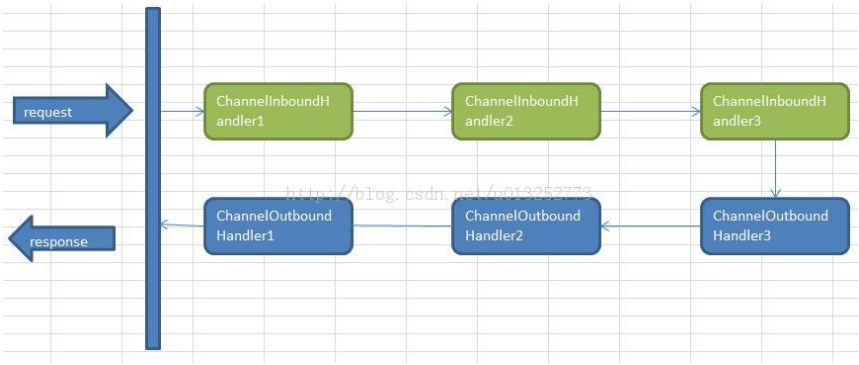
转 netty 学习 （2）Handler的执行顺序

摘要: 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。 参考自：http://blog.csdn.net/u013252773/article/details/21195593

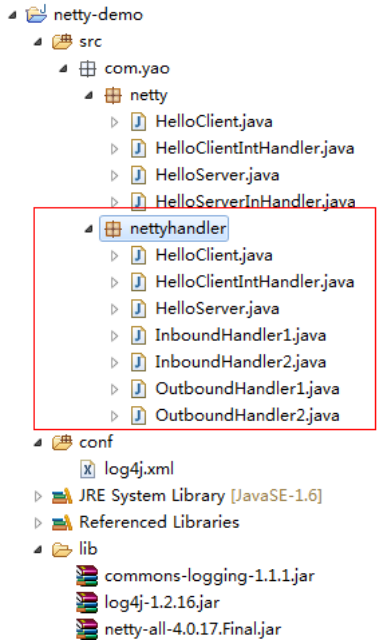
Handler在netty中，无疑占据着非常重要的地位。Handler与Servlet中的filter很像，通过Handler可以完成通讯报文的解码编码、拦截指定的报文、统一对日志错误进行处理、统一对请求进行计数、控制Handler执行与否。一句话，没有它做不到的只有你想不到的。

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：



下面例子涉及的类包括：



一、HelloServer：

```
package com.yao.nettyhandler;

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;

public class HelloServer {
        public void start(int port) throws Exception {
                EventLoopGroup bossGroup = new NioEventLoopGroup();
                EventLoopGroup workerGroup = new NioEventLoopGroup();
                try {
                        ServerBootstrap b = new ServerBootstrap();
                        b.group(bossGroup, workerGroup).channel(NioServerSocketChannel.class)
                                        .childHandler(new ChannelInitializer<SocketChannel>() {
                                                        @Override
                                                        public void initChannel(SocketChannel ch) thro
                                                                // 注册两个OutboundHandler，执行顺序为注
                                                                ch.pipeline().addLast(new OutboundHand
                                                                ch.pipeline().addLast(new OutboundHand
                                                                // 注册两个InboundHandler，执行顺序为注册
                                                                ch.pipeline().addLast(new InboundHandl
                                                                ch.pipeline().addLast(new InboundHandl
                                                        }
                                                }).option(ChannelOption.SO_BACKLOG, 128)
                                        .childOption(ChannelOption.SO_KEEPALIVE, true);

                        ChannelFuture f = b.bind(port).sync();

                        f.channel().closeFuture().sync();
                } finally {
                        workerGroup.shutdownGracefully();
                        bossGroup.shutdownGracefully();
                }
        }

        public static void main(String[] args) throws Exception {
                HelloServer server = new HelloServer();
                server.start(8000);
        }
}
```

二、InboundHandler1：

```
package com.yao.nettyhandler;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class InboundHandler1 extends ChannelInboundHandlerAdapter {
        private static Log logger = LogFactory.getLog(InboundHandler1.class);

        @Override
        public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
                logger.info("InboundHandler1.channelRead: ctx :" + ctx);

                // 通知执行下一个InboundHandler
                //ctx.fireChannelRead(msg);
        }

        @Override
        public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {
                logger.info("InboundHandler1.channelReadComplete");
                ctx.flush();
        }
}
```

三、InboundHandler2：

```
package com.yao.nettyhandler;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class InboundHandler2 extends ChannelInboundHandlerAdapter {
        private static Log      logger  = LogFactory.getLog(InboundHandler2.class);

        @Override
        // 读取Client发送的信息，并打印出来
        public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
                logger.info("InboundHandler2.channelRead: ctx :" + ctx);
                ByteBuf result = (ByteBuf) msg;
                byte[] result1 = new byte[result.readableBytes()];
                result.readBytes(result1);
                String resultStr = new String(result1);
                System.out.println("Client said:" + resultStr);
                result.release();

                ctx.write(msg);
        }

        @Override
        public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {
                logger.info("InboundHandler2.channelReadComplete");
                ctx.flush();
        }

}
```

