

《Netty权威指南（第2版）》

图书基本信息

书名：《Netty权威指南（第2版）》

13位ISBN编号：9787121258013

出版时间：2015-4-20

作者：李林锋

页数：572

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu000.com

《Netty权威指南（第2版）》

内容概要

《Netty 权威指南（第2版）》是异步非阻塞通信领域的经典之作，基于最新版本的Netty 5.0编写，是国内首本深入介绍Netty原理和架构的书籍，也是作者多年实战经验的总结和浓缩。内容不仅包含Java NIO入门知识、Netty的基础功能开发指导、编解码框架定制等，还包括私有协议栈定制和开发、Netty核心类库源码分析，以及Netty的架构剖析。

《Netty 权威指南（第2版）》适合架构师、设计师、软件开发工程师、测试人员以及其他对Java NIO框架、Netty感兴趣的相关人士阅读，通过《Netty 权威指南（第2版）》的学习，读者不仅能够掌握Netty基础功能的使用和开发，更能够掌握Netty核心类库的原理和使用约束，从而在实际工作中更好地使用Netty。

《Netty权威指南（第2版）》

作者简介

李林锋：Netty中国推广者，现华为技术有限公司平台中间件架构与设计部设计师，公司总裁技术创新奖获得者。长期从事高性能通信软件的架构设计和开发工作，有多年在NIO领域的设计、开发和运维经验，精通NIO编程和Netty、Mina等主流NIO框架。目前负责华为软件公司下一代SOA中间件和PaaS平台的架构设计工作。

书籍目录

基础篇走进Java NIO

第1章Java的I/O演进之路 2

- 1.1 I/O基础入门.....3
 - 1.1.1 Linux网络I/O模型简介.....3
 - 1.1.2 I/O多路复用技术.....6
- 1.2 Java的I/O演进.....8
- 1.3 总结..... 10

第2章NIO入门..... 11

- 2.1 传统的BIO编程..... 11
 - 2.1.1 BIO通信模型图..... 12
 - 2.1.2 同步阻塞式I/O创建的TimeServer源码分析..... 13
 - 2.1.3 同步阻塞式I/O创建的TimeClient源码分析..... 16
- 2.2 伪异步I/O编程..... 18
 - 2.2.1 伪异步I/O模型图..... 19
 - 2.2.2 伪异步I/O创建的TimeServer源码分析..... 19
 - 2.2.3 伪异步I/O弊端分析..... 21
- 2.3 NIO编程..... 24
 - 2.3.1 NIO类库简介.. 24
 - 2.3.2 NIO服务端序列图..... 28
 - 2.3.3 NIO创建的TimeServer源码分析..... 30
 - 2.3.4 NIO客户端序列图..... 36
 - 2.3.5 NIO创建的TimeClient源码分析..... 39
- 2.4 AIO编程..... 45
 - 2.4.1 AIO创建的TimeServer源码分析..... 46
 - 2.4.2 AIO创建的TimeClient源码分析..... 51
 - 2.4.3 AIO版本时间服务器运行结果..... 56
- 2.5 4种I/O的对比..... 58
 - 2.5.1 概念澄清..... 58
 - 2.5.2 不同I/O模型对比..... 59
- 2.6 选择Netty的理由..... 60
 - 2.6.1 不选择Java原生NIO编程的原因..... 61
 - 2.6.2 为什么选择Netty..... 62
- 2.7 总结..... 63

入门篇 Netty NIO 开发指南

第3章Netty入门应用..... 66

- 3.1 Netty开发环境的搭建..... 66
 - 3.1.1 下载Netty的软件包..... 67
 - 3.1.2 搭建Netty应用工程..... 67
- 3.2 Netty服务端开发..... 68
- 3.3 Netty客户端开发..... 73
- 3.4 运行和调试..... 76
 - 3.4.1 服务端和客户端的运行..... 76
 - 3.4.2 打包和部署..... 77
- 3.5 总结..... 77

