第5章 递归和分治

一个函数调用了它自己，称为递归。递归和循环可以互相替代。一种程序设计语言，支持递归就可以不需要支持循环，支持循环就可以不需要支持递归。例如，早期的 Lisp 语言，就不支持循环，只支持递归。当然，为了方便使用，程序设计语言一般都既支持递归，也支持循环。

从替代循环的角度看，递归和循环一样是一种手段，可以用来解决任何问题。但是递归更多的是一种解决问题的思想——从这个角度看，也可以称递归是一种算法。

本章主要通过具体例题，分为以下三种情况讲述递归的用途。

(1) 替代多重循环来进行枚举。

(2) 解决用递归形式定义的问题。

(3) 将问题分解为规模更小的子问题进行求解。

上述三个方面其实没有也不需要有严格的区分界限。有的问题，归类为上述不止一种情况都说得过去。例如，5.2 节的例题“绘制雪花曲线”，归类为第(2)或第(3)种情况都可以。

第(3)种情况，如果分解出来的子问题不止一个且相互无重叠，一般来说就可以算是“分治”。

还有一种递归可以称为“间接递归”，例如，函数 A 调用了函数 B，函数 B 调用了函数 C，函数 C 又调用了函数 A，就可以说函数 A、B、C 都间接递归调用了自身。本章不涉及这类递归。

5.1 用递归进行枚举

5.1.1 案例: N 皇后问题(P0230)

将 N 个皇后摆放在一个 N 行 N 列的国际象棋棋盘上，要求任何两个皇后不能互相攻击(两个皇后在同一行，或同一列，或某个正方形的对角线上，就会互相攻击，称为冲突)。输入皇后数 $N(1\leqslant N\leqslant 9)$，输出所有的摆法。无解输出“NO ANSWER”。行列号都从 0 开始算。

输入： 一个整数 N，表示要把 N 个皇后摆放在一个 N 行 N 列的国际象棋棋盘上。

输出： 所有的摆放放案。每个方案一行，依次是第 0 行皇后位置、第 1 行皇后位置、……、第 $N-1$ 行皇后位置。多种方案输出顺序如下：优先输出第 0 行皇后列号小的方案。如果两个方案第 0 行皇后列号一致，那么优先输出第 1 行皇后列号小的方案，以此类推。

样例输入

```
4
```

样例输出

```
1 3 0 2
2 0 3 1
```

解题思路：在皇后数目不确定的情况下，用循环解决比较麻烦，因此可以采用递归的方式进行枚举，程序如下。

```
//prg0350.java
1.  import java.util.*;
2.  class NQueensProblem {
3.        int N;
4.        int [] result;                    //result[i]是第 i 行皇后摆放位置
5.        NQueensProblem(int n_) {
6.            N = n_;         result = new int[N];
7.        }
8.        boolean isOk(int n,int pos) {     //判断第 n 行的皇后放在第 pos 列是否可行
9.        //此时第 0 行到第 n-1 行的皇后的摆放位置已经存放在 result[0]至 result[n-1]中
10.           for (int i = 0;i < n; ++i)
11.               //检查位置 pos 是否会和前 0~n-1 行已经摆好的皇后冲突
12.               if (result[i] == pos ||
13.                       Math.abs(i-n) == Math.abs(result[i] - pos))
14.                   return false;
15.           return true;
16.       }
17.       boolean queen(int i) {
18.           //解决 N 皇后问题，现在第 0 行到第 i-1 行的 i 个皇后已经摆放好了
19.           //要摆放第 i 行的皇后。返回值表示这种情况下最终能否成功
20.         if (i == N) {             //已经摆好了 N 个皇后，说明问题已经解决，输出结果即可
21.             for (int k=0;k<N;++k)
22.                 System.out.print(result[k]+" ");
23.             System.out.println();
24.             return true;
25.         }
26.         boolean succeed = false;
27.         for (int k=0;k<N;++k)          //枚举所有位置
28.             if (isOk(i,k)) {           //看可否将第 i 行皇后摆在第 k 列
29.                 result[i] = k;         //可以摆在第 k 列，就摆上
30.                 succeed = queen(i+1) || succeed;       //接着去摆放第 i+1 行的皇后
31.             }
32.         return succeed;
33.       }
34.       void solve() {
35.           if (!queen(0))
36.               System.out.println("NO ANSWER");
37.       }
38. }
39. public class prg0350  {               //请注意：在 OpenJudge 提交时类名要改成 Main
40.     public static void main(String[] args){
41.         Scanner reader = new Scanner(System.in);
42.         int n = reader.nextInt();
43.         new NQueensProblem(n).solve();
```

```
44.     }
45. }
```

