**Netty4.0 学习笔记**

# Server与Client的通讯

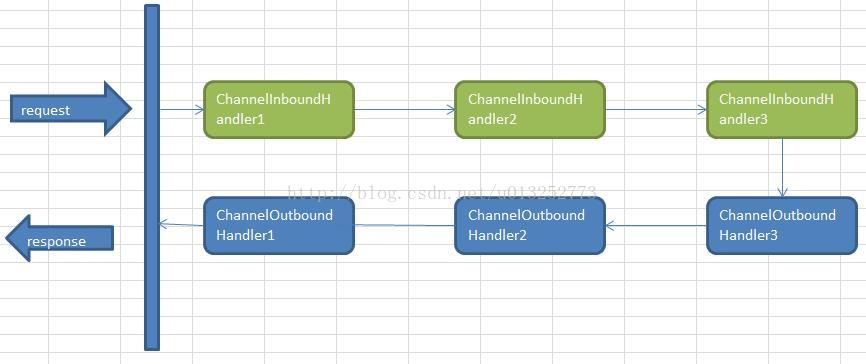
参见it.shanjj.netty4.simpleEx

# Handler的执行顺序

Handler在netty中，无疑占据着非常重要的地位。Handler与Servlet中的filter很像，通过Handler可以完成通讯报文的解码编码、拦截指定的报文、统一对日志错误进行处理、统一对请求进行计数、控制Handler执行与否。一句话，没有它做不到的只有你想不到的。

Netty中的所有handler都实现自ChannelHandler接口。按照输出输出来分，分为ChannelInboundHandler、ChannelOutboundHandler两大类。ChannelInboundHandler对从客户端发往服务器的报文进行处理，一般用来执行解码、读取客户端数据、进行业务处理等；ChannelOutboundHandler对从服务器发往客户端的报文进行处理，一般用来进行编码、发送报文到客户端。

Netty中，可以注册多个handler。ChannelInboundHandler按照注册的先后顺序执行；ChannelOutboundHandler按照注册的先后顺序逆序执行，如下图所示，按照注册的先后顺序对Handler进行排序，request进入Netty后的执行顺序为：



基本的概念就说到这，下面用一个例子来进行验证。该例子模拟Client与Server间的通讯，Server端注册了2个ChannelInboundHandler、2个ChannelOutboundHandler。当Client连接到Server后，会向Server发送一条消息。Server端通过ChannelInboundHandler 对Client发送的消息进行读取，通过ChannelOutboundHandler向client发送消息。最后Client把接收到的信息打印出来。

参见it.shanjj.netty4.handler

**在使用Handler的过程中，需要注意：**

1、ChannelInboundHandler之间的传递，通过调用 ctx.fireChannelRead(msg) 实现；调用ctx.write(msg) 将传递到ChannelOutboundHandler。

2、ctx.write()方法执行后，需要调用flush()方法才能令它立即执行。

3、ChannelOutboundHandler 在注册的时候需要放在最后一个ChannelInboundHandler之前，否则将无法传递到ChannelOutboundHandler。

# 构建简单的http服务

想要实现的目的是：

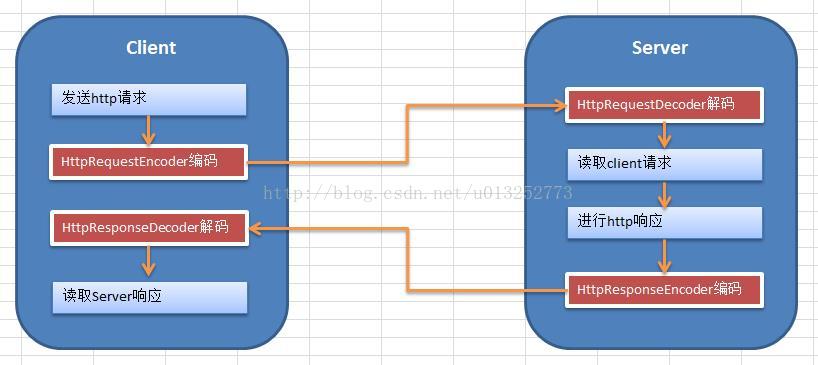
1、Client向Server发送http请求。

2、Server端对http请求进行解析。

3、Server端向client发送http响应。

4、Client对http响应进行解析。

在该实例中，会涉及到http请求的编码、解码，http响应的编码、解码，幸运的是，Netty已经为我们提供了这些工具，整个实例的逻辑图如下所示：



其中红色框中的4个类是Netty提供的，它们其实也是一种Handler，其中Encoder继承自ChannelOutboundHandler，Decoder继承自ChannelInboundHandler，它们的作用是：

