链滴

6.Netty 初认识 -- 粘包拆包 LineBasedFrameDecoder-- 示例二

作者：289306290

原文链接：https://ld246.com/article/1583118747186

来源网站：链滴

TimeServer.java

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

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;
import io.netty.handler.codec.LineBasedFrameDecoder;
import io.netty.handler.codec.string.StringDecoder;

import javax.sound.sampled.Line;

public class TimeServer {

    public void bind(int port) throws Exception {
        // 配置服务端的NIO线程组,他们实际上就是Reactor线程组,专门用于网络事件的处理,
        //一个用于服务端接收客户端的连接,另一个用于进行SocketChannel的网络读写
        EventLoopGroup bossGroup = new NioEventLoopGroup();
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            //ServerBootstrap 是用于启动NIO服务端的辅助启动类,降低服务端的开发复杂度
            ServerBootstrap b = new ServerBootstrap();
            b.group(bossGroup, workerGroup).channel(NioServerSocketChannel.class)
                    .option(ChannelOption.SO_BACKLOG, 1024)
                    .childHandler(new ChildChannelHandler());

            //绑定端口,同步等待成功
            ChannelFuture f = b.bind(port).sync();

            //等待服务端监听端口关闭
            f.channel().closeFuture().sync();
        } finally {
            //优雅退出,释放线程池资源
            bossGroup.shutdownGracefully();
            workerGroup.shutdownGracefully();
        }
    }

    private class  ChildChannelHandler extends ChannelInitializer<SocketChannel> {

        @Override
        protected void initChannel(SocketChannel socketChannel) throws Exception {
            //利用LineBasedFrameDecoder 和StringDecoder粘包拆包
            socketChannel.pipeline().addLast(new LineBasedFrameDecoder(1024));
            socketChannel.pipeline().addLast(new StringDecoder());
            socketChannel.pipeline().addLast(new TimeServerHandler());
        }
    }
```

```
    public static void main(String[] args) throws Exception {
        int port = 8080;
        if (args != null && args.length > 0) {
            try {
                port = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                //采用默认值
            }
        }
        new TimeServer().bind(port);
    }
}
```

TimeServerHandler.java

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

import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelHandlerAdapter;
import io.netty.channel.ChannelHandlerContext;

import java.util.Date;

//对网络事件进行读写操作
public class TimeServerHandler extends ChannelHandlerAdapter {

    private int counter;
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        /*ByteBuf buf = (ByteBuf) msg;
        byte[] req = new byte[buf.readableBytes()];
        buf.readBytes(req);
        String body = new String(req, "UTF-8");
        System.out.println("The time server receive order :" + body);
        String currentTime = "QUERY TIME ORDER".equalsIgnoreCase(body) ? new Date(System.
urrentTimeMillis()).toString() : "BAD ORDER";
        ByteBuf resp = Unpooled.copiedBuffer(currentTime.getBytes());
        ctx.write(resp);
        ctx.flush();*/


        //每读到一条消息后,就计数一次,然后发送应答消息给客户端
        /*ByteBuf buf = (ByteBuf) msg;
        byte[] req = new byte[buf.readableBytes()];
        buf.readBytes(req);
        String body = new String(req, "UTF-8").substring(0,req.length - System.getProperty("line.
eparator").length());
        System.out.println("The time server receive order :" + body + " ; the counter is :" +  ++c
unter);
        String currentTime = "QUERY TIME ORDER".equalsIgnoreCase(body) ? new Date(System.
urrentTimeMillis()).toString() : "BAD ORDER";
        currentTime = currentTime + System.getProperty("line.separator");
```

```
        ByteBuf resp = Unpooled.copiedBuffer(currentTime.getBytes());
        ctx.write(resp);
        ctx.flush();*/


        //粘包拆包
        String body = (String) msg;
       /* byte[] req = new byte[buf.readableBytes()];
        buf.readBytes(req);
        String body = new String(req, "UTF-8").substring(0,req.length - System.getProperty("line.
eparator").length());*/
        System.out.println("The time server receive order :" + body + " ; the counter is :" +  ++c
unter);
        String currentTime = "QUERY TIME ORDER".equalsIgnoreCase(body) ? new Date(System.
urrentTimeMillis()).toString() : "BAD ORDER";
        currentTime = currentTime + System.getProperty("line.separator");
        ByteBuf resp = Unpooled.copiedBuffer(currentTime.getBytes());
        ctx.write(resp);
        ctx.flush();
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Excepti
n {
        ctx.close();
    }

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {
        ctx.flush();
    }
}
```

TimeClient.java

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

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioSocketChannel;
import io.netty.handler.codec.LineBasedFrameDecoder;
import io.netty.handler.codec.string.StringDecoder;

public class TimeClient {
    public void connect(int port,String host) throws Exception {
        //配置客户端NIO线程组
        EventLoopGroup group = new NioEventLoopGroup();
        try {
```