queen 函数返回值为 true 或者 false，queen(i)的返回值表示：当前，前 i 个皇后已经摆好且不冲突，它们的摆法放在 result[0,$i-1$]中(本书中用 $a[x,y]$表示数组或字符串 a 中从 $a[x]$到 $a[y]$这样连续的一段)，在不改变这前 i 个皇后的摆法的前提下，继续往下摆放，最终能否找到至少一种成功的 N 皇后摆法。在第 27 行，函数试图为第 i 行的皇后找到所有和前 i 个皇后不冲突的位置，所谓“枚举”，就体现在本行。每找到一个合适的位置，就摆放之，然后递归调用一次 queen($i+1$)继续后面皇后的摆放；如果一个合适位置都找不到，则返回 false，表示在当前情况下，最终无法摆放成功。

第 24 行：由于函数是递归调用的，所以有几个解，本行就会被执行几次。例如，在第 0 行皇后摆在第 0 列的情况下，执行 queen(1)最终会成功，本行会得到执行；在第 0 行皇后摆放在第 1 列的情况下，执行 queen(1)最终也会成功，本行又会得到执行。实际上，如果有 k 个解的第 0 行皇后都是摆放在第 0 列的，则会有 k 次执行到本行时，第 0 行皇后是摆放在第 0 列的。

第 26 行：succeed 表示在当前情况下，再往下摆能否至少找到一个解，先假定为 false。

第 29 行：假设找到的第一个合法摆放位置是 k_1，第 i 行皇后摆放在第 k_1 列的情况下，会继续执行 queen($i+1$)，从此处一直递归下去，有可能会找到多种 N 皇后的最终合法摆放方案，即多次走到第 21 行，输出多个解。这些解的第 0 行到第 i 行的摆放方案都是相同的，例如，第 i 行都是摆放在 k_1 位置。queen($i+1$)执行过程中经历层层多分支递归，终究会返回，返回后回到第 27 行的 for 循环，继续寻找下一个可行的第 i 行摆放位置 k_2，然后再继续递归下去。

因此上面的程序会输出所有的解。

第 30 行：在第 27 行开始的循环中，只要有一次执行本行时 queen($i+1$)返回 true，succeed 的值就会是 true，意味着在当前情况下最终能够摆放成功。

本程序中，result 数组中的一个元素，本质上就相当于多重循环解法里的一个循环控制变量，它们都是表示一行皇后的位置的。

请注意：如果要将程序改写成找到一个解就结束，则只需要将第 30 行替换为下面两行。

```
//prg0351.java
    if(queen(i+1))                          //接着去摆放第 i+1 行的皇后
        return true;
```

这样的话，摆放第 i 行皇后的时候，一旦发现一个能导致最终摆放成功的摆法，就不会去尝试第 i 行的下一个摆法，因此对每行的皇后，都只找到一个能导致最终成功的摆法，于是程序的第 24 行只会执行 1 次，程序只输出一个解。

N 皇后问题的本质，是有 N 个变量，这 N 个变量取值的某些组合，能够满足某个条件，要求出这些满足条件的组合。下面的奥数问题、接下来的全排列问题、习题中的棋盘问题本质都是如此，都可以用类似 N 皇后问题的办法解决。

