****NIO主要原理和适用****

NIO 有一个主要的类Selector,这个类似一个观察者，只要我们把需要探知的socketchannel告诉Selector,我们接着做别的事情，当有 事件发生时，他会通知我们，传回一组SelectionKey,我们读取这些Key,就会获得我们刚刚注册过的socketchannel,然后，我们从 这个Channel中读取数据，放心，包准能够读到，接着我们可以处理这些数据。

Selector内部原理实际是在做一个对所注册的channel的轮询访问，不断的轮询(目前就这一个算法)，一旦轮询到一个channel有所注册的事情发生，比如数据来了，他就会站起来报告，交出一把钥匙，让我们通过这把钥匙来读取这个channel的内容。

jdk供的无阻塞I/O（NIO）有效解决了多线程服务器存在的线程开销问题，但在使用上略显得复杂一些。在NIO中使用多线程，主要目的已不是为了应对 每个客户端请求而分配独立的服务线程，而是通过多线程充分使用用多个CPU的处理能力和处理中的等待时间，达到提高服务能力的目的。

这段时间在研究NIO，写篇博客来记住学过的东西。还是从最简单的Hello World开始，client多线程请求server端，server接收client的名字，并返回Hello! ＋名字的字符格式给client。当然实际应用并不这么简单，实际可能是访问文件或者数据库获取信息返回给client。非阻塞的NIO有何神秘之处？

代 码：

****1）server端代码****

1. **public** **class** HelloWorldServer {
3. **static** **int** BLOCK = 1024;
4. **static** String name = "";
5. **protected** Selector selector;
6. **protected** ByteBuffer clientBuffer = ByteBuffer.allocate(BLOCK);
7. **protected** CharsetDecoder decoder;
8. **static** CharsetEncoder encoder = Charset.forName("GB2312").newEncoder();
10. **public** HelloWorldServer(**int** port) **throws** IOException {
11. selector = **this**.getSelector(port);
12. Charset charset = Charset.forName("GB2312");
13. decoder = charset.newDecoder();
14. }
16. // 获取Selector
17. **protected** Selector getSelector(**int** port) **throws** IOException {
18. ServerSocketChannel server = ServerSocketChannel.open();
19. Selector sel = Selector.open();
20. server.socket().bind(**new** InetSocketAddress(port));
21. server.configureBlocking(**false**);
22. server.register(sel, SelectionKey.OP\_ACCEPT);
23. **return** sel;
24. }
26. // 监听端口
27. **public** **void** listen() {
28. **try** {
29. **for** (;;) {
30. selector.select();
31. Iterator iter = selector.selectedKeys().iterator();
32. **while** (iter.hasNext()) {
33. SelectionKey key = (SelectionKey) iter.next();
34. iter.remove();
35. process(key);
36. }
37. }
38. } **catch** (IOException e) {
39. e.printStackTrace();
40. }
41. }
43. // 处理事件
44. **protected** **void** process(SelectionKey key) **throws** IOException {
45. **if** (key.isAcceptable()) { // 接收请求
46. ServerSocketChannel server = (ServerSocketChannel) key.channel();
47. SocketChannel channel = server.accept();
48. //设置非阻塞模式
49. channel.configureBlocking(**false**);
50. channel.register(selector, SelectionKey.OP\_READ);
51. } **else** **if** (key.isReadable()) { // 读信息
52. SocketChannel channel = (SocketChannel) key.channel();
53. **int** count = channel.read(clientBuffer);
54. **if** (count > 0) {
55. clientBuffer.flip();
56. CharBuffer charBuffer = decoder.decode(clientBuffer);
57. name = charBuffer.toString();
58. // System.out.println(name);
59. SelectionKey sKey = channel.register(selector,
60. SelectionKey.OP\_WRITE);
61. sKey.attach(name);
62. } **else** {
63. channel.close();
64. }
66. clientBuffer.clear();
67. } **else** **if** (key.isWritable()) { // 写事件
68. SocketChannel channel = (SocketChannel) key.channel();
69. String name = (String) key.attachment();
71. ByteBuffer block = encoder.encode(CharBuffer
72. .wrap("Hello !" + name));

75. channel.write(block);
77. //channel.close();
79. }
80. }
82. **public** **static** **void** main(String[] args) {
83. **int** port = 8888;
84. **try** {
85. HelloWorldServer server = **new** HelloWorldServer(port);
86. System.out.println("listening on " + port);
88. server.listen();
90. } **catch** (IOException e) {
91. e.printStackTrace();
92. }
93. }
94. }

****2）client端代码****

1. **public** **class** HelloWorldClient {
3. **static** **int** SIZE = 10;
4. **static** InetSocketAddress ip = **new** InetSocketAddress("localhost", 8888);
5. **static** CharsetEncoder encoder = Charset.forName("GB2312").newEncoder();
7. **static** **class** Message **implements** Runnable {
8. **protected** String name;
9. String msg = "";
11. **public** Message(String index) {
12. **this**.name = index;
13. }
15. **public** **void** run() {
16. **try** {
17. **long** start = System.currentTimeMillis();
18. //打开Socket通道
19. SocketChannel client = SocketChannel.open();
20. //设置为非阻塞模式
21. client.configureBlocking(**false**);
22. //打开选择器
23. Selector selector = Selector.open();
24. //注册连接服务端socket动作
25. client.register(selector, SelectionKey.OP\_CONNECT);
26. //连接
27. client.connect(ip);
28. //分配内存
29. ByteBuffer buffer = ByteBuffer.allocate(8 \* 1024);
30. **int** total = 0;
32. \_FOR: **for** (;;) {
33. selector.select();
34. Iterator iter = selector.selectedKeys().iterator();
36. **while** (iter.hasNext()) {
37. SelectionKey key = (SelectionKey) iter.next();
38. iter.remove();
39. **if** (key.isConnectable()) {
40. SocketChannel channel = (SocketChannel) key
41. .channel();
42. **if** (channel.isConnectionPending())
43. channel.finishConnect();
44. channel
45. .write(encoder
46. .encode(CharBuffer.wrap(name)));
48. channel.register(selector, SelectionKey.OP\_READ);
49. } **else** **if** (key.isReadable()) {
50. SocketChannel channel = (SocketChannel) key
51. .channel();
52. **int** count = channel.read(buffer);
53. **if** (count > 0) {
54. total += count;
55. buffer.flip();
57. **while** (buffer.remaining() > 0) {
58. **byte** b = buffer.get();
59. msg += (**char**) b;
61. }
63. buffer.clear();
64. } **else** {
65. client.close();
66. **break** \_FOR;
67. }
68. }
69. }
70. }
71. **double** last = (System.currentTimeMillis() - start) \* 1.0 / 1000;
72. System.out.println(msg + "used time :" + last + "s.");
73. msg = "";
74. } **catch** (IOException e) {
75. e.printStackTrace();
76. }
77. }
78. }
80. **public** **static** **void** main(String[] args) **throws** IOException {
82. String names[] = **new** String[SIZE];
84. **for** (**int** index = 0; index < SIZE; index++) {
85. names[index] = "jeff[" + index + "]";
86. **new** Thread(**new** Message(names[index])).start();
87. }
89. }
90. }

原文链接：http://itemdetail.iteye.com/blog/787149