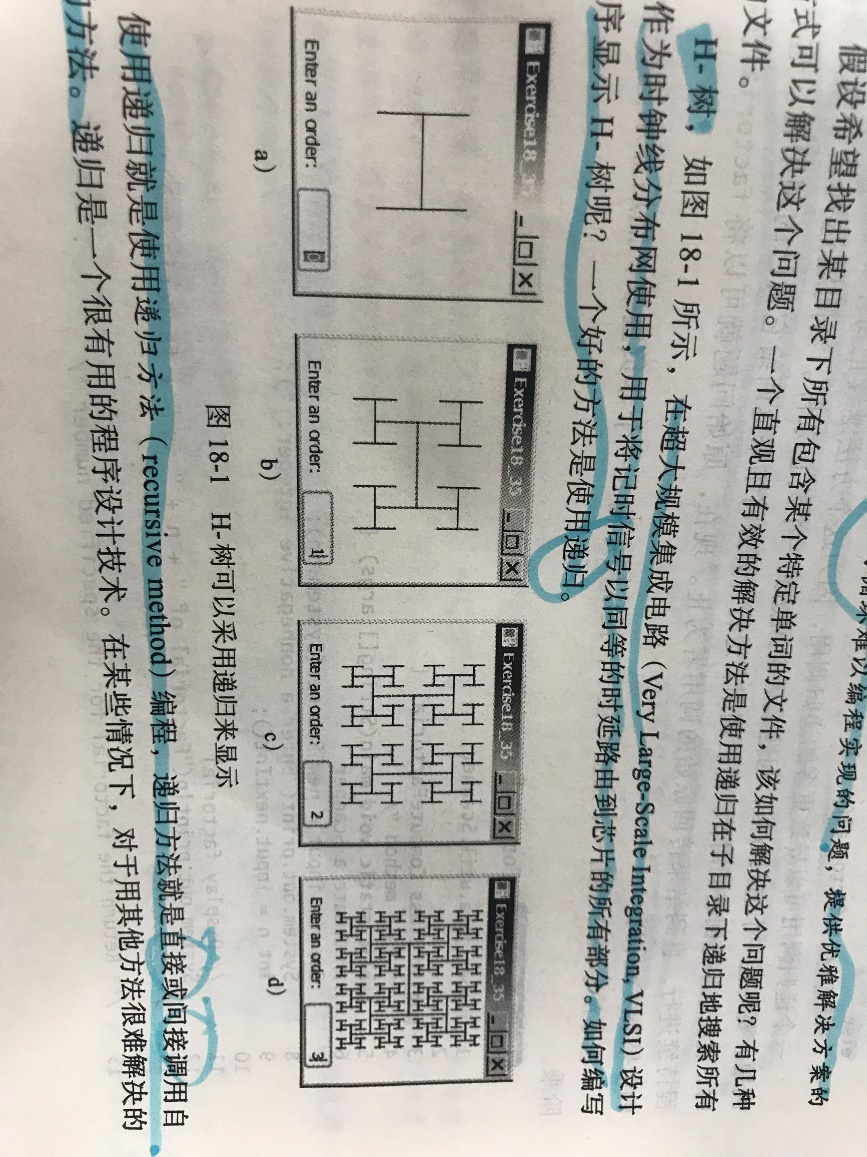
# 第十八章 递归

## 18.1引言

1.要点提示：递归是一种技术，用简单的循环优雅的解决难以编程实现的问题。（递归的思想很普遍、灵活）

2.H-树，在超大规模集成电路设计中作为时钟线分布网使用，用于将计时信号以同等的时延路由到芯片的所有部分，递归是编写H-树的好方法。



3.使用递归就是使用递归方法编程，递归方法就是直接或间接调用自身的方法。

## 18.2示例学习：计算阶乘

1.要点提示：递归方法是调用自身的方法。

2.计算factorial(n)的递归算法可以简单的描述如下：

If(n==0){

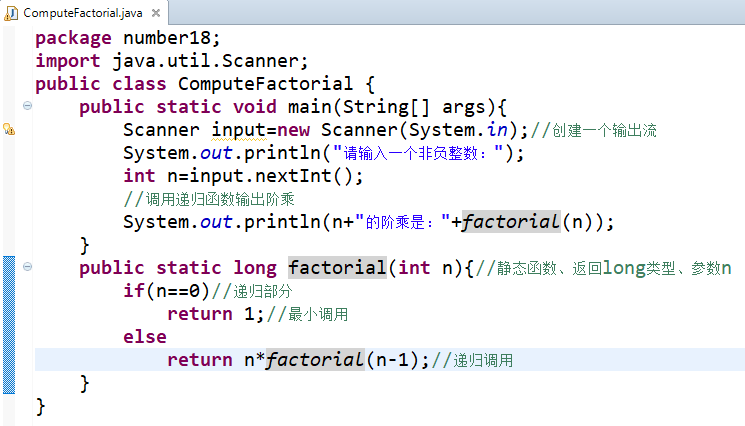
Return 1;

