BlokingQueue阻塞队列

# BlockingQueue阻塞队列介绍

阻塞队列BlockingQueue接口及其实现类都存在**java.util.concurrent包**中。BlockingQueue继承于**java.util.Queue接口**。

接口：java.util.concurrent.BlockingQueue

实现类： **ArrayBlockingQueue(数组阻塞队列)**, **LinkedBlockingQueue** (链表阻塞队列，实现的是**BlockingDeque**接口),

**PriorityBlockingQueue**(优先级阻塞队列)、**DelayQueue**(延时队列)、

**SynchronousQueue**(同步队列)

# java.util.concurrent.BlockingQueue接口

public interface **BlockingQueue**<E> extends **Queue**<E>

**BlockingQueue、BlockingDeque接口存在于java.util.concurrent包中**。

实现的所有父接口：All Superinterfaces: Collection<E>, Iterable<E>, **Queue**<E>

**两个子接口**：All Known Subinterfaces: **BlockingDeque**<E>, **TransferQueue**<E>

分别有文档介绍BlockingDeque和TransferQueue。

所有实现类：All Known Implementing Classes: **ArrayBlockingQueue**, DelayQueue, LinkedBlockingDeque, **LinkedBlockingQueue**, LinkedTransferQueue, **PriorityBlockingQueue**, **SynchronousQueue**

A **Queue** that additionally supports operations **that wait for the queue to become non-empty when retrieving an element**, and **wait for space to become available in the queue when storing an element**.

在Queue的基础上外加两个功能：

### 获取元素的时候，队列若为空则一直阻塞等待；

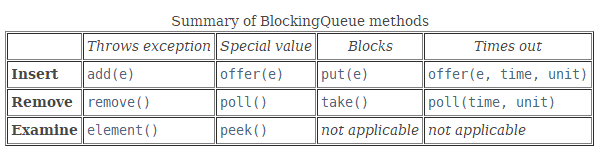
**that wait for the queue to become non-empty when retrieving an element**

### 存储元素的时候，队列若已满则一直阻塞等待。

**wait for space to become available in the queue when storing an element**

# BlockingQueue的方法：四种形式

**BlockingQueue** methods come **in four forms**, with different ways of handling operations that cannot be satisfied immediately, but may be satisfied at some point in the future: **one** throws an exception, **the second** returns a special value (either **null** or **false**, depending on the operation**), the third** blocks the current thread indefinitely until the operation can succeed, and **the fourth** blocks for only a given maximum time limit before giving up. These methods are summarized in the following table:



## BlockingQueue不支持null元素

A **BlockingQueue** does not accept null elements. Implementations throw **NullPointerException** on attempts to add, put or offer a null. A null is used as a sentinel value to indicate failure of poll operations.

## capacity bounded：容量限定

A BlockingQueue may be **capacity bounded**. At any given time it may have a **remainingCapacity** beyond which no additional elements can be put without blocking. A BlockingQueue without any intrinsic(固有) capacity constraints always reports a remaining capacity of **Integer.MAX\_VALUE**.(默认容量为Integer的最大值)

## producer-consumer queues：生产者消费者

**BlockingQueue** implementations are designed to be used **primarily** for **producer-consumer queues**, but additionally support the **Collection** interface. So, for example, it is possible to remove an arbitrary element from a queue using **remove(x).** However, such operations are in general not performed very efficiently, and are intended for only occasional use, such as when a **queued** message is cancelled.

## thread-safe：线程安全的

**BlockingQueue implementations are thread-safe**. All queuing methods achieve their effects atomically using **internal locks or other forms of concurrency control**. However, the bulk Collection operations addAll, containsAll, retainAll and removeAll are not necessarily performed atomically unless specified otherwise in an implementation. So it is possible, for example, for **addAll(c)** to fail (throwing an exception) after adding only some of the elements in c.

## "close" or "shutdown" operation

A BlockingQueue does not intrinsically(固有，本质上) support any kind of **"close" or "shutdown" operation** to indicate that no more items will be added. The needs and usage of such features tend to be **implementation-dependent**. For example, a common **tactic** is for producers to insert special end-of-stream or poison objects, that are interpreted accordingly when taken by consumers.

