Java BlockingQueue 数据结构及其相关

deepwaterooo

May 21, 2022

Contents

1 java.util.concurrent.ArrayBlockingQueue 原理简要分析 **1**
1.1 一、引言 1
1.1.1 它和 LinkedBlockingQueue 存在以下几个不同点: 1
1.2 二、原理分析 2
1.2.1 1、构造方法 2
1.2.2 2、添加元素 3
1.2.3 3、获取元素 3

- https://blog.csdn.net/abc123lzf/article/details/82702123
- https://juejin.cn/post/6844903602444582920
- https://juejin.cn/post/6844903602448760845
- https://juejin.cn/post/6844903602423595015
- https://kkewwei.github.io/elasticsearch_learning/2017/10/02/Condition%E5%8E%9F%E7%90%86%E8%A7%A3%E8%AF%BB/
- https://blog.csdn.net/mocas_wang/article/details/108476505
- https://blog.csdn.net/weixin_54499878/article/details/117924412

1 java.util.concurrent.ArrayBlockingQueue 原理简要分析

1.1 一、引言

- ArrayBlockingQueue 是一个基于静态数组的阻塞队列，可用于实现生产-消费者模型。

1.1.1 它和 LinkedBlockingQueue 存在以下几个不同点:

1. 1、锁的实现不同
   - ArrayBlockingQueue 的入队和出队都是使用的一个锁，意味着只能有一个线程来修改它。
   - LinkedBlockingQueue 采用了两把锁：入队锁和出队锁，可以有一个线程负责生产，一个线程负责消费而不阻塞。
2. 2、内部保存对象的方式不同
   - ArrayBlockingQueue 在加入元素时，直接将元素添加到数组。

- LinkedBlockingQueue 在加入元素时，需要把对象封装成内部类 Node 并拼接到链表尾部。

3. 3、构造阶段
   - ArrayBlockingQueue 在构造阶段必须指定队列最大长度
   - LinkedBlockingQueue 在构造阶段无须指定最大长度(默认最大长度为 Integer.MAX_VALUE)
4. 4、锁的公平
   - ArrayBlockingQueue 可以实现公平锁，而 LinkedBlockingQueue 则只能使用非公平锁

1.2 二、原理分析

```
public class ArrayBlockingQueue<E> extends AbstractQueue<E>
        implements BlockingQueue<E>, java.io.Serializable {
    //存储元素的数组
    final Object[] items;
    //下一个出队元素的数组下标
    int takeIndex;
    //下一个入队元素的数组下标
    int putIndex;
    //元素数量
    int count;
    //锁
    final ReentrantLock lock;
    //用来等待、通知尝试获取元素的线程
    private final Condition notEmpty;
    //用来等待、通知尝试添加元素的线程
    private final Condition notFull;
    //迭代器和这个队列更新数据的中间体
    transient Itrs itrs = null;
}
```

- ArrayBlockingQueue 在内部实现了一个静态数组来存储元素，并通过 takeIndex 和 putIndex 来实现元素的快速入队出队。在并发方面，ArrayBlockingQueue 使用了一个重入锁来保证并发安全性，和 LinkedBlockingQueue 一样采用两个 Condition 用来通知入队出队线程。

1.2.1 1、构造方法

- ArrayBlockingQueue 提供了三种 public 构造方法：

构造方法	解释
ArrayBlockingQueue(int capacity)	构造一个最大大小为 cap
ArrayBlockingQueue(int capacity, boolean fair)	同上，若 fair 为 true 则
ArrayBlockingQueue(int capacity, boolean fair, Collection<? extends E> c)	同上，构造时默认将集合

```
public ArrayBlockingQueue(int capacity, boolean fair) {
    if (capacity <= 0)
        throw new IllegalArgumentException();
    //分配一个 capacity 长度的数组
    this.items = new Object[capacity];
    //创建一个重入锁
    lock = new ReentrantLock(fair);
    //获取这个锁对应的 Condition
    notEmpty = lock.newCondition();
    notFull =  lock.newCondition();
}
```

