Chapter 3 第3章 程序控制语句

在前面章节中，已经介绍过一些简单的 Java 程序，对一些基本语法元素有了一定的讲解。本章将介绍 Java 程序中控制程序执行流程的语句。程序控制语句有 3 类：选择语句，包括 if 语句和 switch 语句；循环语句，包括 for 语句、while 语句和 do-while 语句；跳转语句，包括 break 语句、continue 语句和 return 语句。

学习目标

(1) 了解程序的控制结构。

(2) 理解选择语句执行流程，能熟练运用选择语句。了解 if 语句和 switch 语句的区别。

(3) 理解循环语句执行流程，能熟练运用循环语句。

(4) 理解循环嵌套。

(5) 了解 break 语句、continue 语句和 return 语句的控制转移过程。

3.1 控制结构

解决问题的程序的实现过程是由一系列操作组成的，这些操作之间的执行次序就是程序的控制结构。程序设计的理论和实践证明了这样的事实：任何简单或复杂的程序都可以由顺序结构、选择(分支)结构和循环结构这 3 种基本结构组合而成。同时，这些基本结构之间可以并列和互相包含，但是不允许交叉和从一个结构直接转到另一个结构的内部。所以，这 3 种结构就被称为程序设计的 3 种基本结构，也是结构化程序设计方法强调使用的 3 种基本结构。

结构化程序中的任意基本结构都具有唯一入口和唯一出口，并且程序不会出现死循环。在程序的静态形式与动态执行流程之间具有良好的对应关系。

结构化程序设计方法是最基本的程序设计方法，这种程序设计方法简单，设计出来的程序结构清晰，可读性强，容易理解，易于正确性验证，易于纠正程序中的错误，便于维护，是面向对象程序设计的基础。

所谓顺序，就是按照语句在程序中的先后次序，依次逐条执行。选择或分支则是根据条件选择执行某一条或多条语句，对应实际的情况是根据不同的情况进行不同的处理。循环是需要多次执行同一语句段，但每次执行计算的数据会有所变化。对应实际的

情况是用相同的方法处理大量不同的数据。

如果已经熟练使用 C/C++，那么掌握 Java 的控制语句将很容易。事实上，Java 的控制语句与 C/C++ 中的语句几乎完全相同。当然它们还是有一些差别的——尤其是 break 语句与 continue 语句。

3.2　选择语句

Java 语言的选择语句有两种：if 语句和 switch 语句。选择语句使程序根据条件选择执行某一条或多条语句，控制程序的执行流程。

3.2.1　if 语句

if 语句是 Java 中的条件分支语句，能将程序的执行路径分为两条。if 语句的语法格式如下：

```
if (condition) statement1;
    else statement2;
```

其中，条件 condition 可以是任何返回布尔值的表达式。statement1、statement2 都可以是单个语句，也可以是程序块。根据情况，else 子句可以省略。

if 语句的执行过程如下：如果条件为真，就执行 statement1；否则，执行 statement2。任何时候 statement1、statement2 都不可能同时执行。执行流程如图 3-1 所示。

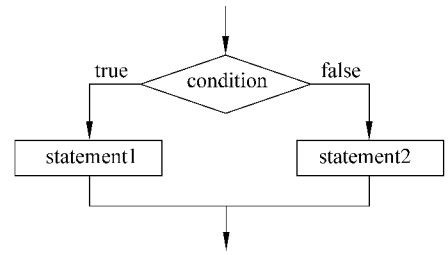

图 3-1　if 语句的执行流程

考虑下面的程序片段：

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

如果 a 小于 b，那么 a 被赋值为 0；否则，b 被赋值为 0。任何情况下都不可能使 a 和 b 都被赋值为 0。

if 语句中的 condition 通常是使用比较运算符对值或变量进行比较的表达式，也可以是使用逻辑运算符的逻辑表达式，还可以是返回值为逻辑值的函数。语句块可以是一条或多条语句。直接跟在 if 或 else 语句后的语句只能有一句。如果需要包含更多的语句，则要建一个程序块。

例 3.1　if 语句的一般用法。

```
public class IfElseDemo {
  public static void main(String[] args) {
    int number=1;
```

```
    if (number==1) {
      System.out.println("Number 等于 1");
    } else {
      System.out.println("Number 不等于  1");
    }
    if (number<2) {
      System.out.println("Number 小于 2");
    }
    if (number >=0) {
      System.out.println("Number 大于等于 0");
    }

    if(number !=0){
      System.out.println("Number 不等于 0");
    }

    Integer number2=new Integer(1);
    if (number2==1) {
     System.out.println("Number2 等于  1");
    } else {
     System.out.println("Number2 不等于  1");
    }

    if (number2.equals(1)) {
      System.out.println("Number2 等于  1");
    } else {
      System.out.println("Number2 不等于  1");
    }
  }
}
```

