

# 栈和队列

## 【本节目标】

1. 栈的概念及使用
2. 队列的概念及使用
3. 相关OJ题

## 1. 栈(Stack)

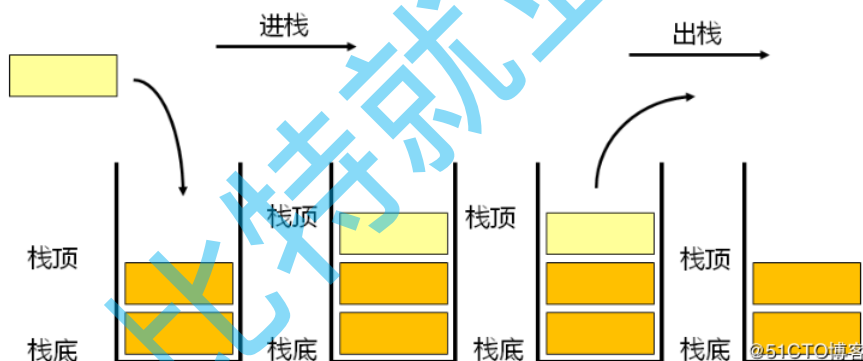
### 1.1 概念

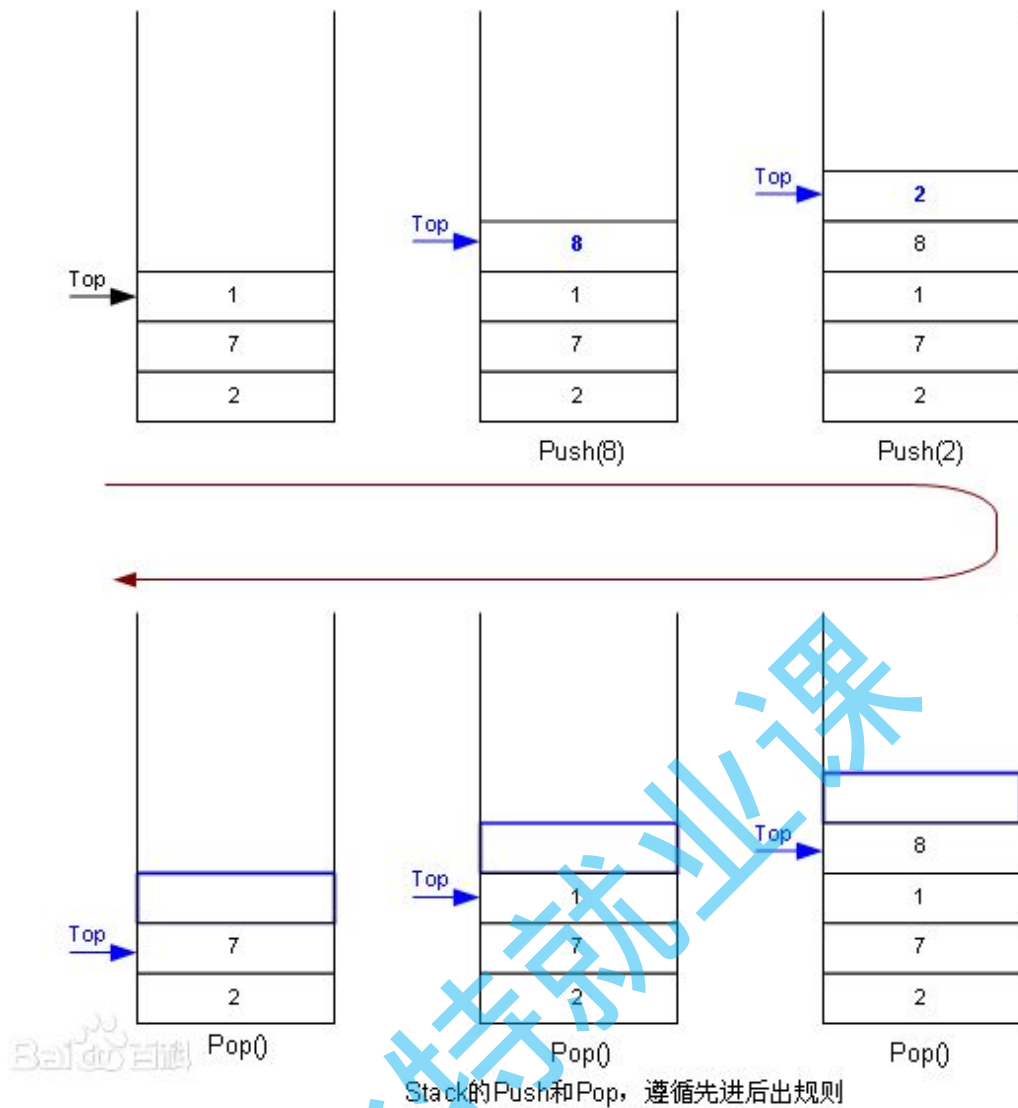
**栈**：一种特殊的线性表，其只允许在固定的一端进行插入和删除元素操作。进行数据插入和删除操作的一端称为栈顶，另一端称为栈底。栈中的数据元素遵守后进先出LIFO（Last In First Out）的原则。

