

队列

@M了个J

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>

码拉松



实力IT教育 www.520it.com

队列 (Queue)

■ 队列是一种特殊的线性表，只能在**头尾两端**进行操作

□ 队尾 (rear)：只能从**队尾添加**元素，一般叫做 **enQueue**，入队

□ 队头 (front)：只能从**队头移除**元素，一般叫做 **deQueue**，出队

□ 先进先出的原则，First In First Out, FIFO

队尾 (rear)



队头 (front)



队列的接口设计

- `int size();` // 元素的数量
- `boolean isEmpty();` // 是否为空
- `void clear();` // 清空
- `void enqueue(E element);` // 入队
- `E dequeue();` // 出队
- `E front();` // 获取队列的头元素

■ 队列的内部实现是否可以直接利用以前学过的数据结构?

□ 动态数组、链表

□ 优先使用双向链表, 因为队列主要是往头尾操作元素

队尾 (rear)



队头 (front)

练习 – 用栈实现队列

■ <https://leetcode-cn.com/problems/implement-queue-using-stacks/>

■ 准备2个栈: inStack、outStack

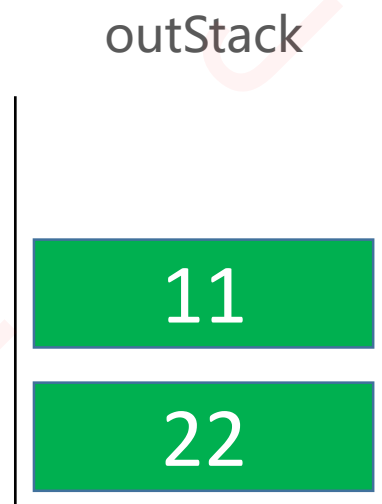
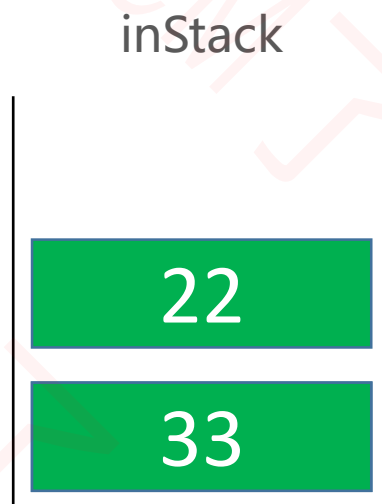
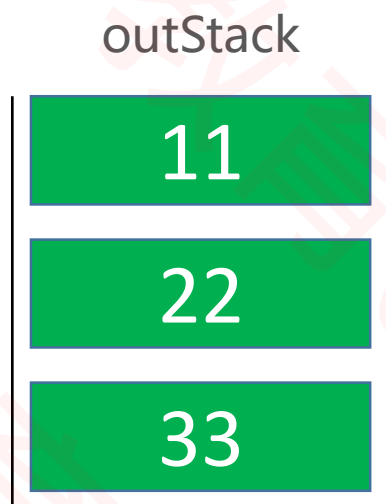
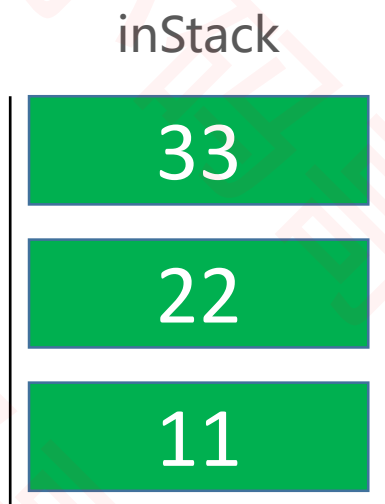
□ 入队时, push 到 inStack 中

□ 出队时

✓ 如果 outStack 为空, 将 inStack 所有元素逐一弹出, push 到 outStack, outStack 弹出栈顶元素

✓ 如果 outStack 不为空, outStack 弹出栈顶元素

■ 假设如下操作: 11入队、22入队、出队、33入队、出队

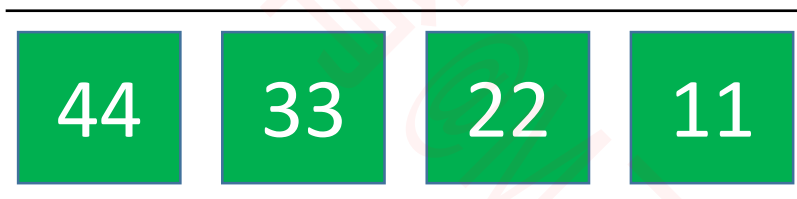


双端队列 (Deque)