5.1.2 案例：奥数问题(P0100)的递归解法

本题就是 4.1.2 节案例“奥数问题”。这个问题和 N 皇后问题很像。可以将每个字母看作一个位置，等式中最多出现 5 个不同字母，所以一共有 5 个位置。在每个位置需要摆一个数，且每个位置上的数不一样。将等式中的字母用其对应位置上的数替换，要使得替换后的等式

成立，且等式中不可以有带多余前导 0 的数出现。下面的递归程序中，这 5 个位置就是数组 a 的 5 个元素，函数 done(i)表示 $a[0,i-1]$这 i 个位置已经摆上数的情况下，接着要在位置 $a[i]$及后面的位置摆数，求最终能否成功。

```
//prg0312.java
1.  import java.util.*;
2.  class AoshuProblem2 {
3.      String s1,s2,s3;
4.      AoshuProblem2(String str1,String str2,String str3) {
5.          s1 = str1; s2 = str2; s3 = str3;
6.      }
7.      int a[] = new int[5];            //a[0]存放'A'表示的数,a[1]存放'B'表示的数……
8.      int toInt(String s) {
9.        //依据 a 将'ABE'这样的字符串转成整数。若有前导 0 则返回-1 表示失败
10.       String result = "";
11.       for(int i=0;i<s.length();++i) {
12.          char c = s.charAt(i);
13.           result += (char)('0' + a[c-'A']);
14.       }
15.       if (result.length() > 1 && result.charAt(0) == '0')
16.            return -1;
17.       return Integer.parseInt(result);
18.     }
19.     private boolean done(int i) {    //被调用时,a[0,i-1]已存放了 i 个字母代表的数
20.       //返回值表示在此情况下最终能否找到解
21.       if (i == 5) {
22.            int n1 = toInt(s1),n2 = toInt(s2),
23.                n3 = toInt(s3);
24.            if (n1 >= 0 && n2 >= 0 && n3 >= 0
25.                     && n1 + n2 == n3) {
26.                  System.out.printf("%d+%d=%d\n",n1,n2,n3);
27.                  return true;
28.            }
29.       }
30.       else {                         //i!= 5
31.         for(int k=0;k<10;++k) {
32.               int j = 0;
33.               for(;j<i;++j)         //本循环判断 k 这个数是否已经被用过
34.                   if(a[j] == k)
35.                       break;        //发现 k 这个数已经被用过
36.               if(j == i) {          //如果 k 这个数还没被用过
37.                   a[i] = k;         //在位置 i 摆数 k,即让第 i 个字母代表数 k
38.                   if (done(i+1))
39.                      return true;
40.               }
41.         }
42.       }
43.       return false;
44.     }
45.     void solve() {
46.        if (!done(0))
47.            System.out.println("No Solution");
48.     }
49. }
```

```
50. public class prg0312 {                    //请注意:在 OpenJudge 提交时类名要改成 Main
51.     public static void main(String[] args){
52.         Scanner reader = new Scanner(System.in);
53.         int n = reader.nextInt();
54.         for(int i=0;i<n;++i) {
55.             String s1 = reader.next(),s2 = reader.next(),
56.                    s3 = reader.next();
57.             new AoshuProblem2(s1,s2,s3).solve();
58.         }
59.     }
60. }
```

本程序在位置 i 摆放数 k 前，先判断能不能摆，能摆了才摆，而不是不管能不能摆，把 5 个位置都摆上数再判断整个完整的摆法是否符合要求，这样可以提高效率。

5.1.3 案例: 全排列(P0240)

给定一个由不同的小写字母组成的字符串，输出这个字符串的所有排列。假设对于小写字母有'a'<'b'<…<'y'<'z'，而且给定的字符串中的字母已经按照从小到大的顺序排列。

