

第一节 详解NIO的前世今生-原理解析

1. 初步了解BIO

1.1 服务端

1.2 客户端

1.3 BIO在单线程情况下无法解决并发问题

1.3.1 如何证明

1.4 BIO在多线程情况下可以解决并发问题

1.4.1 代码层面如何解决

1.4.2 存在的问题

2. 理解NIO

2.1 NIO的设计初衷

2.2 代码层面实现

2.3 问题的所在

2.3.1 natvie方法在哪里

2.3.2 重点分析

1. 初步了解BIO

说起BIO，我们就来谈一下网络编程，其实网络编程，很多人学的都很差。网络编程的基本模型是C/S模型，即两个进程间的通信。服务端提供IP和监听端口，客户端通过连接操作向服务端监听的地址发起连接请求，通过三次握手连接，如果连接成功建立，双方就可以通过套接字进行通信。传统的同步阻塞模型开发中，ServerSocket负责绑定IP地址，启动监听端口；Socket负责发起连接操作。连接成功后，双方通过输入和输出流进行同步阻塞式通信。

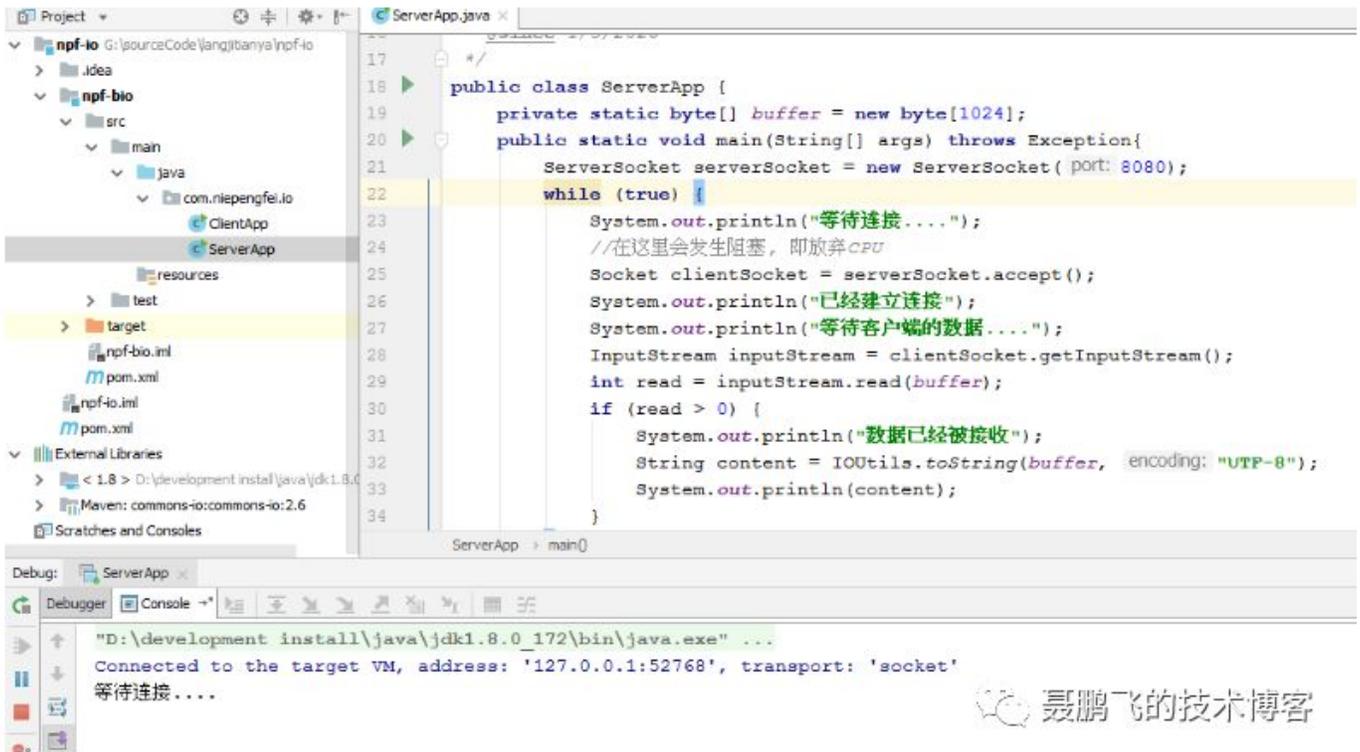
本文的git连接：

<https://github.com/pengfeinie/npf-io>

1.1 服务端

现在我们来写一个很简单的BIO实现的服务端。当我们把服务端启动的时候，服务端将会阻塞在下面的这行代码，目的是等待客户端来连接。

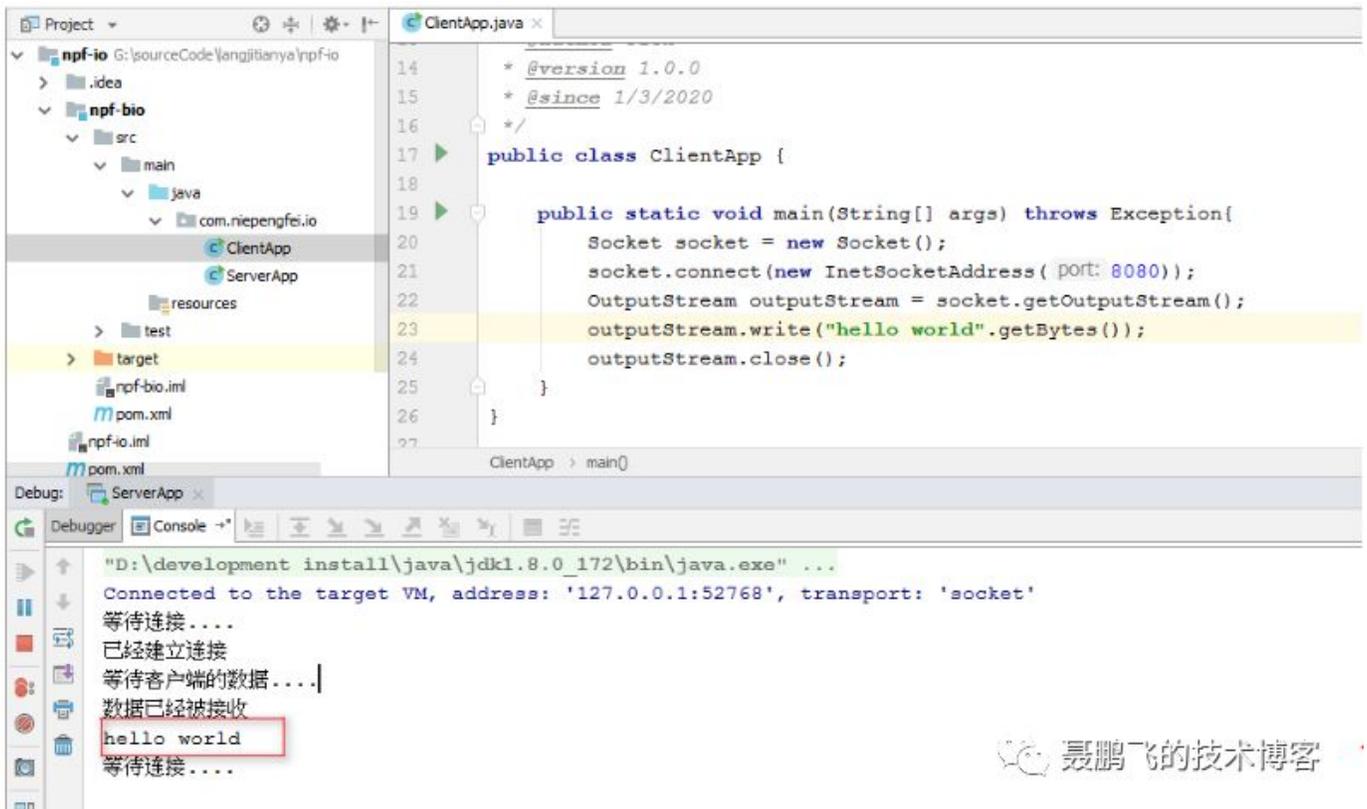
```
serverSocket.accept();
```



聂鹏飞的技术博客

1.2 客户端

当我们运行客户端，服务端将打印出客户端发来的数据，然后程序再次阻塞，等待客户端来连接，因为我们的服务端采用了while 循环。



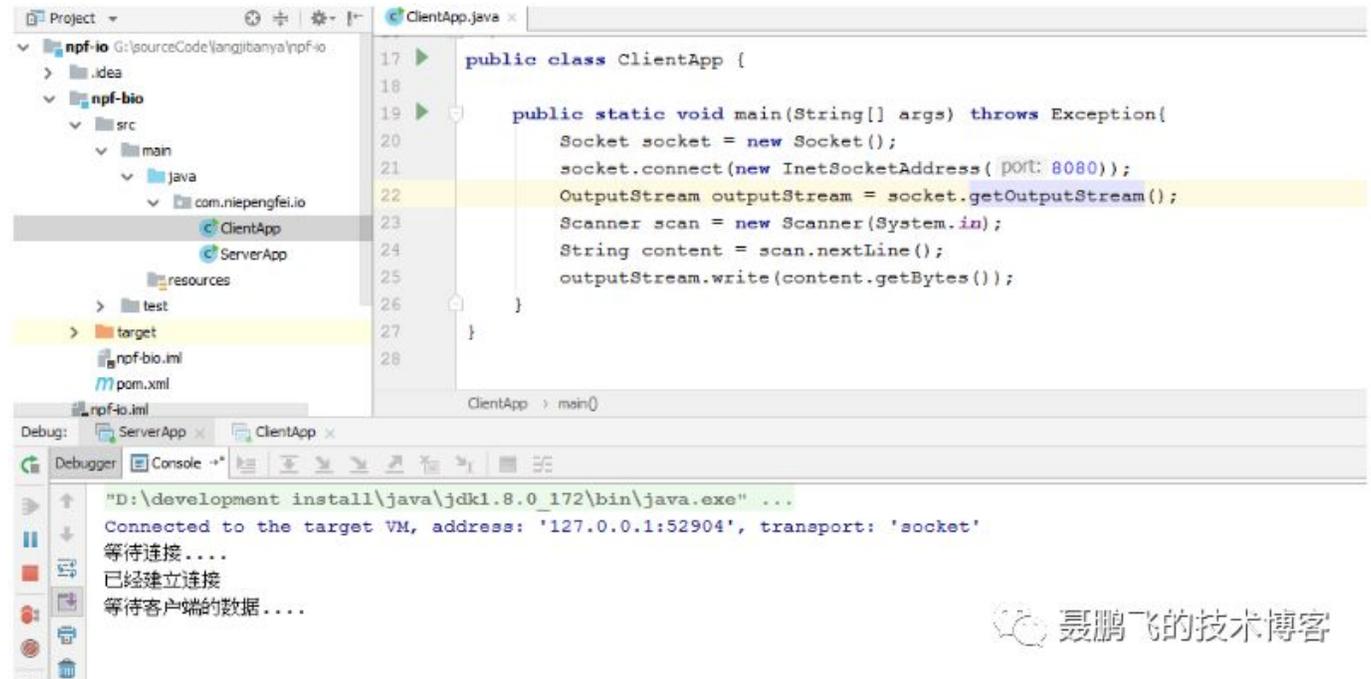
聂鹏飞的技术博客

1.3 BIO在单线程情况下无法解决并发问题

客户端假设如上面所写那样，你永远都体会不到一点，那就是服务端的以下代码也是阻塞的，即read方法是阻塞的。为什么这么说呢？上述那个客户端的例子一点都不好，可以说非常的差劲，但是几乎所有学习 BIO编程的Java程序员都是从这个例子中学习过来的，我真想把最开始写这个例子程序的人摁到地上摩擦，简直是误人子弟。

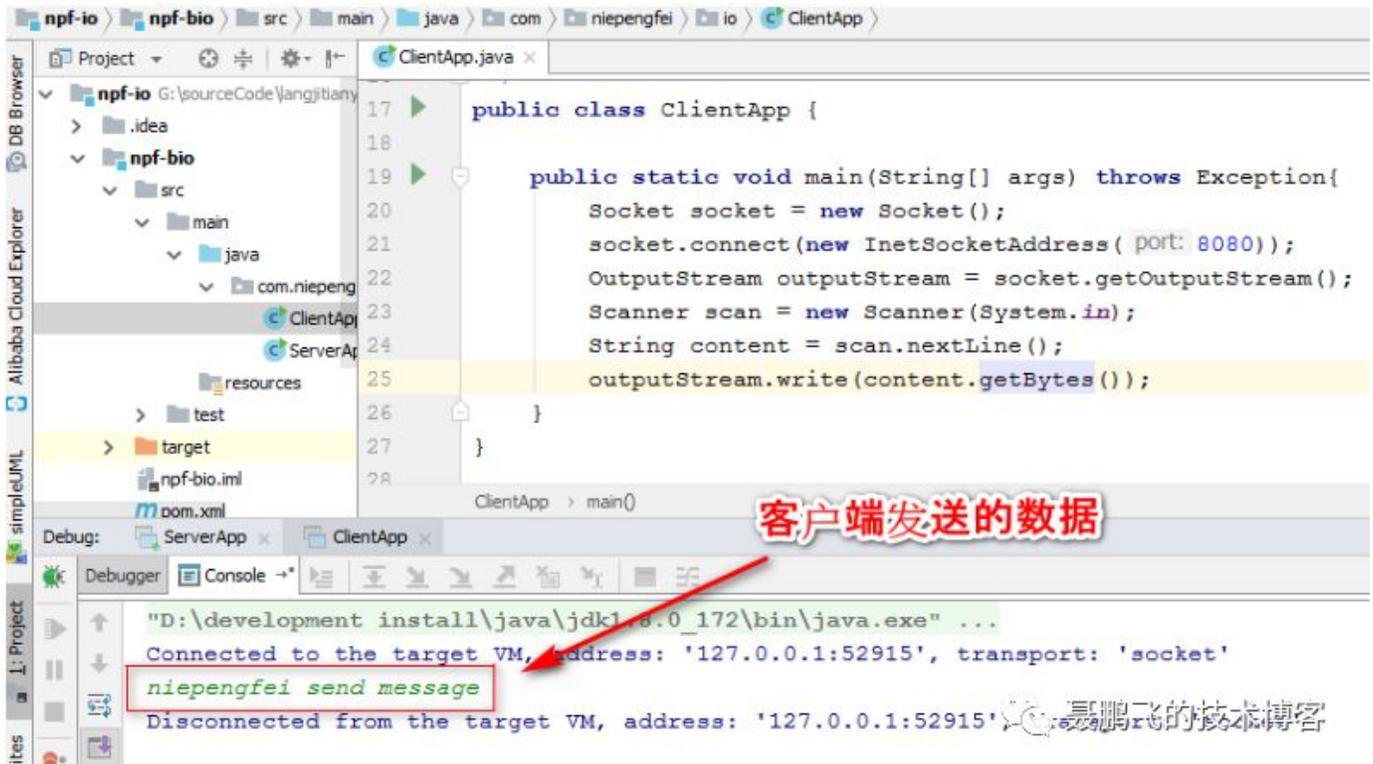
```
InputStream.read(buffer);
```

我们来看一下，我把客户端的代码修改一下。让客户端连接上服务端的时候，不要立即去发数据，而是在那里等待一会，这样你就可以很明显的看到，服务端将会阻塞在read方法上面。不信你可以看看，当我们运行客户端的时候，服务端的控制台将会打印如下信息，这就足以说明服务端已经阻塞在read方法上面，等待客户端发送数据。

