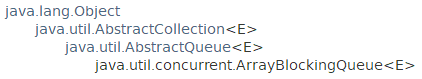
ArrayBlockingQueue和LinkedBlockingQueue

本文章记录：**数组阻塞队列**和**链表阻塞队列**。

# ArrayBlockingQueue数组阻塞队列

## 继承关系介绍



public class **ArrayBlockingQueue**<E> extends **AbstractQueue**<E>

implements **BlockingQueue**<E>, Serializable



存在于**java.util.concurrent**包中。

所有实现的接口：All Implemented Interfaces: Serializable, Iterable<E>, Collection<E>, **BlockingQueue**<E>, **Queue**<E>

since 1.5

## 特征介绍

### 基于**数组**数据结构，有界，FIFO

**A bounded blocking queue backed by an array.** This queue orders elements **FIFO** (first-in-first-out). The head of the queue is that element that has been on the queue **the longest time**. The tail of the queue is that element that has been on the queue **the shortest time**. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.(从尾部插入，从头部删除，先入先出)

数组阻塞队列是基于数组数据结构的，且队列一旦创建，数组的长度也就需要确定，那么在创建数组阻塞队列的时候必须设置队列的容量，见**构造方法**。

### 容量大小固定

**This is a classic "bounded buffer", in which a fixed-sized array holds elements inserted by producers and extracted by consumers**. **Once created, the capacity cannot be changed**. Attempts to put an element into a full queue will result in the operation blocking; attempts to take an element from an empty queue will similarly block.

数组阻塞队列一旦创建，容量大小不可改变。向一个满的队列添加元素会阻塞等待，从一个空队列中获取元素也会阻塞等待。

### an optional fairness policy

This class supports **an optional fairness policy** for ordering waiting producer and consumer threads. **By default, this ordering is not guaranteed**. However, a queue constructed with fairness set to true grants threads access **in FIFO order**. Fairness generally decreases throughput but reduces variability and avoids starvation.

This class and its iterator implement all of the optional methods of the Collection and Iterator interfaces.

## 构造方法

一共可以设置3个参数，必须有int capacity，其次boolean fair公平策略；最后是初始化元素的集合Collection。

### **ArrayBlockingQueue**(int capacity)

Creates an ArrayBlockingQueue with the given (fixed) capacity and default access policy.

### **ArrayBlockingQueue**(int capacity, boolean fair)

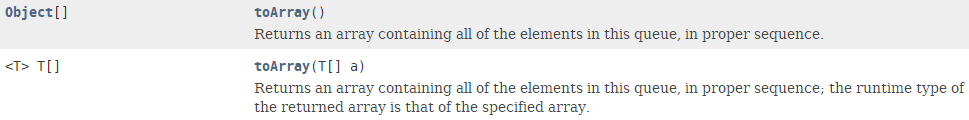
Creates an ArrayBlockingQueue with the given (fixed) capacity and the specified access policy.

### **ArrayBlockingQueue**(int capacity, boolean fair, **Collection**<? extends E> c)

Creates an ArrayBlockingQueue with the given (fixed) capacity, the specified access policy and **initially** **containing the elements of the given collection**, added in traversal order of the collection's iterator.

## 特有方法

### toArray



### spliterator



C．size



D． iterator



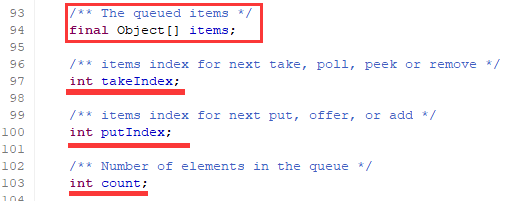
## 源码解读

### 继承与实现



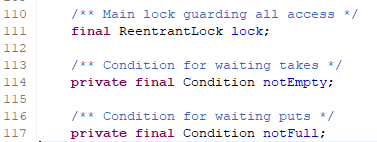
### **数据结构**：数组

队列中的元素都是存放在一个数组items中，通过两个索引takeIndex和putIndex标定位置，利用count记录**队列(数组)**中元素个数。



### 锁机制

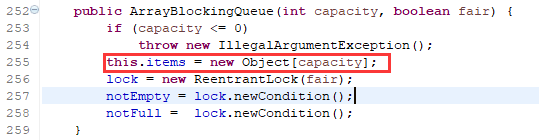
可重入锁ReentrantLock添加两个Condition：notEmpty和notFull。

