* 
* [写博客](https://mp.csdn.net/postedit)
* [发Chat](http://gitbook.cn/new/gitchat/activity?utm_source=csdnblog1)
* [[](https://my.csdn.net/)](https://my.csdn.net/" \t "_blank)

**Netty4.0学习笔记系列之二：Handler的执行顺序**

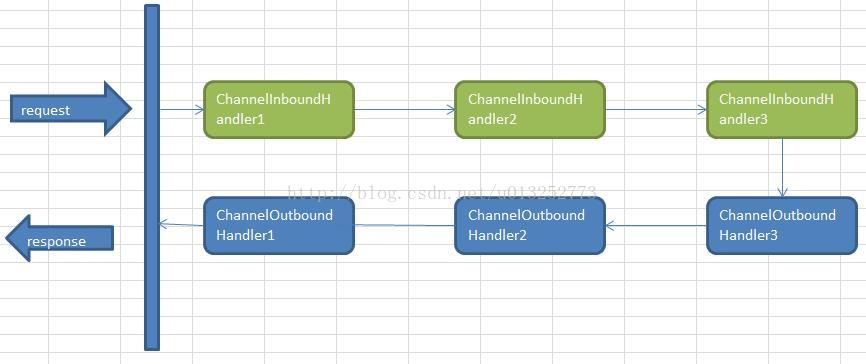
原创 2014年03月13日 23:32:28

* 标签：
* [netty4.0](http://so.csdn.net/so/search/s.do?q=netty4.0&t=blog) /
* [handler](http://so.csdn.net/so/search/s.do?q=handler&t=blog)
* 29600

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。

Server端一共有5个类：HelloServer InboundHandler1 InboundHandler2 OutboundHandler1 OutboundHandler2

1、HelloServer 代码如下

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

1. **package** com.guowl.testmultihandler;
3. **import** io.netty.bootstrap.ServerBootstrap;
4. **import** io.netty.channel.ChannelFuture;
5. **import** io.netty.channel.ChannelInitializer;
6. **import** io.netty.channel.ChannelOption;
7. **import** io.netty.channel.EventLoopGroup;
8. **import** io.netty.channel.nio.NioEventLoopGroup;
9. **import** io.netty.channel.socket.SocketChannel;
10. **import** io.netty.channel.socket.nio.NioServerSocketChannel;
12. **public** **class** HelloServer {
13. **public** **void** start(**int** port) **throws** Exception {
14. EventLoopGroup bossGroup = **new** NioEventLoopGroup();
15. EventLoopGroup workerGroup = **new** NioEventLoopGroup();
16. **try** {
17. ServerBootstrap b = **new** ServerBootstrap();
18. b.group(bossGroup, workerGroup).channel(NioServerSocketChannel.**class**)
19. .childHandler(**new** ChannelInitializer<SocketChannel>() {
20. @Override
21. **public** **void** initChannel(SocketChannel ch) **throws** Exception {
22. // 注册两个OutboundHandler，执行顺序为注册顺序的逆序，所以应该是OutboundHandler2 OutboundHandler1
23. ch.pipeline().addLast(**new** OutboundHandler1());
24. ch.pipeline().addLast(**new** OutboundHandler2());
25. // 注册两个InboundHandler，执行顺序为注册顺序，所以应该是InboundHandler1 InboundHandler2
26. ch.pipeline().addLast(**new** InboundHandler1());
27. ch.pipeline().addLast(**new** InboundHandler2());
28. }
29. }).option(ChannelOption.SO\_BACKLOG, 128)
30. .childOption(ChannelOption.SO\_KEEPALIVE, **true**);
32. ChannelFuture f = b.bind(port).sync();
34. f.channel().closeFuture().sync();
35. } **finally** {
36. workerGroup.shutdownGracefully();
37. bossGroup.shutdownGracefully();
38. }
39. }
41. **public** **static** **void** main(String[] args) **throws** Exception {
42. HelloServer server = **new** HelloServer();
43. server.start(8000);
44. }
45. }

