[**java nio Selector的使用－服务器端**](http://flym.iteye.com/blog/392350)

　　前些时候花了一些时间在研究java.nio的api使用机制,看了好久,也觉得不习惯它的使用方式和用法.毕竟自己对C语言了解太少,也不太了解C语言在网络编程上的用法。对这种底层下的编程太不习惯，还是应该好好了解下底层的东西，要不然就光会使用别人的东西，如果是自己写一个，就写不出来了。

　　从java1.4以来,java nio就出现在java的api中，在日常的使用当中，基本上都是围绕着java.io中的几个inputStream(outputStream)和reader(writer)在转，要想编写一些其他形式的调用，还真不会。我也看了下最新的springframework中的FileCopyUtils中的代码，也是将各种操作集合给java.io来做。好像java.nio用得不是很多。看了下java.nio的描述信息，感觉这是用在网络编程上的。比如文件下载服务，通信服务等地方。自己暂时还用不上网络上的编程，不过等到用的时候还去学，就太晚了。

　　看了下关于Selector的使用方法，官方的说法是一个“多路复用器”，从我自己的角度来讲就感觉像一个服务总线，负责统一各个处理程序的消息注册，统一接收客户端或服务器消息信息，再分发给不同的事件处理程序进行处理。整个流程就一个注册->接收->处理的过程，从使用者的角度来讲，直接使用这些api还不太成熟，毕竟这些api都太底层了，需要了解太多的技术细节，也不太适合像我这种不了解C语言网络编程的人。这周花了三天的时候专门研究了下整个java.nio包，重点看了下关于Selector的运用（datagram和pipe还不太会用），结合了网络上的很多例子（尤其是《java nio》这本书上的例子），对selector总算有了很大的认识，对底层的io编程也有了新的了解。

　　写了个模拟下载的例子，服务器端模拟一个拥有整个硬盘资源的处理程序。客户端通过发送要下载的文件（通过完整文件路径），从而实现由服务器写文件到客户端，客户端保存接收的整体流程。其中，仅涉及到了数据传输的基本运用，即没有运用到网络编程上的urlConnection，也没有用到专门的socket,客户端也没有实现一个文件多线程下载的机制。仅仅作为一个selector的下载练习使用（当然，如果要求不高，也可以用到实际编程的）。

　　服务器端基本思路就是打开链接,绑定端口,接收信息,处理信息。详细过程如下：

　　第一步：创建服务器端socketChannel，并绑定指定端口，注册到selector上。

**Java代码  [收藏代码](javascript:void())**

1. selector = Selector.open();
2. ServerSocketChannel serverSocketChannel=ServerSocketChannel.open();
3. serverSocketChannel.configureBlocking(**false**);
4. serverSocketChannel.socket().bind(**new** InetSocketAddress(1234));
5. serverSocketChannel.register(selector, SelectionKey.OP\_ACCEPT);

　　都是标准的步骤，先open,再配置block为异步的，服务器socket绑定本机端口，注册到selector上，并指定key为ACCEPT。

　　第二步：接收消息，处理信息咯。

**Java代码  [收藏代码](javascript:void())**

1. **for**(; ;) {
2. selector.select();
3. Iterator<SelectionKey> keyIterator = selector.selectedKeys().iterator();
4. **while**(keyIterator.hasNext()) {
5. SelectionKey key = keyIterator.next();
6. **if**(key.isValid())
7. handle(key);
8. keyIterator.remove();
9. }
10. }

　　这也是标准步骤，先进行select，再获得selectedKey,迭代，处理，再remove掉。

在网上，看到有些例子中，对selector.select()中返回的值进行判断，如果返回为o则continue,在我这个程序中，经测试当selector.select()返回为０时，而selector.selectedKeys()确不为０，这样就没有处理信息了。从官方doc上来看，关于select()的返回值解释为“已更新其准备就绪操作集的键的数目，该数目可能为零”，即这个数目指已更新的键集，故在处理中可能键集没有更新，而选择的消息处理keys却不为０，这种情况是正确的。不清楚是不是这个意思，还望高人来解释一下。