■ 双端队列是能在头尾两端添加、删除的队列

□ 英文 deque 是 double ended queue 的简称

队尾 (rear)



队头 (front)

■ `int size();` // 元素的数量

■ `boolean isEmpty();` // 是否为空

■ `void clear();` // 清空

■ `void enqueueRear(E element);` // 从队尾入队

■ `E dequeueFront();` // 从队头出队

■ `void enqueueFront(E element);` // 从队头入队

■ `E dequeueRear();` // 从队尾出队

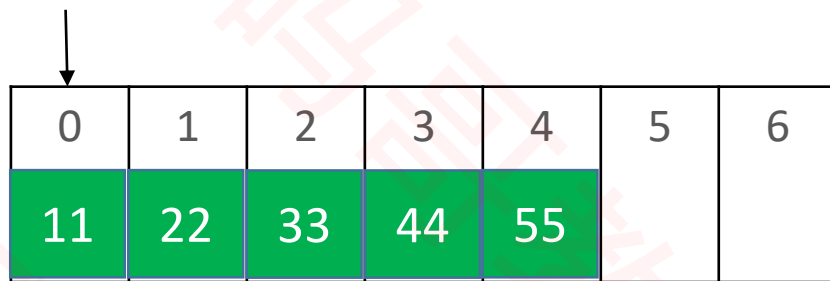
■ `E front();` // 获取队列的头元素

■ `E rear();` // 获取队列的尾元素

循环队列 (Circle Queue)

- 其实队列底层也可以使用动态数组实现，并且各项接口也可以优化到 $O(1)$ 的时间复杂度
- 这个用数组实现并且优化之后的队列也叫做：循环队列

队头 (front)



0	1	2	3	4	5	6
11	22	33	44	55		

- 循环双端队列：可以进行两端添加、删除操作的循环队列

循环队列

```
public E front() {  
    return elements[front];  
}
```

```
public void enqueue(E element) {  
    ensureCapacity(size + 1);  
    elements[index(size)] = element;  
    size++;  
}
```

```
public E dequeue() {  
    E ele = elements[front];  
    elements[front] = null;  
    front = index(1);  
    size--;  
    return ele;  
}
```

```
private int index(int index) {  
    return (front + index) % elements.length;  
}
```

```
private void ensureCapacity(int capacity) {  
    int oldCapacity = elements.length;  
    if (oldCapacity >= capacity) return;  
  
    int newCapacity = oldCapacity + (oldCapacity >> 1);  
    E[] newElementData = (E[]) new Object[newCapacity];  
    for (int i = 0; i < size; i++) {  
        newElementData[i] = elements[index(i)];  
    }  
    elements = newElementData;  
    front = 0;  
}
```

循环双端队列

```
public void enqueueFront(E element) {  
    ensureCapacity(size + 1);
```

```
    front = index(-1);  
    elements[front] = element;  
    size++;
```

```
}
```

```
private int index(int index) {
```

```
    index += front;  
    if (index < 0) {  
        return index + elements.length;  
    }
```

```
    return index % elements.length;
```

```
}
```

```
public E rear() {  
    return elements[rearIndex()];  
}
```

```
private int rearIndex() {  
    return index(size - 1);  
}
```

```
public E dequeueRear() {  
    int rear = rearIndex();  
    E ele = elements[rear];  
    elements[rear] = null;  
    size--;
```

```
    return ele;
```

```
}
```


%运算符优化

- 尽量避免使用乘*、除/、模%、浮点数运算，效率低下

```
private int index(int index) {  
    index += front;  
    return index - (elements.length > index ? 0 : elements.length);  
}
```

```
private int index(int index) {  
    index += front;  
    if (index < 0) {  
        return index + elements.length;  
    }  
    return index - (elements.length > index ? 0 : elements.length);  
}
```

- 已知 $n \geq 0$, $m > 0$

□ $n \% m$ 等价于 $n - (m > n ? 0 : m)$ 的前提条件: $n < 2m$

作业 – 用队列实现栈

- <https://leetcode-cn.com/problems/implement-stack-using-queues/>