2、InboundHandler1

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

1. **package** com.guowl.testmultihandler;
3. **import** io.netty.channel.ChannelHandlerContext;
4. **import** io.netty.channel.ChannelInboundHandlerAdapter;
6. **import** org.slf4j.Logger;
7. **import** org.slf4j.LoggerFactory;
9. **public** **class** InboundHandler1 **extends** ChannelInboundHandlerAdapter {
10. **private** **static** Logger   logger  = LoggerFactory.getLogger(InboundHandler1.**class**);
12. @Override
13. **public** **void** channelRead(ChannelHandlerContext ctx, Object msg) **throws** Exception {
14. logger.info("InboundHandler1.channelRead: ctx :" + ctx);
15. // 通知执行下一个InboundHandler
16. ctx.fireChannelRead(msg);
17. }
19. @Override
20. **public** **void** channelReadComplete(ChannelHandlerContext ctx) **throws** Exception {
21. logger.info("InboundHandler1.channelReadComplete");
22. ctx.flush();
23. }
24. }

3、InboundHandler2

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

1. **package** com.guowl.testmultihandler;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.channel.ChannelHandlerContext;
5. **import** io.netty.channel.ChannelInboundHandlerAdapter;
7. **import** org.slf4j.Logger;
8. **import** org.slf4j.LoggerFactory;
10. **public** **class** InboundHandler2 **extends** ChannelInboundHandlerAdapter {
11. **private** **static** Logger   logger  = LoggerFactory.getLogger(InboundHandler2.**class**);
13. @Override
14. // 读取Client发送的信息，并打印出来
15. **public** **void** channelRead(ChannelHandlerContext ctx, Object msg) **throws** Exception {
16. logger.info("InboundHandler2.channelRead: ctx :" + ctx);
17. ByteBuf result = (ByteBuf) msg;
18. **byte**[] result1 = **new** **byte**[result.readableBytes()];
19. result.readBytes(result1);
20. String resultStr = **new** String(result1);
21. System.out.println("Client said:" + resultStr);
22. result.release();
24. ctx.write(msg);
25. }
27. @Override
28. **public** **void** channelReadComplete(ChannelHandlerContext ctx) **throws** Exception {
29. logger.info("InboundHandler2.channelReadComplete");
30. ctx.flush();
31. }
33. }

4、OutboundHandler1

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

1. **package** com.guowl.testmultihandler;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.channel.ChannelHandlerContext;
5. **import** io.netty.channel.ChannelOutboundHandlerAdapter;
6. **import** io.netty.channel.ChannelPromise;
8. **import** org.slf4j.Logger;
9. **import** org.slf4j.LoggerFactory;
11. **public** **class** OutboundHandler1 **extends** ChannelOutboundHandlerAdapter {
12. **private** **static** Logger   logger  = LoggerFactory.getLogger(OutboundHandler1.**class**);
13. @Override
14. // 向client发送消息
15. **public** **void** write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) **throws** Exception {
16. logger.info("OutboundHandler1.write");
17. String response = "I am ok!";
18. ByteBuf encoded = ctx.alloc().buffer(4 \* response.length());
19. encoded.writeBytes(response.getBytes());
20. ctx.write(encoded);
21. ctx.flush();
22. }

25. }

5、OutboundHandler2

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

1. **package** com.guowl.testmultihandler;
3. **import** io.netty.channel.ChannelHandlerContext;
4. **import** io.netty.channel.ChannelOutboundHandlerAdapter;
5. **import** io.netty.channel.ChannelPromise;
7. **import** org.slf4j.Logger;
8. **import** org.slf4j.LoggerFactory;
10. **public** **class** OutboundHandler2 **extends** ChannelOutboundHandlerAdapter {
11. **private** **static** Logger   logger  = LoggerFactory.getLogger(OutboundHandler2.**class**);
13. @Override
14. **public** **void** write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) **throws** Exception {
15. logger.info("OutboundHandler2.write");
16. // 执行下一个OutboundHandler
17. **super**.write(ctx, msg, promise);
18. }
19. }

