1.递归概念

递归就是方法自己调用自己,每次调用时传入不同的变量.递归有助于编程者解决复杂的问题,同时可以让代码变得简洁

2.递归需要遵守的重要规则

1)执行一个方法时，就创建一个新的受保护的独立空间(栈空间)

2)方法的局部变量是独立的，不会相互影响，比如n变量

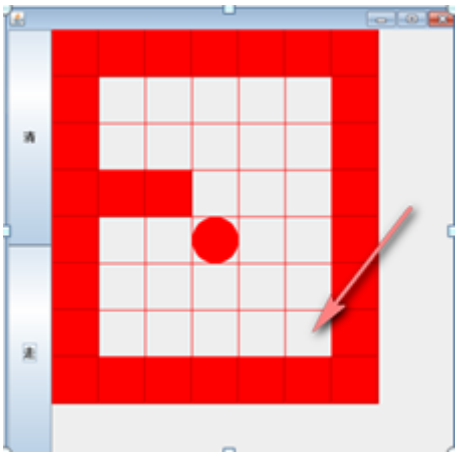
3)如果方法中使用的是引用类型变量(比如数组)，就会共享该引用类型的数据.

4)递归必须向退出递归的条件逼近，否则就是无限递归,出现StackOverflowError，死龟了:)

5)当一个方法执行完毕，或者遇到return，就会返回，遵守谁调用，就将结果返回给谁，同时当方法执行完毕或者返回时，该方法也就执行完毕。

3.迷宫问题

1.问题描述



小球从左上走到右下

2.简单实现

代码实现(注释展示细节)

```
package recuision.迷宫问题;

/**
 * @author jndeng
 * @create 2019-12-16 20:01
 */
public class Maze {
    public static void main(String[] args) {
        // 先创建一个二维数组，模拟迷宫
        int[][] map = new int[8][7];
        // 使用1 表示墙
        // 上下全部置为1
        for (int i = 0; i < 7; i++) {
            map[0][i] = 1;
            map[7][i] = 1;
        }

        // 左右全部置为1
        for (int i = 0; i < 8; i++) {
            map[i][0] = 1;
            map[i][6] = 1;
        }
        //设置挡板, 1 表示
        map[3][1] = 1;
        map[3][2] = 1;
//        map[1][3] = 1;
//        map[2][3] = 1;
        // 输出原地图
        System.out.println("地图的情况");
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 7; j++) {
                System.out.print(map[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println(setWay(map, 1, 1));

        show(map);


    }

    public static void show(int[][] map) {
        System.out.println("地图的情况");
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 7; j++) {
                System.out.print(map[i][j] + " ");
```

```
            }
            System.out.println();
        }
    }
    /**
     * 终点为 右下角
     * 使用递归回溯来给小球找路
     * 说明
     * 1. map 表示地图
     * 2. i,j 表示从地图的哪个位置开始出发 (1,1)
     * 3. 如果小球能到 map[6][5] 位置，则说明通路找到.
     * 4. 约定： 当map[i][j] 为 0 表示该点没有走过 当为 1 表示墙 ； 2 表示通路可以走；
3 表示该点已经走过，但是走不通
     * 5. 在走迷宫时，需要确定一个策略(方法) 下->右->上->左 , 如果该点走不通，再回溯
     * @param map 地图
     * @param i 出发点横坐标
     * @param j 出发点纵坐标
     * @return
     */
    public static boolean setWay(int[][] map, int i, int j) {
        if (map[6][5] == 2) {
            return true;
        } else {
            if (map[i][j] == 0) {
                map[i][j] = 2;
                if (setWay(map, i + 1, j)) {
                    return true;
                } else if (setWay(map, i, j + 1)) {
                    return true;
                } else if (setWay(map, i - 1, j)) {
                    return true;
                } else if (setWay(map, i, j - 1)) {
                    return true;
                } else {
                    map[i][j] = 3;
                    return false;
                }
            } else {
                return false;
            }
        }
    }
}
```

运行结果:

地图的情况
1 1 1 1 1 1 1
1 2 0 0 0 0 1
1 2 2 2 0 0 1
1 1 1 2 0 0 1
1 0 0 2 0 0 1
1 0 0 2 0 0 1
1 0 0 2 2 2 1
1 1 1 1 1 1 1

3.八皇后问题

1.问题说明

皇后问题，是一个古老而著名的问题，是回溯算法的典型案例。该问题是国际西洋棋棋手马克斯·贝瑟尔于1848年提出：在8×8格的国际象棋上摆放八个皇后，使其不能互相攻击，即：任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。【92】

2.解决思路

1)第一个皇后先放第一行第一列

2)第二个皇后放在第二行第一列、然后判断是否OK， 如果不OK，继续放在第二列、第三列、依次把所有列都放完，找到一个合适

3)继续第三个皇后，还是第一列、第二列……直到第8个皇后也能放在一个不冲突的位置，算是找到了一个正确解

4)当得到一个正确解时，在栈回退到上一个栈时，就会开始回溯，即将第一个皇后，放到第一列的所有正确解，全部得到.

5)然后回头继续第一个皇后放第二列，后面继续循环执行 1,2,3,4的步骤

可以用一个一维数组来模拟

3.代码

回溯的实现,回溯是靠每一列的for循环都会全部进行,因此,当下一列完成了所有尝试返回的的时候,会继续for循环,因此,会实现回溯

```
package recuision.八皇后问题;
import com.sun.corba.se.impl.orbutil.CorbaResourceUtil;
/**
 * @author jndeng
 * @create 2019-12-16 21:14
 */
public class Demo {
    static final int max = 8;
    int[] a = new int[max];
    int count = 0;
    public static void main(String[] args) {
```

```
        Demo demo = new Demo();
        demo.check(0);
        System.out.println(demo.count);
    }
    public void check(int n) {
        //递归出口,并且输出所有结果
        if (n == max) {
            show(a);
            return;
        }
        //每一行都会进行全部循环,保证了回溯
        for (int i = 0; i < max; i++) {
            //对于当前列的位置赋值
            a[n] = i + 1;
            //flag标识当前位置是否符合规则,符合规则为true,不符合为false,
            //初始化为true,后面进行检验,检验到不符合置false
            boolean flag = true;
            //从第二行开始判断,第一行不需要判断
            if (n > 0) {
                //对于之前每一行进行检查
                for (int j = n - 1; j > -1; j--) {
                    //已经保证了不会在同一行,只需要检查对角线和每一列
                    //a[j] == i + 1 检查列
                    //Math.abs(j-n)==Math.abs(a[j]-a[n])检查对角线
                    if (a[j] == i + 1||Math.abs(j-n)==Math.abs(a[j]-a[n])) {
                        flag = false;
                        //一旦发现不符合即停止检查
                        break;
                    }
                }
            }
            //都符合要求,通过递归进行下一列
            if (flag) {
                check(n + 1);
            }
        }
    }

    public void show(int[] a) {
        count++;
        for (int i = 0; i < 8; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println();
    }
}
```