

Java实现数独游戏

介绍

九宫格的数独游戏，规则简单却变化无穷。即便你没玩过肯定也听说过，今天我们就使用Java实现数独游戏的核心算法，在感受数独的魅力中通过Java编写桌面程序完成数独计算器，完成对Java基础知识点的学习和巩固。

“数独 sudoku”来自日文，但概念源自“拉丁方块”，是十八世纪瑞士数学家欧拉发明的。游戏规则很简单：在九个九宫格里，填入 1 到 9 的数字，让每个数字在每个行、列及九宫格里都只出现一次。谜题中会预先填入 若干数字，其它宫位则留白，玩家得依谜题中的数字分布状况，逻辑推敲出剩下的空格里是什么数字。

这种风靡日本及欧美的“数独 sudoku”，据说原创者是18世纪的瑞士人，但没有得到应有的关注，直到 20 多年 前，美国人重新挖掘它的魅力，接着日本杂志出版商在八〇年代末期在一本美国杂志上看到这个游戏，带回日本后，增加它的游戏难度，并命名为“数独 sudoku”，“数独”谜戏就此诞生，并逐渐受到日本人的注意、沉迷，日本坊间书局还出版了许多“数独”的书。纽西兰裔英籍退休法官韦恩·古德（Wayne Gould）一九九七年旅游日本时，买了一本数独游戏书，从此就迷上了，进而研究出计算机程序，之后开始供稿给全球十几家报社，立即受到读者的热烈回响，邀他供稿的媒体还正不断增加中；据说，“数独”还成为英国报纸销售量的法宝，连美国纽约时报也无法阻挡它的魅力，开始定期登载。1994年5月30日起，台湾的中国时报也取得古德的授权，每天都刊出一则数独谜题，让这个新玩意第一次出现在台湾的大众媒体上，也是全球第一家引入数独游戏的中文报纸。

方格里摆几个数字，乍看之下好像没什么。但数独好玩之处，就在其中推推敲敲的过程，以及解答出来的成就感。自从台湾引进数独后，玩过的人都说好玩，除非根本没玩过，否则没有听过玩过之后觉得不好玩的。由于规则简单，却变化无穷，在推敲之中完全不必用到数学计算，只需运用逻辑推理能力，所以无论老少中青男女，人人都可以玩。而且容易入手、容易入迷，一玩就上瘾。只需九个九宫格，及 1 到 9 不重复的阿拉伯数字，也超越了文字的障碍，因此自从出现后，从东方到西方，风靡亿万人。有些人认为玩数独是他们缓解工作压力的最佳方式；有些人认为玩数独可以保持头脑灵活，尤其适合老年人；也有些老师和父母觉得玩数独需要耐心、专心和推理能力，所以拿数独当题目出给学生练习，用来训练小孩子。最近英国政府出资的“教师”杂志甚至建议把“数独”引进课堂，因为数独不仅有趣好玩，还可以增进玩者的推理与逻辑机能，所以可以作为学生锻炼脑力的教材喔！

				1	4		8	
7		8						
					8			
5		2						
			6					
						1		8
						9		
	6							
			9	3				

填制规则

数独的游戏规则很简单：在九个九宫格里，填入 1 到 9 的数字，让每个数字在每个行、列及九宫格里都只出现一次就可以过关了。

解谜技巧

数独的解谜技巧，可大分为直观法及候选数法两种。

直观法的特性：

1. 不需任何辅助工具就可应用。所以要玩报章杂志上的数独谜题时，只要有一枝笔就可以开始了。
2. 从接到数独谜题的那一刻起就可以立即开始解题。
3. 初学者或没有计算机辅助时的首要解题方法。
4. 相对而言，能解出的谜题较简单。
5. 主要的技巧：唯一解法、基础摒除法、区块摒除法、唯余解法、矩形摒除法、单元摒除法。

候选数法的特性：

1. 需先建立候选数列表，所以要玩报章杂志上的数独谜题时，因篇幅的影响通常格子不会太大，且候选数列表的建立十分繁琐，所以常需计算机辅助，或使用候选数法的辅助解题用纸。
2. 需先建立候选数列表，所以从接到数独谜题的那一刻起，需经过一段相当的时间才会出现第 1 个解。
3. 需使用高阶直观法技巧或有计算机辅助时的首要解题方法。
4. 相对而言，能解出的谜题较复杂。

