JUC集合: BlockingQueue详解

> JUC里的 BlockingQueue 接口表示一个线程安放入和提取实例的队列。

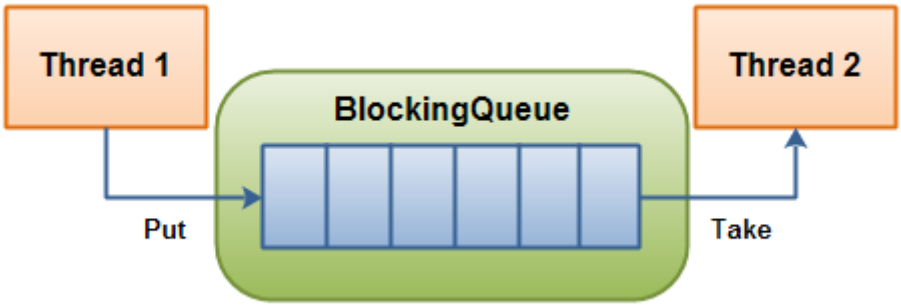
面试问题去理解

- 什么是BlockingDeque?
- BlockingQueue大家族有哪些? ArrayBlockingQueue, DelayQueue, LinkedBlockingQueue, SynchronousQueue...
- BlockingQueue适合用在什么样的场景?
- BlockingQueue常用的方法?
- BlockingQueue插入方法有哪些? 这些方法(`add(o)`,`offer(o)`,`put(o)`,`offer(o, timeout, timeunit)`)的区别是什么?
- BlockingDeque 与BlockingQueue有何关系，请对比下它们的方法?
- BlockingDeque适合用在什么样的场景?
- BlockingDeque大家族有哪些?
- BlockingDeque 与BlockingQueue实现例子?

BlockingQueue和BlockingDeque

BlockingQueue

BlockingQueue 通常用于一个线程生产对象，而另外一个线程消费这些对象的场景。下图是对这个原理的阐述:



一个线程往里边放，另外一个线程从里边取的一个 BlockingQueue。

一个线程将会持续生产新对象并将其插入到队列之中，直到队列达到它所能容纳的临界点。也就是说，它是有限的。如果该阻塞队列到达了其临界点，负责生产的线程将会在往里边插入新对象时发生阻塞。它会一直处于阻塞之中，直到负责消费的线程从队列中拿走一个对象。 负责消费的线程将会一直从该阻塞队列中拿出对象。如果消费线程尝试去从一个空的队列中提取对象的话，这个消费线程将会处于阻塞之中，直到一个生产线程把一个对象丢进队列。

BlockingQueue 的方法

BlockingQueue 具有 4 组不同的方法用于插入、移除以及对队列中的元素进行检查。如果请求的操作不能得到立即执行的话，每个方法的表现也不同。这些方法如下:

	抛异常	特定值	阻塞	超时
插入	add(o)	offer(o)	put(o)	offer(o, timeout, timeunit)
移除	remove(o)	poll(o)	take(o)	poll(timeout, timeunit)
检查	element(o)	peek(o)		

四组不同的行为方式解释:

- 抛异常: 如果试图的操作无法立即执行，抛一个异常。
- 特定值: 如果试图的操作无法立即执行，返回一个特定的值(常常是 true / false)。
- 阻塞: 如果试图的操作无法立即执行，该方法调用将会发生阻塞，直到能够执行。
- 超时: 如果试图的操作无法立即执行，该方法调用将会发生阻塞，直到能够执行，但等待时间不会超过给定值。返回一个特定值以告知该操作是否成功(典型的是 true / false)。

