Protobuf编解码框架

笔记本：	<Inbox>		
创建时间：	12/5/2018 17:53	更新时间：	7/3/2019 17:31
作者：	wangqinlinmail@163.com		
标签：	Netty		

Protobuf编解码框架

Protobuf是一个灵活的，结构化的数据序列化框架，相比于XML等传统的序列化工具，它更小，更快，更简单，Protobuf支持多种结构化一次可以到处使用，设置跨语言使用，
通过代码生成工具可以自动生成不同语言版本的源代码，甚至可以在使用不同版本的数据结构进程间进行数据传输，实现数据结构的向前兼容；
优点：
1：在谷歌内部长期使用，产品成熟度高
2：跨语言，支持多种语言，包括C++，java和python
3：编码后的消息更小，更加有利存储和传输
4：编解码的性能非常高
5：支持不同协议版本的向前兼容
6：支持定义可选和必选字段

Protobuf开发环境搭建

https://github.com/protocolbuffers/protobuf/releases

定义SubscribeReq.proto

```
syntax = "proto2";

package  com.itheima.netty.protobuf;

option java_package = "com.itheima.netty.protobuf";
option java_outer_classname = "SubscribeReqProto";

message SubscribeReq{
    required int32 subReqID = 1;
    required string username = 2;
    required string productName = 3;
    repeated string address = 4;
}
```

定义SubscribeResp.proto

```
syntax = "proto2";

package  com.itheima.netty.protobuf;

option java_package = "com.itheima.netty.protobuf";
option java_outer_classname = "SubscribeRespProto";

message SubscribeResp{
    required int32 subReqID = 1;
    required int32 respCode = 2;
    required string desc = 3;
}
```

生成java代码：

C:\document\tool\protoc-3.6.1-win32\bin> ./protoc.exe ../proto/SubscribeResp.proto --java_out=../proto/ --proto_path=../proto

pom.xml中配置Protobuf坐标：

```
<dependency>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java</artifactId>
    <version>3.4.0</version>
</dependency>
```

Protobuf编解码开发

1：编写测试程序

使用protoc.exe工具根据SubscribeReq.proto，SubscribeResp.proto的两个文件会生成两个.java文件:

SubscribeReqProto.java

SubscribeRespProto.java

```
import java.util.ArrayList;
import java.util.List;
import com.google.protobuf.InvalidProtocolBufferException;
importcom.itheima.netty.protobuf.SubscribeReqProto.SubscribeReq;
public class TestSubscribeReqProto {
    //编码
    private static byte[] encode(SubscribeReqProto.SubscribeReq req){

          return req.toByteArray();
     }
    //解码
     private static SubscribeReqProto.SubscribeReq decode(byte[] body) throws InvalidProtocolBufferException{

          return SubscribeReqProto.SubscribeReq.parseFrom(body);
     }

     private static SubscribeReqProto.SubscribeReq createSubscribeReqProto(){

          SubscribeReqProto.SubscribeReq.Builder builder = SubscribeReqProto.SubscribeReq.newBuilder();
          builder.setSubReqID(1);
          builder.setUsername("Lilinfeng");
          builder.setProductName("Netty book");
          List<String> address = new ArrayList<String>();
          address.add("beijing");
          address.add("shanghai");
          address.add("shengzheng");
          builder.addAllAddress(address);
          return builder.build();
     }

     public static void main(String[] args) throws InvalidProtocolBufferException {

          SubscribeReqProto.SubscribeReq req = createSubscribeReqProto();
          System.out.println("Before encode :"+ req.toString());
          SubscribeReq req2 = decode(encode(req));
          System.out.println("After decode :"+ req2.toString());
          System.out.println("Assert equal:---> "+ req2.equals(req));
```

```
    }
}
```

Netty的Protobuf服务端开发
SubReqServer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.protobuf.ProtobufDecoder;
import io.netty.handler.codec.protobuf.ProtobufEncoder;
import io.netty.handler.codec.protobuf.ProtobufVarint32FrameDecoder;
import io.netty.handler.codec.protobuf.ProtobufVarint32LengthFieldPrepender;
import io.netty.handler.logging.LogLevel;
import io.netty.handler.logging.LoggingHandler;
public class SubReqServer {

