问题

（1）什么是双端队列?

（2）ArrayDeque是怎么实现双端队列的?

（3）ArrayDeque是线程安全的吗?

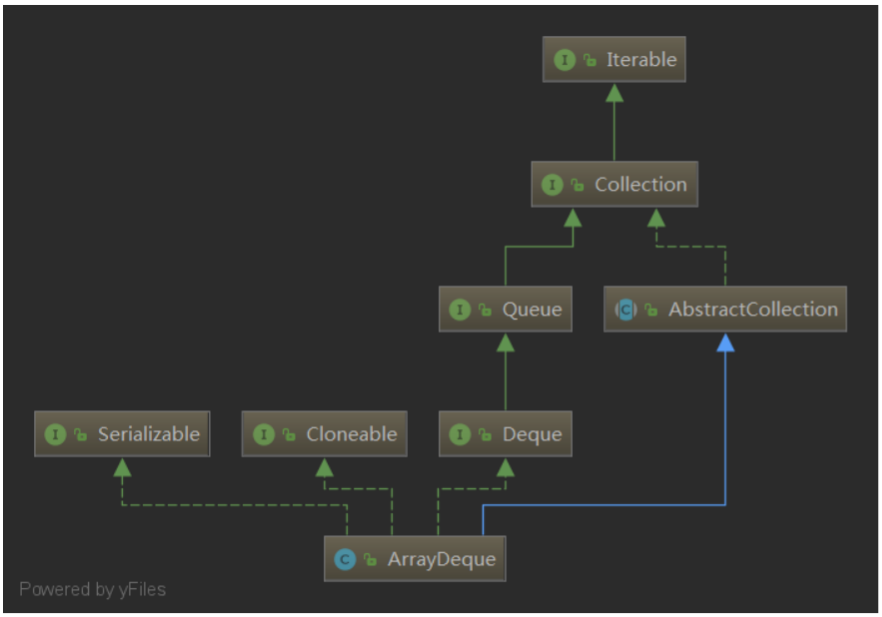
（4）ArrayDeque是有界的吗?

简介

双端队列是一种特殊的队列，它的两端都可以进出元素，故而得名双端队列。

ArrayDeque是一种以数组方式实现的双端队列，它是非线程安全的。

继承体系



通过继承体系可以看，ArrayDeque实现了Deque接口，Deque接口继承自Queue接口，它是对Queue的一种增强。

```java

    public interface Deque<E> extends Queue<E> {
        // 添加元素到队列头
        void addFirst(E e);
        // 添加元素到队列尾
        void addLast(E e);
        // 添加元素到队列头
        boolean offerFirst(E e);
        // 添加元素到队列尾
        boolean offerLast(E e);
        // 从队列头移除元素
        E removeFirst();
        // 从队列尾移除元素
        E removeLast();
        // 从队列头移除元素
        E pollFirst();
        // 从队列尾移除元素
        E pollLast();
        // 查看队列头元素
        E getFirst();
        // 查看队列尾元素
        E getLast();
        // 查看队列头元素
        E peekFirst();
        // 查看队列尾元素
        E peekLast();
        // 从队列头向后遍历移除指定元素
        boolean removeFirstOccurrence(Object o);
        // 从队列尾向前遍历移除指定元素
        boolean removeLastOccurrence(Object o);

        // *** 队列中的方法 ***

        // 添加元素，等于addLast(e)
        boolean add(E e);
         // 添加元素，等于offerLast(e)
        boolean offer(E e);
        // 移除元素，等于removeFirst()
        E remove();
        // 移除元素，等于pollFirst()
        E poll();
        // 查看元素，等于getFirst()
        E element();
        // 查看元素，等于peekFirst()
        E peek();

        // *** 栈方法 ***

        // 入栈，等于addFirst(e)
        void push(E e);
        // 出栈，等于removeFirst()
        E pop();

        // *** Collection中的方法 ***

        // 删除指定元素，等于removeFirstOccurrence(o)
        boolean remove(Object o);
        // 检查是否包含某个元素
        boolean contains(Object o);
        // 元素个数
        public int size();
        // 迭代器
        Iterator<E> iterator();
        // 反向迭代器
```

```
65.         Iterator<E> descendingIterator();
66.     }
67.
```

