概要设计

关于类的设计，需求分析中大概设计了两个类，后来觉得设置一个类应该就可以了，3×3矩阵的判断可以转化成一个二维数组。便于判定每个3×3矩阵内内行每列是否满足1-9只出现了一次，所以就除了本身的数独数组，分别设置了3个判定数组。

该类命名为Sudoku类，包含四个字段及其相应属性分别是：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 属性 | 类型 | 含义 |
| shudu | shudu | int[9,9]数组 | 数独矩阵 |
| hang | Hang | Bool [9,9]数组 | 行判定，每一行代表数独每一行，从上至下依次标号，每一列代表数独该行该数是否出现 |
| lie | Lie | Bool [9,9]数组 | 列判定，每一行代表数独每一列，从左至右依次标号，每一列代表数独该列该数是否出现 |
| sansan | sansan | Bool [9,9]数组 | 行判定，每一行代表数独每一个3\*3小矩阵，从左至右从上至下依次标号，每一列代表数独该小矩阵该数是否出现 |

包含两个函数分别是public void GenerateSudokuEnding（）和public void SolveSudoku（），分别用来生成终局和求解数独。

两个函数之间没有直接关联，是两种关于数独的功能（详细设计发现还是有调用关系的），不过内部会调用相关的字段，比如判断改行是否有重复数字的时候会查看hang，lie，sansan数组，同时会将数字写入shudu数组。

Sudoku类的数据结构大致如图所示：（以下类仅仅是概要设计，还不完全）

class Sudoku

{

public void GenerateSudokuEnding()

{

//填充部分

}

public void SovleSudoku()

{

//填充部分

}

private char[,] shudu = new char[9, 9];//数独矩阵

public char[,] Shudu

{

get { return shudu; }

set { shudu = value; }

}

private bool[,] hang = new bool[9, 9];//行检验矩阵

public bool[,] Hang

{

get { return hang; }

set { hang = value; }

}

private bool[,] lie = new bool[9, 9];//列检验矩阵

public bool[,] Lie

{

get { return lie; }

set { lie = value; }

}

private bool[,] sansan = new bool[9, 9];//3×3检验矩阵

public bool[,] Sansan

{

get { return sansan; }

set { sansan = value; }

}

}

}

软件主操作界面为控制台，所有的参数输入都通过控制台传输给主函数，由主函数来判断具体的命令操作（同时也会判断命令是否正确），根据不同命令参数去执行不同板块：

在控制台输入指令“sudoku.exe -c 有效数字”可以输出改有效数字数量的数独终局到txt文件中。

输入“sudoku.exe -s 绝对路径”可以读取绝对路径中txt文本中的数独，求解并输出结果

当判断为命令1时，会调用Sudoku类的GenerateSudokuEnding函数来生成数独；当命令为2时，会调用Sudoku类的SolveSudoku函数来求解，否则，判定为无效命令，结束程序。

同时设置了一个输出函数static void OutputToTxt(Sudoku sudoku,StreamWriter streamWriter)，统一将生成的数独输出到TXT文件中。

此时，程序的大体框架就构建出来了，接下来就是细化各个模块函数需要实现的算法，算法详细设计。