

总的 Disruptor 的使用场景

Disruptor 它可以用来作为高性能的有界内存队列，适用于两大场景：

- 生产者消费者场景
- 发布订阅 场景

生产者消费者场景。Disruptor 的最常用的场景就是“生产者 - 消费者”场景，对场景的就是“一个生产者、多个消费者”的场景，并且要求顺序处理。

备注，这里和 JCTool 的 MPSC 队列，刚好相反，MPSC 使用于多生产者，单消费者场景

发布订阅 场景：Disruptor 也可以认为是观察者模式的一种实现，实现发布订阅模式。

当前业界开源组件使用 Disruptor 的包括 Log4j2、Apache Storm 等，

Disruptor 使用细分场景

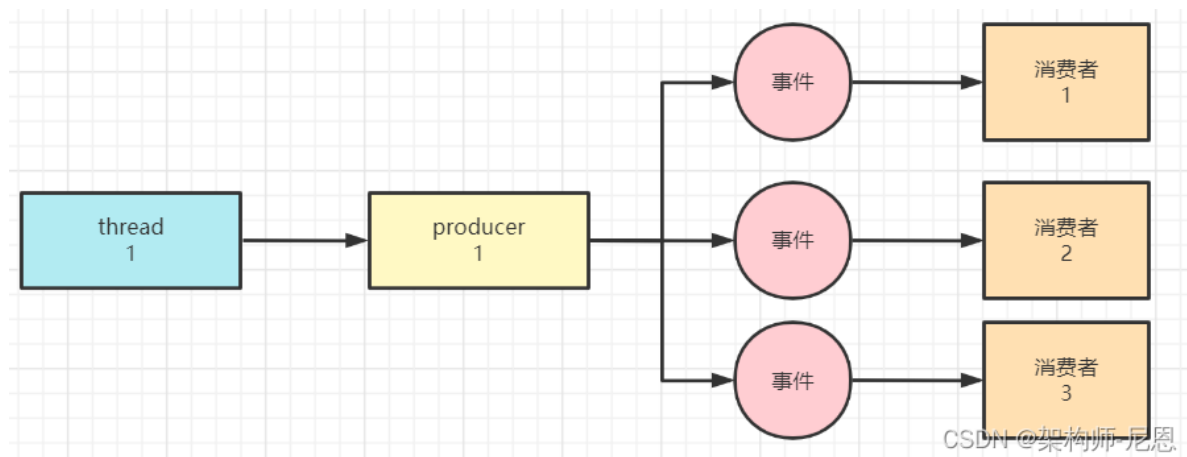
Disruptor 是一个优秀的并发框架，可以使用在多个生产者单消费者场景

- 单生产者多消费者场景
- 多生产者单消费者场景
- 单生产者多消费者场景
- 多个消费者串行消费场景
- 菱形方式执行场景
- 链式并行执行场景
- 多组消费者相互隔离场景
- 多组消费者航道执行模式

单生产者多消费者并行场景

在并发系统中提高性能最好的方式之一就是单一写者原则，对 Disruptor 也是适用的。

如果在生产者单消费者 需求中仅仅有一个事件生产者，那么可以设置为单一生产者模式来提高系统的性能。



ProducerType 的类型

ProducerType 定义了生产者的类型，两类

```
1  .../
16 package com.lmax.disruptor.dsl;
17
18 /**
19  * Defines producer types to support creation of RingBuffer with correct sequencer and publisher.
20  */
21 public enum ProducerType
22 {
23     /**
24      * Create a RingBuffer with a single event publisher to the RingBuffer
25      */
26     SINGLE,
27
28     /**
29      * Create a RingBuffer supporting multiple event publishers to the one RingBuffer
30      */
31     MULTI
32 }
33
```

CSDN @架构师-尼恩

在这种场景下，ProducerType 的类型的 SINGLE

说明：本文会以 pdf 格式持续更新，更多最新尼恩 3 高 pdf 笔记，请从下面的链接获取：[语雀](#) 或者 [码云](#)

单生产者多消费者并行场景的参考代码

参考的代码如下：

```
@org.junit.Test
public void testSimpleProducerDisruptorWithMethodRef() throws InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小，2的指数
    int bufferSize = 1024;
    // 构造 分裂者（事件分发者）
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
        executor,
        ProducerType.SINGLE, //多个生产者
        new YieldingWaitStrategy());
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler
    disruptor.handleEventsWith(LongEventSceneDemo::handleEvent,
        LongEventSceneDemo::handleEvent, LongEventSceneDemo::handleEvent);
    // 开启 分裂者（事件分发）
    disruptor.start();
    // 获取环形队列，用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();

    //1生产者，并发生产数据
    LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override public void run() {
            for (long i = 0; true; i++) {
                /*发布事件*/producer.onData(i);
                ThreadUtil.sleepSeconds(1);
            }
        }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}
```

CSDN @架构师-尼恩

执行结果：

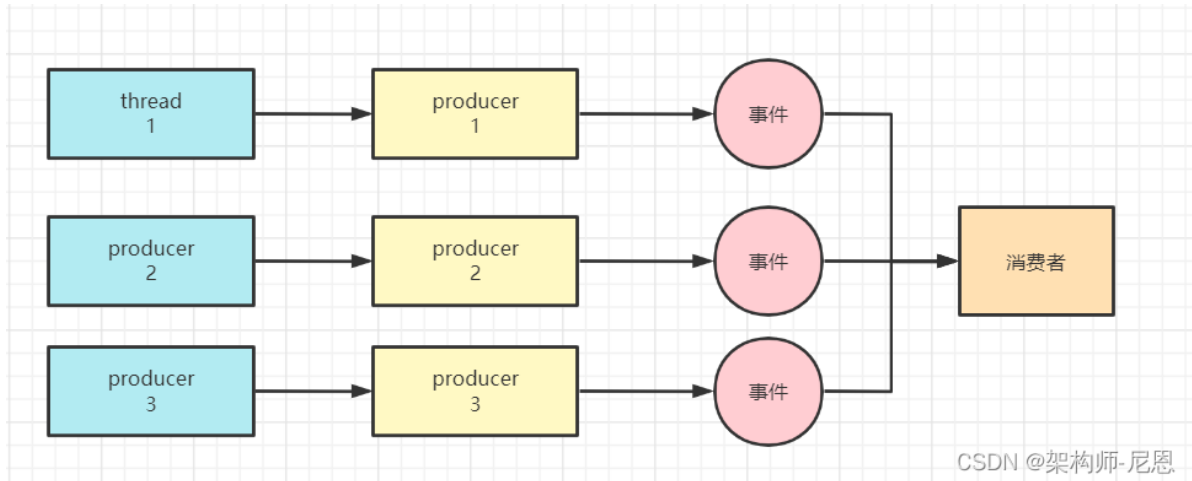
```
Tests passed: 1 of 1 test - 5 s 59 ms
D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:56848', transport: 'socket'
0
0
0
1
1
1
1
2
2
2
3
3
3
4
4
4
4
Disconnected from the target VM, address: '127.0.0.1:56848', transport: 'socket'
Process finished with exit code 0
```

