[代码全屏查看]-JAVA NIO实现Socket服务器与客户端功能

```
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.CancelledKeyException;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
public class Server {
    private Selector selector = getSelector();
    private ServerSocketChannel ss = null;
    private ThreadPoolExecutor threadPool = new ThreadPoolExecutor(10,
10, 500, TimeUnit.MILLISECONDS,
            new ArrayBlockingQueue<Runnable>(20));
    private Map<Integer, SelectionKey> selectionKeyMap = new
ConcurrentHashMap<>();
    private Map<Integer, List<Object>> responseMessageQueue = new
ConcurrentHashMap<>();
    private volatile boolean run = true;
    private volatile boolean isClose = false;
    public Selector getSelector() {
        try {
            return Selector.open();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        return null;
    }
    /**
     * 创建非阻塞服务器绑定5555端口
     */
    public Server() {
        try {
            ss = ServerSocketChannel.open();
            ss.bind(new InetSocketAddress(5555));
            ss.configureBlocking(false);
            if (selector == null) {
                selector = Selector.open();
            }
            ss.register(selector, SelectionKey.OP_ACCEPT);
        } catch (Exception e) {
            e.printStackTrace();
            close();
        }
    }
    public boolean isClose() {
        return isClose;
    }
    /**
     * 关闭服务器
     */
    private void close() {
        run = false;
        isClose = true;
        threadPool.shutdown();
        try {
            if (ss != null) {
                ss.close();
            }
            if (selector != null) {
                selector.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
    }
    /**
     * 启动选择器监听客户端事件
     */
    private void start() {
        threadPool.execute(new Runnable() {
            @Override
            public void run() {
                try {
                    while (run) {
                        if (selector.select(10) == 0) {
                            continue;
                        }
                        Iterator<SelectionKey> iterator =
selector.selectedKeys().iterator();
                        while (iterator.hasNext()) {
                            SelectionKey selectedKey =
iterator.next();
                            iterator.remove();
                            try {
                                if (selectedKey.isReadable()) {
                                    if
(selectionKeyMap.get(selectedKey.hashCode()) != selectedKey) {
                                        selectionKeyMap.put(selectedKe
y.hashCode(), selectedKey);
                                        threadPool.execute(new
ReadClientSocketHandler(selectedKey));
                                    }
                                } else if (selectedKey.isWritable()) {
                                    SocketChannel serverSocketChannel
= (SocketChannel) selectedKey.channel();
                                    selectedKey.interestOps(SelectionK
ey.OP_READ);
                                    List<Object> list =
responseMessageQueue.get(selectedKey.hashCode());
                                    if (list == null) {
                                        list = new LinkedList<Object>
```

```
();
                                responseMessageQueue.put(selec
tedKey.hashCode(), list);
                            }
                            while (list.size() > 0) {
                                Object responseMessage =
list.remove(0);
                                if (responseMessage != null) {
                                    threadPool.execute(new
WriteClientSocketHandler(serverSocketChannel,
                                            responseMessage));
                                }
                            }
                        } else if (selectedKey.isAcceptable())
{
                            ServerSocketChannel ssc =
(ServerSocketChannel) selectedKey.channel();
                            SocketChannel clientSocket =
ssc.accept();
                            if (clientSocket != null) {
                                clientSocket.configureBlocking(
false);
                                clientSocket.register(selector
, SelectionKey.OP_READ | SelectionKey.OP_WRITE);
                            }
                        }
                    } catch (CancelledKeyException cc) {
                        selectedKey.cancel();
                        int hashCode = selectedKey.hashCode();
                        selectionKeyMap.remove(hashCode);
                        responseMessageQueue.remove(hashCode);
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            close();
        }
```

```
            }
        });
    }
    /**
     * 响应数据给客户端线程
     *
     * @author haoguo
     *
     */
    private class WriteClientSocketHandler implements Runnable {
        SocketChannel client;
        Object respnoseMessage;
        WriteClientSocketHandler(SocketChannel client, Object
respnoseMessage) {
            this.client = client;
            this.respnoseMessage = respnoseMessage;
        }
        @Override
        public void run() {
            byte[] responseByteData = null;
            String logResponseString = "";
            if (respnoseMessage instanceof byte[]) {
                responseByteData = (byte[]) respnoseMessage;
                logResponseString = new String(responseByteData);
            } else if (respnoseMessage instanceof String) {
                logResponseString = (String) respnoseMessage;
                responseByteData = logResponseString.getBytes();
            }
            if (responseByteData == null || responseByteData.length ==
0) {
                System.out.println("响应的数据为空");
                return;
            }
            try {
                client.write(ByteBuffer.wrap(responseByteData));
                System.out.println("server响应客户端[" +
client.keyFor(selector).hashCode() + "]数据 :[" + logResponseString
                        + "]");
```

```
            } catch (IOException e) {
                e.printStackTrace();
                try {
                    SelectionKey selectionKey =
client.keyFor(selector);
                    if (selectionKey != null) {
                        selectionKey.cancel();
                        int hashCode = selectionKey.hashCode();
                        responseMessageQueue.remove(hashCode);
                    }
                    if (client != null) {
                        client.close();
                    }
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        }
    }
    /**
     * 读客户端发送数据线程
     *
     * @author haoguo
     *
     */
    private class ReadClientSocketHandler implements Runnable {
        private SocketChannel client;
        private ByteBuffer tmp = ByteBuffer.allocate(1024);
        private SelectionKey selectionKey;
        int hashCode;
        ReadClientSocketHandler(SelectionKey selectionKey) {
            this.selectionKey = selectionKey;
            this.client = (SocketChannel) selectionKey.channel();
            this.hashCode = selectionKey.hashCode();
        }
        @Override
        public void run() {
            try {
```

```
                tmp.clear();
                byte[] data = new byte[0];
                int len = -1;
                while ((len = client.read(tmp)) > 0) {
                    data = Arrays.copyOf(data, data.length + len);
                    System.arraycopy(tmp.array(), 0, data, data.length
- len, len);
                    tmp.rewind();
                }
                if (data.length == 0) {
                    return;
                }
                String readData = new String(data);
                System.out.println("接收到客户端[" + hashCode + "]数据 :
[" + readData.substring(0, 3) + "]");
                // dosomthing
                byte[] response = ("response" + readData.substring(0,
3)).getBytes();
                List<Object> list =
responseMessageQueue.get(hashCode);
                list.add(response);
                client.register(selector, SelectionKey.OP_WRITE);
                // client.register(selector, SelectionKey.OP_WRITE,
response);
            } catch (IOException e) {
                System.out.println("客户端[" + selectionKey.hashCode() +
"]关闭了连接");
                try {
                    SelectionKey selectionKey =
client.keyFor(selector);
                    if (selectionKey != null) {
                        selectionKey.cancel();
                    }
                    if (client != null) {
                        client.close();
                    }
                } catch (IOException e1) {
                    e1.printStackTrace();
```

```
                }
            } finally {
                selectionKeyMap.remove(hashCode);
            }
        }
    }
    public static void main(String[] args) {
        Server server = new Server();
        server.start();
    }
}
```