第4章TCP粘包/拆包问题的解决之道..... 79

- 4.1 TCP粘包/拆包..... 79
 - 4.1.1 TCP粘包/拆包问题说明..... 80

4.1.2 TCP 粘包/拆包发生的原因.....	80
4.1.3 粘包问题的解决策略.....	81
4.2 未考虑TCP 粘包导致功能异常案例.....	82
4.2.1 TimeServer 的改造.....	82
4.2.2 TimeClient 的改造.....	83
4.2.3 运行结果.....	84
4.3 利用LineBasedFrameDecoder 解决TCP 粘包问题.....	85
4.3.1 支持TCP 粘包的TimeServer	86
4.3.2 支持TCP 粘包的TimeClient.....	88
4.3.3 运行支持TCP 粘包的时间服务器程序.....	90
4.3.4 LineBasedFrameDecoder 和StringDecoder 的原理分析.....	91
4.4 总结.....	92
第5章分隔符和定长解码器的应用.....	93
5.1 DelimiterBasedFrameDecoder 应用开发.....	94
5.1.1 DelimiterBasedFrameDecoder 服务端开发....	94
5.1.2 DelimiterBasedFrameDecoder 客户端开发....	97
5.1.3 运行DelimiterBasedFrameDecoder 服务端和客户端.....	99
5.2 FixedLengthFrameDecoder 应用开发.....	101
5.2.1 FixedLengthFrameDecoder 服务端开发.....	101
5.2.2 利用telnet 命令行测试EchoServer 服务端.....	103
5.3 总结.....	104
中级篇 Netty 编解码开发指南	
第6章编解码技术.....	106
6.1 Java 序列化的缺点	107
6.1.1 无法跨语言....	107
6.1.2 序列化后的码流太大.....	107
6.1.3 序列化性能太低.....	110
6.2 业界主流的编解码框架.....	113
6.2.1 Google 的Protobuf 介绍....	113
6.2.2 Facebook 的Thrift 介绍....	115
6.2.3 JBoss Marshalling 介绍.....	116
6.3 总结.....	117
第7章MessagePack 编解码.....	118
7.1 MessagePack 介绍... ..	118
7.1.1 MessagePack 多语言支持..	119
7.1.2 MessagePack Java API 介绍.....	119
7.1.3 MessagePack 开发包下载..	120
7.2 MessagePack 编码器和解码器开发.....	120
7.2.1 MessagePack 编码器开发.....	120
7.2.2 MessagePack 解码器开发	121
7.2.3 功能测试.....	121
7.3 粘包/半包支持.....	124
7.4 总结.....	127
第8章Google Protobuf 编解码.....	128
8.1 Protobuf 的入门.....	129
8.1.1 Protobuf 开发环境搭建.....	129
8.1.2 Protobuf 编解码开发.....	131
8.1.3 运行Protobuf 例程.....	133
8.2 Netty 的Protobuf 服务端开发..	133

8.2.1 Protobuf 版本的图书订购服务端开发.....	134
8.2.2 Protobuf 版本的图书订购客户端开发.....	136
8.2.3 Protobuf 版本的图书订购程序功能测试....	139
8.3 Protobuf 的使用注意事项.....	140
8.4 总结.....	142
第9章JBoss Marshalling 编解码.....	143
9.1 Marshalling 开发环境准备.....	143
9.2 Netty 的Marshalling 服务端开发.....	144
9.3 Netty 的Marshalling 客户端开发.....	147
9.4 运行Marshalling 客户端和服务端例程.....	149
9.5 总结.....	150
高级篇 Netty 多协议开发和应用	
第10章HTTP 协议开发应用.....	154
10.1 HTTP 协议介绍.....	155
10.1.1 HTTP 协议的URL	155
10.1.2 HTTP 请求消息 (HttpRequest)	155
10.1.3 HTTP 响应消息 (HttpResponse)	158
10.2 Netty HTTP 服务端入门开发.....	159
10.2.1 HTTP 服务端例程场景描述.....	160
10.2.2 HTTP 服务端开发.....	160
10.2.3 Netty HTTP 文件服务器例程运行结果....	166
10.3 Netty HTTP+XML 协议栈开发.....	170
10.3.1 开发场景介绍.....	171
10.3.2 HTTP+XML 协议栈设计.....	174
10.3.3 高效的XML 绑定框架JiBx	175
10.3.4 HTTP+XML 编解码框架开发.....	183
10.3.5 HTTP+XML 协议栈测试.....	199
10.3.6 小结.....	201
10.4 总结.....	202
第11章WebSocket 协议开发.....	203
11.1 HTTP 协议的弊端.....	204
11.2 WebSocket 入门.....	204
11.2.1 WebSocket 背景.....	205
11.2.2 WebSocket 连接建立.....	206
11.2.3 WebSocket 生命周期.....	207
11.2.4 WebSocket 连接关闭.....	208
11.3 Netty WebSocket 协议开发.....	209
11.3.1 WebSocket 服务端功能介绍.....	209
11.3.2 WebSocket 服务端开发....	210
11.3.3 运行WebSocket 服务端...	218
11.4 总结.....	219
第12章私有协议栈开发....	221
12.1 私有协议介绍.....	221
12.2 Netty 协议栈功能设计.....	223
12.2.1 网络拓扑图..	223
12.2.2 协议栈功能描述.....	224
12.2.3 通信模型.....	224
12.2.4 消息定义.....	225
12.2.5 Netty 协议支持的字段类型.....	226

