改成如下形式。

```

public class Demo7
//本程序用于计算 1 到 100 各个整数的和
{
    public static void main(String[ ] args)
    {
        //和
        int sum=0;
        for(int n=100;n>0;n--)
        {
            sum+=n;
        }
        System.out.println("1 到 100 各个整数的和:"+sum);
    }
}

```

程序起到的作用是一样的，不过 *n* 只在循环中使用。在 Java 中支持使用多个变量来控制循环的执行，各个变量之间通过逗号隔开，可通过下面的程序来理解这种形式。

```

public class Demo8
{
    public static void main(String[ ] args)
    {
        //使用多个 int 变量来控制 for 循环
        for(int n=20,i=0;i<n;i++,n--)
            System.out.println("n="+n+" i="+i);
    }
}

```

在循环中有两个循环控制变量 *n* 和 *i*，条件是 *i* 小于 *n* 时，在迭代的过程中 *i* 自加，*n* 自减，自加自减是在一次迭代中执行的，程序的运行结果如下。

```

n=20 i=0
n=19 i=1
n=18 i=2
n=17 i=3
n=16 i=4
n=15 i=5
n=14 i=6
n=13 i=7
n=12 i=8
n=11 i=9

```

for 循环的使用是很灵活的，因为它由 3 部分控制，初始化部分、条件测试和迭代运算，使用起来都是很灵活的。下面的程序演示了几种不规范的 for 语句的使用。

```
public class Demo9
{
    public static void main(String[ ] args)
    {
        boolean b=true;
        System.out.println("循环 1");
        for(int i=0;b;i++)                //循环条件一直为 true
        {
            if(i==5)                      //当 i 的值为 5 时
                b=false;                 //改变循环条件为 false
            System.out.println("i="+i);
        }
        int i=0;
        b=true;
        System.out.println("循环 2");
        for(;b;)                          //没有起始条件
        {
            System.out.println("i="+i);
            if(i==5)
                b=false;
            i++;
        }
        System.out.println("循环 3");
        for(;;)                          //没有任何条件的 for 循环
        {
        }
    }
}
```

第一个循环中把一个 `boolean` 类型的变量作为其条件表达式。第二个循环把循环控制变量的声明放在外部，把循环迭代的语句放在了程序循环体中。第三个循环语句是空的，只有格式没有任何内容，实际上它是一个死循环，程序进入它会永远执行循环无法跳出来，直到强行终止程序，该程序才能结束。程序的运行结果如下。

```
循环 1
i=0
i=1
i=2
i=3
i=4
i=5
循环 2
i=0
i=1
i=2
i=3
i=4
i=5
循环 3
```

注意：程序一直没有执行结束。

3.4 跳转语句

跳转语句是指打破程序的正常运行，跳转到其他部分的语句。在 Java 中支持 3 种跳转语句：**break** 语句、**continue** 语句和 **return** 语句。这些语句将程序从一部分跳到程序的另一部分，这对于程序的整个流程是十分重要的。

3.4.1 break 跳出语句

break 语句主要有三种用途。第一，它可以用于跳出 **switch** 语句，前面的 **switch** 语句已经使用了该语句。第二，**break** 语句可以用于跳出循环。第三，可以用于大语句块的跳出。

1. 跳出循环

使用 **break** 语句，可以强行终止循环，即使在循环条件仍然满足的情况下也要跳出循环。使用 **break** 语句跳出循环后，循环被终止，并从循环后下一句处继续执行程序。下面是一个简单的例子。

```
public class Demo10
{
    public static void main(String[ ] args)
    {
        System.out.println("使用 break 的例子");
        for(int i=0;i<50;i++)
        {
            System.out.println("i="+i);
            //当 n 等于 10 的时候，跳出循环语句
            if(i==10)
                break;
        }
        System.out.println("循环跳出");
    }
}
```

程序的运行结果如下。

```
使用 break 的例子
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
循环跳出
```

本来循环应该执行 50 次，但是由于在循环体中，当 **i** 大于 10 的时候会执行 **break** 语句跳出循环，所以循环在执行 11 次后终止，执行其循环下面的语句。