压栈：栈的插入操作叫做进栈/压栈/入栈，入数据在栈顶。

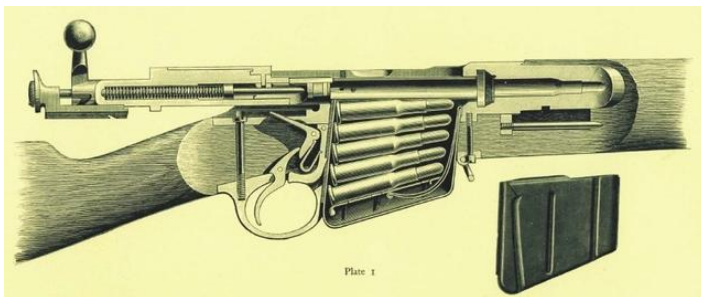
出栈：栈的删除操作叫做出栈。出数据在栈顶。

— 后进先出 ( Last In First Out )





栈在现实生活中的例子：



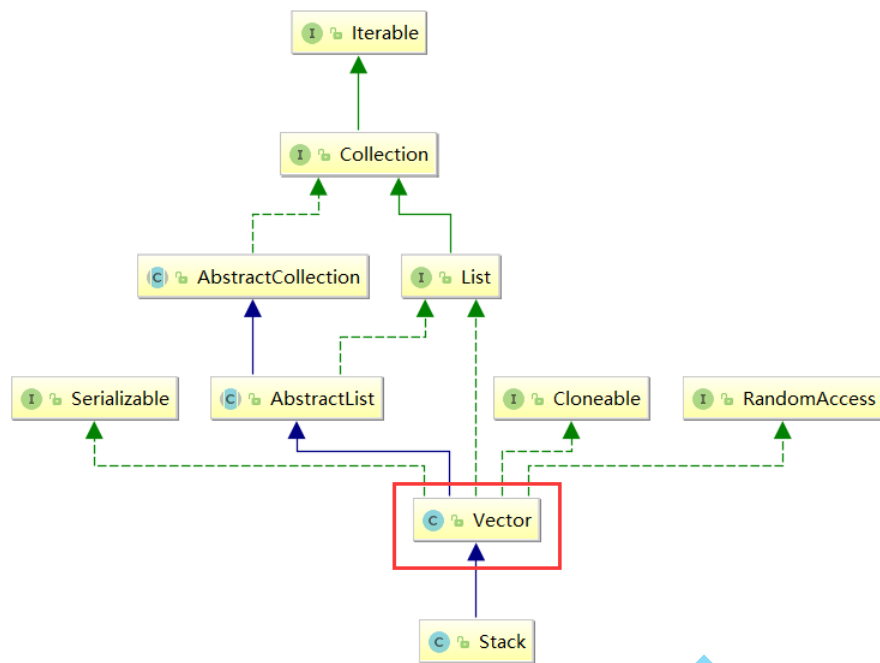
## 1.2 栈的使用

方法	功能
Stack()	构造一个空的栈
E push(E e)	将e入栈，并返回e
E pop()	将栈顶元素出栈并返回
E peek()	获取栈顶元素
int size()	获取栈中有效元素个数
boolean empty()	检测栈是否为空

```
public static void main(String[] args) {
    Stack<Integer> s = new Stack();
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    System.out.println(s.size()); // 获取栈中有效元素个数---> 4
    System.out.println(s.peek()); // 获取栈顶元素---> 4
    s.pop(); // 4出栈，栈中剩余1 2 3，栈顶元素为3
    System.out.println(s.pop()); // 3出栈，栈中剩余1 2，栈顶元素为2
    if(s.empty()){
        System.out.println("栈空");
    }else{
        System.out.println(s.size());
    }
}
```

####

### 1.3 栈的模拟实现



从上图中可以看到，Stack继承了Vector，Vector和ArrayList类似，都是动态的顺序表，不同的是Vector是线程安全的。

```

public class MyStack {
    int[] array;
    int size;

    public MyStack(){
        array = new int[3];
    }

    public int push(int e){
        ensureCapacity();
        array[size++] = e;
        return e;
    }

    public int pop(){
        int e = peek();
        size--;
        return e;
    }

    public int peek(){
        if(empty()){
            throw new RuntimeException("栈为空，无法获取栈顶元素");
        }

        return array[size-1];
    }

    public int size(){

```

```

    return size;
}

public boolean empty(){
    return 0 == size;
}

private void ensureCapacity(){
    if(size == array.length){
        array = Arrays.copyOf(array, size*2);
    }
}
}

```

## 1.4 栈的应用场景

### 1. 改变元素的序列

1. 若进栈序列为 1,2,3,4，进栈过程中可以出栈，则下列不可能的一个出栈序列是 ( )  
A: 1,4,3,2 B: 2,3,4,1 C: 3,1,4,2 D: 3,4,2,1

2. 一个栈的初始状态为空。现将元素1、2、3、4、5、A、B、C、D、E依次入栈，然后再依次出栈，则元素出栈的顺序是 ( )。  
A: 12345ABCDE B: EDCBA54321 C: ABCDE12345 D: 54321EDCBA

### 2. 将递归转化为循环

比如：逆序打印链表

```

// 递归方式
void printList(Node head){
    if(null != head){
        printList(head.next);
        System.out.print(head.val + " ");
    }
}

// 循环方式
void printList(Node head){
    if(null == head){
        return;
    }

    Stack<Node> s = new Stack<>();
    // 将链表中的结点保存在栈中
    Node cur = head;
    while(null != cur){
        s.push(cur);
        cur = cur.next;
    }
}

```

```
// 将栈中的元素出栈
while(!s.empty()){
    System.out.print(s.pop().val + " ");
}
}
```

3. [括号匹配](#)
4. [逆波兰表达式求值](#)
5. [出栈入栈次序匹配](#)

