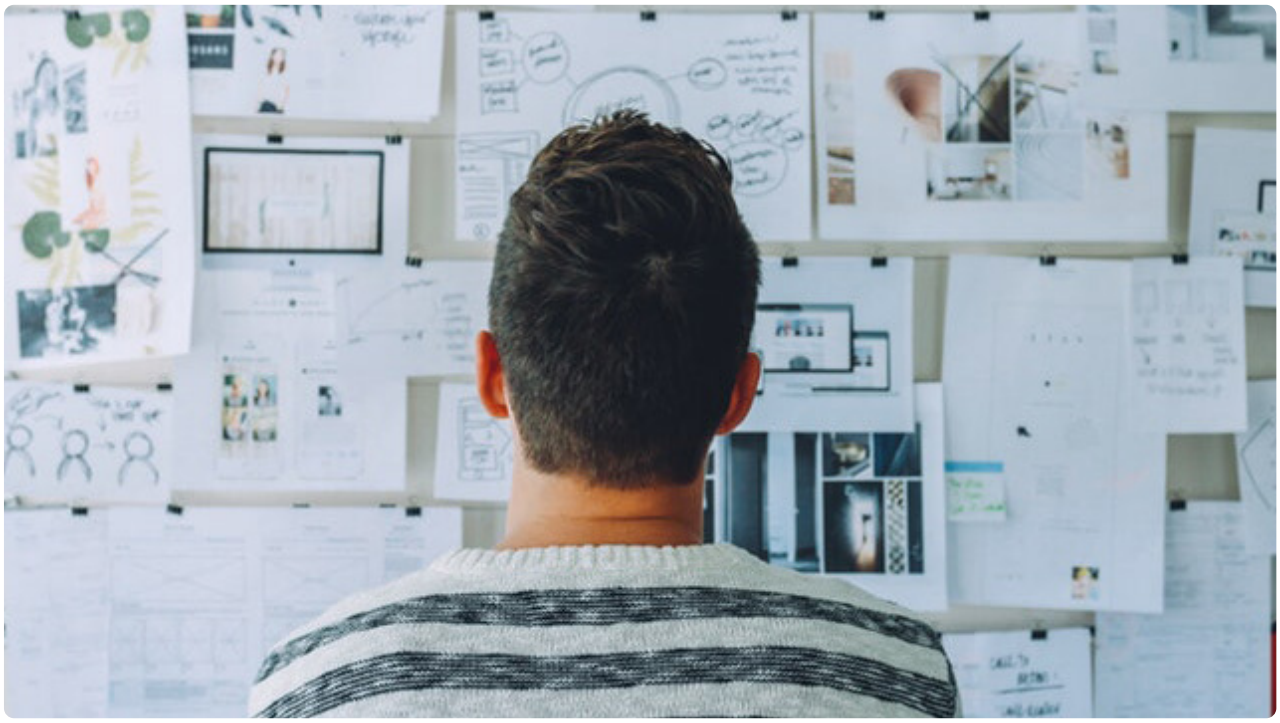
29 一手交钱，一手交货—Exchanger详解

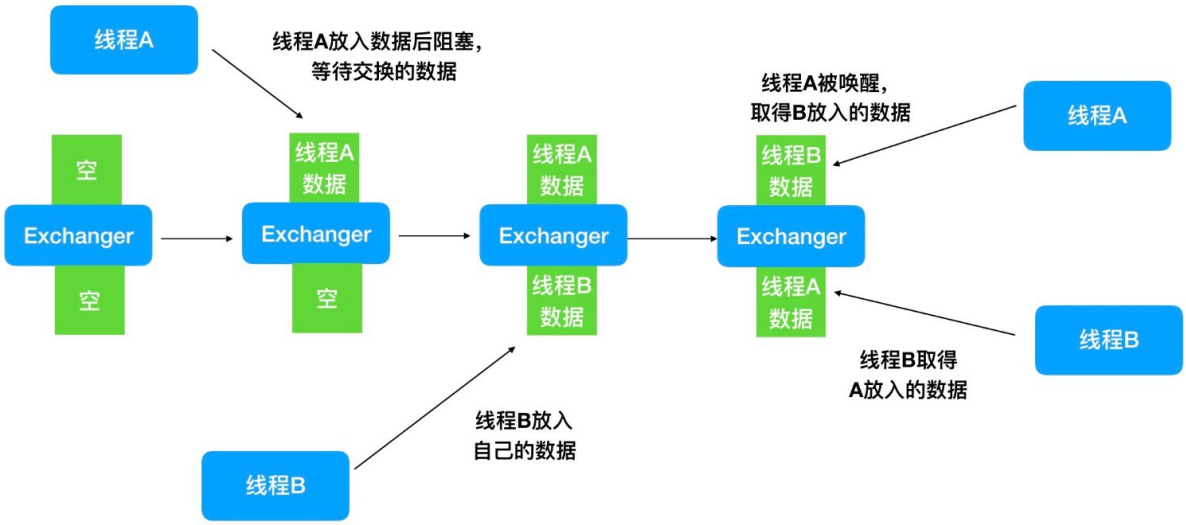
更新时间：2019-12-05 09:48:58



> 世上无难事,只要肯登攀。
>
> ——毛泽东

Exchanger 从字面来看是一个交换器，它用来在线程间交换数据。如果两个线程并行处理，但在某个时刻需要互相交换自己已经处理完的中间数据，然后才能继续往下执行。这个时候就可以使用 Exchanger。