# 使用示例producer-consumer scenario生产者-消费者场景

阻塞队列经典使用场景就是**生产者-消费者场景。**

Usage example, based on a typical **producer-consumer scenario**. Note that a BlockingQueue can safely be used with multiple producers and multiple consumers.

class **Producer** implements **Runnable** {//生产者

**private final BlockingQueue queue**;//阻塞队列

Producer(BlockingQueue q) { queue = q; }//与消费者同一个阻塞队列

public void run() {

try {

while (true) { **queue.put(produce());** }

} catch (InterruptedException ex) { ... handle ...}

}

Object produce() { ... }

}

class **Consumer** implements **Runnable** {//消费者

private final **BlockingQueue** queue;

Consumer(BlockingQueue q) { queue = q; }//与生产者同一个阻塞队列

public void run() {

try {

while (true) { consume(queue.take()); }

} catch (InterruptedException ex) { ... handle ...}

}

void consume(Object x) { ... }

}

class **Setup** {

void main() {

**BlockingQueue q = new SomeQueueImplementation();**

Producer p = new Producer(q);

Consumer c1 = new Consumer(q);

Consumer c2 = new Consumer(q);

new Thread(p).start();

new Thread(c1).start();

new Thread(c2).start();

}

}

# BlockingQueue接口的方法

**核心方法**：A. 添加元素：**put、offer、add** B. 获取元素：**take、poll、remove** C. **drainTo**

## 添加方法

放入数据：

**offer(anObject):**表示如果可能的话,将anObject加到BlockingQueue里,即如果BlockingQueue可以容纳,则返回true,否则返回false.（本方法不阻塞当前执行方法的线程）

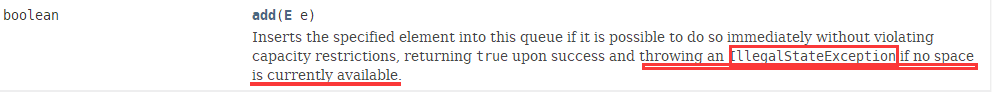
**offer(E o, long timeout, TimeUnit unit),**可以设定等待的时间，如果在指定的时间内，还不能往队列中加入**BlockingQueue**，则返回失败。

**put(anObject):**把anObject加到**BlockingQueue**里,如果BlockQueue没有空间,则调用此方法的线程被阻断，直到BlockingQueue里面有空间再继续.

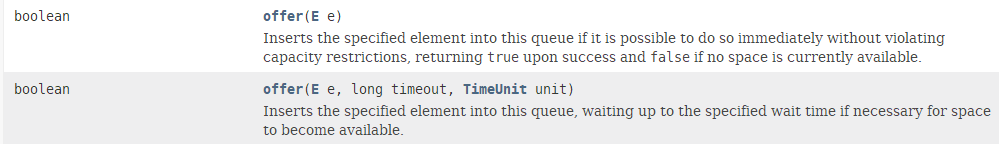
### put方法：Block



### add方法：Throws exception



### offer方法：Special value 和 Timesout



## 获取元素

获取数据：

**poll**(time):取走BlockingQueue里排在首位的对象,若不能立即取出,则可以等time参数规定的时间,取不到时返回null;

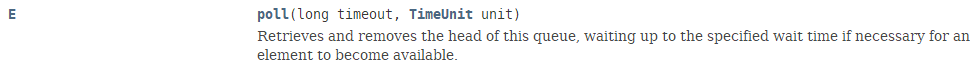
**poll**(long timeout, TimeUnit unit)：从BlockingQueue取出一个队首的对象，如果在指定时间内，队列一旦有数据可取，则立即返回队列中的数据。否则知道时间超时还没有数据可取，返回失败。