}else{

Return n\*factorial(n-1);}//调用自身

一个递归调用可以导致更多的递归调用，因为这个方法继续把每个子问题分解成新的子问题。要终止一个递归方法，问题的最后必须达到一个终止条件。当问题达到这个终止条件时，就将结果返回给调用者。然后调用者进行计算并将结果返回给它自己的调用者。这个过程持续进行，知道结果传回原始调用者为止

3.以下给出一个完整程序，提示用户输入一个非负数，然后显示这个数的阶乘。



如果递归不能使问题简化并最终收敛到基础情况，就有可能出现无限递归。例如，假设将factorial方法错误的写成如下所示：（缺少n==0的情况）

Public static long factorial(int n){return n\*factorial(n-1);}

那么这个方法就会无限的运行下去，并会导致一个StackOverflowError。

4.本节演示了一个调用自身的递归方法，这被称为直接递归。如果方法A调用方法B，方法B又调用方法A，那么这叫做间接递归。

## 18.3示例学习：计算斐波那契数

1.斐波那契数列问题

数列：0 1 1 2 3 5 8 13 21 34 55 89

下标：0 1 2 3 4 5 6 7 8 9 10 11

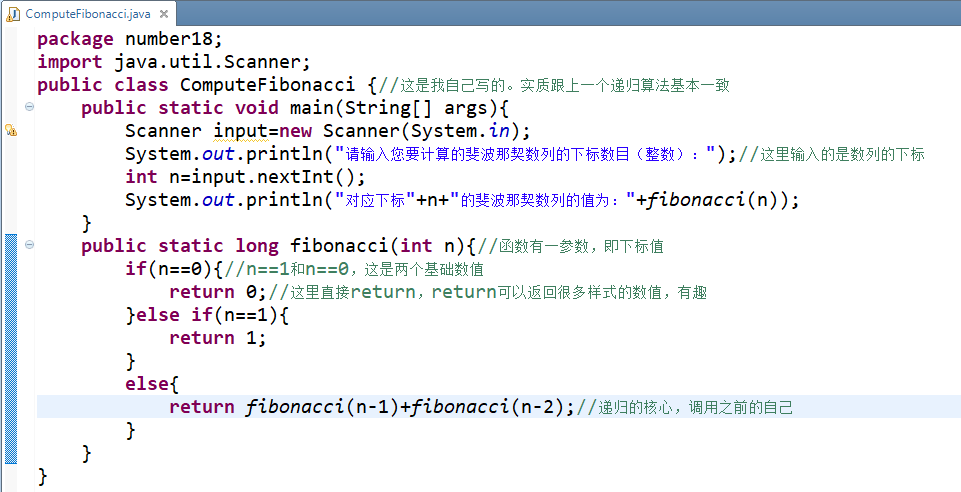
斐波那契数列从0和1开始，之后的每个数列都是序列中前两个数的和。数列可以递归定义为：

