```
public class LinkedBlockingQueue<E> extends AbstractQueue<E>
    implements BlockingQueue<E>, java.io.Serializable {
...
private final int capacity;
private final AtomicInteger count = new AtomicInteger(0);  //原子变量
    private transient Node<E> head;    //单向链表的头部和尾部
    private transient Node<E> last;
    //2 把锁 + 2 个条件
    private final ReentrantLock takeLock = new ReentrantLock();
    private final Condition notEmpty = takeLock.newCondition();
    private final ReentrantLock putLock = new ReentrantLock();
private final Condition notFull = putLock.newCondition();
...
}
```

在其构造函数中，也可以指定队列的总容量。如果不指定，默认为 Integer.MAX_VALUE。

```
public LinkedBlockingQueue() {
this(Integer.MAX_VALUE);
}
public LinkedBlockingQueue(int capacity) {
if (capacity <= 0) throw new IllegalArgumentException();
this.capacity = capacity;
last = head = new Node<E>(null);
}
```

下面看一下其 put/take 实现。

```
public void put(E e) throws InterruptedException {
if (e == null) throw new NullPointerException();
int c = -1;
Node<E> node = new Node(e);
final ReentrantLock putLock = this.putLock;
final AtomicInteger count = this.count;
putLock.lockInterruptibly();
try {
    while (count.get() == capacity) {
        notFull.await();
    }
    enqueue(node);
    c = count.getAndIncrement();
    if (c + 1 < capacity)
        notFull.signal();    //通知其他 put 线程
} finally {
    putLock.unlock();
```