**take**(): 取走BlockingQueue里排在首位的对象,若BlockingQueue为空,阻断进入等待状态直到BlockingQueue有新的数据被加入;

### take：Blocks



### poll：Timesout



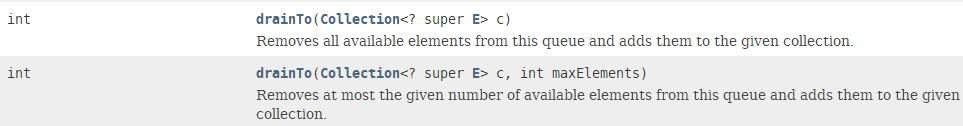
### remove：Throws Exception



## contains：是否含有某个元素



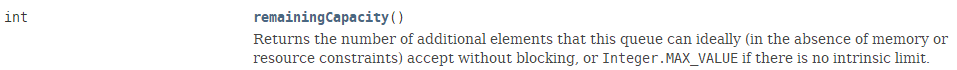
## drainTo



**drainTo**():一次性从BlockingQueue获取所有可用的数据对象（还可以指定获取数据的个数），

　　　　通过该方法，可以提升获取数据效率；不需要多次分批加锁或释放锁。

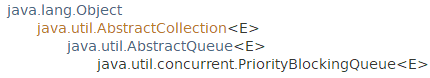
## remainingCapacity：剩余容量



# PriorityBlockingQueue优先级阻塞队列

## 简单介绍

存在于java.util.concurrent包中。



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

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

implements **BlockingQueue**<E>, Serializable

## 特殊功能介绍

### 无界的阻塞队列

优先级阻塞队列是一个无界限的阻塞队列，尽管无界，当无内存分配的时候，会导致内存溢出OutOfMemoryError。

An **unbounded** blocking queue that uses the same ordering rules as class **PriorityQueue** and supplies blocking retrieval operations. While this queue is logically unbounded, attempted additions may fail due to resource exhaustion (causing OutOfMemoryError).

### 不允许null元素

This class does not permit null elements.

### 队列中的元素必须**可比较的**

因为PriorityBlockingQueue队列默认是按照自然顺序排序，因此该队列中的元素必须可比较的，即必须实现了Comparable接口。

A priority queue relying on natural ordering also does not permit insertion of **non-comparable objects** (doing so results **in ClassCastException**).

如public final class Integer extends Number implements **Comparable<Integer>**

## 迭代与遍历

This class and its iterator implement all of the optional methods of the **Collection** and **Iterator** interfaces. The **Iterator** provided in method iterator() is not guaranteed to traverse the elements of the PriorityBlockingQueue in any particular order. If you need **ordered traversal**, consider using **Arrays.sort(pq.toArray())**. Also, method **drainTo** can be used to remove some or all elements in priority order and place them in another collection.

iterator()获得的Iterator对象不保证是指定的顺序，如果需要指定顺序遍历，建议使用**Arrays.sort(pq.toArray());。**

Operations on this class **make no guarantees** about the ordering of elements **with equal priority.** If you need to enforce an ordering, you can define custom classes or comparators that use **a secondary key** to break ties in primary priority values. For example, here is a class that applies first-in-first-out tie-breaking to comparable elements. To use it, you would insert a new **FIFOEntry(anEntry)** instead of a plain entry object.

PriorityBlockingQueue不保证对同等优先级的元素的排序，可以通过指定一个custom class或比较器定义次级比较标准。

class FIFOEntry<E extends Comparable<? super E>>

implements Comparable<FIFOEntry<E>> {

static final AtomicLong seq = new AtomicLong(0);

final long seqNum;

final E entry;

public FIFOEntry(E entry) {

seqNum = seq.getAndIncrement();

this.entry = entry;

}

public E getEntry() { return entry; }

public int compareTo(FIFOEntry<E> other) {

**int res = entry.compareTo(other.entry);**

if (res == 0 && other.entry != this.entry)

res = (seqNum < other.seqNum ? -1 : 1);

return res;

}

}

## 构造方法

参数主要有：1.容量大小initialCapacity；2.集合Collection 3. 比较器Comparator

### **PriorityBlockingQueue**()

Creates a PriorityBlockingQueue with **the default initial capacity (11)** that orders its elements according to their natural ordering.

默认初始容量为11，按照自然顺序排序。

### **PriorityBlockingQueue**(**Collection**<? extends E> c)

Creates a PriorityBlockingQueue containing the elements in the specified collection.

### **PriorityBlockingQueue**(int initialCapacity)

Creates a PriorityBlockingQueue with the specified initial capacity that orders its elements according to their natural ordering.

### **PriorityBlockingQueue**(int initialCapacity, **Comparator**<? super E> comparator)

Creates a PriorityBlockingQueue with the specified initial capacity that orders its elements according to the specified comparator.

## 一般方法介绍

### 添加元素：

put、offer、add同BlockingQueue接口

### 获取元素

take、poll、peek

E take()

Retrieves and removes **the head of this queue**, waiting if necessary until an element becomes available.

