

第3章

Java 语句及其控制结构

任何 Java 程序都是由类和对象组成的，对象和类是由变量与方法组成的，变量由声明变量语句与赋值语句构成，方法由一系列执行不同功能的 Java 语句构成。Java 语句有不同的类型，不同类型的语句其组成成分不同、含义不同、作用不同，有的语句在程序中的位置顺序也有一定的规则，这些也是 Java 语言的基本语法内容。学习 Java 程序设计，就是要了解不同 Java 语句的构成方式、含义与作用，了解如何组织 Java 语句构建 Java 类与对象的框架，定义变量确定类与对象的特性，定义方法实现与控制类的功能。

本章主要解决以下问题：

- Java 程序有哪些主要构成成分？
- Java 类有哪些主要构成成分？
- Java 有哪些类型的语句？
- Java 的选择语句是如何构成的，其作用是什么？
- Java 的循环语句是如何构成的，其作用是什么？
- Java 的跳转语句是如何构成的，其作用是什么？

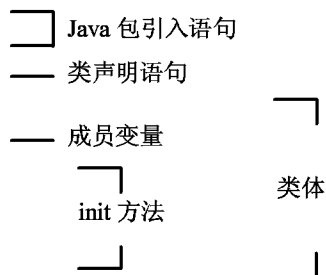
3.1 Java 语句的类型

本节主要介绍 Java 程序的构成成分、类的构成成分、Java 语句的种类以及说明性语句、表达式语句和复合语句的特点。

3.1.1 Java 程序构成

例 3.1 一个程序范例，用来说明 Java 程序的构成成分。

```
import java.applet.Applet;  
import java.awt.*;  
public class GetSquare extends Applet  
{  
    Label label1;  
    public void init() {  
        label1=new Label("前 30 个数的平方");  
        add(label1);  
    }  
}
```



```

public void paint(Graphics g) {
    for (int i=0; i<30; i++) {
        int x=i%10, y=i/10;
        g.drawString(String.valueOf((i+1) * (i+1)), 循环
            20+30*x, 50+25*y);
    }
}

```

paint 方法 类体

1. 程序的主要构成

从例 3.1 可以看到，Java 程序一般包括 Java 包引入（如果有的话）语句、类定义两部分。类定义由类声明语句和类体组成。

2. 主类

Java 程序中，必须含有一个可被外界（通常是 Java 解释器）所直接调用的类，这个类称为这个 Java 程序的主类，即可以被运行的类。主类常用 **public** 关键字来修饰。一个 Java 程序中可以定义多个类，但只能有一个是主类，Java Applet 程序的主类是继承 Applet 的类，如例 3.1 中的 GetSquare 就是主类。Java Application 程序的主类是包含 main 方法的类，整个应用程序从 main 方法开始执行。

3. 类的构成

1) 类声明语句与类体

类简单说是由类声明语句与类体两部分构成的。类声明语句用来说明类的名称、访问权限及类的属性。类体由成员变量和成员方法两部分组成。在例 3.1 的 GetSquare 类中，label1 是添加的成员变量，init 和 paint 是继承下来的成员方法。

2) 成员变量

成员变量为类中定义的变量（又称属性、域），用来说明类的状态和特性。定义成员变量需要声明成员变量的名称、类型或初值。

3) 成员方法

成员方法为类中的方法，用来实现类的功能和行为，是程序设计的关键。Java Application 程序中一定要有 main 主方法，以控制程序进行数据初始化工作，实现不同的功能或调用其他对象、方法完成不同的功能，输出程序运行后的结果。而 Java Applet 程序一般要有 init() 初始化方法，以确定界面的初始状态，由 paint(Graphics g) 画出方法，以展示程序运行后的结果。

方法类似于其他程序设计语言中的函数，可以调用，可以有返回值，可以随意设计其功能。成员方法可以继承父类已有的方法，也可以自定义成员方法。根据需要，方法中可以定义局部变量，但更重要的是组织方法中的语句结构，以实现不同的功能。

