链滴

Spring 整合 Disruptor3

作者：jianzh5
原文链接：https://ld246.com/article/1579135204272
来源网站：链滴

什么是Disruptor

从功能上来看，Disruptor 是实现了“队列”的功能，而且是一个有界队列。那么它的应用场景自然是“生产者-消费者”模型的应用场合了。

可以拿 JDK 的 BlockingQueue 做一个简单对比，以便更好地认识 Disruptor 是什么。

我们知道 BlockingQueue 是一个 FIFO 队列，生产者(Producer)往队列里发布(publish)一项事件(或之为“消息”也可以)时，消费者(Consumer)能获得通知；如果没有事件时，消费者被堵塞，直到生者发布了新的事件。

这些都是 Disruptor 能做到的，与之不同的是，Disruptor 能做更多：

- 同一个“事件”可以有多个消费者，消费者之间既可以并行处理，也可以相互依赖形成处理的先后序(形成一个依赖图)；
- 预分配用于存储事件内容的内存空间；
- 针对极高的性能目标而实现的极度优化和无锁的设计；

以上虽然简单地描述了 Disruptor 是什么，但对于它"能做什么"，还不是那么明白。简而言之，当你要在两个独立的处理过程之间交换数据时，就可以使用 Disruptor 。当然使用队列也可以，只不过 Disuptor 的性能更好。

实战

本文先不具体去阐述Disruptor的工作具体原理，只是简单地将Spring与其整合。整合过程很简单，体步骤如下：

1、在pom文件中引入disruptor

```
<dependency>
   <groupId>com.lmax</groupId>
   <artifactId>disruptor</artifactId>
   <version>3.4.2</version>
</dependency>
```

2、创建事件

```
@Data
public class NotifyEvent {
   private String message;
}
```

3、创建消息工厂用于生产消息

```
public class NotifyEventFactory implements EventFactory {
   @Override
   public Object newInstance() {
      return new NotifyEvent();
   }
}
```

4、创建消费者，此处用于处理业务逻辑

```
public class NotifyEventHandler implements EventHandler<NotifyEvent>,WorkHandler<Notif
```

```
Event> {

    @Override public void onEvent(NotifyEvent notifyEvent, long l, boolean b) throws Exceptio
 {
        System.out.println("接收到消息"); this.onEvent(notifyEvent);

    }

    @Override public void onEvent(NotifyEvent notifyEvent) throws Exception {
        System.out.println(notifyEvent+">>>"+UUID.randomUUID().toString());
    }
}
```

5、自定义异常

```
@Log4j2
public class NotifyEventHandlerException implements ExceptionHandler {
    @Override
    public void handleEventException(Throwable throwable, long sequence, Object event) {
        throwable.fillInStackTrace();
        log.error("process data error sequence ==[{}] event==[{}] ,ex ==[{}]", sequence, event.toS
ring(), throwable.getMessage());
    }

    @Override
    public void handleOnStartException(Throwable throwable) {
        log.error("start disruptor error ==[{}]!", throwable.getMessage());
    }

    @Override
    public void handleOnShutdownException(Throwable throwable) {
        log.error("shutdown disruptor error ==[{}]!", throwable.getMessage());
    }
}
```

6、整合Spring，对Disruptor进行初始化

```
@Service
public class NotifyServiceImpl implements INotifyService, DisposableBean,InitializingBean {
    private Disruptor<NotifyEvent> disruptor;
    private static final int RING_BUFFER_SIZE = 1024 * 1024;

    @Override
    public void destroy() throws Exception {
        disruptor.shutdown();
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        disruptor = new Disruptor<NotifyEvent>(new NotifyEventFactory(),RING_BUFFER_SIZE, E
ecutors.defaultThreadFactory(), ProducerType.SINGLE,new BlockingWaitStrategy());
        disruptor.setDefaultExceptionHandler(new NotifyEventHandlerException());
        disruptor.handleEventsWith(new NotifyEventHandler());
        disruptor.start();
```

```
    }


    @Override
    public void sendNotify(String message) {
        RingBuffer<NotifyEvent> ringBuffer = disruptor.getRingBuffer();
//        ringBuffer.publishEvent(new EventTranslatorOneArg<NotifyEvent, String>() {
//            @Override
//            public void translateTo(NotifyEvent event, long sequence, String data) {
//                event.setMessage(data);
//            }
//        }, message);
        ringBuffer.publishEvent((event, sequence, data) -> event.setMessage(data), message); //lmbda式写法，如果是用jdk1.8以下版本使用以上注释的一段

    }
}
```

7、消息生产接口

```
public interface INotifyService { /** * 发送消息
     * @author jianzhang11
     * @date 2018/4/13 16:52
     * @param message */
    void sendNotify(String message);

}
```

8、在需要调用的地方注入INotifyService并调用sendNotify方法

```
@GetMapping("test")
@ResponseBody public String testLog() {
    log.info("==============");
    notifyService.sendNotify("Hello,World!"); return "hello,world";
}
```