两个线程不用谋面，但是通过 Exchanger 就能实现数据交换，这是如何做到的呢？我们可以认为 Exchanger 有两个槽位，线程可以向其中放入自己想要交换的数据，然后阻塞在此，等待其它线程填充另外的槽位。等两个槽位都填充数据后，Exchanger 会把槽位调换位置，让线程取到另外线程放入的数据。如下图所示：



前面的描述只是为了帮助理解，其实Exchanger 真实的设计更为复杂。下面我们先通过一个简单的例子来看看如何使用 Exchanger。

1、Exchanger 的使用

我们设想如下场景：如果今天上午你需要给你女朋友快递一部手机，但是突然你要去一个紧急的会议。于是你把快递的东西留给你的同事。当快递取件员上门时，他会把快递底单给你同事，你同事则会把快递给取件员。然后你的同事再把快递底单给你。

这个场景里，你和快递员就是两个不同的线程。而你的同事是 Exchanger，负责你和快递员间进行物品交换。我们看看代码应该如何编写：

```
public class ExchangerClient {

    public static void main(String[] args) throws InterruptedException {
        Exchanger<String> workmate = new Exchanger<>();

        Thread michael = new Thread(() -> {
            String threadName = Thread.currentThread().getName();
            try {
                System.out.println(threadName+": I'm Michael and want to delivery a phone to my friend");
                System.out.println(threadName+": There is an emergency meeting");
                System.out.println(threadName+": I give the phone to my workmate.");
                System.out.println(threadName+": He will help me to delivery the phone and return the express document to me");
                String expressDocument = workmate.exchange("---phone---");
                System.out.println(threadName+": I got the express document"+ expressDocument);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        },"Michael");

        Thread deliveryman = new Thread(() -> {
            String threadName = Thread.currentThread().getName();
            try {
                System.out.println(threadName+": I'm a deliveryman");
                System.out.println(threadName+": I'm going to get Michael's express from his workmate and give the express document to his workmate");
                String expressGoods = workmate.exchange("---phone express document----");
                System.out.println(threadName+": I got the goods of "+expressGoods);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        },"Deliveryman");

        michael.start();
        TimeUnit.MILLISECONDS.sleep(100);
        deliveryman.start();
    }
}
```

Michael 通过 Exchanger 的实例 workmate，把 phone 交给了快递员。而快递员也通过 workmate 把快递单据给了 Michael。我们看一下输出：

```
Michael: I'm Michael and want to delivery a phone to my friend
Michael: There is an emergency meeting
Michael: I give the phone to my workmate.
Michael: He will help me to delivery and return the express document to me
Deliveryman: I'm a deliveryman
Deliveryman: I'm going to get Michael's express from his workmate and give the express document to his workmate
Deliveryman: I got the goods of ---phone---
Michael: I got the express document---phone express document----
```

可以看出 Michael 在调用 workmate.exchange("—phone—") 后被阻塞。而当 deliveryman 调用 workmate.exchange("—phone express document——") 后，两个线程才继续往下走，并且进行了数据交换。Michael 发出了货物 “—phone—”，得到了 “—phone express document----”。而 deliveryman 则得到了 “—phone—”，发出了 “—phone express document——”。

