

自我否定的勇气，兼收并蓄的气度

永久域名 <http://suhuanzheng7784877.iteye.com>



suhuanzheng7784877

浏览: 116947 次

性别:

来自: 北京

我现在离线

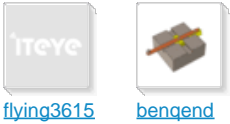
[详细资料](#)

[留言簿](#)

搜索本博客

最近访客

[>>更多访客](#)



博客分类

- [全部博客 \(161\)](#)
- [java SE \(8\)](#)
- [JavaWeb \(19\)](#)
- [JavaEE \(61\)](#)
- [javascript \(0\)](#)
- [eclipse plugin \(1\)](#)
- [orm \(0\)](#)
- [开源项目 \(17\)](#)
- [经验总结 \(20\)](#)
- [linux \(0\)](#)
- [jbpm \(10\)](#)
- [分布式集群 \(18\)](#)
- [程序人生 \(6\)](#)
- [NIO \(1\)](#)

2011-07-12

[Java分布式应用学习笔记01分布式Java应用和...](#)

|

[读金庸故事，品程序人生04韦小宝做事的启示](#)

70 顶

5 踩

3种下载文件程序的思考，为何使用NIO进行异步网络通讯

博客分类: [NIO](#)

Java

多线程

Socket

Mina

nio

1. 前言

现在很多做网络通讯中间代理层的通讯都是使用Java1.4以后推出的NIO进行编写，现在还有很多开源的框架也是封装了NIO的书写细节来帮助大家简写异步非阻塞通讯服务。像MySql的代理中间件amoeba-mysql-proxy就是采用NIO的方式处理client端过来的request，之后与Mysql-Server层的通讯也是采用NIO进行命令消息发送的。再看咱们JavaEye首页介绍的项目xmemcached，其中作者Dennis是其xmemcached的开发人，他也是通过NIO的方式与memcached的Server进行异步通讯的，Dennis的另一个项目yanf4j就是一个NIO框架，xmemcache也是借助这个NIO框架实现的异步非阻塞方式的网络通讯，Apache的MINA框架都是NIO的封装再实现。

那么我们就来回顾一下以前的处理方式，来看看为什么现在要使用NIO来进行异步非阻塞方式的通讯吧，网上很多文章都是几句话将NIO和原始的socket通讯的优劣一带而过，我们这次用一个简单的下载大文件的网络服务程序进行说明。使用3种模式来说明，分别是同步单独线程服务运行模式、传统阻塞多线程模式、使用NIO异步非阻塞模式。

我们设置服务器上有一个1.81GB的电影，格式为RMVB。使用Server进行服务监听，客户端请求到Server，建立网络通讯，进行电影下载。

2. 同步单线程阻塞

使用同步单线程下载，是最原始的socket通讯，服务端的代码如下

Java代码

```
1.  <span style="font-size: x-small;">package server;
2.
3.  import java.io.FileInputStream;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.net.ServerSocket;
7.  import java.net.Socket;
8.
9.  /**
10.   * liuyan
11.   */
12.  public class FilmServer {
13.
14.      public static void main(String[] args) {
15.          FilmServer ms = new FilmServer();
16.          try {
17.              ms.server();
18.          } catch (Exception e) {
19.              e.printStackTrace();
20.          }
21.      }
22.
23.      /**
24.       * 服务器端响应请求
25.       *
26.       * @throws Exception
27.       */
28.      public void server() throws Exception {
29.
```

我的留言簿

>>更多留言

■ 牛人

-- by [gongxihai](#)

■ 好厉害

-- by [yinyu](#)

■ 偶然间发现你的id是素还真。看来道友还挺多 掩面泪奔

-- by [戡武王](#)

其他分类

■ [我的收藏](#) (10)

■ [我的代码](#) (0)

■ [我的书斋](#) (5)

■ [我的论坛主题帖](#) (1)

■ [我的所有论坛帖](#) (49)

■ [我的精华良好帖](#) (0)

■ [我解决的问题](#) (1)

最近加入群组

■ [算法~](#)

■ [系统架构与架构应用](#)

■ [EXT](#)

■ [maven](#)

■ [JBPM @net](#)

存档

■ [2011-09](#) (4)

■ [2011-08](#) (5)

■ [2011-07](#) (7)

■ [更多存档...](#)

评论排行榜

■ [《我奋斗了18年才和你坐在一起喝咖啡》，而...](#)

■ [读金庸故事、品程序人生05刀狂剑痴](#)

■ [3种下载文件程序的思考，为何使用NIO进行...](#)

■ [Java分布式应用学习笔记05多线程下的并发同...](#)

■ [读金庸故事、品程序人生01出世前的修炼](#)


 RSS



```
30. // 0.建立服务器端的server的socket
31. ServerSocket ss = new ServerSocket(9999);
32.
33. while (true) {
34.
35.     // 1.打开socket连接
36.     // 等待客户端的请求
37.     final Socket server = ss.accept();
38.
39.     System.out.println("服务-----请求开始start");
40.
41.     // 2.打开socket的流信息，准备下面的操作
42.     final InputStream is = server.getInputStream();
43.     byte b[] = new byte[1024];
44.
45.     int readCount = is.read(b);
46.
47.     String str = new String(b);
48.
49.     str = str.trim();
50.
51.     final String serverFileName = str;
52.
53.     // 3.对流信息进行读写操作
54.     System.out.println("客户端传过来的信息是: " + str);
55.
56.     System.out.println("线程" + Thread.currentThread().getName() + "启动");
57.
58.     try {
59.
60.         FileInputStream fileInputStream = new FileInputStream(
61.             serverFileName);
62.
63.         // 3.1 服务器回复客户端信息(response)
64.         OutputStream os = server.getOutputStream();
65.
66.         byte[] bfile = new byte[1024];
67.
68.         // 往客户端写
69.         while (fileInputStream.read(bfile) > 0) {
70.             os.write(bfile);
71.         }
72.
73.         fileInputStream.close();
74.
75.         os.close();
76.
77.         // 4.关闭socket
78.         // 先关闭输入流
79.         is.close();
80.
81.         // 最后关闭socket
82.         server.close();
83.
84.     } catch (Exception e) {
85.         // TODO Auto-generated catch block
86.         e.printStackTrace();
87.     }
88.
89.     System.out.println("服务-----请求结束over");
90. }
91.
92. }
```

```
93.
94. }</span>
```