Fib(0)=0;基础

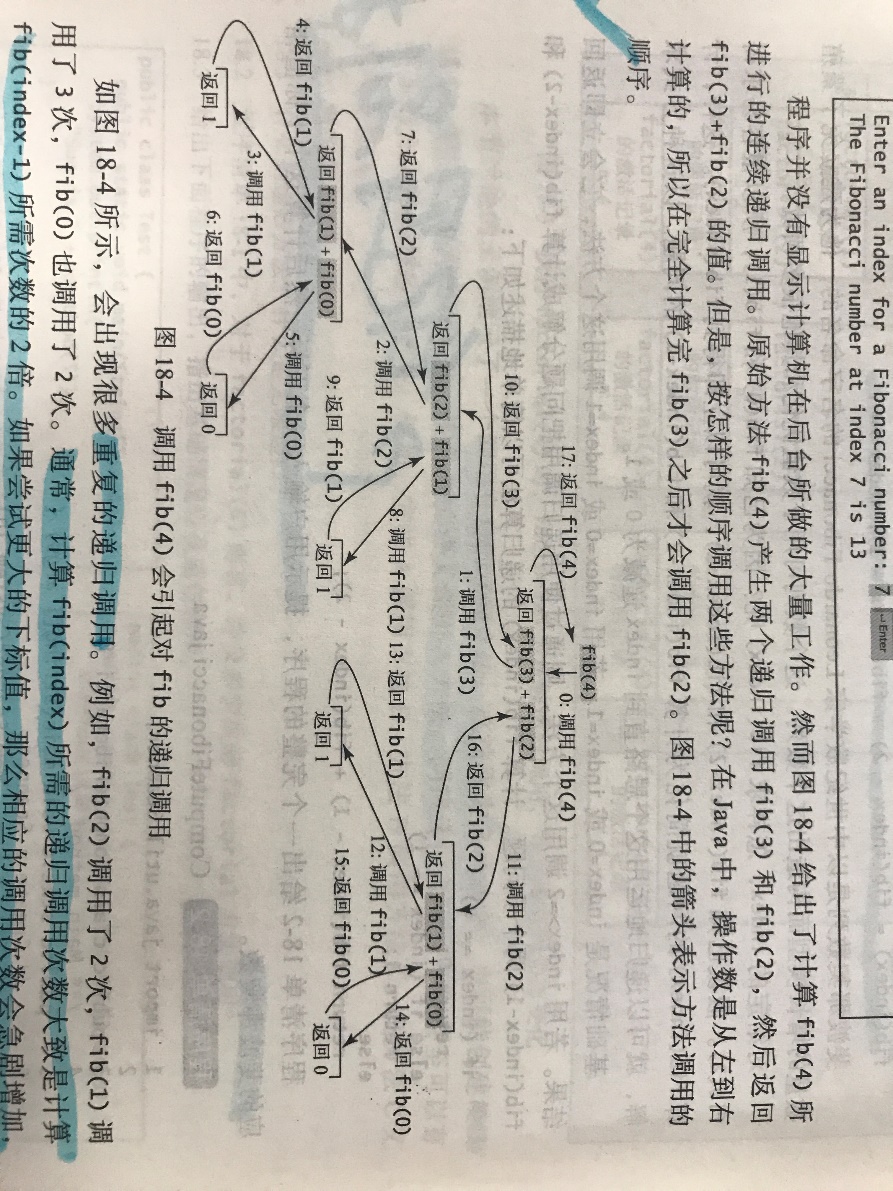
Fib(1)=1;基础

Fib(index)=fib(index-2)+fib(index-1); index>=2

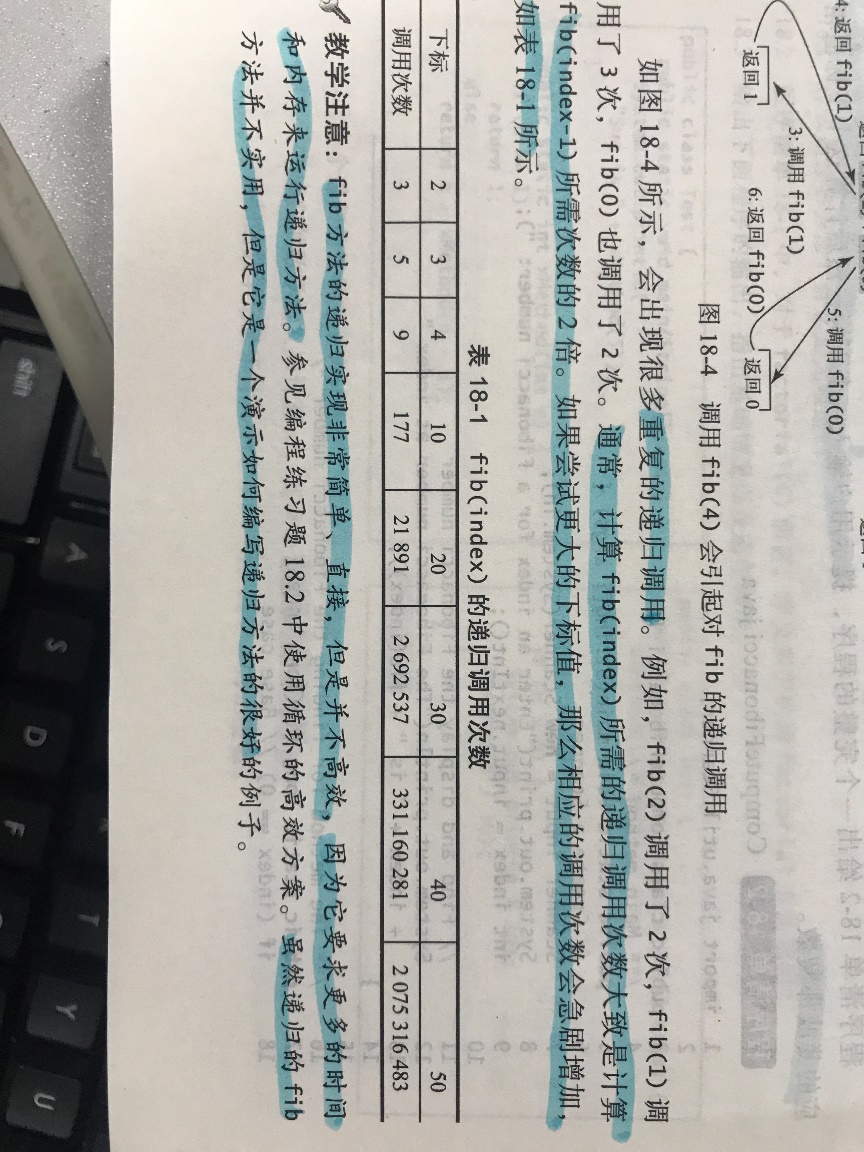
2.以下是示例程序：



3.以下的箭头表示调用的顺序：



如图所示，会发现有很多重复的递归调用。通常，计算fib(index)所需的递归调用次数大致是计算fib(index-1)所需要次数的两倍。如果尝试更大的下标值，那么相应的调用次数会急剧增加，如下表所示：



4.注意：fib方法的递归实现非常简单、直接，但是并不高效，因为它要求更多的时间和内存来运行递归方法。虽然递归的fib方法并不实用，但是它是一个演示如何编写递归方法的很好的例子。

