nio selector 入门 - 黄彪学习笔记

Selector（选择器）是Java NIO中能够检测一到多个NIO通道，并能够知晓通道是否为诸如读写事件做好准备的组件。这样，**一个单独的线程可以管理多个channel，从而管理多个网络连接。为什么使用Selector?**仅用单个线程来处理多个Channels的好处是，只需要更少的线程来处理通道。事实上，可以只用一个线程处理所有的通道。对于操作系统来说，线程之间上下文切换的开销很大，而且每个线程都要占用系统的一些资源（如内存）。因此，使用的线程越少越好。但是，需要记住，现代的操作系统和CPU在多任务方面表现的越来越好，所以多线程的开销随着时间的推移，变得越来越小了。实际上，如果一个CPU有多个内核，不使用多任务可能是在浪费CPU能力。不管怎么说，关于那种设计的讨论应该放在另一篇不同的文章中。在这里，只要知道使用Selector能够处理多个通道就足够了。**Selector的创建**通过调用Selector.open()方法创建一个Selector，如下：
Java代码

```
1. Selector selector = Selector.open();
```

向Selector注册通道为了将Channel和Selector配合使用，必须将channel注册到selector上。通过SelectableChannel.register()方法来实现，如下：
Java代码

```
1. channel.configureBlocking(false);
2. SelectionKey key = channel.register(selector, Selectionkey.OP_READ);
```

与Selector一起使用时，Channel必须处于非阻塞模式下。这意味着不能将FileChannel与Selector一起使用，因为FileChannel不能切换到非阻塞模式。而套接字通道都可以。注意register()方法的第二个参数。这是一个“interest集合”，意思是在通过Selector监听Channel时对什么事件感兴趣。可以监听四种不同类型的事件：ConnectAcceptReadWrite通道触发了一个事件意思是该事件已经就绪。所以，某个channel成功连接到另一个服务器称为“连接就绪”。一个server socket channel准备好接收新进入的连接称为“接收就绪”。一个有数据可读的通道可以说是“读就绪”。等待写数据的通道可以说是“写就绪”。这四种事件用SelectionKey的四个常量来表示：
Java代码

```
1. SelectionKey.OP_CONNECT
2. SelectionKey.OP_ACCEPT
3. SelectionKey.OP_READ
4. SelectionKey.OP_WRITE
```

如果你对不止一种事件感兴趣，那么可以用“位或”操作符将常量连接起来，如下：

Java代码

```
1. int interestSet = SelectionKey.OP_READ | SelectionKey.OP_WRITE;
```

当向Selector注册Channel时，register()方法会返回一个SelectionKey对象。这个对象包含了一些你感兴趣的属性：interest集合ready集合ChannelSelector附加的对象（可选）下面我会描述这些属性。**interest集合**就像向Selector注册通道一节中所描述的，interest集合是你所选择的感兴趣的事件集合。可以通过SelectionKey读写interest集合，像这样：
Java代码

```
1. int interestSet = selectionKey.interestOps();
2. boolean isInterestedInAccept  = interestSet & SelectionKey.OP_ACCEPT;
3. boolean isInterestedInConnect = interestSet & SelectionKey.OP_CONNECT;
4. boolean isInterestedInRead    = interestSet & SelectionKey.OP_READ;
5. boolean isInterestedInWrite   = interestSet & SelectionKey.OP_WRITE;
```

可以看到，用“位与”操作interest 集合和给定的SelectionKey常量，可以确定某个确定的事件是否在interest 集合中。**ready集合**ready 集合是通道已经准备就绪的操作的集合。在一次选择(Selection)之后，你会首先访问这个ready set。Selection将在下一小节进行解释。可以这样访问ready集合：
Java代码

```
1. int readySet = selectionKey.readyOps();
```

可以用像检测interest集合那样的方法，来检测channel中什么事件或操作已经就绪。但是，也可以使用以下四个方法，它们都会返回一个布尔类型：
Java代码

还可以在用register()方法向Selector注册Channel的时候附加对象。如：
Java代码

```
1. SelectionKey key = channel.register(selector, SelectionKey.OP_READ, theObject);
```

通过Selector选择通道一旦向Selector注册了一或多个通道，就可以调用几个重载的select()方法。这些方法返回你所感兴趣的事件（如连接、接受、读或写）已经准备就绪的那些通道。换句话说，如果你对“读就绪”的通道感兴趣，select()方法会返回读事件已经就绪的那些通道。下面是select()方法：（**该方法是阻塞方法**）int select()int select(long timeout)int selectNow()select()阻塞到至少有一个通道在你注册的事件上就绪了。select(long timeout)和select()一样，除了最长会阻塞timeout毫秒(参数)。selectNow()不会阻塞，不管什么通道就绪都立刻返回（译者注：此方法执

行非阻塞的选择操作。如果自从前一次选择操作后，没有通道变成可选择的，则此方法直接返回零。）。select()方法返回的int值表示有多少通道已经就绪。亦即，自上次调用select()方法后有多少通道变成就绪状态。如果调用select()方法，因为有一个通道变成就绪状态，返回了1，若再次调用select()方法，如果另一个通道就绪了，它会再次返回1。如果对第一个就绪的channel没有做任何操作，现在就有两个就绪的通道，但在每次select()方法调用之间，只有一个通道就绪了。selectedKeys()一旦调用了select()方法，并且返回值表明有一个或更多个通道就绪了，然后可以通过调用selector的selectedKeys()方法，访问“已选择键集（selected key set）”中的就绪通道。如下所示：

Java代码

```
1. Set selectedKeys = selector.selectedKeys();
```

当像Selector注册Channel时，Channel.register()方法会返回一个SelectionKey 对象。这个对象代表了注册到该Selector的通道。可以通过SelectionKey的selectedKeySet()方法访问这些对象。可以遍历这个已选择的键集合来访问就绪的通道。如下：

Java代码

```
1. Set selectedKeys = selector.selectedKeys();
2. Iterator keyIterator = selectedKeys.iterator();
3. while(keyIterator.hasNext()) {
4.     SelectionKey key = keyIterator.next();
5.     if(key.isAcceptable()) {
6.         // a connection was accepted by a ServerSocketChannel.
7.     } else if (key.isConnectable()) {
8.         // a connection was established with a remote server.
9.     } else if (key.isReadable()) {
10.        // a channel is ready for reading
11.    } else if (key.isWritable()) {
12.        // a channel is ready for writing
13.    }
14.    keyIterator.remove();
15. }
```

这个循环遍历已选择键集中的每个键，并检测各个键所对应的通道的就绪事件。**注意每次迭代末尾的keyIterator.remove()调用。Selector不会自己从已选择键集中移除SelectionKey实例。必须在处理完通道时自己移除。下次该通道变成就绪时，Selector会再次将其放入已选择键集中。**SelectionKey.channel()方法返回的通道需要转型成你要处理的类型，如ServerSocketChannel或SocketChannel等。**wakeUp()**某个线程调用select()方法后阻塞了，即使没有通道已经就绪，也有办法让其从select()方法返回。只要让其它线程在第一个线程调用select()方法的那个对象上调用

Selector.wakeup()方法即可。阻塞在select()方法上的线程会立马返回。如果有其它线程调用了wakeup()方法，但当前没有线程阻塞在select()方法上，下个调用select()方法的线程会立即“醒来（wake up）”。**close()**用完Selector后调用其close()方法会关闭该Selector，且使注册到该Selector上的所有SelectionKey实例无效。通道本身并不会关闭。----------下面的例子是当客户端发送的字符串，server端接收并打印，然后将接收的字符串反馈给clientserver端代码