服务端这么写代码会有什么问题？咱们先来看客户端代码，之后运行后就知道了。

Java代码 

```
1. <span style="font-size: x-small;">package client;
2.
3. import java.io.FileOutputStream;
4. import java.io.IOException;
5. import java.io.InputStream;
6. import java.io.OutputStream;
7. import java.net.Socket;
8. import java.net.UnknownHostException;
9.
10. /**
11.  * liuyan
12.  * @version 1.0
13.  */
14. public class FilmClient {
15.     public static void main(String[] args) {
16.         for (int i = 1; i <= 2; i++) {
17.             Client client = new Client();
18.             client.i = i;
19.             client.start();
20.         }
21.     }
22. }
23.
24. class Client extends Thread {
25.
26.     int i;
27.
28.     @Override
29.     public void run() {
30.
31.         // 1.建立socket连接
32.         Socket client;
33.         try {
34.             client = new Socket("127.0.0.1", 9999);
35.
36.             // 2.打开socket的流信息，准备下面的操作
37.             OutputStream os = client.getOutputStream();
38.
39.             // 3.写信息
40.             os.write(("d://film/2.rmvb").getBytes());
41.
42.             String fileName = "c://io"+i+".rmvb";
43.
44.             FileOutputStream fileOutputStream = new FileOutputStream(fileName);
45.
46.             // 3.1接收服务器端的反馈
47.             InputStream is = client.getInputStream();
48.             byte b[] = new byte[1024];
49.
50.             while(is.read(b)>0){
51.                 fileOutputStream.write(b);
52.             }
53.
54.             // 4.关闭socket
55.             // 先关闭输出流
56.             os.close();
57.
58.             // 最后关闭socket
59.             client.close();
```

```
60.         } catch (UnknownHostException e) {
61.             // TODO Auto-generated catch block
62.             e.printStackTrace();
63.         } catch (IOException e) {
64.             // TODO Auto-generated catch block
65.             e.printStackTrace();
66.         }
67.     }
68.
69. }</span>
```

客户端启动了2个线程进行下载电影的工作，先启动服务端，再运行客户端，会看笔者本地的硬盘C分区到有如下效果。

 io1.rm vb	2011/7/10 11:44	媒体文件 (.rm vb)	1,208,317 KB
 io2.rm vb	2011/7/10 11:44	媒体文件 (.rm vb)	0 KB


可以看到线程2的下载任务一直是0字节，等第一个线程下载完成后呢，线程2的下载任务才能进行。

 io1.rm vb	2011/7/10 11:44	媒体文件 (.rm vb)	1,899,729 KB
 io2.rm vb	2011/7/10 11:44	媒体文件 (.rm vb)	239,447 KB

服务端的代码造成的问题就是使用传统的sokect网络通讯，那么另一个客户端的线程请求到server端的时候就发生了阻塞的情况，也就是说，服务端相当一个厕所，厕所就有只有一个坑位，来了一个人，相当于客户端请求，那这个人相当于就把坑位给占了，write操作和read操作会阻塞，这个人还没解决完问题呢，下个人就来了，没办法，哥们儿先在门外等等啊，等前一个客户爽完了再给您提供服务好吧。那么如何解决这个占着坑位不让别人用的情况呢？

3. 阻塞的多线程

为了解决以上问题，那么之后很多Server肯定不可能像以上程序那么做，不过以前很多Server都是基于单线程服务改造一下，做成多线程的Server的通讯，修改一下上面的Server代码，如下

Java代码 

```
1.  <span style="font-size: x-small;">package server;
2.
3.  import java.io.FileInputStream;
4.  import java.io.InputStream;
5.  import java.io.OutputStream;
6.  import java.net.ServerSocket;
7.  import java.net.Socket;
8.
9.  /**
10.   *
11.   */
12.  public class FilmServerNewThread {
13.
14.      public static void main(String[] args) {
15.          FilmServerNewThread ms = new FilmServerNewThread();
16.          try {
17.              ms.server();
18.          } catch (Exception e) {
19.              e.printStackTrace();
20.          }
21.      }
22.
23.      /**
24.       * 服务器端响应请求
25.       *
26.       * @throws Exception
27.       */
28.      public void server() throws Exception {
29.
30.          // 0.建立服务器端的server的socket
31.          ServerSocket ss = new ServerSocket(9999);
32.
```

```
33.         while (true) {
34.
35.             // 1.打开socket连接
36.             // 等待客户端的请求
37.             final Socket server = ss.accept();
38.
39.             System.out.println("服务-----请求开始start");
40.
41.             // 2.打开socket的流信息，准备下面的操作
42.             final InputStream is = server.getInputStream();
43.             byte b[] = new byte[1024];
44.
45.             int readCount = is.read(b);
46.
47.             String str = new String(b);
48.
49.             str = str.trim();
50.
51.             final String serverFileName = str;
52.
53.             // 3.对流信息进行读写操作
54.             System.out.println("客户端传过来的信息是: " + str);
55.
56.             if (readCount > 0) {
57.                 new Thread() {
58.
59.                     @Override
60.                     public void run() {
61.
62.                         System.out.println("线程"
63.                             + Thread.currentThread().getN
ame() + "启动");
64.
65.                         try {
66.
67.                             FileInputStream fileInputStream = new
FileInputStream(
68.                                 serverFileName);
69.
70.                             // 3.1 服务器回复客户端信息(response)
71.                             OutputStream os = server.getOutputStream();
72.
73.                             byte[] bfile = new byte[1024];
74.
75.                             // 往客户端写
76.                             while (fileInputStream.read(bfile) >
0) {
77.
78.                                 os.write(bfile);
79.
80.                                 fileInputStream.close();
81.
82.                                 os.close();
83.
84.                                 // 4.关闭socket
85.                                 // 先关闭输入流
86.                                 is.close();
87.
88.                                 // 最后关闭socket
89.                                 server.close();
90.
91.                             } catch (Exception e) {
92.                                 // TODO Auto-generated catch block
```

```
93.         e.printStackTrace();
94.     }
95. }
96. }.start();
97. }
98.
99.     System.out.println("服务-----请求结束over");
100. }
101.
102. }
103. }
```