3.1.2 Java 语句的种类

Java 语句是 Java 标识符的集合，由关键字、常量、变量和表达式构成。简单的 Java

语句以分号“;”作为结束标志，单独的一个分号被看做一个空语句，空语句不做任何事情。复合结构的 Java 语句以大括号“{”作为结束标志。

Java 语句一般分为说明性语句和操作性语句两种类型。

1. 说明性语句

Java 的说明性语句包含包和类引入语句、声明类语句、声明变量语句、声明对象语句等。例如：

```
import java.applet.Applet;           //包引入语句
public class GetSquare extends Applet; //声明类语句
```

2. 操作性语句

Java 的操作性语句包含表达式语句、复合语句、选择语句和循环语句、跳转语句等。Java 规定所有的操作性语句必须放在成员方法中。

下面介绍表达式语句与复合语句的构成，其他操作性语句将分别在后面用一节介绍。

1) 表达式语句

在表达式后边加上分号“;”就是一个表达式语句。表达式语句是最简单的语句，它们被顺序执行，完成相应的操作任务。表达式语句主要有赋值语句和方法调用语句。

例如：

```
int k, i=3, j=4;           //声明变量语句
k=i+j;                     //赋值语句
System.out.println("k="+k); //方法调用语句
```

注意：声明变量时可以直接赋初值，但不属于赋值语句，为声明变量语句。

2) 复合语句

复合语句也称为块（block）语句，是包含在一对大括号“{”中的任意语句序列。与其他语句用分号作结束符不同，复合语句右括号“}”后面不需要分号。尽管复合语句含有任意多个语句，但从语法上讲，一个复合语句被看做一个简单语句。

例 3.2 复合语句示例。

```
class Block
{
    public static void main(String args[])
    {
        int k, i=3, j=4;
        k=i+j;
        System.out.println("k="+k);
        {
            float f;
            f=j+4.5F;
            i++;
            System.out.println("f="+f);
        }
        System.out.println("i="+i);
    }
}
```

```
}  
}
```

从例 3.2 中可以看出, 在 `main` 方法中有两个复合语句嵌套在一起, 复合语句内包含的是表达式语句。第一个复合语句中声明的 3 个整型变量 `k`、`i`、`j`, 不仅在第一个复合语句中起作用, 还在被嵌套的第二个复合语句中起作用。而在第二个复合语句中声明的变量 `f` 仅在第二个复合语句中起作用。

在例 3.2 中, 人为地加入了一个复合语句, 在实际编程中并不多见, 只是为了说明复合语句的用法。复合语句更广泛地应用在结构式语句中, 如选择语句和循环语句。当结构中包含的表达式语句超过一条时, 就要用大括号把它们括起来。

3.2 选择语句

表达式语句与复合语句都是按顺序从上到下逐行执行每条命令。能否改变程序中语句执行的顺序呢? 利用选择语句结构就可以根据条件控制程序流程, 改变语句执行的顺序。

本节的内容主要介绍 Java 的 4 种选择语句: 单分支选择语句 (`if` 语句)、二分支选择语句 (`if...else` 语句)、多分支选择语句 (`if...else if...else` 语句) 和先执行后判定循环 (`do...while`) 的使用方法。

3.2.1 单分支选择语句 (`if` 语句)

`if` 语句的语法格式为:

```
if (条件) s1 语句;
```

这是最简单的单分支结构。当构成条件的逻辑或关系表达式的值为 `true` 时, 执行 `s1` 语句, 否则忽略 `s1` 语句。`s1` 语句可以是复合语句。

3.2.2 二分支选择语句 (`if...else` 语句)

`if` 语句通常与 `else` 语句配套使用, 形成二分支结构。它的语法格式为:

```
if (条件) s1 语句;  
else s2 语句;
```

当构成条件的逻辑或关系表达式的值为 `true` 时, 执行 `s1` 语句, 忽略 `else` 和 `s2` 语句; 否则, 条件表达式的值为 `false`, 程序忽略 `s1` 语句, 执行 `else` 后面的 `s2` 语句。`s1` 和 `s2` 都可以是复合语句。

