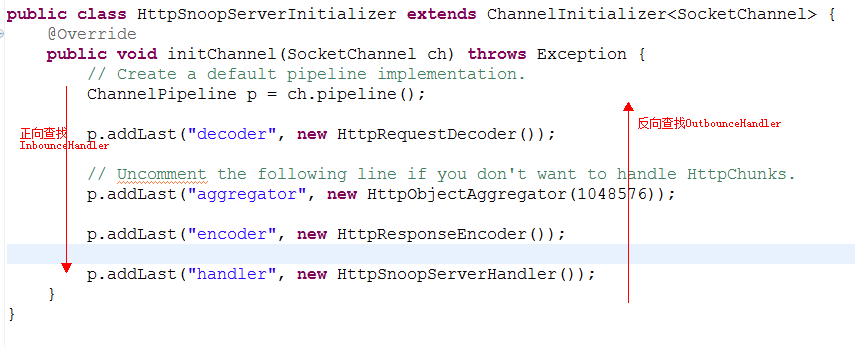
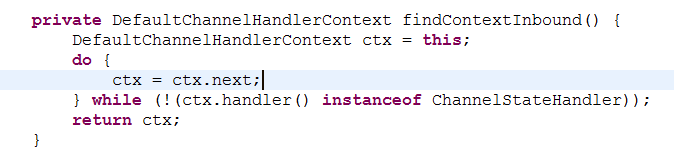
Netty4 ChannelPipeLine分析

**ChannelPipeLine查找ChannelHandler的顺序:**



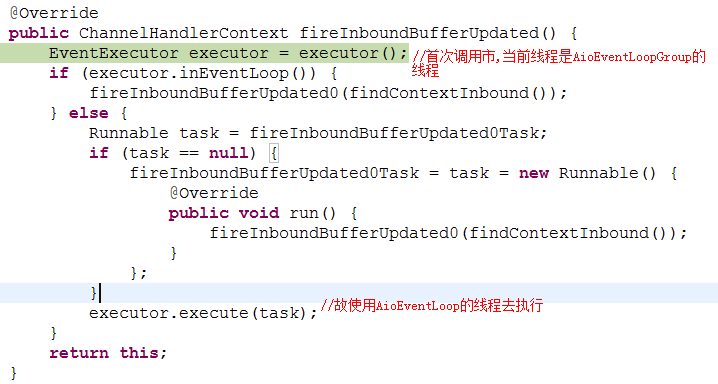
**[DefaultChannelHandlerContext](eclipse-javadoc:%E2%98%82=%20/%3C%7BDefaultChannelHandlerContext.java%E2%98%83DefaultChannelHandlerContext).findContextInbound()方法表示正向查找**

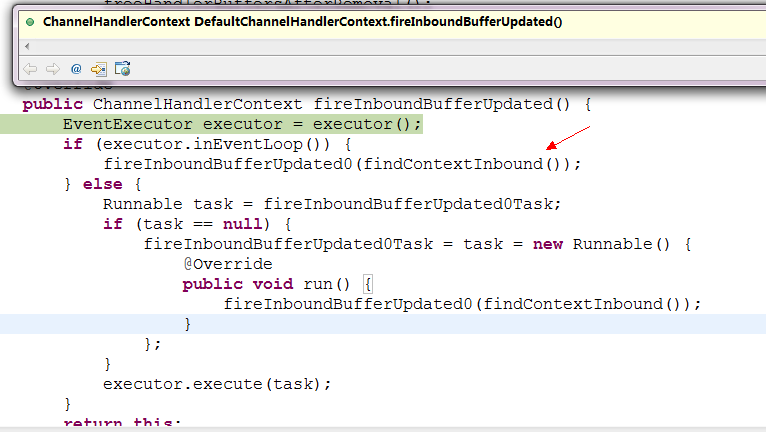
**ChannelStateHandler的实例:**



Netty 4 的 ChannelHandler(s)是以链表的形式连接在一起的,

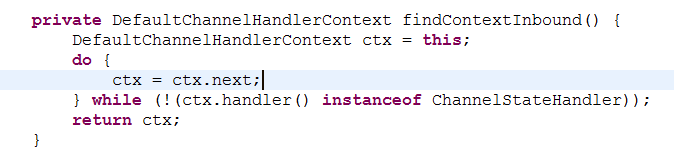
当数据达到时,ChannelPipeline.head(ChannelHandlerContext(含有HeadHandler) )的fireInboundBufferUpdated方法会被调用,现在来看看[DefaultChannelHandlerContext](eclipse-javadoc:%E2%98%82=%20/%3C%7BDefaultChannelHandlerContext.java%E2%98%83DefaultChannelHandlerContext).fireInboundBufferUpdated() 方法:



