# JAVA并发编程深度学习-无锁并行计算框架

## 并发编程与无锁并行计算框架初探

|  |
| --- |
| **public interface** Constants {   **int *EVENT\_NUM\_OHM*** = 100000000;    **int *EVENT\_NUM\_FM*** = 50000000;    **int *EVENT\_NUM\_OM*** = 10000000;   } |

|  |
| --- |
| **public class** ArrayBlockingQueue4Test {  **public static void** main(String[] args) {  **final** ArrayBlockingQueue<Message> queue = **new** ArrayBlockingQueue<>(Constants.***EVENT\_NUM\_OM***);  **final long** startTime = System.*currentTimeMillis*();  **new** Thread(()->{  **long** i = 0;  **while** (i < Constants.EVENT\_NUM\_OHM) {  Message message = **new** Message(i, **"c"** + i);  **try** {  queue.put(message);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  i++;  }  }).start();   **new** Thread(()->{  **int** k = 0;  **while** (k < Constants.EVENT\_NUM\_OHM) {  **try** {  queue.take();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  k++;  }  **long** endTime = System.currentTimeMillis();  System.out.println(**"ArrayBlockingQueue costTime = "** + (endTime - startTime) + **"ms"**);  }).start();  } } |
| ArrayBlockingQueue costTime = 27912ms |

|  |
| --- |
| **public class** DataConsumer **implements** EventHandler<Message> {   **private long startTime**;  **private int i**;   **public** DataConsumer() {  **this**.**startTime** = System.*currentTimeMillis*();  }   **public void** onEvent(Message data, **long** seq, **boolean** bool)  **throws** Exception {  **i**++;  **if** (**i** == Constants.***EVENT\_NUM\_FM***) {  **long** endTime = System.*currentTimeMillis*();  System.***out***.println(**"Disruptor costTime = "** + (endTime - **startTime**) + **"ms"**);  }  } } |

|  |
| --- |
| **public class** DisruptorSingle4Test {  **public static void** main(String[] args) {  **int** ringBufferSize = 65536;  **final** Disruptor<Message> disruptor = **new** Disruptor<>(  **new** EventFactory<Message>() {  @Override  **public** Message newInstance() {  **return new** Message();  }  },  ringBufferSize,  Executors.*newSingleThreadExecutor*(),  ProducerType.***SINGLE***,  **new** YieldingWaitStrategy()  );  DataConsumer consumer = **new** DataConsumer();  disruptor.handleEventsWith(consumer);  disruptor.start();  **new** Thread(()->{  RingBuffer<Message> ringBuffer = disruptor.getRingBuffer();  **for** (**long** i = 0; i < Constants.EVENT\_NUM\_FM; i++) {  **long** seq = ringBuffer.next();  Message data = ringBuffer.get(seq);  data.setId(i);  data.setName(**"c"** + i);  ringBuffer.publish(seq);  }  }).start();  } } |
| Disruptor costTime = 7458ms |