我们的算法就是根据候选数法来设计的。

这里先提一个问题，下图的数独号称最难的题目，你能快速完成吗？

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

核心算法介绍与实现

对于数独游戏的研究，我们不免要研究数独游戏的完全解的生成算法。对于完全解的生成过程，我们一般是采用回溯法来产生整个九宫格的所有数据。而对于九九八十一格的数独游戏完整解生成，我们尝试按常规的回溯方法来实现，不免会出现回溯的解空间过于庞大而导致回溯的时间过长而无法满足游戏中我们产生游戏完全解的需要。

为此，我们需要一种高效的完全解的生成算法，以满足我们在数独游戏中快速的产生完全解。对此，我们需要对数独进行进一步的探索和研究。

由于代码不多，不需要 IDE，推荐大家使用 VIM。

打开终端，编写 Sudoku.java:

```
Terminal 终端 - shiyanlou@cc75ac6ea50a: ~
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shiyanlou:~/ $ vim Sudoku.java [20:19:56]
```

编写 Calculate 类

Calculate 类是本算法中最重要的部分，为了便于使用，很多函数和变量都声明为静态的。在类中，我们将完成数独空格的填充和计算。而且 Calculate 类实现了 Runnable 接口：

步骤：

1. 定义实现 Runnable 接口。
2. 覆盖 Runnable 接口中的 run 方法，将线程要运行的代码存放在 run 方法中。

3. 通过 Thread 类建立线程对象。
4. 将 Runnable 接口的子类对象作为实际参数传递给 Thread 类的构造函数。
5. 调用 Thread 类的 start 方法开启线程并调用 Runnable 接口子类 run 方法。

先介绍一下 Calculate 类中所有的成员：

```
class Calculate implements Runnable {
    // boo用于判断该格是否为空
    public static boolean[][] boo = new boolean[9][9];

    //计算指定行的值
    public static int upRow = 0;

    //计算指定列值
    public static int upColumn = 0;

    //将存储九宫格中的数据
    public static int[][] b = new int[9][9];

    //查找没有填入数值的空格
    public static void flyBack(boolean[][] judge,int row,int column){}

    //遍历所有可能的值
    public static void arrayAdd(ArrayList<Integer> array,TreeSet<Integer> tree){}
    public static ArrayList<Integer> assume(int row,int column){}

    //添加每格可能的选项
    public void run(){

    //分析九宫格是否完成
    public void judge(){

    }
}
```

分析：

- 二维数组 boo 用于判断该格是否为空。如果已经填入了数值，就不要再填了。
- 二维数据 b 将存储九宫格中的数据。
- flyBack 函数用于查找没有填入数值的空格。
- arrayAdd 函数添加新的数值（1~9）到一行中。如果数据已经有了，跳过，没有就继续赋值。
- assume 主要是判断在同行同列同一个小九宫格内哪些数值没有被填充，添加备选的数字，就是候选法的思想。
- run 函数开始运行整个程序，生成最后的结果。

编写 flyBack() 函数

```
public static void flyBack(boolean[][] judge, int row, int column) {
    // 生成临时变量s，具体下面会介绍
    int s = column * 9 + row;
    s--;

    // 取商的值，实际就是column的值
```

```

int quotient = s / 9;

// 取余数的值，实际是取(row-1)%9
int remainder = s % 9;

// 判断是否满足条件
if (judge[remainder][quotient]) {
    flyBack(judge, remainder, quotient);
} else {
    // 赋值给upRow
    upRow = remainder;
    // 赋值给upColumn
    upColumn = quotient;
}
}
}

```

我们分析一下这段代码：

quotient 是指商，remainder 是指余数，

```

//此处为伪代码
s=column*9+row
s--=(column*9+row)-1
quotient = s/9
    = ((column*9+row)-1)/9
    = column + (row-1)/9 // 因为row-1<9，这里的除法只保留整数部分
    = column
remainder=s%9
    = ((column*9+row)-1)%9
    = (row-1)%9 //column*9能被9整除