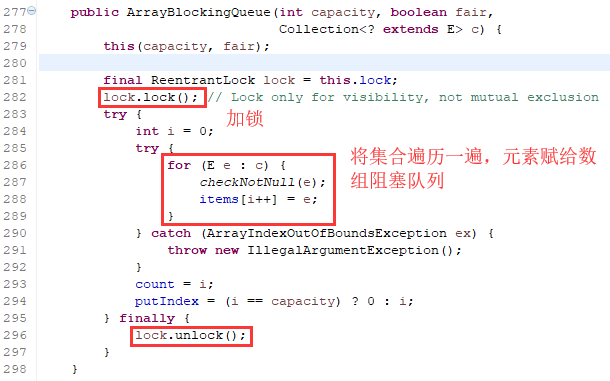


### 构造方法初始化

数组阻塞队列**ArrayBlockingQueue**的公平策略对应的就是可重入锁ReentrantLock的公平策略，缺省为非公平策略。

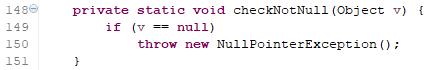






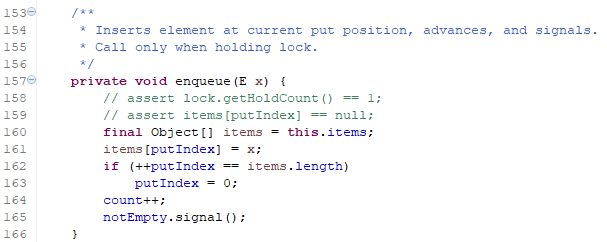
### 数组阻塞队列不支持null元素，否则抛出NullPointerException

