# Netty粘包拆包解决方案 ☆

尹吉欢 · 2018-03-09 · 0条评论 · 4009人阅读

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

## 前言

本篇文章是Netty专题的第六篇，前面五篇文章如下：

- 高性能NIO框架Netty入门篇 (http://cxytiandi.com/blog/detail/17345)
- 高性能NIO框架Netty-对象传输 (http://cxytiandi.com/blog/detail/17403)
- 高性能NIO框架Netty-整合kryo高性能数据传输 (http://cxytiandi.com/blog/detail/17436)
- 高性能NIO框架Netty-整合Protobuf高性能数据传输 (http://cxytiandi.com/blog/detail/17469)
- Netty4自带编解码器详解 (http://cxytiandi.com/blog/detail/17547)

## TCP黏包拆包

TCP是一个流协议，就是没有界限的一长串二进制数据。TCP作为传输层协议并不不了解上层业务数据的具体含义，它会根据TCP缓冲区的实际情况进行数据包的划分，所以在业务上认为是一个完整的包，可能会被TCP拆分成多个包进行发送，也有可能把多个小的包封装成一个大的数据包发送，这就是所谓的TCP粘包和拆包问题。

## 怎么解决?

1. 消息定长度，传输的数据大小固定长度，例如每段的长度固定为100字节，如果不够空位补空格
2. 在数据包尾部添加特殊分隔符，比如下划线，中划线等
3. 将消息分为消息头和消息体，消息头中包含表示信息的总长度

Netty提供了多个解码器，可以进行分包的操作，分别是：

- LineBasedFrameDecoder （回车换行分包）
- DelimiterBasedFrameDecoder（特殊分隔符分包）
- FixedLengthFrameDecoder（固定长度报文来分包）
- LengthFieldBasedFrameDecoder（自定义长度来分包）

## 制造粘包和拆包问题

为了验证我们的解码器能够解决这种粘包和拆包带来的问题，首先我们就制造一个这样的问题，以此用来做对比。

服务端：

```java
public static void main(String[] args) {
        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 {
                        ch.pipeline().addLast("decoder", new StringDecoder());
                        ch.pipeline().addLast("encoder", new StringEncoder());
                        ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
                            @Override
                            public void channelRead(ChannelHandlerContext ctx, Object msg) {
                                System.err.println("server:" + msg.toString());
                                ctx.writeAndFlush(msg.toString() + "你好" );
                            }
                        });
                    }
                })
                .option(ChannelOption.SO_BACKLOG, 128)
                .childOption(ChannelOption.SO_KEEPALIVE, true);

        try {
            ChannelFuture f = bootstrap.bind(2222).sync();
             f.channel().closeFuture().sync();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            workerGroup.shutdownGracefully();
            bossGroup.shutdownGracefully();
        }
    }
```

客户端我们发送一个比较长的字符串，如果服务端收到的消息是一条，那么就是对的，如果是多条，那么就有问题了。

```java
public static void main(String[] args) {
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        Channel channel = null;
        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 {
                    ch.pipeline().addLast("decoder", new StringDecoder());
                    ch.pipeline().addLast("encoder", new StringEncoder());
                    ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
                        @Override
                        public void channelRead(ChannelHandlerContext ctx, Object msg) {
                            System.err.println("client:" + msg.toString());
                        }
                    });
                }
            });

            ChannelFuture f = b.connect("127.0.0.1", 2222).sync();
            channel = f.channel();
            StringBuilder msg = new StringBuilder();
            for (int i = 0; i < 100; i++) {
                msg.append("hello yinjihuan");
            }
            channel.writeAndFlush(msg);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```

首先启动服务端，然后再启动客户端，通过控制台可以看到服务接收的数据分成了2次,这就是我们要解决的问题。

```
server:hello yinjihuanhello....
server:o yinjihuanhello...
```

## LineBasedFrameDecoder

用LineBasedFrameDecoder 来解决需要在发送的数据结尾加上回车换行符，这样LineBasedFrameDecoder 才知道这段数据有没有读取完整。

改造服务端代码，只需加上LineBasedFrameDecoder 解码器即可,构造函数的参数是数据包的最大长度。