```

分析了之后可以看出，函数的作用是计算同列的上一行元素值。如果它为空，就赋值给 upRow 和 upColumn，如果依然满足条件，就继续递归。

这段代码看上去很麻烦，实际上就是找还没有填入数值的空格。

编写 arrayAdd() 函数

arrayAdd() 填充某行的值，遍历所有可能的值。

```

public static void arrayAdd(ArrayList<Integer> array, TreeSet<Integer> tree) {
    // 遍历1~10
    for (int z = 1; z < 10; z++) {
        // flag3默认为true，判断z是否符合条件
        boolean flag3 = true;

        // 它就是一个迭代器
        Iterator<Integer> it = tree.iterator();

        // tree如果没有遍历完继续遍历
        while (it.hasNext()) {
            // 将列表中的值赋给b
            int b = it.next().intValue();
            if (z == b) {
                flag3 = false;
            }
        }
    }
}

```

```

        break;
    }
}

// 如果判断z没有出现在过tree中，就将它添加进去
if (flag3) {
    array.add(new Integer(z));
}

// 初始化flag3
flag3 = true;
}
}

```

由于数独的规则，每行每列每个小九宫格 1 ~ 9 不能重复，所以填写 arrayAdd() 函数来添加 tree 中没有的元素，如果有了就跳过。

编写 assume() 函数

这个函数的作用是分析每个格子可能的解，并将其放到数组中，之后 run() 函数会调用它来分析可能的解。

```

public static ArrayList<Integer> assume(int row, int column) {
    // 创建数组array
    ArrayList<Integer> array = new ArrayList<Integer>();
    TreeSet<Integer> tree = new TreeSet<Integer>();

    // 添加同一列其他的元素值
    for (int a = 0; a < 9; a++) {

        // 如果该格不为空，就添加到tree中
        if (a != column && b[row][a] != 0) {
            tree.add(new Integer(b[row][a]));
        }
    }

    // 添加同行的其他元素
    for (int c = 0; c < 9; c++) {

        // 如果该格满足添加，就添加到tree中
        if (c != row && b[c][column] != 0) {
            tree.add(new Integer(b[c][column]));
        }
    }

    // 这里使用了整型除法只保留整数部分的特点，获取元素在同一个九宫格的行
    for (int a = (row/3)*3; a < (row/3+1)*3; a++)
    {
        // 获取元素在同一个九宫格的列
        for (int c = (column/3)*3; c < (column/3 + 1)*3; c++) {
            // 如果元素满足条件都添加到tree中
            if ((!(a == row && c == column)) && b[a][c] != 0) {
                tree.add(new Integer(b[a][c]));
            }
        }
    }
}

```

```

    }
    arrayAdd(array, tree);
    return array;
}

```

为了提高算法的效率，我们将大九宫格分成 9 个小九宫格，主要是分析在同行同列同一个小九宫格内哪些数值已经被填充了，然后地调用 arrayAdd() 函数，添加该格备选的数值。

编写 run() 函数

run() 就是用来赋值计算的，然后调用 arrayAdd() 函数和 assume() 函数来判断。

```

public void run() {
    // 初始化变量行，列
    int row = 0, column = 0;

    // flag用来判断该格子是否填入正确
    boolean flag = true;
    for (int a = 0; a < 9; a++) {
        for (int c = 0; c < 9; c++) {
            if (b[a][c] != 0) {
                /* boo的作用是找出填入数据的空格，
                * 填入数据的空格是谜面，我们需要根据这些信息解谜题
                */
                boo[a][c] = true;
            } else {
                // 为空的格子是需要填入数据的部分
                boo[a][c] = false;
            }
        }
    }

    /* arraylist是一个二维的序列，它的每一个值都是一个数组指针，
    * 存放了该格子可能的解，当一个解错误时，调用下一个解，
    * 这也就是前面介绍的数独解法。
    */
    ArrayList<Integer>[][] utilization = new ArrayList[9][9];

    while (column < 9) {
        if (flag == true) {
            row = 0;
        }
        while (row < 9) {
            if (b[row][column] == 0) {
                if (flag) {
                    ArrayList<Integer> list = assume(row, column); //
                    utilization[row][column] = list;
                }

                // 如果没有找到可能的解，说明前面的值有错误，就回溯到之前的格子进行修改
                if (utilization[row][column].isEmpty()) {
                    // 调用flyBack函数寻找合适的row和column
                    flyBack(boo, row, column);

                    // 将row返回到合适的位子