CSDN @架构师-尼恩

以上用例的具体减少，请参见 尼恩 《100wqps 日志平台实操，视频》

多生产者单消费者场景

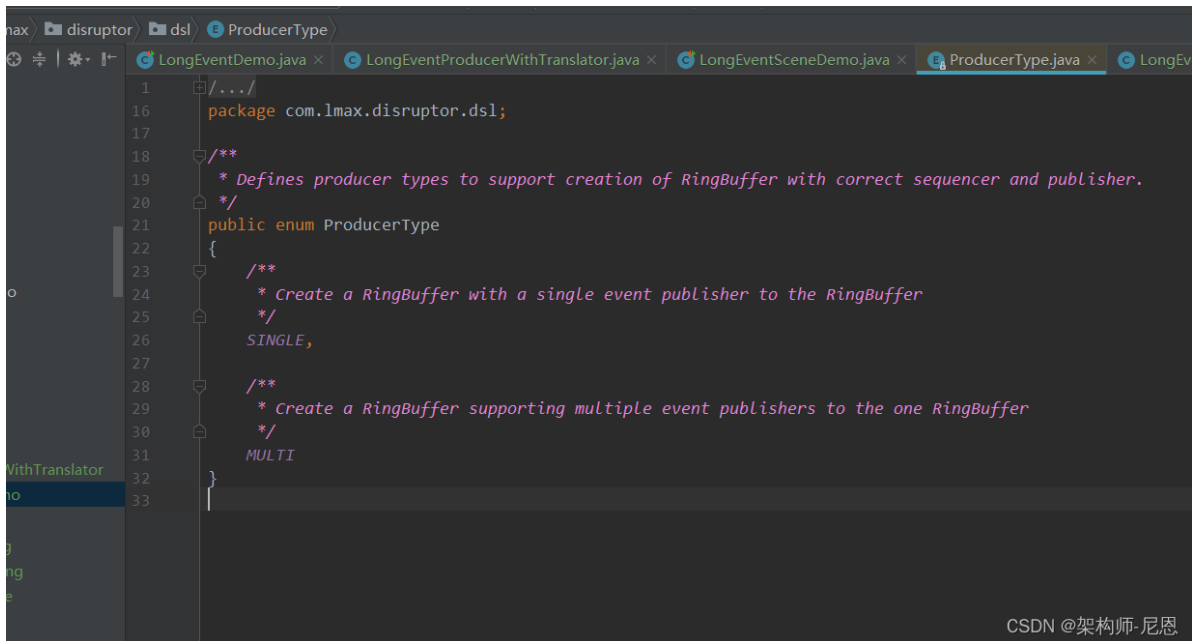
该场景较为简单，就是多个生产者，单个消费者



其实，消费者也可以是多个

ProducerType 的类型

ProducerType 定义了生产者的类型， 两类



在这种场景下，ProducerType 的类型的 MULTI

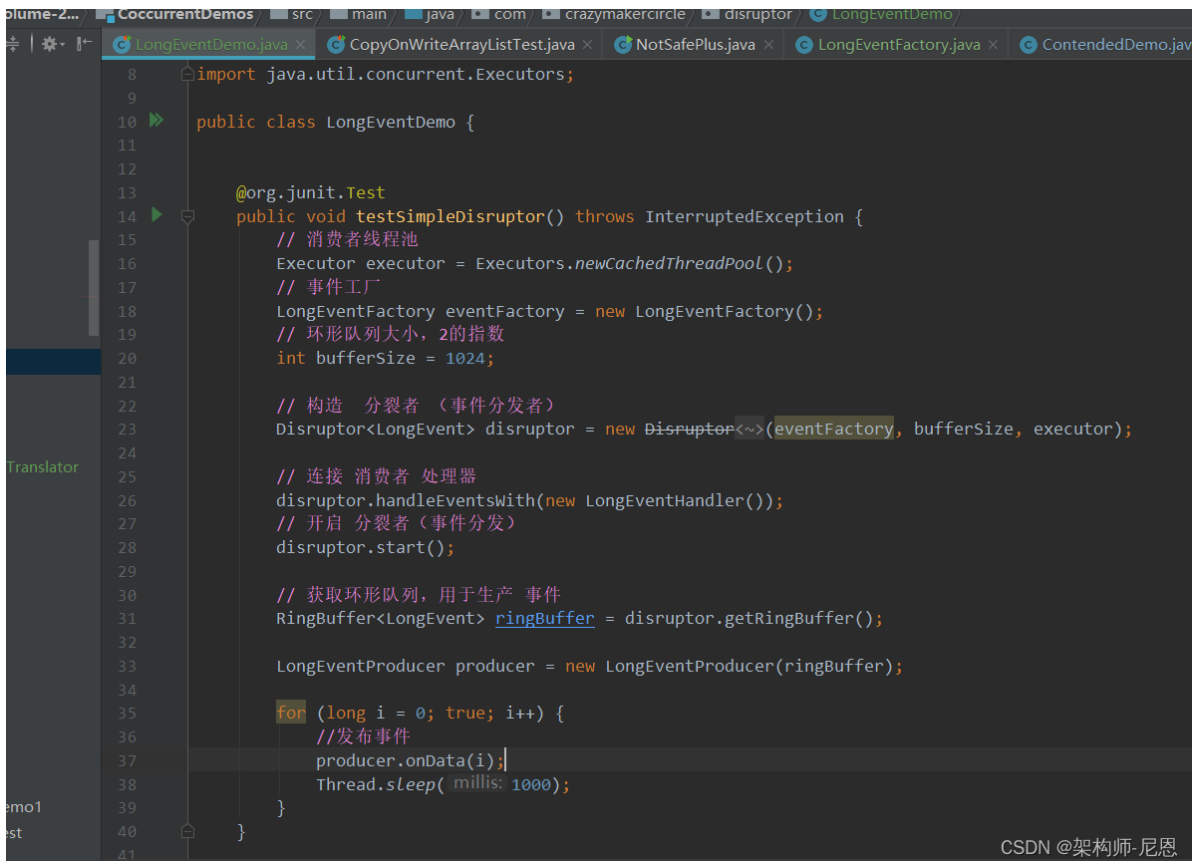
多生产者场景的要点

在代码编写维度，多生产者单消费者场景的要点如下：

- 创建 Disruptor 的时候，将 ProducerType.SINGLE 改为 ProducerType.MULTI，
- 编写多线程生产者的相关代码即可。

多生产者场景的参考代码

参考的代码如下：



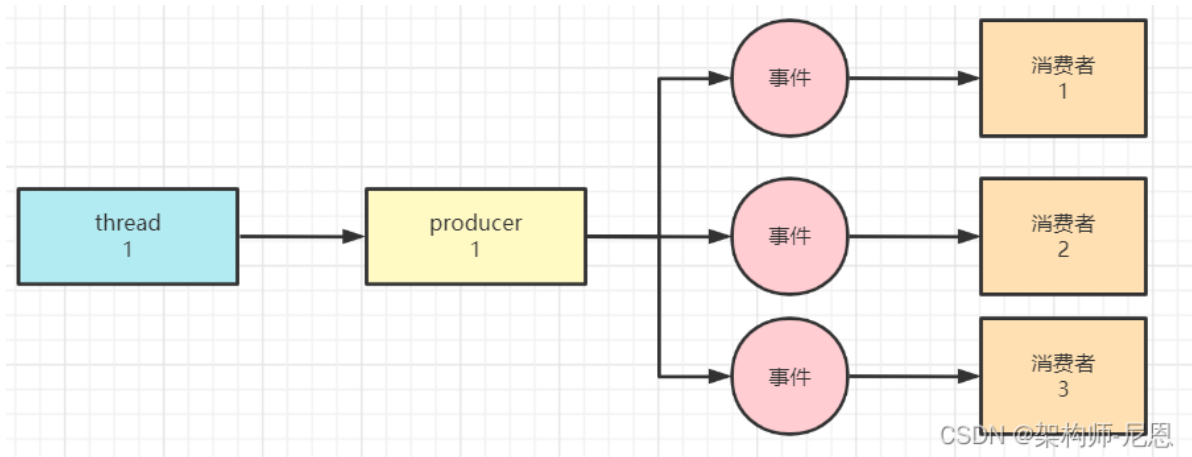
运行的结果如下

```
LongEventSceneDemo.testSimpleDisruptorWithMethodRef
va-high-concurrency-core-Programming-Volume-2... CoccurentDemos src main java com crazymakercircle disruptor
project main java com.crazymakercircle
LongEventDemo.java CopyOnWriteArrayListTest.java NotSafePlus.java
35 for (long i = 0; true; i++) {
36     //发布事件
LongEventDemo testSimpleDisruptor()
g: LongEventSceneDemo.testSimpleDisruptorWithMethodRef
Debugger Console
Tests passed: 1 of 1 test - 5 s 230 ms
D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:51918', transport: 'socket'