例 3.3 比较两个数的大小并按升序输出, 运行结果如图 3.1 所示。

```
class C1 {  
    public static void main(String args[]) {  
        double d1=23.4, d2=35.1;  
        if(d2>=d1)  
        {   System.out.println(d1);  
            System.out.println(d2);  
        }  
    }  
}
```

```
----- 运行 -----  
23.4  
35.1
```

图 3.1 升序排列数据

```

else
    {System.out.println(d2);
    System.out.println(d1);}
}
}

```

3.2.3 多分支选择语句（if…else if…else 语句）

对于超过二分支选择的情况，可以使用多分支选择语句（if…else if…else 语句）。它的语法格式为：

```

if (条件 1) s1 语句;
else if (条件 2) s2 语句;
else if (条件 3) s3 语句;
else s4 语句;

```

在这里依次计算条件的值，如果某个条件的值为 **true**，就执行它后面的语句，其余语句被忽略；所有条件的值都为 **false**，就执行最后一个 **else** 后的 **s4** 语句。**s1**、**s2**、**s3** 和 **s4** 都可以是复合语句。

例3.4 下面是一个嵌套使用 if…else 语句与 if…else if…else 语句构造多分支程序的例子，判断某一年是否为闰年。闰年的条件是符合下面二者之一：能被 4 整除，但不能被 400 整除；能被 4 整除，又能被 100 整除。输出结果如图 3.2 所示。

```

public class LeapYear {
    public static void main(String args[]) {
        boolean leap;    int year=2008;
        //方法 1
        if ((year%4==0 && year%100!=0) || (year%400==0)) System.out.println
(year+" 年是闰年");
        else System.out.println(year+" 年不是闰年");
        //方法 2
        year=2020;
        if (year%4!=0) leap=false;
        else if (year%100!=0) leap=true;
        else if (year%400!=0) leap=false;
        else leap=true;
        if (leap==true) System.out.println(year+" 年是闰年");
        else System.out.println(year+" 年不是闰年");
        //方法 3
        year=2050;
        if (year%4==0) {
            if (year%100==0) {
                if (year%400==0) leap=true;
                else leap=false;
            }
            else leap=false;
        }
        else leap=false;
        if (leap==true) System.out.println(year+" 年是闰年");
    }
}

```

运行
2008 年是闰年
2020 年是闰年
2050 年不是闰年

图 3.2 判别是否为闰年

```

        else System.out.println(year+" 年不是闰年");
    }
}

```

说明:

方法 1 用一个逻辑表达式包含了所有的闰年条件;方法 2 使用了多分支选择语句(if…else if…else 语句);方法 3 通过大括号“{}”对 if…else 进行匹配来实现闰年的判断。大家可以根据程序对比这 3 种方法,体会其中的联系和区别,在不同的场合选用适当的方法。

3.2.4 开关语句 (switch 语句)

可以看出虽然嵌套的条件语句可实现多个分支处理,但嵌套太多时容易出错和混乱,这时可以使用开关语句 switch 处理。使用它可以容易地写出判断条件,特别是有很多条件选项的时候。

开关语句 switch 的语法格式为:

```

switch (条件) {
    case 常量 1:
        语句 1;
        break;
    case 常量 2:
        语句 2;
        break;
    ...
    default:
        语句 n;
}

```

其中 switch、case、default 是关键字, default 子句可以省略。开关语句先计算条件的值,然后将条件值与各个常量比较,如果条件值与某个常量相等,就执行该常量后面的语句。如果都不相等,就执行 default 下面的语句。如果无 default 子句,就什么都不执行,直接跳出开关语句。

使用开关语句时,注意以下两点。

- case 后面的常量必须是整数或字符型,而且不能有相同的值。
- 通常在每一个 case 中都通过 break 语句提供一个出口,使流程跳出开关语句。否则,在第一个满足条件的 case 后面的所有语句都会被执行,这种情况叫做落空。看下面分别加上和去掉 break 语句的例子。

例 3.5 switch 语句的使用,有 break 语句。本程序当温度变量 c 小于 10℃时,显示“有点冷”,c 小于 25℃时,显示“正合适”,c 小于 35℃时,显示“有点热”,c 大于 35℃时,显示“太热了”,输出结果为“28℃有点热”。

```

public class W1 {
    public static void main(String args[]) {
        int c=28;
        switch (c<10?1:c<25?2:c<35?3:4) {
            case 1:
                System.out.println(" "+c+"℃ 有点冷");