`break` 循环仅用于跳出其所在的循环语句，如果该循环嵌入在另一个循环中，只是跳出一个循环，另一个循环还会继续执行。看下面的程序。

```
public class Demo11
{
    public static void main(String[ ] args)
    {
        System.out.println("使用 break 的例子");
        //外循环 for 语句
        for(int k=0;k<3;k++)
        {
            System.out.println("第"++k+"次外循环");
            k--;
            //内循环
            for(int i=0;i<50;i++)
            {
                System.out.println("内循环: "+"i="+i);
                if(i==3)
                    break;
            }
            System.out.println("循环跳出");
        }
    }
}
```

程序中有一个 `for` 循环，它会被执行三次，在该循环中有一个嵌套循环语句，该循环会在执行第四次的时候跳出循环。注意：为了使程序的输出结果便于理解，虽然 `k` 的值为 0、1、2，但程序在输出结果中表现为 1、2、3，使用的方式如下。

```
System.out.println("第"++k+"次外循环");
k--;
```

程序的运行结果如下。

```
使用 break 的例子
第 1 次外循环
内循环: i=0
内循环: i=1
内循环: i=2
内循环: i=3
第 2 次外循环
内循环: i=0
内循环: i=1
内循环: i=2
内循环: i=3
第 3 次外循环
内循环: i=0
内循环: i=1
内循环: i=2
内循环: i=3
循环跳出
```

2. 跳出语句块

在 Java 中，语句块是可以标记的，它的格式如下。

```
标签名:
{
    //一系列操作
}
```

下面是一段合法的程序。

```
first:{
    System.out.println("语句块 first 中");
    System.out.println("还在语句块 first 中");
}
```

在语句块中可以使用 break 语句跳出语句块，格式如下。

```
break 语句块名;
```

下面是一个完整的程序。

```
public class Demo12
{
    public static void main(String[ ] args)
    {
        //带标记的语句块
        first: {
            second: {
                third: {
                    for (int i = 0; i < 3; i++)
                    {
                        System.out.println("third:" + i);
                        //当 n 等于 2 的时候跳出语句块 second
                        if (i == 2)
                            break second;
                    }
                }
                //该语句永远不会被执行
                System.out.println("在 second 语句块中");
            }
            System.out.println("在 first 语句块中");
        }
    }
}
```

在程序中定义了 3 个语句块 first、second、third，其中 second 嵌入在 first 中，third 嵌入在 second 中，在这 3 个块中分别有一些语句需要执行。在 third 语句块中有一个循环。

```
for (int i = 0; i < 3; i++)
{
    System.out.println("third:" + i);
    if (i == 2)
        break second;
}
```

由于该循环总会执行到 i==2，当 i==2 的时候会跳出语句块 second，进入到语句块 first，所以在 second 语句块中的语句永远不会被执行。程序的运行结果如下。

```
third:0
```



```
third:1
third:2
在 first 语句块中
```

在有循环嵌套的情况下，由于 `break` 语句只能跳出其所在的最小循环，可能有时候需要跳出外循环，所以只使用 `break` 语句就不能完成这个任务。跳出语句块可以用于跳出外循环，下面的示例演示了该过程。

```
public class Demo13
{
    public static void main(String[ ] args)
    {
        //使用 out 标记语句块
        out: {
            for (int i = 0; i < 20; i++)
            {
                System.out.println("外循环" + i);
                for (int j = 0; j < 20; j++)
                {
                    System.out.println("内循环" + j);
                    if (j == 10)
                        //使用 break 语句跳出 out 语句块
                        break out;
                }
            }
        }
    }
}
```

在程序中定义了一个嵌套的 `for` 循环，本来这两个循环都需要执行 20 次（每次执行），在内循环中使用了如下的定义。

```
for (int j = 0; j < 20; j++)
{
    System.out.println("内循环" + j);
    if (j == 10)
        break out;
}
```