0
0
0
1
1
1
2
2
2
3
3
3
4
4
4
4
Disconnected from the target VM, address: '127.0.0.1:51918', transport: 'socket'
Process finished with exit code 0
CSDN @架构师-尼恩
```

以上用例的具体减少，请参见 尼恩《100wqps 日志平台实操，视频》

单生产者多消费者竞争场景

该场景中，生产者为一个，消费者为多个，多个消费者之间，存在着竞争关系，也就是说，对于同一个事件 event，多个消费者不重复消费



disruptor 如何设置多个竞争消费者？

首先，得了解一下，disruptor 框架的两个设置消费者的方法

大概有两点：

- 消费者需要实现 WorkHandler 接口，而不是 EventHandler 接口
- 使用 handleEventsWithWorkerPool 设置 disruptor 的消费者，而不是 handleEventsWith 方法

在 disruptor 框架调用 start 方法之前，有两个方法设置消费者：

- disruptor.handleEventsWith(EventHandler ... handlers)，将多个 EventHandler 的实现类传入方法，封装成一个 EventHandlerGroup，实现多消费者消费。
- disruptor.handleEventsWithWorkerPool(WorkHandler ... handlers)，将多个 WorkHandler 的实现类传入方法，封装成一个 EventHandlerGroup 实现多消费者消费。

那么，以上的 Disruptor 类的 handleEventsWith，handleEventsWithWorkerPool 方法的联系及区别是什么呢？

相同的在于：

两者共同点都是，将多个消费者封装到一起，供框架消费事件。

第一个不同点在于：

对于某一条事件 event，

handleEventsWith 方法返回的 EventHandlerGroup，Group 中的每个消费者都会对 event 进行消费，各个消费者之间不存在竞争。