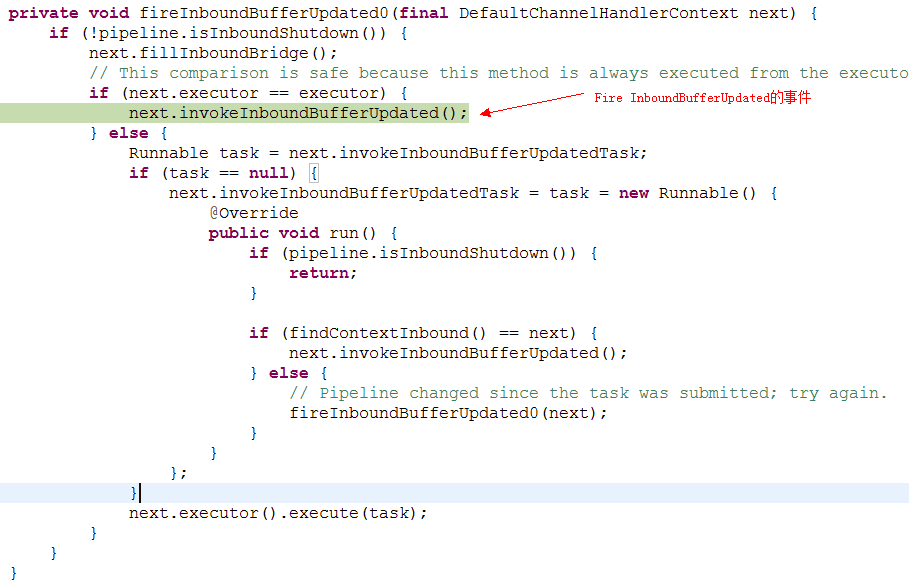


[**Default****ChannelHandlerContext**](eclipse-javadoc:%E2%98%82=%20/%3C%7BDefaultChannelHandlerContext.java%E2%98%83DefaultChannelHandlerContext)**.findContextInbound()方法表示正向查找**

**ChannelStateHandler的实例:**

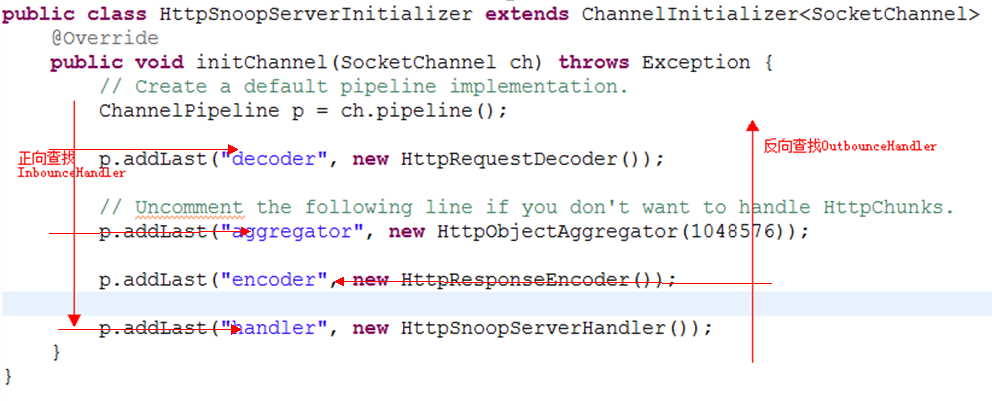


**找到ChannelStateHandler后,执行其ChannelHandlerContext的invokeInboundBufferUpdated事件**



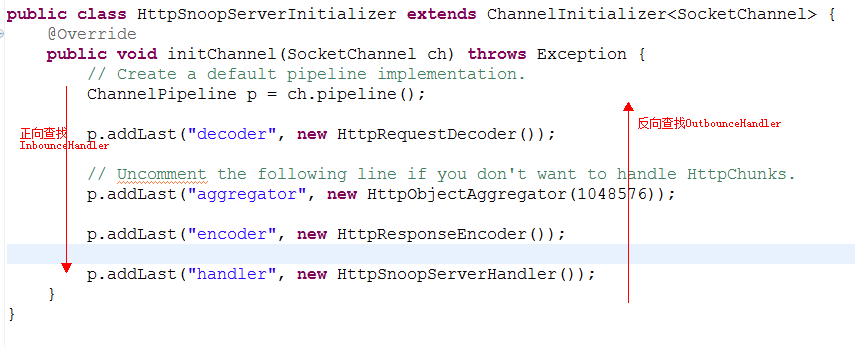
结论:　ChannelPipeLine中的ChannelHandler(s)由其对应的ChannelHandlerContext(s)之间形成了链式调用关系:

先是ChannelInboundHandler(s)对应的ChannelHandlerContext(s)之间形成了对invokeInboundBufferUpdated链式调用关系(此时,非ChannelInboundHandler类型的会被跳过),如图:



其中正方法上decoder,aggregator,handler的inboundBufferUpdated方法会被调用 ,而因为encoder不是ChannelStateHandler所以会被跳过.