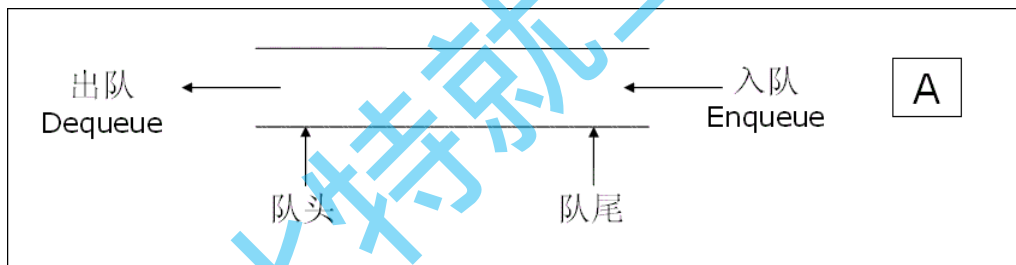
## 1.5 概念区分

栈、虚拟机栈、栈帧有什么区别呢？

## 2. 队列(Queue)

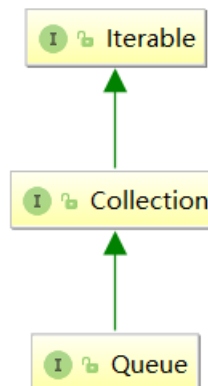
### 2.1 概念

**队列**：只允许在一端进行插入数据操作，在另一端进行删除数据操作的特殊线性表，队列具有先进先出FIFO(First In First Out) 入队列：进行插入操作的一端称为**队尾 (Tail/Rear)** 出队列：进行删除操作的一端称为**队头 (Head/Front)**



### 2.2 队列的使用

在Java中，**Queue**是个接口，底层是通过链表实现的。



方法	功能
boolean offer(E e)	入队列
E poll()	出队列
peek()	获取队头元素
int size()	获取队列中有效元素个数
boolean isEmpty()	检测队列是否为空

注意：Queue是个接口，在实例化时必须实例化LinkedList的对象，因为LinkedList实现了Queue接口。

