MessagePack编解码框架

笔记本：	<Inbox>		
创建时间：	12/5/2018 16:36	更新时间：	12/6/2018 9:05
作者：	wangqinlinmail@163.com		
标签：	Netty		

MessagePack编解码框架：

MessagePack是一个高效的二进制序列化框架，它像JSON一样，支持不同语言间的数据交换，但是它的性能更快，序列化之后的码流也更小；

pom.xml中配置MessagePack坐标：

```
<dependency>
    <groupId>org.msgpack</groupId>
    <artifactId>msgpack</artifactId>
    <version>0.6.12</version>
</dependency>

<dependency>
    <groupId>org.javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.22.0-GA</version>
</dependency>
```

MessagePack解码：

```
import java.util.List;
import org.msgpack.MessagePack;
importio.netty.buffer.ByteBuf;
importio.netty.channel.ChannelHandlerContext;
importio.netty.handler.codec.MessageToMessageDecoder;
public class MsgpackDecoder extends MessageToMessageDecoder<ByteBuf> {
    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf msg, List<Object>
out) throws Exception {

        /**
         * MessagePack的read方法将其反序列化为Object对象
         */
        final byte[] array;
        final int length = msg.readableBytes();
        array = new byte[length];
        msg.getBytes(msg.readerIndex(), array,0,length);
        MessagePack messagePack = new MessagePack();
        out.add(messagePack.read(array));
    }
}
```

MessagePack编码：

```
import org.msgpack.MessagePack;
importio.netty.buffer.ByteBuf;
importio.netty.channel.ChannelHandlerContext;
```

```
importio.netty.handler.codec.MessageToByteEncoder;
public class MsgpackEncoder extends MessageToByteEncoder<Object> {
     @Override
     protected void encode(ChannelHandlerContext ctx, Object msg, ByteBuf out)
throws Exception {
          /**
           * MsgpackEncoder继承MessageToByteEncoder，他负责将Object类型的POJO对象编
为byte数组，然后写到ByteBuf
           */
          MessagePack msgPack = new MessagePack();
          byte[] raw = msgPack.write(msg);
          out.writeBytes(raw);
     }
}
```

EchoClient端：

```
importio.netty.bootstrap.Bootstrap;
importio.netty.channel.ChannelFuture;
importio.netty.channel.ChannelInitializer;
importio.netty.channel.ChannelOption;
importio.netty.channel.nio.NioEventLoopGroup;
importio.netty.channel.socket.SocketChannel;
importio.netty.channel.socket.nio.NioSocketChannel;
importio.netty.handler.codec.LengthFieldBasedFrameDecoder;
importio.netty.handler.codec.LengthFieldPrepender;
public class EchoClient {
     private final String host;
     private final int port;
     public EchoClient(String host, int port) {
          super();
          this.host = host;
          this.port = port;
     }
     public void run() throws InterruptedException {
          NioEventLoopGroup group = new NioEventLoopGroup();
          try {
              Bootstrap b = new Bootstrap();
              b.group(group).channel(NioSocketChannel.class).option(ChannelOption.TCP_NODELAY,
true)
                        .option(ChannelOption.CONNECT_TIMEOUT_MILLIS, 3000)
                        .handler(new ChannelInitializer<SocketChannel>() {
                             @Override
                             protected void initChannel(SocketChannel ch) throws
Exception {
                                  ch.pipeline().addLast("frameDecoder", new
LengthFieldBasedFrameDecoder(65535, 0, 2, 0,2));
                                  ch.pipeline().addLast("msgpack decoder", new
MsgpackDecoder());
                                  ch.pipeline().addLast("frameEncoder", new
LengthFieldPrepender(2));
                                  ch.pipeline().addLast("msgpack encoder", new
MsgpackEncoder());
                                  ch.pipeline().addLast(new EchoClientHandle());
                             }
                        });
              ChannelFuture f = b.connect(host, port).sync();
```

```
            f.channel().closeFuture().sync();
        } finally {
            group.shutdownGracefully();
        }
    }
    public static void main(String[] args) throws Exception {
        int port = 8080;
        if (args != null && args.length > 0) {
            try {
                port = Integer.valueOf(args[0]);
            } catch (Exception e) {
            }
        }
        new EchoClient("127.0.0.1", 8080).run();
    }
}
```

EchoClientHandle端:

```
importio.netty.channel.ChannelHandlerContext;
importio.netty.channel.ChannelInboundHandlerAdapter;
public class EchoClientHandle extends ChannelInboundHandlerAdapter {

    private int count;
    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
        UserInfo[] users = getUsers();
        for (UserInfo userInfo : users) {
            ctx.write(userInfo);
        }
        ctx.flush();
    }
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws
Exception {
        System.out.println("Client receive the msgpack message【  " + ++count +
"  】times:【" + msg + "】");
        if (count < 5) {
            ctx.write(msg);
        }
    }


    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) throws Exception
{
        ctx.flush();
    }

    private UserInfo[] getUsers() {
        UserInfo[] userInfos = new UserInfo[5];
        for (int i = 0; i < 5; i++) {
            UserInfo userInfo = new UserInfo();
            userInfo.setAge(String.valueOf(i));
            userInfo.setUserName("ABCDEF --->"+i);
```

```
            userInfos[i] = userInfo;
        }
        return userInfos;
    }
}
```

EchoServer端：

```
import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;
import io.netty.handler.codec.LengthFieldBasedFrameDecoder;
import io.netty.handler.codec.LengthFieldPrepender;
import io.netty.handler.logging.LogLevel;
import io.netty.handler.logging.LoggingHandler;
public class EchoServer {
    public void bind(int port) throws InterruptedException {
        NioEventLoopGroup bossGroup = new NioEventLoopGroup();
        NioEventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            ServerBootstrap b = new ServerBootstrap();
            b.group(bossGroup,
workerGroup).channel(NioServerSocketChannel.class).option(ChannelOption.SO_BACKLOG,
1024)
                    .handler(new
LoggingHandler(LogLevel.INFO)).childHandler(new ChannelInitializer<SocketChannel>
() {
                        @Override
                        protected void initChannel(SocketChannel ch) throws
Exception {
                            ch.pipeline().addLast("frameDecoder", new
LengthFieldBasedFrameDecoder(65535, 0, 2, 0, 2));
                            ch.pipeline().addLast("msgpack decoder", new
MsgpackDecoder());
                            ch.pipeline().addLast("frameEncoder", new
LengthFieldPrepender(2));
                            ch.pipeline().addLast("msgpack encoder", new
MsgpackEncoder());
                            ch.pipeline().addLast(new EchoServerHandler());
                        }
                    });
            ChannelFuture f = b.bind(port).sync();
            f.channel().closeFuture().sync();
        } finally {
            bossGroup.shutdownGracefully();
            workerGroup.shutdownGracefully();
        }
    }
    public static void main(String[] args) throws Exception {
        int port = 8080;
        if (args != null && args.length > 0) {
            try {
                port = Integer.valueOf(args[0]);
```

```
                } catch (Exception e) {
                }
            }
            new EchoServer().bind(port);
        }
}
```

EchoServerHandler端

```
importio.netty.channel.ChannelHandlerContext;
importio.netty.channel.ChannelInboundHandlerAdapter;
public class EchoServerHandler extends ChannelInboundHandlerAdapter {
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws
Exception {

            System.out.println("service receive the msgpack message :" + msg);
            ctx.writeAndFlush(msg);
    }
    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause)
throws Exception {
            cause.printStackTrace();
            ctx.close();
    }
}
```