文章目录

dec 之

主要由两部分构造：

- MessageToMessageEncoder ，将消息**编码**成另一种消息。
- MessageToMessageDecoder ，将消息**解码**成另一种消息。

所以，我们先来看 MessageToMessageDecoder 的代码实现，再看 MessageToMessageEncoder 的代码实现，最后看 MessageToMessageCodec 的代码实现。

> 老艿艿：因为笔者平时比较少用这三个类，所以本文会分享的比较简单噢。

2. MessageToMessageEncoder

`io.netty.handler.codec.MessageToMessageEncoder` ，继承 ChannelOutboundHandlerAdapter 抽象类，将消息**编码**成另一种消息。

2.1 构造方法

```
/**
 * 类型匹配器
 */
private final TypeParameterMatcher matcher;

protected MessageToMessageEncoder() {
    matcher = TypeParameterMatcher.find(this, MessageToMessageEncoder.class, "I");
}

protected MessageToMessageEncoder(Class<? extends I> outboundMessageType) {
    matcher = TypeParameterMatcher.get(outboundMessageType);
}
```

文章目录


```
ould be handled. If {@code false} it will be passed to
 ChannelPipeline}.

) throws Exception {
```

```
ject msg, ChannelPromise promise) throws Exception {
```

```
            // 创建 CodecOutputList 对象
            out = CodecOutputList.newInstance();
            // 转化消息类型
            @SuppressWarnings("unchecked")
            I cast = (I) msg;
            try {
                // 将消息编码成另外一个消息
                encode(ctx, cast, out);
            } finally {
                // 释放 cast 原消息
                ReferenceCountUtil.release(cast);
            }

            // 如果未编码出消息，抛出异常
            if (out.isEmpty()) {
                // 回收 CodecOutputList 对象
                out.recycle();
                out = null;
                // 抛出异常
                throw new EncoderException(StringUtil.simpleClassName(this) + " must produce at least
            }
        } else {
            // 直接下一个节点
            ctx.write(msg, promise);
        }
    } catch (EncoderException e) {
        throw e;
    } catch (Throwable t) {
        throw new EncoderException(t);
    } finally {
        if (out != null) {
            final int sizeMinusOne = out.size() - 1;
            // 只编码出一条消息
```

文章目录


```
                节点，使用 voidPromise ，即不需要回调
                omise for our extra writes to reduce GC-Pressure
                /netty/issues/2525
                x.voidPromise();
                 == voidPromise;
                e; i ++) {
                p);
                使用 promise ，即需要回调
                usOne), promise);
                }
                // 回收 CodecOutputList 对象
                out.recycle();
            }
        }
    }
```

- 代码比较简单，胖友自己看注释。

2.4 encode

```
/**
 * Encode from one message to an other. This method will be called for each written message that can b
 * by this encoder.
 *
 * @param ctx           the {@link ChannelHandlerContext} which this {@link MessageToMessageEncoder} b
 * @param msg           the message to encode to an other one
 * @param out           the {@link List} into which the encoded msg should be added
 *                      needs to do some kind of aggregation
 * @throws Exception    is thrown if an error occurs
 */
protected abstract void encode(ChannelHandlerContext ctx, I msg, List<Object> out) throws Exception;
```

2.5 子类

MessageToMessageEncoder 子类比较多，感兴趣的胖友，可以看看负责实现 FrameEncoder 的两个类：

- `io.netty.handler.codec.string.LineEncoder` ，基于**指定行分隔符**的 FrameEncoder 实现类。
  - 代码比较简单，胖友可以直接撸。
- `io.netty.handler.codec.LengthFieldPrepender` ，基于基于**消息头指定消息长度**的 FrameEncoder 实现类。

文章目录

sageToByteEncoder》一起撸。

ler

，继承 ChannelInboundHandlerAdapter 类，将消息**解码**成另一

```
essageToMessageDecoder.class, "I");

ds I> inboundMessageType) {
    matcher = TypeParameterMatcher.get(inboundMessageType);
}
```

3.2 acceptInboundMessage

```
public boolean acceptInboundMessage(Object msg) throws Exception {
    return matcher.match(msg);
}
```

