废话不多说先上代码！！！

简单的服务端和客户端代码，5个服务端监听了5个端口，都注册在Selector中，每一个服务端都接受客户端访问，将已连接通道都注册到selector

服务端

```java
public class NioTest12 {
 public static void main(String[] args) throws Exception {
 int[] ports = new int[5];

 ports[0] = 5000;
 ports[1] = 5001;
 ports[2] = 5002;
 ports[3] = 5003;
 ports[4] = 5004;

  Selector selector = Selector.open();

 // System.out.println(SelectorProvider.provider().getClass());
 // System.out.println(sun.nio.ch.DefaultSelectorProvider.create().getClass());
  for (int i = 0; i < ports.length; i++) {
  ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
  serverSocketChannel.configureBlocking(false);//调整阻塞模式, false不阻塞
  InetSocketAddress address = new InetSocketAddress(ports[i]);
  serverSocketChannel.bind(address);//监听端口
  //注册accept事件到selector
  serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);

  System.out.println("监听端口: " + ports[i]);
  }
  while (true){
  int numbers = selector.select();//获取所有selectKey键
  System.out.println("number:" + numbers);

  Set<SelectionKey> selectionKeys = selector.selectedKeys();
  System.out.println("selectionKeys:" + selectionKeys);

  Iterator<SelectionKey> iterator = selectionKeys.iterator();
  while (iterator.hasNext()){
  SelectionKey selectionKey = iterator.next();
if(selectionKey.isAcceptable()){
ServerSocketChannel serverSocketChannel = (ServerSocketChannel) selectionKey.channel();
SocketChannel socketChannel = serverSocketChannel.accept();//等待连接连上
socketChannel.configureBlocking(false);

socketChannel.register(selector, SelectionKey.OP_READ);//将连接注册到selector上，关注事件是读

iterator.remove();//事件用完必须要移除掉,不然会异常

System.out.println("获得客户端连接： " + socketChannel);
}else if(selectionKey.isReadable()){//读事件被选取
SocketChannel socketChannel = (SocketChannel) selectionKey.channel();
//进行读操作
int bytesRead = 0;
while (true){
ByteBuffer byteBuffer = ByteBuffer.allocate(512);

byteBuffer.clear();

int read = socketChannel.read(byteBuffer);

if(read <= 0){
break;
}

byteBuffer.flip();

socketChannel.write(byteBuffer);

bytesRead += read;
}
System.out.println("读取" + bytesRead + ", 来自于" + socketChannel);

iterator.remove();//事件用完必须要移除掉
}
}
}
}
}
```

客户端

```java
public class NioTest12_Client {
 public static void main(String[] args) throws Exception{
 try {
 SocketChannel socketChannel = SocketChannel.open(new
InetSocketAddress("localhost", 5000));
 socketChannel.configureBlocking(false);
 ByteBuffer byteBuffer = ByteBuffer.allocate(512);
 byteBuffer.put("hello,server! I am client.".getBytes(Charset.defaultChar
set()));
 byteBuffer.flip();
 socketChannel.write(byteBuffer);
  }finally {
  }
  }
}
```