12.2.6	Netty 协议的编解码规范..	227
12.2.7	链路的建立..	229
12.2.8	链路的关闭..	230
12.2.9	可靠性设计..	230
12.2.10	安全性设计	232
12.2.11	可扩展性设计.....	232
12.3	Netty 协议栈开发..	233
12.3.1	数据结构定义.....	233
12.3.2	消息编解码..	237
12.3.3	握手和安全认证.....	241
12.3.4	心跳检测机制.....	245
12.3.5	断连重连.....	248
12.3.6	客户端代码..	249
12.3.7	服务端代码..	251
12.4	运行协议栈.....	252
12.4.1	正常场景.....	252
12.4.2	异常场景：服务端宕机重启.....	253
12.4.3	异常场景：客户端宕机重启.....	256
12.5	总结.....	256
第13章	服务端创建.....	258
13.1	原生NIO 类库的复杂性.....	259
13.2	Netty 服务端创建源码分析....	259
13.2.1	Netty 服务端创建时序图.	260
13.2.2	Netty 服务端创建源码分析.....	263
13.3	客户端接入源码分析.....	272
13.4	总结.....	275
第14章	客户端创建.....	276
14.1	Netty 客户端创建流程分析....	276
14.2.1	Netty 客户端创建时序图.	276
14.2.2	Netty 客户端创建流程分析.....	277
14.2	Netty 客户端创建源码分析....	278
14.2.1	客户端连接辅助类Bootstrap.....	278
14.2.2	客户端连接操作.....	281
14.2.3	异步连接结果通知.....	283
14.2.4	客户端连接超时机制.....	284
14.3	总结.....	286
源码分析篇 Netty 功能介绍和源码分析		
第15章	ByteBuf 和相关辅助类.....	288
15.1	ByteBuf 功能说明.	288
15.1.1	ByteBuf 的工作原理.....	289
15.1.2	ByteBuf 的功能介绍.....	294
15.2	ByteBuf 源码分析.	308
15.2.1	ByteBuf 的主要类继承关系.....	309
15.2.2	AbstractByteBuf 源码分析.....	310
15.2.3	AbstractReferenceCountedByteBuf 源码分析.....	319
15.2.4	UnpooledHeapByteBuf 源码分析.....	321
15.2.5	PooledByteBuf 内存池原理分析.....	326
15.2.6	PooledDirectByteBuf 源码分析.....	329
15.3	ByteBuf 相关的辅助类功能介绍.....	332