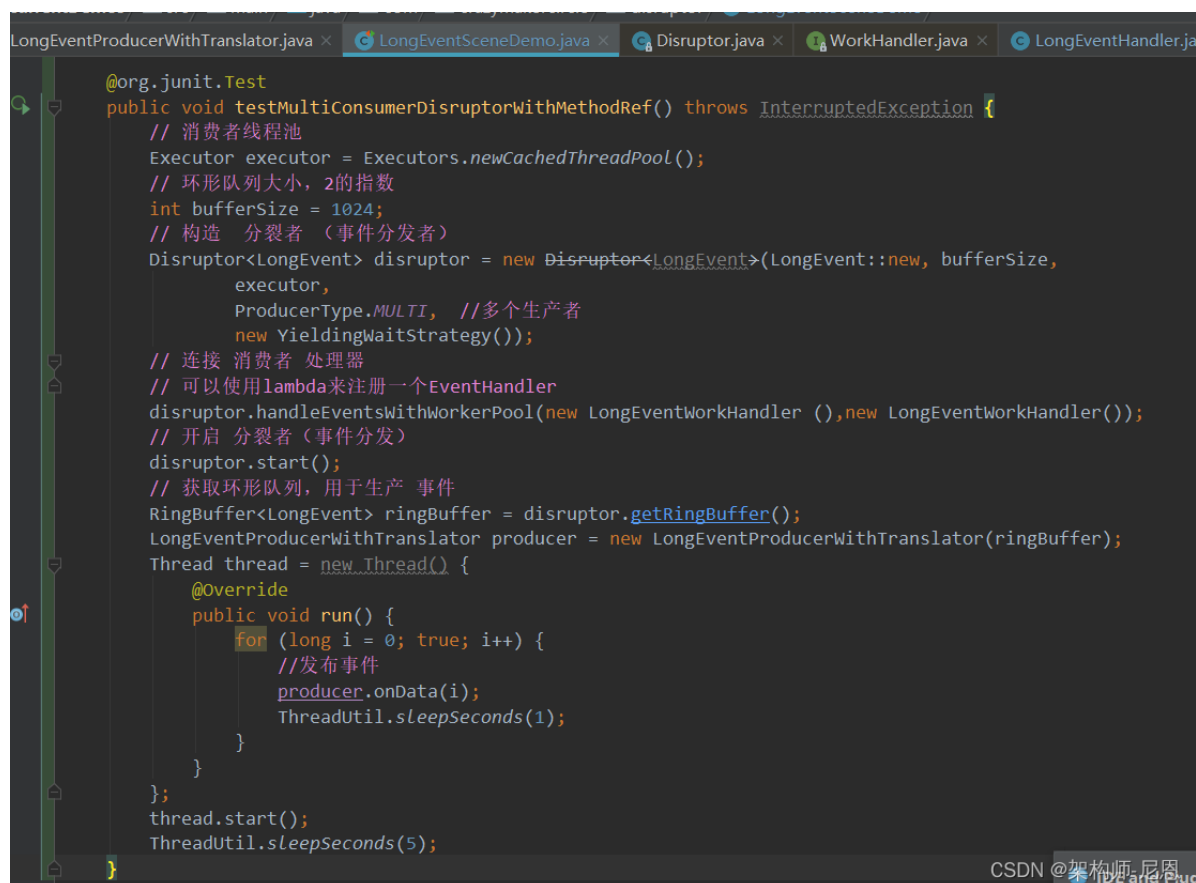
handleEventsWithWorkerPool 方法返回的 EventHandlerGroup，Group 的消费者对于同一条事件 event 不重复消费；也就是，如果 c0 消费了事件 m，则 c1 不再消费事件 m。

另外一个不同：

在设置消费者的时候，Disruptor 类的 handleEventsWith，handleEventsWithWorkerPool 方法所传入的形参不同。对于独立消费的消费，应当实现 EventHandler 接口。对于不重复消费的消费，应当实现 WorkHandler 接口。

因此，根据消费者集合是否独立消费事件，可以对不同的接口进行实现。也可以对两种接口同时实现，具体消费流程由 disruptor 的方法调用决定。

演示代码如下：

The image is a screenshot of an IDE window showing a Java file named LongEventSceneDemo.java. The code is a JUnit test that demonstrates the use of a Disruptor with multiple consumers. It includes comments in Chinese explaining the steps: creating an executor, setting buffer size, constructing the Disruptor with a multi-producer type and yielding wait strategy, connecting consumers using the WorkerPool method, starting the Disruptor, and then starting a producer thread that publishes events to the ring buffer. The code is as follows:

```
@org.junit.Test
public void testMultiConsumerDisruptorWithMethodRef() throws InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小，2的指数
    int bufferSize = 1024;
    // 构造 分裂者（事件分发者）
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
        executor,
        ProducerType.MULTI, //多个生产者
        new YieldingWaitStrategy());
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler
    disruptor.handleEventsWithWorkerPool(new LongEventWorkHandler(), new LongEventWorkHandler());
    // 开启 分裂者（事件分发）
    disruptor.start();
    // 获取环形队列，用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
    LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override
        public void run() {
            for (long i = 0; true; i++) {
                //发布事件
                producer.onData(i);
                ThreadUtil.sleepSeconds(1);
            }
        }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}
```

执行结果

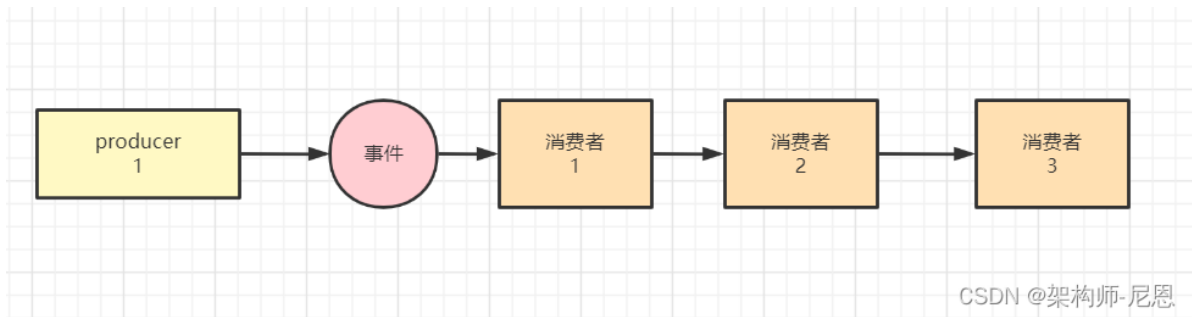
```
g: LongEventSceneDemo.testMultiConsumerDisruptorWithMethodRef x
Debugger Console →
Tests passed: 1 of 1 test – 5 s 65 ms
D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:56708', transport: 'socket'
0
1
2
3
4
Disconnected from the target VM, address: '127.0.0.1:56708', transport: 'socket'
Process finished with exit code 0
CSDN @架构师-尼恩
```