    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,
100)
                    .handler(new
LoggingHandler(LogLevel.INFO)).childHandler(new ChannelInitializer<SocketChannel>
() {
                        @Override
                        protected void initChannel(SocketChannel ch) throws
Exception {
                            ch.pipeline().addLast(new
ProtobufVarint32FrameDecoder());
                            ch.pipeline().addLast(new
ProtobufDecoder(SubscribeReqProto.SubscribeReq.getDefaultInstance()));
                            ch.pipeline().addLast(new
ProtobufVarint32LengthFieldPrepender());
                            ch.pipeline().addLast(new ProtobufEncoder());
                            ch.pipeline().addLast(new
SubReqServerHandler());
                        }
                    });
            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 SubReqServer().bind(port);
    }
}
```

SubReqServerHandler

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

        SubscribeReqProto.SubscribeReq req =
(SubscribeReqProto.SubscribeReq)msg;
        if("LilinFeng".equalsIgnoreCase(req.getUsername())){
            System.out.println("Service accept client subscribe req :["+
req.toString()+"]");
            ctx.writeAndFlush(Resp(req.getSubReqID()));
        }
    }
    private SubscribeRespProto.SubscribeResp Resp(int subReqID){
        SubscribeRespProto.SubscribeResp.Builder builder =
SubscribeRespProto.SubscribeResp.newBuilder();
        builder.setSubReqID(subReqID);
        builder.setRespCode(0);
        builder.setDesc("Netty book order succeed, 3 day later,sent to the
desigated address");
        return builder.build();
    }
    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause)
throws Exception {
        cause.printStackTrace();
        ctx.close();
    }
}
```

Netty的Protobuf客户端开发
SubReqClient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.protobuf.ProtobufDecoder;
importio.netty.handler.codec.protobuf.ProtobufEncoder;
importio.netty.handler.codec.protobuf.ProtobufVarint32FrameDecoder;
importio.netty.handler.codec.protobuf.ProtobufVarint32LengthFieldPrepender;
public class SubReqClient {

    public void connect(int port,String host) 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(new
ProtobufVarint32FrameDecoder());
                            ch.pipeline().addLast(new
ProtobufDecoder(SubscribeRespProto.SubscribeResp.getDefaultInstance()));
                            ch.pipeline().addLast(new
ProtobufVarint32LengthFieldPrepender());
                            ch.pipeline().addLast(new ProtobufEncoder());
                            ch.pipeline().addLast(new
SubReqClientHandle());
                        }
                    });
            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 SubReqClient().connect(8080,"127.0.0.1");
    }
}
```

SubReqClientHandle

```
import java.util.ArrayList;
import java.util.List;
importio.netty.channel.ChannelHandlerContext;
importio.netty.channel.ChannelInboundHandlerAdapter;
public class SubReqClientHandle extends ChannelInboundHandlerAdapter {
    public SubReqClientHandle() {
    }
    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
        for (int i = 0; i < 10; i++) {
            ctx.write(subReq(i));
        }
        ctx.flush();
    }
    private static SubscribeReqProto.SubscribeReq subReq(int i) {
        SubscribeReqProto.SubscribeReq.Builder builder =
SubscribeReqProto.SubscribeReq.newBuilder();
```

```
            builder.setSubReqID(i);
            builder.setUsername("Lilinfeng");
            builder.setProductName("Netty book");
            List<String> address = new ArrayList<String>();
            address.add("beijing");
            address.add("shanghai");
            address.add("shengzheng");
            builder.addAllAddress(address);
            return builder.build();
        }
        @Override
        public void channelRead(ChannelHandlerContext ctx, Object msg) throws
Exception {
            System.out.println("Receive server response :[" + msg + "]");
        }
        @Override
        public void channelReadComplete(ChannelHandlerContext ctx) throws Exception
{
            ctx.flush();
        }
        @Override
        public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause)
throws Exception {
            cause.printStackTrace();
            ctx.close();
        }
}
```

Protobuf的使用注意事项：

ProtobufDecoder仅仅负责解码，它不支持读半包，在ProtobufDecoder前面，一定要有能够处理读半包的解码器，有以下三种方式可以选择：

1：使用Netty提供的ProtobufVarint32FrameDecoder,处理半包

2：继承Netty提供的通用半包解码器LengthFieldBasedFrameDecoder

3：继承ByteToMessageDecoder类，自己处理半包消息