```
            Bootstrap b = new Bootstrap();
            b.group(group).channel(NioSocketChannel.class)
                    .option(ChannelOption.TCP_NODELAY, true)
                    .handler(new ChannelInitializer<SocketChannel>() {
                        @Override
                        protected void initChannel(SocketChannel socketChannel) throws Exception {
                            socketChannel.pipeline().addLast(new LineBasedFrameDecoder(1024));
                            socketChannel.pipeline().addLast(new StringDecoder());
                            socketChannel.pipeline().addLast(new TimeClientHandler());
                        }
                    });
            //发起异步连接操作
            ChannelFuture f = b.connect(host, port).sync();

            //等等客户端链路关闭
            f.channel().closeFuture().sync();
        } finally {
            //优雅退出,释放NIO线程组
            group.shutdownGracefully();
        }
    }


    public static void main(String[] args) throws Exception{
        int port = 8080;
        if (args != null && args.length > 0) {
            try {
                port = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                e.printStackTrace();
            }
        }
        new TimeClient().connect(port, "127.0.0.1");
    }
}
```

TimeClientHandler.java

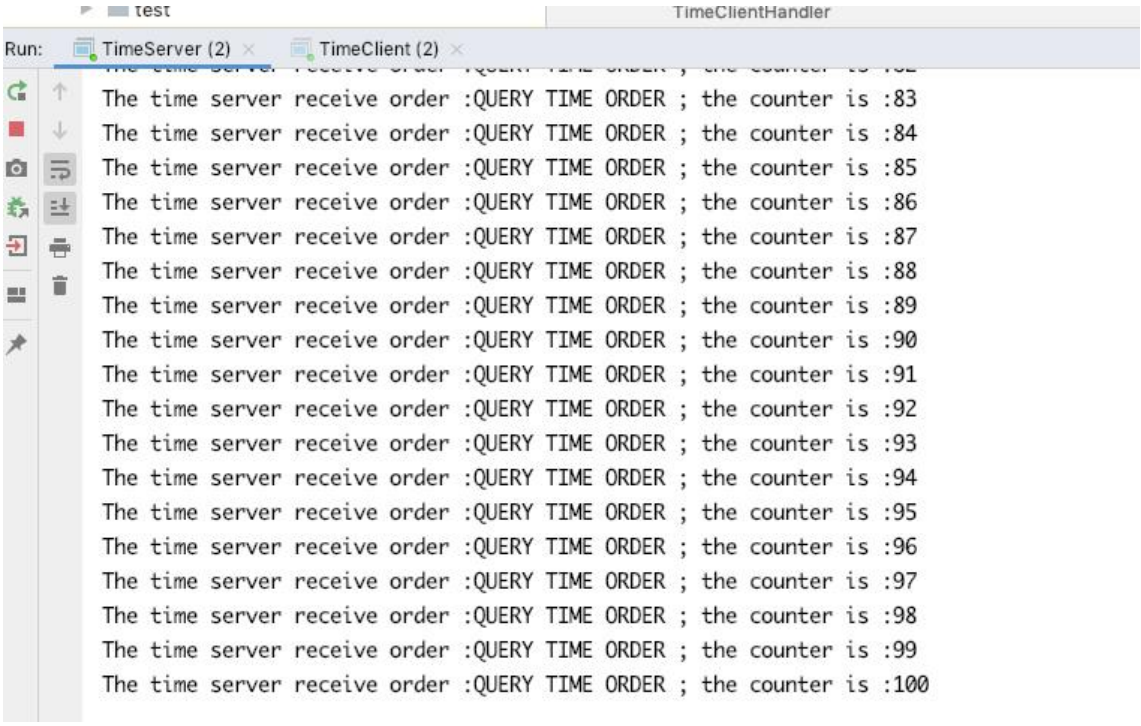
```
package club.wujingjian.com.wujingjian.netty.client;

import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelHandlerAdapter;
import io.netty.channel.ChannelHandlerContext;

import java.util.logging.Logger;

/**
 * 当客户端和服务端TCP链路建立成功之后,Netty的NIO线程会调用channelActive方法,发送查询时的指令给服务端,
 * 调用ChannelHandlerContext的writeAndFlush方法将请求消息发送给服务端,
 * 当服务端返回应答消息时,channelRead方法被调用,Netty从ByteBuf中读取并打印应答消息
 * 当发生异常时,调用exceptionCaught方法,打印异常日志,释放客户端资源
```

```
 */
public class TimeClientHandler extends ChannelHandlerAdapter {

    private static final Logger logger = Logger.getLogger(TimeClientHandler.class.getName());

    private final ByteBuf firstMessage;

    private int counter;

    private byte[] req;

    public TimeClientHandler(){
        req = ("QUERY TIME ORDER" + System.getProperty("line.separator")).getBytes();
        firstMessage = Unpooled.buffer(req.length);
        firstMessage.writeBytes(req);
    }

