三个核心算法： push、pop、peek

Push：往栈顶添加元素

Pop：从栈顶移除元素

Peek：获取栈顶元素

栈的初始化：

**package** com.youkeda;

**public** **class** **YKDStack** {

String[] data;

**int** size;

*// #1. 初始化栈，默认大小为20 默认初始化*

**public** **YKDStack**(){

**this**.initData(20);

}

*// #2. 初始化栈，并传入栈空间大小 自定义初始化*

**public** **YKDStack**(**int** size){

**this**.initData(size);

}

*// 初始化数组*

**private** **void** **initData**(**int** size){

**this**.size = size;

**this**.data = **new** String[**this**.size];

}

}

栈的push方法

添加一个top变量记录栈顶在数组中的位置

当栈为空的时候，top 为 -1。

当只有 1 个元素时，top 为 0。

当有 10 个元素时，top 为 9。

push的核心逻辑有三点：

1. 优先判断是否数组越界

2. 更改变量top的值

3.往数组中添加一个元素

代码如下：

*// 添加元素*

**public** **boolean** **push**(String value) {

*// 数组越界了*

**if** (**this**.top >= **this**.size - 1) {

**return** **false**;

}

*// top 栈顶 +1*

**this**.top++;

*// 设置栈顶元素*

**this**.data[**this**.top] = value;

**return** **true**;

}

栈 – 简单寻路算法

1.地图划分区域：可移动区域（‘ ’），不可移动区域(‘ # ’)，起点和终点

Char[][] map 储存整个地图信息

**char**[][] map = {

{'#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},

{'#', ' ', ' ', '#', ' ', ' ', ' ', '#', ' ', '#'},

{'#', ' ', ' ', '#', ' ', ' ', ' ', '#', ' ', '#'},

{'#', ' ', ' ', ' ', ' ', '#', '#', ' ', ' ', '#'},

{'#', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', '#'},

{'#', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', '#'},

{'#', ' ', '#', ' ', ' ', ' ', '#', ' ', ' ', '#'},

{'#', ' ', '#', '#', '#', ' ', '#', '#', ' ', '#'},

{'#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

{'#', '#', '#', '#', '#', '#', '#', '#', '#', '#'}

};

2.寻找路径

因为终点在起点右下方，所以选择 右->下->左->上 方向进行探索；

右{0，1}，下{1，0}，左{0，-1}，上{0，-1}

*// 存储4个方向*

**private** **int**[][] directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

最开始的起点在栈底，每探索一个方格，则往栈里push一个元素，如果遇到一条死路，则pop回退到上一个元素并且换个方向继续探索。

为了区别 # 和空格，用 \* 表示方格已经被访问过，用 @ 表示方格为死路。

因此我们新建一个Tile对象，表示在栈中的信息，包括方格位置和当前遍历的方向：

**public** **class** **Tile** {

*// 方格位置*

**private** Point point;

*// 方格当前遍历的方向*

**private** **int** direction = 0;

}

**public** **class** **Game** {

**private** **char**[][] map;

*// 存储探索的路径节点*

**private** YKDStack<Tile> steps = **new** YKDStack<>();

*// 存储4个方向*

**private** **int**[][] directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

**public** **Game**(**char**[][] map) {

**this**.map = map;

}

......

}

完善findPath 方法

第一步：将起点添加到栈中

第二步：循环获取栈顶元素，修改方向准备探索

第三步：根据方向添加新探索节电到栈中，或者设置为死路并pop栈顶元素

第四步：回溯整个过程 当while循环执行完成后，只有两种情况，第一种，栈为空，遍历完能走到的所有方格，都没有找到终点；第二种，找到终点，break跳出循环。