第5章

方　　法

结构化语言(如 C 语言)实现软件系统模块化的最小单位是函数(方法),软件系统由很多函数构成,函数之间存在复杂的调用关系。Java 作为纯面向对象语言,实现软件系统模块化的最小单位是类,软件系统由很多类构成类之间存在继承、依赖、关联等关系。Java 程序中是否还需要方法?Java 方法的定义与使用是否与 C 等结构化语言一样?作为一门优秀的程序设计语言,Java 方法是否有新特征?本章将为读者一一解答这些问题。

本章内容

(1) 传统方法。

(2) 形参长度可变方法。

(3) 方法重载。

(4) 递归方法。

方法定义及调用

5.1 传统方法

5.1.1 方法的概念

方法是一段可重复调用的代码块。软件开发中,把完成相同功能的代码块定义为方法能提高软件开发效率以及软件可维护性。例如,学生成绩管理系统中,任课教师需要对课程成绩排序,学生辅导员和教学秘书也需要对课程成绩排序。如果 3 个子系统分别编写学生成绩排序代码会显得非常烦琐,并且修改也比较困难,如果改进了排序算法,需要在 3 个不同位置同时修改。如果把成绩排序功能定义成一个方法,只要在 3 个不同位置调用即可。

第一种方式,如图 5-1(a)所示,改进成绩排序算法,需要修改 3 个位置的代码块;第二种方式,如图 5-1(b)所示,仅需要修改一个位置的代码块。

有些书把方法称为函数,与本书的方法是相同概念,只是称呼方式不一样。

5.1.2 定义及调用传统方法

Java 提供了定义方法的多种形式,定义传统方法需要确定返回值类型、方法名和参数等三要素。

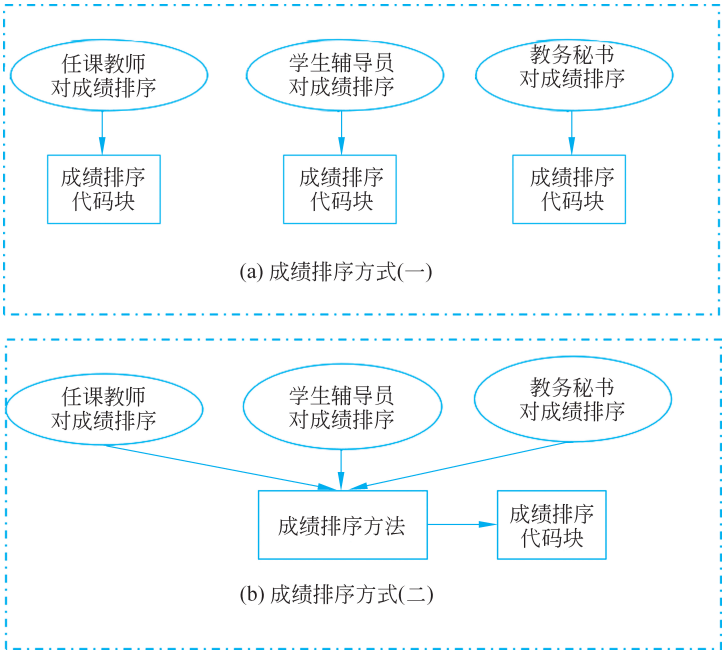

图 5-1　教务管理系统成绩排序的两种实现方式

【语法格式 5-1】　定义传统方法。

```
public static 返回值类型 方法名(类型 形参 1,类型 形参 2,…){
    方法体;                                    //方法主体
    [return 表达式];                           //方法返回值
}
```


方法定义及调用程序

上面的语法格式中,方法声明前增加了 public static 关键字,表示该方法能直接被 main() 方法调用。

Java 调用传统方法的形式与 C 语言一样。

【语法格式 5-2】　调用传统方法。

```
方法名(实参 1,实参 2,…)
```

下面代码段,第 2 行定义方法 max(int x,int y)返回两个整数的较大者,第 6 行调用 max(20,30)方法。

```
1   public class Demo {
2       public static int max(int x,int y) {        //定义方法 max()
3           return x>y?x:y;
4       }
5       public static void main(String[] args) {
6           System.out.println(max(20,30)           //调用方法 max();
7       }
8   }
```