程序的运行结果如图 3-2 所示。

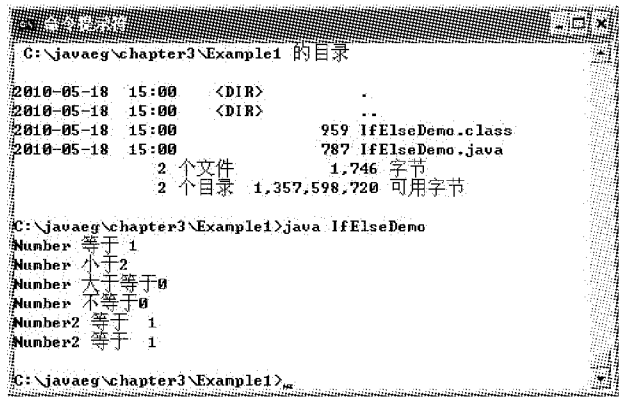

图 3-2　IfElseDemo 示例程序的运行结果

Java 中最常用的是相等的判断；对于数字还有大于或小于的判断；而对于 Integer 对象类型则使用 equals 来判断。

很多时候，会遇到多分支的情况。if 语句也可以用来处理多重分支，此时需要使用嵌套 if 语句。嵌套 if 语句是指该 if 语句为另一个 if 或者 else 语句的执行目标。在使用嵌套 if 语句时，需注意的要点是：一个 else 语句总是对应着和它在同一个块中的最近的 if 语句，而且该 if 语句没有与其他 else 语句相关联。基于嵌套 if 语句的通用编程结构被称为 if-else-if 阶梯。它的语法格式如下：

```
if(condition1)
    statement1;
else if(condition2)
    statement2;
else if(condition3)
    statement3;
⋮
else
    statementn+1;
```

执行流程如图 3-3 所示。

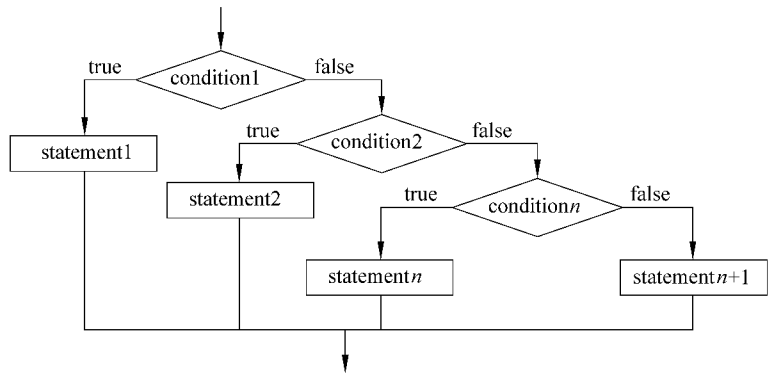

图 3-3　if-else-if 阶梯的执行流程

若干个条件表达式从上到下被求值。一旦找到为真的条件，就执行与它关联的语句，而该阶梯的其他部分就被忽略了。如果所有的条件都不为真，则执行最后的 else 语句。最后的 else 语句经常被作为默认的条件，即如果所有其他条件都不满足，就执行最后的 else 语句。如果没有最后的 else 语句，而且所有其他的条件都失败，那程序就不做任何动作。

例 3.2　通过使用 if-else-if 阶梯来确定某个月是什么季节。

```
class IfElseSeasonDemo {
    public static void main(String args[]) {
        int month=4;                                    //April
        String season;
```

```
        if(month==12||month==1||month==2)
            season="Winter";
        else if(month==3||month==4||month==5)
            season="Spring";
        else if(month==6||month==7||month==8)
            season="Summer";
        else if(month==9||month==10||month==11)
            season="Autumn";
        else
            season="Bogus Month";
        System.out.println("April is in the"+season+".");
    }
}
```