Client端有两个类：HelloClient  HelloClientIntHandler

1、HelloClient

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

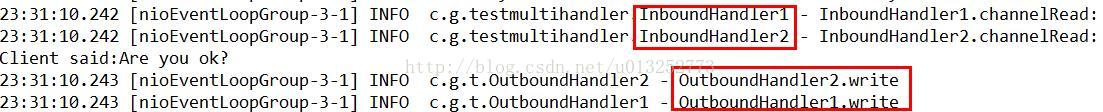
1. **package** com.guowl.testmultihandler;
3. **import** io.netty.bootstrap.Bootstrap;
4. **import** io.netty.channel.ChannelFuture;
5. **import** io.netty.channel.ChannelInitializer;
6. **import** io.netty.channel.ChannelOption;
7. **import** io.netty.channel.EventLoopGroup;
8. **import** io.netty.channel.nio.NioEventLoopGroup;
9. **import** io.netty.channel.socket.SocketChannel;
10. **import** io.netty.channel.socket.nio.NioSocketChannel;
12. **public** **class** HelloClient {
13. **public** **void** connect(String host, **int** port) **throws** Exception {
14. EventLoopGroup workerGroup = **new** NioEventLoopGroup();
16. **try** {
17. Bootstrap b = **new** Bootstrap();
18. b.group(workerGroup);
19. b.channel(NioSocketChannel.**class**);
20. b.option(ChannelOption.SO\_KEEPALIVE, **true**);
21. b.handler(**new** ChannelInitializer<SocketChannel>() {
22. @Override
23. **public** **void** initChannel(SocketChannel ch) **throws** Exception {
24. ch.pipeline().addLast(**new** HelloClientIntHandler());
25. }
26. });
28. // Start the client.
29. ChannelFuture f = b.connect(host, port).sync();
30. f.channel().closeFuture().sync();
31. } **finally** {
32. workerGroup.shutdownGracefully();
33. }
34. }
36. **public** **static** **void** main(String[] args) **throws** Exception {
37. HelloClient client = **new** HelloClient();
38. client.connect("127.0.0.1", 8000);
39. }
40. }

2、HelloClientIntHandler

**[java]** [view plain](https://blog.csdn.net/u013252773/article/details/21195593) [copy](https://blog.csdn.net/u013252773/article/details/21195593)

1. **package** com.guowl.testmultihandler;
3. **import** io.netty.buffer.ByteBuf;
4. **import** io.netty.channel.ChannelHandlerContext;
5. **import** io.netty.channel.ChannelInboundHandlerAdapter;
7. **import** org.slf4j.Logger;
8. **import** org.slf4j.LoggerFactory;
10. **public** **class** HelloClientIntHandler **extends** ChannelInboundHandlerAdapter {
11. **private** **static** Logger   logger  = LoggerFactory.getLogger(HelloClientIntHandler.**class**);
12. @Override
13. // 读取服务端的信息
14. **public** **void** channelRead(ChannelHandlerContext ctx, Object msg) **throws** Exception {
15. logger.info("HelloClientIntHandler.channelRead");
16. ByteBuf result = (ByteBuf) msg;
17. **byte**[] result1 = **new** **byte**[result.readableBytes()];
18. result.readBytes(result1);
19. result.release();
20. ctx.close();
21. System.out.println("Server said:" + **new** String(result1));
22. }
23. @Override
24. // 当连接建立的时候向服务端发送消息 ，channelActive 事件当连接建立的时候会触发
25. **public** **void** channelActive(ChannelHandlerContext ctx) **throws** Exception {
26. logger.info("HelloClientIntHandler.channelActive");
27. String msg = "Are you ok?";
28. ByteBuf encoded = ctx.alloc().buffer(4 \* msg.length());
29. encoded.writeBytes(msg.getBytes());
30. ctx.write(encoded);
31. ctx.flush();
32. }
33. }

server端执行结果为：



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。

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u013252773/article/details/2119559