```java
public void initChannel(SocketChannel ch) throws Exception {
     ch.pipeline().addLast(new LineBasedFrameDecoder(10240));
     ch.pipeline().addLast("decoder", new StringDecoder());
     ch.pipeline().addLast("encoder", new StringEncoder());
     ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
           @Override
           public void channelRead(ChannelHandlerContext ctx, Object msg) {
               System.err.println("server:" + msg.toString());
               ctx.writeAndFlush(msg.toString() + "你好");
           }
    });
}
```

改造客户端发送代码，再数据后面加上回车换行符

```java
ChannelFuture f = b.connect("127.0.0.1", 2222).sync();
channel = f.channel();
StringBuilder msg = new StringBuilder();
for (int i = 0; i < 100; i++) {
    msg.append("hello yinjihuan");
}
channel.writeAndFlush(msg + System.getProperty("line.separator"));
```

## DelimiterBasedFrameDecoder

DelimiterBasedFrameDecoder和LineBasedFrameDecoder差不多，DelimiterBasedFrameDecoder可以自己定义需要分割的符号，比如下划线，中划线等等。

改造服务端代码，只需加上DelimiterBasedFrameDecoder解码器即可,构造函数的参数是数据包的最大长度。我们用下划线来分割。

```java
 public void initChannel(SocketChannel ch) throws Exception {
      ch.pipeline().addLast(new DelimiterBasedFrameDecoder(10240, Unpooled.copiedBuffer(
"_".getBytes())));
      ch.pipeline().addLast("decoder", new StringDecoder());
      ch.pipeline().addLast("encoder", new StringEncoder());
      ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
            @Override
            public void channelRead(ChannelHandlerContext ctx, Object msg) {
                System.err.println("server:" + msg.toString());
                ctx.writeAndFlush(msg.toString() + "你好");
            }
     });
}
```

改造客户端发送代码，再数据后面加上下划线

```java
ChannelFuture f = b.connect("127.0.0.1", 2222).sync();
channel = f.channel();
StringBuilder msg = new StringBuilder();
for (int i = 0; i < 100; i++) {
    msg.append("hello yinjihuan");
}
channel.writeAndFlush(msg + "_");
```

## FixedLengthFrameDecoder

FixedLengthFrameDecoder是按固定的数据长度来进行解码的，也就是说你客户端发送的每条消息的长度是固定的，下面我们看看怎么使用。
服务端还是一样，增加FixedLengthFrameDecoder解码器即可。

```java
 public void initChannel(SocketChannel ch) throws Exception {
      ch.pipeline().addLast(new FixedLengthFrameDecoder(1500));
      ch.pipeline().addLast("decoder", new StringDecoder());
      ch.pipeline().addLast("encoder", new StringEncoder());
      ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
            @Override
            public void channelRead(ChannelHandlerContext ctx, Object msg) {
                System.err.println("server:" + msg.toString());
                ctx.writeAndFlush(msg.toString() + "你好");
            }
     });
}
```

客户端，msg输出的长度就是1500

```java
ChannelFuture f = b.connect("127.0.0.1", 2222).sync();
channel = f.channel();
StringBuilder msg = new StringBuilder();
for (int i = 0; i < 100; i++) {
    msg.append("hello yinjihuan");
}
System.out.println(msg.length());
channel.writeAndFlush(msg);
```

## LengthFieldBasedFrameDecoder

服务端代码：

```java
 public void initChannel(SocketChannel ch) throws Exception {
      ch.pipeline().addLast("frameDecoder", new LengthFieldBasedFrameDecoder(Integer.MAX_VALUE, 0, 4, 0, 4));
      ch.pipeline().addLast("frameEncoder", new LengthFieldPrepender(4));
      ch.pipeline().addLast("decoder", new StringDecoder());
      ch.pipeline().addLast("encoder", new StringEncoder());
      ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
            @Override
            public void channelRead(ChannelHandlerContext ctx, Object msg) {
                System.err.println("server:" + msg.toString());
                ctx.writeAndFlush(msg.toString() + "你好");
            }
     });
}
```

客户端，直接发送就行

```java
ChannelFuture f = b.connect("127.0.0.1", 2222).sync();
channel = f.channel();
StringBuilder msg = new StringBuilder();
for (int i = 0; i < 100; i++) {
    msg.append("hello yinjihuan");
}
channel.writeAndFlush(msg);
```