15.3.1 ByteBufHolder.....	332
15.3.2 ByteBufAllocator	333
15.3.3 CompositeByteBuf	334
15.3.4 ByteBufUtil ..	336
15.4 总结.....	337
第16章Channel和Unsafe	338
16.1 Channel功能说明.	338
16.1.1 Channel的工作原理.....	339
16.1.2 Channel的功能介绍.....	340
16.2 Channel源码分析.	343
16.2.1 Channel的主要继承关系类图.....	343
16.2.2 AbstractChannel源码分析.....	344
16.2.3 AbstractNioChannel源码分析.....	347
16.2.4 AbstractNioByteChannel源码分析.....	350
16.2.5 AbstractNioMessageChannel源码分析.....	353
16.2.6 AbstractNioMessageServerChannel源码分析.....	354
16.2.7 NioServerSocketChannel源码分析.....	355
16.2.8 NioSocketChannel源码分析.....	358
16.3 Unsafe功能说明...	364
16.4 Unsafe源码分析...	365
16.4.1 Unsafe继承关系类图.....	365
16.4.2 AbstractUnsafe源码分析.	366
16.4.3 AbstractNioUnsafe源码分析.....	375
16.4.4 NioByteUnsafe源码分析.	379
16.5 总结.....	387
第17章ChannelPipeline和ChannelHandler.....	388
17.1 ChannelPipeline功能说明.....	389
17.1.1 ChannelPipeline的事件处理.....	389
17.1.2 自定义拦截器.....	391
17.1.3 构建pipeline	392
17.1.4 ChannelPipeline的主要特性.....	393
17.2 ChannelPipeline源码分析.....	393
17.2.1 ChannelPipeline的类继承关系图.....	393
17.2.2 ChannelPipeline对ChannelHandler的管理.....	393
17.2.3 ChannelPipeline的inbound事件.....	396
17.2.4 ChannelPipeline的outbound事件.....	397
17.3 ChannelHandler功能说明.....	398
17.3.1 ChannelHandlerAdapter功能说明.....	399
17.3.2 ByteToMessageDecoder功能说明.....	399
17.3.3 MessageToMessageDecoder功能说明.....	400
17.3.4 LengthFieldBasedFrameDecoder功能说明.....	400
17.3.5 MessageToByteEncoder功能说明.....	404
17.3.6 MessageToMessageEncoder功能说明.....	404
17.3.7 LengthFieldPrepender功能说明.....	405
17.4 ChannelHandler源码分析.....	406
17.4.1 ChannelHandler的类继承关系图.....	406
17.4.2 ByteToMessageDecoder源码分析.....	407
17.4.3 MessageToMessageDecoder源码分析.....	410
17.4.4 LengthFieldBasedFrameDecoder源码分析.....	411

17.4.5 MessageToByteEncoder 源码分析.....	415
17.4.6 MessageToMessageEncoder 源码分析.....	416
17.4.7 LengthFieldPrepender 源码分析.....	417
17.5 总结.....	418
第18章EventLoop 和EventLoopGroup.....	419
18.1 Netty 的线程模型..	419
18.1.1 Reactor 单线程模型.....	420
18.1.2 Reactor 多线程模型.....	421
18.1.3 主从Reactor 多线程模型	422
18.1.4 Netty 的线程模型.....	423
18.1.5 最佳实践.....	424
18.2 NioEventLoop 源码分析.....	425
18.2.1 NioEventLoop 设计原理..	425
18.2.2 NioEventLoop 继承关系类图.....	426
18.2.3 NioEventLoop.....	427
18.3 总结.....	436
第19章Future 和Promise	438
19.1 Future 功能.....	438
19.2 ChannelFuture 源码分析.....	443
19.3 Promise 功能介绍.	445
19.4 Promise 源码分析.	447
19.4.1 Promise 继承关系图.....	447
19.4.2 DefaultPromise	447
19.5 总结.....	449
架构和行业应用篇 Netty 高级特性	
第20章Netty 架构剖析.....	452
20.1 Netty 逻辑架构.....	452
20.1.1 Reactor 通信调度层.....	453
20.1.2 职责链ChannelPipeline ...	453
20.1.3 业务逻辑编排层 (Service ChannelHandler)	454
20.2 关键架构质量属性.....	454
20.2.1 高性能.....	454
20.2.2 可靠性.....	457
20.2.3 可定制性.....	460
20.2.4 可扩展性.....	460
20.3 总结.....	460
第21章Java 多线程编程在Netty 中的应用.....	461
21.1 Java 内存模型与多线程编程..	461
21.1.1 硬件的发展和多任务处理.....	461
21.1.2 Java 内存模型.....	462
21.2 Netty 的并发编程实践.....	464
21.2.1 对共享的可变数据进行正确的同步.....	464
21.2.2 正确使用锁..	465
21.2.3 volatile 的正确使用.....	467
21.2.4 CAS 指令和原子类.....	470
21.2.5 线程安全类的应用.....	472
21.2.6 读写锁的应用.....	476
21.2.7 线程安全性文档说明.....	477
21.2.8 不要依赖线程优先级.....	478

