1、概述

在netty 启动的时候会设置相关的ChannelOption, 无论是在ServerBootstrap还是在Bootstrap,

接下来解释一下常用的ChannelOption

2、常用ChannelOption

ChannelOption.SO\_BACKLOG (一般用于option–>boss)

BACKLOG用于构造服务端套接字ServerSocket对象，标识当服务器请求处理线程都处于工作是(用完了)，用于临时存放已完成三次握手的请求的队列的最大长度。如果未设置或所设置的值小于1，Java将使用默认值50。

ChannelOption.SO\_REUSEADDR (一般用于option–>boss)

SO\_REUSEADDR 对应的是socket选项中SO\_REUSEADDR，这个参数表示允许重复使用本地地址和端口，例如，某个服务占用了TCP的8080端口，其他服务再对这个端口进行监听就会报错，SO\_REUSEADDR这个参数就是用来解决这个问题的，该参数允许服务公用一个端口，这个在服务器程序中比较常用，例如某个进程非正常退出，对一个端口的占用可能不会立即释放，这时候如果不设置这个参数，其他进程就不能立即使用这个端口。

ChannelOption.ALLOCATOR

Netty参数，ByteBuf的分配器(重用缓冲区)，默认值为ByteBufAllocator.DEFAULT，4.0版本为UnpooledByteBufAllocator，4.1版本为PooledByteBufAllocator。该值也可以使用系统参数io.netty.allocator.type配置，使用字符串值：“unpooled”，“pooled”。

额外解释， Netty4.1使用对象池，重用缓冲区(可以直接只用这个配置)

bootstrap.option(ChannelOption.ALLOCATOR, PooledByteBufAllocator.DEFAULT);

bootstrap.childOption(ChannelOption.ALLOCATOR, PooledByteBufAllocator.DEFAULT);

ChannelOption.RCVBUF\_ALLOCATOR （一般用于option->boss）

Netty参数，用于Channel分配接受Buffer的分配器，默认值为AdaptiveRecvByteBufAllocator.DEFAULT，是一个自适应的接受缓冲区分配器，能根据接受到的数据自动调节大小。可选值为FixedRecvByteBufAllocator，固定大小的接受缓冲区分配器。

ChannelOption.TCP\_NODELAY (一般用于childOption)

TCP\_NODELAY 对应于socket选项中的TCP\_NODELAY，该参数的使用和Nagle算法有关，Nagle算法是将小的数据包组装为更大的帧进行发送，而不会来一个数据包发送一次，目的是为了提高每次发送的效率，因此在数据包没有组成足够大的帧时，就会延迟该数据包的发送，虽然提高了网络负载却造成了延时，TCP\_NODELAY参数设置为true，就可以禁用Nagle算法，即使用小数据包即时传输。

或者

TCP\_NODELAY就是用于启用或关闭Nagle算法。如果要求高实时性，有数据发送时就马上发送，就将该选项设置为true关闭Nagle算法；如果要减少发送次数减少网络交互，就设置为false等累积一定大小后再发送。默认为false。

ChannelOption.SO\_SNDBUF 和ChannelOption.SO\_RCVBUF (一般用于childOption)

SO\_SNDBUF 和 SO\_RCVBUF对应socket中的SO\_SNDBUF和SO\_RCVBUF参数，即设置发送缓冲区和接收缓冲区的大小，发送缓冲区用于保存发送数据，直到发送成功，接收缓冲区用于保存网络协议站内收到的数据，直到程序读取成功。

或者

SO\_RCVBUF参数，TCP数据接收缓冲区大小。该缓冲区即TCP接收滑动窗口，linux操作系统可使用命令：cat /proc/sys/net/ipv4/tcp\_rmem查询其大小。一般情况下，该值可由用户在任意时刻设置，但当设置值超过64KB时，需要在连接到远端之前设置。

SO\_SNDBUF参数，TCP数据发送缓冲区大小。该缓冲区即TCP发送滑动窗口，linux操作系统可使用命令：cat /proc/sys/net/ipv4/tcp\_smem查询其大小。