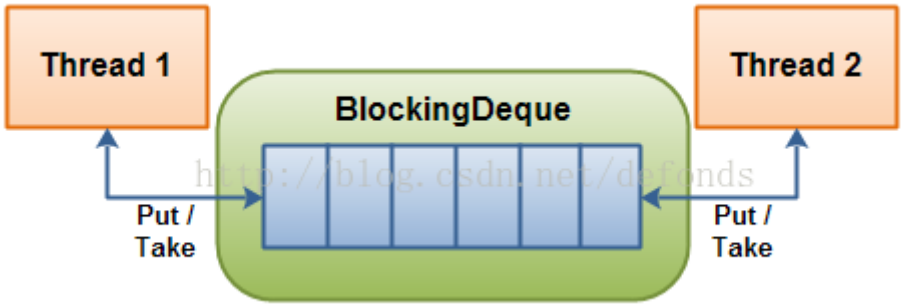
无法向一个 BlockingQueue 中插入 null。如果试图插入 null，BlockingQueue 将会抛出一个 NullPointerException。 可以访问到 BlockingQueue 中的所有元素，而不仅仅是开始和结束的元素。比如说，将一个对象放入队列之中以等待处理，但你的应用想要将其取消掉。那么可以调用诸如 remove(o) 方法来将队列之中的特定对象进行移除。但是这么干效率并不高(译者注: 基于队列的数据结构，获取除开始或结束位置的其他对象的效率不会太高)，因此尽量不要用这一类的方法，除非确实不得不那么做。

BlockingDeque

java.util.concurrent 包里的 BlockingDeque 接口表示一个线程安放入和提取实例的双端队列。

BlockingDeque 类是一个双端队列，在不能够插入元素时，它将阻塞住试图插入元素的线程；在不能够抽取元素时，它将阻塞住试图抽取的线程。 deque(双端队列) 是 "Double Ended Queue" 的缩写。因此，双端队列是一个可以从任意一端插入或者抽取元素的队列。

在线程既是一个队列的生产者又是这个队列的消费者的时候可以使用到 BlockingDeque。如果生产者线程需要在队列的两端都可以插入数据，消费者线程需要在队列的两端都可以移除数据，这个时候也可以使用 BlockingDeque。BlockingDeque 图解:



BlockingDeque 的方法

一个 BlockingDeque - 线程在双端队列的两端都可以插入和提取元素。 一个线程生产元素，并把它们插入到队列的任意一端。如果双端队列已满，插入线程将被阻塞，直到一个移除线程从该队列中移出了一个元素。如果双端队列为空，移除线程将被阻塞，直到一个插入线程向该队列插入了一个新元素。

BlockingDeque 具有 4 组不同的方法用于插入、移除以及对双端队列中的元素进行检查。如果请求的操作不能得到立即执行的话，每个方法的表现也不同。这些方法如下:

	抛异常	**特定值**	**阻塞**	**超时**
插入	addFirst(o)	offerFirst(o)	putFirst(o)	offerFirst(o, timeout, timeunit)
移除	removeFirst(o)	pollFirst(o)	takeFirst(o)	pollFirst(timeout, timeunit)
检查	getFirst(o)	peekFirst(o)		

	抛异常	**特定值**	**阻塞**	**超时**
插入	addLast(o)	offerLast(o)	putLast(o)	offerLast(o, timeout, timeunit)
移除	removeLast(o)	pollLast(o)	takeLast(o)	pollLast(timeout, timeunit)
检查	getLast(o)	peekLast(o)		

四组不同的行为方式解释:

- 抛异常: 如果试图的操作无法立即执行，抛一个异常。
- 特定值: 如果试图的操作无法立即执行，返回一个特定的值(常常是 true / false)。
- 阻塞: 如果试图的操作无法立即执行，该方法调用将会发生阻塞，直到能够执行。
- 超时: 如果试图的操作无法立即执行，该方法调用将会发生阻塞，直到能够执行，但等待时间不会超过给定值。返回一个特定值以告知该操作是否成功(典型的是 true / false)。

BlockingDeque 与*BlockingQueue*关系

BlockingDeque 接口继承自 BlockingQueue 接口。这就意味着你可以像使用一个 BlockingQueue 那样使用 BlockingDeque。如果你这么干的话，各种插入方法将会把新元素添加到双端队列的尾端，而移除方法将会把双端队列的首端的元素移除。正如 BlockingQueue 接口的插入和移除方法一样。

以下是 BlockingDeque 对 BlockingQueue 接口的方法的具体内部实现:

BlockingQueue	**BlockingDeque**
add()	addLast()
offer() x 2	offerLast() x 2
put()	putLast()
remove()	removeFirst()
poll() x 2	pollFirst()
take()	takeFirst()