21.3 总结.....	479
第22章高性能之道.....	480
22.1 RPC 调用性能模型分析.....	480
22.1.1 传统RPC 调用性能差的三宗罪.....	480
22.1.2 I/O 通信性能三原则.....	481
22.2 Netty 高性能之道..	482
22.2.1 异步非阻塞通信.....	482
22.2.2 高效的Reactor 线程模型	482
22.2.3 无锁化的串行设计.....	485
22.2.4 高效的并发编程.....	486
22.2.5 高性能的序列化框架.....	486
22.2.6 零拷贝.....	487
22.2.7 内存池.....	491
22.2.8 灵活的TCP 参数配置能力.....	494
22.3 主流NIO 框架性能对比.....	495
22.4 总结.....	497
第23章可靠性.....	498
23.1 可靠性需求.....	498
23.1.1 宕机的代价..	498
23.1.2 Netty 可靠性需求.....	499
23.2 Netty 高可靠性设计	500
23.2.1 网络通信类故障.....	500
23.2.2 链路的有效性检测.....	507
23.2.3 Reactor 线程的保护	510
23.2.4 内存保护.....	513
23.2.5 流量整形.....	516
23.2.6 优雅停机接口.....	519
23.3 优化建议.....	520
23.3.1 发送队列容量上限控制... ..	520
23.3.2 回推发送失败的消息.....	521
23.4 总结.....	521
第24章安全性.....	522
24.1 严峻的安全形势....	522
24.1.1 OpenSSL Heart bleed 漏洞.....	522
24.1.2 安全漏洞的代价.....	523
24.1.3 Netty 面临的安全风险.....	523
24.2 Netty SSL 安全特性.....	525
24.2.1 SSL 单向认证.....	525
24.2.2 SSL 双向认证.....	532
24.2.3 第三方CA 认证.....	536
24.3 Netty SSL 源码分析.....	538
24.3.1 客户端.....	538
24.3.2 服务端.....	541
24.3.3 消息读取.....	544
24.3.4 消息发送.....	545
24.4 Netty 扩展的安全特性.....	546
24.4.1 IP 地址黑名单机制.....	547
24.4.2 接入认证.....	548
24.4 总结.....	550

第25章 Netty 未来展望.....	551
25.1 应用范围.....	551
25.2 技术演进.....	552
25.3 社区活跃度.....	552
25.4 Road Map	552
25.5 总结.....	553
附录A Netty 参数配置表....	554

精彩短评

- 1、完全不适合对于netty陌生的新手读，先读完netty in action在回头来看这本书，有一些地方可以查漏补缺。缺点就不说了，豆瓣评论中说道的各种缺点仍然适用于第二本。属于在国内科技书籍当中出版质量也比较低的一本书。但是还是有参考意义。。给个两分意思一下。
- 2、四个字：华而不实，浪费钱和时间
- 3、netty的书国内比较少，有这么一本入门的不错了，叫权威指南书名太大了，作者用心，但是内容还不够达到这个级别。
- 4、是的作者抄代码凑字数太多。阅读顺序建议按章节从后往前读，后面的源码分析和架构的图解还比较好，前面例子随便瞄一下就好。
- 5、有助于了解和熟悉Netty
- 6、作为入门还不错，整理的部分内容清晰易懂。不过标题起的太大了。。
- 7、这本书可以不用买了，抛开书的质量不谈，这版Netty5本身已经被放弃了
<https://github.com/netty/netty/issues/4466>
- 8、东西越写越少，前面没啥深度的东西占了大量篇幅，后面需要详细分析的东西又一笔带过，浮于表面，没有什么深入的地方，很多东西都是片段，没有一个主线能串起来的，浮躁
- 9、不多的入门书，要想了解netty还得看源码
- 10、内容没说的那么差，代码是贴的有点多了
- 11、书很一般，里面讲的东西其实再官网文档里都找得到。
前面几章都在介绍Java bio/nio/aio（百度一下一大把），中间介绍了下netty基础（官网文档都能找到的东西），然后就是各种代码例子（其实去下载netty源码包，example下面都有这些类似的例子），最后几章在介绍netty的源码（还没看，不知道怎么样）。
再说说里面的例子，就更烂了，跟大学写作业一样，随便写写的，没有一点参考价值。
- 12、nio的代表，越来越多的cs采用netty交互，值得一读。
- 13、源码部分还可以，提供了思路，不过其他的部分，只是浅浅而谈，介绍api，而且贴代码和console太多，影响排版
- 14、看的1版电子版，买了纸质的2版。。或许换个名字，评分会好些？
- 15、回过头又重新看了一遍netty权威指南，书确实是只有一般性，主要是这方面的书籍也不多，书中的一些类机构图不清晰。线程模型讲解的非常混乱。可以作为入门书籍，单靠这本书很难理解，结合netty in action这两本书互补可以加深理解，再去看源码。
- 16、读过即便，市面上几乎是唯一一本全面介绍Netty功能的书。作者还分析了一部分Netty源码，可见功力扎实。
- 17、此书虽然评价不是很高，但是毋庸置疑的是，它是一本国内讲netty比较全面的技术书，可以说，对入门帮助非常大。
- 18、实在没别的选择就读读吧
- 19、方法浅显易懂多是API的一些介绍作为入门还是不错的
- 20、很好的工具书，netty网络上的资源比较少，这本书上有丰富的例子，入门很不错。
- 21、只是大体翻了一下，没有细读。整体上来讲我觉得还是不错的，不像一些评论那么不友好。问题是章节安排详略不当，比如编解码部分，必要性不大，原理讲清楚了一个案例就可以了。还有源码分析部分，近三分之一篇幅，书太厚了。我喜欢In Action系列的风格，要点清楚，不扣细节，注重实战。细节的东西有需要的读者自己会去补充。作为技术同行，深知做技术的辛苦，能系统地写一本技术书，必定踩过很多坑，熬过很多不眠夜。支持原创。
- 22、很多内容需要仔细深入阅读和理解才能有效果