ChannelOption.CONNECT\_TIMEOUT\_MILLIS： (一般用于Bootstrap或者childOption)

Netty参数，连接超时毫秒数，默认值30000毫秒即30秒。

ChannelOption.SO\_LINGER (一般用于childOption)

Socket参数，关闭Socket的延迟时间，默认值为-1，表示禁用该功能。-1表示socket.close()方法立即返回，但OS底层会将发送缓冲区全部发送到对端。0表示socket.close()方法立即返回，OS放弃发送缓冲区的数据直接向对端发送RST包，对端收到复位错误。非0整数值表示调用socket.close()方法的线程被阻塞直到延迟时间到或发送缓冲区中的数据发送完毕，若超时，则对端会收到复位错误。

ChannelOption.SO\_KEEPALIVE

  Socket参数，连接保活，默认值为False。启用该功能时，TCP会主动探测空闲连接的有效性。可以将此功能视为TCP的心跳机制，需要注意的是：默认的心跳间隔是7200s即2小时。Netty默认关闭该功能。

ChannelOption.WRITE\_BUFFER\_HIGH\_WATER\_MARK (一般用于childOption)

  Netty参数，写高水位标记，默认值64KB。如果Netty的写缓冲区中的字节超过该值，Channel的isWritable()返回False。

ChannelOption.WRITE\_BUFFER\_LOW\_WATER\_MARK (一般用于childOption)

  Netty参数，写低水位标记，默认值32KB。当Netty的写缓冲区中的字节超过高水位之后若下降到低水位，则Channel的isWritable()返回True。写高低水位标记使用户可以控制写入数据速度，从而实现流量控制。推荐做法是：每次调用channl.write(msg)方法首先调用channel.isWritable()判断是否可写。

ChannelOption.AUTO\_READ (一般用于childOption)

  Netty参数，自动读取，默认值为True。Netty只在必要的时候才设置关心相应的I/O事件。对于读操作，需要调用channel.read()设置关心的I/O事件为OP\_READ，这样若有数据到达才能读取以供用户处理。该值为True时，每次读操作完毕后会自动调用channel.read()，从而有数据到达便能读取；否则，需要用户手动调用channel.read()。需要注意的是：当调用config.setAutoRead(boolean)方法时，如果状态由false变为true，将会调用channel.read()方法读取数据；由true变为false，将调用config.autoReadCleared()方法终止数据读取。

ChannelOption.MAX\_MESSAGES\_PER\_READ

  Netty参数，一次Loop读取的最大消息数，对于ServerChannel或者NioByteChannel，默认值为16，其他Channel默认值为1。默认值这样设置，是因为：ServerChannel需要接受足够多的连接，保证大吞吐量，NioByteChannel可以减少不必要的系统调用select。

ChannelOption.WRITE\_SPIN\_COUNT

  Netty参数，一个Loop写操作执行的最大次数，默认值为16。也就是说，对于大数据量的写操作至多进行16次，如果16次仍没有全部写完数据，此时会提交一个新的写任务给EventLoop，任务将在下次调度继续执行。这样，其他的写请求才能被响应不会因为单个大数据量写请求而耽误。

ChannelOption.MESSAGE\_SIZE\_ESTIMATOR

  Netty参数，消息大小估算器，默认为DefaultMessageSizeEstimator.DEFAULT。估算ByteBuf、ByteBufHolder和FileRegion的大小，其中ByteBuf和ByteBufHolder为实际大小，FileRegion估算值为0。该值估算的字节数在计算水位时使用，FileRegion为0可知FileRegion不影响高低水位

ChannelOption.SINGLE\_EVENTEXECUTOR\_PER\_GROUP

  Netty参数，单线程执行ChannelPipeline中的事件，默认值为True。该值控制执行ChannelPipeline中执行ChannelHandler的线程。如果为True，整个pipeline由一个线程执行，这样不需要进行线程切换以及线程同步，是Netty4的推荐做法；如果为False，ChannelHandler中的处理过程会由Group中的不同线程执行。

————————————————

版权声明：本文为CSDN博主「zhongzunfa」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/zhongzunfa/article/details/94590670