程序案例 5-1 演示了传统方法的定义与调用。第 5～17 行定义了方法,该方法返回值

类型 void,方法名 printBook(),形参 String 类型。第 20～25 行定义了方法,该方法返回值类型 double,方法名 total(),形参是 double 类型的一维数组。第 27 行调用方法 printBook(),实参字符串"老子",第 29 行调用方法 total(),实参是 double 类型一维数组 score。程序运行结果如图 5-2 所示。

图 5-2 程序案例 5-1 运行结果

【程序案例 5-1】 定义及调用传统方法。

```
1   package chapter05;
2   public class Demo0501 {
3       //==========自定义传统方法 1==========
4       //返回值类型,方法名,形参
5       public static void printBook(String name) {
6           if (name.equals("孔子")) {
7               System.out.println("孔子的代表作《论语》");
8               return;
9           } else if (name.equals("老子")) {
10              System.out.println("老子的代表作《道德经》");
11              return;
12          } else if (name.equals("司马光")) {
13              System.out.println("司马光的代表作《资治通鉴》");
14          } else {
15              System.out.println("数据库中不存在 " + name + " 的作品");
16          }
17      }
18      //==========自定义传统方法 2==========
19      //返回值类型,方法名,形参
20      public static double total(double[] arr) {          //方法签名
21          double sum = 0;
22          for (double d : arr)                            //foreach 语句
23              sum = sum + d;
24          return sum;
25      }
26      public static void main(String[] args) {
27          printBook("老子");                              //调用方法 printBook()
28          double[] score = { 88.5, 92.5, 78 };
29          System.out.println("成绩总分: " + total(score));  //调用方法 total()
30      }
31  }
```

5.1.3 参数传递方式

C/C++ 语言中,方法的实参向方法形参传递数据有两种方式:传递值和传递指针。为了增强系统安全性,Java 取消了指针,但提供了引用数据类型,如数组、类、接口等。在 Java 语言中,方法的实参向形参传递参数也有两种方式:传递基本数据类型和传递引用类型。如果传递引用类型(相当于 C/C++ 传递指针),形参的改变影响实参。

程序案例 5-2 演示了实参向形参传递引用类型(数组)。第 6 行修改了形参数组 arr 位置 location 的值,这种改变影响了第 11 行实参数组 tempArr。程序运行结果如图 5-3 所示。

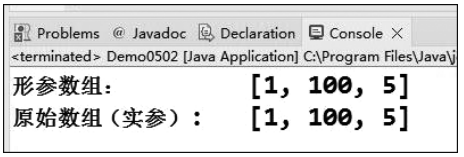

图 5-3　程序案例 5-2 运行结果

【程序案例 5-2】　实参向形参传递数组。

```
1   package chapter05;
2   import java.util.Arrays;
3   public class Demo0502 {
4       //把数组 arr 中 location 位置元素的值修改为 newValue
5       public static void update(int[] arr, int location, int newValue) {
6           arr[location] = newValue;              //修改 location 位置的元素
7           String str=Arrays.toString(arr);
8           System.out.println("形参数组:\t\t"+str);
9       }
10      public static void main(String[] args) {
11          int tempArr[] = { 1, 3, 5 };           //静态初始化定义数组
12          update(tempArr, 1, 100);               //传递数组(引用,相当于指针)
13          String strTempArr = Arrays.toString(tempArr);
14          System.out.print("原始数组(实参):\t"+strTempArr);
15      }
16  }
```

程序说明：运行结果显示形参与实参一致，表示形参的改变影响了实参。

5.2　形参长度可变方法

5.2.1　形参长度可变方法的概念

开发软件项目时，可能存在调用方法时才能确定形参数量的情况，由于传统方法的形参数量是固定的，不能满足该需求。满足用户需求是软件系统的不懈追求。例如，total()方法返回若干整数之和，若干整数可能以整型数组的形式存在(int []arr)，也可能是数量不固定的零散整数(形式如 3,4,7 或者 18,14,17,13,19)。

Java 1.5 之后，提供了定义形参长度可变方法的能力，允许为方法指定数量不固定的形参。

5.2.2　定义形参长度可变方法

【语法格式 5-3】　定义形参长度可变方法。

```
public  static  返回值类型 方法名(类型 形参 1,类型 形参 2, 类型…形参) {
    方法体;                                    //方法主体
    [return 表达式;]                           //方法返回值
}
```