通过下面方法实现

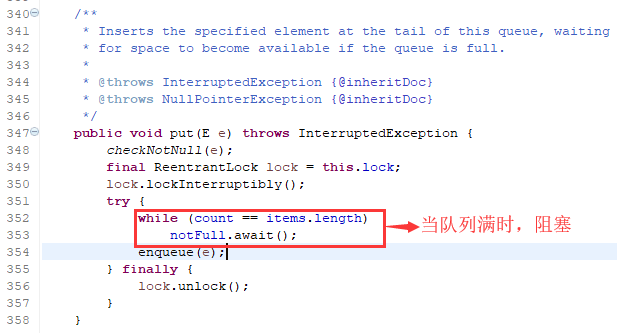


### 向队列中添加元素

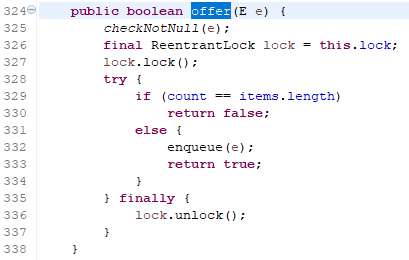
最基本的私有方法：enqueue

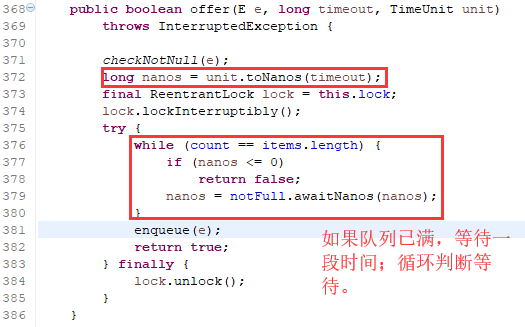


阻塞方法：put



返回特殊值Special Value的offer方法



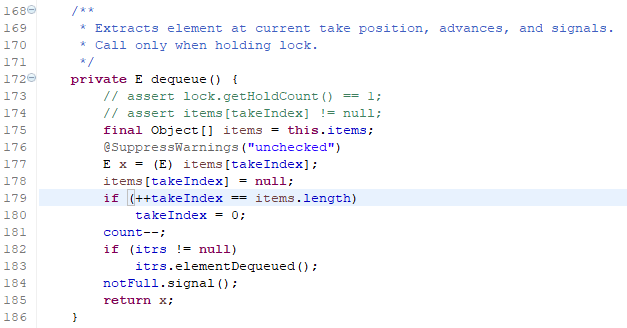


add方法

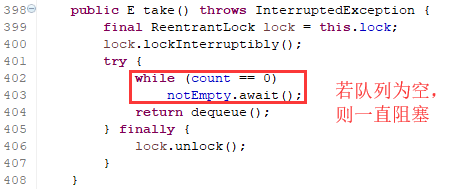


### 获取元素

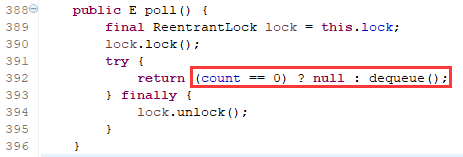
最基本的私有方法：dequeue

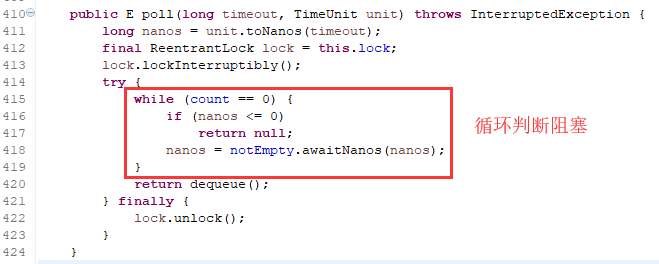


阻塞方法：take()

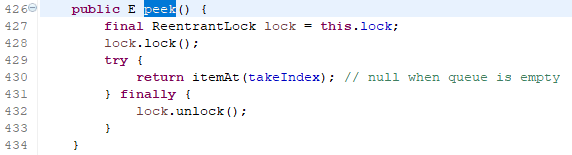


poll方法：返回null表示队列为空，因此不允许null元素。





peek方法



### toArray方法



System.arraycopy方法为native方法。



### 其他方法源代码自己查看吧。

## 数组阻塞队列的核心方法

添加元素与获取元素是最重要的元素，分为4类，一一对应。

其中**put和take**都是阻塞方法；

offer和poll为返回特殊值或等待时间的返回特殊值方法；

add和peek方法

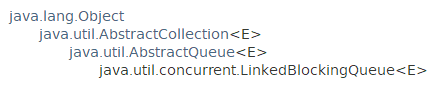
# LinkedBlockingQueue链表阻塞队列

## 简单介绍

public class **LinkedBlockingQueue**<E> extends AbstractQueue<E>

implements **BlockingQueue**<E>, Serializable

All Implemented Interfaces: Serializable, Iterable<E>, Collection<E>, **BlockingQueue**<E>, Queue<E>



This class is a member of *the Java Collections Framework*. *Since:1.5*

## 功能实现

### 基于链表的、容量可选的阻塞队列、FIFO

容量可选：当指定大小时，就是有界的；不指定容量时，默认大小为Integer.MAX\_VALUE，认为是无界的。

**An optionally-bounded blocking queue based on linked nodes**. This queue orders elements **FIFO** (first-in-first-out). The **head** of the queue is that element that has been on the queue the longest time. The **tail** of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. **Linked queues** typically have higher throughput than array-based queues but less predictable performance in most concurrent applications.