以上的Server就是在原始的socket基础上加了线程，每一个Client请求过来后，整个Server主线程不必处于阻塞状态，接收请求后直接另起一个新的线程来处理和服务端的交互，就是往客户端发送二进制包。这个在新线程中虽然阻塞，但是对于服务主线程没有阻塞的影响，主线程依然通过死循环监听着客户端的一举一动。另一个客户端的线程发起请求后就再起一个新的线程对象去为客户端服务。执行效果如下

 io1.rmvb	2011/7/10 12:23	媒体文件 (.rmvb)	77,473 KB
 io2.rmvb	2011/7/10 12:23	媒体文件 (.rmvb)	27,678 KB

2个线程互不影响，各自下载各自的。当然从非常严格的意义来讲，str变量在十分高并发的情况下有线程安全问题，这个咱暂且忽略，就着眼于低并发的情况。这个问题是什么呢，就是如果客户端请求比较多了，那么为每一个客户端开辟一个新的线程对象来处理网络传输的请求，需要创建个线程对象，而且这个线程对象从时间上来讲还是处于长连接，这个就比较消费系统资源，这个打开进程管理器就可以看到。而且每一个线程内部都是阻塞的，也没有说完全利用好这个新创建的线程。还拿刚才上厕所所举例子，好比现在不止一个坑位了，来了一个用户我这边就按照工程师的厕所坑位图建立一个新的坑位，客户来了，不用等待老坑位，用新创建的坑位就行了。等那个老坑位用完了，自然有垃圾回收器去消灭那个一次性的坑位的，腾出资源位置为了建立新的坑位。长时间连接的意思，相当于这个人上厕所的时间非常长，便秘??需要拉一天才能爽完.....老的坑位一时半会儿回收不了，新的坑位需要有空间为其建造茅房以便满足客户端的“急切方便”需要。久而久之，线程数目一多，系统就挂了的概率就增多了（谁也别想上，全玩完了）。

4. 异步非阻塞

使用JDK1.4的NIO可以适当的解决上面的问题，异步 I/O 是一种 没有阻塞地读写数据的方法。通常，在代码进行 read() 调用时，代码会阻塞直至有可供读取的数据。同样， write() 调用将会阻塞直至数据能够写入。异步 I/O 调用不会阻塞。相反，您将注册对特定 I/O 事件的兴趣 — 可读的数据的到达、新的套接字连接，等等，而在发生这样的事件时，系统将会告诉您。异步 I/O 的一个优势在于，它允许您同时根据大量的输入和输出执行 I/O。同步程序常常要求助于轮询，或者创建许许多多的线程以处理大量的连接。使用异步 I/O，您可以监听任何数量的通道上的事件，不用轮询，也不用额外的线程。还是举上公共厕所例子，虽然这个例子有点臭臭的。您现在有“便便”的需求了，不用那么麻烦，看看公共厕所是否有人占领，也不用给您另起个新坑位，您就拿一根我们服务端定制的容器和一个很粗管子，这个坐便器的大小因您那个地方的尺寸而定，坐便器往您的那个地方一放，再将坐便器和管子一连接，OK，您就敞开了“爽”吧。不用担心，这个管子自然会连接到相应的肥料厂家，将您的排泄物有效回收加以利用的。您完了事，擦擦屁股，关上管子该干嘛还干嘛就行了。另一个人也有这个需求，没问题，每个要我们提供服务的人都用这根管子，和自己的坐便器就行了，管子很粗，谁来连这个管子都行，有多少都行啊。下面我们来看基于NIO的网络下载程序