第二种服务端

只有一个线程接受客户端多连接，互发消息

```java
public class NioServer {
 private static Map<String, SocketChannel> clientMap = new HashMap<>();//
维护所有客户端对信息

 public static void main(String[] args) throws Exception {
 //固定模板代码
 ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
 serverSocketChannel.configureBlocking(false);
 ServerSocket serverSocket = serverSocketChannel.socket();
 serverSocket.bind(new InetSocketAddress(8899));
  //服务端注册到选择器
  Selector selector = Selector.open();
  serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);

  //服务端监听
  while (true){
  try {
  selector.select();//这种调用在没有通道就绪时将无限阻塞,有感兴趣事件发生时通
过

  Set<SelectionKey> selectionKeys = selector.selectedKeys();//获取selectio
nKeys集合
```

```
selectionKeys.forEach(selectionKey -> {
final SocketChannel client;
try {
if(selectionKey.isAcceptable()){//判断是否有客户端连接
ServerSocketChannel server = (ServerSocketChannel) selectionKey.channel();//获得server对象
client = server.accept();//获取对应的客户端对象client
client.configureBlocking(false);//配置成非阻塞
client.register(selector, SelectionKey.OP_READ);//注册到selector，感兴趣事件为读
// client.register(selector, SelectionKey.OP_WRITE);

String key = "[" + client.getRemoteAddress() + "]";//设置地址为键
System.out.println(key + "已连接");
clientMap.put(key, client);//放入map
}else if(selectionKey.isReadable()){//读事件触发
client = (SocketChannel)selectionKey.channel();//获取client对象
ByteBuffer byteBuffer = ByteBuffer.allocate(1024);

int count = client.read(byteBuffer);//读取
if(count > 0){
byteBuffer.flip();

String receiveMessage = String.valueOf(Charset.defaultCharset().decode(byteBuffer).array());//获取返回消息
System.out.println(client + ":" + receiveMessage);

String senderKey = "[" + client.getRemoteAddress() + "]";//组织返回消息的key

clientMap.entrySet().forEach(entry -> {
ByteBuffer wirteBuffer = ByteBuffer.allocate(1024);
String sendMessage = "来自" + senderKey + "的消息：" + receiveMessage;//组织返回消息
wirteBuffer.put(sendMessage.getBytes(Charset.defaultCharset()));
wirteBuffer.flip();

SocketChannel curClient = entry.getValue();//获取当前被选中的client

try {
curClient.write(wirteBuffer);//写入
} catch (IOException e) {
e.printStackTrace();
}
});
}
}
}catch (Exception e){
e.printStackTrace();
}
});
selectionKeys.clear();//当次循环完，清空集合

}catch (Exception e){
e.printStackTrace();
}
}
}
}
```

第二种客户端

```
public class NioClient {
 public static void main(String[] args) {
 try {
 SocketChannel socketChannel = SocketChannel.open();
 socketChannel.configureBlocking(false);

 Selector selector = Selector.open();
 socketChannel.register(selector, SelectionKey.OP_CONNECT);//注册

  socketChannel.connect(new InetSocketAddress("localhost", 8899));

  while (true){
  selector.select();
  Set<SelectionKey> selectionKeys = selector.selectedKeys();

  selectionKeys.forEach(selectionKey -> {
  if(selectionKey.isConnectable()){
  SocketChannel client = (SocketChannel)selectionKey.channel();//获取clien
t对象
  if(client.isConnectionPending()){//判断连接是否就绪
  try {
```

```
client.finishConnect();//完成连接
ByteBuffer writeBuffer = ByteBuffer.allocate(1024);
writeBuffer.put((LocalDateTime.now()+"已经连接").getBytes(Charset.defaultCharset()));
writeBuffer.flip();
client.write(writeBuffer);
//JDK5自带线程池，单个线程
ExecutorService executorService = Executors.newSingleThreadExecutor(Executors.defaultThreadFactory());
executorService.submit(()->{
while (true){
try {
writeBuffer.clear();
InputStreamReader inputStream = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(inputStream);
String sendMessage = br.readLine();
writeBuffer.put(sendMessage.getBytes(Charset.defaultCharset()));
writeBuffer.flip();
client.write(writeBuffer);
}catch (Exception e){
e.printStackTrace();
}
}
});


} catch (IOException e) {
e.printStackTrace();
}
}
try {
client.register(selector, SelectionKey.OP_READ);//注册read事件
} catch (ClosedChannelException e) {
e.printStackTrace();
}
}else if(selectionKey.isReadable()){
SocketChannel client = (SocketChannel) selectionKey.channel();

ByteBuffer readBuffer = ByteBuffer.allocate(1024);

try {
```

```java
int count = client.read(readBuffer);
if(count > 0){
String receiveMessage = new String(readBuffer.array(), 0, count);
System.out.println(receiveMessage);
}
} catch (IOException e) {
e.printStackTrace();
}
}
});
selectionKeys.clear();//循环结束清空selectionKeys
}
}catch (Exception e){
e.printStackTrace();
}
}
}
```