The optional capacity bound constructor argument serves as a way to prevent excessive queue expansion. The capacity, if unspecified, is equal to **Integer.MAX\_VALUE**. Linked nodes are dynamically created upon each insertion unless this would bring the queue above capacity.

This class and its iterator implement all of the optional methods of the Collection and Iterator interfaces.

## 构造方法

### LinkedBlockingQueue()

Creates a LinkedBlockingQueue with a capacity **of Integer.MAX\_VALUE**.

### LinkedBlockingQueue(Collection<? extends E> c)

Creates a LinkedBlockingQueue with a capacity of Integer.MAX\_VALUE, initially containing the elements of the given collection, added in traversal order of the collection's iterator.

### LinkedBlockingQueue(int *capacity*)

Creates a LinkedBlockingQueue with the given (fixed) capacity.

## 一般方法

### 添加元素

add、put、offer

### 获取元素

take、poll、peek

### remove

### toArray

Object[] toArray()

Returns an array containing all of the elements in this queue, in proper sequence.

<T> T[] toArray(T[] a)

Returns an array containing all of the elements in this queue, in proper sequence; the runtime type of the returned array is that of the specified array.

### size、remainingCapacity

int **size**()

Returns the number of elements in this queue.

int **remainingCapacity**()

Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.

## 线程池使用LinkedBlockingQueue

线程池建议使用LinkedBlockingQueue，而不使用**LinkedList**的原因？



## 源码分析

LinkedBlockingQueue的容量可以指定具体大小也可以不指定，默认为Integer.MAX\_VALUE。

Integer的最大值很大，可以认为是无界的。因此，如果在生产者速度大于消费者速度的时候，很可能造成堆溢出。

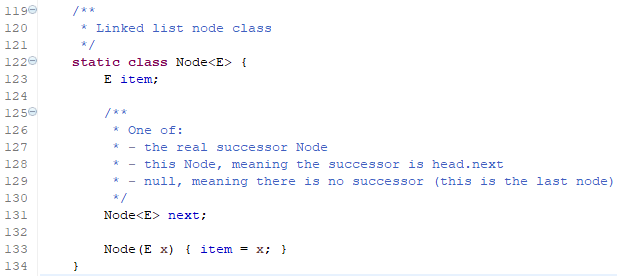
### LinkedBlockingQueue<E>



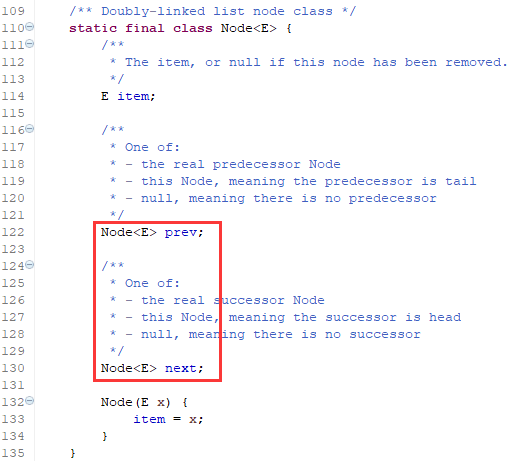
### 数据结构：链表

链表通过定义一个Node<E>实现**链表**结构。

由于Queue只是单向的，因此只有next。对于双向队列，其有next和pre。



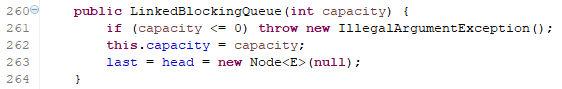
LinkedBlockingDeque的数据结构为：**prev和next**。