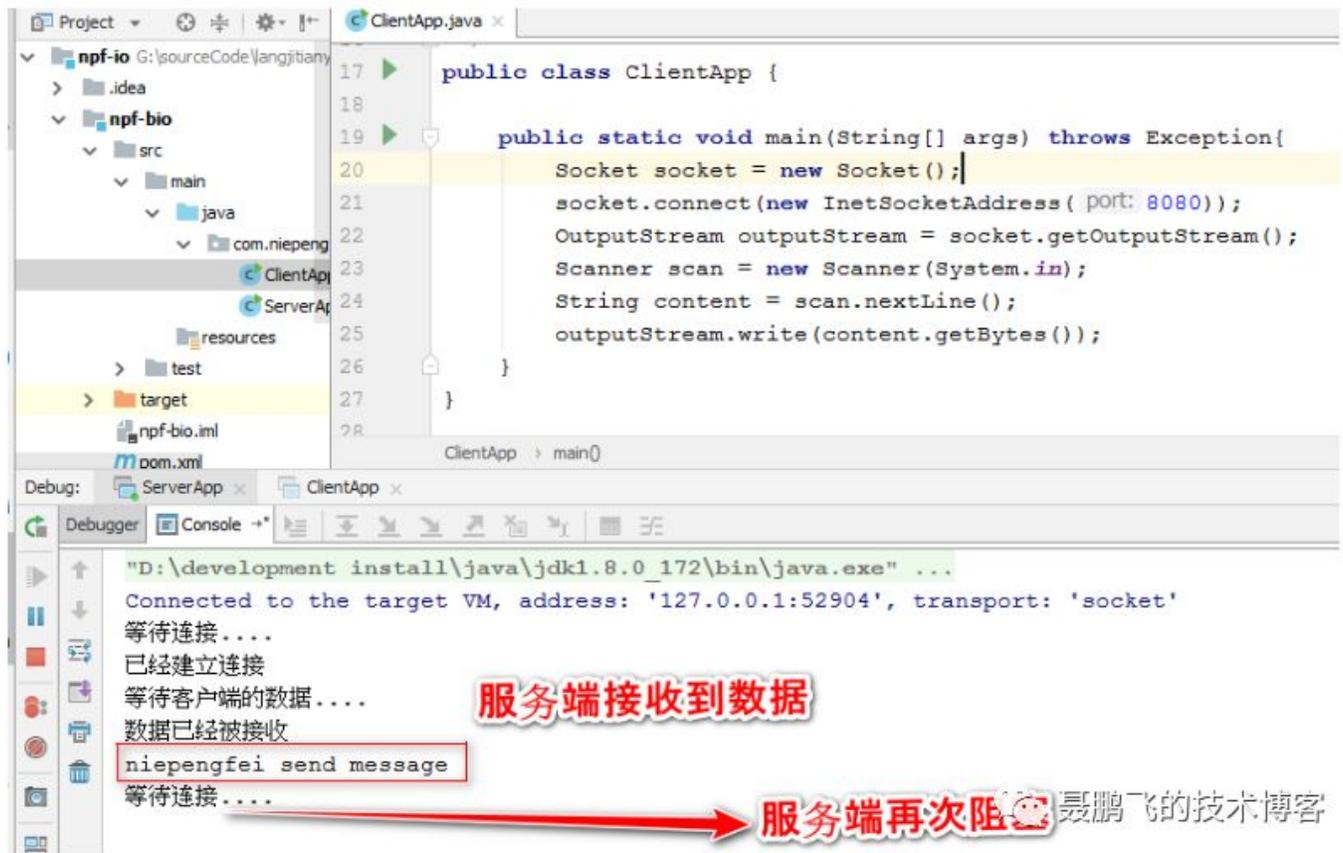


聂鹏飞的技术博客

之后，我在客户端的控制台发送数据，随即你可以在服务端看到接收的数据，然后服务端打印出来之后，再次阻塞在accept方法上面，等待客户端的连接。



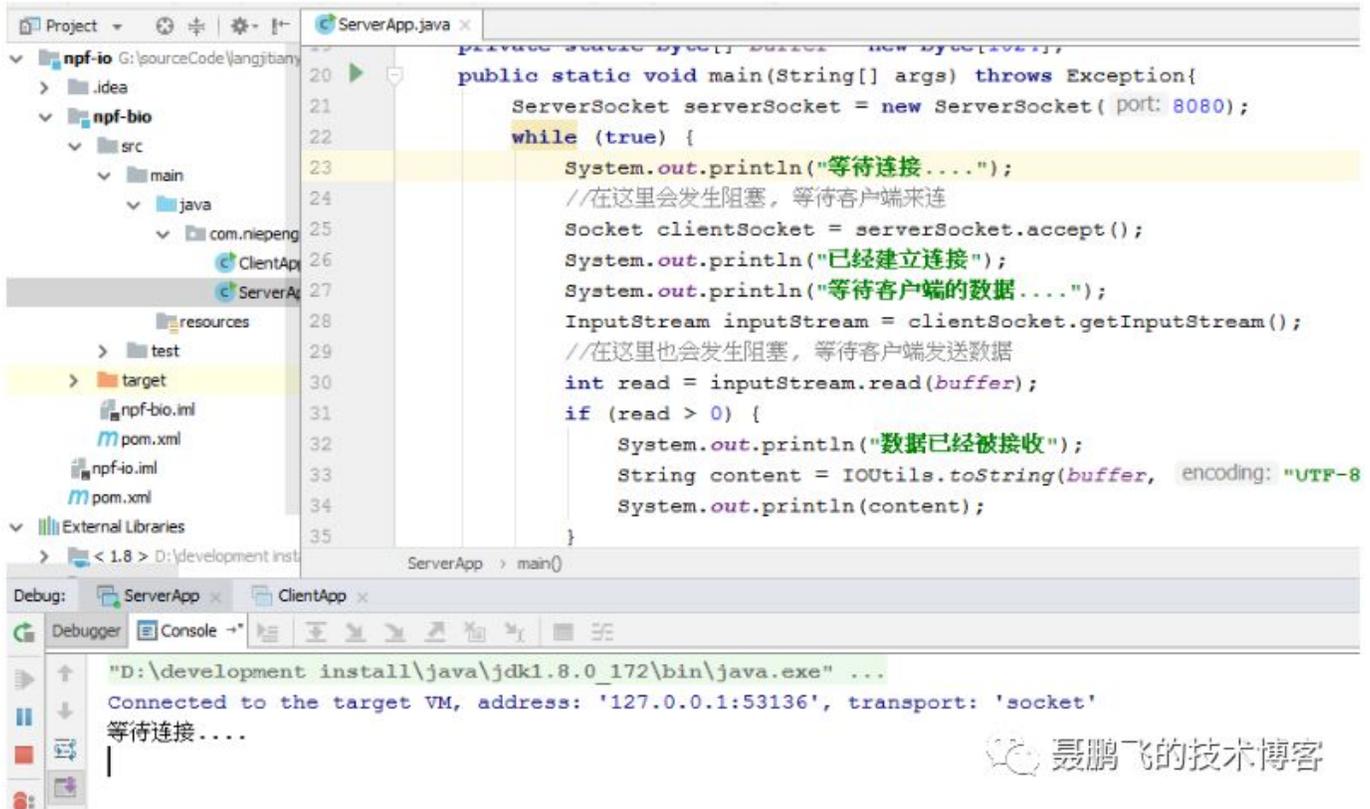
然后我们再到服务的控制台看下：



通过以上分析，采用传统的BIO编程的话，会造成两个地方阻塞。第一个就是accept，等待客户端来连。第二个就是read，等待客户端发送数据。这两个阻塞会造成什么问题呢？那就决定了如果要让BIO实现并发，那么就必须借助多线程。写到这里，可能还是有人不明白，我真的是醉了，好吧。如果你还不相信，那么下面我就用例子来证明这些理论，好吧。

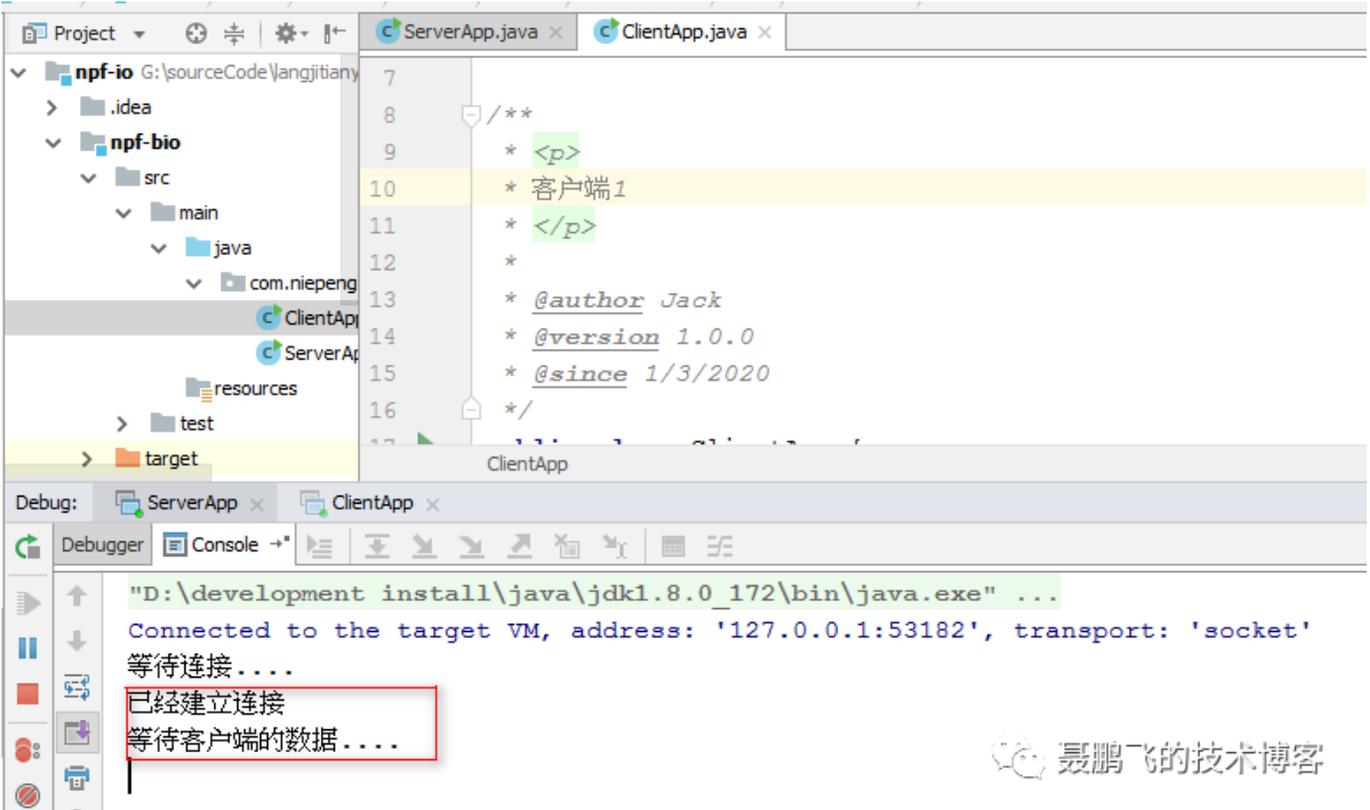
1.3.1 如何证明

第一步：我们启动服务端。那么服务端必须阻塞在accept方法上面，等待客户端来连。



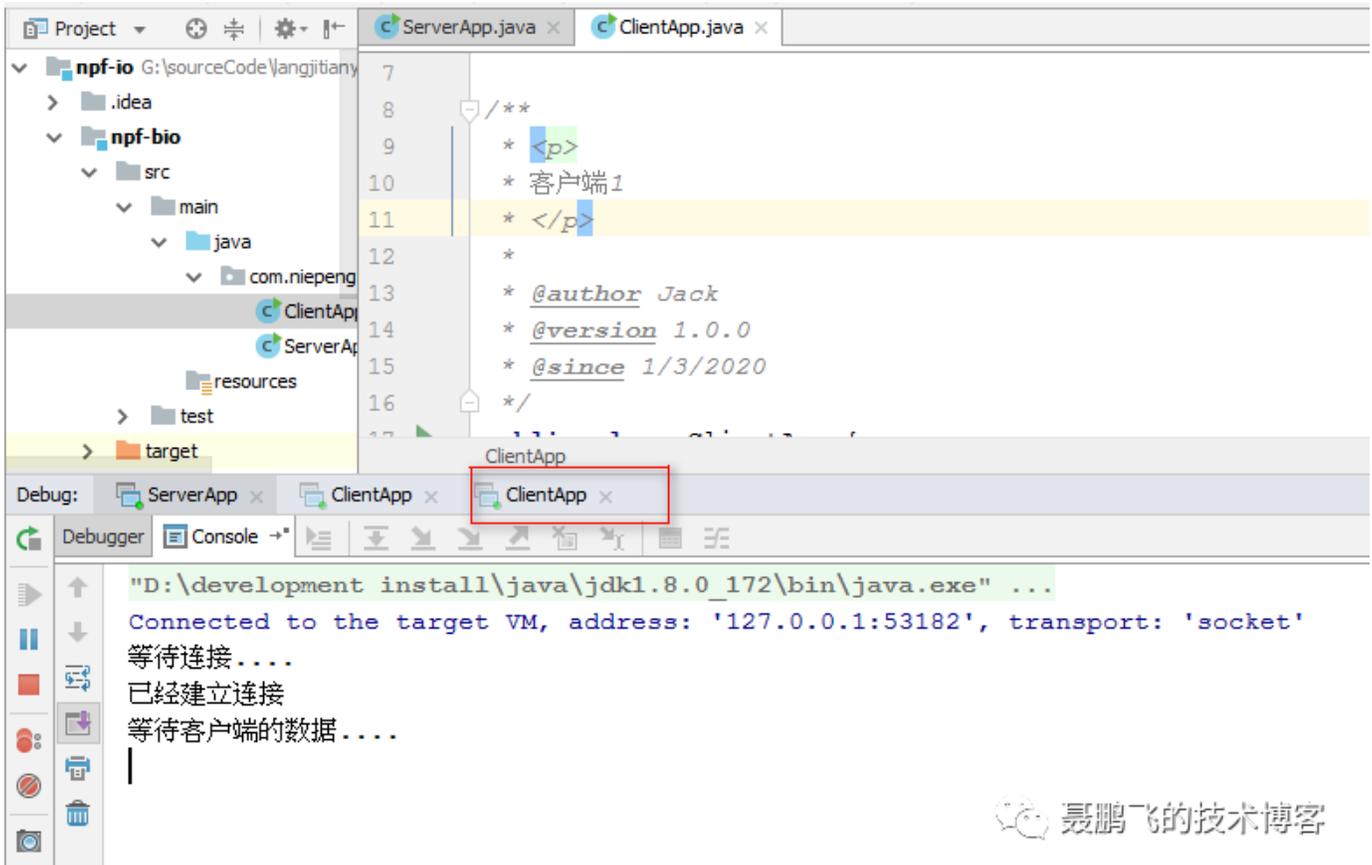
聂鹏飞的技术博客

第二步：我们启动客户端，连接到服务端之后，一直不发送数据，让服务端一直阻塞在read方法上面。如果能在服务端控制台看到如下信息，那就证明我们的客户端连接上来了。

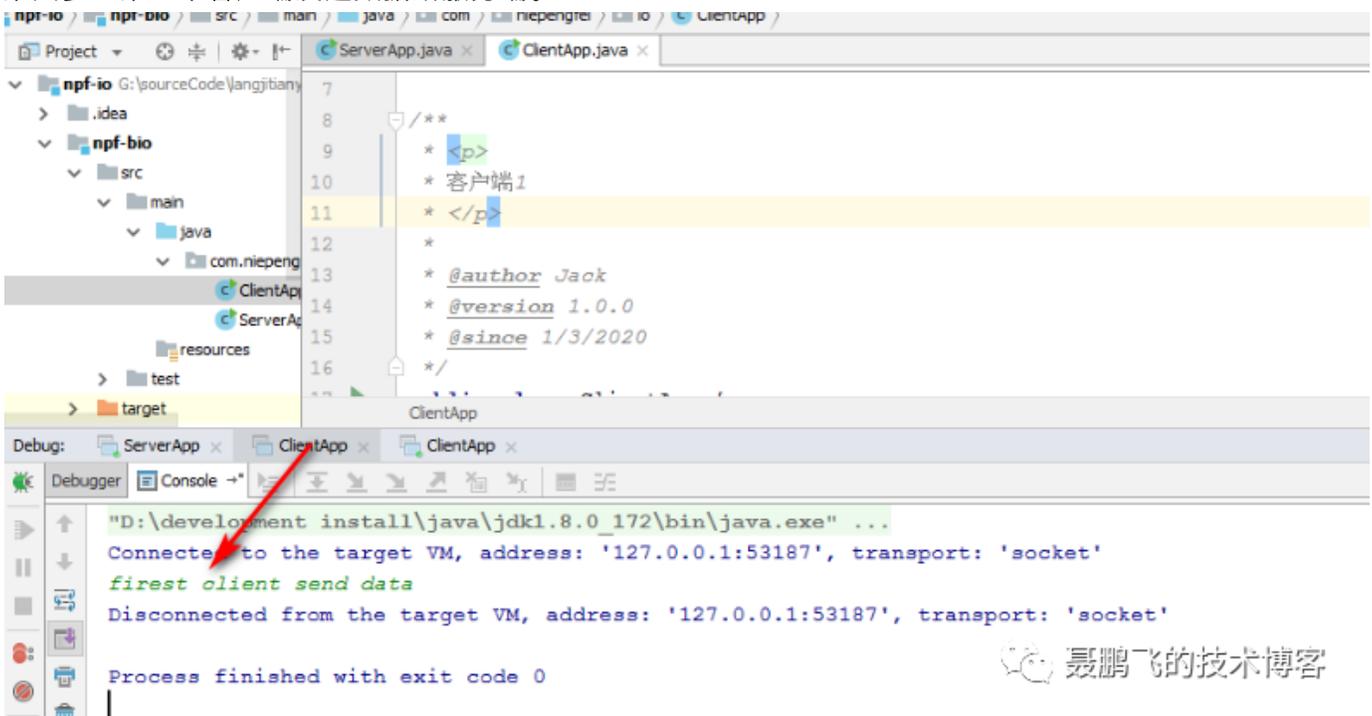


聂鹏飞的技术博客

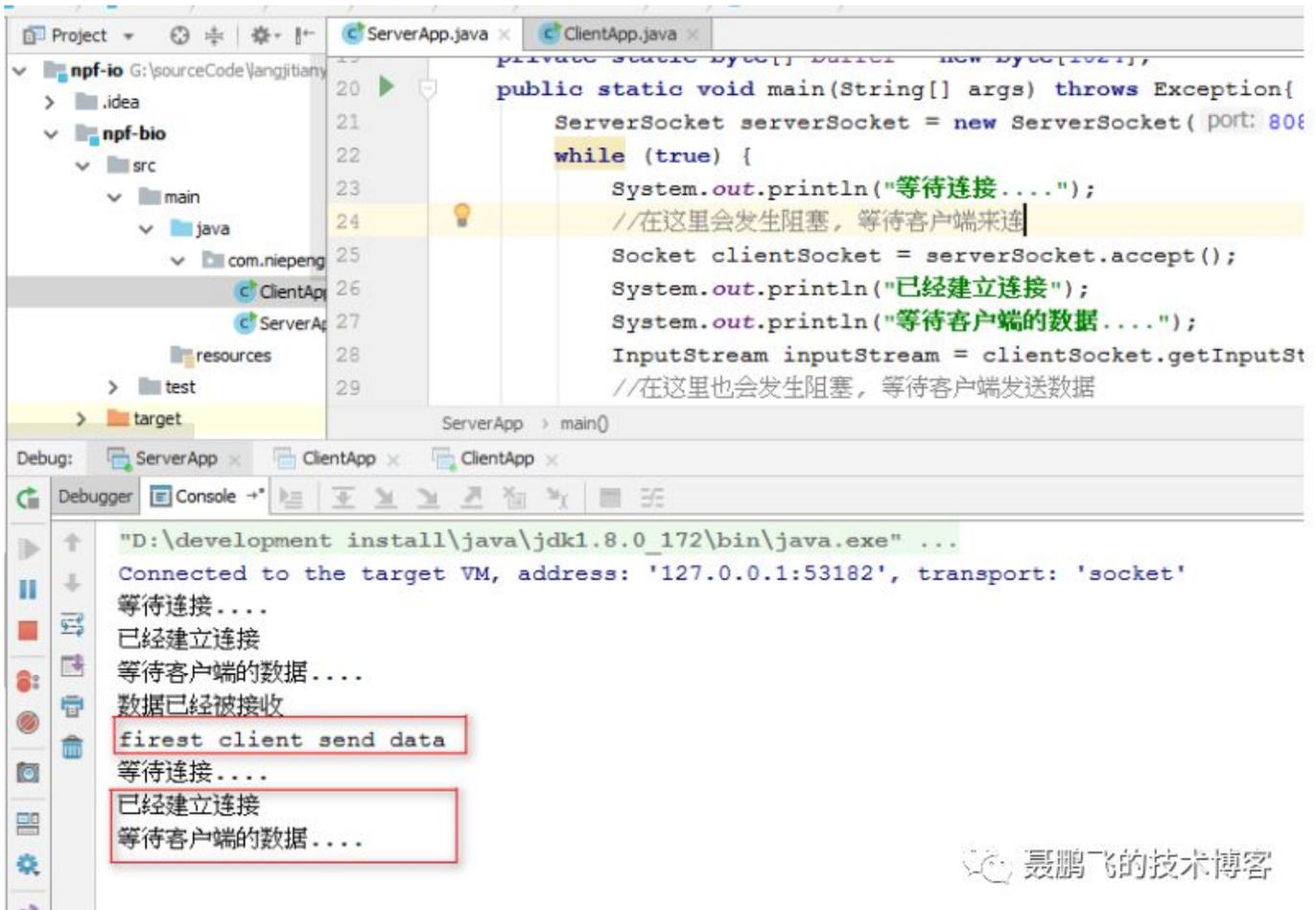
第三步：我们再启动另一个客户端，发现怎么也连接不上服务端。因为服务端没有打印任何信息，目前服务端控制台出现的信息还是第一个客户端连接上来的时候打印的。那么这个服务端相当于瘫痪了，因为没有任何客户端可以连接上来了。直到我们的第一个客户端发数据上来了，第二个客户端才可以连接上，不信你可以继续往下看。