```



```

        row = upRow;

        // 将column返回到合适的位子
        column = upColumn;

        // 初始化有问题的格子
        b[row][column] = 0;
        column--;
        flag = false;
        break;
    } else {
        // 将备选数组中第一个值赋给b
        b[row][column] = utilization[row][column].get(0);

        // 因为上面已经赋值了，所以就删除掉第一个数值
        utilization[row][column].remove(0);
        flag = true;

        //判断是否所有的格子都填入正确，然后将正确的结果输出到屏幕上
        judge();
    }
} else {
    // 如果r为false，说明还有格子没填入数据，就继续遍历
    flag = true;
}
row++;
}
column++;
}
}

```

在 run() 函数中填写空格的地方，我们的想法是将一行一行的分析，每个点都可能有几个值，我们用一个数组 utilization 来存放所有可能的值，在这个值的基础上填写下一个空格，当填写不动的时候回溯到这里，填写为 utilization 数组里的下一个值。

编写判断函数judge()

```

public void judge()
{
    boolean r = true;

    // 查找还没有填入数据的格子
    for (int a1 = 0; a1 < 9; a1++) {
        for (int b1 = 0; b1 < 9; b1++) {
            if (r == false) {
                break;
            }

            // 如果 b[a1][b1] 需要计算，就将其提取出来
            if (b[a1][b1] == 0) {
                r = false;
            }
        }
    }
}

```



```
// 如果r为true，则所有的格子都填入了数据，说明九宫格就完成了，此时输出结果到屏幕上
if (r) {
    for (int a1 = 0; a1 < 9; a1++) {
        for (int b1 = 0; b1 < 9; b1++) {
            Myframe.filed[a1][b1].setText(b[a1][b1] + "");
        }
    }
}
}
```

编写界面

编写 MyFrame 类

由于我们的结构是九个 TextField 外加两个 button 控件，计算，关闭。基于这些需求，所以编写的内容如下：

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

// 继承界面类
class Myframe extends JFrame {
    public static Object obj = new Object();
    // 创建九宫格界面
    public final static JTextField[][] filed = new JTextField[9][9];

    public Myframe() {
        // 初始化界面，让所有的格子都等于空
        for (int a = 0; a < 9; a++) {
            for (int b = 0; b < 9; b++) {
                filed[a][b] = new JTextField();
                filed[a][b].setText("");
            }
        }

        // 编写布局，把textfield添加到布局中
        JPanel jpan = new JPanel();
        jpan.setLayout(new GridLayout(9, 9));
        for (int a = 8; a > -1; a--) {
            for (int b = 0; b < 9; b++) {
                jpan.add(filed[b][a]);
            }
        }
    }
}
```

```

// 界面布局为居中
add(jpan, BorderLayout.CENTER);
JPanel jpb = new JPanel();

// 设置两个按钮，计算和退出
JButton button1 = new JButton("calc");
JButton button2 = new JButton("close");

// 将按钮添加到界面上
jpb.add(button1);
jpb.add(button2);

// 给按钮添加监听器，就是添加事件响应函数
button1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        synchronized (obj) {
            for (int a = 0; a < 9; a++) {
                for (int b3 = 0; b3 < 9; b3++) {
                    int pp = 0;
                    // 获取九宫格中的已填入数据的值，这些就是谜面
                    if (!filed[a][b3].getText().trim().equals("")) {
                        pp = Integer.parseInt(filed[a][b3].getText().trim());
                        Calculate.b[a][b3] = pp;
                    }
                }
            }
        }

        synchronized (obj) {
            // 开启线程计算九宫格的答案
            new Thread(new Calculate()).start();
        }
    }
});