　　第三步：就是handle方法了，处理消息事件。

**Java代码  [收藏代码](javascript:void())**

1. **if**(key.isAcceptable()) {
2. ServerSocketChannel channel = (ServerSocketChannel) key.channel();
3. SocketChannel socketChannel = channel.accept();
4. socketChannel.configureBlocking(**false**);
5. socketChannel.register(selector, SelectionKey.OP\_READ);//注册读事件
6. map.put(socketChannel, **new** Handle());//把socket和handle进行绑定
7. }
8. //用map中的handle处理read和write事件,以模拟多个文件同时进行下载
9. **if**(key.isReadable() || key.isWritable()) {
10. SocketChannel socketChannel = (SocketChannel) key.channel();
11. **final** Handle handle = map.get(socketChannel);
12. **if**(handle != **null**)
13. handle.handle(key);
14. }

　　在以上方法中，我在主方法中仅处理appcet事件，再为每个连接到的socketChannel注册读事件，再在读消息处理中注册写事件。而读和写消息处理，我用了一个内部类来处理，即每个内部类来绑定一个socketChannel,单独处理每个socketChannel。这样的处理，是满足客户端对服务器端发起多个请求，来下载不同的文件，这样服务器端就可为不同的客户端socketChannel定制不同的处理程序了。内部类的定义如下：

**Java代码  [收藏代码](javascript:void())**

1. **private** **class** Handle{
2. **private** StringBuilder message;
3. **private** **boolean** writeOK = **true**;
4. **private** ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
5. **private** FileChannel fileChannel;
6. **private** String fileName;
7. }

 　　message指由客户端发送的信息，在此定义此信息为客户端请求的文件信息。由message来得到服务器端的文件名路径信息，并保存到fileName中，fileChannel即为由此fileName取得的channel。byteBuffer就是用来写数据的字节数据缓冲器了。

　　handle单独处理读和写事件，在读事件中，解析文件名，并注册写事件，代码如下：

**Java代码  [收藏代码](javascript:void())**

1. **if**(key.isReadable()) {
2. SocketChannel socketChannel = (SocketChannel) key.channel();
3. **if**(writeOK)
4. message = **new** StringBuilder();
5. **while**(**true**) {
6. byteBuffer.clear();
7. **int** r = socketChannel.read(byteBuffer);
8. **if**(r == 0)
9. **break**;
10. **if**(r == -1) {
11. socketChannel.close();
12. key.cancel();
13. **return**;
14. }
15. message.append(**new** String(byteBuffer.array(), 0, r));
16. }
17. //将接收到的信息转化成文件名,以映射到服务器上的指定文件
18. **if**(writeOK && invokeMessage(message)) {
19. socketChannel.register(selector, SelectionKey.OP\_WRITE);
20. writeOK = **false**;
21. }
22. }

 以上代码就主要是读信息，并解析信息成一个文件名，并注册写事件了。当然还处理客户端断开连接事件，读到信息为－１时，断开连接。其中处理文件信息代码如下：

**Java代码  [收藏代码](javascript:void())**

1. String m = message.toString();
2. **try** {
3. File f = **new** File(m);
4. **if**(!f.exists())
5. **return** **false**;
6. fileName = m;
7. **return** **true**;
8. } **catch**(Exception e) {
9. **return** **false**;
10. }

其中就是将message转化成一个fileName,以供在写的时候能够从fileName中取得fileChannel,此方法保存fileName是存在的。

　　下面看写事件的处理：

**Java代码  [收藏代码](javascript:void())**

1. //向客户端写数据
2. **if**(key.isWritable()) {
3. **if**(!key.isValid())
4. **return**;
5. SocketChannel socketChannel = (SocketChannel) key.channel();
6. **if**(fileChannel == **null**)
7. fileChannel = **new** FileInputStream(fileName).getChannel();
8. byteBuffer.clear();
9. **int** w = fileChannel.read(byteBuffer);
10. //如果文件已写完,则关掉key和socket
11. **if**(w <= 0) {
12. fileName = **null**;
13. fileChannel.close();
14. fileChannel = **null**;
15. writeOK = **true**;
16. socketChannel.close();
17. key.channel();
18. **return**;
19. }
20. byteBuffer.flip();
21. socketChannel.write(byteBuffer);
22. }

 　　写处理中，主要就是打开本地的文件channel将fileChannel中的数据写到socketChannel中，如果数据已经写完毕，则关掉相应channel。

　　至此，服务器端的信息就处理完毕，运行这个程序就只需要在main方法中，new().call()就可以了。当然，有服务器端还需要客户端才行，客户端信息请参照下一笔记。