## 18.4使用递归解决问题

1.递归方法具有以下特点：

①这些方法使用if-else语句或switch语句来引导不同的情况。

②一个或多个基础情况（最简单的情况）用来停止递归。

③每次递归调用都会简化原始问题，让它不断接近基础情况，直到它变成这种基础情况为止。

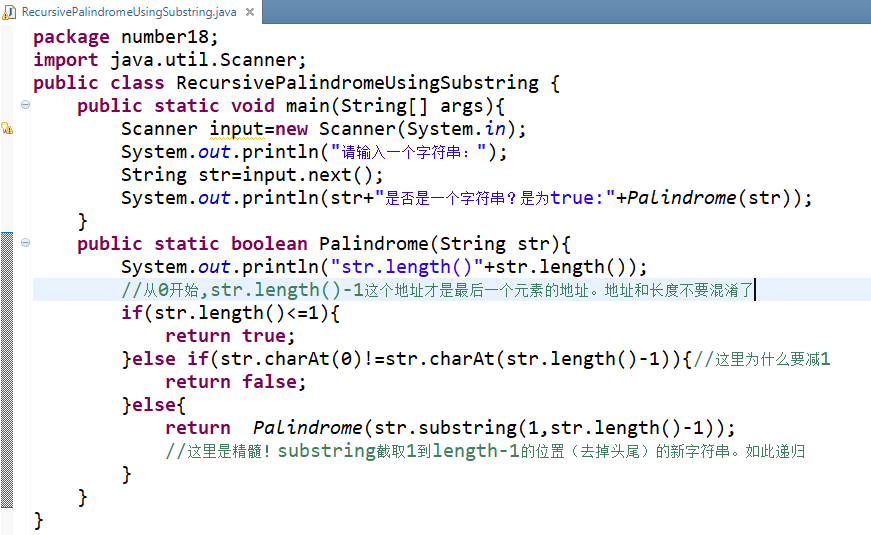
通常，要使用递归解决问题，就要将这个问题分解为子问题。每个子问题几乎与原始问题是一样的，只是规模小一点。可以使用相同的方法来递归解决问题。

2.如果一个字符串从左到右和从右到左是一样的，那么它就是一个回文串。例如：mom和dad都是回文串。检查一个字符串是否为回文串的问题可以分解为两个子问题：

①检查字符串中第一个字符和最后一个字符是否相等；

②忽略两端的字符之后检查子串的其余部分是否是回文串；

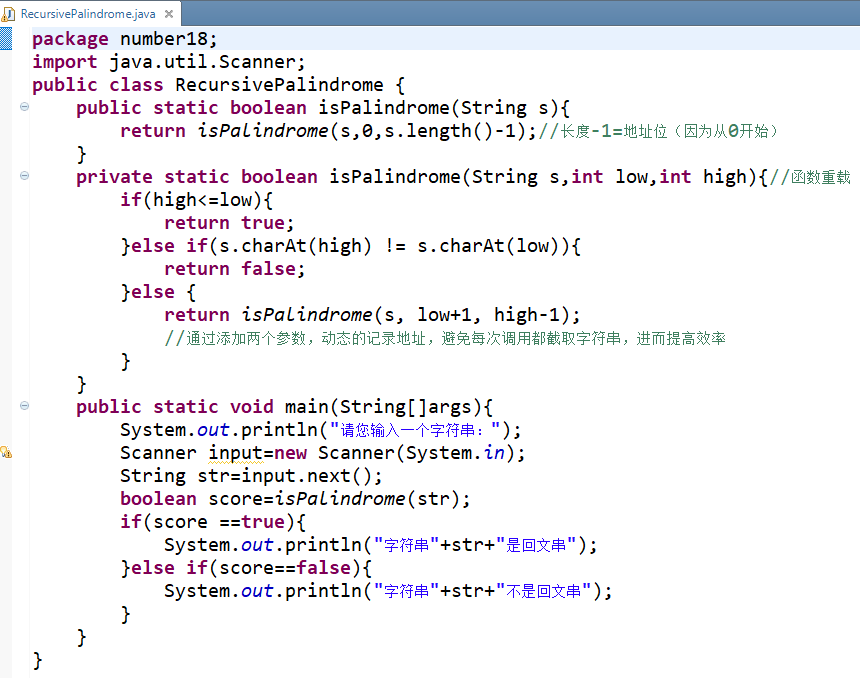
以下是源程序：



## 18.5递归辅助方法

1.要点提示：有时候可以通过针对要解决的初始问题的类似问题定义一个递归方法，来找初始问题的解决方法。这个新方法称为“递归辅助方法”。初始问题可以通过调用递归辅助方法来得到解决。

2.因为以上程序为每次递归调用都要创建一个新字符串，因此它不够高效。为避免创建新的字符串，可以使用low和high下标来表明子串的范围。这两个下标必须传递给递归方法。由于原始方法是Pllindrome（String str），因此必须产生一个新方法来isPalindrome（String s,int low ,int high）来接收关于字符串的额外信息，如下程序所示：