定义方法时，最后一个形参类型后增加省略号，表明该形参能接受个数不确定的实参，

多个实参被当成数组传入。

定义形参长度可变方法需注意两点：①方法中只能定义一个可变形参；②如果这个方法还有其他形参，它们要放在可变形参之前。

编译时，编译器会把最后一个可变形参转换为一个数组形参，并在编译的 class 文件中标上记号，表明该方法的实参个数可变。

例如，下面代码段定义了形参长度可变方法 total()，表明调用该方法时，可向第 3 个参数传入 int 类型的一维数组，也可传入数量不确定的若干整数。

```
1   //计算数组 arr 中 begin~end 的元素之和
2   //最后一个参数 arr 为可变形参
3   public static int total(int begin, int end, int… arr) {
4       int sum = 0;
5       if (!(begin >= 0 && end <= arr.length))
6           return sum;
7       for (int i = begin; i < end; i++)
8           sum += arr[i];
9       return sum;
10  }
```

5.2.3 调用形参长度可变方法

调用形参长度可变方法有两种形式：①实参是对应类型的一维数组；②实参是若干对应类型的变量或常量。

【语法格式 5-4】 调用形参长度可变方法。

```
方法名(实参 1, 实参 2, 数组名)
```

或者

```
方法名(实参 1, 实参 2, 实参 x1, 实参 x2,…,实参 n)
```

程序案例 5-3 演示了调用形参长度可变方法。第 15 行定义了形参长度可变方法 total()，main()方法中采用两种形式调用 total()；第 5、6 行向 total()方法传入的实参是一维数组；第 8 行前面两个参数表示 begin 和 end，第 3～6 个参数(9,8,7,6)转换成数组后传给方法 total()的第 3 个形参。程序运行结果如图 5-4 所示。

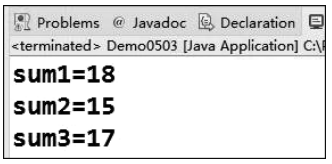

图 5-4 程序案例 5-3 运行结果

【程序案例 5-3】 调用形参长度可变方法。

```
1   package chapter05;
2   public class Demo0503 {
3       public static void main(String[] args) {
4           int[] arr = { 3, 4, 5, 6 };
5           int sum1 = total(0, arr.length, arr);            //第一种调用,实参是数组
6           int sum2 = total(0, 3, new int[] { 4, 5, 6, 7, 8 });  //第一种调用,实参是数组
7           //begin end 不确定实参
8           int sum3 = total(0, 2, 9, 8, 7, 6);   //第二种调用,不确定个数的实参:9,8,7,6
```

```
9           System.out.println("sum1="+sum1);
10          System.out.println("sum2="+sum2);
11          System.out.println("sum3="+sum3);
12      }
13      //计算数组 arr 中 begin~end 的元素之和
14      //最后一个参数 arr 为可变形参
15      public static int total(int begin, int end, int… arr) {
16          int sum = 0;
17          if (!(begin >= 0 && end <= arr.length))
18              return sum;
19          for (int i = begin; i < end; i++)
20              sum += arr[i];
21          return sum;
22      }
23  }
```

5.3　方法重载


方法重载

C 程序中，一个文件中的方法不能同名，这种规范具有避免调用方法时出现混淆、提高调用方法效率等优点。但在编程实践中存在不足，如定义不同数据类型的加法方法，addString(String str1, String str2)实现两个数字字符串数值相加、addTwoInt(int x,int y)实现两个 int 类型数据相加、addThreeInt(int x, int y, int z)实现 3 个 int 类型数据相加，这种方式要求编程人员记忆多个不同名的加法方法，增加了编程人员负担。

Java 改进了方法的定义和调用机制，允许一个类中定义多个同名方法，但这些方法的参数类型和参数个数不同，称这些方法是重载(Overload)。编译阶段，Java 编译器根据每个方法的参数类型和个数决定调用哪个具体方法。通过方法重载，同名方法能完成不同功能，减轻了编程人员的记忆负担。例如，定义加法方法，add (String str1, String str2)实现两个数字字符串的数值相加，add (int x,int y)实现两个 int 类型数据相加，add (int x, int y, int z)实现 3 个 int 类型数据相加，3 个方法名都是 add，通过参数类型不同实现不同功能。

