Netty4自带编解码器详解 ☆

尹吉欢 · 2018-03-07 · 1条评论 · 2747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专题的第五篇，前面四篇文章如下：

- 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)

作为一个高性能的异步、NIO通信框架，编解码框架是Netty的重要组成部分。

从网络中读取消息，需要经过解码，将二进制的数据报转换成应用层协议消息，才能够被应用逻辑识别。同样的道理，客户端发送给服务器的消息，也需要经过编码转换成二进制字节数组（Netty就是ByteBuf）才能够发送到网络对端。编码和解码功能是NIO框架必不可少的一部分。

Netty为了降低用户的开发难度，对原始的NIO进行封装，提供了常用的功能和API，屏蔽了底层的实现细节。对于不想了解底层实现的用户，使用Netty自带的编解码器非常容易，都能够快速上手，提高开发效率。

Netty在这方面做得非常好，对编解码功能，提供了通用的编解码框架可以让用户扩展，又提供了常用的一些编解码类让用户直接使用。

Netty自带的编解码功能列表如下：

- String
- Protobuf
- Base64
- Object
- 其他等等.....

本篇文章只讲解我列出来的几个，还有一些像粘包的解码器我们后面单独写文章进行讲解。

String编解码

String编解码在Netty中对应的类是**io.netty.handler.codec.string.StringEncoder**和**io.netty.handler.codec.string.StringDecoder**，提供字符串数据的传输编解码工作。

关于String编解码的使用这边不做过多讲解，可以参考我的《高性能NIO框架Netty入门篇》(http://cxytiandi.com/blog/detail/17345)中的使用。

Protobuf编解码

Protobuf编解码在Netty中对应的类是**io.netty.handler.codec.protobuf.ProtobufDecoder**和**io.netty.handler.codec.protobuf.ProtobufEncoder**，提供基于Protobuf序列化的数据传输编解码工作。

关于Protobuf编解码的使用这边不做过多讲解，可以参考我的《高性能NIO框架Netty-整合Protobuf高性能数据传输》(http://cxytiandi.com/blog/detail/17469)中的使用。

Base64编解码

base64的使用需要在String的基础上，不然消息是无法直接传递。

服务端

```java
/**
 * Base64编解码示例
 * @author yinjihuan
 *
 */
public class Base64EncoderAndDecoderServer {
    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("base64Decoder", new Base64Decoder());
                        ch.pipeline().addLast("base64Encoder", new Base64Encoder());
                        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

```
/**
 * Base64编解码示例
 * @author yinjihuan
 *
 */
public class Base64EncoderAndDecoderClient {
    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("base64Decoder", new Base64Decoder());
                    ch.pipeline().addLast("base64Encoder", new Base64Encoder());
                    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();
            channel.writeAndFlush("hello yinjihuan");
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Object编解码

Object编解码在Netty中对应的类是**io.netty.handler.codec.serialization.ObjectEncoder**和**io.netty.handler.codec.serialization.ObjectDecoder**，提供基于对象序列化的数据传输编解码工作。

之前我们在《高性能NIO框架Netty-对象传输》(http://cxytiandi.com/blog/detail/17403)中通过自定义编码器来实现了PO对象的传输，今天就用Netty自带的Object来进行编解码工作。

传输对象

```
1. public class ObjectMessage implements Serializable {
2.     private static final long serialVersionUID = -7543514952950971498L;
3.     private String id;
4.     private String content;
5. 
6.     public String getId() {
7.         return id;
8.     }
9. 
10.     public void setId(String id) {
11.         this.id = id;
12.     }
13. 
14.     public String getContent() {
15.         return content;
16.     }
17. 
18.     public void setContent(String content) {
19.         this.content = content;
20.     }
21. 
22. }
```

服务端

```java
/**
 * Object编解码示例
 * @author yinjihuan
 *
 */
public class ObjectEncoderAndDecoderServer {
    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 ObjectDecoder(ClassResolvers.cacheDisabled(
                                this.getClass().getClassLoader()
                        )));
                        ch.pipeline().addLast("encoder", new ObjectEncoder());
                        ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
                            @Override
                            public void channelRead(ChannelHandlerContext ctx, Object msg) {
                                ObjectMessage m = (ObjectMessage) msg;
                                System.err.println("server:" + m.getContent());
                            }
                        });
                    }
                })
                .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();
        }
    }
```

```
43. }
```

客户端

```java
/**
 * Object编解码示例
 * @author yinjihuan
 *
 */
public class ObjectEncoderAndDecoderClient {
    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 ObjectDecoder(ClassResolvers.cacheDisabled(
                            this.getClass().getClassLoader()
                    )));
                    ch.pipeline().addLast("encoder", new ObjectEncoder());
                    ch.pipeline().addLast(new ChannelInboundHandlerAdapter() {
                        @Override
                        public void channelRead(ChannelHandlerContext ctx, Object msg) {
                            ObjectMessage m = (ObjectMessage) msg;
                            System.err.println("client:" + m.getContent());
                        }
                    });
                }
            });

            ChannelFuture f = b.connect("127.0.0.1", 2222).sync();
            channel = f.channel();
            ObjectMessage m = new ObjectMessage();
            m.setContent("hello yinjihuan");
            channel.writeAndFlush(m);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

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