二

36 有哪几种常见的阻塞队列?

本课时我们主要讲解有哪几种常见的阻塞队列。

BlockingQueue 接口的实现类都被放在了 J.U.C 包中，本课时将对常见的和常用的实现类进行介绍，包括 ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue、PriorityBlockingQueue，以及 DelayQueue。

ArrayBlockingQueue

让我们先从最基础的 ArrayBlockingQueue 说起。ArrayBlockingQueue 是最典型的**有界队列**，其内部是用数组存储元素的，利用 ReentrantLock 实现线程安全。

我们在创建它的时候就需要指定它的容量，之后也不可以再扩容了，在构造函数中我们同样可以指定是否是公平的，代码如下：

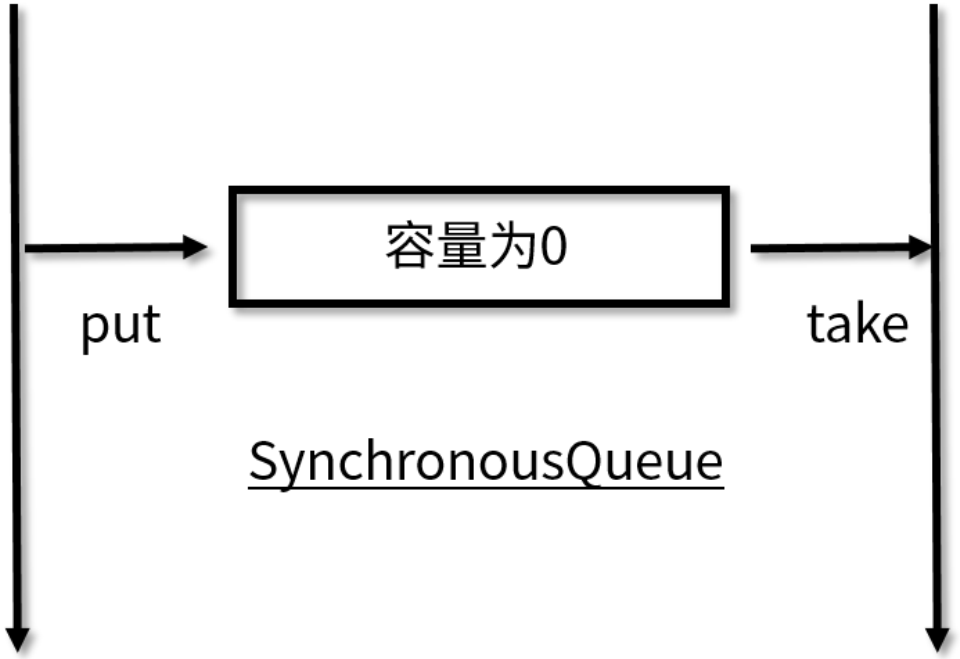
```
ArrayBlockingQueue(int capacity, boolean fair)
```

第一个参数是容量，第二个参数是是否公平。正如 ReentrantLock 一样，如果 ArrayBlockingQueue 被设置为非公平的，那么就存在插队的可能；如果设置为公平的，那么等待了最长时间的线程会被优先处理，其他线程不允许插队，不过这样的公平策略同时会带来一定的性能损耗，因为非公平的吞吐量通常会高于公平的情况。

LinkedBlockingQueue

正如名字所示，这是一个内部用链表实现的 BlockingQueue。如果我们不指定它的初始容量，那么它容量默认就为整型的最大值 Integer.MAX_VALUE，由于这个数非常大，我们通常不可能放入这么多的数据，所以 LinkedBlockingQueue 也被称作无界队列，代表它几乎没有界限。

SynchronousQueue



如图所示，SynchronousQueue 最大的不同之处在于，它的容量为 0，所以没有一个地方来暂存元素，导致每次取数据都要先阻塞，直到有数据被放入；同理，每次放数据的时候也会阻塞，直到有消费者来取。

需要注意的是，SynchronousQueue 的容量不是 1 而是 0，因为 SynchronousQueue 不需要去持有元素，它所做的就是直接传递（direct handoff）。由于每当需要传递的时候，SynchronousQueue 会把元素直接从生产者传给消费者，在此期间并不需要做存储，所以如果运用得当，它的效率是很高的。

另外，由于它的容量为 0，所以相比于一般的阻塞队列，SynchronousQueue 的很多方法的实现是很有意思的，我们来举几个例子：

SynchronousQueue 的 peek 方法永远返回 null，代码如下：

```
public E peek() {
    return null;
}
```

因为 peek 方法的含义是取出头结点，但是 SynchronousQueue 的容量是 0，所以连头结点都没有，peek 方法也就没有意义，所以始终返回 null。同理，element 始终会抛出 NoSuchElementException 异常。

而 SynchronousQueue 的 size 方法始终返回 0，因为它内部并没有容量，代码如下：

```
public int size() {
    return 0;
}
```

直接 return 0，同理，isEmpty 方法始终返回 true：

```
public boolean isEmpty() {
    return true;
}
```

因为它始终都是空的。

PriorityBlockingQueue

前面我们所说的 ArrayBlockingQueue 和 LinkedBlockingQueue 都是采用先进先出的顺序进行排序，可是如果有的时候我们需要自定义排序怎么办呢？这时就需要使用 PriorityBlockingQueue。

PriorityBlockingQueue 是一个支持优先级的无界阻塞队列，可以通过自定义类实现 compareTo() 方法来指定元素排序规则，或者初始化时通过构造器参数 Comparator 来指定排序规则。同时，插入队列的对象必须是可比较大小的，也就是 Comparable 的，否则会抛出 ClassCastException 异常。

它的 take 方法在队列为空的时候会阻塞，但是正因为它是无界队列，而且会自动扩容，所以它的队列永远不会满，所以它的 put 方法永远不会阻塞，添加操作始终都会成功，也正因为如此，它的成员变量里只有一个 Condition：

```
private final Condition notEmpty;
```

这和之前的 ArrayBlockingQueue 拥有两个 Condition（分别是 notEmpty 和 notFull）形成了鲜明的对比，我们的 PriorityBlockingQueue 不需要 notFull，因为它永远都不会满，真是“有空间就可以任性”。

DelayQueue

DelayQueue 这个队列比较特殊，具有“延迟”的功能。我们可以设定让队列中的任务延迟多久之后执行，比如 10 秒钟之后执行，这在例如“30 分钟后未付款自动取消订单”等需要延迟

执行的场景中被大量使用。

它是无界队列，放入的元素必须实现 Delayed 接口，而 Delayed 接口又继承了 Comparable 接口，所以自然就拥有了比较和排序的能力，代码如下：

```
public interface Delayed extends Comparable<Delayed> {

    long getDelay(TimeUnit unit);

}
```

可以看出这个 Delayed 接口继承自 Comparable，里面有一个需要实现的方法，就是 getDelay。这里的 getDelay 方法返回的是“还剩下多长的延迟时间才会被执行”，如果返回 0 或者负数则代表任务已过期。

元素会根据延迟时间的长短被放到队列的不同位置，越靠近队列头代表越早过期。

DelayQueue 内部使用了 PriorityQueue 的能力来进行排序，而不是自己从头编写，我们在工作中可以学习这种思想，对已有的功能进行复用，不但可以减少开发量，同时避免了“重复造轮子”，更重要的是，对学到的知识进行合理的运用，让知识变得更灵活，做到触类旁通。

总结

以上就是本课时的内容，我们对于 ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue、PriorityBlockingQueue 以及 DelayQueue 这些常见的和常用的阻塞队列的特点进行了讲解。

上一页 下一页