以上用例的具体减少，请参见 尼恩 《100wqps 日志平台实操，视频》

多个消费者串行消费场景

在 多个消费者串行消费场景中，多个消费者，可以按照次序，消费消息。

比如：一个用户注册的 Event，需要有一个 Handler 来存储信息，一个 Hanlder 来发邮件等等。



多个消费者串行消费场景案例

```
@org.junit.Test
public void testMultiSerialConsumerDisruptorWithMethodRef() throws InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小，2的指数
    int bufferSize = 1024;
    // 构造 分裂者（事件分发者）
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
        executor,
        ProducerType.SINGLE, //多个生产者
        new YieldingWaitStrategy());
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler
    disruptor.handleEventsWith(LongEventSceneDemo::handleEvent)
        .then(LongEventSceneDemo::handleEvent2)
        .then(LongEventSceneDemo::handleEvent3);
    // 开启 分裂者（事件分发）
    disruptor.start();
    // 获取环形队列，用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
    //1生产者，并发生产数据
    LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override public void run() {
            for (long i = 0; true; i++) {
                /*发布事件*/producer.onData(i);
                ThreadUtil.sleepSeconds(1);
            }
        }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}
```

CSDN @架构师-尼恩

执行结果

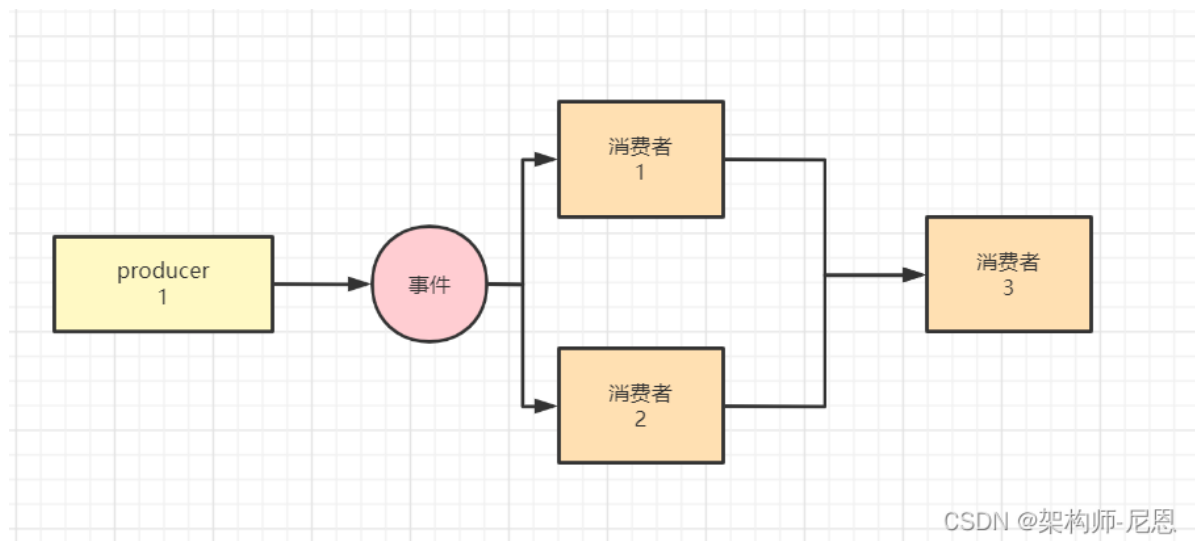
```
LongEventSceneDemo.testWithSceneConsumerDisruptorWithMethodOverload ...
bugger Console
Tests passed: 1 of 1 test - 5 s 66 ms
Lo: 5 s 66 ms D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
5 s 66 ms Connected to the target VM, address: '127.0.0.1:57169', transport: 'socket'
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 0
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 0
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 0
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 1
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 1
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 1
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 2
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 2
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 2
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 3
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 3
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 3
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 4
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 4
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 4
Disconnected from the target VM, address: '127.0.0.1:57169', transport: 'socket'
Process finished with exit code 0
```

CSDN @架构师-尼恩

菱形方式执行场景

场景特点

先并发，后串行



CSDN @架构师-尼恩

菱形方式执行场景案例


```

9
10
11 @org.junit.Test
12 public void testCurrentThenSerialConsumerDisruptorWithMethodRef() throws InterruptedException {
13     // 消费者线程池
14     Executor executor = Executors.newCachedThreadPool();
15     // 环形队列大小, 2的指数
16     int bufferSize = 1024;
17     // 构造 分裂者 (事件分发者)
18     Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
19         executor,
20         ProducerType.SINGLE, //多个生产者
21         new YieldingWaitStrategy());
22     // 连接 消费者 处理器
23     // 可以使用lambda来注册一个EventHandler
24     disruptor.handleEventsWith(LongEventSceneDemo::handleEvent, LongEventSceneDemo::handleEvent2)
25         .then(LongEventSceneDemo::handleEvent3);
26     // 开启 分裂者 (事件分发)
27     disruptor.start();
28     // 获取环形队列, 用于生产 事件
29     RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
30     // 1生产者, 并发生产数据
31     LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
32     Thread thread = new Thread() { @Override public void run() { for (long i = 0; true; i++)
33         { /*发布事件*/ producer.onData(i); ThreadUtil.sleepSeconds(1); } }
34     };
35     thread.start();
36     ThreadUtil.sleepSeconds(5);
37 }
38
39 }

```

CSDN @架构师-尼恩

执行结果

```

>> Tests passed: 1 of 1 test - 5 s 64 ms
D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe -agent
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 0
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 0
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 0
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 1
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 1
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 1
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 2
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 2
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 2
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 3
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 3
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 3
[pool-1-thread-2|LongEventSceneDemo.handleEvent2]: 4
[pool-1-thread-1|LongEventSceneDemo.handleEvent]: 4
[pool-1-thread-3|LongEventSceneDemo.handleEvent3]: 4
Disconnected from the target VM, address: '127.0.0.1:49515', transport: CSDN@架构师-尼恩

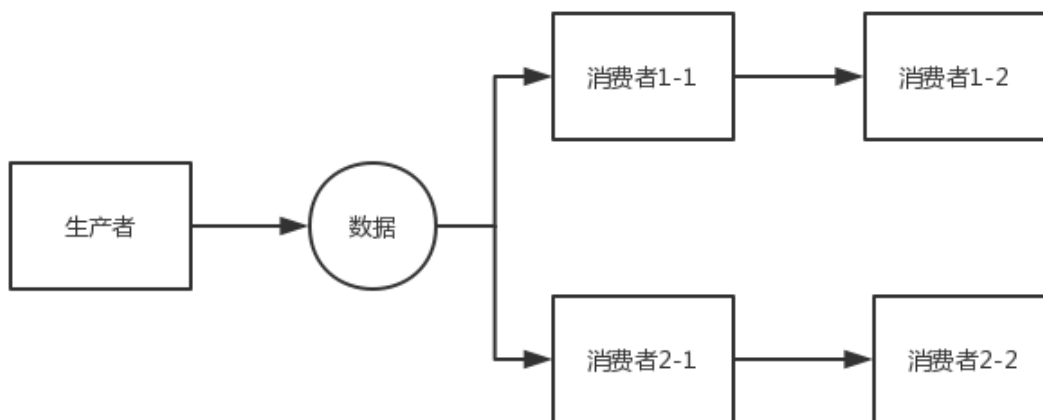
```