方法重载是 Java 非常重要的一个特性，以前的程序中已经接触到，如利用 System.out.println()方法能输出任何类型的数据。println()方法是一个重载多次的方法。

```
1   System.out.println("为梦想而努力奋斗!");        //输出字符串
2   System.out.println(960);                        //输出 int 型数据
3   System.out.println(true);                       //输出 boolean 型数据
```


方法重载程序

程序案例 5-4 演示了方法重载。第 17 行 add(int a,int b)方法有两个 int 类型的形参，第 21 行 add(String a,String b)方法有两个 String 类型的形参，第 27 行 add(int a, int b, int c)方法有 3 个 int 类型的形参。第 9 行调用第 17 行的 add()方法，第 10 行调用第 21 行的 add()方法，第 13 行调用第 27 行的 add()方法。程序运行结果如图 5-5 所示。

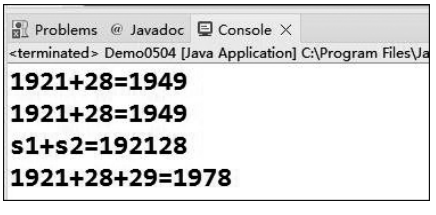

图 5-5　程序案例 5-4 运行结果

【程序案例 5-4】 方法重载。

```
1   package chapter05;
2   public class Demo0504 {
3       public static void main(String[] args) {
4           int a = 1921;
5           int b = 28;
6           int c = 29;
7           String s1 = "1921";
8           String s2 = "28";
9           System.out.println(a + "+" + b + "=" + add(a, b));  //调用重载方法 add
10          System.out.println(s1 + "+" + s2 + "=" + add(s1, s2));  //调用重载方法 add
11          System.out.println("s1+s2=" + s1 + s2);              //字符串连接
12          //System.out.println(s1+"+"+a+"="+add(s1,a));     //出现错误提示
13          System.out.println(a + "+" + b + "+" + c + "=" + add(a, b, c));
14      }
15      //重载方法 add()
16      //该方法有两个 int 类型的形参
17      public static int add(int a, int b) {
18          return a + b;
19      }
20      //该方法有两个 String 类型的形参
21      public static int add(String a, String b) {
22          int x = Integer.parseInt(a);
23          int y = Integer.parseInt(b);
24          return x + y;
25      }
26      //该方法有 3 个 int 类型的形参
27      public static int add(int a, int b, int c) {
28          return a + b + c;
29      }
30  }
```

方法重载是 Java 语言非常重要的特性之一,它减轻了编程人员负担,提高了软件开发效率,使软件结构更加清晰。

定义方法重载时注意以下 3 个问题。

(1) 一个类中(或者有继承关系的类之间)的方法才能重载,不同类中的方法不能重载。例如,下面代码段,第 2 行类 DemoX 中定义的方法 add(int x,int y)与第 7 行类 DemoY 中定义的方法 add(int x,int y),这两个方法的签名一样,但它们不是重载关系。

```
1   class DemoX{
2       int add(int x,int y) {
3           return x+y;
4       }
5   }
6   class DemoY{
7       int add(int x,int y) {
8           return x+y;
9       }
10  }
```

(2) 方法重载有 3 种形式:①方法声明的形参个数不同;②方法声明中对应的形参类

型不同；③方法声明的形参个数和类型都不同。

（3）方法的返回值类型不同不能作为方法重载的依据。例如，下面代码段类 DemoX 中，第 2 行定义了 int add(int x,int y)，该方法的返回值类型 int；第 5 行定义了 String add (int x,int y)，该方法的返回值类型 String，虽然这两个方法的返回值类型不同（分别是 int 和 String），但形参一样，因此它们不是重载方法。

```
1   class DemoX{
2       int add(int x,int y){
3           return x+y;
4       }
5       String add(int x,int y) {
6           return String.valueOf(x)+String.valueOf(y);
7       }
8   }
```

5.4　递归方法

Java 语言与其他高级语言（如 C/C++、Python）一样，都提供了方法递归调用功能。递归方法使程序更加简单清晰，而且还会使程序的执行逻辑与数理逻辑保持一致。图 5-6 显示了递归方法的调用过程。本书仅介绍 Java 具有定义递归方法的能力，关于递归方法的详细内容可参考有关书籍。