Java代码 ☆

```
1.  <span style="font-size: x-small;">package server;
2.
3.  import java.io.FileInputStream;
4.  import java.io.IOException;
5.  import java.net.InetSocketAddress;
6.  import java.nio.ByteBuffer;
7.  import java.nio.CharBuffer;
8.  import java.nio.channels.FileChannel;
9.  import java.nio.channels.SelectionKey;
10. import java.nio.channels.Selector;
11. import java.nio.channels.ServerSocketChannel;
12. import java.nio.channels.SocketChannel;
13. import java.nio.charset.Charset;
```

```
14. import java.nio.charset.CharsetDecoder;
15. import java.util.Iterator;
16.
17. /**
18.  *
19.  * @author liuyan
20.  *
21.  */
22. public class NIOServer {
23.     static int BLOCK = 500*1024;
24.
25.     /**
26.      * 处理客户端的内部类，专门负责处理与用户的交互
27.      */
28.     public class HandleClient {
29.         protected FileChannel channel;
30.         protected ByteBuffer buffer;
31.         String filePath;
32.
33.         /**
34.          * 构造函数，文件的管道初始化
35.          * @param filePath
36.          * @throws IOException
37.          */
38.         public HandleClient(String filePath) throws IOException {
39.
40.             //文件的管道
41.             this.channel = new FileInputStream(filePath).getChannel();
42.
43.             //建立缓存
44.             this.buffer = ByteBuffer.allocate(BLOCK);
45.             this.filePath = filePath;
46.         }
47.
48.         /**
49.          * 读取文件管道中数据到缓存中
50.          * @return
51.          */
52.         public ByteBuffer readBlock() {
53.             try {
54.
55.                 //清除缓冲区的内容，posistion设置为0，limit设置为缓冲的容量大小capa
56.                 city buffer.clear();
57.
58.                 //读取
59.                 int count = channel.read(buffer);
60.
61.                 //将缓存的中的posistion设置为0，将缓存中的limit设置在原始posistion
62.                 位置上 buffer.flip();
63.                 if (count <= 0)
64.                     return null;
65.             } catch (IOException e) {
66.                 e.printStackTrace();
67.             }
68.             return buffer;
69.         }
70.
71.         /**
72.          * 关闭服务端的文件管道
73.          */
74.         public void close() {
75.             try {
```


```
76.         channel.close();
77.     } catch (IOException e) {
78.         e.printStackTrace();
79.     }
80. }
81. }
82.
83. protected Selector selector;
84. protected String filename = "d:\\film\\60.rmvb"; // a big file
85. protected ByteBuffer clientBuffer = ByteBuffer.allocate(BLOCK);
86. protected CharsetDecoder decoder;
87.
88. // 构造服务端，服务管道等等
89. public NIOServer(int port) throws IOException {
90.     selector = this.getSelector(port);
91.     Charset charset = Charset.forName("GB2312");
92.     decoder = charset.newDecoder();
93. }
94.
95. // 获取Selector
96. // 构造服务端，服务管道等等
97. protected Selector getSelector(int port) throws IOException {
98.     ServerSocketChannel server = ServerSocketChannel.open();
99.     Selector sel = Selector.open();
100.    server.socket().bind(new InetSocketAddress(port));
101.    server.configureBlocking(false);
102.
103.    // 刚开始就注册链接事件
104.    server.register(sel, SelectionKey.OP_ACCEPT);
105.    return sel;
106. }
107.
108. // 服务启动的开始入口
109. public void listen() {
110.     try {
111.         for (;;) {
112.
113.             // ?
114.             selector.select();
115.             Iterator<SelectionKey> iter = selector.selectedKeys()
116.                 .iterator();
117.             while (iter.hasNext()) { // 首先是最感兴趣的连接事件
118.                 SelectionKey key = iter.next();
119.
120.                 //
121.                 iter.remove();
122.
123.                 // 处理事件
124.                 handleKey(key);
125.             }
126.         }
127.     } catch (IOException e) {
128.         e.printStackTrace();
129.     }
130. }
131.
132. // 处理事件
133. protected void handleKey(SelectionKey key) throws IOException {
134.     if (key.isAcceptable()) { // 接收请求
135.
136.         // 允许网络连接事件
137.         ServerSocketChannel server = (ServerSocketChannel) key.channel();
138.         SocketChannel channel = server.accept();
139.         channel.configureBlocking(false);
```



```
140.
141.         //网络管道准备处理读事件
142.         channel.register(selector, SelectionKey.OP_READ);
143.     } else if (key.isReadable()) { // 读信息
144.         SocketChannel channel = (SocketChannel) key.channel();
145.
146.         //从客户端读过来的数据块
147.         int count = channel.read(clientBuffer);
148.         if (count > 0) {
149.
150.             //读取过来的缓存进行有效分割，posistion设置为0，保证从缓存的有效位置
开始读取，limit设置为原先的posistion上
151.             //这样一来从posistion~limit这段缓存数据是有效，可利用的
152.             clientBuffer.flip();
153.
154.             //对客户端缓存块进行编码
155.             CharBuffer charBuffer = decoder.decode(clientBuffer);
156.             System.out.println("Client >>download>>" + charBuffer.toString(
));
157.
158.             //对网络管道注册写事件
159.             SelectionKey wKey = channel.register(selector,
160.                 SelectionKey.OP_WRITE);
161.
162.             //将网络管道附着上一个处理类HandleClient，用于处理客户端事件的类
163.             wKey.attach(new HandleClient(charBuffer.toString()));
164.         } else{
165.             //如客户端没有可读事件，关闭管道
166.             channel.close();
167.         }
168.
169.         clientBuffer.clear();
170.     } else if (key.isWritable()) { // 写事件
171.         SocketChannel channel = (SocketChannel) key.channel();
172.
173.         //从管道中将附着处理类对象HandleClient取出来
174.         HandleClient handle = (HandleClient) key.attachment();
175.
176.         //读取文件管道，返回数据缓存
177.         ByteBuffer block = handle.readBlock();
178.         if (block != null){
179.             //System.out.println("---"+new String(block.array()));
180.
181.             //写给客户端
182.             channel.write(block);
183.         }else {
184.             handle.close();
185.             channel.close();
186.         }
187.     }
188. }
189.
190. public static void main(String[] args) {
191.     int port = 12345;
192.     try {
193.         NIOserver server = new NIOserver(port);
194.         System.out.println("Listernint on " + port);
195.         while (true) {
196.             server.listen();
197.         }
198.     } catch (IOException e) {
199.         e.printStackTrace();
200.     }
201. }
```

202. }

ServerSocketChannel相当于我们说的那个大粗管子，在它上面注册了很多这个管子感兴趣的事件，比如大便、小便、酒醉后吐的污垢都是它关心的。至于谁来控制管道应该关心的事件，是由管道通过Selector注册事件完成的，Selector相当于一个大管道的维护员了。管道必须得有服务商的厂家维护吧，不能滥用吧。Selector就是个管家，负责管道的事件监听的。XXXXBuffer相当于咱们说的坐便器，它是以块为单位进行管道疏通的，假如您的尺寸特别大，估计您排出的那个玩意也小不了，就配置一个大点的缓存传给服务那边，当然，您这边得到的服务端返回的加工后肥料，返给您的也是和您配置的尺寸有关系的。客户端的代码如下