3.3 channelRead

```
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
    // 创建 CodecOutputList 对象
    CodecOutputList out = CodecOutputList.newInstance();
    try {
        // 判断是否为匹配的消息
        if (acceptInboundMessage(msg)) {
            // 转化消息类型
            @SuppressWarnings("unchecked")
            I cast = (I) msg;
            try {
                // 将消息解码成另外一个消息
                decode(ctx, cast, out);
            } finally {
                // 释放 cast 原消息
                ReferenceCountUtil.release(cast);
            }
        } else {
            // 不匹配，添加到 out
            out.add(msg);
```

文章目录

peline 中

));

```
 * Decode from one message to an other. This method will be called for each written message that can b
 * by this encoder.
 *
 * @param ctx           the {@link ChannelHandlerContext} which this {@link MessageToMessageDecoder} b
 * @param msg           the message to decode to an other one
 * @param out           the {@link List} to which decoded messages should be added
 * @throws Exception    is thrown if an error occurs
 */
protected abstract void decode(ChannelHandlerContext ctx, I msg, List<Object> out) throws Exception;
```

4. MessageToMessageCodec

`io.netty.handler.codec.MessageToMessageCodec` ，继承 ChannelDuplexHandler 类，通过**组合** MessageToMessageEncoder 和 MessageToMessageDecoder 的功能，从而实现编解码的 Codec **抽象类**。

> 老艿艿：从实现的方式上，和 ByteToMessageCodec 非常相似。

4.1 构造方法

```
public abstract class MessageToMessageCodec<INBOUND_IN, OUTBOUND_IN> extends ChannelDuplexHandler {

    /**
     * Encoder 对象
     */
    private final MessageToMessageEncoder<Object> encoder = new MessageToMessageEncoder<Object>() {
```

文章目录


```
ject msg) throws Exception {
cceptOutboundMessage(msg);
```

```
text ctx, Object msg, List<Object> out) throws Exceptio
tx, (OUTBOUND_IN) msg, out);
```

```
t> decoder = new MessageToMessageDecoder<Object>() {
```

```
ect msg) throws Exception {
cceptInboundMessage(msg);
```

```
        @Override
        @SuppressWarnings("unchecked")
        protected void decode(ChannelHandlerContext ctx, Object msg, List<Object> out) throws Exceptio
            MessageToMessageCodec.this.decode(ctx, (INBOUND_IN) msg, out);
        }
    };

    /**
     * Decoder 的类型匹配器
     */
    private final TypeParameterMatcher inboundMsgMatcher;
    /**
     * Encoder 的类型匹配器
     */
    private final TypeParameterMatcher outboundMsgMatcher;

    protected MessageToMessageCodec() {
        inboundMsgMatcher = TypeParameterMatcher.find(this, MessageToMessageCodec.class, "INBOUND_IN")
        outboundMsgMatcher = TypeParameterMatcher.find(this, MessageToMessageCodec.class, "OUTBOUND_IN
    }

    protected MessageToMessageCodec(Class<? extends INBOUND_IN> inboundMessageType, Class<? extends OU
        inboundMsgMatcher = TypeParameterMatcher.get(inboundMessageType);
        outboundMsgMatcher = TypeParameterMatcher.get(outboundMessageType);
    }

    // ...  省略非构造方法相关
}
```

4.2 MessageToMessageEncoder 相关方法

文章目录


```
ject msg, ChannelPromise promise) throws Exception {


ified message can be decoded by this codec.


 throws Exception {


andlerContext, Object, List)

ntext ctx, OUTBOUND_IN msg, List<Object> out) throws Ex
```

4.3 MessageToMessageDecoder 相关方法

```
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
    decoder.channelRead(ctx, msg);
}

/**
 * Returns {@code true} if and only if the specified message can be encoded by this codec.
 *
 * @param msg the message
 */
public boolean acceptOutboundMessage(Object msg) throws Exception {
    return outboundMsgMatcher.match(msg);
}

/**
 * @see MessageToMessageDecoder#decode(ChannelHandlerContext, Object, List)
 */
protected abstract void decode(ChannelHandlerContext ctx, INBOUND_IN msg, List<Object> out) throws Exc
```

666. 彩蛋

还是一篇小水文。嘿嘿。

推荐阅读文章：

- wade&luffy 《ChannelHandler》
- 堆码时刻 《Netty In Action中文版 - 第八章：附带的ChannelHandler和Codec》

文章目录