程序中定义了两个重载isPalindrome方法。第一个方法isPalindrome(String s)检查一个字符串是否是回文串，而第二个方法isPalindrome(String s,int low,int high)检查一个子串s(low..high)是否是回文串。第一个方法将low=0和high=s.length()-1的字符串s传递给第二个方法。第二个方法采用递归调用，检查不断缩减的子串是否为回文串。在递归程序设计中定义第二个方法来接收附加的参数是一个常用的设计技巧，这样的方法称为“递归辅助方法”。

### 18.5.1递归选择排序

1.选择排序法是先找到列表中的最小数并和第一个元素交换。然后，在剩余的数中找到最小数，再将它和剩余列表中的第一个元素交换，这样的过程进行下去，直到表中仅剩一个数为止。

2.可以将递归选择排序分解为两个子问题：（这个完全可以仿照上面的程序开写）

①找出列表中的最小数，然后将它与第一个数交换。

②忽略第一个数，对余下的较小一些的列表进行递归排序。



说明：程序中定义了两个重载的sort方法。第一个方法sort（double[ ] list）对数组list[0..list.length-1]进行排序，而第二个方法sort[double[ ]list,int low,int high]对数组list[low..high]进行排序。第二个方法采用递归调用，对不断变小的数组进行排序。

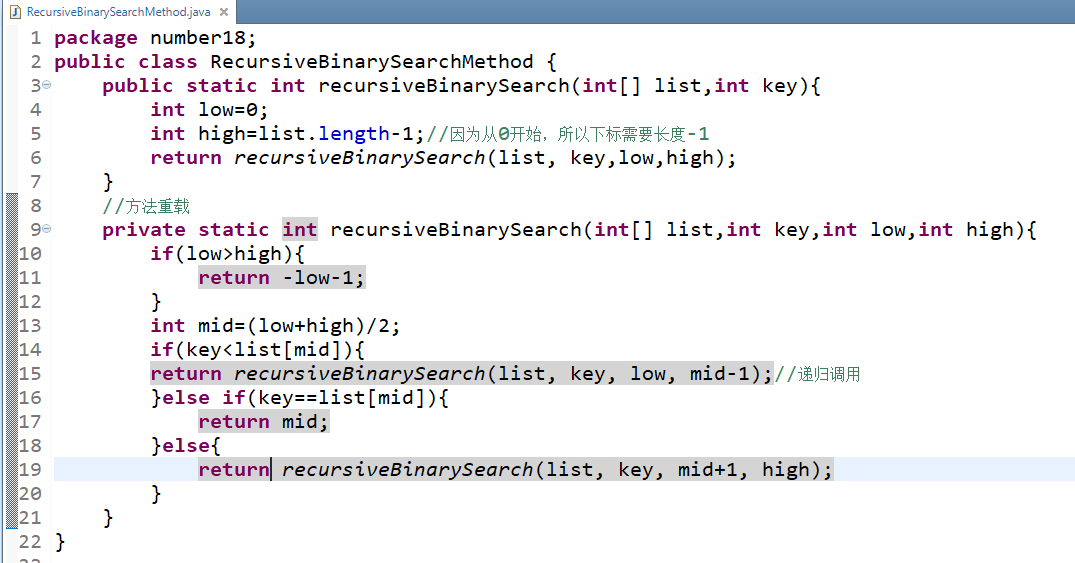
### 18.5.2递归二分查找

1.使用二分查找的前提条件是数组元素必须已经排序。二分查找首先将关键字与数组的中间元素进行比较，考虑下面三种情况

情况1：如果关键字比中间元素小，那么只需要在前一半数组元素中进行递归查找；

情况2：如果关键字和中间元素相等，则匹配成功，查找结束。

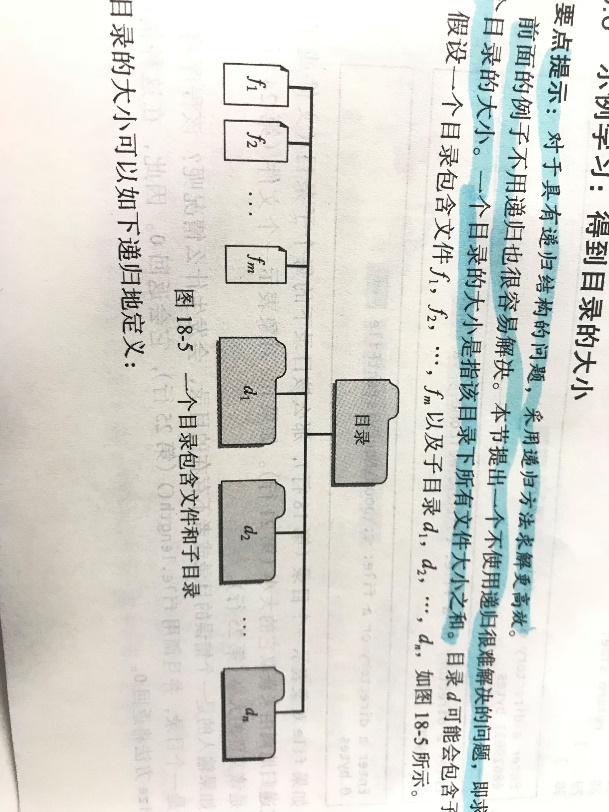
情况3：如果关键字比中间元素大，那么只需要在后一半数组元素中进行递归查找。



## 18.6示例学习：得到目录的大小

1.要点提示：对于具有递归结构的问题，采用递归的方法求解更高效。

2.前面的例子不用递归也很好解决，下面提出一个不用递归很难解决的问题，即求出一个目录的大小。一个目录的大小是指该目录下所有文件的大小之和。



File类中的两个方法对这个问题是很有用的：

①length( )方法返回一个文件的大小；

②listFiles( )方法返回一个目录下的File对象构成的数组；

下面给出一个程序，提示用户给出一个目录或者文件，然后显示它的大小：