第四步：第一个客户端发送数据给服务端。



这个时候，你看服务端的控制台。第一个客户端的数据已经接收到了，第二个客户端也连接上来了。



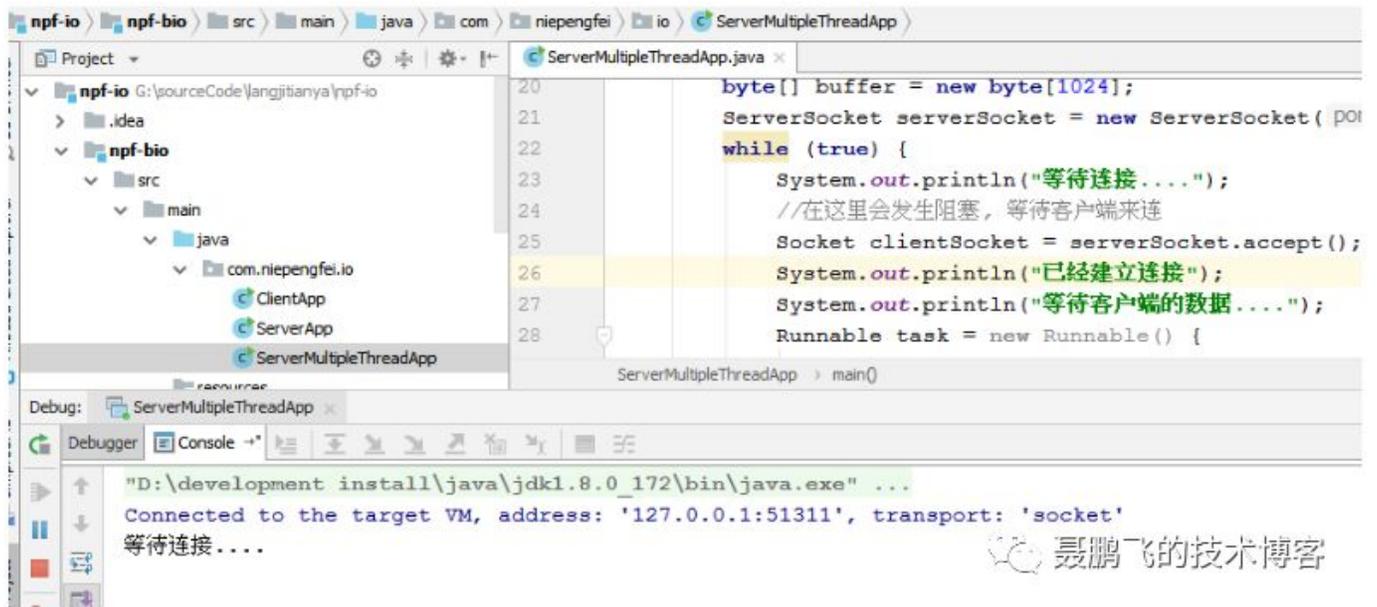
聂鹏飞的技术博客

说白了，BIO在单线程的情况下，是不能实现并发的，因为它在accept和read方法上面阻塞了。那如何解决呢？请看下面。

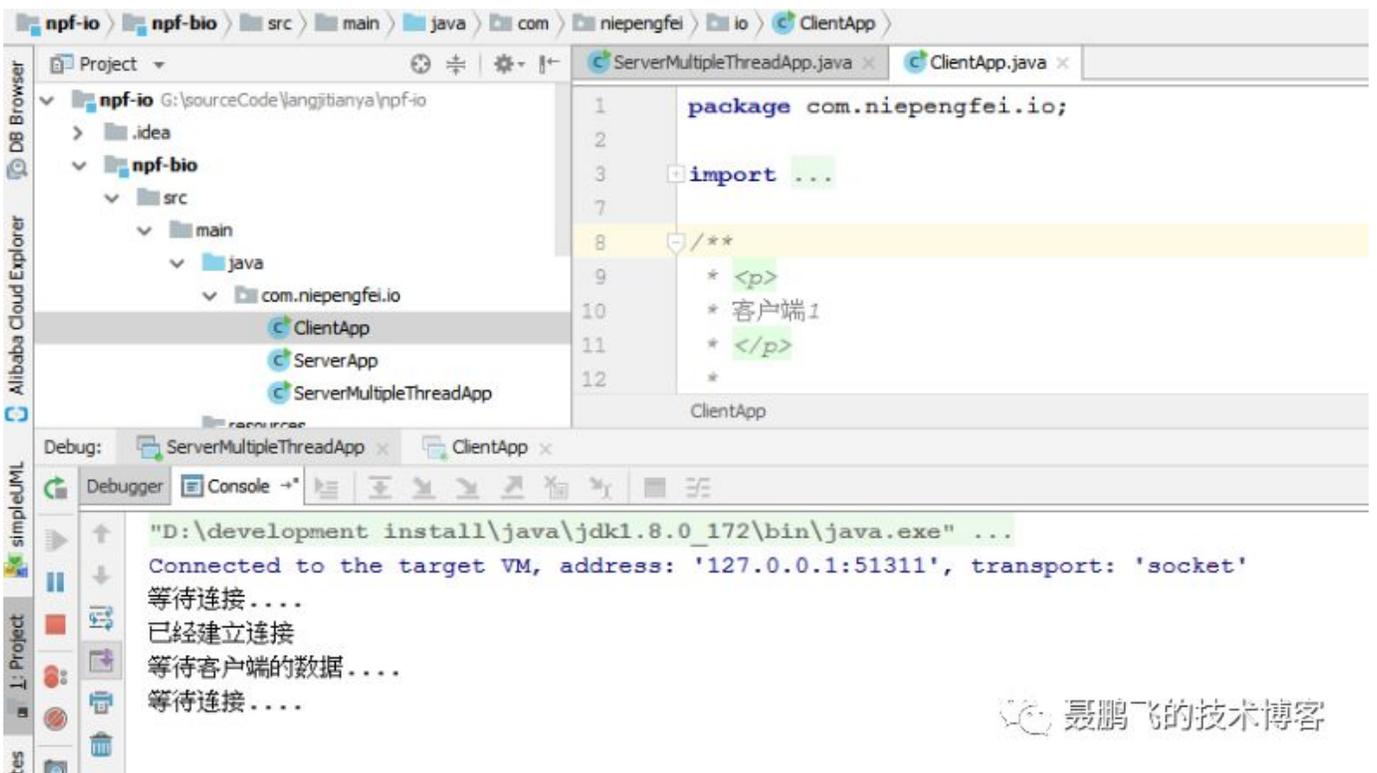
1.4 BIO在多线程情况下可以解决并发问题

采用 **BIO 通信模型** 的服务端，通常由一个独立的 Acceptor 线程负责监听客户端的连接。我们一般通过在 `while(true)` 循环中，服务端会调用 `accept()` 方法等待接收客户端的连接的方式监听请求，一旦接收到一个连接请求，就可以建立通信套接字在这个通信套接字上进行读写操作，此时不能再接收其他客户端连接请求，只能等待同当前连接的客户端的操作执行完成，不过可以通过多线程来支持多个客户端的连接，如下图所示。

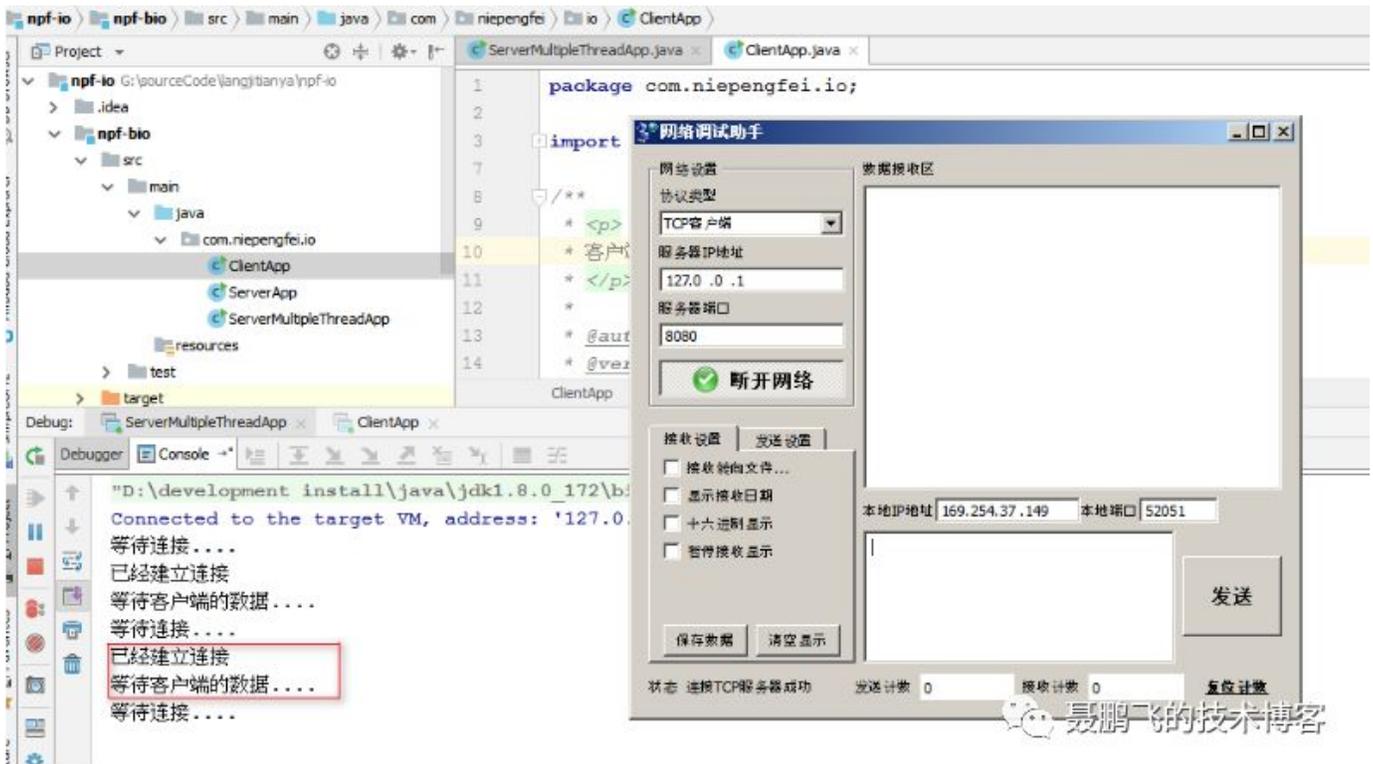
如果要让 **BIO 通信模型** 能够同时处理多个客户端请求，就必须使用多线程（主要原因是 `socket.accept()`、`socket.read()`、`socket.write()` 涉及的三个主要函数都是同步阻塞的），也就是说它在接收到客户端连接请求之后为每个客户端创建一个新的线程进行链路处理，处理完成之后，通过输出流返回应答给客户端，线程销毁。这就是典型的 **一请求一应答通信模型**。



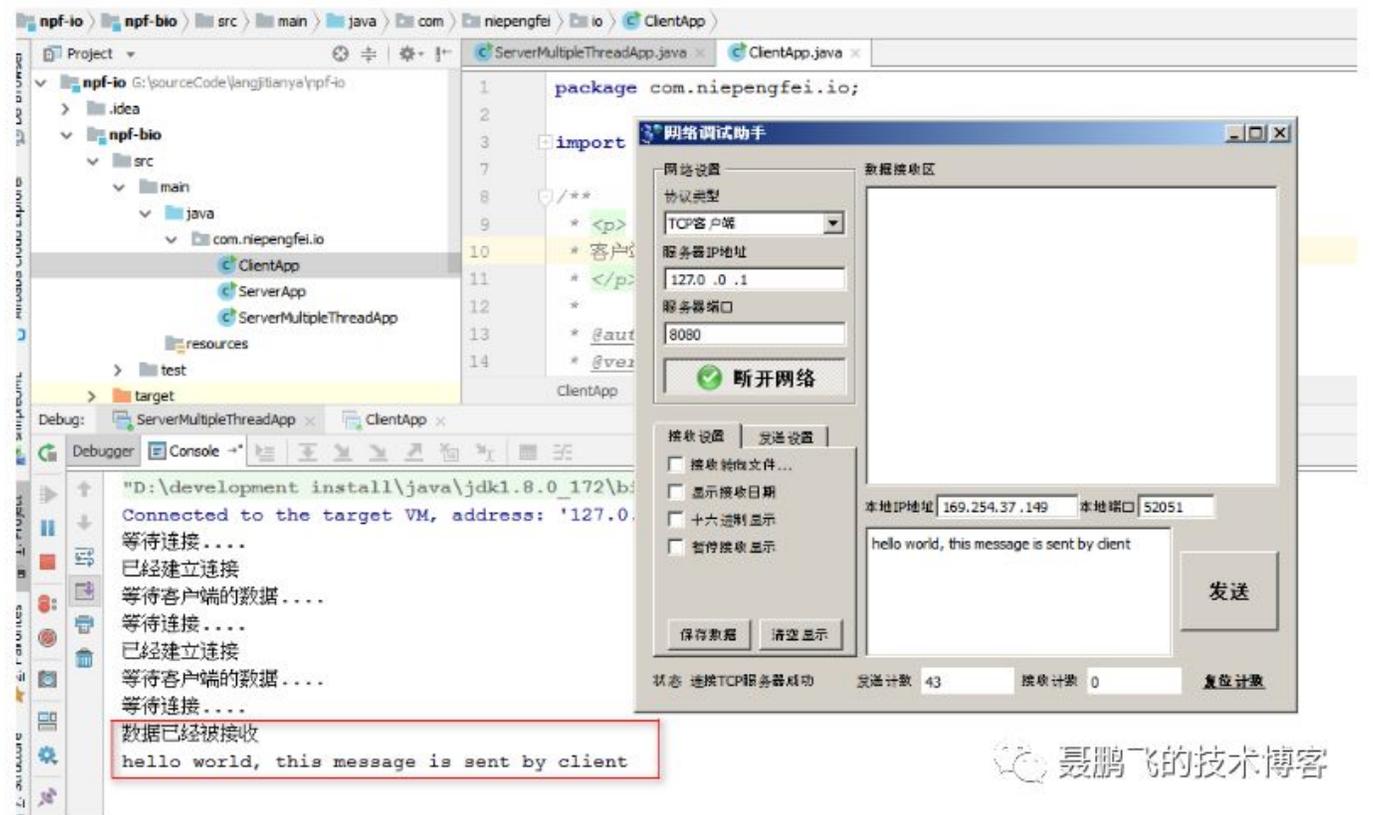
第二步：启动第一个客户端。让这个客户端和服务端建立连接，但是不发送数据。



第三步：让另一个客户端和服务端建立连接。



很明显，这个客户端显然连接上了服务端。此时我们发送数据看看。



从这里可以看出，BIO在多线程情况下，确实是可以解决并发问题的。

1.4.2 存在的问题

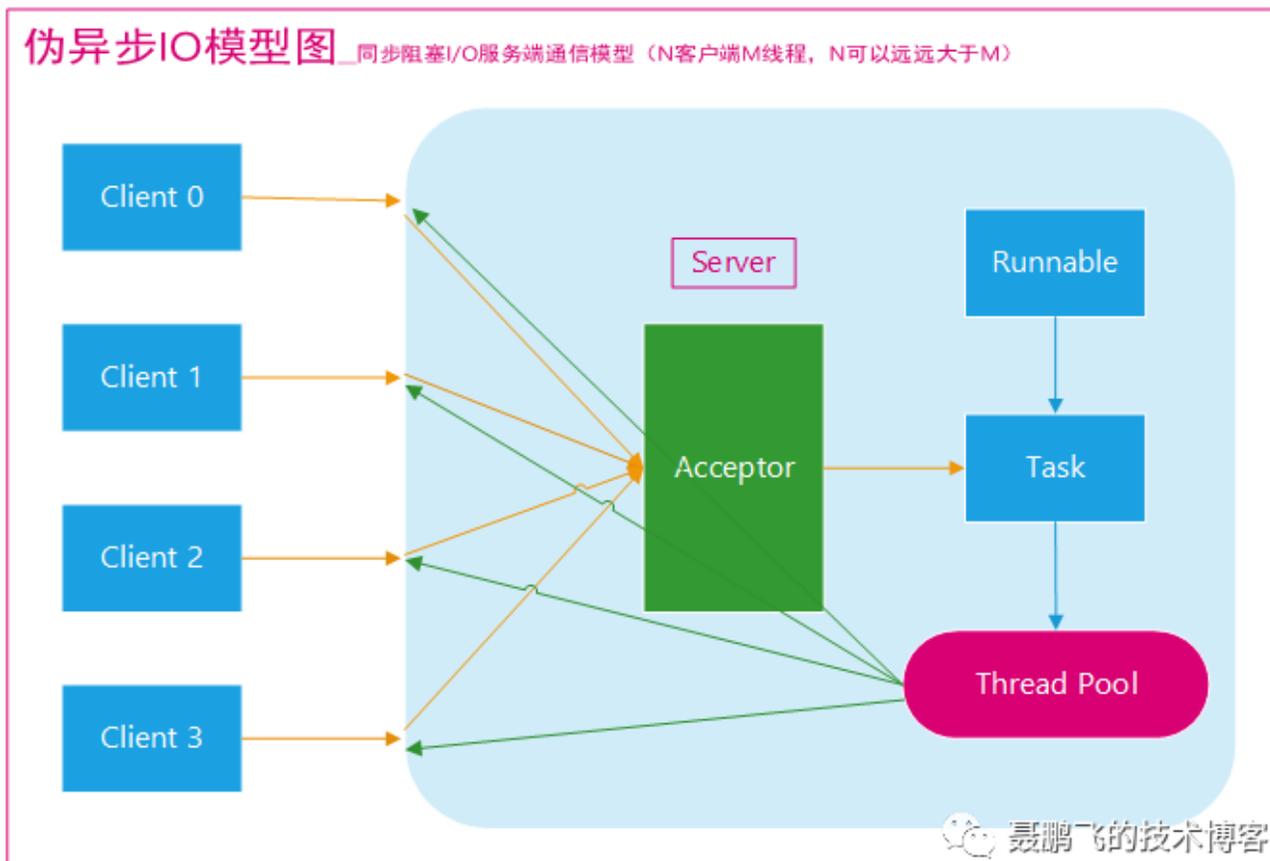
我们可以设想一下，如果连接到服务端的客户端不做任何事情的话就会造成不必要的线程开销，不过我们可以通过 **线程池机制** 改善，线程池还可以让线程的创建和回收成本相对较低。使用 `FixedThreadPool`