Java代码 

```
1.  <span style="font-size: x-small;">package client;
2.
3.  import java.io.FileNotFoundException;
4.  import java.io.FileOutputStream;
5.  import java.io.IOException;
6.  import java.net.InetSocketAddress;
7.  import java.nio.ByteBuffer;
8.  import java.nio.CharBuffer;
9.  import java.nio.channels.SelectionKey;
10. import java.nio.channels.Selector;
11. import java.nio.channels.SocketChannel;
12. import java.nio.charset.Charset;
13. import java.nio.charset.CharsetEncoder;
14. import java.util.Iterator;
15. import java.util.concurrent.ExecutorService;
16. import java.util.concurrent.Executors;
17.
18. /**
19.  *
20.  * @author liuyan
21.  *
22.  */
23. public class NIOClient {
24.     static int SIZE = 2;
25.     final static int bufferSize = 500 * 1024;
26.     static InetSocketAddress ip = new InetSocketAddress("localhost", 12345);
27.     static CharsetEncoder encoder = Charset.forName("GB2312").newEncoder();
28.
29.     static class Download implements Runnable {
30.         protected int index;
31.         String outfile = null;
32.
33.         public Download(int index) {
34.             this.index = index;
35.             this.outfile = "c:\\\" + index + ".rmvb";
36.         }
37.
38.         public void run() {
39.
40.             FileOutputStream fout = null;
41.             // FileChannel fcout = null;
42.             try {
43.                 fout = new FileOutputStream(outfile);
44.                 // fcout = fout.getChannel();
45.             } catch (FileNotFoundException e1) {
46.                 // TODO Auto-generated catch block
47.                 e1.printStackTrace();
48.             }
49.
50.             try {
51.                 long start = System.currentTimeMillis();
52.
53.                 // 打开客户端socket管道
```

```
54.         SocketChannel client = SocketChannel.open();
55.
56.         // 客户端的管道的通讯模式
57.         client.configureBlocking(false);
58.
59.         // 选择器
60.         Selector selector = Selector.open();
61.
62.         // 往客户端管道上注册感兴趣连接事件
63.         client.register(selector, SelectionKey.OP_CONNECT);
64.
65.         // 配置IP
66.         client.connect(ip);
67.
68.         // 配置缓存大小
69.         ByteBuffer buffer = ByteBuffer.allocate(bufferSize);
70.         int total = 0;
71.         FOR: for (;;) {
72.
73.             // 阻塞，返回发生感兴趣事件的数量
74.             selector.select();
75.
76.             // 相当于获得感兴趣事件的集合迭代
77.             Iterator<SelectionKey> iter = selector.selectedKeys()
78.
79.                 .iterator();
80.
81.             while (iter.hasNext()) {
82.
83.                 SelectionKey key = iter.next();
84.
85.                 System.out.println("-----Thread "
86.                                     + index + "-----"
87.
88.                                     +key.readyOps());
89.
90.                 // 删除这个马上就要被处理的事件
91.                 iter.remove();
92.
93.                 // 感兴趣的是可连接的事件
94.                 if (key.isConnectable()) {
95.
96.                     // 获得该事件中的管道对象
97.                     SocketChannel channel = (SocketChannel)
98.
99.                         key.channel();
100.
101.                     // 如果该管道对象已经连接好了
102.                     if (channel.isConnectionPending())
103.                         channel.finishConnect();
104.
105.                     // 往管道中写一些块信息
106.                     channel.write(encoder.encode(CharBuffer.wrap(
107.
108.                         "d://film/1.rm"
109.
110.                         +key.readyOps())));
111.
112.                     // 之后为该客户端管道注册新的感兴趣的事件
113.                     channel.register(selector, SelectionKey.OP_READ);
114.
115.                 } else if (key.isReadable()) {
116.
117.                     // 由事件获得通讯管道
118.                     SocketChannel channel = (SocketChannel)
```

```
1) key

111.                                     .channel();
112.
113.                                     // 从管道中读取数据放到缓存中
114.                                     int count = channel.read(buffer);
115.                                     System.out.println("count:" + count);
116.
117.                                     if (count > 0) {
118.
119.                                         // 统计读取的字节数目
120.                                         total += count;
121.
122.                                         // 这样一来从posistion~limit这
123.