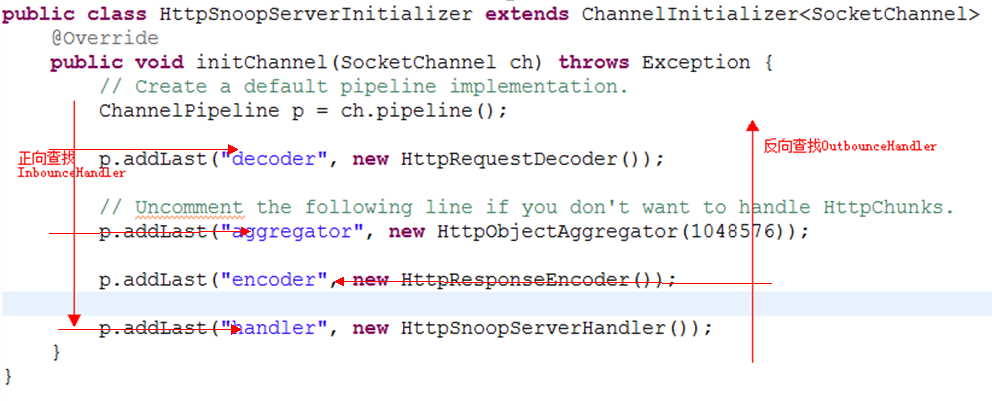
**故验证了以下的结论: ChannelPipeLine中ChannelHandler的执行顺序:**



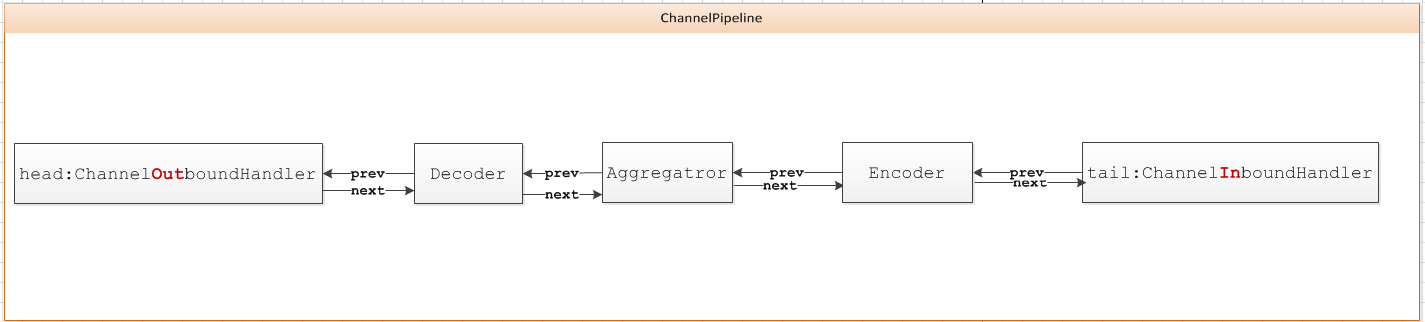
Nettyt利用一个ThreadLocal的Buffer来 **Share same ThreadLocal for all decoder/encoders to minimize memory usage**

关于Netty的Zero-Copy的方式:

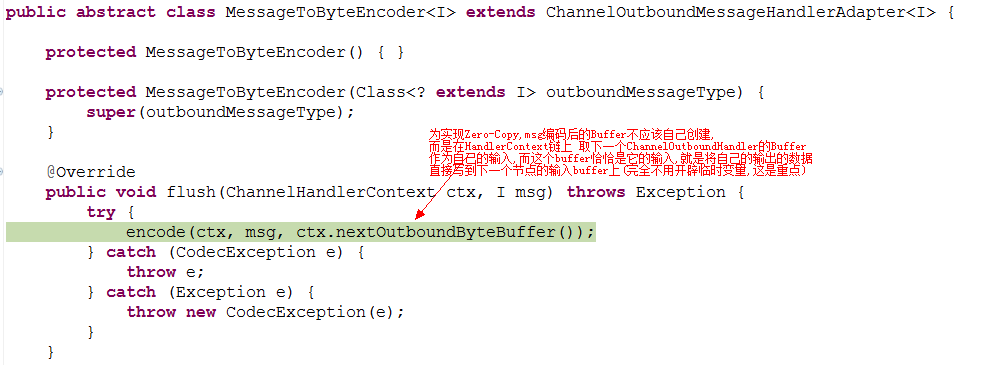
1: 当ChannelInboundHandler



以下为这条处理器链的排列情况:



现在来看看HttpResponseEncoder的执行情况,此时它将要执行编码操作,它**取它的下一个ChannelOutboundHandler(HeadHandler)的输入buffer,作为自己的输出buffer,**中间完全不用临时生成一个buffer,这是好想法!! 而说回inbound方向上也是一样的,**decoder将自己的输入结果直接写到agregator的输入buffer上**,而aggregator也将输出结果直接写到handler的输入buffer上,完成不用开辟临时buffer. 所以**数据流在整个inbound到outbound过程中,是完全不需要开辟新buffer的**,这节省了好多 空间开辟与释放的开销. 在其中一些细节上,Netty也可以在同一次ChannelHandler链的执行过程中,共用一个ThreadLocal<OutputMessageBuf>来避免Buf的创建.



ChannelHandlerContext中的关键方法

