在说明这个问题之前我们首先看一个示例, 运行一下看下结果, 如下, 这里将整个代码都列出了, 如果要实验的话可直接使用:

```
1.  import java.io.IOException;
2.  import java.net.InetSocketAddress;
3.  import java.nio.ByteBuffer;
4.  import java.nio.CharBuffer;
5.  import java.nio.channels.SelectionKey;
6.  import java.nio.channels.Selector;
7.  import java.nio.channels.ServerSocketChannel;
8.  import java.nio.channels.SocketChannel;
9.  import java.nio.charset.Charset;
10. import java.util.Iterator;
11. import java.util.concurrent.atomic.AtomicInteger;
12.
13. public class AfterReceiveReadEventShouldUnregisterIt {
14.     private ServerSocketChannel serverSocketChannel;
15.     private final int port = 8989;
16.     private final AtomicInteger COUNTER = new AtomicInteger(1);
17.     private Charset charset = Charset.forName("UTF-8");
18.     private static boolean unregisterEvent = false;
19.
20.     public static void main(String[] args) throws IOException, InterruptedException {
21.         if (args.length >= 1) {
22.             unregisterEvent = Boolean.valueOf(args[0]);
23.         }
24.
25.         System.out.println("unregisterEvent: " + unregisterEvent);
26.         AfterReceiveReadEventShouldUnregisterIt server = new AfterReceiveReadEventShouldUnr
    egisterIt();
27.         server.service();
28.     }
29.
30.     public AfterReceiveReadEventShouldUnregisterIt() throws IOException {
31.         // 阻塞模式
32.         serverSocketChannel = ServerSocketChannel.open();
33.         serverSocketChannel.socket().setReuseAddress(false);
34.         serverSocketChannel.bind(new InetSocketAddress(port));
35.         System.out.println("Server start.");
36.     }
37.
38.     public void service() throws IOException, InterruptedException {
39.         SocketChannel channel = serverSocketChannel.accept();
40.         channel.configureBlocking(false);
41.         System.out.println("Accept a new connection.");
42.
43.         Selector selector = Selector.open();
44.         channel.register(selector, SelectionKey.OP_READ);
45.
46.         while (true) {
47.             int n = selector.select();
48.             if (n == 0) {
49.                 continue;
50.             }
51.
52.             Iterator<SelectionKey> keyIterator = selector.selectedKeys().iterator();
```

```
53.                 while (keyIterator.hasNext()) {
54.                     SelectionKey key = keyIterator.next();
55.                     keyIterator.remove();
56.
57.                     if (unregisterEvent) {
58.                         System.err.println("Unregister event.");
59.                         int readyOps = key.readyOps();
60.                         // // 注销事件
61.                         System.out.println("before unregister interestOps: " + key.interestOps(
       ));    // 1(OP_READ)
62.                         key.interestOps(key.interestOps() & ~readyOps);
63.                         System.out.println("after  unregister interestOps: " + key.interestOps(
       ));    // 0
64.                     }
65.
66.                     if (key.isReadable()) {
67.                         Thread.sleep(200);  // 防止起了太多的线程
68.
69.                         System.out.println("process read event.");
70.                         new ReadTask(key, COUNTER.getAndAdd(1)).start();
71.                     }
72.                 }
73.             }
74.         }
75.
76.         private class ReadTask extends Thread {
77.             private SelectionKey key;
78.             private int count;
79.             private String name;
80.
81.             public ReadTask(SelectionKey key, int count) {
82.                 this.key = key;
83.                 this.count = count;
84.                 this.name = "ReadTask-" + count;
85.             }
86.
87.             @Override
88.             public void run() {
89.                 // 如果读操作延迟(也就是休眠了一段时间)
90.                 int sec = (count == 1) ? 2 : 1;
91.                 System.out.println(name + " sleep " + sec + " second.");
92.                 try {
93.                     Thread.sleep(sec * 1000);
94.                 } catch (InterruptedException e) {
95.                     e.printStackTrace();
96.                 }
97.
98.                 SocketChannel channel = (SocketChannel) key.channel();
99.                 ByteBuffer buffer = ByteBuffer.allocate(64);
100.                try {
101.                    int size = channel.read(buffer);
102.                    buffer.flip();
103.                    System.err.println("----------" + name + " receive " + size + " bytes, deco
        ded msg: " + this.decode(buffer) + "----------");
104.                } catch (IOException e) {
105.                    e.printStackTrace();
106.                }
107.            }
```

```
108.
109.            public String decode(ByteBuffer buffer) {
110.                CharBuffer charBuffer = charset.decode(buffer);
111.                return charBuffer.toString();
112.            }
113.        }
114.
115.
116.    }
```

运行程序的时候可以通过加入运行参数 `true` 、 `false` 来控制是否要注销事件来查看不同的运行结果.这里我们只关注于读事件.好,下面我们查看一下注销与不注销事件程序的运行结果:

不注销事件 unregisterEvent=false

```
unregisterEvent: false
Server start.
Accept a new connection.
process read event.
ReadTask-1 sleep 2 second.
process read event.
ReadTask-2 sleep 1 second.
process read event.
ReadTask-3 sleep 1 second.
process read event.
ReadTask-4 sleep 1 second.
process read event.
ReadTask-5 sleep 1 second.
process read event.
ReadTask-6 sleep 1 second.
----------ReadTask-2 receive 5 bytes, decoded msg: hello----------
process read event.
ReadTask-7 sleep 1 second.
----------ReadTask-3 receive 0 bytes, decoded msg: ----------
----------ReadTask-4 receive 0 bytes, decoded msg: ----------
----------ReadTask-5 receive 0 bytes, decoded msg: ----------
----------ReadTask-1 receive 0 bytes, decoded msg: ----------
----------ReadTask-6 receive 0 bytes, decoded msg: ----------
----------ReadTask-7 receive 0 bytes, decoded msg: ----------
```

注销事件 unregisterEvent=true

```
unregisterEvent: true
Server start.
Accept a new connection.
before unregister interestOps: 1
Unregister event.
after  unregister interestOps: 0
process read event.
ReadTask-1 sleep 2 second.
----------ReadTask-1 receive 5 bytes, decoded msg: hello----------
```

不注销 vs 注销

首先说明下, 读事件的处理过程是分发到一个线程中进行处理的.

从上面的结果可以看到, 如果说有读事件发生, 我们可能没有及时处理(将数据从 channel 中读取处理)又没有将其注销, 那么下一次迭代的时候, 仍然会触发读事件, 这样一个 channel 中的数据就会被多个线程进行处理.可想而知, 这对我们来说是很糟糕的, 会有一堆的问题(如数据如何进行重组).

通常情况下, 服务器的做法是在接收到 `OP_READ` 事件之后,在下一次进行 `selector.select()` 操作之前会将 `OP_READ` 先注销掉,以防止一个连接的读操作会被分到多个线程中.接下来看下 Tomcat、Jetty 对读事件的处理流程.