BlockingQueue	BlockingDeque
element()	getFirst()
peek()	peekFirst()

BlockingQueue 的例子

这里是一个 Java 中使用 BlockingQueue 的示例。本示例使用的是 BlockingQueue 接口的 ArrayBlockingQueue 实现。首先，BlockingQueueExample 类分别在两个独立的线程中启动了一个 Producer 和 一个 Consumer。Producer 向一个共享的 BlockingQueue 中注入字符串，而 Consumer 则会从中把它们拿出来。

```
public class BlockingQueueExample {

    public static void main(String[] args) throws Exception {

        BlockingQueue queue = new ArrayBlockingQueue(1024);

        Producer producer = new Producer(queue);
        Consumer consumer = new Consumer(queue);

        new Thread(producer).start();
        new Thread(consumer).start();

        Thread.sleep(4000);
    }
}
```

以下是 Producer 类。注意它在每次 put() 调用时是如何休眠一秒钟的。这将导致 Consumer 在等待队列中对象的时候发生阻塞。

```
public class Producer implements Runnable{

    protected BlockingQueue queue = null;

    public Producer(BlockingQueue queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            queue.put("1");
            Thread.sleep(1000);
            queue.put("2");
            Thread.sleep(1000);
            queue.put("3");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

以下是 Consumer 类。它只是把对象从队列中抽取出来，然后将它们打印到 System.out。

```
public class Consumer implements Runnable{
```

```
    protected BlockingQueue queue = null;

    public Consumer(BlockingQueue queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            System.out.println(queue.take());
            System.out.println(queue.take());
            System.out.println(queue.take());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

数组阻塞队列 *ArrayBlockingQueue*

ArrayBlockingQueue 类实现了 BlockingQueue 接口。

ArrayBlockingQueue 是一个有界的阻塞队列，其内部实现是将对象放到一个数组里。有界也就意味着，它不能够存储无限多数量的元素。它有一个同一时间能够存储元素数量的上限。你可以在对其初始化的时候设定这个上限，但之后就无法对这个上限进行修改了(译者注: 因为它是基于数组实现的，也就具有数组的特性: 一旦初始化，大小就无法修改)。 ArrayBlockingQueue 内部以 FIFO(先进先出)的顺序对元素进行存储。队列中的头元素在所有元素之中是放入时间最久的那个，而尾元素则是最短的那个。 以下是在使用 ArrayBlockingQueue 的时候对其初始化的一个示例:

```
BlockingQueue queue = new ArrayBlockingQueue(1024);
queue.put("1");
Object object = queue.take();
```

以下是使用了 Java 泛型的一个 BlockingQueue 示例。注意其中是如何对 String 元素放入和提取的:

```
BlockingQueue<String> queue = new ArrayBlockingQueue<String>(1024);
queue.put("1");
String string = queue.take();
```

延迟队列 *DelayQueue*

DelayQueue 实现了 BlockingQueue 接口。

DelayQueue 对元素进行持有直到一个特定的延迟到期。注入其中的元素必须实现 java.util.concurrent.Delayed 接口，该接口定义:

```
public interface Delayed extends Comparable<Delayed< {
    public long getDelay(TimeUnit timeUnit);
}
```

DelayQueue 将会在每个元素的 getDelay() 方法返回的值的时间段之后才释放掉该元素。如果返回的是 0 或者负值，延迟将被认为过期，该元素将会在 DelayQueue 的下一次 take 被调用的时候被释放掉。

传递给 getDelay 方法的 getDelay 实例是一个枚举类型，它表明了将要延迟的时间段。TimeUnit 枚举将会取以下值:

- DAYS
- HOURS
- INUTES
- SECONDS
- MILLISECONDS
- MICROSECONDS
- NANOSECONDS

正如你所看到的，Delayed 接口也继承了 java.lang.Comparable 接口，这也就意味着 Delayed 对象之间可以进行对比。这个可能在对 DelayQueue 队列中的元素进行排序时有用，因此它们可以根据过期时间进行有序释放。 以下是使用 DelayQueue 的例子:

```
public class DelayQueueExample {

    public static void main(String[] args) {
        DelayQueue queue = new DelayQueue();
        Delayed element1 = new DelayedElement();
        queue.put(element1);
        Delayed element2 = queue.take();
    }
}
```