    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
//        ctx.writeAndFlush(firstMessage);
        ByteBuf message = null;
        //建立连接之后,循环发送100条消息,每发送一条就刷新一次,保证每条消息都会被写入Channel中,
        for (int i = 0; i < 100; i++) {
            message = Unpooled.buffer(req.length);
            message.writeBytes(req);
            ctx.writeAndFlush(message);
        }
    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        //每收到一条响应就打印一次计数消息
        /*ByteBuf buf = (ByteBuf) msg;
        byte[] req = new byte[buf.readableBytes()];
        buf.readBytes(req);
        String body = new String(req, "UTF-8");
        System.out.println("Now is :" + body + " ; the counter is :" + ++counter);*/

        //粘包拆包
        /*ByteBuf buf = (ByteBuf) msg;
        byte[] req = new byte[buf.readableBytes()];
        buf.readBytes(req);
        String body = new String(req, "UTF-8");*/
        String body = (String) msg;
        System.out.println("Now is :" + body + " ; the counter is :" + ++counter);
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
        //释放资源
        logger.warning("Unexpected exception from downstream : " + cause.getMessage());
        ctx.close();
    }
```

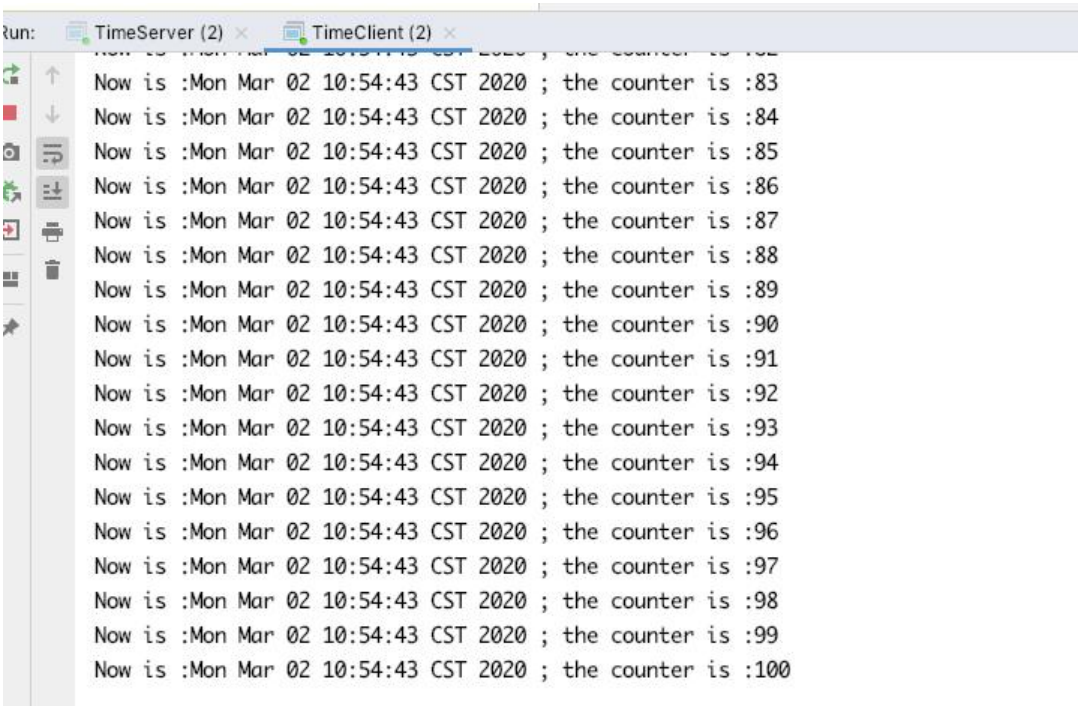
```
}
```

输出:

服务端:



客户端:



总结:

LineBasedFrameDecoder的工作原理是它一次遍历ByteBuf中的可读字节,判断看是否有"\n" 或者"\r\",如果有,

就以此位置为结束位置,从可读索引到结束位置区间的字节就组成了一行。它是以换行符为结束标志的码器,支持

携带结束符或者不携带结束符两种解码方式,同时支持配置单行的最大长度。如果连续读取到最大长度仍然没有

发现换行符，就会抛出异常,同时忽略掉之前读到的异常码流。

StringDecoder的功能非常简单,就是将接收到的对象转换成字符串,然后继续调用后面的handler。LInBasedFrameDecoder + StringDecoder 组合就是按行切换的文本解码器，它被设计用来支持TCP的包和拆包。

Netty提供了多种支持TCP粘包/拆包的解码器,满足用户的不通诉求。

本文实例如果发送多次消息而不使用LineBasedFrameDecoder 则允许结果和逾期不一致(发送了未理的粘包拆包问题)