两个线程通过 Exchanger 实现了数据的交换。

2、多于两个线程使用 Exchanger

Exchanger 支持两个线程数据互换，那么你有没有想过多于两个线程使用 Exchanger 做数据交换会怎么样呢？我们把上买呢例子改一下，假如另外一个同事 Green 也想通过 workmate 协助发送快递，那么会怎么样？

```
public static void main(String[] args) throws InterruptedException {
    Exchanger<String> workmate = new Exchanger<>();

    Thread michael = new Thread(() -> {
        String threadName = Thread.currentThread().getName();
        try {
            System.out.println(threadName+": I'm Michael and want to delivery a phone to my friend");
            System.out.println(threadName+": There is an emergency meeting");
            System.out.println(threadName+": I give the phone to my workmate.");
            System.out.println(threadName+": He will help me to delivery the phone and return the express document to me");
            String expressDocument = workmate.exchange("---phone---");
            System.out.println(threadName+": I got the express document"+ expressDocument);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    },"Michael");

    Thread green = new Thread(() -> {
        String threadName = Thread.currentThread().getName();
        try {
            System.out.println(threadName+": I'm green and want to delivery a macbook to my friend");
            System.out.println(threadName+": There is an emergency meeting");
            System.out.println(threadName+": I give the macbook to my workmate.");
            System.out.println(threadName+": He will help me to delivery the macbook and return the express document to me");
            String expressDocument = workmate.exchange("---macbook---");
            System.out.println(threadName+": I got the express document"+ expressDocument);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    },"Green");
    Thread deliveryman = new Thread(() -> {
        String threadName = Thread.currentThread().getName();
        try {
            System.out.println(threadName+": I'm a deliveryman");
            System.out.println(threadName+": I'm going to get Michael's express from his workmate and give the express document to his workmate");
            String expressGoods = workmate.exchange("---phone express document----");
            System.out.println(threadName+": I got the goods of "+expressGoods);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    },"Deliveryman");

    michael.start();
    TimeUnit.MILLISECONDS.sleep(100);
    green.start();
    TimeUnit.MILLISECONDS.sleep(100);
    deliveryman.start();
}
```

我们加入 Green 线程，并且在 Michael 线程启动后，先启动 Green 线程。输出如下：

```
Michael: I'm Michael and want to delivery a phone to my friend
Michael: There is an emergency meeting
Michael: I give the phone to my workmate.
Michael: He will help me to delivery the phone and return the express document to me
Green: I'm green and want to delivery a macbook to my friend
Green: There is an emergency meeting
Green: I give the macbook to my workmate.
Green: He will help me to delivery the macbook and return the express document to me
Green: I got the express document---phone---
Michael: I got the express document---macbook---
Deliveryman: I'm a deliveryman
Deliveryman: I'm going to get Michael's express from his workmate and give the express document to his workmate
```

可见 Michael 和 Green 进行了数据交换，双方都没有和 Deliveryman 做数据交换。结果是双方的快递都没送到快递员手中，而快递员由于没有收到快递，所以一直被阻塞住了。

根据以上实验结果我们总结一下：

1、Exchanger 支持多个线程做数据交换；

2、多个线程使用同一个 Exchanger 做数据交换时，结果随机，只要凑满一对，就会进行交换。

3、Exchanger实现分析

Exchanger 虽然使用起来很简单，但是其源码还是比较复杂的。下面我们来分析它实现的原理。

Exchanger 类中的注解很详细地阐述了其实现原理，并且用一段伪代码描述了它的原理：

```
for (;;) {
  if (slot is empty) {                // offer
    place item in a Node;
    if (can CAS slot from empty to node) {
      wait for release;
      return matching item in node;
    }
  }
  else if (can CAS slot from node to empty) { // release
    get the item in node;
    set matching item in node;
    release waiting thread;
  }
  // else retry on CAS failure
}
```

可以看到 Exhcanger 使用 slot 作为容器，保存数据 item。当 A 线程进入时，如果发现 slot 是空的，则把 item 放入 node，然后放入 slot。A 线程此时开始阻塞。B 线程进入后发现 slot 不为空，则从 slot 中取出node，把 slot 置空。此线程从 node 中取出数据，并且把自己的数据放入 node 中，然后通知阻塞的线程恢复，阻塞线程恢复后从 node 中取出数据。从而实现了数据的交换。

Java5 的时候，采用的就是这种算法。一般来说没有什么问题。但是当大量并发时，会存在激烈的竞争。于是在 Java6 开始，加入 slot 数组，让不同线程使用不同的 slot，提升并发的性能。不过在没有并发的时候还是使用一个 slot 来做交换。

我们看下核心的 exchange 方法代码：

```
public V exchange(V x) throws InterruptedException {
    Object v;
    Object item = (x == null) ? NULL_ITEM : x; // translate null args
    if ((arena != null ||
         (v = slotExchange(item, false, 0L)) == null &&
         ((Thread.interrupted() || // disambiguates null return
           (v = arenaExchange(item, false, 0L)) == null)))
        throw new InterruptedException();
    return (v == NULL_ITEM) ? null : (V)v;
}
```

arena 为 slot 数组。可以看到 arena 为 null 时，还是使用的 slot 交换算法 slotExchange。如果 arena 不为 null进入 if 的 && 的第二个条件判断。在判断中执行了 arenaExchange，此时通过 arena 也就是 slot 数组完成交换。

具体的算法不再详述，和上面伪代码的思想是一致的。

4、总结

Exchanger 在我们实际开发中使用不多，但是我们也应该了解其用法和原理，以便在合适场景出现时，我们能识别出应该采用 Exchanger。在 Exchanger 类的注解中，可以看到它适用于基因算法。因为不同基因的搭配测试就应该是随机进行交换的。此外还适用于管道的设计（pipeline design），让数据在管道中做交换。

}

28 人齐了，一起行动—CyclicBarrier详解

30 限量供应，不好意思您来晚了—Semaphore详解