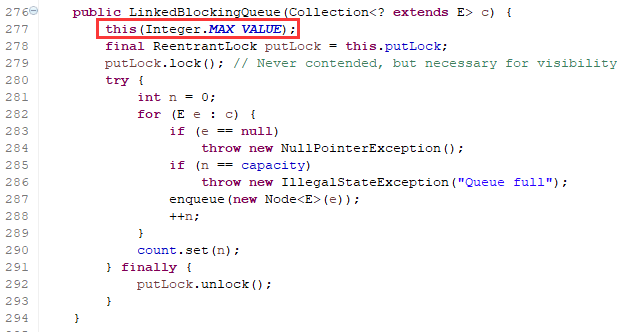


### 构造方法

缺省容量为Integer.MAX\_VALUE







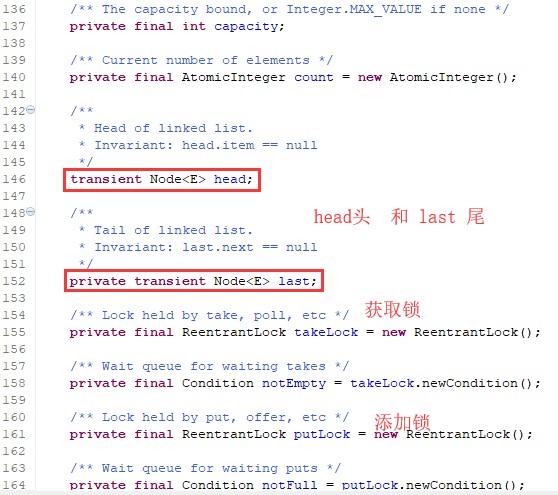
### size和remainingCapacity





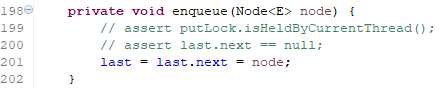
### 基本属性

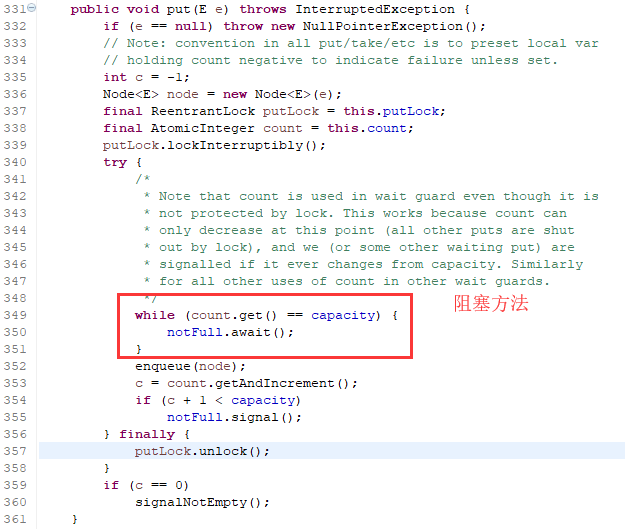
定义了两个可重入锁takeLock和putLock，每个锁有个Condition为**notEmpty**和**notFull**。



