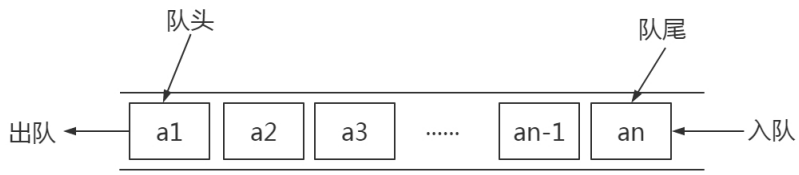
定义



队列（queue）是只允许在一端进行插入操作，而在另一端进行删除操作的线性表

说明

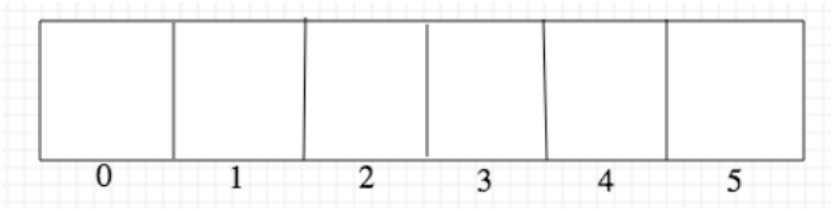
队列同样可以使用数组实现，也可以通过链表来实现

队列抽象数据模型（ADT）

```java
/**
 * 队列 - 抽象数据类型（ADT）
 */
public interface QueueADT<E> {

    void clear();                           // 清空
    boolean isEmpty();                      // 是否为空
    E toll();                               // 出队列
    void add(E e) throws Exception;         // 入队列
     int size();                             // 获取队列的长度
     int length();                           // 获取队列的元素个数
     boolean isFull();                       // 判断是否队列已满
}
```

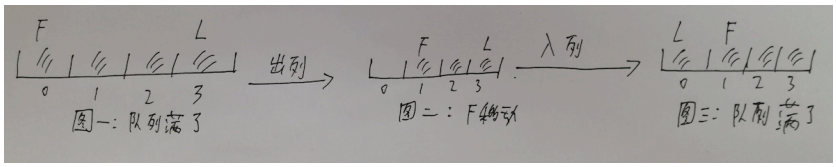
队列实现方式 之 数组



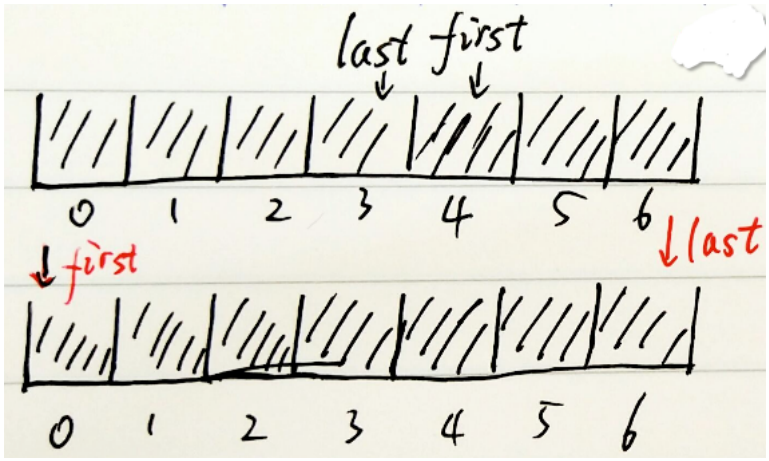
普通队列的局限性

1. 从右边入，左边出。如果当左边出队列，右边的所有元素都需要向左边移动一格，时间复杂度 O(N)
2. 从右边入，左边出。如果当左边出队列，右边的元素不动，那么左边的元素出去的位置就会留空，浪费存储，而且队列容量减小

图解循环队列



如果队列为空，入队列，first 和 last相等
如果first在last左边， last不在最右边（last < size - 1），直接在last后面追加，并且移动last指针
如果first在last左边， last在最右边，如果队列不是满的，那么将元素加入到最左侧，last指针指向最左侧
如果first在last右侧，判断是否满了，如果队列不是满的，那么将元素在last后面追加，并且移动last指针



