



链滴

3. Netty 初认识 --NIO 入门 --- 示例 (完整代码)

作者: [289306290](#)

原文链接: <https://ld246.com/article/1582789368221>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

首先启动TimeServer 服务端开始监听

其次启动TimeClient 连接服务端,并发送固定指令QUERY TIME ORDER查询当前时间

最后服务端返回当前时间

具体代码如下(可运行)

TimeServer.java

```
package club.wujingjian.com.wujingjian.nio.server;

public class TimeServer {

    public static void main(String[] args) {
        int port = 9090;

        if (args != null && args.length > 0) {
            try {
                port = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                e.printStackTrace();
            }
        }
        MultiplexerTimeServer timeServer = new MultiplexerTimeServer(port);

        new Thread(timeServer, "NIO-MultiplexerTimeServer-001").start();
    }
}
```

MultiplexerTimeServer.java

```
package club.wujingjian.com.wujingjian.nio.server;

import club.wujingjian.com.wujingjian.nio.Util;

import java.io.IOException;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.Date;
import java.util.Iterator;
import java.util.Set;

public class MultiplexerTimeServer implements Runnable {

    private Selector selector;

    private ServerSocketChannel servChannel;
```

```

private volatile boolean stop;

/**
 * 初始化多路复用器、绑定监听端口
 * 进行资源初始化,创建多路复用器Selector、ServerSocketChannel,
 * 对Channel和TCP参数进行配置, (例如,将ServerSocketChannel设置为异步非阻塞模式,它的back
 * 设置为1024)
 */
public MultiplexerTimeServer(int port) {
    try {
        selector = Selector.open();
        servChannel = ServerSocketChannel.open();
        servChannel.configureBlocking(false);
        servChannel.socket().bind(new InetSocketAddress(port), 1024);
        //将ServerSocketChannel注册到Selector,监听SelectionKey.OP_ACCEPT操作位
        servChannel.register(selector, SelectionKey.OP_ACCEPT);
        System.out.println(Util.nowDate() + "The time server is start in port : " + port);
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
    }
}

public void stop(){
    this.stop = true;
}
@Override
public void run() {
    while (!stop) {
        /**
         * 循环遍历selector,休眠时间为1s
         */
        try {
            selector.select(1000);
            Set<SelectionKey> selectedKeys = selector.selectedKeys();
            Iterator<SelectionKey> it = selectedKeys.iterator();
            SelectionKey key = null;
            while (it.hasNext()) {
                key = it.next();
                it.remove();
                try {
                    handleInput(key);
                } catch (Exception e) {
                    key.cancel();
                    if (key.channel() != null) {
                        key.channel().close();
                    }
                }
            }
        } catch (Throwable throwable) {
            throwable.printStackTrace();
        }
    }
}
}

```

//多路复用器关闭后,所有注册在上面的channel 和pipe 等资源都会被自动去注册并关闭,所以不要重复释放资源

```
    if (selector != null) {
        try {
            selector.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private void handleInput(SelectionKey key) throws IOException {
    if (key.isValid()) {
        /**
         *处理新接入的客户端请求信息,根据SelectionKey的操作位进行判断即可获知网络事件的类型
         */
        if (key.isAcceptable()) {
            //接收新连接
            ServerSocketChannel ssc = (ServerSocketChannel) key.channel();
            //通过ServerSocketChannel的accept()接收客户端的连接请求并创建SocketChannel实例
            相当于三次握手
            SocketChannel sc = ssc.accept();
            sc.configureBlocking(false);
            //添加新连接到多路复用器Selector
            sc.register(selector, SelectionKey.OP_READ);
        }
        if (key.isReadable()) {
            /**
             * 读取客户端的请求消息,首先创建一个ByteBuffer,由于事先无法得知客户端发送的码流大
             作为例程, 我们开辟一个1K的缓冲区,
             * 然后调用SocketChannel的read方法读取请求码流.
             * 注意: 由于我们已经将SocketChannel设置为异步非阻塞模式,因此他的read是非阻塞的,
             ocketChannel.read()的返回值有以下三种情况
             * 返回值大于0: 读取到了字节,对字节进行编解码.
             * 返回值等于0: 没有读取到字节,属于正常场景,忽略
             * 返回值为-1: 链路已经关闭,需要关闭SocketChannel,释放资源
             */
            SocketChannel sc = (SocketChannel) key.channel();
            ByteBuffer readBuffer = ByteBuffer.allocate(1024);
            int readBytes = sc.read(readBuffer);
            if (readBytes > 0) {
                //读取到码流以后,我们进行解码,首先对readBuffer进行flip操作,作用是将缓冲区当前的
                limit设置为position,position设置为0,用于后续对缓冲区的读取操作
                readBuffer.flip();
                //根据缓冲区可读的字节个数创建字节数组
                byte[] bytes = new byte[readBuffer.remaining()];
                //将缓冲区可读的字节数组复制到新创建的字节数组bytes中.
                readBuffer.get(bytes);
                String body = new String(bytes, "UTF-8");
                System.out.println(Util.nowDate() + "The time server receive order : " + body);
                //如果请求指令是"QUERY TIME ORDER" 则把服务器的当前时间编码后返回给客户端
            }
        }
    }
}
```

```

        String currentTime = "QUERY TIME ORDER".equalsIgnoreCase(body) ? new Date(
system.currentTimeMillis()).toString() : "BAD ORDER";
        System.out.println(Util.nowDate() + "服务器准备返回时间为:" + currentTime);
        doWrite(sc, currentTime);
    } else if (readBytes < 0) {
        //对端链路关闭
        key.cancel();
        sc.close();
    } else{
        // 读取到0字节,忽略
    }
}
}
}