124.                                         // buffer.flip();
125.
126.                                         buffer.clear();
127.
128.                                         // 往输出文件中去了
129.                                         if (count < bufferSize) {
130.
131.                                             byte[] overByte = ne
132.
133.                                             for (int index = 0;
134.
135.                                             overByte[ind
136.
137.                                             }
138.
139.                                             fout.write(overByte);
140.
141.                                             } else {
142.
143.                                             fout.write(buffer.arr
144.
145.                                             ay());
146.
147.                                             }
148.
149.                                             } else {
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
1000.
1001.
1002.
1003.
1004.
1005.
1006.
1007.
1008.
1009.
1010.
1011.
1012.
1013.
1014.
1015.
1016.
1017.
1018.
1019.
1020.
1021.
1022.
1023.
1024.
1025.
1026.
1027.
1028.
1029.
1030.
1031.
1032.
1033.
1034.
1035.
1036.
1037.
1038.
1039.
1040.
1041.
1042.
1043.
1044.
1045.
1046.
1047.
1048.
1049.
1050.
1051.
1052.
1053.
1054.
1055.
1056.
1057.
1058.
1059.
1060.
1061.
1062.
1063.
1064.
1065.
1066.
1067.
1068.
1069.
1070.
1071.
1072.
1073.
1074.
1075.
1076.
1077.
1078.
1079.
1080.
1081.
1082.
1083.
1084.
1085.
1086.
1087.
1088.
1089.
1090.
1091.
1092.
1093.
1094.
1095.
1096.
1097.
1098.
1099.
1100.
1101.
1102.
1103.
1104.
1105.
1106.
1107.
1108.
1109.
1110.
1111.
1112.
1113.
1114.
1115.
1116.
1117.
1118.
1119.
1120.
1121.
1122.
1123.
1124.
1125.
1126.
1127.
1128.
1129.
1130.
1131.
1132.
1133.
1134.
1135.
1136.
1137.
1138.
1139.
1140.
1141.
1142.
1143.
1144.
1145.
1146.
1147.
1148.
1149.
1150.
1151.
1152.
1153.
1154.
1155.
1156.
1157.
1158.
1159.
1160.
1161.
1162.
1163.
1164.
1165.
1166.
1167.
1168.
1169.
1170.
1171.
1172.
1173.
1174.
1175.
1176.
1177.
1178.
1179.
1180.
1181.
1182.
1183.
1184.
1185.
1186.
1187.
1188.
1189.
1190.
1191.
1192.
1193.
1194.
1195.
1196.
1197.
1198.
1199.
1200.
1201.
1202.
1203.
1204.
1205.
1206.
1207.
1208.
1209.
1210.
1211.
1212.
1213.
1214.
1215.
1216.
1217.
1218.
1219.
1220.
1221.
1222.
1223.
1224.
1225.
1226.
1227.
1228.
1229.
1230.
1231.
1232.
1233.
1234.
1235.
1236.
1237.
1238.
1239.
1240.
1241.
1242.
1243.
1244.
1245.
1246.
1247.
1248.
1249.
1250.
1251.
1252.
1253.
1254.
1255.
1256.
1257.
1258.
1259.
1260.
1261.
1262.
1263.
1264.
1265.
1266.
1267.
1268.
1269.
1270.
1271.
1272.
1273.
1274.
1275.
1276.
1277.
1278.
1279.
1280.
1281.
1282.
1283.
1284.
1285.
1286.
1287.
1288.
1289.
1290.
1291.
1292.
1293.
1294.
1295.
1296.
1297.
1298.
1299.
1300.
1301.
1302.
1303.
1304.
1305.
1306.
1307.
1308.
1309.
1310.
1311.
1312.
1313.
1314.
1315.
1316.
1317.
1318.
1319.
1320.
1321.
1322.
1323.
1324.
1325.
1326.
1327.
1328.
1329.
1330.
1331.
1332.
1333.
1334.
1335.
1336.
1337.
1338.
1339.
1340.
1341.
1342.
1343.
1344.
1345.
1346.
1347.
1348.
1349.
1350.
1351.
1352.
1353.
1354.
1355.
1356.
1357.
1358.
1359.
1360.
1361.
1362.
1363.
1364.
1365.
1366.
1367.
1368.
1369.
1370.
1371.
1372.
1373.
1374.
1375.
1376.
1377.
1378.
1379.
1380.
1381.
1382.
1383.
1384.
1385.
1386.
1387.
1388.
1389.
1390.
1391.
1392.
1393.
1394.
1395.
1396.
1397.
1398.
1399.
1400.
1401.
1402.
1403.
1404.
1405.
1406.
1407.
1408.
1409.
1410.
1411.
1412.
1413.
1414.
1415.
1416.
1417.
1418.
1419.
1420.
1421.
1422.
1423.
1424.
1425.
1426.
1427.
1428.
1429.
1430.
1431.
1432.
1433.
1434.
1435.
1436.
1437.
1438.
1439.
1440.
1441.
1442.
1443.
1444.
1445.
1446.
1447.
1448.
1449.
1450.
1451.
1452.
1453.
1454.
1455.
1456.
1457.
1458.
1459.
1460.
1461.
1462.
1463.
1464.
1465.
1466.
1467.
1468.
1469.
1470.
1471.
1472.
1473.
1474.
1475.
1476.
1477.
1478.
1479.
1480.
1481.
1482.
1483.
1484.
1485.
1486.
1487.
1488.
1489.
1490.
1491.
1492.
1493.
1494.
1495.
1496.
1497.
1498.
1499.
1500.
1501.
1502.
1503.
1504.
1505.
1506.
1507.
1508.
1509.
1510.
1511.
1512.
1513.
1514.
1515.
1516.
1517.
1518.
1519.
1520.
1521.
1522.
1523.
1524.
1525.
1526.
1527.
1528.
1529.
1530.
1531.
1532.
1533.
1534.
1535.
1536.
1537.
1538.
1539.
1540.
1541.
1542.
1543.
1544.
1545.
1546.
1547.
1548.
1549.
1550.
1551.
1552.
1553.
1554.
1555.
1556.
1557.
1558.
1559.
1560.