队列满了的两种情况

代码实现

```java
package day3.队列;


public class QueueByArray<E> implements QueueADT<E> {

    private E[] array;          // 元素存放数组
    private int first;          // 队列第一个元素的下标
    private int last;           // 队列第二个元素的下标
```

```
9     private int size = 5;       // 队列的长度
10     private int length;         // 元素个数
11
12     public QueueByArray(){
13         array = (E[]) new Object[size];
14         first = 0;
15         last = 0;
16         length = 0;
17     }
18
19     public QueueByArray(int size){
20         this();
21         this.size = size;
22     }
23
24     @Override
25     public void clear() {
26         first = 0;
27         last = 0;
28         length = 0;
29         for (int i = 0 ; i < size ; i++){
30             array[i] = null;
31         }
32     }
33
34     @Override
35     public boolean isEmpty() {
36         return length == 0 ? true : false;
37     }
38
39     /**
40      * 出队列 ：  只需要移动头指针
41      * @return
42      */
43     @Override
44     public E toll() {
45         if (isEmpty()){
46             return null;
47         }
48         E e = array[first];
49         array[first] = null;
50         if (first < size - 1){
51             first++;
52         }else {
53             first = 0;
54         }
55         length--;
56         if (isEmpty()){
57             first = 0;
58             last = 0;
59         }
60         return e;
61     }
62
```

```
63
64      /**
65       * 入队列
66       * @param e
67       * @throws Exception
68       */
69      @Override
70      public void add(E e) throws Exception {
71          if (isFull()){
72              throw new Exception("队列已满");
73          }
74          if (isEmpty()){
75              array[0] = e;
76              first = 0;
77              last = 0;
78          }else {
79              if (last < size - 1){
80                  array[++last] = e;
81              }else {
82                  array[0] = e;
83                  last = 0;
84              }
85          }
86          length++;
87      }
88
89      @Override
90      public int size() {
91          return size;
92      }
93
94      @Override
95      public int length() {
96          return length;
97      }
98
99
100      @Override
101      public boolean isFull() {
102          return length == size ? true : false;
103      }
104
105      public void print(){
106          System.out.println(toString());
107      }
108
109      @Override
110      public String toString() {
111          StringBuffer buffer = new StringBuffer();
112          buffer.append(" size : " + size + " , length : " + length );
113          int start = first;
114          int cnt = 0;
115          if (length > 0){
116              buffer.append(" ,遍历元素 : ");
```

```
            for (int i = 0 ; i < length ; i ++){
                E e = array[start];
                buffer.append(e.toString());
                if (cnt < length - 1){
                    buffer.append(",");
                }
                if (start < size - 1){
                    start++;
                }else {
                    start = 0;
                }
                cnt++;
            }
        }
        return buffer.toString();
    }

    public static void main(String[] args) throws Exception {
        QueueByArray<String> queue = new QueueByArray();
        queue.print();
        System.out.println("入栈a");
        queue.add("a");
        queue.print();
        System.out.println("入栈b");
        queue.add("b");
        queue.print();
        System.out.println("入栈c");
        queue.add("c");
        System.out.println("入栈d");
        queue.add("d");
        System.out.println("入栈e");
        queue.add("e");
        queue.print();
        System.out.println("出栈 : " + queue.toll());
        System.out.println("出栈 : " + queue.toll());
        System.out.println("出栈 : " + queue.toll());
        queue.print();
        queue.add("1");
        queue.add("2");
        queue.print();

        System.out.println("出栈 : " + queue.toll());
        System.out.println("出栈 : " + queue.toll());
        System.out.println("出栈 : " + queue.toll());
        System.out.println("出栈 : " + queue.toll());
        queue.print();
    }
}

/* 输出

 size : 5 , length : 0
入栈a
 size : 5 , length : 1 ,遍历元素 : a
```

```
入栈b
 size : 5 , length : 2 ,遍历元素 : a,b
入栈c
入栈d
入栈e
 size : 5 , length : 5 ,遍历元素 : a,b,c,d,e
出栈 : a
出栈 : b
出栈 : c
 size : 5 , length : 2 ,遍历元素 : d,e
 size : 5 , length : 4 ,遍历元素 : d,e,1,2
出栈 : d
出栈 : e
出栈 : 1
出栈 : 2
 size : 5 , length : 0

*/
```

队列实现方式 之 链表

说明

用链表实现栈和队列都十分方便

代码实现

```java
package day3.队列;

import java.util.Iterator;

public class QueueByLink<E> implements Iterable<E>{

    private class Node {
        E data;
        Node next;
    }

    // 指向第一个节点
    private Node first;
    // 指向最后一个节点
    private Node last;
    private int N;

    public int size() {
        return N;
    }

    public boolean isEmpty() {
        return N == 0;
    }


```

```
27
28      // 入列，表尾加入元素
29      public void enqueue(E e) {
30          Node oldlast = last;
31          last = new Node();
32          last.data = e;
33          // last应该指向null，但是新的结点next默认就是null
34          // 如果是第一个元素，则last和first指向同一个，即第一个
35          if (isEmpty()) {
36              first = last;
37          } else {
38              oldlast.next = last;
39          }
40          N++;
41      }
42
43      // 出列，即删除表头元素
44      public E dequeue() {
45          E e = first.data;
46          Node next = first.next;
47          // 这两行有助于垃圾回收
48          first.data = null;
49          first.next = null;
50          first = next;
51          N--;
52          // 最后一个元素被删除，first自然为空了，但是last需要置空。
53          // 注意是先减再判断是否为空
54          if (isEmpty()) {
55              last = null;
56          }
57          return e;
58      }
59      public E getHead() {
60          return first.data;
61      }
62
63      public void clear() {
64          while (first != null) {
65              Node next = first.next;
66              // 下面两行帮助垃圾回收
67              first.next = null;
68              first.data = null;
69              first = next;
70          }
71          // 所有元素都空时，last也没有有所指了。记得last置空
72          last = null;
73          N = 0;
74      }
75
76      @Override
77      public Iterator<E> iterator() {
78          return new Iterator<E>() {
79              private Node current = first;
80
```

```
            @Override
            public boolean hasNext() {
                return current != null;
            }

            @Override
            public E next() {
                E e = current.data;
                current = current.next;
                return e;
            }
        };
    }

    @Override
    public String toString() {
        Iterator<E> it = iterator();

        if (!it.hasNext()) {
             return "[]";
         }

         StringBuilder sb = new StringBuilder();
         sb.append("[");
         while (true) {
             E E = it.next();
             sb.append(E);
             if (!it.hasNext()) {
                 return sb.append("]").toString();
             }

             sb.append(", ");
         }
     }

     public static void main(String[] args) {
         QueueByLink<Integer> a = new QueueByLink<>();
         a.enqueue(1);
         a.enqueue(2);
         a.enqueue(3);
         a.enqueue(4);
         System.out.println(a);
         a.dequeue();
         a.dequeue();
         System.out.println(a); // [3, 4]
         a.clear();
         System.out.println(a.size()); // 0
         a.enqueue(999);
         a.enqueue(777);
         System.out.println(a.getHead()); // 999
     }
}
```