程序的运行结果如图 3-4 所示。

图 3-4 IfElseSeasonDemo 示例程序的运行结果

无论给 month 什么值，该阶梯中有而且只有一个语句执行。

3.2.2 switch 语句

在多重分支的实现中，通常使用 switch 语句。switch 语句提供了更为清晰的多重分支结构，可以使代码更加简练易读。switch 语句提供了一种基于一个表达式的值来使程序执行不同部分的简单方法。因此，它提供了一个比一系列 if-else-if 语句更好的选择。switch 语句的通用形式如下：

```
switch (expression) {
  case value1:
    statement sequence1
    break;
  case value2:
    statement sequence2
    break;
  ⋮
  case valueN:
    statement sequencen
```

```
    break;
  default:
    statement sequencen+1
}
```

表达式 expression 必须为 byte、short、int 或 char 类型。每个 case 语句后的值 value 必须是与表达式类型兼容的特定的一个常量(它必须为一个常量,而不是变量)。重复的 case 值是不允许的。

switch 语句的执行过程如下:表达式的值与每个 case 语句中的常量作比较,如果发现了一个与之相匹配的,则执行该 case 语句后的代码,如果没有一个 case 常量与表达式的值相匹配,则执行 default 语句;当然,default 语句是可选的,如果没有相匹配的 case 语句,也没有 default 语句,则什么也不执行。执行流程如图 3-5 所示。

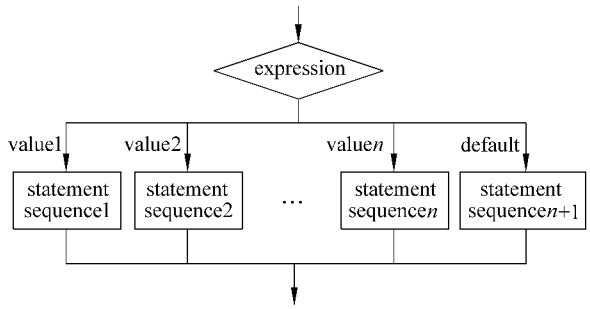

图 3-5　switch 语句的执行流程

在 case 语句序列中的 break 语句使程序从 switch 语句退出,执行 switch 语句后的语句块。当执行 break 语句时,程序从 switch 语句后的第一行代码开始继续执行。这有一种“跳出”switch 语句的效果。

例 3.3　使用 switch 语句的简单例子。

```
class SwitchDemo {
    public static void main(String args[]) {
        for(int i=0; i<6; i++)
            switch(i) {
            case 0:
                System.out.println("i is zero.");
            break;
            case 1:
                System.out.println("i is one.");
            break;
            case 2:
                System.out.println("i is two.");
            break;
            case 3:
                System.out.println("i is three.");
```

```
            break;
            default:
                System.out.println("i is greater than 3.");
            }
        }
}
```

程序的运行结果如图 3-6 所示。

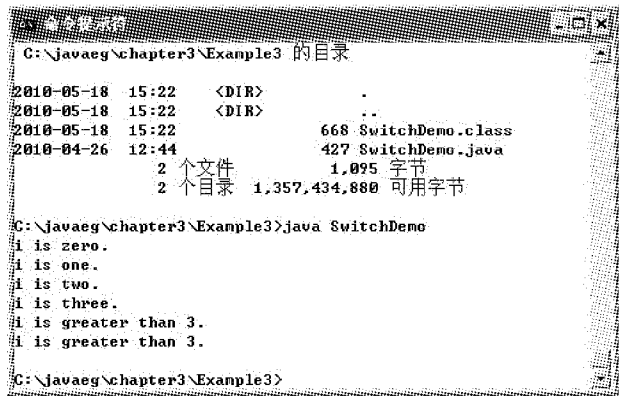

图 3-6 SwitchDemo 示例程序的运行结果

示例中，每循环一次，与 i 值匹配的 case 常量后的相关语句被执行，其他语句则被忽略。当 i 大于 3 时，没有可以匹配的 case 常量，因此执行 default 后的语句。break 语句是可选的。根据需要，break 语句可以省略，这样，程序将继续执行下一个 case 语句。

例 3.4 省略了部分 break 语句。

```
class SwitchSeasonDemo {
    public static void main(String args[]) {
        int month=4;
        String season;
        switch (month) {
        case 12:
        case 1:
        case 2:
            season="Winter";
        break;
        case 3:
        case 4:
        case 5:
            season="Spring";
        break;
        case 6:
        case 7:
        case 8:
```

```
            season="Summer";
        break;
        case 9:
        case 10:
        case 11:
            season="Autumn";
        break;
        default:
            season="Bogus Month";
        }
        System.out.println("April is in the"+season+".");
    }
}
```