```

public static void main(String[] args) {
    Queue<Integer> q = new LinkedList<>();
    q.offer(1);
    q.offer(2);
    q.offer(3);
    q.offer(4);
    q.offer(5); // 从队尾入队列
    System.out.println(q.size());
    System.out.println(q.peek()); // 获取队头元素

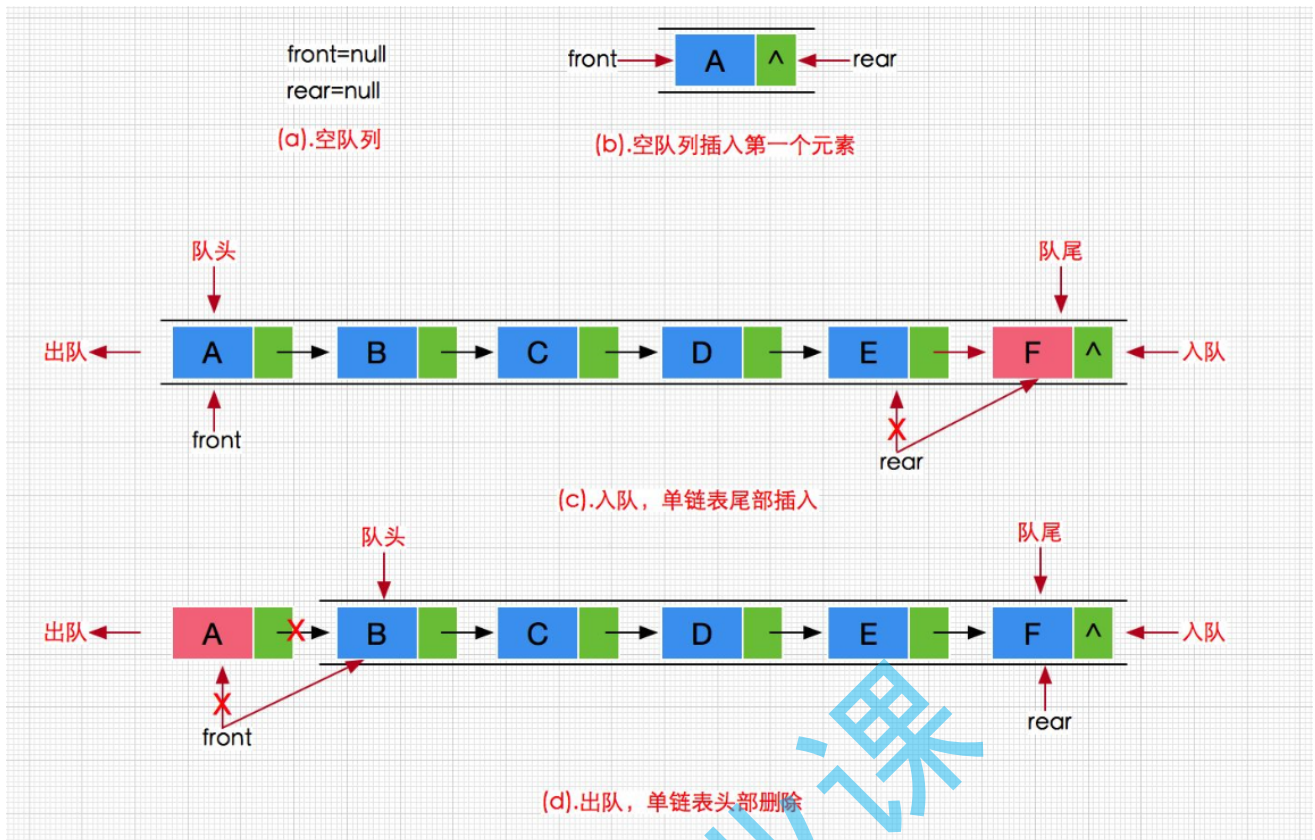
    q.poll();
    System.out.println(q.poll()); // 从队头出队列，并将删除的元素返回

    if(q.isEmpty()){
        System.out.println("队列空");
    }else{
        System.out.println(q.size());
    }
}