链式并行执行场景

场景特点

多组消费者形成 并行链, 特点是:

- 链内 串行
- 链间 并行



场景案例

```
occurentDemos | src | main | java | com | crazymakercircle | disruptor | LongEventSceneDemo
LongEventProducerWithTranslator.java | LongEventSceneDemo.java | EventHandlerGroup.java | Disruptor.java | WorkHandler.java
@org.junit.Test
public void testLinkSerialConsumerDisruptorWithMethodRef() throws InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小, 2的指数
    int bufferSize = 1024;
    // 构造 分裂者 (事件分发者)
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
        executor,
        ProducerType.SINGLE, //多个生产者
        new YieldingWaitStrategy());
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler
    disruptor.handleEventsWith(LongEventSceneDemo::handleEvent1)
        .then(LongEventSceneDemo::handleEvent2);
    disruptor.handleEventsWith(LongEventSceneDemo::handleEvent3)
        .then(LongEventSceneDemo::handleEvent4);
    // 开启 分裂者 (事件分发)
    disruptor.start();
    // 获取环形队列, 用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
    // 1生产者, 并发生产数据
    LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override
        public void run() {
            for (long i = 0; true; i++) { producer.onData(i); ThreadUtil.sleepSeconds(1); }
        }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}
```

CSDN @架构师-尼恩

执行结果

```
Debugger Console
>> Tests passed: 1 of 1 test - 5 s 62 ms

D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:49609', transport: 'socket'
[pool-1-thread-1]LongEventSceneDemo.handleEvent1]: 0
[pool-1-thread-3]LongEventSceneDemo.handleEvent3]: 0
[pool-1-thread-2]LongEventSceneDemo.handleEvent2]: 0
[pool-1-thread-4]LongEventSceneDemo.handleEvent4]: 0
[pool-1-thread-3]LongEventSceneDemo.handleEvent3]: 1
[pool-1-thread-1]LongEventSceneDemo.handleEvent1]: 1
[pool-1-thread-4]LongEventSceneDemo.handleEvent4]: 1
[pool-1-thread-2]LongEventSceneDemo.handleEvent2]: 1
[pool-1-thread-3]LongEventSceneDemo.handleEvent3]: 2
[pool-1-thread-1]LongEventSceneDemo.handleEvent1]: 2
[pool-1-thread-4]LongEventSceneDemo.handleEvent4]: 2
[pool-1-thread-2]LongEventSceneDemo.handleEvent2]: 2
[pool-1-thread-1]LongEventSceneDemo.handleEvent1]: 3
[pool-1-thread-3]LongEventSceneDemo.handleEvent3]: 3
[pool-1-thread-2]LongEventSceneDemo.handleEvent2]: 3
[pool-1-thread-4]LongEventSceneDemo.handleEvent4]: 3
[pool-1-thread-1]LongEventSceneDemo.handleEvent1]: 4
[pool-1-thread-3]LongEventSceneDemo.handleEvent3]: 4
[pool-1-thread-2]LongEventSceneDemo.handleEvent2]: 4
[pool-1-thread-4]LongEventSceneDemo.handleEvent4]: 4
Disconnected from the target VM, address: '127.0.0.1:49609', transport: 'socket'
```

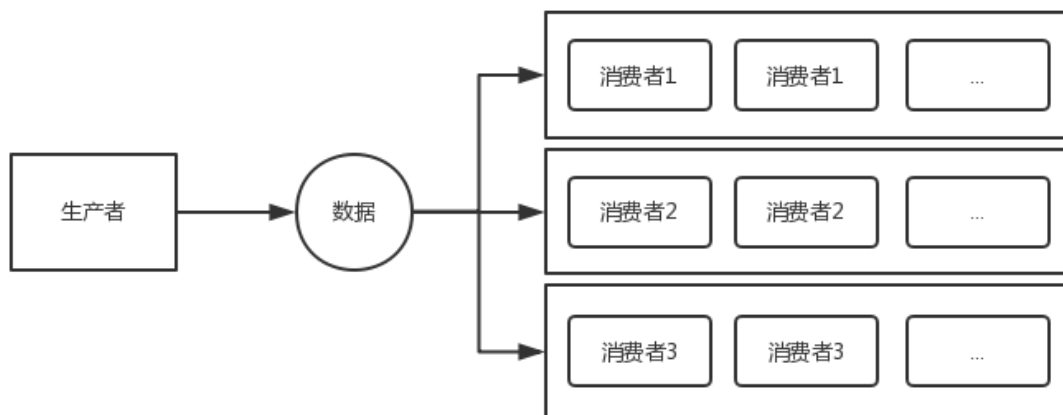
CSDN @架构师-尼恩

多组消费者相互隔离场景

场景特点

多组消费者 相互隔离，特点是：

- 组内 相互竞争
- 组间 相互隔离



场景案例

```

@org.junit.Test
public void testIsolatedDisruptorWithMethodRef() throws InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小, 2的指数
    int bufferSize = 1024;
    // 构造 分裂者 (事件分发者)
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
        executor,
        ProducerType.SINGLE, //多个生产者
        new YieldingWaitStrategy());
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler
    disruptor.handleEventsWithWorkerPool(new LongEventWorkHandler(), new LongEventWorkHandler());
    disruptor.handleEventsWithWorkerPool(new LongEventWorkHandler2(), new LongEventWorkHandler2());
    // 开启 分裂者 (事件分发)
    disruptor.start();
    // 获取环形队列, 用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
    // 1生产者, 并发生产数据
    LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override
        public void run() { for (long i = 0; true; i++) { producer.onData(i); ThreadUtil.sleepSeconds(1); } }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}

