栈

数据结构与算法

从操作特性看，栈是一种「操作受限制」的线性表，只允许在一端插入或删除数据。

从功能上看，数组或者链表确实可以替代栈，但特定的数据结构是对特定场景的抽象，而且，数组或链表暴露了太多的操作接口，操作上的确灵活自由，但使用时就比较不可控，自然也就更容易出错。

栈主要包括两个操作，入栈和出栈，也就是在栈顶插入一个数据和从栈顶删除一个数据。

栈既可以用数组实现，也可以用链表实现。用数组实现的叫顺序栈，用链表实现的叫链式栈。

```
// 基于数组实现的顺序栈
public class ArrayStack{
    private String[] items;    //数组
    private int count;         //栈中元素的个数
    private int n;             //栈的大小

// 初始化数组，申请一个大小为 n 的数组空间
public ArrayStack(int n){
    ithis.items = new String[n];
    this.n = n;
    this.count = 0;
    }

//入栈操作
public boolean push(String item){
    //数组空间不够了，直接返回 false，入栈失败。
    if(count == n) return fasle;
    //将 item 放到下标为 count 的位置，并且 count 加 1
    item[count] = item;
    ++count;
    return true;
    }

//出栈操作
public String pop(){
    //栈为空，则直接返回 null
    if(count == 0) return null;
    //返回下标为 count - 1 的数组元素，并且栈中元素个数count 减 1
    String tmp = items[count -1];
    --count;
    return tmp;
    }
}
```

不管是顺序栈还是链式栈，我们存储数据只需要一个大小为 n 的数组就够了，在入栈和出栈过程中，只需要一两个临时变量存储空间，所以空间复杂度为 $O(1)$。这里存储数据需要一个大小为 n 的数组，并不是说空间复杂度就是 $O(n)$，这 n 个空间是必须的，无法省掉。我们说的空间复杂度，是指除了原本的数据存储空间外，算法运行还需要额外的存储空间。

栈在软件工程中的实际应用，经典场景为**函数调用栈**。操系统给每个线程分配了一块独立的内存空间，这块内存被组织成「栈」这种结构，用来存储函数调用时的临时变量。每进入一个函数，就会将临时变量作为一个栈帧入栈，当被调用函数执行完成，返回之后，将这个函数对应的栈帧出栈。

参考自：极客时间《数据结构与算法之美 》专栏