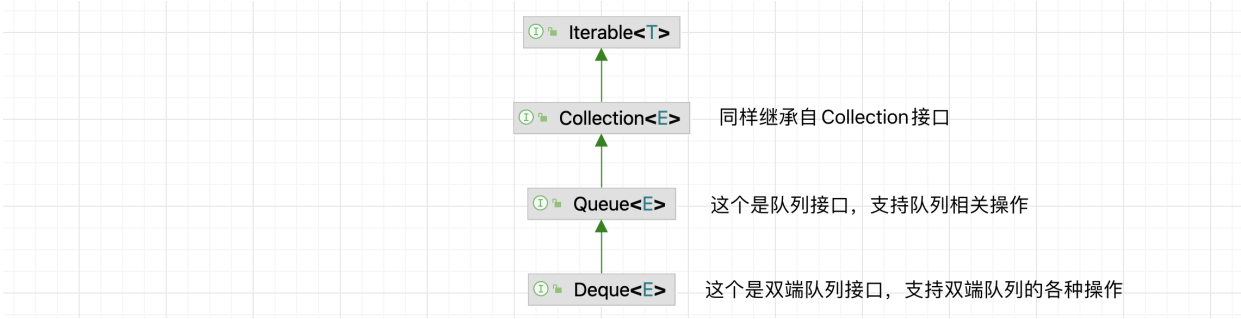
📖 青空的霞光B站教程

📅 2024年4月17日

> ps:要是我早点学这些，我觉得当时我的操作系统的实验，应该就不用那么抓狂

Queue和Deque

⭐ 前面数据结构学习了队列，在前面的`LinkedList`中会发现它也实现了一个`Deque`接口点开`Deque`会发现它继承自`Queue`接口，`Queue`就是队列




```
public class LinkedList<E>
    extends AbstractSequentialList<E>
    implements List<E>, Deque<E>, Cloneable, java.io.Serializable
{
```

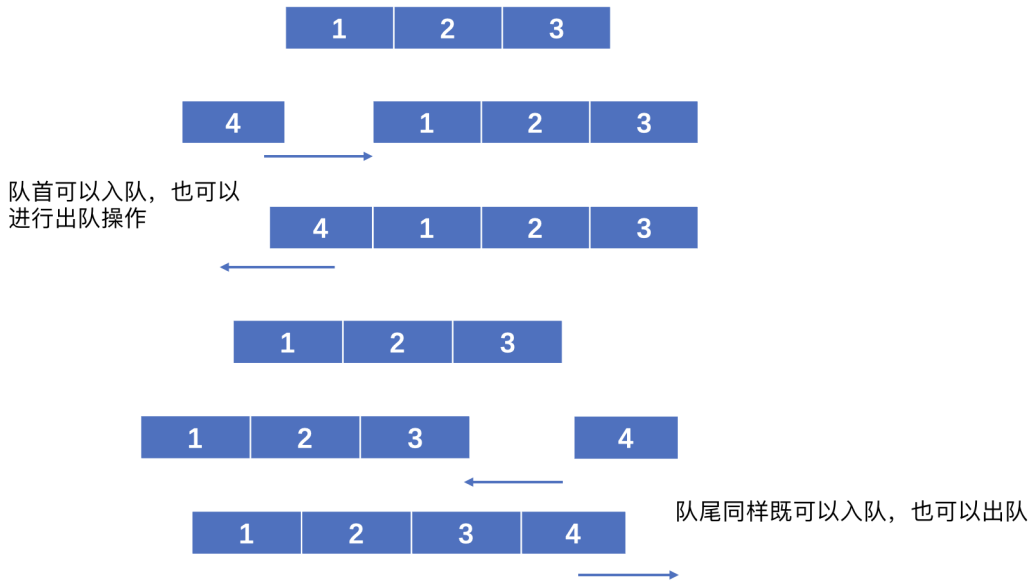
⭐ 查看一下队列的源代码

```
public interface Queue<E> extends Collection<E> {
    //队列的添加操作，是在队尾进行插入（只不过List也是一样的，默认都是尾插）
    //如果插入失败，会直接抛出异常
    boolean add(E e);

    //同样是添加操作，但是插入失败不会抛出异常
    boolean offer(E e);

    //移除队首元素，但是如果队列已经为空，那么会抛出异常
    E remove();

    //同样是移除队首元素，但是如果队列为空，会返回null
    E poll();

    //仅获取队首元素，不进行出队操作，但是如果队列已经为空，那么会抛出异常
    E element();

    //同样是仅获取队首元素，但是如果队列为空，会返回null
    E peek();
}
```

⭐ 前面的数据类型笔记使用队列底层用的就是链表结构，那么现在`LinkedList`是它的实现类，也可以直接作为队列来使用

```java
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.add("AA");
        queue.add("BB");
        System.out.println(queue.poll());
        System.out.println(queue);
    }


//输出
AA
[BB]
```