输入： 输入只有一行，是一个由不同的小写字母组成的字符串，已知字符串的长度为1～6。

输出： 输出这个字符串的所有排列方式，每行一个排列。要求字母序比较小的排列在前面。字母序如下定义。

已知 $S=s_1s_2\cdots s_k$，$T=t_1t_2\cdots t_k$，则 $S<T$ 等价于，存在 $p(1\leqslant p\leqslant k)$，使得 $s_1=t_1$，$s_2=t_2$，…，$s_{p-1}=t_{p-1}$，$s_p<t_p$ 成立。

样例输入

```
abc
```

样例输出

```
abc
acb
bac
bca
cab
cba
```

本题的本质就是在 n 个位置(编号 0～$n-1$)摆 n 个不同字母，要求给出符合“每个字母只出现一次”这个要求的所有摆法。解题程序如下。

```
//prg0360.java
1.  import java.util.*;
2.  class AllPermutations {
3.      String lst;                              //存放输入的字符串
4.      char [] result;                          //result[i]表示第 i 个位置摆放的字母
5.      boolean [] used;                    //used[i]表示 lst 中的第 i 个字母是否已经用过
6.      int n;                                   //排列的长度是 n 个字符
7.      void permutation(int i) {                //从第 i 个位置起摆放字母
8.          if (i == n) {          //条件满足则说明 n 个位置都摆上字母了，即发现了一个排列
9.              for (char x:result)
10.                 System.out.print(x);
11.             System.out.println("");
```

```
12.             return;
13.         }
14.         for (int k=0;k<n;++k)
15.             if (!used[k]) {                    //第 k 个字母还没用过
16.                 result[i] = lst.charAt(k);     //在第 i 个位置摆上第 k 个字母
17.                 used[k] = true;
18.                 permutation(i+1);              //从第 i+1 个位置起继续往下摆
19.                 used[k] = false;
20.             }
21.     }
22.     void solve() {
23.         Scanner reader = new Scanner(System.in);
24.         lst = reader.next();
25.         n = lst.length();
26.         result = new char [n];
27.         used = new boolean[n];
28.         for (int i=0;i<n;++i)
29.             used[i] = false;
30.         permutation(0);                        //从第 0 个位置开始摆放字母
31.     }
32. }
33. public class prg0360  {                  //请注意:在 OpenJudge 提交时类名要改成 Main
34.     public static void main(String[] args){
35.         new AllPermutations().solve();
36.     }
37. }
```

函数 permutation(i)表示,在 0～i－1 这 i 个位置已经摆好字母的情况下,从第 i 个位置开始继续摆放字母。位置 k 摆放的字母,记录在 result[k]中(k＝0,1,…,n－1)。

第 14 行:枚举所有在第 i 个位置可能摆放的字母,k 是字母在 lst 中的下标。由于在每个位置枚举字母的时候都是按照字母从小到大的顺序进行,所以最终会按从小到大的顺序得到一个个排列,类似于 N 皇后问题得到解的顺序。

第 15 行:一个排列中,每个字母只能用一次,因此用数组元素 used[k]记录字母 lst[k]是否已经被用过。lst[k]还没用过,就可以摆在位置 i。摆上后就要将 used[k]设置为 true,表示 lst[k]已经被使用,如第 17 行所示。

第 19 行:下次循环就要在位置 i 尝试摆放别的字母。要在位置 i 摆别的字母,就应该将刚才在第 16 行摆放在位置 i 的字母 lst[k]拿走,那么字母 lst[k]就应该恢复成没用过,这样在后续位置还可以摆放它。因此要让 used[k]＝false。

5.2 解决用递归形式定义的问题

有一些问题或者概念,本身的定义就是递归形式的。例如“自然数 n 的阶乘”这个概念,可以定义成 1×2×3×…×(n－1)×n,也可以用以下两句话来定义。

(1) 1 的阶乘是 1。

(2) n＞1 时,n 的阶乘等于 n 乘以(n－1)的阶乘。

第(2)句话,定义“阶乘”这个概念的时候,用到了“阶乘”这个词,看上去像循环定义,让人无法理解。但是,由于有语句(1)的存在,上面这两句“自然数 n 的阶乘”的定义,就是严密且可以理解的。例如,若问“3 的阶乘是什么”,要先回答“2 的阶乘是什么”;要回答“2 的阶乘是