四、OutboundHandler1 ：

```
package com.yao.nettyhandler;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelOutboundHandlerAdapter;
import io.netty.channel.ChannelPromise;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
```

```
public class OutboundHandler1 extends ChannelOutboundHandlerAdapter {
        private static Log      logger  = LogFactory.getLog(OutboundHandler1.class);
        @Override
        // 向client发送消息
        public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception {
                logger.info("OutboundHandler1.write");
                String response = "I am ok!";
                ByteBuf encoded = ctx.alloc().buffer(4 * response.length());
                encoded.writeBytes(response.getBytes());
                ctx.write(encoded);
                ctx.flush();
        }


}
```

五、OutboundHandler2：

```
package com.yao.nettyhandler;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelOutboundHandlerAdapter;
import io.netty.channel.ChannelPromise;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;


public class OutboundHandler2 extends ChannelOutboundHandlerAdapter {
        private static Log      logger  = LogFactory.getLog(OutboundHandler2.class);

        @Override
        public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception {
                logger.info("OutboundHandler2.write");
                // 执行下一个OutboundHandler
                super.write(ctx, msg, promise);
        }
}
```

下面是客户端

六、HelloClient：

```
package com.yao.nettyhandler;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioSocketChannel;

public class HelloClient {
        public void connect(String host, int port) throws Exception {
                EventLoopGroup workerGroup = new NioEventLoopGroup();

                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(new HelloClientIntHandler());
                                }
                        });
```

```
                // Start the client.
                ChannelFuture f = b.connect(host, port).sync();
                f.channel().closeFuture().sync();
            } finally {
                workerGroup.shutdownGracefully();
            }
        }

        public static void main(String[] args) throws Exception {
            HelloClient client = new HelloClient();
            client.connect("127.0.0.1", 8000);
        }
}
```

七、HelloClientIntHandler：

```
package com.yao.nettyhandler;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class HelloClientIntHandler extends ChannelInboundHandlerAdapter {
        private static Log      logger  = LogFactory.getLog(HelloClientIntHandler.class);
        @Override
        // 读取服务端的信息
        public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
                logger.info("HelloClientIntHandler.channelRead");
                ByteBuf result = (ByteBuf) msg;
                byte[] result1 = new byte[result.readableBytes()];
                result.readBytes(result1);
                result.release();
                ctx.close();
                System.out.println("Server said:" + new String(result1));
        }
        @Override
        // 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
        public void channelActive(ChannelHandlerContext ctx) throws Exception {
                logger.info("HelloClientIntHandler.channelActive");
                String msg = "Are you ok?";
                ByteBuf encoded = ctx.alloc().buffer(4 * msg.length());
                encoded.writeBytes(msg.getBytes());
                ctx.write(encoded);
                ctx.flush();
        }
}
```

八、总结：

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。

4、Handler的消费处理放在最后一个处理。

分类：netty4　字数：1120

yaokangjun

程序员 广州

+ 关注

粉丝 21 | 博文 62 | 码字总数 29408

相关博客

netty 学习 （4）混合使用coder和handler
yaokangjun
5816 0

Netty4.0学习笔记系列之二：Handler的执行顺序
如月王子
166 0

Netty4.0学习笔记系列之四：混合使用coder和handler
如月王子
155 0

评论 (10)

Ctrl+Enter 发表评论

剑指天涯
1楼 2015/05/30 21:37
茅厕顿开，非常感谢！

引鸩怼子
2楼 2015/09/20 20:02
茅厕顿开，非常感谢！0

itlikejava
3楼 2015/11/30 13:25
Handler与Servlet中的filter很像，这句话的前提是在一个channel 里面，

387951323
4楼 2016/03/23 20:44
> 引用来自“剑指天涯”的评论
> 茅厕顿开，非常感谢！

非常好

墨竹
5楼 2016/07/06 11:07
总结的不错

阔以编码的二胡选手
6楼 2016/07/19 16:47
突然茅塞顿开

wangao029
7楼 2016/08/01 11:53
我觉得 Handler 是一个用来做回调的类，也就是回调函数。或者可以理解它为一种观察者模式

阔以编码的二胡选手
8楼 2016/08/02 14:28
> 引用来自“wangao029”的评论
> 我觉得 Handler 是一个用来做回调的类，也就是回调函数。或者可以理解它为一种观察者模式

这位兄台说的让我更加茅塞顿开，惊天地泣鬼神，正瞌睡也不瞌睡，一口气能敲五百航代码了，也不喘气。

老板来瓶82年雪碧

9楼 2016/08/04 15:17

讲解的很清晰，非常明白。感谢

目光冰凉

10楼 2016/11/15 16:35

6