```

```

private void doWrite(SocketChannel channel, String response) throws IOException{
    if (response != null && response.trim().length() > 0) {
        //将字符串编码成字节数组,
        byte [] bytes = response.getBytes();
        //根据字节数组容量创建ByteBuffer
        ByteBuffer writeBuffer = ByteBuffer.allocate(bytes.length);
        //将字节数组复制到缓冲区中
        writeBuffer.put(bytes);
        //将缓冲区的position设置为0,以便准备输出
        writeBuffer.flip();
        /**
        *将缓冲区中的字节数组发送出去
        * 注意: 由于SocketChannel是异步非阻塞的,它并不能保证一次性把需要发送的字节数组发
       完毕,此时会出现"写半包"的问题,
        * 我们需要注册写操作,不断轮询Selector将没有发送完的ByteBuffer发送完毕,可以通过By
        eBuffer的hasRemaining()方法
        * 判断消息是否发送完成。此处仅仅是个入门示例,没有处理"写半包"的情况.
        */
        channel.write(writeBuffer);
        if (!writeBuffer.hasRemaining()) {
            System.out.println(Util.nowDate() + "发送成功");
        }
    }
}
}
}

```

下面是客户端代码

TimeClient.java

```
package club.wujingjian.com.wujingjian.nio.client;
```

```

public class TimeClient {
    public static void main(String[] args) {
        int port = 9090;
        if (args != null && args.length > 0) {
            try {

```

```

        port = Integer.parseInt(args[0]);
    } catch (NumberFormatException e) {
        e.printStackTrace();
    }
}
new Thread(new TimeClientHandle("127.0.0.1",port, "TimeClient-001")).start();
}
}

```

TimeClientHandle.java

```
package club.wujingjian.com.wujingjian.nio.client;
```

```
import club.wujingjian.com.wujingjian.nio.Util;
```

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;
```

```
public class TimeClientHandle implements Runnable {
```

```
    private String host;
    private int port ;
    private Selector selector;
    private SocketChannel socketChannel;
    private volatile boolean stop;
```

```
    /**
     * 构造函数初始化NIO多路复用器和SocketChannel对象，并将socketChannel设置为异步非阻塞
     式
```

```
    * @param host
    * @param port
    */
```

```
    public TimeClientHandle(String host, int port) {
        this.host = host == null ? "127.0.0.1": host;
        this.port = port;
        try {
            selector = Selector.open();
            socketChannel = SocketChannel.open();
            socketChannel.configureBlocking(false);
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

```
@Override
public void run() {
```

```

try {
    //发送连接请求,作为示例,连接是成功的,所以不需要做重连操作,实际代码中,可能要考虑重连,
    能放到while循环外面
    doConnect();
} catch (IOException e) {
    e.printStackTrace();
    System.exit(1);
}
while (!stop) {
    try {
        selector.select(1000);
        Set<SelectionKey> selectedKeys = selector.selectedKeys();
        Iterator<SelectionKey> it = selectedKeys.iterator();
        SelectionKey key = null;
        //当有就绪的Channel时候执行handleInput方法
        while (it.hasNext()) {
            key = it.next();
            it.remove();
            try {
                handleInput(key);
            } catch (Exception e) {
                if (key != null) {
                    key.cancel();
                    if (key.channel() != null) {
                        key.channel().close();
                    }
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
}