什么”，就要回答“1 的阶乘是什么”。按照语句(1)，1 的阶乘是 1，因此往回倒推就可以知道 3 的阶乘是什么了。

可以说，在上面的两句话的阶乘定义中，语句(2)的形式是递归的，而语句(1)就是递归的终止条件。

按照这种定义方式，求自然数 n 的阶乘的函数可以写成：

```
int factorial(int n) {
   if (n == 1)
      return 1;
   return n * factorial(n-1);
}
```

这是最简单的用递归函数解决递归形式的问题的例子。

在上面的程序中，判断“n == 1”是否为真是基本操作，即进行次数最多的操作。设求 n 阶乘的基本操作次数为 $T(n)$，则有：

$$T(n)=1+T(n-1)=1+1+T(n-2)=1+1+1+T(n-3)=\cdots$$
$$=1+1+\cdots+T(1)=1+1+\cdots+1(n\text{ 个 }1)$$

所以函数的复杂度是 $O(n)$。

5.2.1 案例：波兰表达式(P0250)

波兰表达式是一种把运算符前置的算术表达式。例如，一般形式的表达式 2 + 3 的波兰表示法为+ 2 3。波兰表达式的优点是计算时不需要考虑运算符的优先级，因此不必用括号改变运算次序，例如，(2 + 3) * 4 的波兰表示式为 * + 2 3 4。本题求解波兰表达式的值，其中运算符包括+、-、*、/ 4 个。

输入： 输入为一行，其中运算符和运算数之间都用空格分隔，运算数是浮点数。

输出： 输出为一行，表达式的值。

样例输入

```
* + 11.0 12.0 + 24.0 35.0
```

样例输出

```
1357.000000
```

虽然什么是“波兰表达式”不难理解，但是题目其实并没有给出“波兰表达式”的准确定义。波兰表达式可以用递归的形式准确定义如下。

(1) 一个数是一个波兰表达式，其值就是该数本身。

(2) 若一个波兰表达式不是一个数，则其形式为：“运算符 波兰表达式 1 波兰表达式 2”，其值是以“波兰表达式 1”的值作为第一操作数，以“波兰表达式 2”的值作为第二操作数，进行“运算符”所代表的运算后的值。“运算符”有加减乘除 4 种，分别表示为“+”“-”“*”“/”。

根据上述定义，可以写出解题程序如下。

```
//prg0370.java
1.  import java.util.*;
2.  class PolishExpression {
3.      int N;
4.      String [] exp;       //exp[i]要么是一个数的字符串形式如"11.0",要么是一个运算符
5.      double polish() {   //从 exp[N]处开始取出若干元素构成一个波兰表达式并返回其值
```

```
6.          int M = N;
7.          ++ N;
8.          if (exp[M].equals("+"))
9.              return polish() + polish();
10.         else if (exp[M].equals("-"))
11.             return polish() - polish();
12.         else if (exp[M].equals( "*"))
13.             return polish() * polish();
14.         else if (exp[M].equals( "/"))
15.             return polish() / polish();
16.         else
17.             return Double.parseDouble(exp[M]);
18.     }
19.     void solve() {
20.         Scanner reader = new Scanner(System.in);
21.         exp = reader.nextLine().split(" ");
22.         N = 0;
23.         System.out.printf("%.6f\n",polish());
24.     }
25. }
26. public class prg0370  {            //请注意:在 OpenJudge 提交时类名要改成 Main
27.     public static void main(String [] args)  {
28.         new PolishExpression().solve();
29.     }
30. }
```

