200 岛屿数量

```
Label：递归
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。
```

- 深度优先（感染）

```
class Solution {
     public int numIslands(char[][] grid) {
        int islandNum = 0;

        for(int i = 0; i < grid.length; i++){
            for(int j = 0; j < grid[0].length; j++){
                if(grid[i][j] == '1'){
                    infect(grid, i, j);
                    islandNum++;
                }
            }
        }
        return islandNum;
    }

    //感染函数（其实就是标记）
    private void infect(char[][] grid, int i, int j){
        if(i < 0 || i >= grid.length ||
                j < 0 || j >= grid[0].length || grid[i][j] != '1'){
            return;
        }
        grid[i][j] = '2';  // 因为相邻，所以感染 上、下、左、右
        infect(grid, i + 1, j);
        infect(grid, i - 1, j);
        infect(grid, i, j + 1);
        infect(grid, i, j - 1);
    }
}
```

- 广度优先

```
class Solution {
    public int numIslands(char[][] grid) {
            if (grid == null || grid.length == 0) return 0;

        int nr = grid.length;
        int nc = grid[0].length;
        int num_islands = 0;

        for (int r = 0; r < nr; ++r) {
            for (int c = 0; c < nc; ++c) {
                if (grid[r][c] == '1') {
                    ++num_islands;
                    grid[r][c] = '0'; // 先把本节点标记
                    Queue<Integer> neighbors = new LinkedList<>();
                    neighbors.add(r * nc + c); // 保存一个坐标，用到时计算就行
                    while (!neighbors.isEmpty()) {
                        int id = neighbors.remove();
                        int row = id / nc;
                        int col = id % nc;
                        // 标记所有邻居节点，并将其入队
                        if (row - 1 >= 0 && grid[row-1][col] == '1') {
                            neighbors.add((row-1) * nc + col);
                            grid[row-1][col] = '0'; //
                        }
                        if (row + 1 < nr && grid[row+1][col] == '1') {
                            neighbors.add((row+1) * nc + col);
                            grid[row+1][col] = '0';
                        }
                        if (col - 1 >= 0 && grid[row][col-1] == '1') {
                            neighbors.add(row * nc + col-1);
                            grid[row][col-1] = '0';
                        }
                        if (col + 1 < nc && grid[row][col+1] == '1') {
                            neighbors.add(row * nc + col+1);
                            grid[row][col+1] = '0';
                        }
                    }
                }
            }
        }
        return num_islands;
    }
}
```