## 18.7示例学习：汉诺塔

1.要点提示：汉诺塔问题是一个经典的递归例子。用递归可以很容易的解决这个问题，但是不使用递归则很难解决。

这个问题是将指定个数却大小不同的盘子从一个塔移动到另一个塔，移动要遵循下面的规则：

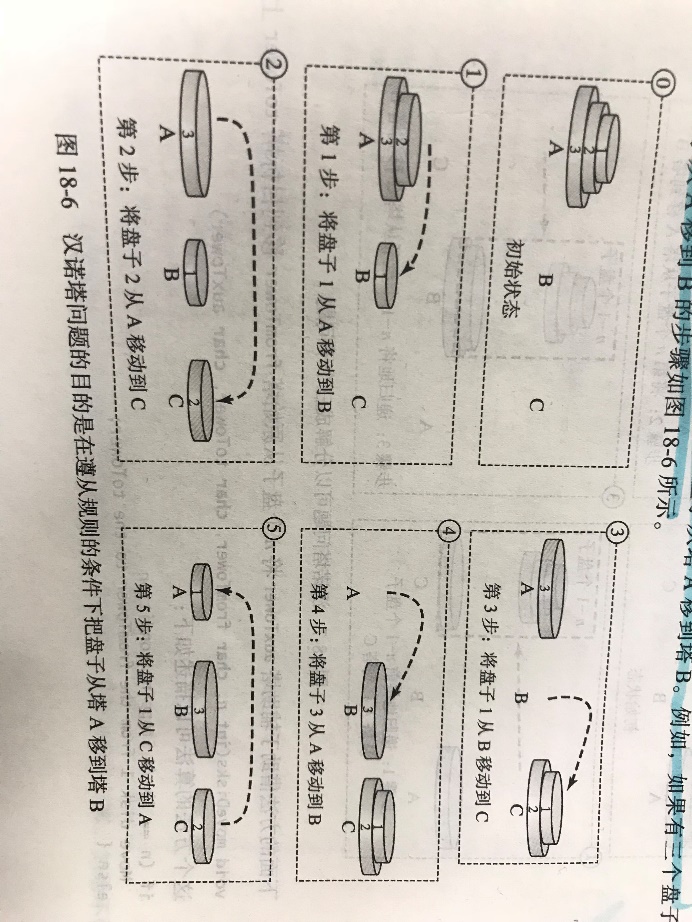
①n个盘子标记为1,2…n，而三个塔标记为A、B和C

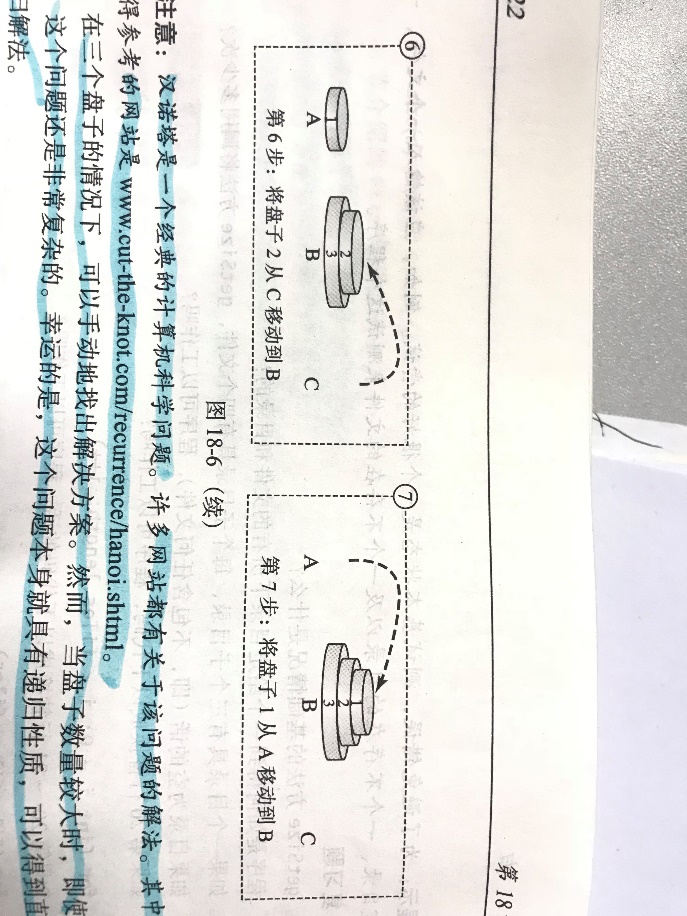
②任何时候盘子都不能放在比它小的盘子上面

③初始状态所有的盘子都放在A上

④每次只能移动一个盘子，并且这个盘子必须在塔顶位置

这个问题的目标是借助塔C把所有的盘子从A移动到B，如图所示：





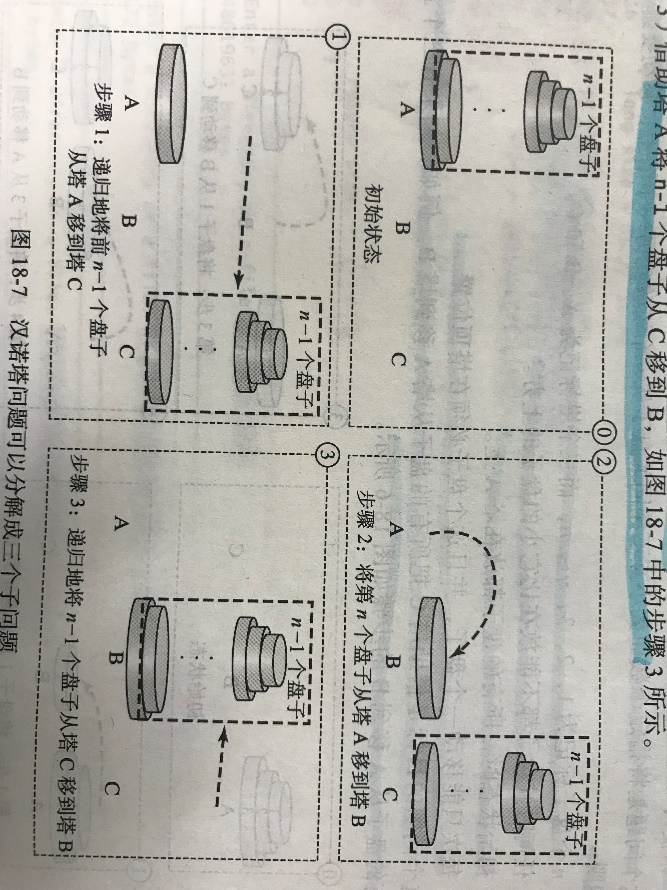
2.汉诺塔是一个经典的计算机科学问题。在三个盘子的情况下，可以手动的找出解决方案。然而，当盘子数量较大时，即使是四个，这个问题也会非常复杂。幸运的是，这个问题本身具备递归性质，可以得到直观的递归解法。

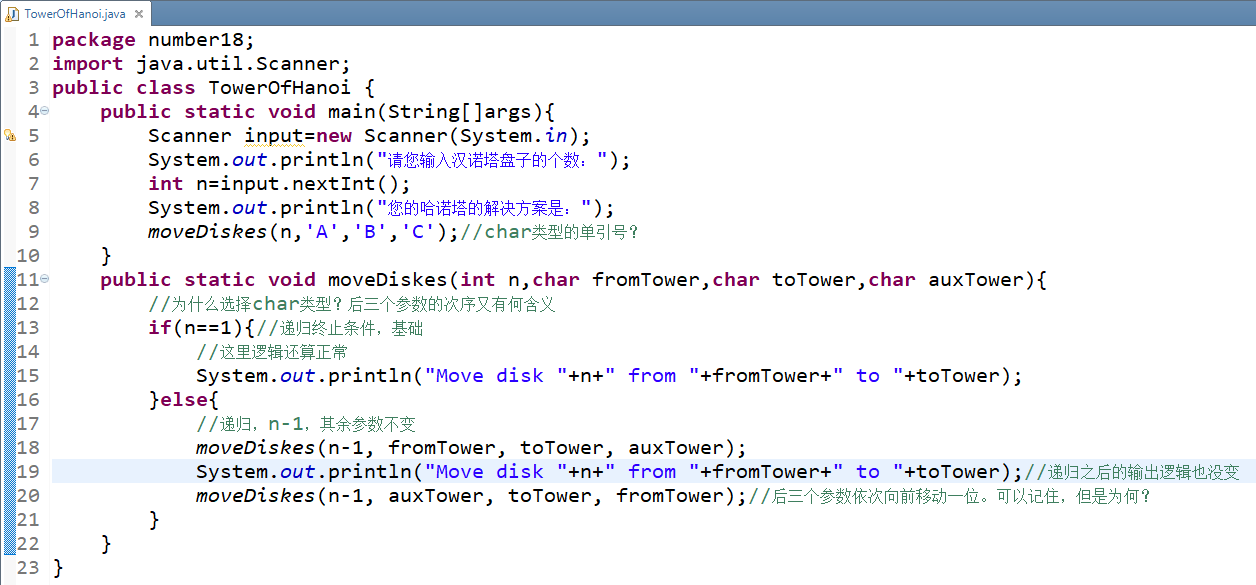
问题的基础情况是n=1。若n=1，就可以简单的把盘子从A移动到B。当n>1时，可以将原始问题拆解为下面三个子问题，然后依次解决。