可以有效地控制了线程的最大数量，保证了系统有限的资源的控制，实现了N(客户端请求数量):M(处理客户端请求的线程数量)的I/O模型（N 可以远远大于 M），即BIO在多线程情况下可以解决并发问题。

我们再设想一下当客户端并发访问量增加后这种模型会出现什么问题？

在 Java 虚拟机中，线程是宝贵的资源，线程的创建和销毁成本很高，除此之外，线程的切换成本也是很高的。尤其在 Linux 这样的操作系统中，线程本质上就是一个进程，创建和销毁线程都是重量级的系统函数。如果并发访问量增加会导致线程数急剧膨胀可能会导致线程堆栈溢出、创建新线程失败等问题，最终导致进程宕机或者僵死，不能对外提供服务。

为了解决BIO面临的一个客户端请求需要一个线程处理的问题，后来有人对它的线程模型进行了优化——后端通过一个线程池来处理多个客户端的请求接入，形成客户端个数N：线程池最大线程数M的比例关系，其中N可以远远大于M。通过线程池可以灵活地调配线程资源，设置线程的最大值，防止由于海量并发接入导致线程耗尽。



采用线程池和任务队列可以让BIO解决并发问题，它的模型图如上图所示。当有新的客户端接入时，将客户端的 Socket 封装成一个Task（该任务实现java.lang.Runnable接口）投递到后端的线程池中进行处理，JDK 的线程池维护一个消息队列和 N 个活跃线程，对消息队列中的任务进行处理。

由于线程池可以设置消息队列的大小和最大线程数，因此，它的资源占用是可控的，无论多少个客户端并发访问，都不会导致资源的耗尽和宕机。该方案采用了线程池实现，因此避免了为每个请求都创建一个独立线程造成的线程资源耗尽问题。不过因为它的底层仍然是同步阻塞的BIO模型，因此无法从根本上解决问题。在活动连接数不是特别高的情况下，这种模型是比较不错的，可以让每一个连接专注于自己的 I/O 并且编程模型简单，也不用过多考虑系统的过载、限流等问题。线程池本身就是一个天然的漏斗，可以缓冲

一些系统处理不了连接或请求。但是，当面对十万甚至百万级连接的时候，传统的 BIO 模型是无能为力的。因此，我们需要一种更高效的 I/O 处理模型来应对更高的并发量。

2. 理解NIO

2.1 NIO的设计初衷

基于BIO的缺陷，NIO被设计出来，就是为了在单线程情况下，可以解决并发问题。基于前面的分析，BIO之所以在单线程情况下不能解决并发问题的实质是accept和read方法都是阻塞的。那么很简单，NIO就是让这两个方法不阻塞不就可以了嘛。

2.2 代码层面实现

可以看到如下的代码，详细的注释已在代码中说明。例如：

第一步：当client1来连接的时候，accept方法获取连接立即返回，假设此时client1没有发数据过来的话，此时read方法也不会阻塞的。程序继续执行，再次进入while循环。

第二步：当再次进入while循环的时候，此时若还没有其他的客户端来连，那么此时accept方法立即返回null。

第三步：程序继续执行，若此时client1发来了数据，然后我们会发现，服务端已经接收不到数据了，因为我们已经丢失了client1的那个socket连接。那么怎么办呢？

```
/*  
 * @author Jack  
 * @version 1.0.0  
 * @since 1/11/2020  
 */  
public class ServerNioApp {  
  
    private static ByteBuffer buffer = ByteBuffer.allocate(1024);  
    public static void main(String[] args) throws Exception{  
        ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();  
        serverSocketChannel.bind(new InetSocketAddress(port: 8080));  
        //设置服务端等待连接 非阻塞，意思是在调用accept方法时，不会再阻塞  
        //如果有客户端来连，那么服务端就会创建一个socket与当前连接上来的客户端进行通信  
        //如果没有客户端来连，那么该方法立即返回，且返回值是null  
        serverSocketChannel.configureBlocking(false);  
        while (true){  
            SocketChannel clientSocket = serverSocketChannel.accept(); ①  
            //设置读取客户端发送过来的数据的动作是非阻塞的  
            //如果客户端有发送数据过来，那么自然会读取到  
            //如果客户端没有发送数据过来，那么read方法立即返回，不会阻塞  
            clientSocket.configureBlocking(false);  
            int read = clientSocket.read(buffer); ②  
            if (read > 0) {  
                System.out.println(new String(buffer.array()));  
            }  
        }  
    }  
}
```

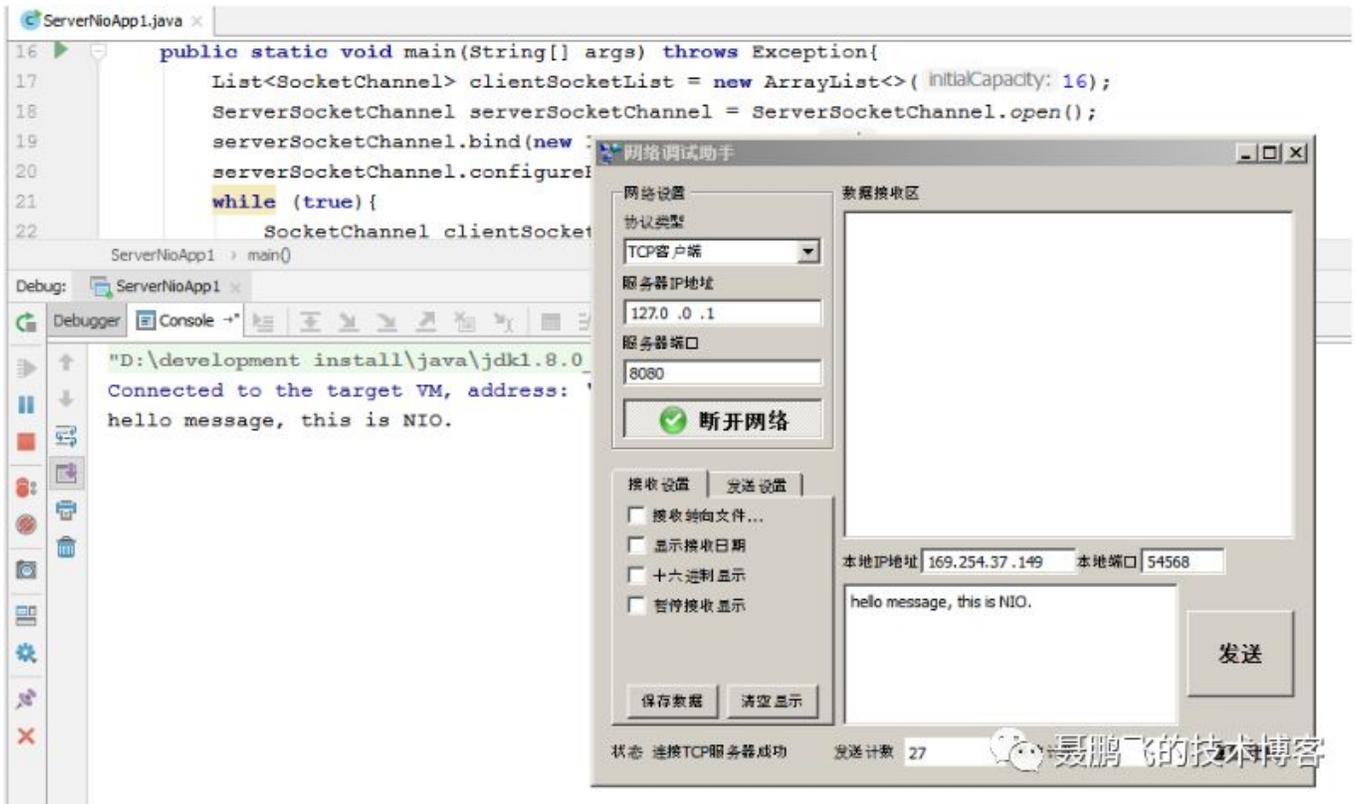
针对上面的问题，我们如何改进呢？那么我们是不是要有一个集合来存这个已经连上来的客户端，是的，我们的分析是没有错的，那么请看下面的代码。

```
ServerNioApp1.java x
16 public static void main(String[] args) throws Exception{
17     List<SocketChannel> clientSocketList = new ArrayList<>( initialCapacity: 16);
18     ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
19     serverSocketChannel.bind(new InetSocketAddress( port: 8080));
20     serverSocketChannel.configureBlocking(false);
21     while (true){
22         SocketChannel clientSocket = serverSocketChannel.accept();
23         if (clientSocket == null) {
24             List<SocketChannel> clientList = new ArrayList<>(clientSocketList);
25             for (SocketChannel socketChannel : clientList) {
26                 ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
27                 int read = socketChannel.read(byteBuffer);
28                 byteBuffer.flip();
29                 if (read > 0) {System.out.println(new String(byteBuffer.array()));}
30                 else if (read < 0){ clientSocketList.remove(socketChannel); }
31             }
32         } else {
33             clientSocket.configureBlocking(false);
34             clientSocketList.add(clientSocket);
35             List<SocketChannel> clientList = new ArrayList<>(clientSocketList);
36             for (SocketChannel socketChannel : clientList) {
37                 ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
38                 int read = socketChannel.read(byteBuffer);
39                 byteBuffer.flip();
40                 if (read > 0) { System.out.println(new String(byteBuffer.array())); }
41                 else if (read < 0){ clientSocketList.remove(socketChannel); }
42             }

```

我们有一个list集合来存储连接到服务端的客户端，如果有人连接上来，那么就放入到集合当中，随后遍历该集合，判断是否有客户端发送数据过来。如果没有人连接上来，那么也会遍历集合，目的是判断之前连接上来的客户端是否有发送数据过来。

我们启动服务端，并在客户端进行测试，如下：



接着，我们再开另外一个客户端，发现也是可以连接上来的，并且第二个客户端发数据到服务端，服务端是可以接收到的，随后第一个客户端再发了一条数据，服务端同样是可以接收到的。



2.3 问题的所在

2.3.1 natvie方法在哪里

根据上面的分析，应用程序需要不断的去循环那个list集合。我们不应该将这个集合放在应用程序中去执行，应该放在操作系统的内核中去执行。那该怎么去解决呢？通过调用操作系统底层的select函数，就是将这个集合交给操作系统去循环。select是操作系统函数，可以实现该功能。那么我问你，Java是怎么调

用操作系统函数呢？很简单，Java可以通过JNI方式调用native方法，而在native方法中去手动的调用操作系统的函数。不信你可以看我的分析。当你写这行代码的时候。

```
ServerSocket serverSocket = new ServerSocket(8080);
```

 聂鹏飞的技术博客

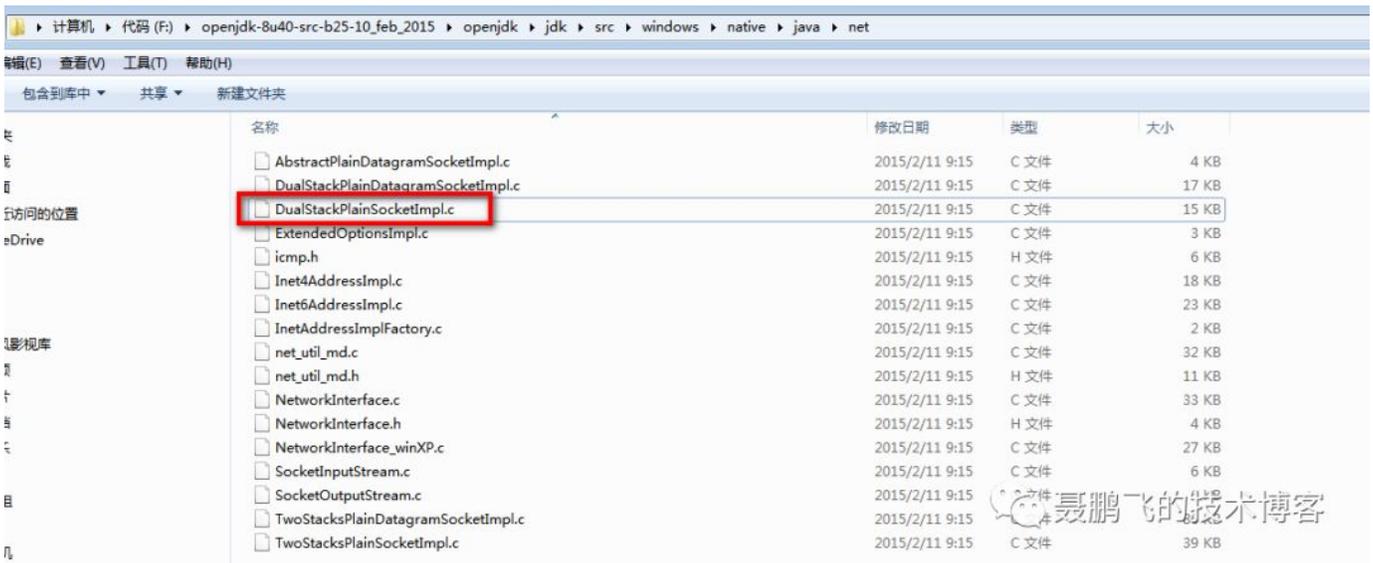
实际底层会调用如下的代码：

```
DualStackPlainSocketImpl.java x
98     }
99     |
100  @ void socketBind(InetAddress address, int port) throws IOException {
101     int nativefd = checkAndReturnNativeFD();
102
103     if (address == null)
104         throw new NullPointerException("inet address argument is null.");
105
106     bind0(nativefd, address, port, exclusiveBind);
107     if (port == 0) {
108         localport = LocalPort0(nativefd);
109     } else {
110         localport = port;
111     }
112
113     this.address = address;
114 }
```