若 `j==10` 的话则直接跳出 `out` 语句块，这样所有的循环就不再被执行了，这样就可以终止外循环。程序的运行结果如下。

```
外循环 0
内循环 0
内循环 1
内循环 2
内循环 3
内循环 4
内循环 5
内循环 6
内循环 7
内循环 8
内循环 9
内循环 10
```

3.4.2 continue 继续语句

虽然 `break` 语句可以执行跳出循环，但是有时候要停止一次循环剩余的部分，同时还要继续执行下次循环，这时候需要使用 `continue` 语句来实现。示例程序如下。

```
public class Demo14
{
    public static void main(String[ ] args)
    {
        for (int i = 1; i < 51; i++)
        {
            System.out.print(i+" ");
            if(i%5!=0)
                //当 n 不能整除 5 的时候继续进行循环
                continue;
            else
                System.out.println("*****");
        }
    }
}
```

在程序中每五个数换一行。当 `i` 除以 5 不等于零的时候，继续执行下一次循环；当能整除的时候则换行。程序的运行结果如下。

1	2	3	4	5	*****
6	7	8	9	10	*****
11	12	13	14	15	*****
16	17	18	19	20	*****
21	22	23	24	25	*****
26	27	28	29	30	*****
31	32	33	34	35	*****
36	37	38	39	40	*****
41	42	43	44	45	*****
46	47	48	49	50	*****

`continue` 语句可以使用语句块来实现对程序各个循环的控制，示例程序如下。

```
public class Demo15
{
    public static void main(String[ ] args)
    {
        //用 out 标记外边的 for 语句块
        out:for(int i=0;i<10;i++)
            for(int j=0;j<10;j++)
                if(j>=i)
                {
                    System.out.println();
                    //用 continue 跳出当前 for 循环，执行外循环
                    continue out;
                }
            else
                System.out.print(" i="+i+" j="+j);;
    }
}
```

程序通过标签的形式来标记第一个 `for` 循环，当需要的时候会执行 `continue out` 来继续下

一次的外部循环把剩余的语句省略掉。当然程序可能还有其他更好的写法，这里只是作为一个解释 `continue` 语句的例子，理解其用法即可，程序的运行结果如下。

```
i=1 j=0
i=2 j=0 i=2 j=1
i=3 j=0 i=3 j=1 i=3 j=2
i=4 j=0 i=4 j=1 i=4 j=2 i=4 j=3
i=5 j=0 i=5 j=1 i=5 j=2 i=5 j=3 i=5 j=4
i=6 j=0 i=6 j=1 i=6 j=2 i=6 j=3 i=6 j=4 i=6 j=5
i=7 j=0 i=7 j=1 i=7 j=2 i=7 j=3 i=7 j=4 i=7 j=5 i=7 j=6
i=8 j=0 i=8 j=1 i=8 j=2 i=8 j=3 i=8 j=4 i=8 j=5 i=8 j=6 i=8 j=7
i=9 j=0 i=9 j=1 i=9 j=2 i=9 j=3 i=9 j=4 i=9 j=5 i=9 j=6 i=9 j=7 i=9 j=8
```

3.4.3 return 返回语句

还有一个跳转语句——`return` 语句，`return` 语句用于一个方法显示的返回，它把程序的控制权交给方法的调用者，该语句在方法中会经常被用到。但是由于现在还没有对方法的内容进行讲解，这里只是先举一个简单的小例子来演示其使用。

```
public class Demo16
{
    public static void main(String[ ] args)
    {
        for(int i=0;i<10;i++)
            if(i<5)
                System.out.println("第"+i+"次循环");
            else if(i==5)
                return;
            //下面的语句永远不会执行
            else
                System.out.println("第"+i+"次循环");
    }
}
```

在程序中有一个循环，当循环执行五次后就执行 `return` 语句，这时候当前方法结束。由于该方法是主方法，所以程序退出。程序运行结果如下。

```
第 0 次循环
第 1 次循环
第 2 次循环
第 3 次循环
第 4 次循环
```