第 3 行：N 相当于一个指针，表示 polish 函数被调用时，应该从字符串数组 exp 中下标为 N 的元素开始，取出若干个元素（假设 x 个）构成一个波兰表达式，计算出其值并返回。而且 polish 函数还必须让 N 变为 $N+x$，以便下一次调用 polish 函数时可以从正确的位置继续取元素。例如，$N==0$ 时调用 polish()，一定是将 exp 中的所有元素都取出作为一个波兰表达式。

第 8、9 行：按照波兰表达式定义，如果 exp[M]是“+”，则其后面一定跟着两个波兰表达式。“+”取走后，N 的值被加 1。然后调用一次 polish()取出第一个波兰表达式并算出值，且让 N 推进到合适位置，再调用一次 polish()取出第二个波兰表达式，算出值，和第一个波兰表达式的值相加后返回。当然，第二次调用 polish()的时候也会推进 N 到合适位置。

第 17 行：按照波兰表达式定义，如果 exp[M]不是运算符，则其一定是一个数。那么取出的波兰表达式就是 exp[M]，其值为 exp[M]代表的那个数。

由于只需要从头到尾扫描整个表达式一遍，所以复杂度为 $O(n)$。

★★5.2.2 案例：绘制雪花曲线

绘制雪花曲线，更是典型的以递归形式定义的问题。

要进行绘图，需要导入 javax.swing 包和 java.awt 包。前者中的 JFrame 类用于生成一个窗口，后者中的 Graphics 类用于画图。Graphics 对象可以看作窗口上的画板，调用 Graphics 对象的 drawLine()方法，可以绘制线段。

绘图是在一个窗口中进行的，创建了一个 JFrame 对象，就创建了一个窗口。JFrame 对象的 setSize()方法可以设置窗口大小。窗口是一个平面直角坐标系，窗口的左上角是坐标系原点，即其坐标是(0,0)。规定正东方向是 0°，正南方向是 90°，正西方向是 180°，正北方向是