Deque中新增了以下几类方法：

（1）First，表示从队列头操作元素；

（2）Last，表示从队列尾操作元素；

（3）push(e)，pop()，以栈的方式操作元素的方法；

源码分析

主要属性

```
1.     // 存储元素的数组
2.     transient Object[] elements; // non-private to simplify nested class access
3.     // 队列头位置
4.     transient int head;
5.     // 队列尾位置
6.     transient int tail;
7.     // 最小初始容量
8.     private static final int MIN_INITIAL_CAPACITY = 8;
9.
```

从属性我们可以看到，ArrayDeque使用数组存储元素，并使用头尾指针标识队列的头和尾，其最小容量是8。

主要构造方法

```
1.     // 默认构造方法，初始容量为16
2.     public ArrayDeque() {
3.         elements = new Object[16];
4.     }
5.     // 指定元素个数初始化
6.     public ArrayDeque(int numElements) {
7.         allocateElements(numElements);
8.     }
9.     // 将集合c中的元素初始化到数组中
10.    public ArrayDeque(Collection<? extends E> c) {
11.        allocateElements(c.size());
12.        addAll(c);
13.    }
14.    // 初始化数组
15.    private void allocateElements(int numElements) {
16.        elements = new Object[calculateSize(numElements)];
17.    }
18.    // 计算容量，这段代码的逻辑是算出大于numElements的最接近的2的n次方且不小于8
19.    // 比如，3算出来是8，9算出来是16，33算出来是64
20.    private static int calculateSize(int numElements) {
21.        int initialCapacity = MIN_INITIAL_CAPACITY;
22.        // Find the best power of two to hold elements.
23.        // Tests "<=" because arrays aren't kept full.
24.        if (numElements >= initialCapacity) {
25.            initialCapacity = numElements;
26.            initialCapacity |= (initialCapacity >>>  1);
27.            initialCapacity |= (initialCapacity >>>  2);
28.            initialCapacity |= (initialCapacity >>>  4);
29.            initialCapacity |= (initialCapacity >>>  8);
30.            initialCapacity |= (initialCapacity >>> 16);
31.            initialCapacity++;
```

```
32.
33.             if (initialCapacity < 0)   // Too many elements, must back off
34.                 initialCapacity >>>= 1;// Good luck allocating 2 ^ 30 elements
35.         }
36.         return initialCapacity;
37.     }
38.
```

通过构造方法，我们知道默认初始容量是16，最小容量是8。

入队

入队有很多方法，我们这里主要分析两个，addFirst(e)和addLast(e)。

```
1.     // 从队列头入队
2.     public void addFirst(E e) {
3.         // 不允许null元素
4.         if (e == null)
5.             throw new NullPointerException();
6.         // 将head指针减1并与数组长度减1取模
7.         // 这是为了防止数组到头了边界溢出
8.         // 如果到头了就从尾再向前
9.         // 相当于循环利用数组
10.        elements[head = (head - 1) & (elements.length - 1)] = e;
11.        // 如果头尾挨在一起了，就扩容
12.        // 扩容规则也很简单，直接两倍
13.        if (head == tail)
14.            doubleCapacity();
15.    }
16.    // 从队列尾入队
17.    public void addLast(E e) {
```

```
18.        // 不允许null元素
19.        if (e == null)
20.            throw new NullPointerException();
21.        // 在尾指针的位置放入元素
22.        // 可以看到tail指针指向的是队列最后一个元素的下一个位置
23.        elements[tail] = e;
24.        // tail指针加1，如果到数组尾了就从头开始
25.        if ( (tail = (tail + 1) & (elements.length - 1)) == head)
26.            doubleCapacity();
27.    }
28.
```