①借助塔B将前n-1个盘子从A移动到C，如步骤1

②将盘子n从A移动到B，如步骤2

③借助塔A将n-1个盘子从C移到B，如步骤3





## 18.9递归与迭代

1.要点提示：递归是程序控制的另一种形式，实质上就是不同循环控制的重复。使用循环时，可以制定一个循环体。循环控制结构控制循环体的重复。在递归中，方法重复调用自己。必须使用一条选择语句来控制是否继续调用递归方法。

2.迭代与递归的区别？何时采用递归或者迭代？

任何用递归解决的问题都可以用非递归的迭代来解决。递归有很多副作用：它消耗了太多的时间并占用了太多的内存。那么为什么还要用它呢？因为在某些情况下，本质上有递归特性的问题很难用其他方法来解决，而递归可以给出一个清晰、简单的解决方案。像目录大小问题、汉诺塔和分形问题的例子都是不使用递归就很难解决的问题。应该根据解决问题的本质和我们对这个问题的理解来决定是用递归还是用迭代。根据经验，选择使用递归还是迭代的原则，就是看它能否给出一个反应问题本质的直观解法。如果迭代的解决方案是显而易见的，那就使用迭代。迭代通常比递归更高效。

3.递归会产生相当大的系统开销。程序每调用一个方法，系统就要给方法中所有的局部变量和参数分配空间。这就要占用大量的内存，还需要额外的时间来管理内存。

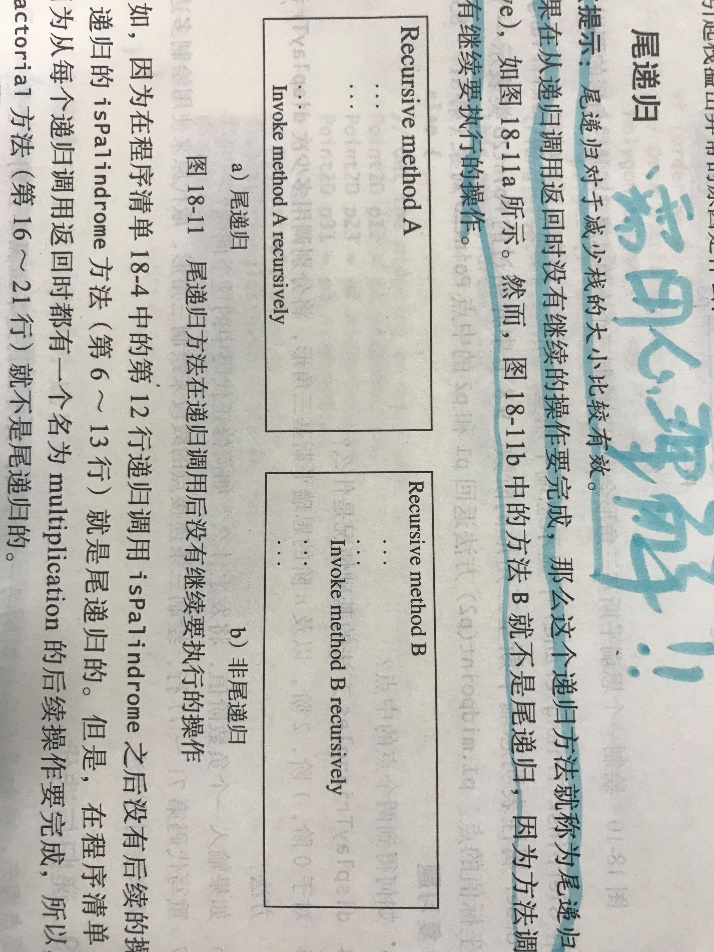
4.注意：递归程序可能会用完内存，引起一个StackOverflowError错误。

5.提示：如果关注程序的性能，就要避免使用递归，因为它会比迭代占用更多的时间且浪费更多的内存。通常，递归用于解决本质上有递归特性的问题，例如汉诺塔问题、递归目录、以及斯瑞平斯基三角形。

## 18.10尾递归

1.要点提示：尾递归对于减少栈的大小比较有效。

2.如果从递归调用返回时没有继续的操作要完成，那么这个递归方法就称为“尾递归”，如图a所示。然而，图b中的方法b就不是尾递归，因为方法调用返回后还有继续要执行的操作。



尾递归更可取：因为当最后一个递归调用结束时，方法也结束，因此，无需将中间的调用存储在栈中。编译器可以优化尾递归以减小栈空间。

通常，可以使用辅助参数将非尾递归的方法转换为尾递归方法。这些参数被用于保存结果。思路是将后续的操作以一种方式结合到辅助参数中，这样递归调用中将不再有后续操作。可以定义一个带辅助参数的新的辅助递归方法，这个方法可以重载原始方法，具有相同的名字而标签不同。