```

//多路复用器关闭后,所有注册在上面的channel和pipe等资源都会被自动去注册并关闭,所以不要重复释放资源

```

if (selector != null) {
    try {
        selector.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

```

private void handleInput(SelectionKey key) throws IOException {
    if (key.isValid()) {
        //判断是否连接成功
        SocketChannel sc = (SocketChannel) key.channel();
        if (key.isConnectable()) { //如果SelectionKey处于连接状态,说明服务端已经返回ACK应答
            这是我们需对结果进行判断
            if (sc.finishConnect()) { //如果sc.finishConnect()返回true,说明客户端连接成功,
                System.out.println(Util.nowDate() + "客户端连接成功");
            }
        }
    }
}

```

```

        sc.register(selector, SelectionKey.OP_READ); //将SocketChannel注册到多路复用器上
        注册SelectionKey.OP_READ操作位,监听网络读操作,
        doWrite(sc); //然后发送请求消息给服务端
    } else {
        System.out.println("连接失败,进程退出");
        System.exit(1); //连接失败,进程退出
    }
}

if (key.isReadable()) {
    ByteBuffer readBuffer = ByteBuffer.allocate(1024);
    int readBytes = sc.read(readBuffer);
    if (readBytes > 0) {
        readBuffer.flip();
        byte[] bytes = new byte[readBuffer.remaining()];
        readBuffer.get(bytes);
        String body = new String(bytes, "UTF-8");
        System.out.println(Util.nowDate() + "Now is ." + body);
        this.stop = true;
    } else if (readBytes < 0) {
        //对端链路关闭
        key.cancel();
        sc.close();
    } else {
        //读到0 字节,忽略
    }
}
}
}

private void doConnect() throws IOException {
    //如果直连成功,则注册到多路复用器上,发送请求消息,读应答
    if (socketChannel.connect(new InetSocketAddress(host, port))) {
        socketChannel.register(selector, SelectionKey.OP_READ);
        doWrite(socketChannel);
    } else {
        /**
         * 如果没有直接连接成功,则说明服务端没有返回TCP握手应答消息,但这并不代表连接失败.
         * 我们需要将SocketChannel注册到多路复用器Selector上,注册SelectionKey.OP_CONNEC
         * 当服务端返回TCP syn-ack消息后,Selector就能够轮询到这个SocketChannel处于连接就
         状态
         */
        socketChannel.register(selector, SelectionKey.OP_CONNECT);
    }
}

private void doWrite(SocketChannel sc) throws IOException {
    byte[] req = "QUERY TIME ORDER".getBytes();
    ByteBuffer writeBuffer = ByteBuffer.allocate(req.length);
    writeBuffer.put(req);
    writeBuffer.flip();
    sc.write(writeBuffer); //此时可能会存在"半包写"问题,实际工作中要考虑
    if (!writeBuffer.hasRemaining()) { //此时如果缓冲区中消息全部发送完成, 打印如下消息Send

```

```
Order 2 server succeed.
```

```
        System.out.println(Util.nowDate() + "Send order 2 server succeed.");  
    }  
}  
}
```

先允许服务器端，后允许客户端代码最后输出分别为：



```
TimeServer x TimeClient x  
/Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java ...  
时间戳 1582788183431 The time server is start in port : 9090  
时间戳 1582788188464 The time server receive order : QUERY TIME ORDER  
时间戳 1582788188464 服务器准备返回时间为 :Thu Feb 27 15:23:08 CST 2020  
时间戳 1582788188465 发送成功  
|  
  
TimeServer x TimeClient x  
/Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java ...  
时间戳 1582788188456 客户端连接成功  
时间戳 1582788188460 Send order 2 server succeed.  
时间戳 1582788188466 Now is :Thu Feb 27 15:23:08 CST 2020  
  
Process finished with exit code 0
```