```

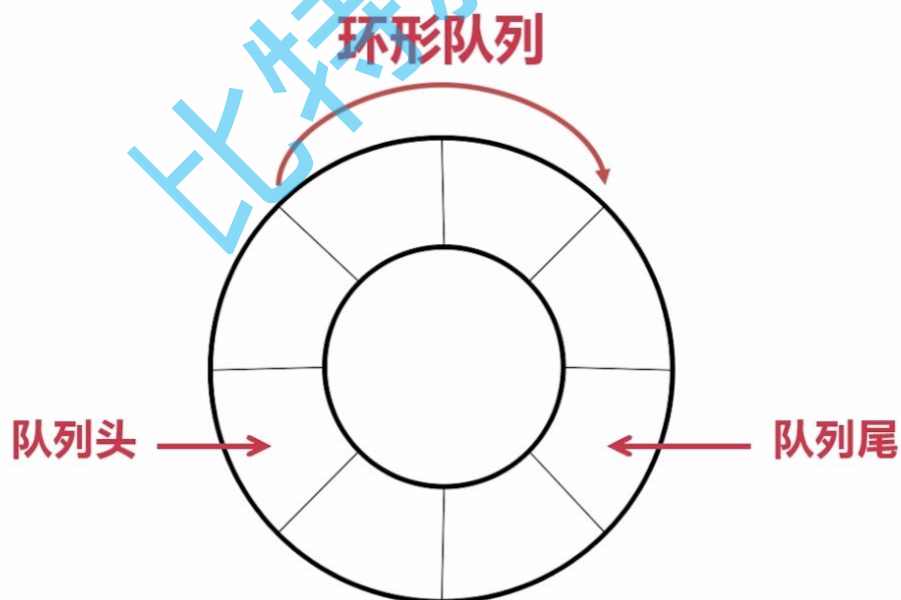
## 2.3 队列模拟实现

队列中既然可以存储元素，那底层肯定要有能够保存元素的空间，通过前面线性表的学习了解到常见的空间类型有两种：**顺序结构**和**链式结构**。同学们思考下：**队列的实现使用顺序结构还是链式结构好？**