《Netty权威指南（第2版）》

精彩书评

- 1、ByteBuffer.allocate(1024)是分配了1MB缓冲区，这是怎么算的？难道不是1KB吗？这么明显的错误也能写出来~~~~~
- 2、书很一般，里面讲的东西其实再官网文档里都找得到。前面几章都在介绍Java bio/nio/aio（百度一下一大把），中间介绍了下netty基础（官网文档都能找到的东西），然后就是各种代码例子（其实去下载netty源码包，example下面都有这些类似的例子），最后几章在介绍netty的远吗（还没看，不知道怎么样）。再说说里面的例子，就更烂了，跟大学写作业一样，随便写写的，没有一点参考价值
- 3、”《Netty 权威指南（第2版）》是异步非阻塞通信领域的经典之作“，这个Title是谁给你封的？大段大段的代码。一个功能的代码，本来就占两页，说是为了改进效率在某个函数就增加了一行代码，就又重新打印了两页代码，并且控制台的输出结果愣是占了三四页。你才是大神。恩，用了十几页口水说的话，根本就没营养好伐？幸好我没买你的书，第一次差评，因为这本书已经影响到国产技术书的信誉了。
- 4、虽然书里有一些api罗列、源码分析一些不讨喜的部分，但是整体还是很好的，适合于对Netty有初步的使用能力、希望系统化了解学习的朋友们。也许给一星的都是大牛吧，这种级别对他们太low了，不过大牛直接看看英文文档，扫扫源码不就好了吗，不需要再看国人写的书了吧。

章节试读

1、《Netty权威指南（第2版）》的笔记-第6页

目前支持I/O多路复用的系统调用有select、pselect、poll、epoll，在Linux网络编程过程中，很长一段时间都使用select做轮询和网络事件通知，然而select的一些固有缺陷导致了它的应用受到了很大的限制，最终Linux不得不在新的内核版本中寻找select的替代方案，最终选择了epoll。

2、《Netty权威指南（第2版）》的笔记-第115页

Facebook的Thrift介绍

对于当时的Facebook来说，创造Thrift是为了解决Facebook各系统间大数据量的传输通信以及系统之间语言环境不同需要跨平台的特性，因此Thrift可以支持多种程序语言，如C++、C#、Cocoa、Erlang、Haskell、Java、Ocaml、Perl、PHP、Python、Ruby和Smalltalk。

3、《Netty权威指南（第2版）》的笔记-第27页

探索多路复用器Selector，它是Java NIO编程的基础，熟练地掌握Selector对于NIO编程至关重要。多路复用器提供选择已就绪的任务的能力。简单来讲，Selector会不断地轮询注册在其上的Channel，如果某个Channel上面发生读或者写事件，这个Channel就处于就绪状态，会被Selector轮询出来，然后通过SelectionKey可以获取就绪Channel的集合，进行后续的I/O操作。

