一、题目

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1：

```
输入：grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
输出：1
```

示例 2：

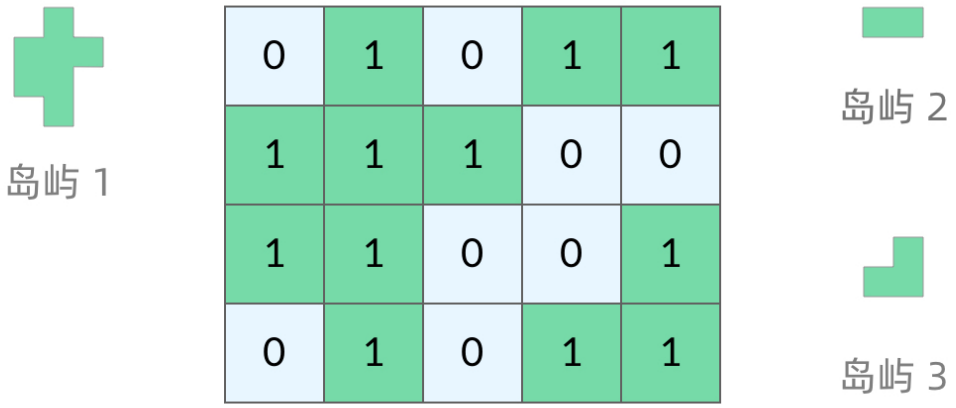
```
输入：grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
输出：3
```

提示：

```
m == grid.length
n == grid[i].length
1 <= m, n <= 300
grid[i][j] 的值为 '0' 或 '1'
```

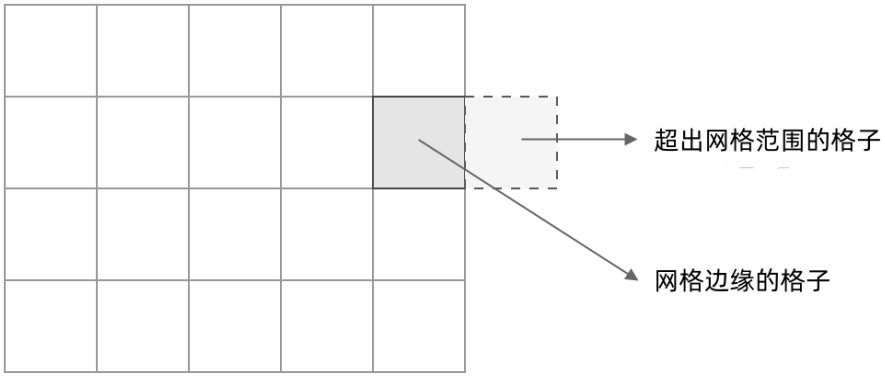
二、思路

1. 根据题意，每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成，故下面这种情况是有3个岛屿。



2. 我们可以遍历遍历这个二维网格，只要遍历到1，就说明有一个岛屿，至于这个岛屿多大，我们可以采用DFS(深度优先搜索)把整个岛屿遍历完。因为上下左右才能形成岛屿，所以我们DFS的时候只需要深度遍历上下左右相邻的格子即可。

3. 由于是上下左右遍历，所以又会出现临界条件。如下图




```
if (r < 0 || c < 0 || r >= row || c >= col) {
    return;
}
```

4. 因此在DFS时需要判断当前格子是否在网格内，不在网格内就return

```
if (grid[r][c] == '0') {
    return;
}
```

5. 由于可能会出现递归死循环，故每遍历一个陆地格子都需要将它置'0'(或者其他数字表示已经遍历过的)
6. 最后用一个变量num存储岛屿的数量即可

三、代码

```
public class LeetCode_200 {
    public static void main(String[] args) {
        char[][] grid = {{'1','1','0','0','0'},{'1','1','0','0','0'},
{'0','0','1','0','0'},{'0','0','0','1','1'}};
        System.out.println("岛屿的数量为："+numIslands(grid));//3
    }

    public static int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }
        int row = grid.length;
        int col = grid[0].length;
        int num = 0;
        for (int r = 0; r < row; r++) {
            for (int c = 0; c < col; c++) {
                if (grid[r][c] == '1') {
                    num++;
                    dfs(grid, r, c);
                }
            }
        }
        return num;
    }

    private static void dfs(char[][] grid, int r, int c) {
        int row = grid.length;
        int col = grid[0].length;
        if (r < 0 || c < 0 || r >= row || c >= col || grid[r][c] == '0') {
            return;
        }
        grid[r][c] = '0';
        dfs(grid, r - 1, c);
        dfs(grid, r + 1, c);
        dfs(grid, r, c - 1);
        dfs(grid, r, c + 1);

    }
}
```

> 注：可以通过复制以上代码到 https://tool.lu/coderunner/ 验证代码

四、复杂度

- 时间复杂度：O(MN)，其中 M 和 N 分别为行数和列数。
- 空间复杂度：O(MN)，在最坏情况下，整个网格均为陆地，深度优先搜索的深度达到 MN。