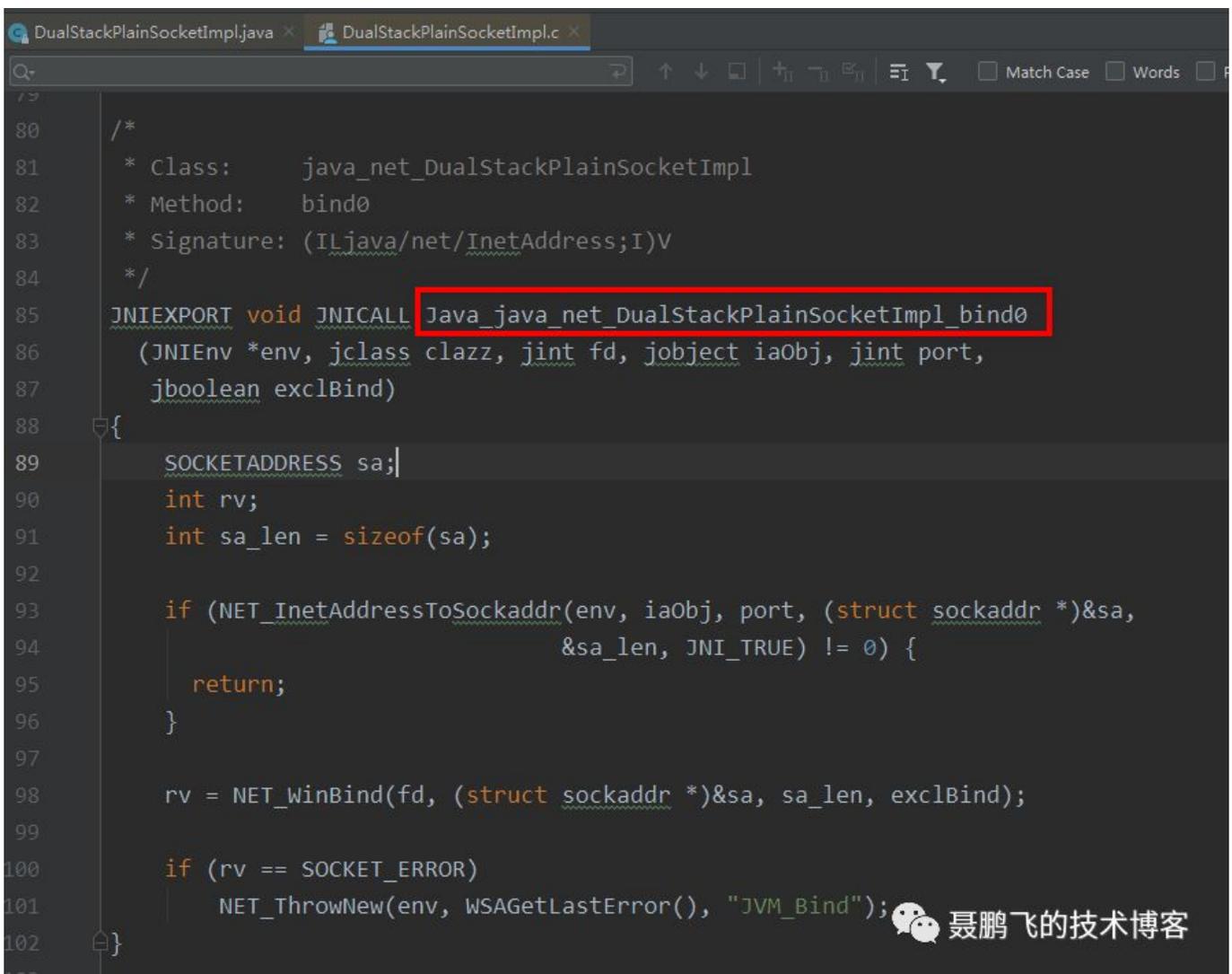
 聂鹏飞的技术博客

即bind0方法，意为本地方法，即c语言实现的方法，那这个方法到底存在哪里呢？即JVM当中，那什么是JVM呢，说白了，就是你装好的那个java.exe程序，那么这个java.exe就是一个编译好的程序，那这个java.exe所属的项目到底是哪个呢？那就是openjdk，源码的下载地址如下<http://hg.openjdk.java.net/>，现在的问题是，我们只要下载好这个openjdk的源码，然后在源码里面去找这个bind0方法就好了。然后去看看bind0方法内部是如何实现的。

windows平台中的实现

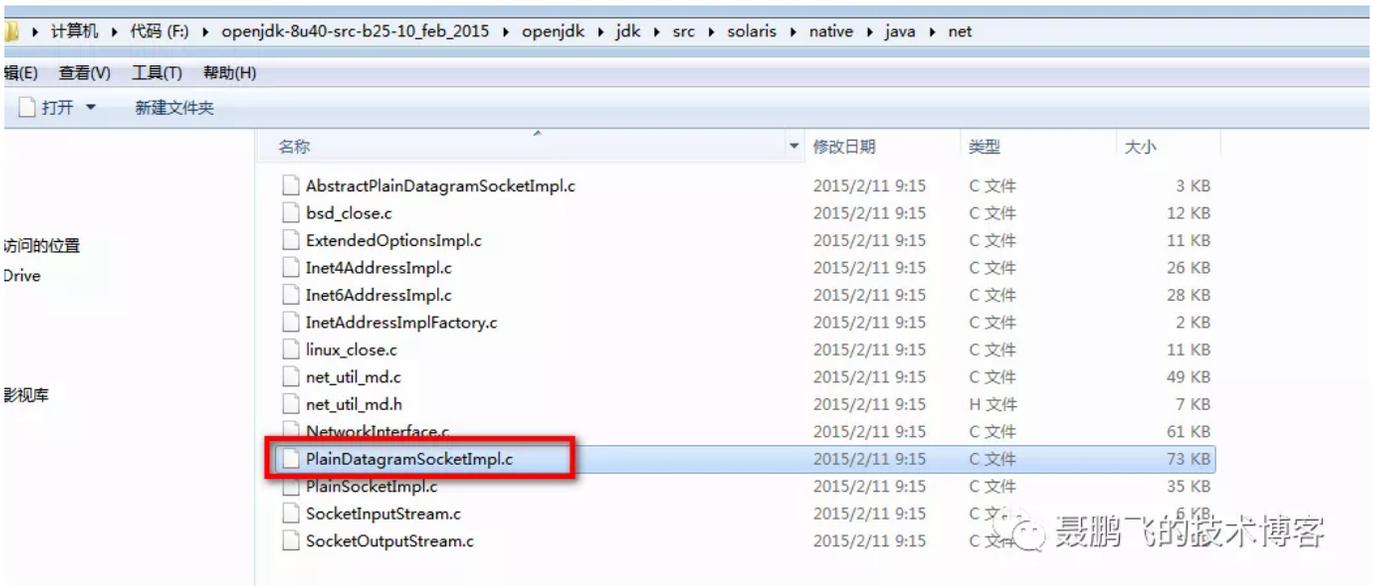


我们打开这个文件，具体查看bind0到底是如何实现的。

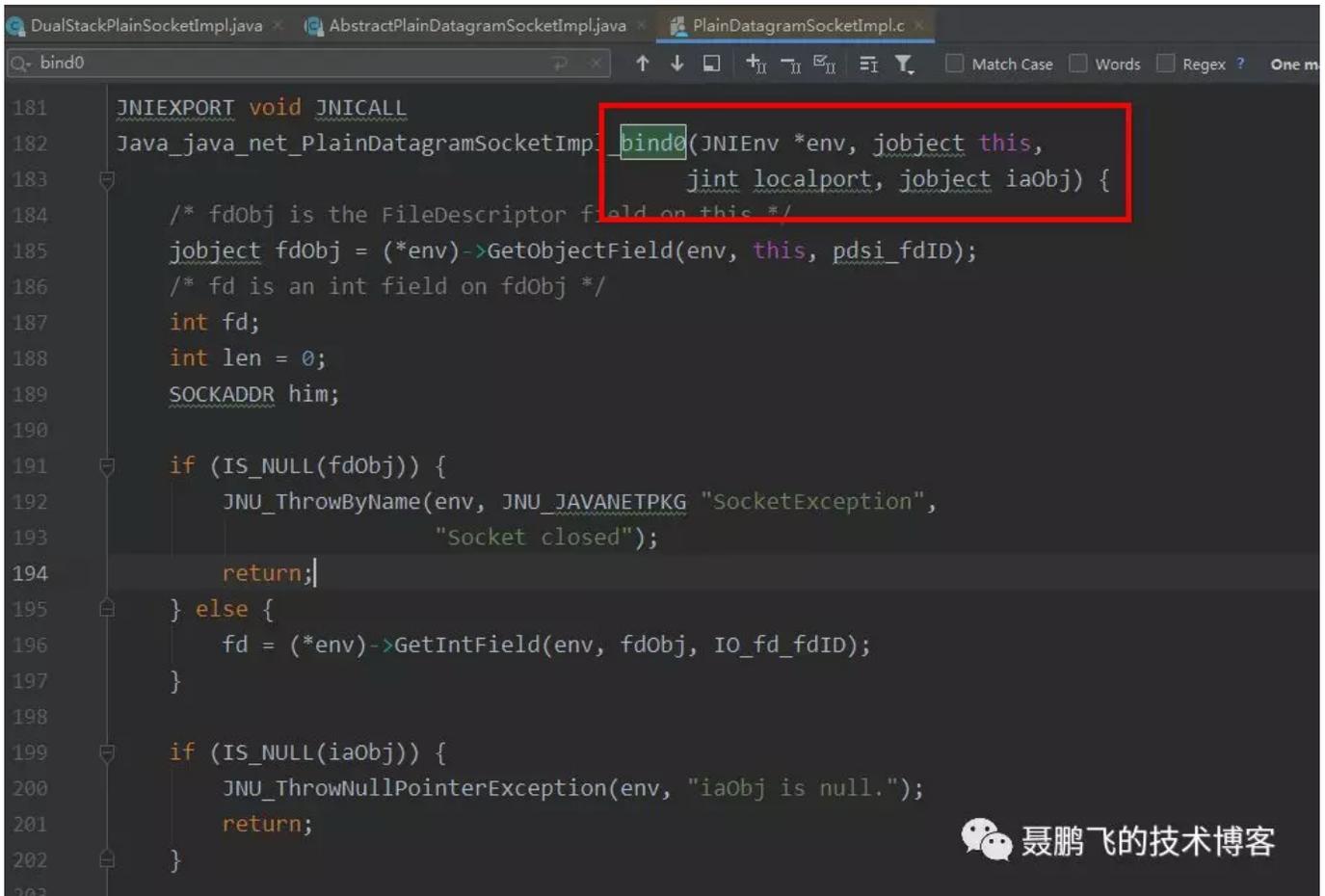


由于我们查看的是windows的bind0的实现，所以在源码中，你可以很清楚的看到调用了windows操作系统的函数。

linux平台中的实现



我们打开这个文件，具体查看bind0到底是如何实现的。



2.3.2 重点分析

windows平台的分析

当你调用这个方法的时候，底层是如何实现的呢？



追踪到源码，底层调用的代码如下：

```
package sun.nio.ch;

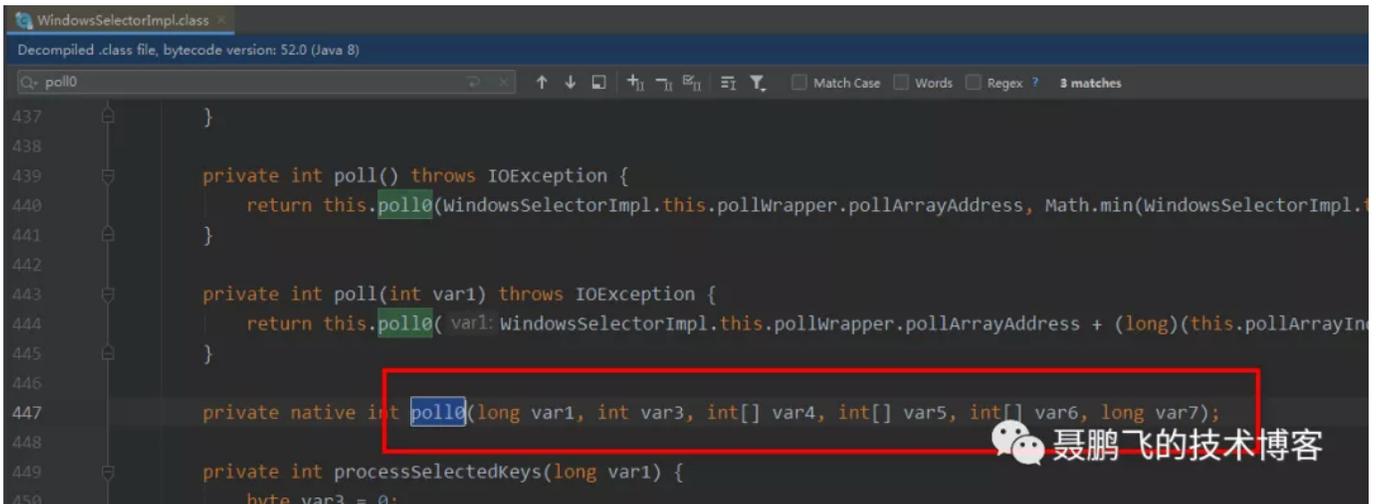
import java.nio.channels.spi.SelectorProvider;

public class DefaultSelectorProvider {
    private DefaultSelectorProvider() {
    }

    public static SelectorProvider create() {
        return new WindowsSelectorProvider();
    }
}
```

聂鹏飞的技术博客

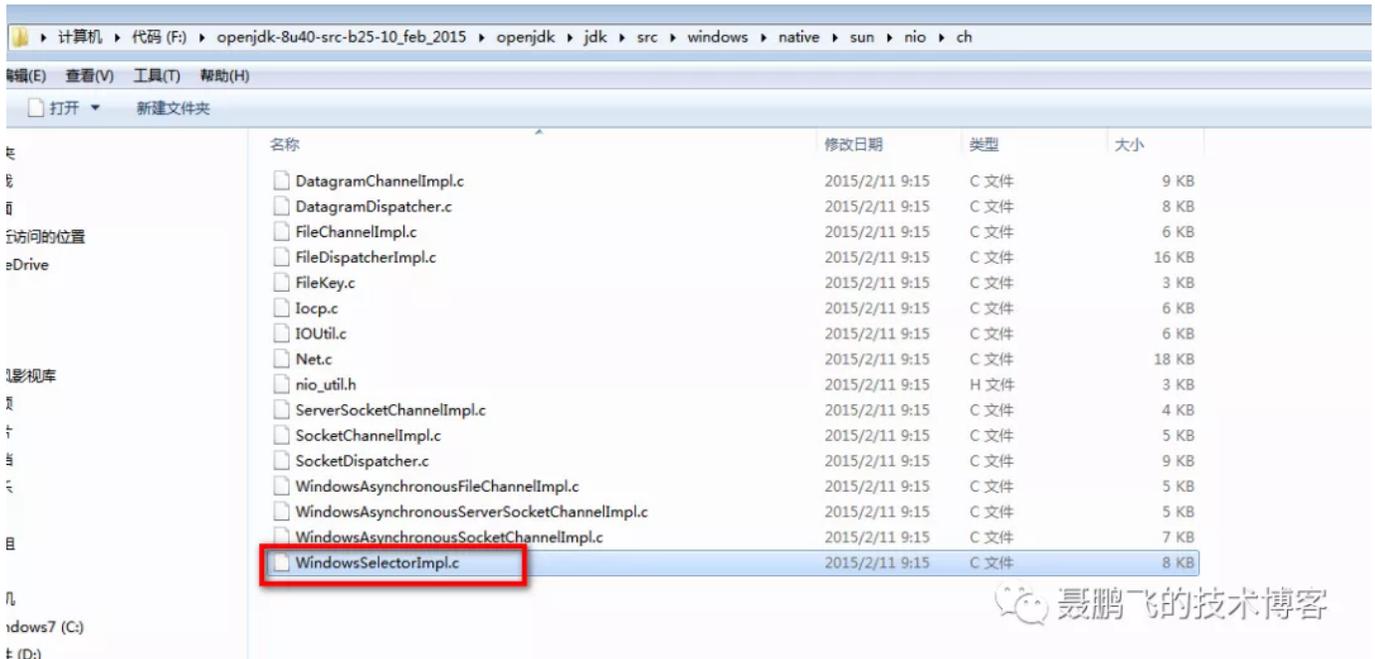
可以发现，在windows平台下，直接就new出来了一个WindowsSelectorProvider对象。分析到这里，你肯定也能猜到，在windows平台下，肯定是这个对象里面调用了native方法。



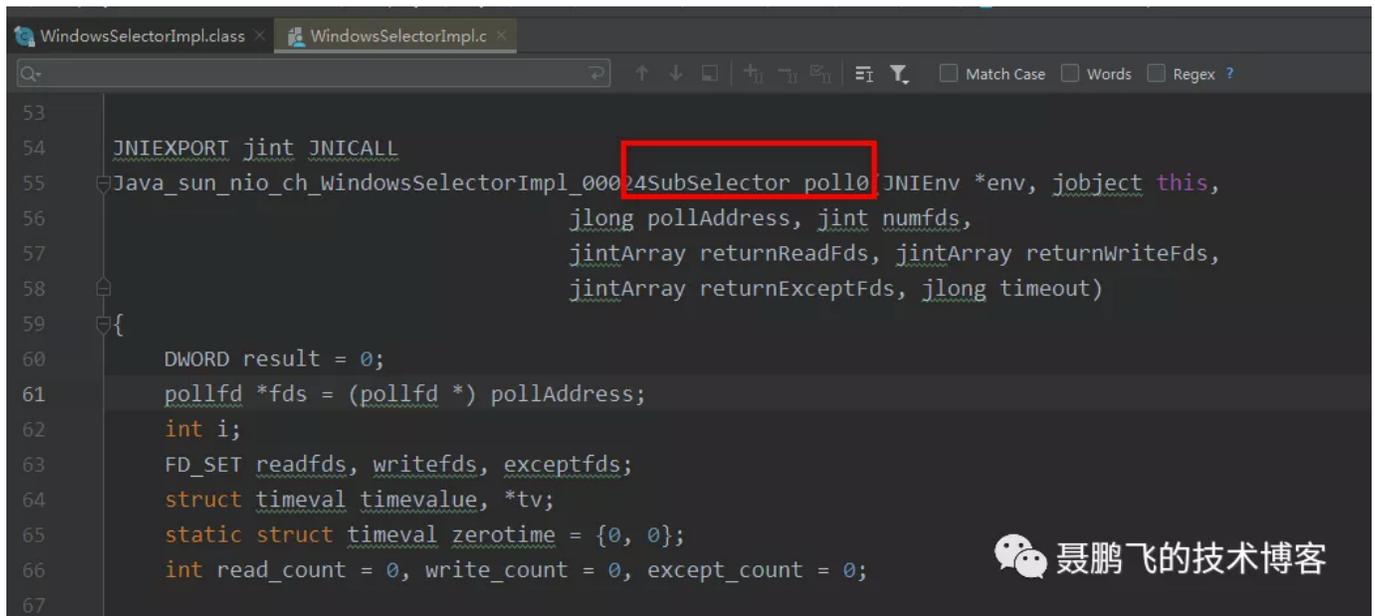
```
WindowsSelectorImpl.class
Decompiled .class file, bytecode version: 52.0 (Java 8)
poll0
437 }
438
439 private int poll() throws IOException {
440     return this.poll0(WindowsSelectorImpl.this.pollWrapper.pollArrayAddress, Math.min(WindowsSelectorImpl.
441 }
442
443 private int poll(int var1) throws IOException {
444     return this.poll0( var1: WindowsSelectorImpl.this.pollWrapper.pollArrayAddress + (long)(this.pollArrayIn
445 }
446
447 private native int poll0(long var1, int var3, int[] var4, int[] var5, int[] var6, long var7);
448
449 private int processSelectedKeys(long var1) {
450     byte var3 = 0;
```

聂鹏飞的技术博客

即在这个poll0方法中，调用了windows平台下的select函数。说白了，在windows底层，就是调用了这个方法，将上述我们一直讨论的那个list集合交给操作系统的。接下来我们具体查看一下该函数的实现。