## 2.4 循环队列

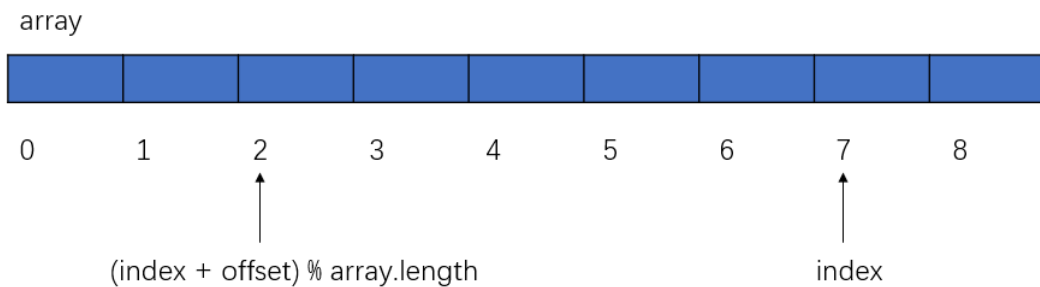
实际中我们有时还会使用一种队列叫循环队列。如操作系统课程讲解生产者消费者模型时就会使用循环队列。环形队列通常使用数组实现。



### 数组下标循环的小技巧

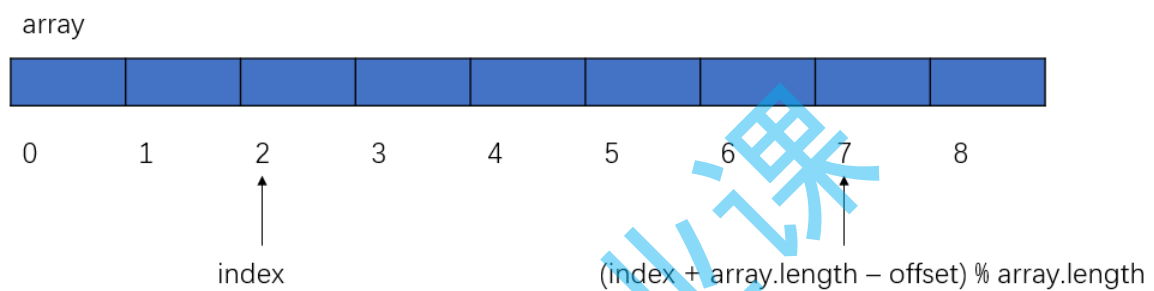
1. 下标最后再往后(offset 小于 `array.length`):  $index = (index + offset) \% array.length$





index = 7  
array.length = 9  
offset = 4  
结果 = 2

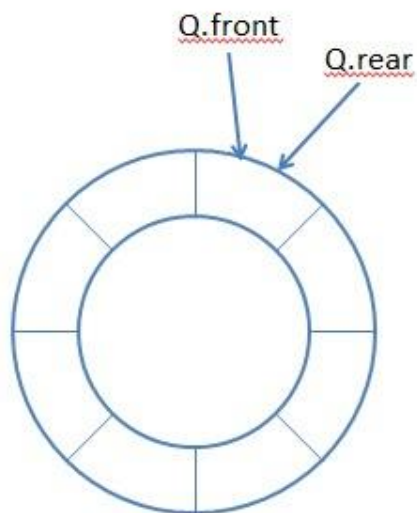
2. 下标最前再往前(offset 小于 array.length):  $\text{index} = (\text{index} + \text{array.length} - \text{offset}) \% \text{array.length}$



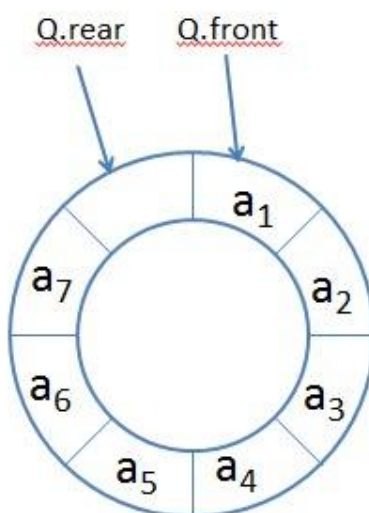
index = 2  
array.length = 9  
offset = 4  
结果 = 7