1561.
1562.
1563.
1564.
1565.
1566.
1567.
1568.
1569.
1570.
1571.
1572.
1573.
1574.
1575.
1576.
1577.
1578.
1579.
1580.
1581.
1582.
1583.
1584.
1585.
1586.
1587.
1588.
1589.
1590.
1591.
1592.
1593.
1594.
1595.
1596.
1597.
1598.
1599.
1600.
1601.
1602.
1603.
1604.
1605.
1606.
1607.
1608.
1609.
1610.
1611.
1612.
1613.
1614.
1615.
1616.
1617.
1618.
1619.
1620.
1621.
1622.
1623.
1624.
1625.
1626.
1627.
1628.
1629.
1630.
1631.
1632.
1633.
1634.
1635.
1636.
1637.
1638.
1639.
1640.
1641.
1642.
1643.
1644.
1645.
1646.
1647.
1648.
1649.
1650.
1651.
1652.
1653.
1654.
1655.
1656.
1657.
1658.
1659.
1660.
1661.
1662.
1663.
1664.
1665.
1666.
1667.
1668.
1669.
1670.
1671.
1672.
1673.
1674.
1675.
1676.
1677.
1678.
1679.
1680.
1681.
1682.
1683.
1684.
1685.
1686.
1687.
1688.
1689.
1690.
1691.
1692.
1693.
1694.
1695.
1696.
1697.
1698.
1699.
1700.
1701.
1702.
1703.
1704.
1705.
1706.
1707.
1708.
1709.
1710.
1711.
1712.
1713.
1714.
1715.
1716.
1717.
1718.
1719.
1720.
1721.
1722.
1723.
1724.
1725.
1726.
1727.
1728.
1729.
1730.
1731.
1732.
1733.
1734.
1735.
1736.
1737.
1738.
1739.
1740.
1741.
1742.
1743.
1744.
1745.
1746.
1747.
1748.
1749.
1750.
1751.
1752.
1753.
1754.
1755.
1756.
1757.
1758.
1759.
1760.
1761.
1762.
1763.
1764.
1765.
1766.
1767.
1768.
1769.
1770.
1771.
1772.
1773.
1774.
1775.
1776.
1777.
1778.
1779.
1780.
1781.
1782.
1783.
1784.
1785.
1786.
1787.
1788.
1789.
1790.
1791.
1792.
1793.
1794.
1795.
1796.
1797.
1798.
1799.
1800.
1801.
1802.
1803.
1804.
1805.
1806.
1807.
1808.
1809.
1810.
1811.
1812.
1813.
1814.
1815.
1816.
1817.
1818.
1819.
1820.
1821.
1822.
1823.
1824.
1825.
1826.
1827.
1828.
1829.
1830.
1831.
1832.
1833.
1834.
1835.
1836.
1837.
1838.
1839.
1840.
1841.
1842.
1843.
1844.
1845.
1846.
1847.
1848.
1849.
1850.
1851.
1852.
1853.
1854.
1855.
1856.
1857.
1858.
1859.
1860.
1861.
1862.
1863.
1864.
1865.
1866.
1867.
1868.
1869.
1870.
1871.
1872.
1873.
1874.
1875.
1876.
1877.
1878.
1879.
1880.
1881.
1882.
1883.
1884.
1885.
1886.
1887.
1888.
1889.
1890.
1891.
1892.
1893.
1894.
1895.
1896.
1897.
1898.
1899.
1900.
1901.
1902.
1903.
1904.
1905.
1906.
1907.
1908.
1909.
1910.
1911.
1912.
1913.
1914.
1915.
1916.
1917.
1918.
1919.
1920.
1921.
1922.
1923.
1924.
1925.
1926.
1927.
1928.
1929.
1930.
1931.
1932.
1933.
1934.
1935.
1936.
1937.
1938.
1939.
1940.
1941.
1942.
1943.
1944.
1945.
1946.
1947.
1948.
1949.
1950.
1951.
1952.
1953.
1954.
1955.
1956.
1957.
1958.
1959.
1960.
1961.
1962.
1963.
1964.
1965.
1966.
1967.
1968.
1969.
1970.
1971.
1972.
1973.
1974.
1975.
1976.
1977.
1978.
1979.
1980.
1981.
1982.
1983.
1984.
1985.
1986.
1987.
1988.
1989.
1990.
1991.
1992.
1993.
1994.
1995.
1996.
1997.
1998.
1999.
2000.
2001.
2002.
2003.
2004.
2005.
2006.
2007.
2008.
2009.
2010.
2011.
2012.
2013.
2014.
2015.
2016.
2017.
2018.
2019.
2020.
2021.
2022.
2023.
2024.
2025.
2026.
2027.
2028.
2029.
2030.
2031.
2032.
2033.
2034.
2035.
2036.
2037.
2038.
2039.
2040.
2041.
2042.
2043.
2044.
2045.
2046.
2047.
2048.
2049.
2050.
2051.
2052.
2053.
2054.
2055.
2056.
2057.
2058.
2059.
2060.
2061.
2062.
2063.
2064.
2065.
2066.
2067.
2068.
2069.
2070.
2071.
2072.
2073.
2074.
2075.
2076.
2077.
2078.
2079.
2080.
2081.
2082.
2083.
2084.
2085.
2086.
2087.
2088.
2089.
2090.
2091.
2092.
2093.
2094.
2095.
2096.
2097.
2098.
2099.
2100.
2101.
2102.
2103.
2104.
2105.
2106.
2107.
2108.
2109.
2110.
2111.
2112.
2113.
2114.
2115.
2116.
2117.
2118.
2119.
2120.
2121.
2122.
2123.
2124.
2125.
2126.
2127.
2128.
2129.
2130.
2131.
2132.
2133.
2134.
2135.
2136.
2137.
2138.
2139.
2140.
2141.
2142.
2143.
2144.
2145.
2146.
2147.
2148.
2149.
2150.
2151.
2152.
2153.
2154.
2155.
2156.
2157.
2158.
2159.
2160.
2161.
2162.
2163.
2164.
2165.
2166.
2167.
2168.
2169.
2170.
2171.
2172.
2173.
2174.
2175.
2176.
2177.
2178.
2179.
2180.
2181.
2182.
2183.
2184.
2185.
2186.
2187.
2188.
2189.
2190.
2191.
2192.
2193.
2194.
2195.
2196.
2197.
2198.
2199.
2200.
2201.
2202.
2203.
2204.
2205.
2206.
2207.
2208.
2209.
2210.
2211.
2212.
2213.
2214.
2215.
2216.
2217.
2218.
2219.
2220.
2221.
2222.
2223.
2224.
2225.
2226.
2227.
2228.
2229.
2230.
2231.
2232.
2233.
2234.
2235.
2236.
2237.
2238.
2239.
2240.
2241.
2242.
2243.
2244.
2245.
2246.
2247.
2248.
2249.
2250.
2251.
2252.
2253.
2254.
2255.
2256.
2257.
2258.
2259.
2260.
2261.
2262.
2263.
2264.
2265.
2266.
2267.
2268.
2269.
2270.
2271.
2272.
2273.
2274.
2275.
2276.
2277.
2278.
2279.
2280.
2281.
2282.
2283.