270°。当然也可以说正北方向是－90°，正西方向是－180°。

绘制雪花曲线须在窗口上进行。雪花曲线也称为科赫曲线，其递归定义如下。

(1) 长为 size，方向为 x(单位：弧度)的 0 阶雪花曲线，是沿方向 x 绘制的一根长为 size 的线段。

(2) 长为 size，方向为 x 的 n 阶雪花曲线，由以下 4 部分依次拼接组成。

① 长为 size/3，方向为 x 的 $n-1$ 阶雪花曲线。

② 长为 size/3，方向为 $x-\pi/3$ 的 $n-1$ 阶雪花曲线。

③ 长为 size/3，方向为 $x+\pi/3$ 的 $n-1$ 阶雪花曲线。

④ 长为 size/3，方向为 x 的 $n-1$ 阶雪花曲线。

图 5.1～图 5.3 是几个雪花曲线的示意图。

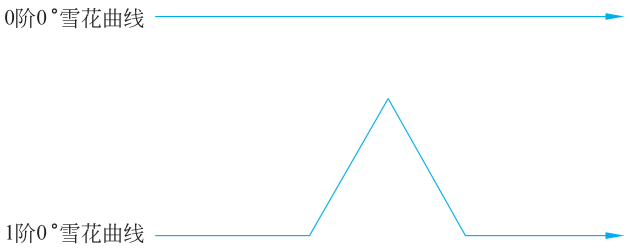

图 5.1　0 阶和 1 阶雪花曲线

图 5.2　2 阶雪花曲线

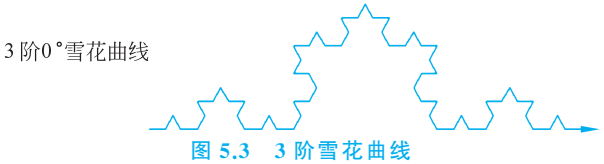

图 5.3　3 阶雪花曲线

绘制长度为 600 像素，方向为 0 的 3 阶雪花曲线的程序如下。

```
//prg0380.java
1.  import javax.swing.*;                    //图形界面需要
2.  import java.awt.*;                       //画图需要
3.  class SnowCurveWindow extends JFrame {
4.    public void drawSnowCurve(Graphics g, int n,int x,int y,
5.              double size,double dir) {
6.          //在 g 上绘制一个 n 阶长度为 size,方向为 dir(单位:弧度)的雪花曲线,起点坐标为(x,y)
7.          if(n == 0) {                     //下面画一条 0 阶雪花曲线,即一个线段
8.              int x2 = (int)Math.round(x + Math.cos(dir) * size);
9.              int y2 = (int)Math.round(y + Math.sin(dir) * size);
10.             g.drawLine(x,y,x2,y2);       //从起点(x,y)到终点(x2,y2)画一条线段
11.         }
12.         else {
13.             double delta[] = {0,-Math.PI/3,Math.PI/3,0};  //PI 是 π
14.             size /= 3;
15.             for(int i=0;i<4;++i) {
16.                 drawSnowCurve(g,n-1,x,y,size,dir+delta[i]);
```

```
17.                 x = (int) Math.round((x + Math.cos(dir+delta[i]) * size));
18.                 y = (int) Math.round((y + Math.sin(dir+delta[i]) * size));
19.             }
20.         }
21.     }
22.     public SnowCurveWindow(int n,int x,int y,
23.                         int size,double dir){
24.         setSize(800,600);              //JFrame 的方法,设置窗体大小为 800px * 600px
25.         setLocationRelativeTo(null); //让窗口在屏幕上居中
26.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27.         setContentPane(new JPanel() {
28.             public void paint(Graphics g){
29.                 drawSnowCurve(g,n,x,y,size,dir);
30.             }
31.         });
32.         setVisible(true);
33.     }
34. }
35. public class prg0380 {
36.     public static void main(String[] args) {
37.         new SnowCurveWindow (3,100,400,600,0);
38.         //绘制 3 阶长度为 600,方向为 0 的雪花曲线,起点坐标为(100,400)
39.     }
40. }
```

程序运行结果如图 5.4 所示。

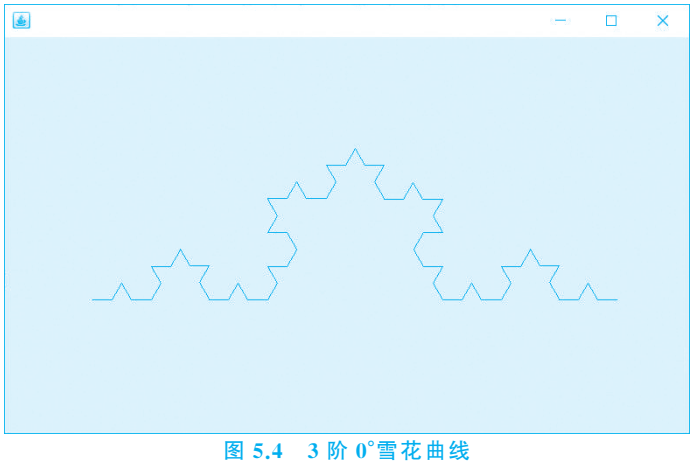

图 5.4　3 阶 0°雪花曲线

类 SnowCurveWindow 从 JFrame 派生而来,第 37 行创建一个 SnowCurveWindow 对象,即创建了一个窗口。SnowCurveWindow 构造方法中调用的各种以“set”开头的方法,都是 JFrame 类的方法。

第 27 行:本行 new 出来的对象,是一个匿名类的对象,该匿名类从 JPanel 类派生而来并重写了 paint()方法。JPanel 对象可以看作一个面板,可以在上面绘图。setContentPane 在窗口上放置该匿名类对象,窗口显示和刷新时,会调用该匿名类对象的 paint()方法并传入画布对象 Graphics g。因此要将绘制雪花曲线的代码,即对 drawSnowCurve 函数的调用,写在 paint()方法中。

第 13～19 行:按照雪花曲线的递归定义,一条 dir 方向上的长为 size 的 n 阶雪花曲线,应