我们打开这个文件，内容如下：



并且在poll0方法内部，显示的调用了select函数。如下所示：

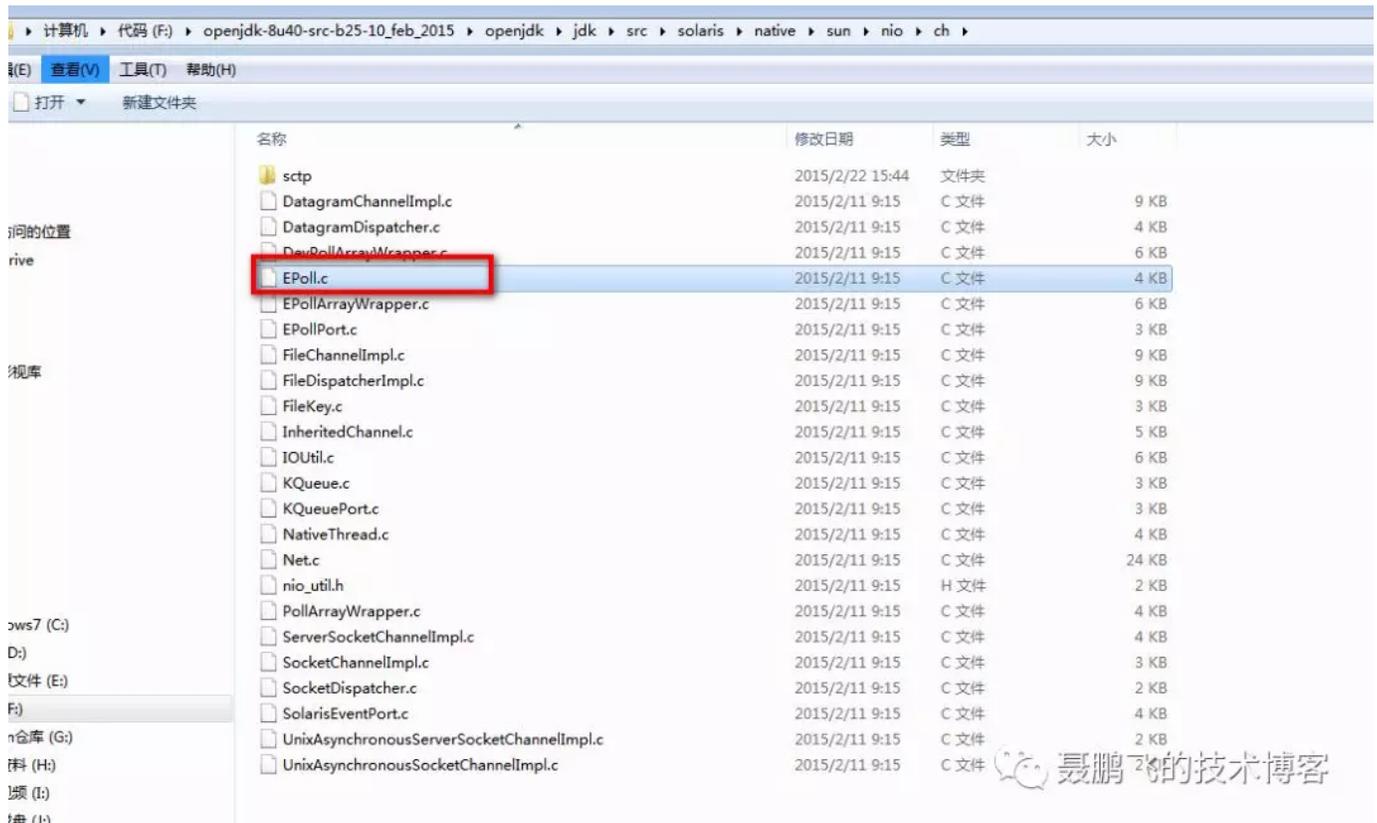
```
WindowsSelectorImpl.class x WindowsSelectorImpl.c x
Q- ↑ ↓ □ ↶ ↷ ↻ ⌵ ⌶ ⌷ ⌸ ⌹ ⌺ ⌻ ⌼ ⌽ ⌾ ⌿ Ⓚ Ⓛ Ⓜ Ⓨ Ⓩ ⓐ ⓑ ⓓ ⓔ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ ⓩ ⓪ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿ Ⓚ Ⓛ Ⓜ Ⓨ Ⓩ ⓐ ⓑ ⓓ ⓔ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ ⓩ ⓪ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿
96
97 readfds.fd_count = read_count;
98 writefds.fd_count = write_count;
99 exceptfds.fd_count = except_count;
100
101 /* Call select */
102 if ((result = select(0 , &readfds, &writefds, &exceptfds, tv))
103     == SOCKET_ERROR) {
104     /* Bad error - this should not happen frequently */
105     /* Iterate over sockets and call select() on each separately */
106     FD_SET errreadfds, errwritefds, errexceptfds;
107     readfds.fd_count = 0;
108     writefds.fd_count = 0;
109     exceptfds.fd_count = 0;
110     for (i = 0; i < numfds; i++) {
111         /* prepare select structures for the i-th socket */
112         errreadfds.fd_count = 0;
113         errwritefds.fd_count = 0;
114         errexceptfds.fd_count = 0;
115     }
116     /* Call select again */
117     result = select(0 , &readfds, &writefds, &exceptfds, tv);
118     if (result == SOCKET_ERROR) {
119         /* Bad error - this should not happen frequently */
120         /* Iterate over sockets and call select() on each separately */
121         FD_SET errreadfds, errwritefds, errexceptfds;
122         readfds.fd_count = 0;
123         writefds.fd_count = 0;
124         exceptfds.fd_count = 0;
125         for (i = 0; i < numfds; i++) {
126             /* prepare select structures for the i-th socket */
127             errreadfds.fd_count = 0;
128             errwritefds.fd_count = 0;
129             errexceptfds.fd_count = 0;
130         }
131         /* Call select again */
132         result = select(0 , &readfds, &writefds, &exceptfds, tv);
133     }
134 }
135
136 return result;
137 }
138
139 #endif
140
141 #endif
142
143 #endif
144
145 #endif
146
147 #endif
148
149 #endif
150
151 #endif
152
153 #endif
154
155 #endif
156
157 #endif
158
159 #endif
160
161 #endif
162
163 #endif
164
165 #endif
166
167 #endif
168
169 #endif
170
171 #endif
172
173 #endif
174
175 #endif
176
177 #endif
178
179 #endif
180
181 #endif
182
183 #endif
184
185 #endif
186
187 #endif
188
189 #endif
190
191 #endif
192
193 #endif
194
195 #endif
196
197 #endif
198
199 #endif
200
201 #endif
202
203 #endif
204
205 #endif
206
207 #endif
208
209 #endif
210
211 #endif
212
213 #endif
214
215 #endif
216
217 #endif
218
219 #endif
220
221 #endif
222
223 #endif
224
225 #endif
226
227 #endif
228
229 #endif
230
231 #endif
232
233 #endif
234
235 #endif
236
237 #endif
238
239 #endif
240
241 #endif
242
243 #endif
244
245 #endif
246
247 #endif
248
249 #endif
250
251 #endif
252
253 #endif
254
255 #endif
256
257 #endif
258
259 #endif
260
261 #endif
262
263 #endif
264
265 #endif
266
267 #endif
268
269 #endif
270
271 #endif
272
273 #endif
274
275 #endif
276
277 #endif
278
279 #endif
280
281 #endif
282
283 #endif
284
285 #endif
286
287 #endif
288
289 #endif
290
291 #endif
292
293 #endif
294
295 #endif
296
297 #endif
298
299 #endif
300
301 #endif
302
303 #endif
304
305 #endif
306
307 #endif
308
309 #endif
310
311 #endif
312
313 #endif
314
315 #endif
316
317 #endif
318
319 #endif
320
321 #endif
322
323 #endif
324
325 #endif
326
327 #endif
328
329 #endif
330
331 #endif
332
333 #endif
334
335 #endif
336
337 #endif
338
339 #endif
340
341 #endif
342
343 #endif
344
345 #endif
346
347 #endif
348
349 #endif
350
351 #endif
352
353 #endif
354
355 #endif
356
357 #endif
358
359 #endif
360
361 #endif
362
363 #endif
364
365 #endif
366
367 #endif
368
369 #endif
370
371 #endif
372
373 #endif
374
375 #endif
376
377 #endif
378
379 #endif
380
381 #endif
382
383 #endif
384
385 #endif
386
387 #endif
388
389 #endif
390
391 #endif
392
393 #endif
394
395 #endif
396
397 #endif
398
399 #endif
400
401 #endif
402
403 #endif
404
405 #endif
406
407 #endif
408
409 #endif
410
411 #endif
412
413 #endif
414
415 #endif
416
417 #endif
418
419 #endif
420
421 #endif
422
423 #endif
424
425 #endif
426
427 #endif
428
429 #endif
430
431 #endif
432
433 #endif
434
435 #endif
436
437 #endif
438
439 #endif
440
441 #endif
442
443 #endif
444
445 #endif
446
447 #endif
448
449 #endif
450
451 #endif
452
453 #endif
454
455 #endif
456
457 #endif
458
459 #endif
460
461 #endif
462
463 #endif
464
465 #endif
466
467 #endif
468
469 #endif
470
471 #endif
472
473 #endif
474
475 #endif
476
477 #endif
478
479 #endif
480
481 #endif
482
483 #endif
484
485 #endif
486
487 #endif
488
489 #endif
490
491 #endif
492
493 #endif
494
495 #endif
496
497 #endif
498
499 #endif
500
501 #endif
502
503 #endif
504
505 #endif
506
507 #endif
508
509 #endif
510
511 #endif
512
513 #endif
514
515 #endif
516
517 #endif
518
519 #endif
520
521 #endif
522
523 #endif
524
525 #endif
526
527 #endif
528
529 #endif
530
531 #endif
532
533 #endif
534
535 #endif
536
537 #endif
538
539 #endif
540
541 #endif
542
543 #endif
544
545 #endif
546
547 #endif
548
549 #endif
550
551 #endif
552
553 #endif
554
555 #endif
556
557 #endif
558
559 #endif
560
561 #endif
562
563 #endif
564
565 #endif
566
567 #endif
568
569 #endif
570
571 #endif
572
573 #endif
574
575 #endif
576
577 #endif
578
579 #endif
580
581 #endif
582
583 #endif
584
585 #endif
586
587 #endif
588
589 #endif
590
591 #endif
592
593 #endif
594
595 #endif
596
597 #endif
598
599 #endif
600
601 #endif
602
603 #endif
604
605 #endif
606
607 #endif
608
609 #endif
610
611 #endif
612
613 #endif
614
615 #endif
616
617 #endif
618
619 #endif
620
621 #endif
622
623 #endif
624
625 #endif
626
627 #endif
628
629 #endif
630
631 #endif
632
633 #endif
634
635 #endif
636
637 #endif
638
639 #endif
640
641 #endif
642
643 #endif
644
645 #endif
646
647 #endif
648
649 #endif
650
651 #endif
652
653 #endif
654
655 #endif
656
657 #endif
658
659 #endif
660
661 #endif
662
663 #endif
664
665 #endif
666
667 #endif
668
669 #endif
670
671 #endif
672
673 #endif
674
675 #endif
676
677 #endif
678
679 #endif
680
681 #endif
682
683 #endif
684
685 #endif
686
687 #endif
688
689 #endif
690
691 #endif
692
693 #endif
694
695 #endif
696
697 #endif
698
699 #endif
700
701 #endif
702
703 #endif
704
705 #endif
706
707 #endif
708
709 #endif
710
711 #endif
712
713 #endif
714
715 #endif
716
717 #endif
718
719 #endif
720
721 #endif
722
723 #endif
724
725 #endif
726
727 #endif
728
729 #endif
730
731 #endif
732
733 #endif
734
735 #endif
736
737 #endif
738
739 #endif
740
741 #endif
742
743 #endif
744
745 #endif
746
747 #endif
748
749 #endif
750
751 #endif
752
753 #endif
754
755 #endif
756
757 #endif
758
759 #endif
760
761 #endif
762
763 #endif
764
765 #endif
766
767 #endif
768
769 #endif
770
771 #endif
772
773 #endif
774
775 #endif
776
777 #endif
778
779 #endif
780
781 #endif
782
783 #endif
784
785 #endif
786
787 #endif
788
789 #endif
790
791 #endif
792
793 #endif
794
795 #endif
796
797 #endif
798
799 #endif
800
801 #endif
802
803 #endif
804
805 #endif
806
807 #endif
808
809 #endif
810
811 #endif
812
813 #endif
814
815 #endif
816
817 #endif
818
819 #endif
820
821 #endif
822
823 #endif
824
825 #endif
826
827 #endif
828
829 #endif
830
831 #endif
832
833 #endif
834
835 #endif
836
837 #endif
838
839 #endif
840
841 #endif
842
843 #endif
844
845 #endif
846
847 #endif
848
849 #endif
850
851 #endif
852
853 #endif
854
855 #endif
856
857 #endif
858
859 #endif
860
861 #endif
862
863 #endif
864
865 #endif
866
867 #endif
868
869 #endif
870
871 #endif
872
873 #endif
874
875 #endif
876
877 #endif
878
879 #endif
880
881 #endif
882
883 #endif
884
885 #endif
886
887 #endif
888
889 #endif
890
891 #endif
892
893 #endif
894
895 #endif
896
897 #endif
898
899 #endif
900
901 #endif
902
903 #endif
904
905 #endif
906
907 #endif
908
909 #endif
910
911 #endif
912
913 #endif
914
915 #endif
916
917 #endif
918
919 #endif
920
921 #endif
922
923 #endif
924
925 #endif
926
927 #endif
928
929 #endif
930
931 #endif
932
933 #endif
934
935 #endif
936
937 #endif
938
939 #endif
940
941 #endif
942
943 #endif
944
945 #endif
946
947 #endif
948
949 #endif
950
951 #endif
952
953 #endif
954
955 #endif
956
957 #endif
958
959 #endif
960
961 #endif
962
963 #endif
964
965 #endif
966
967 #endif
968
969 #endif
970
971 #endif
972
973 #endif
974
975 #endif
976
977 #endif
978
979 #endif
980
981 #endif
982
983 #endif
984
985 #endif
986
987 #endif
988
989 #endif
990
991 #endif
992
993 #endif
994
995 #endif
996
997 #endif
998
999 #endif
1000
1001 #endif
1002
1003 #endif
1004
1005 #endif
1006
1007 #endif
1008
1009 #endif
1010
1011 #endif
1012
1013 #endif
1014
1015 #endif
1016
1017 #endif
1018
1019 #endif
1020
1021 #endif
1022
1023 #endif
1024
1025 #endif
1026
1027 #endif
1028
1029 #endif
1030
1031 #endif
1032
1033 #endif
1034
1035 #endif
1036
1037 #endif
1038
1039 #endif
1040
1041 #endif
1042
1043 #endif
1044
1045 #endif
1046
1047 #endif
1048
1049 #endif
1050
1051 #endif
1052
1053 #endif
1054
1055 #endif
1056
1057 #endif
1058
1059 #endif
1060
1061 #endif
1062
1063 #endif
1064
1065 #endif
1066
1067 #endif
1068
1069 #endif
1070
1071 #endif
1072
1073 #endif
1074
1075 #endif
1076
1077 #endif
1078
1079 #endif
1080
1081 #endif
1082
1083 #endif
1084
1085 #endif
1086
1087 #endif
1088
1089 #endif
1090
1091 #endif
1092
1093 #endif
1094
1095 #endif
1096
1097 #endif
1098
1099 #endif
1100
1101 #endif
1102
1103 #endif
1104
1105 #endif
1106
1107 #endif
1108
1109 #endif
1110
1111 #endif
1112
1113 #endif
1114
1115 #endif
1116
1117 #endif
1118
1119 #endif
1120
1121 #endif
1122
1123 #endif
1124
1125 #endif
1126
1127 #endif
1128
1129 #endif
1130
1131 #endif
1132
1133 #endif
1134
1135 #endif
1136
1137 #endif
1138
1139 #endif
1140
1141 #endif
1142
1143 #endif
1144
1145 #endif
1146
1147 #endif
1148
1149 #endif
1150
1151 #endif
1152
1153 #endif
1154
1155 #endif
1156
1157 #endif
1158
1159 #endif
1160
1161 #endif
1162
1163 #endif
1164
1165 #endif
1166
1167 #endif
1168
1169 #endif
1170
1171 #endif
1172
1173 #endif
1174
1175 #endif
1176
1177 #endif
1178
1179 #endif
1180
1181 #endif
1182
1183 #endif
1184
1185 #endif
1186
1187 #endif
1188
1189 #endif
1190
1191 #endif
1192
1193 #endif
1194
1195 #endif
1196
1197 #endif
1198
1199 #endif
1200
1201 #endif
1202
1203 #endif
1204
1205 #endif
1206
1207 #endif
1208
1209 #endif
1210
1211 #endif
1212
1213 #endif
1214
1215 #endif
1216
1217 #endif
1218
1219 #endif
1220
1221 #endif
1222
1223 #endif
1224
1225 #endif
1226
1227 #endif
1228
1229 #endif
1230
1231 #endif
1232
1233 #endif
1234
1235 #endif
1236
1237 #endif
1238
1239 #endif
1240
1241 #endif
1242
1243 #endif
1244
1245 #endif
1246
1247 #endif
1248
1249 #endif
1250
1251 #endif
1252
1253 #endif
1254
1255 #endif
1256
1257 #endif
1258
1259 #endif
1260
1261 #endif
1262
1263 #endif
1264
1265 #endif
1266
1267 #endif
1268
1269 #endif
1270
1271 #endif
1272
1273 #endif
1274
1275 #endif
1276
1277 #endif
1278
1279 #endif
1280
1281 #endif
1282
1283 #endif
1284
1285 #endif
1286
1287 #endif
1288
1289 #endif
1290
1291 #endif
1292
1293 #endif
1294
1295 #endif
1296
1297 #endif
1298
1299 #endif
1300
1301 #endif
1302
1303 #endif
1304
1305 #endif
1306
1307 #endif
1308
1309 #endif
1310
1311 #endif
1312
1313 #endif
1314
1315 #endif
1316
1317 #endif
1318
1319 #endif
1320
1321 #endif
1322
1323 #endif
1324
1325 #endif
1326
1327 #endif
1328
1329 #endif
1330
1331 #endif
1332
1333 #endif
1334
1335 #endif
1336
1337 #endif
1338
1339 #endif
1340
1341 #endif
1342
1343 #endif
1344
1345 #endif
1346
1347 #endif
1348
1349 #endif
1350
1351 #endif
1352
1353 #endif
1354
1355 #endif
1356
1357 #endif
1358
1359 #endif
1360
1361 #endif
1362
1363 #endif
1364
1365 #endif
1366
1367 #endif
1368
1369 #endif
1370
1371 #endif
1372
1373 #endif
1374
1375 #endif
1376
1377 #endif
1378
1379 #endif
1380
1381 #endif
1382
1383 #endif
1384
1385 #endif
1386
1387 #endif
1388
1389 #endif
1390
1391 #endif
1392
1393 #endif
1394
1395 #endif
1396
1397 #endif
1398
1399 #endif
1400
1401 #endif
1402
1403 #endif
1404
1405 #endif
1406
1407 #endif
1408
1409 #endif
1410
1411 #endif
1412
1413 #endif
1414
1415 #endif
1416
1417 #endif
1418
1419 #endif
1420
1421 #endif
1422
1423 #endif
1424
1425 #endif
1426
1427 #endif
1428
1429 #endif
1430
1431 #endif
1432
1433 #endif
1434
1435 #endif
1436
1437 #endif
1438
1439 #endif
1440
1441 #endif
1442
1443 #endif
1444
1445 #endif
1446
1447 #endif
1448
1449 #endif
1450
1451 #endif
1452
1453 #endif
1454
1455 #endif
1456
1457 #endif
1458
1459 #endif
1460
1461 #endif
1462
1463 #endif
1464
1465 #endif
1466
1467 #endif
1468
1469 #endif
1470
1471 #endif
1472
1473 #endif
1474
1475 #endif
1476
1477 #endif
1478
1479 #endif
1480
1481 #endif
1482
1483 #endif
1484
1485 #endif
1486
1487 #endif
1488
1489 #endif
1490
1491 #endif
1492
1493 #endif
1494
1495 #endif
1496
1497 #endif
1498
1499 #endif
1500
1501 #endif
1502
1503 #endif
1504
1505 #endif
1506
1507 #endif
1508
1509 #endif
1510
1511 #endif
1512
1513 #endif
1514
1515 #endif
1516
1517 #endif
1518
1519 #endif
1520
1521 #endif
1522
1523 #endif
1524
1525 #endif
1526
1527 #endif
1528
1529 #endif
1530
1531 #endif
1532
1533 #endif
1534
1535 #endif
1536
1537 #endif
1538
1539 #endif
1540
1541 #endif
1542
1543 #endif
1544
1545 #endif
1546
1547 #endif
1548
1549 #endif
1550
1551 #endif
1552
1553 #endif
1554
1555 #endif
1556
1557 #endif
1558
1559 #endif
1560
1561 #endif
1562
1563 #endif
1564
1565 #endif
1566
1567 #endif
1568
1569 #endif
1570
1571 #endif
1572
1573 #endif
1574
1575 #endif
1576
1577 #endif
1578
1579 #endif
1580
1581 #endif
1582
1583 #endif
1584
1585 #endif
1586
1587 #endif
1588
1589 #endif
1590
1591 #endif
1592
1593 #endif
1594
1595 #endif
1596
1597 #endif
1598
1599 #endif
1600
1601 #endif
1602
1603 #endif
1604
1605 #endif
1606
1607 #endif
1608
1609 #endif
1610
1611 #endif
1612
1613 #endif
1614
1615 #endif
1616
1617 #endif
1618
1619 #endif
1620
1621 #endif
1622
1623 #endif
1624
1625 #endif
1626
1627 #endif
1628
1629 #endif
1630
1631 #endif
1632
1633 #endif
1634
1635 #endif
1636
1637 #endif
1638
1639 #endif
1640
1641 #endif
1642
1643 #endif
1644
1645 #endif
1646
1647 #endif
1648
1649 #endif
1650
1651 #endif
1652
1653 #endif
1654
1655 #endif
1656
1657 #endif
1658
1659 #endif
1660
1661 #endif
1662
1663 #endif
1664
1665 #endif
1666
1667 #endif
1668
1669 #endif
1670
1671 #endif
1672
1673 #endif
1674
1675 #endif
1676
1677 #endif
1678
1679 #endif
1680
1681 #endif
1682
1683 #endif
1684
1685 #endif
1686
1687 #endif
1688
1689 #endif
1690
1691 #endif
1692
1693 #endif
1694
1695 #endif
1696
1697 #endif
1698
1699 #endif
1700
1701 #endif
1702
1703 #endif
1704
1705 #endif
1706
1707 #endif
1708
1709 #endif
1710
1711 #endif
1712
1713 #endif
1714
1715 #endif
1716
1717 #endif
1718
1719 #endif
1720
1721 #endif
1722
1723 #endif
1724
1725 #endif
1726
1727 #endif
1728
1729 #endif
1730
1731 #endif
1732
1733 #endif
1734
1735 #endif
1736
1737 #endif
1738
1739 #endif
1740
1741 #endif
1742
1743 #endif
1744
1745 #endif
1746
1747 #endif
1748
1749 #endif
1750
1751 #endif
1752
1753 #endif
1754
1755 #endif
1756
1757 #endif
1758
1759 #endif
1760
1761 #endif
1762
1763 #endif
1764
1765 #endif
1766
1767 #endif
1768
1769 #endif
1770
1771 #endif
1772
1773 #endif
1774
1775 #endif
1776
1777 #endif
1778
1779 #endif
1780
1781 #endif
1782
1783 #endif
1784
1785 #endif
1786
1787 #endif
1788
1789 #endif
1790
1791 #endif
1792
1793 #endif
1794
1795 #endif
1796
1797 #endif
1798
1799 #endif
1800
1801 #endif
1802
1803 #endif
1804
1805 #endif
1806
1807 #endif
1808
1809 #endif
1810
1811 #endif
1812
1813 #endif
1814
1815 #endif
1816
1817 #endif
1818
1819 #endif
1820
1821 #endif
1822
1823 #endif
1824
1825 #endif
1826
1827 #endif
1828
1829 #endif
1830
1831 #endif
1832
1833 #endif
1834
1835 #endif
1836
1837 #endif
1838
1839 #endif
1840
1841 #endif
1842
1843 #endif
1844
1845 #endif
1846
1847 #endif
1848
1849 #endif
1850
1851 #endif
1852
1853 #endif
1854
1855 #endif
1856
1857 #endif
1858
1859 #endif
1860
1861 #endif
1862
1863 #endif
1864
1865 #endif
1866
1867 #endif
1868
1869 #endif
1870
1871 #endif
1872
1873 #endif
1874
1875 #endif
1876
1877 #endif
1878
1879 #endif
1880
1881 #endif
1882
1883 #endif
1884
1885 #endif
1886
1887 #endif
1888
1889 #endif
1890
1891 #endif
1892
1893 #endif
1894
1895 #endif
1896
1897 #endif
1898
1899 #endif
1900
1901 #endif
1902
1903 #endif
1904
1905 #endif
1906
1907 #endif
1908
1909 #endif
1910
1911 #endif
1912
1913 #endif
1914
1915 #endif
1916
1917 #endif
1918
1919 #endif
1920
1921 #endif
1922
1923 #endif
1924
1925 #endif
1926
1927 #endif
1928
1929 #endif
1930
1931 #endif
1932
1933 #endif
1934
1935 #endif
1936
1937 #endif
1938
1939 #endif
1940
1941 #endif
1942
1943 #endif
1944
1945 #endif
1946
1947 #endif
1948
1949 #endif
1950
1951 #endif
1952
1953 #endif
1954
1955 #endif
1956
1957 #endif
1958
1959 #endif
1960
1961 #endif
1962
1963 #endif
1964
1965 #endif
1966
1967 #endif
1968
1969 #endif
1970
1971 #endif
1972
1973 #endif
1974
1975 #endif
1976
1977 #endif
1978
1979 #endif
1980
1981 #endif
1982
1983 #endif
1984
1985 #endif
1986
1987 #endif
1988
1989 #endif
1990
1991 #endif
1992
1993 #endif
1994
1995 #endif
1996
1997 #endif
1998
1999 #endif
2000
2001 #endif
2002
2003 #endif
2004
2005 #endif
2006
2007 #endif
2008
2009 #endif
2010
2011 #endif
2012
2013 #endif
2014
2015 #endif
2016
2017 #endif
2018
2019 #endif
2020
2021 #endif
2022
2023 #endif
2024
2025 #endif
2026
2027 #endif
2028
2029 #endif
2030
2031 #endif
2032
2033 #endif
2034
2035 #endif
2036
2037 #endif
2038
2039 #endif
2040
2041 #endif
2042
2043 #endif
2044
2045 #endif
2046
2047 #endif
2048
2049 #endif
2050
2051 #endif
2052
2053 #endif
2054
2055 #endif
2056
2057 #endif
2058
2059 #endif
2060
2061 #endif
2062
2063 #endif
2064
2065 #endif
2066
2067 #endif
2068
2069 #endif
2070
2071 #endif
2072
2073 #endif
2074
2075 #endif
2076
2077 #endif
2078
2079 #endif
2080
2081 #endif
2082
2083 #endif
2084
2085 #endif
2086
2087 #endif
2088
2089 #endif
2090
2091 #endif
2092
2093 #endif
2094
2095 #endif
2096
2097 #endif
2098
2099 #endif
2100
2101 #endif
2102
2103 #endif
2104
2105 #endif
2106
2107 #endif
2108
2109 #endif
2110
2111 #endif
2112
2113 #endif
2114
2115 #endif
2116
2117 #endif
2118
2119 #endif
2120
2121 #endif
2122
2123 #endif
2124
2125 #endif
2126
2127 #endif
2128
2129 #endif
2130
2131 #endif
2132
2133 #endif
2134
2135 #endif
2136
2137 #endif
2138
2139 #endif
2140
2141 #endif
2142
2143 #endif
2144
2145 #endif
2146
2147 #endif
2148
2149 #endif
2150
2151 #endif
2152
2153 #endif
2154
2155 #endif
2156
2157 #endif
2158
2159 #endif
2160
2161 #endif
2162
2163 #endif
2164
2165 #endif
2166
2167 #endif
2168
2169 #endif
2170
2171 #endif
2172
2173 #endif
2174
2175 #endif
2176
2177 #endif
2178
2179 #endif
2180
2181 #endif
2182
2183 #endif
2184
2185 #endif
2186
2187 #endif
21
```

```

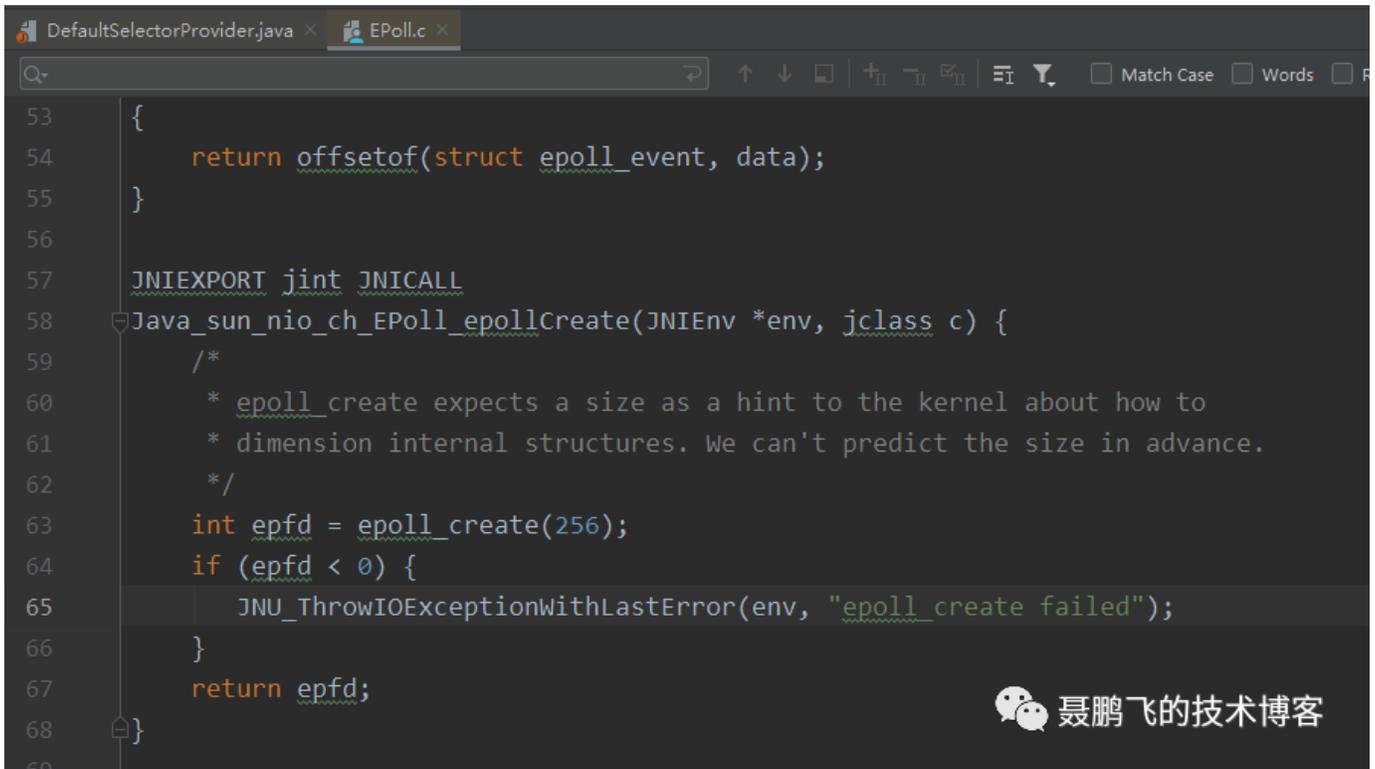
47         c = (Class<SelectorProvider>)Class.forName(cn);
48     } catch (ClassNotFoundException x) {
49         throw new AssertionError(x);
50     }
51     try {
52         return c.newInstance();
53     } catch (IllegalAccessException | InstantiationException x) {
54         throw new AssertionError(x);
55     }
56
57 }
58
59 /**
60  * Returns the default SelectorProvider.
61  */
62 public static SelectorProvider create() {
63     String osname = AccessController
64         .doPrivileged(new GetPropertyAction("os.name"));
65     if (osname.equals("SunOS"))
66         return createProvider( cn: "sun.nio.ch.DevPollSelectorProvider");
67     if (osname.equals("Linux"))
68         return createProvider( cn: "sun.nio.ch.EPollSelectorProvider");
69     return new sun.nio.ch.PollSelectorProvider();
70 }
71
72 }
73