```

```

        break;
    case 2:
        System.out.println(" "+c+"°C 正合适");
        break;
    case 3:
        System.out.println(" "+c+"°C 有点热");
        break;
    default:
        System.out.println(" "+c+"°C 太热了");
    }
}
}
}

```

例 3.6 switch 语句的使用，无 break 语句。输出结果如图 3.3 所示。

```

public class W2 {
    public static void main(String args[]) {
        int c=28;
        switch (c<10?1:c<25?2:c<35?3:4) {
            case 1:
                System.out.println(" "+c+"°C 有点冷");
            case 2:
                System.out.println(" "+c+"°C 正合适");
            case 3:
                System.out.println(" "+c+"°C 有点热");
            default:
                System.out.println(" "+c+"°C 太热了");
        }
    }
}

```

```

----- 运行 -----
28°C 有点热
28°C 太热了

```

图 3.3 无 break 语句的结果

说明：

通过这两个例子可以看出 break 语句的作用。例 3.6 由于缺少 break 语句，使得程序执行完 case 3 下面的语句，紧接着又执行了 default 下面的语句。

因为 case 后面的常量必须是整数或字符型。这两个程序采用了转换方法，将判断条件的取值最终转换为数值，请读者自行分析三元运算符（`c<10?1:c<25?2:c<35?3:4`）的作用。

3.3 循环语句

到目前为止，我们看到的都是线性的程序，即每行命令只有一次执行的机会（选择语句则会忽略若干行）。能否重复执行一些语句呢？利用循环语句可以解决这个问题。循环可使程序根据一定的条件重复执行某一部分程序语句，直到满足终止循环条件为止。

本节的内容主要介绍 Java 的 3 种循环语句：确定次数循环（for）、先判定后执行循环（while）和先执行后判定循环（do...while）的使用方法。

3.3.1 确定次数循环语句（for 循环）

如果希望程序的一部分内容按固定的次数重复执行，可使用 for 循环。for 循环采用一

个计数器控制循环次数，每循环一次计数器就加 1，直到完成给定的循环次数为止。

例 3.7 该程序利用 for 循环语句为一维数组中的每个元素赋值，然后按逆序输出，结果如图 3.4 所示。

```
public class 例3_7 {
    public static void main(String args[]) {
        int i;
        int a[]=new int[5];
        for (i=0; i<5; i++) a[i]=i;
        for (i=a.length-1; i>=0; i--) System.out.println("a["+i+"] = "+a[i]);
    }
}
```

```
----- 运行 -----
a[4] = 4
a[3] = 3
a[2] = 2
a[1] = 1
a[0] = 0
```

图 3.4 逆序输出数组

----- 运行 -----	
摄氏度	华氏度
0	32
5	41
10	50
15	59
20	68
25	77
30	86
35	95
40	104

图 3.5 摄氏度到华氏度的转换表

例 3.8 按 5 度的增量打印出一个从摄氏度到华氏度的转换表，输出结果如图 3.5 所示。

```
class CtoF {
    public static void main (String args[]) {
        int fahr,cels;
        System.out.println("摄氏度 华氏度");

        for (cels=0; cels<=40; cels+=5) {
            fahr=cels*9/5+32;
            System.out.println(" "+cels+" "+fahr);
        }
    }
}
```

从上面的例子中归纳出 for 循环的语法格式为：

```
for (表达式 1; 表达式 2; 表达式 3)
    循环体
```

其中表达式 1 指出计数器的初值，是一个赋值语句；表达式 2 指出循环结束条件，是一个逻辑表达式；表达式 3 指出计数器每次的增量，是一个赋值语句。

说明：

计数器可在 for 语句之前定义，也可在循环括号中定义。计数器增量为 1 时常写成增量运算的形式，以加快运算速度。根据需要，增量可以大于 1。增量计算也可以放在循环体中进行，即把表达式 3 移到循环体内的适当位置，原位置为空。