（1）入队有两种方式，从队列头或者从队列尾；

（2）如果容量不够了，直接扩大为两倍；

（3）通过取模的方式让头尾指针在数组范围内循环；

（4）x & (len - 1) = x % len，使用&的方式更快；

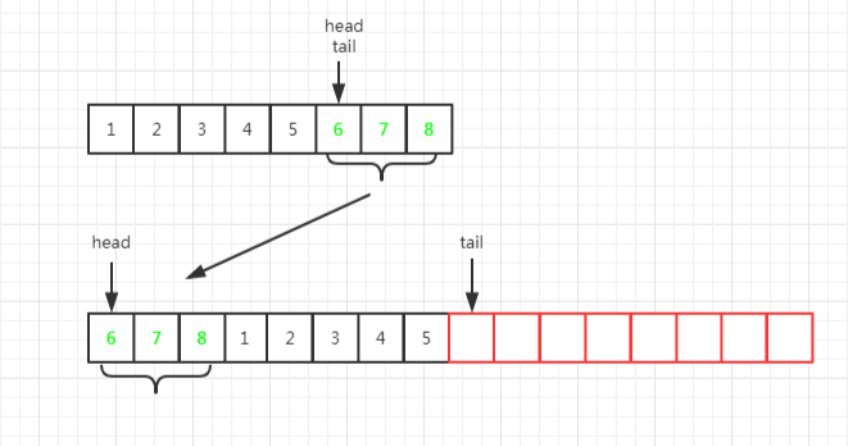
扩容

```
1.     private void doubleCapacity() {
2.         assert head == tail;
3.         // 头指针的位置
4.         int p = head;
5.         // 旧数组长度
6.         int n = elements.length;
7.         // 头指针离数组尾的距离
8.         int r = n - p; // number of elements to the right of p
9.         // 新长度为旧长度的两倍
```

```
        int newCapacity = n << 1;
        // 判断是否溢出
        if (newCapacity < 0)
            throw new IllegalStateException("Sorry, deque too big");
        // 新建新数组
        Object[] a = new Object[newCapacity];
        // 将旧数组head之后的元素拷贝到新数组中
        System.arraycopy(elements, p, a, 0, r);
        // 将旧数组下标0到head之间的元素拷贝到新数组中
        System.arraycopy(elements, 0, a, r, p);
        // 赋值为新数组
        elements = a;
        // head指向0，tail指向旧数组长度表示的位置
        head = 0;
        tail = n;
    }

```

扩容这里迁移元素可能有点绕，请看下面这张图来理解。



出队

出队同样有很多方法，我们主要看两个，pollFirst()和pollLast()。

```
    // 从队列头出队
    public E pollFirst() {
        int h = head;
        @SuppressWarnings("unchecked")
        // 取队列头元素
        E result = (E) elements[h];
        // 如果队列为空，就返回null
        if (result == null)
            return null;
        // 将队列头置为空
        elements[h] = null;     // Must null out slot
        // 队列头指针右移一位
        head = (h + 1) & (elements.length - 1);
        // 返回取得的元素
        return result;
    }
    // 从队列尾出队
    public E pollLast() {
        // 尾指针左移一位
        int t = (tail - 1) & (elements.length - 1);
        @SuppressWarnings("unchecked")
        // 取当前尾指针处元素
        E result = (E) elements[t];
        // 如果队列为空返回null
        if (result == null)
```

```
            return null;
        // 将当前尾指针处置为空
        elements[t] = null;
        // tail指向新的尾指针处
        tail = t;
        // 返回取得的元素
        return result;
    }

```

（1）出队有两种方式，从队列头或者从队列尾；

（2）通过取模的方式让头尾指针在数组范围内循环；

（3）出队之后没有缩容哈哈^^

栈

前面我们介绍Deque的时候说过，Deque可以直接作为栈来使用，那么ArrayDeque是怎么实现的呢?

```
    public void push(E e) {
        addFirst(e);
    }

    public E pop() {
        return removeFirst();
    }

```

是不是很简单，入栈出栈只要都操作队列头就可以了。

总结

（1）ArrayDeque是采用数组方式实现的双端队列；

（2）ArrayDeque的出队入队是通过头尾指针循环利用数组实现的；

（3）ArrayDeque容量不足时是会扩容的，每次扩容容量增加一倍；

（4）ArrayDeque可以直接作为栈使用；