```

如果程序运行在Linux平台的话，就会采用
 sun.nio.ch.EPollSelectorProvider
 说明在高版本的JDK当中，linux平台中已经去除了select，采用了epoll。



我们打开这个文件，进行查看。



```
53 {
54     return offsetof(struct epoll_event, data);
55 }
56
57 JNIEXPORT jint JNICALL
58 Java_sun_nio_ch_EPoll_epollCreate(JNIEnv *env, jclass c) {
59     /*
60      * epoll_create expects a size as a hint to the kernel about how to
61      * dimension internal structures. We can't predict the size in advance.
62      */
63     int epfd = epoll_create(256);
64     if (epfd < 0) {
65         JNU_ThrowIOExceptionWithLastError(env, "epoll_create failed");
66     }
67     return epfd;
68 }
```

 聂鹏飞的技术博客

2.3.3 如何理解与查看系统函数

接下来我给你看一下操作系统的函数。整个linux系统，你可以简单的理解为是c语言写的，那么既然是c语言写的，那么linux系统肯定对外提供了c语言函数，即当前操作系统的函数。

第一：查看socket函数。使用命令：man 2 socket

```
1 npfdev1 x +
ssh://root@192.168.1.10:22 Linux Programmer's Manual SOCKET(2)

NAME
    socket - create an endpoint for communication

SYNOPSIS
    #include <sys/types.h>          /* See NOTES */
    #include <sys/socket.h>

    int socket(int domain, int type, int protocol);

DESCRIPTION
    socket() creates an endpoint for communication and returns a descriptor.

    The domain argument specifies a communication domain; this selects the protocol family which will be used for
    formats include:

    Name                Purpose                Man page
    AF_UNIX, AF_LOCAL   Local communication    unix(7)
    AF_INET              IPv4 Internet protocols ip(7)
    AF_INET6             IPv6 Internet protocols ipv6(7)
    AF_IPX               IPX - Novell protocols
    AF_NETLINK           Kernel user interface device netlink(7)
    AF_X25               ITU-T X.25 / ISO-8208 protocol x25(7)
    AF_AX25              Amateur radio AX.25 protocol
    AF_ATMPVC            Access to raw ATM PVCs
    AF_APPLETALK         Appletalk                ddp(7)
    AF_PACKET            Low level packet interface packet(7)

    The socket has the indicated type, which specifies the communication semantics. Currently defined types are:

    SOCK_STREAM          Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data tra
    SOCK_DGRAM           Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
    SOCK_SEQPACKET       Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams
    system call.
    SOCK_RAW             Provides raw network protocol access.
    SOCK_RDM             Provides a reliable datagram layer that does not guarantee ordering.
    SOCK_PACKET          Obsolete and should not be used in new programs; see packet(7).

    Some socket types may not be implemented by all protocol families; for example, SOCK_SEQPACKET is not implemen
    Since Linux 2.6.27, the type argument serves a second purpose: in addition to specifying a socket type,
    socket():

    SOCK_NONBLOCK       Set the O_NONBLOCK file status flag on the new open file description. Using this flag saves e
    SOCK_CLOEXEC         Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor. See the description of th

    The protocol specifies a particular protocol to be used with the socket. Normal
    protocol can be specified as 0. However, it is possible that many protocols may exist, in which case a parti
```

