Netty重点

1. **ByteBuffer**

ByteBuf outBuf=UnpooledByteBufAllocator.DEFAULT.heapBuffer();性能一般

ByteBuf outBuf=PooledByteBufAllocator.DEFAULT.heapBuffer();性能一般

ByteBuf outBuf=PooledByteBufAllocator.DEFAULT.buffer();性能较好

ByteBuf outBuf=UnpooledByteBufAllocator.DEFAULT.buffer();性能较好

1. **引用计数**

**引用计数：**  
netty中使用引用计数机制来管理资源,当一个实现ReferenceCounted的

对象实例化时, 引用计数置1.客户代码中需要保持一个该对象的引用时需

要调用接口的retain方法将计数增1.对象使用完毕时调用release将计数1，

当引用计数变为0时,对象将释放所持有的底层资源或将资源返回资源池。

**内存泄露**

按上述规则使用Direct和Pooled的ByteBuf尤其重要.对于DirectBuf,其内存不受、VM垃圾回收控制只有在调用release导致计数为0时才会主动释放内存,而PooledByteBuf只有在release后才能被回收到池中以循环利用.如果客户代码没有按引用计数规则使用这两种对象,将会导致内存泄露.

1. **Channelhandler**

**SimpleChannelInboundHandler<T>：**

处理入站的数据我们只需要重新实现channelRead0方法，当channelRead真正被调用的时候我们的逻辑才会被处理。这里使用的是模板模式，让主要的处理逻辑保持不变，让变化的步骤通过接口实现来完成

值得注意的是对于SimpleChannelInboundHandler入站的数据，当被读取之后可能会执行ReferenceCountUtil.release(msg)释放资源。底层是实现ReferenceCounted，当新的对象初始化的时候计数为1，retain()方法实现其他地方的引用计数加1，release()方法实现应用减一，当计数减少到0的时候会被显示清除，再次访问被清除的对象会出现访问冲突。因此，当我们实现自己的Handler的时候如果希望将客户端发送过来的数据发送到客户端，可能在上述finally中已经释放了资源（writeAndFlush是异步处理），所以会出现异常情况。

**ChannelInboundHandler：**

重写channelRead方法时，需要释放ByteBuf相关的内存，可以使用Netty提供了一个工具方法，ReferenceCountUtil.release()