### **drainTo**：转到集合中(排水流干)

删除队列中所有元素，并转移到集合中。

int **drainTo**(Collection<? super E> c)

Removes all available elements from this queue and adds them to the given collection.

int **drainTo**(Collection<? super E> c, int **maxElements**)

Removes at most the given number of available elements from this queue and adds them to the given collection.

### comparator：获取比较器

Comparator<? super E> comparator()

Returns the **comparator** used to order the elements in this queue, or **null** if this queue uses the natural ordering of its elements.

### size、remainingCapacity

**size**：获取队列中元素的个数；

Returns the number of elements in this collection.

**remainingCapacity**：返回Integer.MAX\_VALUE，因为优先级阻塞队列无界。

Always returns **Integer.MAX\_VALUE** because a PriorityBlockingQueue is not capacity constrained.

### toArray:转成数组

Object[] toArray()

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

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

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

### iterator 、spliterator

Iterator<E> **iterator**()

Returns an iterator over the elements in this queue.

Spliterator<E> **spliterator**()

Returns a Spliterator over the elements in this queue.

### remove(e)、clear()

boolean remove(Object o)：删除元素

Removes a single instance of the specified element from this queue, if it is present.

void clear():清空队列

Atomically removes all of the elements from this queue.

# DelayQueue延时队列

## 简单介绍

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

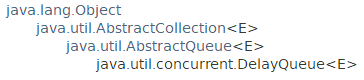
DelayQueue也是Java集合框架中的一个成员。

DelayQueue存在于java.util.concurrent包中。

public class **DelayQueue**<**E extends Delayed**> **extends** AbstractQueue<E>

implements **BlockingQueue**<E>

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



## 特殊功能

### DelayQueue<E extends **Delayed**>

从这里看出DelayQueue中的元素必须是实现了Delayed接口，Delayed接口中只定义一个方法**long getDelay(TimeUnit unit)**。

### 无界限

只能获取延时过期的元素，否则返回null。

延时过期的判断依据：**getDelay(TimeUnit.NANOSECONDS)**方法返回0或负数。

**An unbounded blocking queue** of Delayed elements, in which an element can only be taken when its delay has expired. The head of the queue is that Delayed element whose delay expired furthest in the past. If no delay has expired there is no head and **poll** will return **null**. Expiration occurs when an element's **getDelay(TimeUnit.NANOSECONDS)** method returns a value less than or equal to zero. Even though **unexpired elements** cannot be removed using **take** or **poll**, they are otherwise treated as normal elements. For example, the **size** method returns the count of both expired and unexpired elements.

### 不允许null元素

**This queue does not permit null elements**.

### 迭代与遍历

**iterator()** 获取的迭代器不保证延时队列按照特定顺序获取元素。

This class and its iterator implement all of the optional methods of the **Collection** and **Iterator** interfaces. The Iterator provided in method **iterator()** is not guaranteed to traverse the elements of the **DelayQueue** in any particular order.

## 构造方法

DelayQueue()初始队列为空

Creates a new DelayQueue that is **initially empty.**

DelayQueue(Collection<? extends E> c)

Creates a DelayQueue initially containing the elements of the given collection of Delayed instances.

## 一般方法

### 添加元素：put、offer、add

offer两种：1.返回特定值，2.时间溢出timeout

boolean offer(E e) : Inserts the specified element into this delay queue.

boolean offer(E e, long timeout, TimeUnit unit) : Inserts the specified element into this delay queue.

### 获取元素：take、poll、peek

take：获取并删除；peek只获取不删除。

poll与offer对应，也是两种。

E poll()

Retrieves and removes the head of this queue, or returns null if this queue has no elements with an expired delay.

E poll(long timeout, TimeUnit unit)

Retrieves and removes the head of this queue, waiting if necessary until an element with an expired delay is available on this queue, or the specified wait time expires.

### drainTo

int drainTo(Collection<? super E> c)

Removes all available elements from this queue and adds them to the given collection.

int drainTo(Collection<? super E> c, int maxElements)

Removes at most the given number of available elements from this queue and adds them to the given collection.

### toArray

Object[] **toArray**()

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

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

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

### **size、remainingCapacity**

int size()： Returns the number of elements in this collection.

int remainingCapacity() ：Always returns Integer.MAX\_VALUE because a DelayQueue is not capacity constrained.

### **iterator**

Iterator<E> iterator()

Returns an iterator over all the elements (both expired and unexpired) in this queue.

