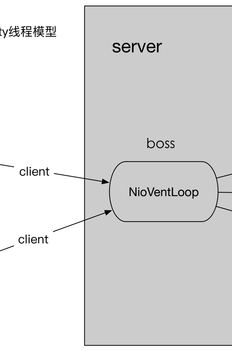
Netty学习笔记

```
public class NettyServer {

    public static void main(String[] args) throws InterruptedException {

        EventLoopGroup bossGroup = new NioEventLoopGroup( nThreads: 1, new MyThreadFactory( threadNamePrefix: "Boss-"));
        EventLoopGroup workerGroup = new NioEventLoopGroup( nThreads: 4, new MyThreadFactory( threadNamePrefix: "Worker-"));

        //woker线程 构建socketchannel时候，通信必须走的一层chandler
        ChannelInitializer<SocketChannel> channelInitializer = new ChannelInitializer<SocketChannel>() {
            @Override
            public void initChannel(SocketChannel ch) throws Exception {

                ChannelPipeline pipeline = ch.pipeline();
                pipeline.addLast( name: "encode", new StringEncoder());
                pipeline.addLast( name: "decode", new StringDecoder());
                pipeline.addLast( name: "handler", new NettyServerHandler());
            }
        };

        ServerBootstrap serverBootstrap = new ServerBootstrap();
        ServerBootstrap handler = serverBootstrap.group(bossGroup, workerGroup)
                .option(ChannelOption.SO_BACKLOG,  value: 4096)
                .option(ChannelOption.SO_REUSEADDR,  value: true)
                .option(ChannelOption.SO_KEEPALIVE,  value: false)
                .option(ChannelOption.SO_SNDBUF,  value: 65536)
                .option(ChannelOption.SO_RCVBUF,  value: 65536)
                .childOption(ChannelOption.TCP_NODELAY,  value: true)
                .channel(NioServerSocketChannel.class)
                .childHandler(channelInitializer);

        handler.bind(new InetSocketAddress( hostname: "127.0.0.1",  port: 9999)).sync();

    }

    static class NettyServerHandler extends SimpleChannelInboundHandler<String> {

        @Override
        public void channelRegistered(ChannelHandlerContext ctx) throws Exception {
            System.out.println("threadName:" + Thread.currentThread().getName() + "   客户端注册上来了");
            super.channelRegistered(ctx);
        }

        @Override
        protected void channelRead0(ChannelHandlerContext ctx, String msg) throws Exception {

            System.out.println("threadName:" + Thread.currentThread().getName() + "   服务端接受到消息--->" + msg.toString());

            ctx.channel().writeAndFlush( msg: "你好啊 客户端小姐姐");

        }

    }
}
```



NioEventLoop（死循环）

1、`provider.openSelector().select()`
2、processSelectedKeys
3、run taskQueue

注意:
a.指定时间内发生select空轮训超过512次，`rebuildSelector();`
b.当有task任务，selectNow()

```
     * around the infamous epoll 100% CPU bug.
     */
    public void rebuildSelector() {
        if (!inEventLoop()) {
            execute(new Runnable() {
                @Override
                public void run() { rebuildSelector0(); }
            });
            return;
        }
        rebuildSelector0();
    }
```

NioEventLoop的常见写法

if(当前线程不是NioEventLoop线程){
    需要封装成task提交到NioEventLoop的taskQueue中等待执行
}else{
    直接执行
}

每个客户端连接上来都会经历boss-Pipeline

client和server端通信发送数据会经理worker-Pipeline

boss线程处理selectedKeys:【连接事件】
io.netty.channel.nio.AbstractNioMessageChannel.NioMessageUnsafe#read

1、获取注册上来的java_channel,封装成NioSocketChannel
2、pipeline.fireChannelRead(channel); 经过pipeline把channel交给worker线程
3、

---

worker线程处理selectedKeys【IO事件】
io.netty.channel.nio.AbstractNioByteChannel.NioByteUnsafe#read

1、读取java_channel通道中的数据
2、把数据发给pipeline读取
3、解码—>业务逻辑处理—>编码—>发送【黏包拆包处理】

疑问点：

1、如何拆包？
　简单来说，收到数据后，判断是否符合一个正常的数据包格式，如果不符合，继续等待接受数据，然后再判断。

2、server端通过channel发送消息给client端如何处理的？
　ctx.channel().writeAndFlush("你好啊 客户端小姐姐"); //哇哦 这里面的代码需要研究研究哦
　简单理解就是pipeline中找outBound，一个个执行invokeWriteAndFlush
　最后调用到head头，往ChannelOutboundBuffer写数据, flush的时候才是把buffer中的数据写到javaChannel

3、netty零copy？
　组合流、直接内存

4、业务线程直接在handler里面处理么？
　如果直接在handler里面处理，使用的是NioEventLoop的线程，业务复杂，会占用处理事件，那么这条线程IO处理的性能/频率 会下降，并发量上不去
　dubbo/rocketmq handler中拿到接收到的消息后，直接给业务线程池处理，处理完channel直接回写回去。
　netty支持在handler中传入事件处理器，这样的话，这个handler对应的操作 就会在指定的线程池中处理。

　那么netty的这种方式和dubbo/rocketmq这种方式有啥不同呢？ 为什么阿里的小伙伴不直接使用netty提供的这个方案呢？
思考，我们可以一起讨论。

福利：强烈推荐netty官方的文章（简书 闪电侠这个兄弟 总结的非常到位）： https://netty.io/wiki/related-articles.html