// button2很简单，调用api关闭程序
button2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        System.exit(0);
    }
});

// 设置界面的布局
add(jpb, BorderLayout.SOUTH);
}
}

```

我们在 button 上添加了监听器，分别用于获取界面信息，生成计算结果和关闭程序。

编写主函数

```

public class Sudoku{
    public static void main(String[] args) {
        Myframe myf=new Myframe();
    }
}

```

```

myf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//设置主界面的名称
myf.setTitle("sudoku");

//设置界面的大小
myf.setSize(500,500);

//设置主程序可见
myf.setVisible(true);
}
}

```

完整的代码如下：

<http://labfile.oss.aliyuncs.com/courses/704/Sudoku.java>

代码编写完毕后，开始编译和运行，

```

javac Sudoku.java //这里会报一个错，没有关系，直接运行
java Sudoku

```

编译效果图：

```

shiyanolou:~/ $ javac Sudoku.java [20:24:25]
注：Sudoku.java使用了未经检查或不安全的操作。
注：有关详细信息，请使用 -Xlint:unchecked 重新编译。

```

运行效果图：

```

shiyanolou:~/ $ wget http://labfile.oss.aliyuncs.com/courses/704/Sudoku.java [20:24:15]
--2016-12-25 20:24:21-- http://labfile.oss.aliyuncs.com/courses/704/Sudoku.java
正在解析主机 labfile.oss.aliyuncs.com (labfile.oss.aliyuncs.com) 10.157.220.8
正在连接 labfile.oss.aliyuncs.com:80: 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度：7208 (7.0K) [text/x-java]
正在保存至：“Sudoku.java”

100%[=====]
2016-12-25 20:24:22 (271 MB/s)

shiyanolou:~/ $ ls
Code Desktop Sudoku.java
shiyanolou:~/ $ javac Sudoku.java
注：Sudoku.java使用了未经检查或不安全的操作。
注：有关详细信息，请使用 -Xlint:unchecked 重新编译。
shiyanolou:~/ $ java Sudoku

```

用时 0s

[20:24:22]

[20:24:25]

[20:24:40]

解决实际谜题

8	4		9					
	7			4		5		1
5	1				6			8
		8	7	5		6	2	
2	9	5						
			2	9		8	5	
		3		6	7			
1	8							7
					3		1	5

将题目导入我们的程序，很快就可以得到结果了，为自己的成功鼓掌吧！

```

shiyanolou:~/ $ wget http://labfile.oss.aliyuncs.com/courses/704/Sudoku.java [20:24:15]
--2016-12-25 20:24:21-- http://labfile.oss.aliyuncs.com/courses/704/Sudoku.java
正在解析主机 labfile.oss.aliyuncs.com
正在连接 labfile.oss.aliyuncs.com (
已发出 HTTP 请求，正在等待回应... 2
长度：7208 (7.0K) [text/x-java-sou
正在保存至：“Sudoku.java”

100%[=====
2016-12-25 20:24:22 (271 MB/s) - 已
shiyanolou:~/ $ ls
Code Desktop Sudoku.java
shiyanolou:~/ $ javac Sudoku.java
注：Sudoku.java使用了未经检查或不安
注：有关详细信息，请使用 -Xlint:unc
shiyanolou:~/ $ java Sudoku
shiyanolou:~/ $ java Sudoku

```

8	4	2	9	1	5	7	3	6
3	7	6	8	4	2	5	9	1
5	1	9	3	7	6	2	4	8
4	3	8	7	5	1	6	2	9
2	9	5	6	3	8	1	7	4
7	6	1	2	9	4	8	5	3
9	5	3	1	6	7	4	8	2
1	8	4	5	2	9	3	6	7
6	2	7	4	8	3	9	1	5

本文来自蓝桥云课：

<https://www.lanqiao.cn/courses/704>

其他参考

- 其他实现-基于JavaFX和DLX算法实现的数独游戏，可快速生成和解出数独
 - <https://gitee.com/zhangwanjun/sudoku/tree/master>
 - [sudoku-master.zip](#)