# Delayed接口

## 简单介绍

public interface **Delayed** extends **Comparable**<Delayed>

父接口All Superinterfaces: Comparable<Delayed>

子接口All Known Subinterfaces: RunnableScheduledFuture<V>, ScheduledFuture<V>

A mix-in style interface for marking **objects that should be acted upon after a given delay**.

An implementation of this interface must define a **compareTo** method that provides an ordering **consistent with its getDelay method**.

Delayed接口的实现类必须实现compareTo和getDelay两个方法。

## 方法

**long getDelay(TimeUnit unit)**

Returns the remaining delay associated with this object, in the given time unit.

**int compareTo(T o)**

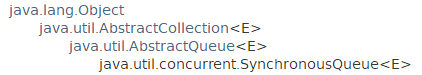
Compares this object with the specified object for order.

# SynchronousQueue同步队列

## 简单介绍

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

implements **BlockingQueue**<E>, Serializable



实现的接口All Implemented Interfaces:

Serializable, Iterable<E>, Collection<E>, **BlockingQueue**<E>, Queue<E>

This class is a member of the Java Collections Framework.

## 功能介绍

### 同步性质

A blocking queue in which each insert operation must wait for a corresponding **remove** operation by another thread, and vice versa. **A synchronous queue** does not have any internal capacity, not even a capacity of one. You cannot peek at a synchronous queue because an element is only present when you try to remove it; you cannot insert an element (using any method) unless another thread is trying to remove it; you cannot iterate as there is nothing to iterate. The head of the queue is the element that the first queued inserting thread is trying to add to the queue; if there is no such queued thread then no element is available for removal and poll() will return null. For purposes of other Collection methods (for example contains), a SynchronousQueue acts as an empty collection.

### 不能插入null元素

This queue does not permit null elements.

**Synchronous queues** are similar to rendezvous channels used in CSP and Ada. They are well suited for handoff designs, in which an object running in one thread must sync up with an object running in another thread in order to hand it some information, event, or task.

### fairness

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.

### iterator

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

## 构造方法2个

SynchronousQueue()

Creates a SynchronousQueue with nonfair access policy.

SynchronousQueue(boolean fair)

Creates a SynchronousQueue with the specified fairness policy.

## 一般方法

SynchronousQueue同步队列中很多方法都是采用了缺省返回值，也就是很多方法是没有意义的。

### 添加元素方法put 、offer

void put(E e) Adds the specified element to this queue, waiting if necessary for another thread to receive it.

**boolean offer(E e)**

Inserts the specified element into this queue, if another thread is waiting to receive it.

**boolean offer(E e, long timeout, TimeUnit unit)**

Inserts the specified element into this queue, waiting if necessary up to the specified wait time for another thread to receive it.

### 获取元素方法take、poll、peek(始终返回null)

E take()

Retrieves and removes the head of this queue, waiting if necessary for another thread to insert it.

E poll()

Retrieves and removes the head of this queue, if another thread is currently making an element available.

E poll(long timeout, TimeUnit unit)

Retrieves and removes the head of this queue, waiting if necessary up to the specified wait time, for another thread to insert it.

E peek() Always returns null.

### drainTo

int drainTo(Collection<? super E> c)

Removes all available elements from this queue and adds them to the given collection.

int drainTo(Collection<? super E> c, int maxElements)

Removes at most the given number of available elements from this queue and adds them to the given collection.

### 删除、情况操作

void clear() Does nothing.

boolean remove(Object o) Always returns false.

boolean removeAll(Collection<?> c) Always returns false.

boolean retainAll(Collection<?> c) Always returns false.

### size、remainingCapacity始终返回0

int size() Always returns zero.

int remainingCapacity() Always returns zero.

boolean isEmpty() Always returns true.

### contains、containsAll

boolean contains(Object o) Always returns false.

boolean containsAll(Collection<?> c) Returns false unless the given collection is empty.

### iterator：返回一个空的迭代器

Iterator<E> iterator() Returns **an empty iterator** in which hasNext always returns false.

Spliterator<E> spliterator()

Returns an empty spliterator in which calls to Spliterator.trySplit() always return null.

### toArray

Object[] **toArray**() Returns **a zero-length array**.

<T> T[] **toArray**(T[] a) Sets the zeroeth element of the specified array to **null** (if the array has non-zero length) and returns it.