程序案例 5-5 演示了递归方法。第 8 行 factorial()方法自己调用自己（即递归调用），问题规模不断缩小，一直向递归结束条件逼近，num 等于 1 时结束递归。程序运行结果如图 5-7 所示。

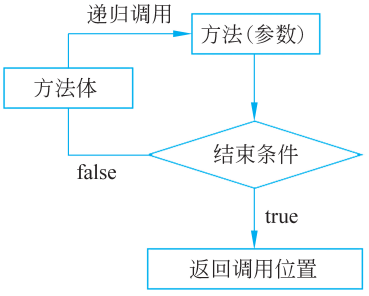

图 5-6　递归方法的调用过程

图 5-7　程序案例 5-5 运行结果

【程序案例 5-5】　递归方法计算 n!。

```
1   package chapter05;
2   public class Demo0505 {
3       public static void main(String[] args) {
4           int number=10;                                          //定义整数
5           System.out.println(number+"!="+factorial(number));  //调用递归方法
6       }
7       //定义递归方法计算 1×2×3×4×…×N
8       public static long factorial(int num){
```

```
9           if(1==num)                                          //递归结束条件
10              return 1;
11          else
12              return factorial(num-1) * num;                  //递归调用方法
13      }
14  }
```

5.5 小　　结

本章主要知识点如下。

(1) 方法是一段可重复调用的代码块,定义方法需要确定返回值类型、方法名和形参。

(2) Java 取消了指针类型,提供了类似的引用类型,实参向形参传递引用类型时(如数组),形参的改变将影响实参,类似于 C/C++ 语言实参向形参传递指针类型。

(3) Java 1.5 提供了形参长度可变方法,该方法要求可变参数放在形参列表的最后位置,并且只有一个可变形参。

(4) 方法重载是 Java 语言非常重要的特性之一,其指一个类中的方法名相同,但方法的参数类型或者参数个数不同。

(5) Java 提供了定义递归方法的能力。

5.6 习　　题

5.6.1 填空题

1. Java 程序中,如果方法的形参是数组,则调用该方法时传递的是数组的(　　)。

2. 在使用 Arrays.binarySearch()方法查找一维数组中的某个元素前,需要利用 Arrays 中的(　　)方法对该一维数组进行排序。

3. Java 程序中,定义多个名字相同但参数类型或参数个数不同的方法,称这些方法是(　　)。在(　　)阶段,Java 编译器根据每个方法的参数类型和个数决定调用哪个具体方法。

5.6.2 选择题

1. (　　)不是重载方法。

A. int print(String str){}
　 int print(String str,int num){}

B. int print(String str,int num){}
　 int print(int num,String str){}

C. int print(String str){}
　 static print(String str){}

D. int print(double x){}
　 int print(float x){}

2. 数组名作为方法形参,调用该方法时实参向形参传递的是(　　)。

A. 数组的元素　　B. 数组的栈地址

C. 数组自身　　D. 数组的引用

3. 方法声明正确的是(　　)。

A. void method(int …x,int y ,int z){ }

B. void method(int x,int …y ,int z){ }

C. void method(int x,int y,int …z){ }

D. void method(int …x,int …y,int z){ }

4. 定义如下类 Base,(　　)是 setNum()方法的重载方法。

```
class Base{
    public void setNum (int a,int b,float c){  }
}
```

A. protected float setNum (int a,int b,float c){return c;}

B. void setNum (int a,int b,float c){ }

C. int setNum (int a,int b,int c){ return a;}

D. protected int setNum (int x,int y,float z){return b;}

5. (　　)是 void getSort(int x)的重载方法。

A. public getSort(float x)

B. int getSort(int y)

C. double getSort(int x,int y)

D. void get(int x,int y)

5.6.3　简答题

1. 简述方法重载的实现方式,并举例说明。

2. 简述调用方法时实参向形参传递基本类型数据与引用类型数据之间的区别。

5.6.4　编程题

1. int maxArray(int [] arr)和 int maxArray(int [][] arr)两个方法返回数组所有元素的最大值,编写程序定义这两个方法并测试。

2. 斐波那契数列在现代物理、准晶体结构、化学等领域都有直接应用。斐波那契数列公式:$F(0)=0$,$F(1)=1$, $F(n)=F(n-1)+F(n-2)$。利用递归算法编程实现该公式。