高性能NIO框架Netty-对象传输 ☆

尹吉欢 · 2018-03-03 · 0条评论 · 4640人阅读

java (http://cxytiandi.com/article/search/java) netty (http://cxytiandi.com/article/search/netty)

上篇文章高性能NIO框架Netty入门篇 (http://cxytiandi.com/blog/detail/17345)我们对Netty做了一个简单的介绍，并且写了一个入门的Demo,客户端往服务端发送一个字符串的消息，服务端回复一个字符串的消息，今天我们来学习下在Netty中怎么使用对象来传输数据。

上篇文章中传输字符串我们用的是框架自带的StringEncoder，StringDecoder编解码器，现在想要通过对象来传输数据，该怎么弄呢?

既然StringEncoder和StringDecoder可以传输字符串，我们来看看这2个类的源码不就知道它们到底做了一些什么工作。

StringEncoder

```java
public class StringEncoder extends MessageToMessageEncoder<CharSequence> {

    // TODO Use CharsetEncoder instead.
    private final Charset charset;

    /**
     * Creates a new instance with the current system character set.
     */
    public StringEncoder() {
        this(Charset.defaultCharset());
    }

    /**
     * Creates a new instance with the specified character set.
     */
    public StringEncoder(Charset charset) {
        if (charset == null) {
            throw new NullPointerException("charset");
        }
        this.charset = charset;
    }

    @Override
    protected void encode(ChannelHandlerContext ctx, CharSequence msg, List<Object> out) t
hrows Exception {
        if (msg.length() == 0) {
            return;
        }

        out.add(ByteBufUtil.encodeString(ctx.alloc(), CharBuffer.wrap(msg), charset));
    }
}
```

通过继承MessageToMessageEncoder，重写encode方法来进行编码操作，就是将字符串进行输出即可。

StringDecoder

```
public class StringDecoder extends MessageToMessageDecoder<ByteBuf> {

    // TODO Use CharsetDecoder instead.
    private final Charset charset;

    /**
     * Creates a new instance with the current system character set.
     */
    public StringDecoder() {
        this(Charset.defaultCharset());
    }

    /**
     * Creates a new instance with the specified character set.
     */
    public StringDecoder(Charset charset) {
        if (charset == null) {
            throw new NullPointerException("charset");
        }
        this.charset = charset;
    }

    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out) throws Exception {
        out.add(msg.toString(charset));
    }
}
```

继承MessageToMessageDecoder，重写decode方法，将ByteBuf数据直接转成字符串进行输出，解码完成。

通过上面的源码分析，我们发现编解码的原理无非就是在数据传输前进行一次处理，接收后进行一次处理，在网络中传输的数据都是字节，我们现在想要传PO对象，那么必然需要进行编码和解码2个步骤，我们可以自定义编解码器来对对象进行序列化，然后通过ByteBuf的形式进行传输, 传输对象需要实现java.io.Serializable接口。

首先我们定义一个传输对象，实现序列化接口，暂时先定义2个字段，一个ID,用来标识客户端，一个内容字段，代码如下：

```java
public class Message implements Serializable {
    private static final long serialVersionUID = -7543514952950971498L;
    private String id;
    private String content;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

}
```

传输对象定好后，定义对象的编解码器。

对象编码器

将对象序列化成字节，通过ByteBuf形式进行传输，ByteBuf是一个byte存放的缓冲区，提供了读写操作。

```java
public class MessageEncoder extends MessageToByteEncoder<Message> {

    @Override
    protected void encode(ChannelHandlerContext ctx, Message message, ByteBuf out) throws Exception {
        byte[] datas = ByteUtils.objectToByte(message);
        out.writeBytes(datas);
        ctx.flush();
    }

}
```

对象解码器

接收ByteBuf数据，将ByteBuf反序列化成对象

```
1. public class MessageDecoder extends ByteToMessageDecoder {
2. 
3.     @Override
4.     protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws
   Exception {
5.         Object obj = ByteUtils.byteToObject(ByteUtils.read(in));
6.         out.add(obj);
7.     }
8. 
9. }
```

