```
/*
编写一个程序，通过已填充的空格来解决数独问题。

一个数独的解法需遵循如下规则：

    数字 1-9 在每一行只能出现一次。
    数字 1-9 在每一列只能出现一次。
    数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

空白格用 '.' 表示。


Note:

    给定的数独序列只包含数字 1-9 和字符 '.' 。
    你可以假设给定的数独只有唯一解。
    给定数独永远是 9x9 形式的。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/sudoku-solver
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 由于用例告知一定至少有一个解即可,回溯法步骤.
  - 两重循环开始填写，当检测到空格就进入填写判断，先行后列，一行一行填完，遇到错误则返回上一层进行处理.
  - 遇到'.',循环遍历$1 \sim 9$,只要有一个能进就去填下一个'.',直到出现矛盾,再一次出栈$unchoose$,或者把底层循环跑完,得到一个正确的结果.
  - 优化方法:增加一个计数用来记录当前正在处理的行数,免得每次都需要从第0行第0列开始找.

方法一:C++_回溯法

```
// https://leetcode-cn.com/problems/sudoku-solver/

class Solution
{

    private:


        bool isValidc   (   char                    c       ,
                            int                     row     ,
                            int                     col     ,
                            vector<vector<char>>&   board
                        )
        {

            for(int i = 0 ; i < 9 ; i++)
            {
```

```
                // 横
                if(board[row][i]==c)
                {
                    return false;
                }
                // 竖
                if(board[i][col]==c)
                {
                    return false;
                }
                //小正方形
                if(board[(row / 3)*3 + i / 3][(col / 3) * 3 + i % 3]==c)
                {
                    return false;
                }

            }

            return true;

        }


        bool helper( vector<vector<char>>& board )
        {
            /* 每次都从第0行第0列这个点找起,防止漏点.*/
            for(int i = 0 ; i < 9 ; i++)
            {
                for(int j = 0 ; j < 9 ; j++)
                {
                    if(board[i][j]=='.')
                    {
                        for(char c = '1'; c<='9';c++)
                        {
                            if(isValidc(c,i,j,board))
                            {
                                board[i][j] = c;
                                if(helper(board))
                                    return true;
                                board[i][j] = '.';
                            }
                        }

                        return false; /*九个数没一个数能插入的,说明上一层处理得不对*/
                    }
                }

            }


            return true;
        }


    public:
        void solveSudoku(vector<vector<char>>& board)
        {
```

```
            helper(board);
        }
};
/*
执行结果:
通过
显示详情
执行用时 :20 ms, 在所有 cpp 提交中击败了43.56% 的用户
内存消耗 :8.5 MB, 在所有 cpp 提交中击败了97.42%的用户
*/
```

方法二:C++_回溯法优化方法

```
class Solution
{

    private:


        bool isvalidc   (   char                    c       ,
                            int                     row     ,
                            int                     col     ,
                            vector<vector<char>>&   board
                        )
        {

            for(int i = 0 ; i < 9 ; i++)
            {
                // 横
                if(board[row][i]==c)
                {
                    return false;
                }
                // 竖
                if(board[i][col]==c)
                {
                    return false;
                }
                //小正方形
                if(board[(row / 3)*3 + i / 3][(col / 3) * 3 + i % 3]==c)
                {
                    return false;
                }

            }

            return true;

        }


        bool helper( vector<vector<char>>& board ,int row)
        {

            /*每次不必从第0行第0列开始找*/
            for(int i = row ; i < 9 ; i++)
```

```
            {
                for(int j = 0 ; j < 9 ; j++) //  这里j不能初始化为 col ,会漏掉一堆点
                {
                    if(board[i][j]=='.')
                    {
                        for(char c = '1'; c<='9';c++)
                        {
                            if(isValidc(c,i,j,board))
                            {
                                board[i][j] = c;
                                if(helper(board,(i*9+j+1)/9))
                                    return true;
                                board[i][j] = '.';
                            }
                        }

                        return false; /*九个数没一个数能插入的,说明上一层处理得不对*/
                    }
                }

            }


            return true;
        }


    public:
        void solveSudoku(vector<vector<char>>& board)
        {
            helper(board,0);
        }
};


/*
执行结果:
通过
显示详情
执行用时 :20 ms, 在所有 cpp 提交中击败了43.56% 的用户
内存消耗 :8.7 MB, 在所有 cpp 提交中击败了83.95%的用户
*/
```

AlimyBreak
2019.12.01