⭐ `Deque`为双端队列，双端队列既可以当做普通队列使用，也可以当做栈来使用，我们来看看Java中是如何定义的`Deque`双端队列接口的




```java
//在双端队列中，所有的操作都有分别对应队首和队尾的
public interface Deque<E> extends Queue<E> {
    //在队首进行插入操作
    void addFirst(E e);

    //在队尾进行插入操作
    void addLast(E e);

    //不用多说了吧？
    boolean offerFirst(E e);
    boolean offerLast(E e);

    //在队首进行移除操作
    E removeFirst();

    //在队尾进行移除操作
    E removeLast();

```

```
    //不用多说了吧？
    E pollFirst();
    E pollLast();

    //获取队首元素
    E getFirst();

    //获取队尾元素
    E getLast();

        //不用多说了吧？
    E peekFirst();
    E peekLast();

    //从队列中删除第一个出现的指定元素
    boolean removeFirstOccurrence(Object o);

    //从队列中删除最后一个出现的指定元素
    boolean removeLastOccurrence(Object o);

    // *** 队列中继承下来的方法操作是一样的，这里就不列出了 ***

    ...

    // *** 栈相关操作已经帮助我们定义好了 ***

    //将元素推向栈顶
    void push(E e);

    //将元素从栈顶出栈
    E pop();


    // *** 集合类中继承的方法这里也不多种介绍了 ***

    ...

    //生成反向迭代器，这个迭代器也是单向的，但是是next方法是从后往前进行遍历的
    Iterator<E> descendingIterator();

}
```

1 它可以作为栈使用，它在源码内部就定义了栈的功能，同样也可以使用`LinkedList`

```
    public static void main(String[] args) {
        Deque<String> stack = new LinkedList<>();
        stack.push("AAA");
        stack.push("BBB");
        stack.push("CCC");
        System.out.println(stack);
        System.out.println(stack.pop());
```

```
        System.out.println(stack.pop());
        System.out.println(stack.pop());
    }


//输出，先入后出
[CCC, BBB, AAA]
CCC
BBB
AAA
```

2️⃣ 同样它也支持迭代器，不过他这里定义了一个新的迭代器，反向迭代器，在`LinkedList`内又实现方法，也就是从后往前

```
    private class DescendingIterator implements Iterator<E> {
        private final ListItr itr = new ListItr(size());
        public boolean hasNext() {
            return itr.hasPrevious();
        }
        public E next() {
            return itr.previous();
        }
        public void remove() {
            itr.remove();
        }
    }
```

代码测试

```
    public static void main(String[] args) {
        Deque<String> deque = new LinkedList<>();
        deque.add("AAA");
        deque.add("BBB");
        deque.add("CCC");
        Iterator<String> stringIterator = deque.descendingIterator();
        while (stringIterator.hasNext()){
            System.out.println(stringIterator.next());
        }
    }

//输出 从后往前输出
CCC
BBB
AAA
```

⭐ 除了`LinkedList`实现接口外还有两个不是很常用的实现类

```java
public static void main(String[] args) {
    Deque<String> deque = new ArrayDeque<>();   //数组实现的栈和队列
    Queue<String> queue = new PriorityQueue<>();  //优先级队列
}
```

1 这里我对`PriorityQueue`比较感兴趣，让我联想到了操作系统中的进程状态的转换也是需要使用到优先级，它传入的是一个`Comparator`(可以看泛型的函数式接口)比较器

```java
    public PriorityQueue(Comparator<? super E> comparator) {
        this(DEFAULT_INITIAL_CAPACITY, comparator);
    }
```

```java
public static void main(String[] args) {
    Queue<Integer> queue = new PriorityQueue<>();
    queue.offer(10);
    queue.offer(4);
    queue.offer(5);
    System.out.println(queue.poll());
    System.out.println(queue.poll());
    System.out.println(queue.poll());
}
```

2 可以通过传入实现一个新的`Comparator`比较器来重新定义这个优先级，但是他这里只是出队的顺序是**按照优先级的，而保存并不是**

```java
    public static void main(String[] args) {

        //创建优先队列
        PriorityQueue<Integer> queue = new PriorityQueue<>(new
Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o2-o1;
            }
        });

        queue.offer(10);
        queue.offer(4);
        queue.offer(5);
        System.out.println(queue.poll());
        System.out.println(queue.poll());
        System.out.println(queue.poll());
    }

//使用lambda简化
    public static void main(String[] args) {

        //创建优先队列
        PriorityQueue<Integer> queue = new PriorityQueue<>((o1, o2) -> o2-o1);
```

```
24
25          queue.offer(10);
26          queue.offer(4);
27          queue.offer(5);
28          System.out.println(queue.poll());
29          System.out.println(queue.poll());
30          System.out.println(queue.poll());
31      }
```