　　服务器端代码随附件中。

* [DownloadServer.rar](http://dl.iteye.com/topics/download/08a2c458-0052-3bb3-b548-da0cc29ae31a) (1.7 KB)
* 下载次数: 328

### [java nio Selector的使用－客户端](http://flym.iteye.com/blog/392373)

　　接上一篇，客户端的程序就相对于简单了，只需要负责连接，发送下载文件名，再读数据就行了。主要步骤就是注册->连接服务器->发送下载请求->读数据->断开连接。

　　第一步：注册，并注册connect事件。

**Java代码  [收藏代码](javascript:void())**

1. **if**(selector == **null**)
2. selector = Selector.open();
3. SocketChannel channel = SocketChannel.open();
4. channel.configureBlocking(**false**);
5. channel.connect(**new** InetSocketAddress("localhost", 1234));
6. channel.register(selector, SelectionKey.OP\_CONNECT);

　　以上步骤，在实际中发现configure和connet并不能随意交换位置，即不能将connect放在configureBlocking的前面，更不能在open中直接添加InetSocketAddress参数了。在官方doc中，open(InetAddress)的解释是“这种便捷方法的工作方式就像以下过程一样：调用 open() 方法、在得到的套接字通道上调用 connect 方法、向其传递 remote，然后返回该通道。”，但不知道为什么在connect之后，就不能配置block了，导致无法进行数据下载和通信。

　　第二步：处理connect事件

**Java代码  [收藏代码](javascript:void())**

1. //连接事件
2. **if**(key.isConnectable()) {
3. SocketChannel socketChannel = (SocketChannel) key.channel();
4. **if**(socketChannel.isConnectionPending())
5. socketChannel.finishConnect();
6. socketChannel.write(ByteBuffer.wrap(serverFileName.getBytes()));//向服务器发信息,信息中即服务器上的文件名
7. socketChannel.register(selector, SelectionKey.OP\_READ);
8. }

　　在以上步骤中，要完成连接之后，向服务器端发送了下载的文件名的数据信息，并注册read事件。服务器端在接收到相应文件名之后，就开启write事件向客户端进行发送数据了，客户端此时就可以进行数据读取了。

　　第三步：处理read事件

**Java代码  [收藏代码](javascript:void())**

1. **if**(key.isReadable()) {
2. SocketChannel socketChannel = (SocketChannel) key.channel();
3. byteBuffer.clear();
4. **if**(!socketChannel.isConnected())
5. **return** **null**;
6. //向本机下载文件创建文件channel
7. **if**(fileChannel == **null**)
8. fileChannel = **new** RandomAccessFile(localFileName, "rw").getChannel();
9. **int** r = socketChannel.read(byteBuffer);
10. //如果文件下载完毕,则关掉channel,同时关掉socketChannel
11. **if**(r <= 0) {
12. **if**(fileChannel != **null**)
13. fileChannel.close();
14. channel.close();
15. key.cancel();
16. **return** **null**;
17. }
18. byteBuffer.flip();
19. //写到下载文件中
20. fileChannel.write(byteBuffer);
21. }

　　就是处理读信息，如果数据已经读取完毕，则完成相应下载保存文件的文件流，并退出程序。

　　这样，整个客户端就完成了，在运行时，我同时启用１０个线程来向服务器端读同一个文件，并保存为不同的文件备份，以达到模拟数据传输的功能。如下所示：

**Java代码  [收藏代码](javascript:void())**

1. ExecutorService executorService = Executors.newSingleThreadExecutor();
2. **for**(**int** i = 0; i < 10; i++) {
3. executorService.submit(**new** DownloadClient<Object>("d:/log4j.log", "d:/down" + i + ".log"));
4. }
5. executorService.shutdown();

　　整个selector仅是一个作为练习用的小例子，如果用在实际代码中，还需要处理不同的异常和相应的逻辑等。对于学习还是有一定的帮助的。希望对你有用。

　　随附客户端代码。

* [DownloadClient.rar](http://dl.iteye.com/topics/download/aa47a3c6-9104-3b22-ada7-c4dd70ef9c28) (1.3 KB)
* 下载次数: 277