程序的运行结果如图 3-7 所示。

图 3-7　SwitchSeasonDemo 示例程序的运行结果

该程序演示如果没有 break 语句，程序将继续执行下面的每一个 case 语句，直到遇到 break 语句，或者 switch 语句的末尾。

switch 语句有以下 3 个重要的特性需注意。

(1) switch 语句仅能测试相等的情况，而 if 语句可计算任何类型的布尔表达式。

(2) 在同一个 switch 语句中没有两个相同的 case 常量。

(3) switch 语句通常比一系列嵌套 if 语句更有效。

当 Java 编译器编译一个 switch 语句时，Java 编译器将检查每个 case 常量并且建立一个“跳转表”，这个表将用来在表达式值的基础上选择执行路径。因此，如果需要在一组值中做出选择，switch 语句将比与之等效的 if-else 语句效率要高。

3.3　循环语句

在编写 Java 程序时，往往会对大量不同的数据使用相同方法进行处理，如对班级内的每位同学的各科成绩计算平均分。在这种情况下，计算平均分的操作可以用一组语句编写，重复多次执行该组语句，每次计算出一位同学的平均分。这种多次重复执行同一组语句的过程称为循环。在 Java 语言中，为了满足不同的需要，提供了如下多条实现循

环的语句。

(1) while 语句；

(2) for 语句；

(3) do-while 语句。

在循环语句中，不能无限次地重复执行某组语句。无限次地重复执行某组语句的循环称为死循环，包含死循环的程序永远不能结束。所以，循环语句都有结束条件，如条件为 true 或 false 时结束循环，或者指定循环的次数等。

3.3.1 while 语句

while 语句是 Java 语言最基本的循环语句。当它的条件(即控制表达式)为真时，while 语句重复执行一个语句或语句块。while 语句的语法格式如下：

```
while(condition) {
    //body of loop
}
```

条件 condition 可以是任何布尔表达式。只要条件表达式为真，循环体就被执行。当条件 condition 为假时，程序控制就传递到循环后面紧跟的语句行。如果只有单个语句需要重复，大括号是不必要的。执行流程如图 3-8 所示。

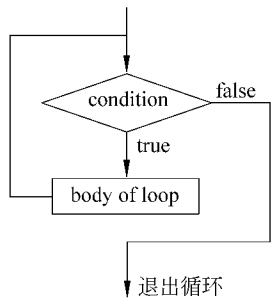

图 3-8 while 语句的执行流程

例 3.5 从 10 开始进行减计数，打印出 10 行 tick。

```
class WhileDemo {
    public static void main(String args[]) {
        int n=10;
        while(n>0) {
            System.out.println("tick"+n);
            n--;
        }
    }
}
```

程序的运行结果如图 3-9 所示。

因为 while 语句在循环体执行前判断条件，计算条件表达式。若条件为假，则循环体一次也不会执行。如下面的程序片段，循环体从未执行过。

```
int a=10, b=20;
while(a>b)
    System.out.println("This will not be displayed");
```

while 循环(或 Java 的其他任何循环语句)的循环体可以为空。仅由一个分号组成的空语句(null statement)在 Java 的语法上是合法的。

图 3-9　WhileDemo 示例程序的运行结果

例 3.6　循环体为空。

```
class NoBodyofLoopDemo {
    public static void main(String args[]) {
        int i, j;
        i=10;
        j=50;
        while(++i<--j) ;
        System.out.println("Midpoint is"+i);
    }
}
```

该示例程序的功能是找出变量 i 和变量 j 的中间点。程序的运行结果如图 3-10 所示。

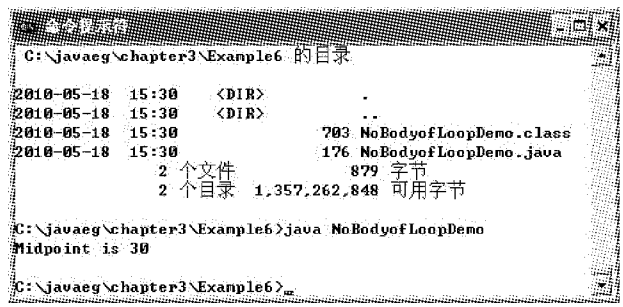

图 3-10　NoBodyofLoopDemo 示例程序的运行结果

程序中的 while 循环的条件判断是先值 i 自增，值 j 自减，然后比较这两个值。如果新的值 i 仍比新的值 j 小，则进行循环。如果 i 等于或大于 j，则循环停止。在退出循环前，i 将保存原始 i 和 j 的中间值。在专业化的 Java 代码中，一些可以由控制表达式本身