- 这个构造方法比较简单，主要是完成实例变量的赋值操作

```
public ArrayBlockingQueue(int capacity, boolean fair, Collection<? extends E> c) {
    this(capacity, fair);
    final ReentrantLock lock = this.lock;
    lock.lock();
    try {
        int i = 0;
        try {
            for (E e : c) {
                checkNotNull(e);
                items[i++] = e;
            }
        } catch (ArrayIndexOutOfBoundsException ex) {
            throw new IllegalArgumentException();
        }
        //更新元素数量计数器
        count = i;
        //更新出队指针
        putIndex = (i == capacity) ? 0 : i;
    } finally {
        lock.unlock();
    }
}
```

- 上述构造方法在完成变量的赋值操作后，还会将集合 c 中所有元素加入到队列中。但需要注意的是：1、集合不能有 null 元素，否则会抛出 NullPointerException。2、集合元素的数量不能超过这个队列的最大长度，否则会抛出 IllegalArgumentException。

1.2.2 2、添加元素

- ArrayBlockingQueue 提供了一下 API 来添加元素：

方法	作用
boolean add(E e)	尝试调用 offer 添加元素，添加失败抛出 IllegalState
boolean offer(E e)	无阻塞地添加元素，如果队列已满则直接返回 false
boolean offer(E e, long timeout, TimeUnit unit)	阻塞地添加元素，如果队列已满但最多等待 timeout
void put(E e)	阻塞地添加元素，如果队列已满会阻塞到被 interrupt

- 其中，put 方法和 offer(E,long,TimeUnit) 在阻塞过程中可被 interrupt。

1. put 方法分析

```
public void put(E e) throws InterruptedException {
    checkNotNull(e);
    final ReentrantLock lock = this.lock;
    //加锁，保证线程安全
    lock.lockInterruptibly();
    try {
        //当队列已满时会调用 await 使当前线程等待
        while (count == items.length)
            notFull.await();
        //入队
        enqueue(e);
    } finally {
        lock.unlock();
    }
}
```

- ArrayBlockingQueue 采用内部方法 enqueue 来完成入队操作：

```
private void enqueue(E x) {
    final Object[] items = this.items;
    //将元素 x 放入 putIndex 位置
    items[putIndex] = x;
    //增加入队下标，若等于入队长度则从 0 开始
    if (++putIndex == items.length)
        putIndex = 0;
```

```
    //增加数组元素
    count++;
    //激活一个等待获取元素的线程
    notEmpty.signal();
}
```

- enqueue 方法直接将元素插入到数组的 putIndex 位置，并将 putIndex 加 1（或设为 0），然后激活一个等待元素的线程。

1.2.3 3、获取元素

- ArrayBlockingQueue 提供了一下 API 来获取元素：

方法	作用
E poll()	获取元素并删除队首元素 (出队)
E take()	获取元素并删除队首元素 (出队), 若队列没有元素则阻塞
E poll(long timeout, TimeUnit unit)	获取元素并删除队首元素 (出队), 若队列没有元素则至多等待 time
E peek()	获取队首元素，如果队列为空返回 null

1. take 方法分析

```
public E take() throws InterruptedException {
    final ReentrantLock lock = this.lock;
    lock.lockInterruptibly();
    try {
        //如果队列为空则让当前线程等待
        while (count == 0)
            notEmpty.await();
        return dequeue();
    } finally {
        lock.unlock();
    }
}
```

- ArrayBlockingQueue 采用内部方法 enqueue 来完成出队操作：

```
private E dequeue() {
    final Object[] items = this.items;
    @SuppressWarnings("unchecked")
    //根据 takeIndex 获取元素
    E x = (E) items[takeIndex];
    //删除数组中的 takeIndex 位置的元素
    items[takeIndex] = null;
    //takeIndex 下标加 1
    if (++takeIndex == items.length)
        takeIndex = 0;
    //元素数量计数器减 1
    count--;
    if (itrs != null)
        itrs.elementDequeued();
    //激活一个等待入队的线程
    notFull.signal();
    return x;
}
```