### 如何区分空与满

1. 通过添加 size 属性记录
2. 保留一个位置
3. 使用标记



(a) 空的循环队列



(b) 满的循环队列

为了使用  $Q.rear = Q.front$  来区别是队空还是队满，我们常常认为出现左图时的情况即为队满的情况，此时： $rear + 1 = front$

[http://blog.csdn.net/zhang\\_xinxu](http://blog.csdn.net/zhang_xinxu)

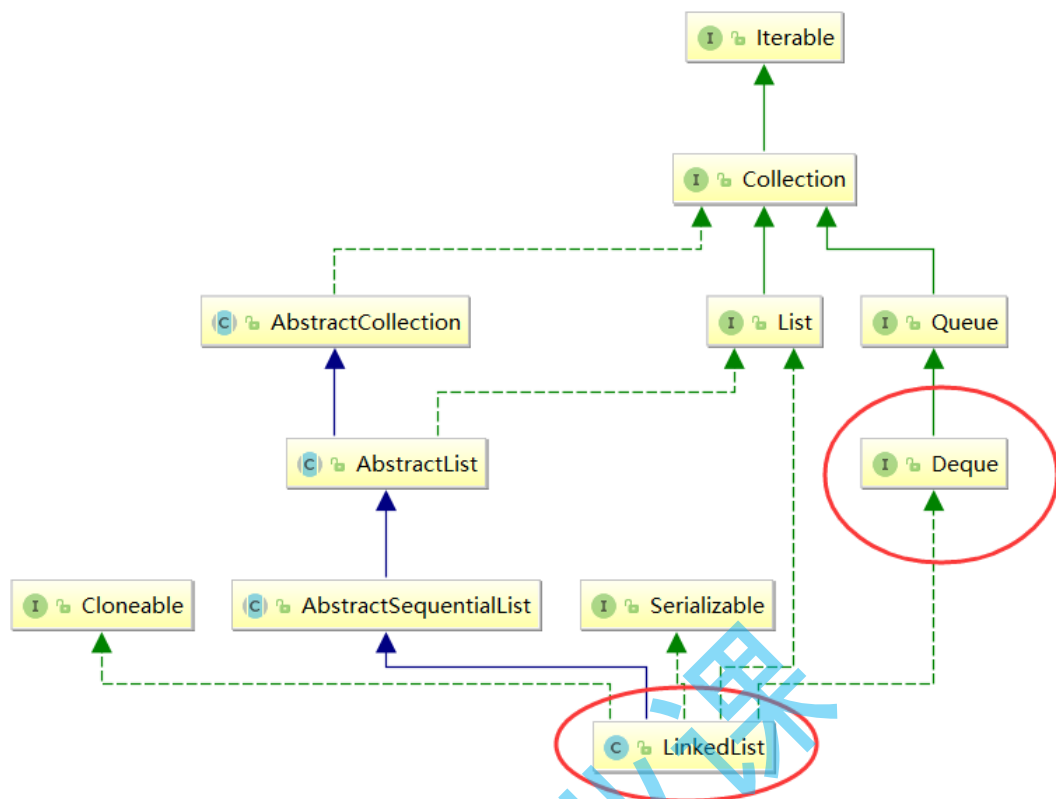
#### 设计循环队列

### 3. 双端队列 (Deque)

双端队列 (deque) 是指允许两端都可以进行入队和出队操作的队列，deque 是 “double ended queue” 的简称。那就说明元素可以从队头出队和入队，也可以从队尾出队和入队。



Deque 是一个接口，使用时必须创建 LinkedList 的对象。



由于在实际开发中Deque使用的并不是非常多，同学们下去自己可以验证下。

#### 4. 面试题

1. 用队列实现栈。 [OJ链接](#)
2. 用栈实现队列。 [OJ链接](#)
3. 实现一个最小栈。 [OJ链接](#)