3.5 程序控制语句使用实例

本节主要通过两个小程序来综合应用前面介绍的各种语句，只有学会综合使用它们才能编写出符合逻辑的优秀程序，读者可自己体会一下本节两个例子的编程方法。这两个例子一个是乘法表，另一个是螺旋矩阵，实现过程都是相对比较复杂的。

3.5.1 乘法表

九九乘法表是按照一个梯形排列的，从 1×1 到 9×9 。需要注意的是，在九九乘法表中只有 2×3 ，而没有 3×2 ，所以这里在使用嵌套循环时，就要使用一个小技巧。乘法表的程序代码如下。

```
public class Print99 {
    public static void main(String[] args) {
        System.out.println("99 乘法表");
        System.out.print(" ");
        //首先打印出第一行 1-9
        for(int i=1;i<=9;i++)
            System.out.print(i+" ");
        System.out.println();
        for(int i=1;i<=9;i++){
            //每一行打印出当前是第几行
            System.out.print(i+" ");
            for(int j=1;j<=9;j++)
                if(j<=i)
                    //打印出计算结果
                    System.out.print(i*j+"");
            //每执行完一次该循环后换行
            System.out.println();
        }
    }
}
```

程序的运行结果如下。

99 乘法表	1	2	3	4	5	6	7	8	9
1	1								
2	2	4							
3	3	6	9						
4	4	8	12	16					
5	5	10	15	20	25				
6	6	12	18	24	30	36			
7	7	14	21	28	35	42	49		
8	8	16	24	32	40	48	56	64	
9	9	18	27	36	45	54	63	72	81

3.5.2 螺旋矩阵

螺旋矩阵是指一个呈螺旋状的矩阵，它的数字由第一行开始到右边不断变大，向下变大，向左变大，向上变大，如此循环。该程序相对乘法表要更复杂一些，不过只要能一步步地按所执行的循环操作，还是比较容易理解的。螺旋矩阵的程序代码如下。

```
import java.io.*;
public class RingDemo
{
    public static void main(String[] args)
    {
        ...
    }
}
```

```

String strIn = "";
System.out.print("请输入矩阵的行数列数:");
InputStreamReader input = new InputStreamReader(System.in);
BufferedReader buff = new BufferedReader(input);
try
{
    strIn = buff.readLine();
} catch (IOException e)
{
    System.out.println(e.toString());
}
int int1 = Integer.parseInt(strIn);
int n = int1;
System.out.println("这是行列数为" + n + "的螺旋型数组:");
int intA = 1;           //初始化
int[ ][ ] array = new int[n][n];
int intB;
if (n % 2 != 0)
{
    intB = n / 2 + 1;    //奇数时 i 循环次数
} else
{
    intB = n / 2;        //偶数时 i 循环次数
}
for (int i = 0; i < intB; i++)
{
    //从外到里循环
    //从左到右横的开始
    for (int j = i; j < n - i; j++)
    {
        array[i][j] = intA;
        intA++;
    }
    //从上到下纵
    for (int k = i + 1; k < n - i; k++)
    {
        array[k][n - i - 1] = intA;
        intA++;
    }
    //从右到左横
    for (int l = n - i - 2; l >= i; l--)
    {
        array[n - i - 1][l] = intA;
        intA++;
    }
    //从下到上纵
    for (int m = n - i - 2; m > i; m--)
    {
        array[m][i] = intA;
        intA++;
    }
}
//输出数组
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        System.out.print(array[i][j] + " ");
    }
    System.out.println();
}

```

```
}  
}  
}
```

程序运行结果如下。

```
请输入矩阵的行列数:7  
这是行列数为 7 的螺旋型数组:  
1 2 3 4 5 6 7  
24 25 26 27 28 29 8  
23 40 41 42 43 30 9  
22 39 48 49 44 31 10  
21 38 47 46 45 32 11  
20 37 36 35 34 33 12  
19 18 17 16 15 14 13
```

3.6 小结

本章的主要内容是介绍 Java 的流程控制语句，流程控制语句是程序语言的灵魂，灵活地使用流程控制语句可以使程序清晰地按照要求来执行。所以读者需要好好体会各种语句的使用方法，这是编程的基础。