```

CSDN @架构师-尼恩

执行结果

```

D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:51217', transport: 'socket'
[pool-1-thread-3|LongEventWorkHandler2.onEvent]: 0
[pool-1-thread-1|LongEventWorkHandler.onEvent]: 0
[pool-1-thread-4|LongEventWorkHandler2.onEvent]: 1
[pool-1-thread-2|LongEventWorkHandler.onEvent]: 1
[pool-1-thread-1|LongEventWorkHandler.onEvent]: 2
[pool-1-thread-3|LongEventWorkHandler2.onEvent]: 2
[pool-1-thread-4|LongEventWorkHandler2.onEvent]: 3
[pool-1-thread-2|LongEventWorkHandler.onEvent]: 3
[pool-1-thread-3|LongEventWorkHandler2.onEvent]: 4
[pool-1-thread-1|LongEventWorkHandler.onEvent]: 4
Disconnected from the target VM, address: '127.0.0.1:51217', transport: 'socket'

Process finished with exit code 0

```

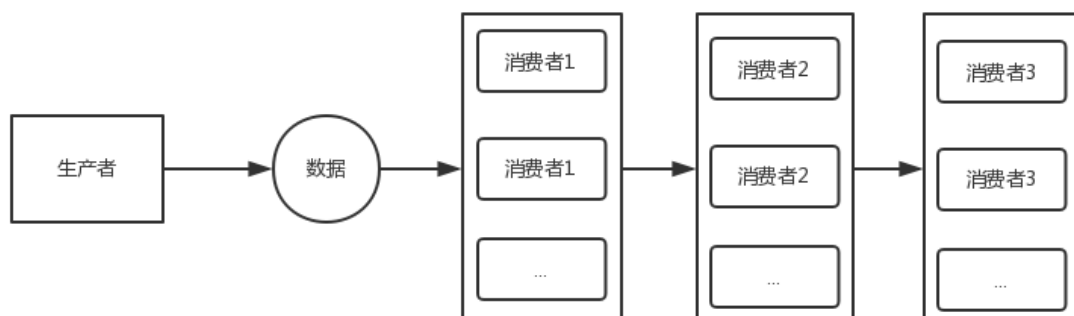
CSDN @架构师-尼恩

多组消费者航道执行模式

场景特点

多组消费者形成 并行链, 特点是:

- 组内 相互竞争
- 组之间串行依次执行



场景案例

组之间串行依次执行, 组内有多个实例竞争执行

```
EventSceneDemo.java x EventHandlerGroup.java x Disruptor.java x WorkHandler.java x LongEventHandler.java x
public void testChannelModelDisruptorWithMethodRef() throws InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小, 2的指数
    int bufferSize = 1024;
    // 构造 分裂者 (事件分发者)
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new, bufferSize,
        executor,
        ProducerType.SINGLE, //多个生产者
        new YieldingWaitStrategy());
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler
    disruptor.handleEventsWithWorkerPool(new LongEventWorkHandler(), new LongEventWorkHandler())
        .thenHandleEventsWithWorkerPool(new LongEventWorkHandler2(), new LongEventWorkHandler2());
    // 开启 分裂者 (事件分发)
    disruptor.start();
    // 获取环形队列, 用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
    // 1生产者, 并发生产数据
    LongEventProducerWithTranslator producer = new LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override
        public void run() { for (long i = 0; true; i++) { producer.onData(i); ThreadUtil.sleepSeconds(1); } }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}
```

CSDN @架构师-尼恩

执行效果

```
ger Console →
Tests passed: 1 of 1 test - 5 s 68 ms
D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:51360', transport: 'socket'
[pool-1-thread-1|LongEventWorkHandler.onEvent]: 0
[pool-1-thread-3|LongEventWorkHandler2.onEvent]: 0
[pool-1-thread-2|LongEventWorkHandler.onEvent]: 1
[pool-1-thread-4|LongEventWorkHandler2.onEvent]: 1
[pool-1-thread-1|LongEventWorkHandler.onEvent]: 2
[pool-1-thread-3|LongEventWorkHandler2.onEvent]: 2
[pool-1-thread-2|LongEventWorkHandler.onEvent]: 3
[pool-1-thread-4|LongEventWorkHandler2.onEvent]: 3
[pool-1-thread-1|LongEventWorkHandler.onEvent]: 4
[pool-1-thread-3|LongEventWorkHandler2.onEvent]: 4
Disconnected from the target VM, address: '127.0.0.1:51360', transport: 'socket'

Process finished with exit code 0

CSDN @架构师-尼恩
```

说明: 本文会以 pdf 格式持续更新, 更多最新尼恩 3 高 pdf 笔记, 请从下面的链接获取: [语雀](#) 或者 [码云](#)