使用循环语句时常常会遇到死循环的情况，就是无限制地循环下去。所以在使用 for 循环时，要注意初值、终值和增量的搭配。终值大于初值时，增量应为正值，终值小于初值时，增量应为负值。编程时必须密切关注计数器的改变，这是实现正常循环避免陷入死循环的关键。

3.3.2 先判定后执行循环语句（while 循环）

while 循环不像 for 循环那么复杂，while 循环只要定义一个条件判断语句，便可以进行循环操作，看下面的例子。

例 3.9 这个程序可进行人机交互，从键盘输入数字 1、2、3，可显示得到的奖品；如果输入其他数字或字符显示“没有奖品给你！”。其中使用了开关语句和 while 循环语句。输出结果如图 3.6 所示。这个程序需要在“命令提示符”窗口中运行，这样才能进行人机交互。

```
import java.io.*;
class G1 {
    public static void main(String args[]) throws IOException {
        char ch;
        System.out.println("按 1/2/3 数字键可得大奖!");
        System.out.println("按空格键后再按回车键可退出循环操作.");

        while ((ch=(char)System.in.read())!=' ') {
            System.in.skip(2);      //跳过回车键
            switch (ch) {
                case '1':
                    System.out.println("你得到一辆车!");
                    break;
                case '2':
                    System.out.println("你得到一台彩电!");
                    break;
                case '3':
                    System.out.println("你得到一台冰箱!");
                    break;
                default:
                    System.out.println("没有奖品给你!");
            }
        }
    }
}
```



图 3.6 交互结果

说明:

while 循环从键盘读入字符直至输入一个空格并按回车键后退出。程序里引用了 java.io 程序包，使用了系统的输入方法 System.in.read() 和 throws IOException 抛出异常，有关 Java 的异常处理机制将在第 6 章进行介绍。

循环内部的第一条语句（System.in.skip(2);）处理回车键，将每次输入时按下的回车键

屏蔽掉。switch 语句先把从键盘读入的字符和给定字符比较，当发现相等时，就显示得到的奖品名称。然后经由 break 语句跳出 switch 语句，程序回到等待键盘输入的状态，进行下一轮循环。

从上面的例子归纳出 while 循环的语法格式为：

```
while (条件)
    循环体
```

其中 while 是关键字。每次循环之前都要计算条件的值，其值为 true 时，执行一次循环体中的语句，然后再计算条件，决定是否再次执行循环体中的语句；如果条件的值为 false，跳出循环体，执行循环体下面的语句。

注意：while 循环中的条件是逻辑表达式或关系表达式，所以循环体中一定要有改变条件的语句，使条件的值变为 false，否则会陷入死循环。

3.3.3 先执行后判定循环语句（do...while 循环）

do...while 循环与 while 循环相反，是先执行循环体中的语句，再计算 while 后面的条件，若条件值为 false 则跳出循环，否则继续下一轮循环。

什么时候使用 do...while 循环呢？有些情况下，不管条件的值是 true 还是 false，都希望把指定的语句至少执行一次，那么就应使用 do...while 循环，看下面的例子。

例 3.10 求 $1+2+\dots+100$ 之和，输出结果为 “ $1+2+\dots+100=5050$ ”。

```
class Sum {
    public static void main(String args[]) {
        int n=1; int sum=0;
        do sum+=n++;
        while (n<=100);
        System.out.println("1+2+...+100 =" +sum);
    }
}
```

归纳 do...while 循环的语法格式为：

```
do { 循环体 }
while (条件);
```

其中 do、while 是关键字。程序首先执行 do 下面的循环体，然后计算 while 后面条件的值，如果其值为 true，则重复执行循环体，否则，结束循环。

与 while 循环相同，do 循环在循环体中也一定要有改变条件值为 false 的语句，否则会陷入死循环。do...while 循环控制并不是很常用，但有时却非常重要，使用时特别注意不要忘记了 while 语句结尾处的分号 “;”。

3.3.4 嵌套使用循环语句

在实际使用中，上面介绍的循环语句可嵌套使用，如例 3.11 所示。

例 3.11 分别用 while、do 和 for 循环语句实现累计求和。输出结果如图 3.7 所示。