```

```
165.
166.         // 启用线程池
167.         ExecutorService exec = Executors.newFixedThreadPool(SIZE);
168.         for (int index = 1; index <= SIZE; index++) {
169.             exec.execute(new Download(index));
170.         }
171.         exec.shutdown();
172.
173.         long endTime = System.currentTimeMillis();
174.
175.         long timeLong = endTime - startTime;
176.
177.         System.out.println("下载时间: " + timeLong);
178.
179.     }
180. }
```

效果和上一个程序的效果差不多，只是时间上和内存资源占有率上有所提高，当然本机仅仅启动了几个线程，如果客户端启动更多线程，NIO的方式节约资源的效果是明显的，宕机概率小于阻塞IO方式很多。

5. 总结

其实NIO想写得更多，但是看到网络上已经有很多资料了，就不再展开了，非一篇所能尽述的了。当然了，NIO也是有场景的，比如适合与长连接的请求，以为NIO维护管道、缓存块、时间选择器等等也需要资源消耗的，而且比传统IO的对象们要重量级。所以原始IO也并不是一无是处，现在还是有很多socket中间件还是已然使用第二种方式。

代码在附件中~~~

[FilmServer.rar](#) (20.4 KB)
下载次数: 286

[查看图片附件](#)

[上海自考专/本科](#)

12个月,上海自考专/本科签约班 周期短,合格率高,文凭硬,学费低
[www.zili.cn](#)

Google 提供的广告



分享到:

◀ [Java分布式应用学习笔记01分布式Java应用和...](#) | [读金庸故事，品程序人生04韦小宝做事的启示](#) ▶

08:54 | [评论 / 浏览 \(39 / 8111\)](#) | 分类:[编程语言](#) | [相关推荐](#) [MORE](#)

评论

39 楼 [endiya](#) 23 分钟前

讲的不错,通俗易懂;顶.👍

38 楼 [angel243fly](#) 15 小时前

很不错的文章，让初学者也能看懂。
话说我正吃饭呢。。。

37 楼 [mouzhaonu1985](#) 18 小时前

这位大哥的例子真的让人很很胃口啊，不过确实挺形象的。重口味

36 楼 [houfeng0923](#) 昨天

novoland 写道

hi NIO 并非AIO，它是同步非阻塞IO，所以文章直接将NIO称呼为异步IO有点不妥

说的是呀。异步的AIO在jdk7中才刚发布。

35 楼 [object_object](#) 昨天

NIOServer 服务器类 的main 启动方法里面 加了个while (true)循环监听，调用server.listen();就已经在循环监听了，这里是不是 重复监听了啊？？？

34 楼 [novoland](#) 2011-09-14

hi NIO 并非AIO，它是同步非阻塞IO，所以文章直接将NIO称呼为异步IO有点不妥

33 楼 [jingyunxia198111](#) 2011-09-04


 小伙子很不错啊


32 楼 [firstlove8856](#) 2011-08-21

这个篇文章有点意思，值得回味。呵呵

31 楼 [suhuanzheng7784877](#) 2011-08-15


tan4836128 写道


很实惠的文章，性价比高高的阅读价值，没理由不顶一顶

至于举例，通俗易懂很好，读不下去的朋友只能哪儿凉快哪儿呆着去了

兄弟你言重了，没到那种高度。而且有些人适合比较严肃的风格，有些人适合比较风趣的风格。个人品味不太一样吧。

30 楼 [tan4836128](#) 2011-08-12

很实惠的文章，性价比高高的阅读价值，没理由不顶一顶

至于举例，通俗易懂很好，读不下去的朋友只能哪儿凉快哪儿呆着去了

29 楼 [OLIVER_kahn111](#) 2011-08-10

不是吧，博主这都能想到。自愧不如，哈哈

28 楼 [daur](#) 2011-08-09

呀，楼主屎例给我留下深刻印象哇，大大的up~~

27 楼 [lxhna](#) 2011-08-02

<http://suhuanzheng7784877.iteye.com/blog/1122131>
好东西啊

26 楼 [suhuanzheng7784877](#) 2011-07-27

yuediaoyuan0809 写道

上厕所这种事，怎能公开呢。

恩，只是个比喻罢了。。。。。

25 楼 [yuediaoyuan0809](#) 2011-07-27

上厕所这种事，怎能公开呢。

24 楼 [suhuanzheng7784877](#) 2011-07-19

自我检讨：

在代码中NIOServer.java中的main函数有一个死循环，死循环的内容是server.listen();

这个死循环应该去掉。
直接写成`server.listen()`;

多谢Dping 兄提出的问题，感激不尽.....

23 楼 [柏新星](#) 2011-07-16



不错正在看你的代码呢 呵呵

22 楼 [kevinming](#) 2011-07-15

好文章，喜欢这张循序渐进的文章，而且寓意深刻👍

21 楼 [suhuanzheng7784877](#) 2011-07-15

JDK7支持新的AIO的方式直接利用操作系统接口完成网络、文件的读写。

而像P2P那些软件底层是C++代码，当然了人家也不会让咱们看到代码的，他们的底层核心原理就是使用AIO更好的文件读写、网络传播数据包、缓存优化等。

参考资料：

<http://www.newsmth.net/bbsanc.php?path=%2Fgroups%2Fcomp.faq%2FJava%2Fprinciple%2Fjdk7%2FM.1258247388.N0>

20 楼 [suhuanzheng7784877](#) 2011-07-15

jd2bs 写道

楼主的例子是下载多个文件

下次可以介绍下将大文件分块然后多线程下载 最后合并为一个完整文件的细节

就像传统的FlashGet，NetAnts；比较新的迅雷，在通信细节上各有什么不同

还可以讲讲p2p的BT，Emule
如果都讲清楚了 估计都能写一本书了

呵呵，兄弟抬举了，真没到那个程度，而且P2P是另一种思维，和单点多线程多任务Server下载十分不同。

自我检讨一下：

这个例子实在比较恶心，本人下次绝不那么举例子了。

实例比较基础，像断点续传、缓存取区域优化等等都没涉及到。

当然了这些内容也不是一两篇blog能说清楚的。

谢谢大家抬举，这个帖子能top这么多天.....实在是本人没预料到的。。。

发表评论



[您还没有登录,请您登录后再发表评论](#)