# 栈

后进者先出，先进者后出，这就是典型的“栈”结构。即从栈的操作特性上来看，栈是一种“操作受限”的线性表，只允许在一端插入和删除数据。

## 栈与数组、链表相比的优势

事实上，从功能上来说，数组或链表确实可以替代栈，但你要知道，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。当某个数据集合只涉及在一端插入和删除数据，并且满足后进先出、先进后出的特性，我们就应该首选“栈”这种数据结构。

事实上，栈既可以用数组来实现，也可以用链表来实现。用数组实现的栈，我们叫作顺序栈，用链表实现的栈，我们叫作链式栈。

## 如何实现一个栈

### 基于数组实现

// 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) {

this.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 false;

// 将item放到下标为count的位置，并且count加一

items[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减一

String tmp = items[count-1];

--count;

return tmp;

}

}

### 基于链表实现

/\*\*

\* 基于链表实现的栈。

\*

\* Author: Zheng

\*/

public class StackBasedOnLinkedList {

private Node top = null;

public void push(int value) {

Node newNode = new Node(value, null);

// 判断是否栈空

if (top == null) {

top = newNode;

} else {

newNode.next = top;

top = newNode;

}

}

/\*\*

\* 我用-1表示栈中没有数据。

\*/

public int pop() {

if (top == null) return -1;

int value = top.data;

top = top.next;

return value;

}

public void printAll() {

Node p = top;

while (p != null) {

System.out.print(p.data + " ");

p = p.next;

}

System.out.println();

}

private static class Node {

private int data;

private Node next;

public Node(int data, Node next) {

this.data = data;

this.next = next;

}

public int getData() {

return data;

}

}

}

### 支持动态扩容的顺序栈

实现一个支持动态扩容的栈，我们只需要底层依赖一个支持动态扩容的数组就可以了。当栈满了之后，我们就申请一个更大的数组，将原来的数据搬移到新数组中。

