### 8.  Speaking in POJO instead of ByteBuf

All the examples we have reviewed so far used a [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) as a primary data structure of a protocol message. In this section, we will improve the TIME protocol client and server example to use a [POJO](http://en.wikipedia.org/wiki/POJO) instead of a [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html).

The advantage of using a POJO in your [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) is obvious; your handler becomes more maintainable and reusable by separating the code which extracts information from [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) out from the handler. In the TIME client and server examples, we read only one 32-bit integer and it is not a major issue to use [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) directly. However, you will find it is necessary to make the separation as you implement a real world protocol.

First, let us define a new type called UnixTime.

1 package io.netty.example.time;

2

import java.util.Date;

4

public class UnixTime {

6 private final int value;

8 public UnixTime(int value) {

this.value = value;

10 }

12 public int getValue() {

return value;

14 }

16 @Override

public String toString() {

18 return new Date(value \* 1000L).toString();

}

20 }

We can now revise the TimeDecoder to return a UnixTime instead of a [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html).

1 @Override

2 protected Object decode(

[ChannelHandlerContext](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandlerContext.html) ctx, [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) in) {

4 if (in.readableBytes() < 4) {

return null;

6 }

8 return new UnixTime(int.readInt());(24)

}

|  |  |
| --- | --- |
| [(24)](http://netty.io/4.0/guide/#example.time6.co1) | [ByteToMessageDecoder](http://static.netty.io/4.0/api/io/netty/handler/codec/ByteToMessageDecoder.html) and [ReplayingDecoder](http://static.netty.io/4.0/api/io/netty/handler/codec/ReplayingDecoder.html) allow you to return an object of any type. If they were restricted to return only a [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html), we would have to insert another [ChannelHandler](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandler.html) which transforms a [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) into aUnixTime. |

With the updated decoder, the TimeClientHandler does not use [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html) anymore so we need to change it to receive UnixTime messages name.

1 public class TimeClientHandler extends [ChannelInboundMessageHandlerAdapter](http://static.netty.io/4.0/api/io/netty/channel/ChannelInboundMessageHandlerAdapter.html)<UnixTime> {

2

@Override

4 public void messageReceived([ChannelHandlerContext](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandlerContext.html) ctx, Unixtime m) {

System.out.println(m);

6 ctx.close();

}

8

@Override

10 public void exceptionCaught([ChannelHandlerContext](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandlerContext.html) ctx, Throwable cause) {

cause.printStackTrace();

12 ctx.close();

}

14 }

Much simpler and elegant, right? The same technique can be applied on the server side. Let us update theTimeServerHandler first this time:

1 @Override

2 public void channelActive([ChannelHandlerContext](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandlerContext.html) ctx) {

UnixTime time = new UnixTime(System.currentTimeMillis() / 1000);

4 [ChannelFuture](http://static.netty.io/4.0/api/io/netty/channel/ChannelFuture.html) f = ctx.write(time);

f.addListener([ChannelFutureListener](http://static.netty.io/4.0/api/io/netty/channel/ChannelFutureListener.html).CLOSE);

6 }

Now, the only missing piece is an encoder, which is an implementation of [ChannelHandler](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandler.html) that translates a UnixTimeback into bytes that can be written to the outbound [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html). It's much simpler than writing a decoder because there's no need to deal with packet fragmentation and assembly when encoding a message.

1 package io.netty.example.time;

2

public class TimeEncoder extends [MessageToByteEncoder](http://static.netty.io/4.0/api/io/netty/handler/codec/MessageToByteEncoder.html)<UnixTime> {

4

@Override

6 protected void encode([ChannelHandlerContext](http://static.netty.io/4.0/api/io/netty/channel/ChannelHandlerContext.html) ctx, UnixTime(25) msg, ByteBuf out) {

out.writeInt(m.getValue());(26)

8 }

}

|  |  |
| --- | --- |
| [(25)](http://netty.io/4.0/guide/#example.time7.co1) | An encoder overrides the encode. |
| [(26)](http://netty.io/4.0/guide/#example.time7.co2) | All needs to get done is to encode the message to bytes and write it to the outbound [ByteBuf](http://static.netty.io/4.0/api/io/netty/buffer/ByteBuf.html), like shown here. |

The last task left is to insert a TimeEncoder into the [ChannelPipeline](http://static.netty.io/4.0/api/io/netty/channel/ChannelPipeline.html) on the server side, and it is left as a trivial exercise.