将上篇文章中服务端的编解码器改成对象编解码器：

```java
public class ImServer {

    public void run(int port) {
        EventLoopGroup bossGroup = new NioEventLoopGroup();
        EventLoopGroup workerGroup = new NioEventLoopGroup();

        ServerBootstrap bootstrap = new ServerBootstrap();
        bootstrap.group(bossGroup, workerGroup)
                .channel(NioServerSocketChannel.class)
                .childHandler(new ChannelInitializer<SocketChannel>() {
                    @Override
                    public void initChannel(SocketChannel ch) throws Exception {
                        //实体类传输数据，jdk序列化
                        ch.pipeline().addLast("decoder", new MessageDecoder());
                        ch.pipeline().addLast("encoder", new MessageEncoder());
                        ch.pipeline().addLast(new ServerPoHandler());
                        //字符串传输数据
                        /*ch.pipeline().addLast("decoder", new StringDecoder());
                        ch.pipeline().addLast("encoder", new StringEncoder());
                        ch.pipeline().addLast(new ServerStringHandler());*/
                    }
                })
                .option(ChannelOption.SO_BACKLOG, 128)
                .childOption(ChannelOption.SO_KEEPALIVE, true);

        try {
            ChannelFuture f = bootstrap.bind(port).sync();
             f.channel().closeFuture().sync();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            workerGroup.shutdownGracefully();
            bossGroup.shutdownGracefully();
        }
    }

}
```

接下来编写服务端的消息处理类：

```
public class ServerPoHandler extends ChannelInboundHandlerAdapter {

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        Message message = (Message) msg;
        System.err.println("server:" + message.getId());
        ctx.writeAndFlush(message);
    }

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

}
```

服务端改造好了之后，就要改造客户端了，同样的道理，客户端和服务端的编解码器都要一致才行。

客户端连接时指定对象编解码器和对象消息处理类，代码如下：

```java
public class ImConnection {

    private Channel channel;

    public Channel connect(String host, int port) {
        doConnect(host, port);
        return this.channel;
    }

    private void doConnect(String host, int port) {
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            Bootstrap b = new Bootstrap();
            b.group(workerGroup);
            b.channel(NioSocketChannel.class);
            b.option(ChannelOption.SO_KEEPALIVE, true);
            b.handler(new ChannelInitializer<SocketChannel>() {
                @Override
                public void initChannel(SocketChannel ch) throws Exception {
                    //实体类传输数据，jdk序列化
                    ch.pipeline().addLast("decoder", new MessageDecoder());
                    ch.pipeline().addLast("encoder", new MessageEncoder());
                    ch.pipeline().addLast(new ClientPoHandler());

                    //字符串传输数据
                    /*ch.pipeline().addLast("decoder", new StringDecoder());
                    ch.pipeline().addLast("encoder", new StringEncoder());
                    ch.pipeline().addLast(new ClientStringHandler());*/
                }
            });

            ChannelFuture f = b.connect(host, port).sync();
            channel = f.channel();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

客户端消息处理类：

```
/**
 * 当编解码器为实体对象时时用来接收数据
 * @author yinjihuan
 *
 */
public class ClientPoHandler extends ChannelInboundHandlerAdapter {

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        Message message = (Message) msg;
        System.out.println("client:" + message.getContent());
    }

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

}
```

客户端启动类也需要改造，将发送字符串的消息变成对象消息

```
public class ImClientApp {
    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 2222;
        Channel channel = new ImConnection().connect(host, port);
        //对象传输数据
        Message message = new Message();
        message.setId(UUID.randomUUID().toString().replaceAll("-", ""));
        message.setContent("hello yinjihuan");
        channel.writeAndFlush(message);
        //字符串传输数据
        //channel.writeAndFlush("yinjihuan");
    }
}
```

源码参考：https://github.com/yinjihuan/netty-im (https://github.com/yinjihuan/netty-im)

欢迎加入我的知识星球，一起交流技术，免费学习猿天地的课程（http://cxytiandi.com/course） (http://cxytiandi.com/course）)