这个socket函数的意思，linux已经说的很清楚了。即：

socket – create an endpoint for communication
创建一个终端以用于进行通信。

第二：查看select函数。使用命令： man 2 select

```
npfdev1 x +
[root@npfdev1 ~]# man 2 select

SELECT(2)                                Linux Programmer's Manual                                SELECT(2)

NAME
    select, pselect, FD_CLR, FD_ISSET, FD_SET, FD_ZERO - synchronous I/O multiplexing

SYNOPSIS
    /* According to POSIX.1-2001 */
    #include <sys/select.h>

    /* According to earlier standards */
    #include <sys/time.h>
    #include <sys/types.h>
    #include <unistd.h>

    int select(int nfds, fd_set *readfds, fd_set *writefds,
               fd_set *exceptfds, struct timeval *timeout);

    void FD_CLR(int fd, fd_set *set);
    int  FD_ISSET(int fd, fd_set *set);
    void FD_SET(int fd, fd_set *set);
    void FD_ZERO(fd_set *set);

    #include <sys/select.h>

    int pselect(int nfds, fd_set *readfds, fd_set *writefds,
                fd_set *exceptfds, const struct timespec *timeout,
                const sigset_t *sigmask);

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

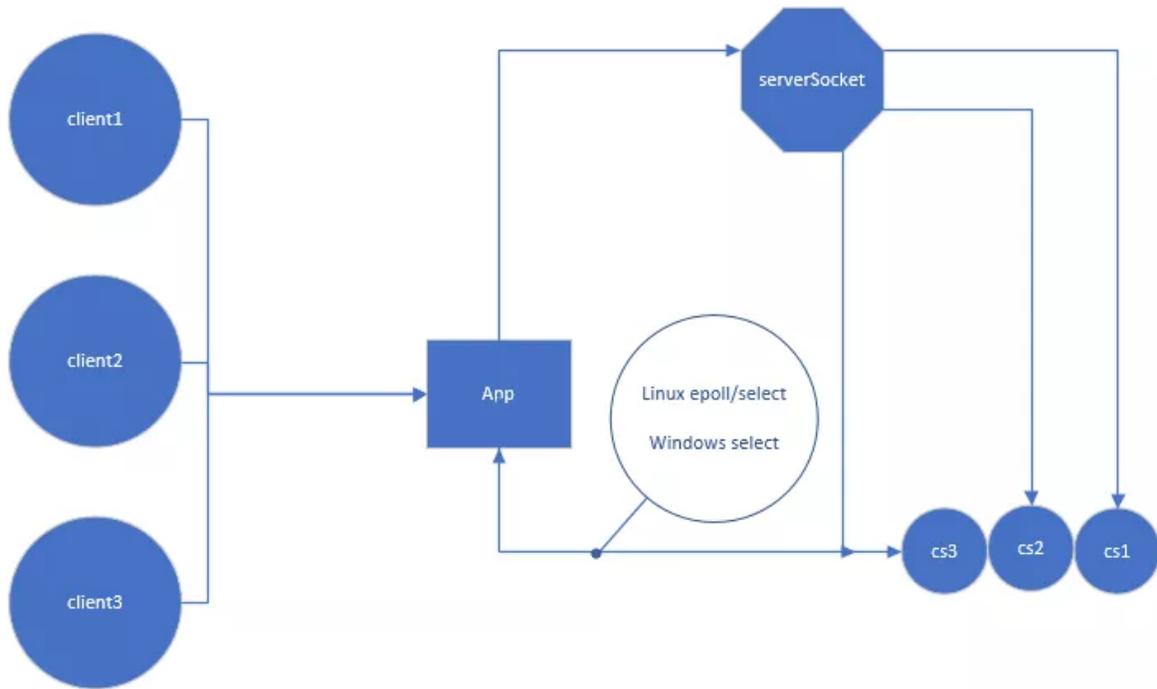
    pselect(): _POSIX_C_SOURCE >= 200112L || _XOPEN_SOURCE >= 600

DESCRIPTION
    select() and pselect() allow a program to monitor multiple file descriptors, waiting until one or more of th
    A file descriptor is considered ready if it is possible to perform the corresponding I/O operation (e.g., rea

    The operation of select() and pselect() is identical, with three differences:

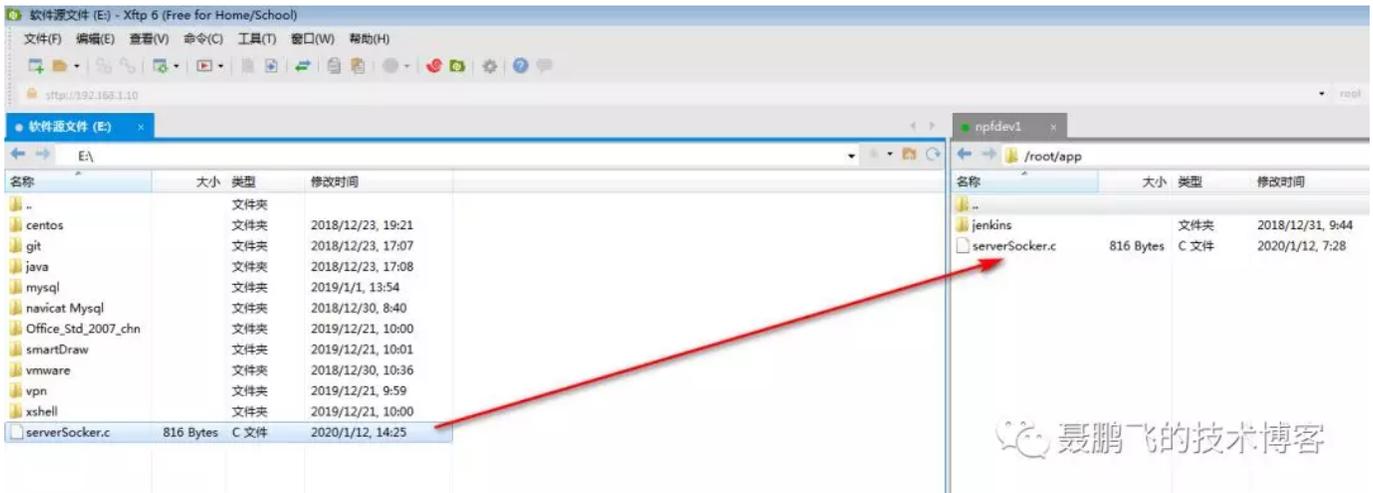
    (i) select() uses a timeout that is a struct timeval (with seconds and microseconds), while pselect() uses
    (ii) select() may update the timeout argument to indicate how much time was left. pselect() does not chang
    (iii) select() has no sigmask argument, and behaves as pselect() called with NULL sigmask.
```

谈到这个select函数，我们不得不提一下NIO的思想，就是说sun公司的那帮人是怎么想到将list集合交给操作系统去循环的。



聂鹏飞的技术博客

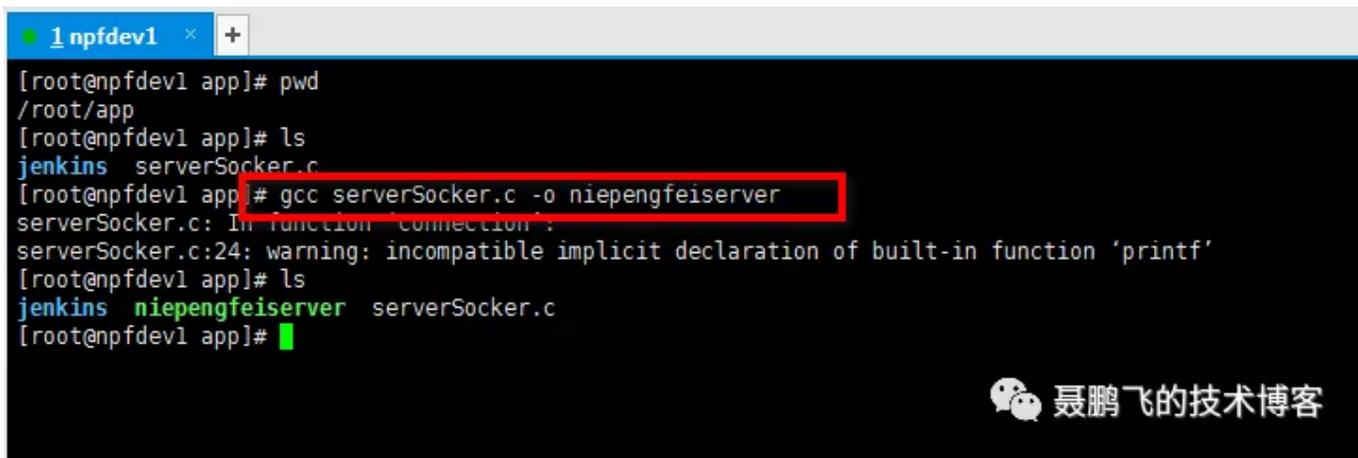
接下来我们来自己写一个c语言程序，去调用linux系统函数。那么请看我如何写的，耐心一点好吧，我接下来就写了。



聂鹏飞的技术博客

运行如下命令进行编译：

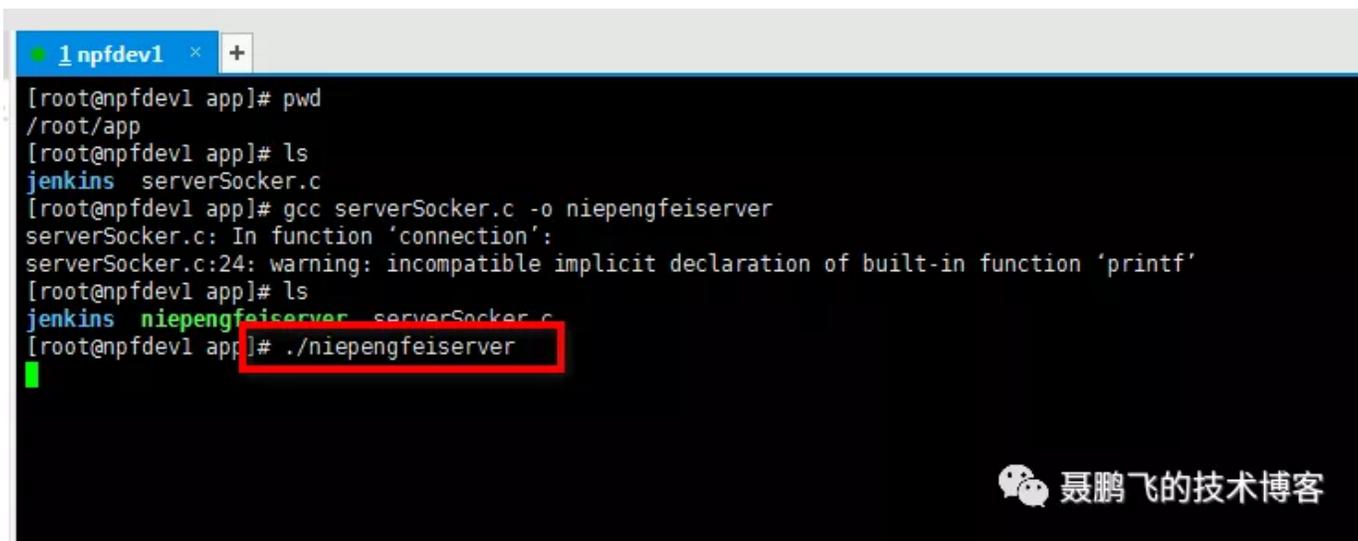
```
1 npfdev1 x +
[root@npfdev1 app]# pwd
/root/app
[root@npfdev1 app]# ls
jenkins serverSocket.c
[root@npfdev1 app]# gcc serverSocket.c -o niepengfeiserver
serverSocket.c: In function 'connection':
serverSocket.c:24: warning: incompatible implicit declaration of built-in function 'printf'
[root@npfdev1 app]# ls
jenkins niepengfeiserver serverSocket.c
[root@npfdev1 app]#
```



 聂鹏飞的技术博客

现在我们运行刚才编译好的程序，如下：

```
1 npfdev1 x +
[root@npfdev1 app]# pwd
/root/app
[root@npfdev1 app]# ls
jenkins serverSocket.c
[root@npfdev1 app]# gcc serverSocket.c -o niepengfeiserver
serverSocket.c: In function 'connection':
serverSocket.c:24: warning: incompatible implicit declaration of built-in function 'printf'
[root@npfdev1 app]# ls
jenkins niepengfeiserver serverSocket.c
[root@npfdev1 app]# ./niepengfeiserver
```



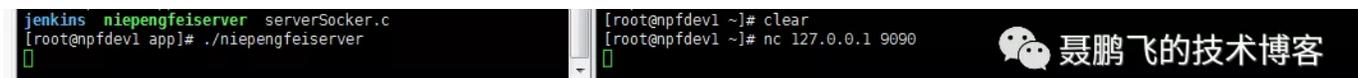
 聂鹏飞的技术博客

目前程序已经运行成功，正在等待连接。我们开一个客户端，运行命令：

```
nc 127.0.0.1 9090
```

```
jenkins niepengfeiserver serverSocket.c
[root@npfdev1 app]# ./niepengfeiserver
[ ]

[root@npfdev1 ~]# clear
[root@npfdev1 ~]# nc 127.0.0.1 9090
[ ]
```

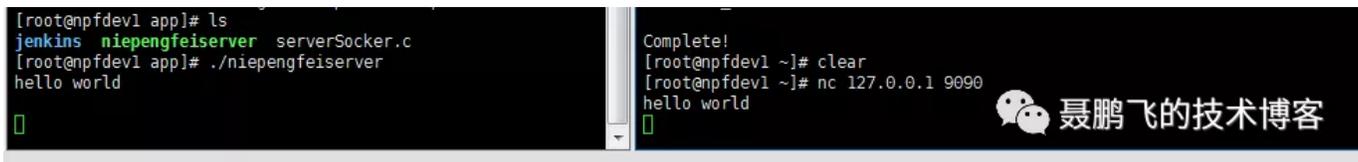


 聂鹏飞的技术博客

从上图可以看出，已经连接成功，之后我们在利用客户端向服务端发送一条数据，进行测试。

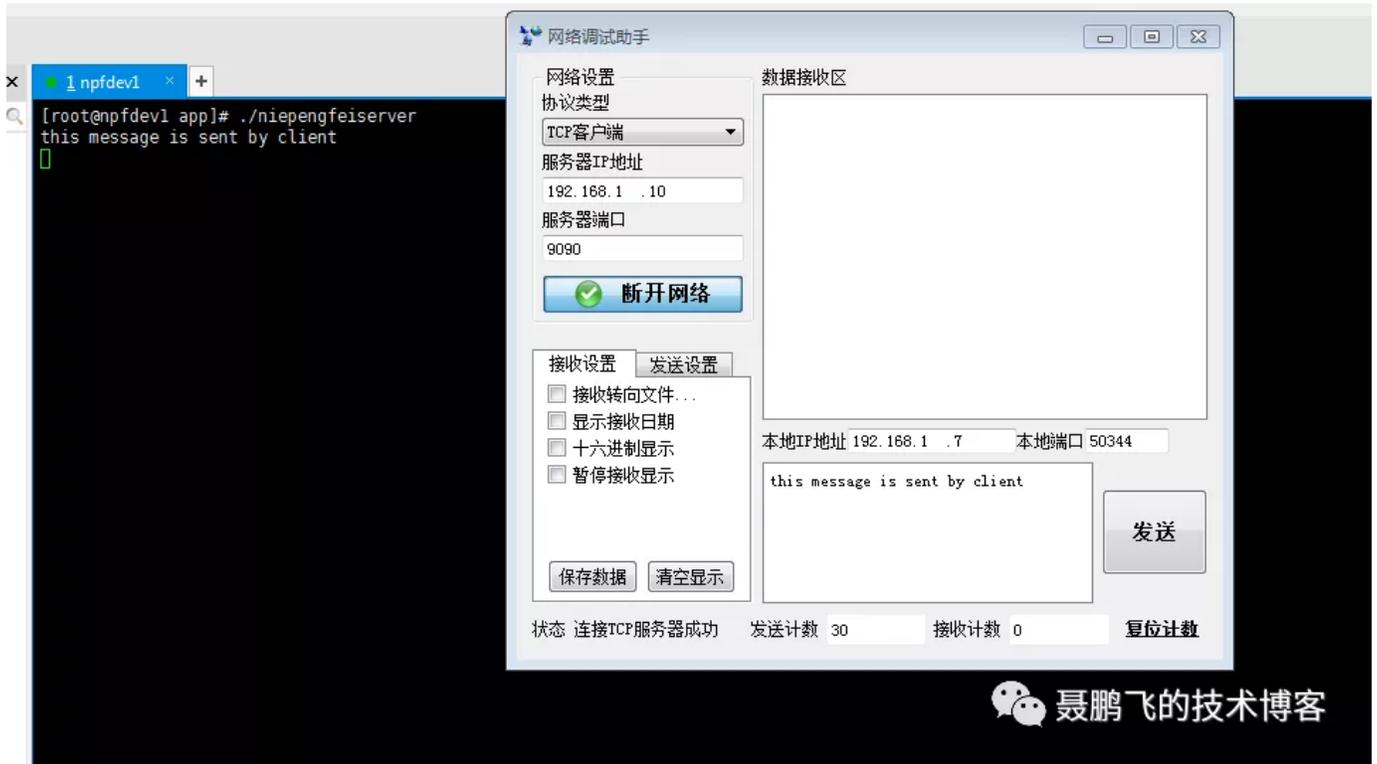
```
[root@npfdev1 app]# ls
jenkins niepengfeiserver serverSocket.c
[root@npfdev1 app]# ./niepengfeiserver
hello world
[ ]

Complete!
[root@npfdev1 ~]# clear
[root@npfdev1 ~]# nc 127.0.0.1 9090
hello world
[ ]
```



 聂鹏飞的技术博客

以下是通过客户端工具进行测试。



到此，我们在linux平台写出了服务端。