DelayedElement 是我所创建的一个 DelayedElement 接口的实现类，它不在 java.util.concurrent 包里。你需要自行创建你自己的 Delayed 接口的实现以使用 DelayQueue 类。

链阻塞队列 *LinkedBlockingQueue*

LinkedBlockingQueue 类实现了 BlockingQueue 接口。

LinkedBlockingQueue 内部以一个链式结构(链接节点)对其元素进行存储。如果需要的话，这一链式结构可以选择一个上限。如果没有定义上限，将使用 Integer.MAX_VALUE 作为上限。

LinkedBlockingQueue 内部以 FIFO(先进先出)的顺序对元素进行存储。队列中的头元素在所有元素之中是放入时间最久的那个，而尾元素则是最短的那个。 以下是 LinkedBlockingQueue 的初始化和使用示例代码:

```
BlockingQueue<String> unbounded = new LinkedBlockingQueue<String>();
BlockingQueue<String> bounded   = new LinkedBlockingQueue<String>(1024);
bounded.put("Value");
String value = bounded.take();
```

具有优先级的阻塞队列 *PriorityBlockingQueue*

PriorityBlockingQueue 类实现了 BlockingQueue 接口。

PriorityBlockingQueue 是一个无界的并发队列。它使用了和类 java.util.PriorityQueue 一样的排序规则。你无法向这个队列中插入 null 值。 所有插入到 PriorityBlockingQueue 的元素必须实现 java.lang.Comparable 接口。因此该队列中元素的排序就取决于你自己的 Comparable 实现。 注意 PriorityBlockingQueue 对于具有相等优先级(compare() == 0)的元素并不强制任何特定行为。

同时注意，如果你从一个 PriorityBlockingQueue 获得一个 Iterator 的话，该 Iterator 并不能保证它对元素的遍历是以优先级为序的。 以下是使用 PriorityBlockingQueue 的示例:

```
BlockingQueue queue   = new PriorityBlockingQueue();
//String implements java.lang.Comparable
queue.put("Value");
String value = queue.take();
```

同步队列 *SynchronousQueue*

SynchronousQueue 类实现了 BlockingQueue 接口。

SynchronousQueue 是一个特殊的队列，它的内部同时只能够容纳单个元素。如果该队列已有一元素的话，试图向队列中插入一个新元素的线程将会阻塞，直到另一个线程将该元素从队列中抽走。同样，如果该队列为空，试图向队列中抽取一个元素的线程将会阻塞，直到另一个线程向队列中插入了一条新的元素。 据此，把这个类称作一个队列显然是夸大其词了。它更多像是一个汇合点。

BlockingDeque 的例子

既然 BlockingDeque 是一个接口，那么你想要使用它的话就得使用它的众多的实现类的其中一个。java.util.concurrent 包提供了以下 BlockingDeque 接口的实现类: LinkedBlockingDeque。

以下是如何使用 BlockingDeque 方法的一个简短代码示例:

```
BlockingDeque<String> deque = new LinkedBlockingDeque<String>();
deque.addFirst("1");
deque.addLast("2");

String two = deque.takeLast();
String one = deque.takeFirst();
```

链阻塞双端队列 *LinkedBlockingDeque*

LinkedBlockingDeque 类实现了 BlockingDeque 接口。

deque(双端队列) 是 "Double Ended Queue" 的缩写。因此，双端队列是一个你可以从任意一端插入或者抽取元素的队列。

LinkedBlockingDeque 是一个双端队列，在它为空的时候，一个试图从中抽取数据的线程将会阻塞，无论该线程是试图从哪一端抽取数据。

以下是 LinkedBlockingDeque 实例化以及使用的示例:

```
BlockingDeque<String> deque = new LinkedBlockingDeque<String>();
deque.addFirst("1");
deque.addLast("2");

String two = deque.takeLast();
String one = deque.takeFirst();
```