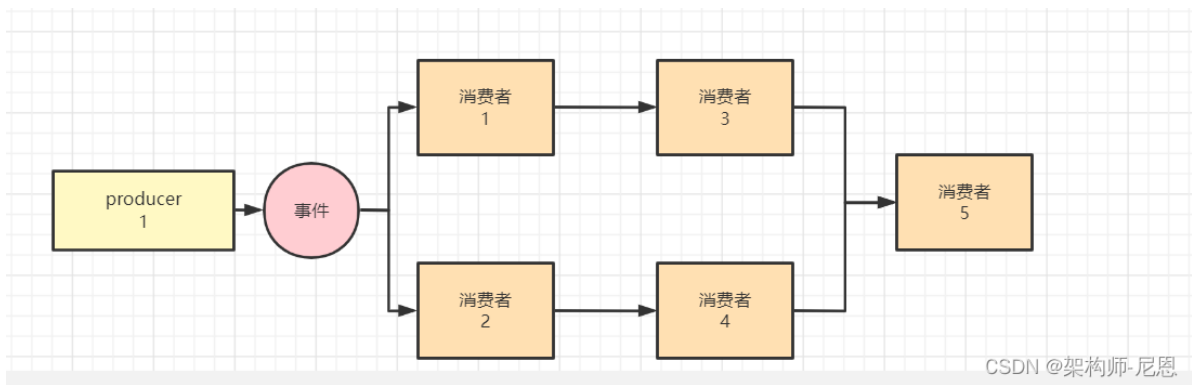
六边形执行顺序

这是一种比较复杂的场景

场景特点

单边内部是有序的

边和边之间是并行的



参考代码

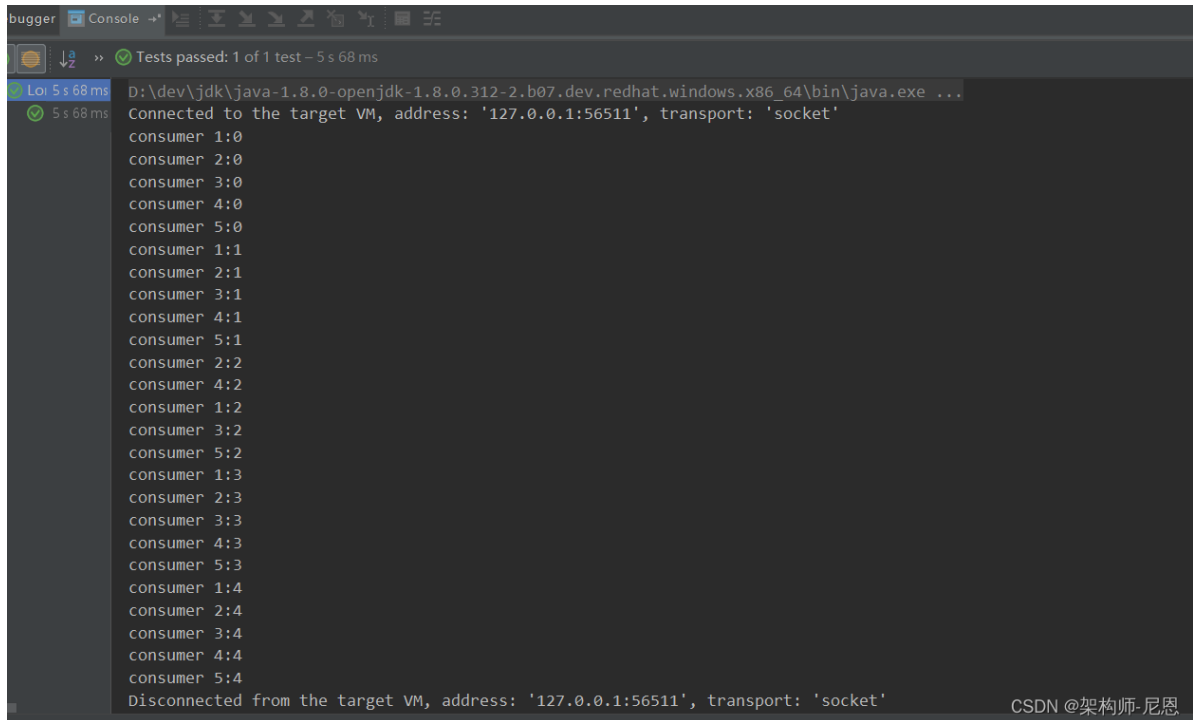
```
@org.junit.Test
public void testHexagonConsumerDisruptorWithMethodRef() throws
InterruptedException {
    // 消费者线程池
    Executor executor = Executors.newCachedThreadPool();
    // 环形队列大小, 2的指数
    int bufferSize = 1024;
    // 构造 分裂者 (事件分发者)
    Disruptor<LongEvent> disruptor = new Disruptor<LongEvent>(LongEvent::new,
bufferSize,
        executor,
        ProducerType.SINGLE, //多个生产者
        new YieldingWaitStrategy());

    EventHandler consumer1 = new LongEventHandlerWithName("consumer 1");
    EventHandler consumer2 = new LongEventHandlerWithName("consumer 2");
    EventHandler consumer3 = new LongEventHandlerWithName("consumer 3");
    EventHandler consumer4 = new LongEventHandlerWithName("consumer 4");
    EventHandler consumer5 = new LongEventHandlerWithName("consumer 5");
    // 连接 消费者 处理器
    // 可以使用lambda来注册一个EventHandler

    disruptor.handleEventsWith(consumer1, consumer2);
    disruptor.after(consumer1).handleEventsWith(consumer3);
    disruptor.after(consumer2).handleEventsWith(consumer4);
    disruptor.after(consumer3, consumer4).handleEventsWith(consumer5);
    // 开启 分裂者 (事件分发)
    disruptor.start();
    // 获取环形队列, 用于生产 事件
    RingBuffer<LongEvent> ringBuffer = disruptor.getRingBuffer();
    //1生产者, 并发生产数据
    LongEventProducerWithTranslator producer = new
LongEventProducerWithTranslator(ringBuffer);
    Thread thread = new Thread() {
        @Override
        public void run() {
            for (long i = 0; true; i++) {
                producer.onData(i);
                ThreadUtil.sleepSeconds(1);
            }
        }
    };
    thread.start();
    ThreadUtil.sleepSeconds(5);
}
```

```
}
```

执行结果



The screenshot shows a debugger window with a console pane. The console displays the output of a test execution. The test passed, and the output shows a sequence of messages from a target VM, including connection status, consumer IDs, and disconnection status. The console also shows the time taken for the test (5 s 68 ms) and the time taken for the test to pass (5 s 68 ms).

```
bugger Console +*
>> Tests passed: 1 of 1 test - 5 s 68 ms
Lo: 5 s 68 ms D:\dev\jdk\java-1.8.0-openjdk-1.8.0.312-2.b07.dev.redhat.windows.x86_64\bin\java.exe ...
5 s 68 ms Connected to the target VM, address: '127.0.0.1:56511', transport: 'socket'
consumer 1:0
consumer 2:0
consumer 3:0
consumer 4:0
consumer 5:0
consumer 1:1
consumer 2:1
consumer 3:1
consumer 4:1
consumer 5:1
consumer 2:2
consumer 4:2
consumer 1:2
consumer 3:2
consumer 5:2
consumer 1:3
consumer 2:3
consumer 3:3
consumer 4:3
consumer 5:3
consumer 1:4
consumer 2:4
consumer 3:4
consumer 4:4
consumer 5:4
Disconnected from the target VM, address: '127.0.0.1:56511', transport: 'socket'
CSDN @架构师-尼恩
```