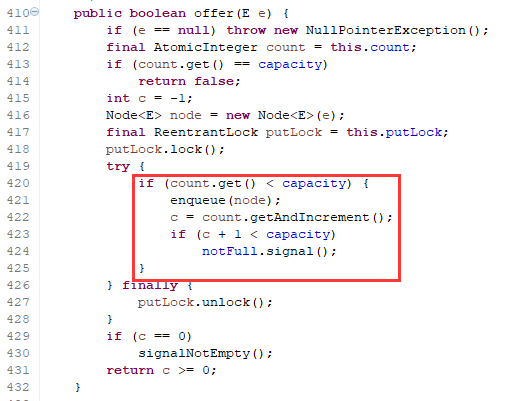
### 添加元素

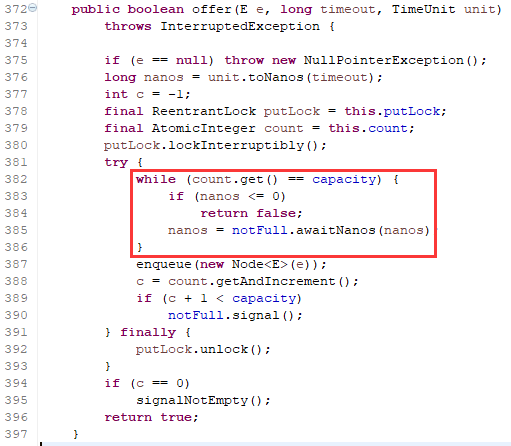
最基本的enqueue方法：





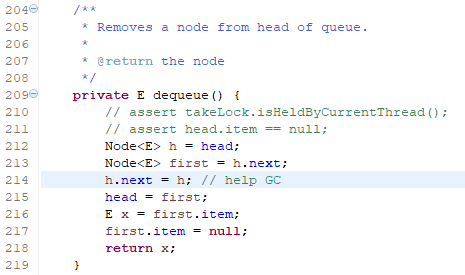
offer



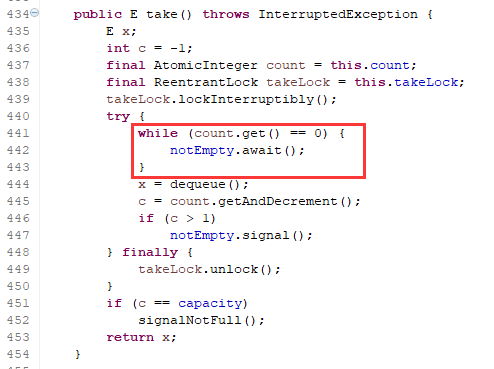


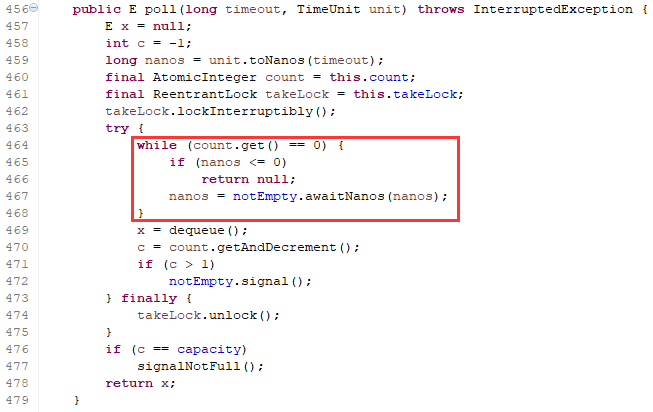
### 获取元素

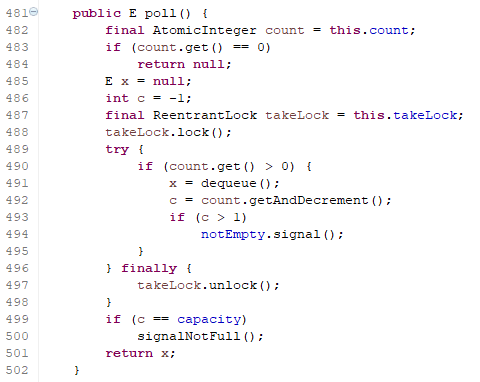
最基本的方法dequeue：

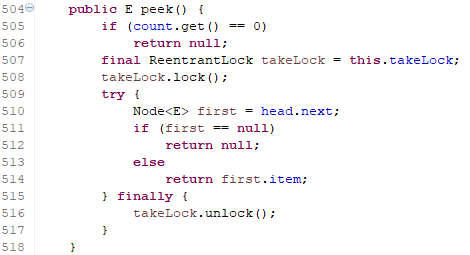


take方法

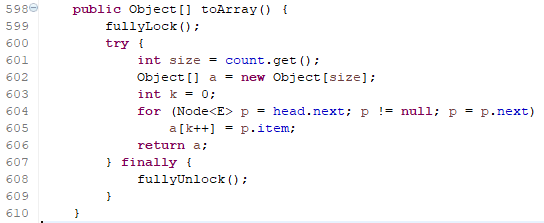








### toArray



内部有个类型强转：