1、HttpRequestEncoder：对httpRequest进行编码。

2、HttpRequestDecoder：把流数据解析为httpRequest。

3、HttpResponsetEncoder：对httpResponset进行编码。

4、HttpResponseEncoder：把流数据解析为httpResponse。

参见it.shanjj.netty4.http

**注意事项：**

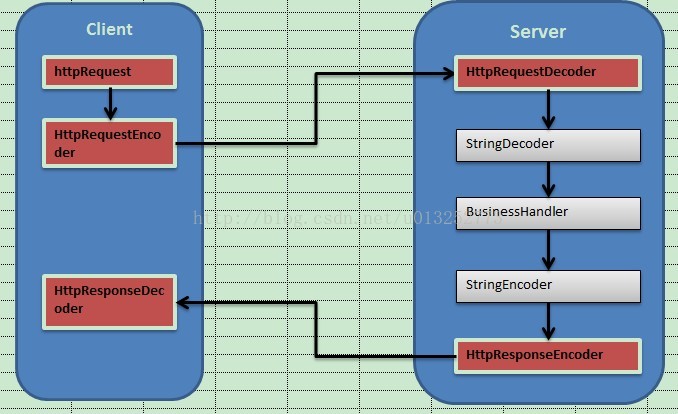
1、可以通过在Netty的Chanel中发送HttpRequest对象，完成发送http请求的要求，同时可以对HttpHeader进行设置。

2、可以通过HttpResponse发送http响应，同时可以对HttpHeader进行设置。

3、上面涉及到的http对象都是Netty自己封装的，不是标准的。

# 混合使用coder和handler

Handler如何使用在前面的例子中已经有了示范，那么同样是扩展自ChannelHandler的Encoder和Decoder，与Handler混合后又是如何使用的？本文将通过一个实际的小例子来展示它们的用法。

该例子模拟一个Server和Client，两者之间通过http协议进行通讯，在Server内部通过一个自定义的StringDecoder把httprequest转换成String。Server端处理完成后，通过StringEncoder把String转换成httpresponse，发送给客户端。具体的处理流程如图所示：

其中红色框中的Decoder、Encoder及request都是Netty框架自带的，灰色框中的三个类是我自己实现的。

Server端的类有：Server、 StringDecoder、 BusinessHandler、 StringEncoder四个类。

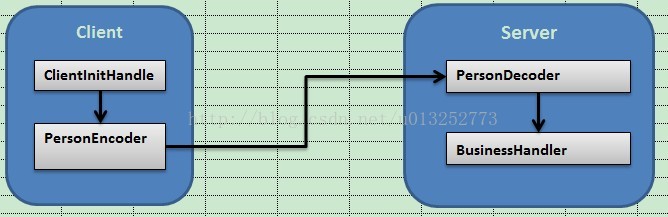
Client端有两个类：Client、 ClientInitHandler 两个类

参见it.shanjj.netty4.handlerCoderMix

# 自定义通讯协议

Netty中，通讯的双方建立连接后，会把数据按照ByteBuf的方式进行传输，例如http协议中，就是通过HttpRequestDecoder对ByteBuf数据流进行处理，转换成http的对象。基于这个思路，我自定义一种通讯协议：Server和客户端直接传输Java对象。

实现的原理是通过Encoder把java对象转换成ByteBuf流进行传输，通过Decoder把ByteBuf转换成java对象进行处理，处理逻辑如下图所示：



参见it.shanjj.netty4.definedProtocol

通过上述代码，实现了Server端与Client端直接使用person对象进行通信的目的。基于此，可以构建更为复杂的场景：Server端同时支撑多种协议，不同的协议采用不同的Decoder进行解析，解析结果保持统一，这样业务处理类可以保持接口一致。下一节将编写这样一个案例。

本例中需要注意的事项是：

1、Person对象必须实现Serializable接口，否则不能进行序列化。

2、PersonDecoder读取ByteBuf数据的时候，并没有对多次流式数据进行处理，而是简单的一次性接收，如果数据量大的情况下，可能会出现数据不完整，这个问题会在后续的学习中解决。