4、《Netty权威指南（第2版）》的笔记-第110页

我们评判一个编解码框架的优劣时，往往会考虑以下几个因素：

- 1)是否支持跨语言，支持语言种类是否丰富。
- 2) 编码后的码流大小。
- 3) 编解码的性能。
- 4) 类库是否小巧，API使用是否方便。
- 5) 使用者需要手工开发的工作量和难度。

5、《Netty权威指南（第2版）》的笔记-第45页

使用NIO编程的优点总结如下：

- 1、客户端发起的连接操作是异步的，可以通过在多路复用器注册OP_CONNECT等待后续结果，不需要像之前的客户端那样被同步阻塞。
- 2、SocketChannel的读写操作都是异步的。如果没有可读写的的数据它不会同步等待，直接返回，这样I/O通信线程就可以处理其他的链路，不需要同步等待这个链路可用。
- 3、线程模型的优化：由于JDK的Selector在Linux等主流操作系统上通过epoll实现，它没有连接句柄数的限制（只受限于操作系统的最大句柄数或者对单个进程的句柄限制），这意味着一个Selector线程可以同时处理成千上万客户端连接，而且性能不会随着客户端的增加而线性下降。因此，它非常适合做高性能，高负载的网络服务器。

6、《Netty权威指南（第2版）》的笔记-第79页

4.1 TCP粘包/拆包

TCP是个“流”协议，所谓流，就是没有界限的一串数据。大家可以想想河里的流水，它们是连成一片的，其间并没有分界线。TCP底层并不了解上层业务数据的具体含义，它会根据TCP缓冲区的实际情况进行包的划分，所以在业务上认为，一个完整的包可能会被TCP拆分成多个包进行发送，也有可

能把多个小的包封装成一个大的数据包发送，这就是所谓的TCP粘包和拆包问题。

7、《Netty权威指南（第2版）》的笔记-第7页

熟悉TCP编程的读者可能都知道，无论是服务端还是客户端，当我们读取或者发送消息的时候，都需要考虑TCP底层的粘包/拆包机制。

8、《Netty权威指南（第2版）》的笔记-第4页

- ## 阻塞IO模型
- ## 非阻塞IO模型
- ## 复用IO模型
- ## 信号驱动IO模型
- ## 异步IO

9、《Netty权威指南（第2版）》的笔记-第114页

Protobuf另一个比较吸引人的地方就是它的数据描述文件和代码生成机制，利用数据描述文件对数据结构进行说明的优点如下：

文本化的数据结构描述语言，可以实现语言和平台无关，特别适合异构系统间的集成。

通过标识字段的顺序，可以实现协议的前向兼容。

自动代码生成，不需要手工编写同样数据结构的C++和Java版本。

方便后续的管理和维护。相比于代码，结构化的文档更容易管理和维护。

10、《Netty权威指南（第2版）》的笔记-第107页

6.1 Java序列化的缺点

在远程服务调用（RPC）时，很少直接使用Java序列化进行消息的编解码和传输：

1、无法跨语言。

无法跨语言，是Java序列化最致命的问题。对于跨进程的服务调用，服务提供者可能会使用C++或者其他语言开发，当我们需要和异构语言进程交互时，Java序列化就难以胜任了。

由于Java序列化技术是Java语言内部的私有协议，其他语言并不支持，对于用户来说它完全是黑盒。对于Java序列化后的字节数组，别的语言无法进行反序列化，这就严重阻碍了它的应用。

目前几乎所有流行的Java RPC通信框架，都没有使用Java序列化作为编解码框架，原因就是在于它无法跨语言，而这些RPC框架往往需要支持跨语言调用。

11、《Netty权威指南（第2版）》的笔记-第2页

Java的NIO编程并没有流行起来，究其原因如下。

1、大多数高性能服务器，被C和C++语言盘踞，由于它们可以直接使用操作系统的异步I/O能力，所以对JDK的NIO并不关心。

2、移动互联网尚未兴起，基于Java的大规模分布式系统极少，很多中小型应用服务对于异步I/O的诉求不是很强烈。

3、高性能、高可靠性领域，例如银行、证券、电信等，依然以C++为主导，Java充当打杂的角色，NIO暂时没有用武之地。

4、当时主流的J2EE服务器，几乎全都基于同步阻塞I/O构建，例如Servlet、Tomcat等，由于它们应用广泛，如果这些容器不支持NIO，用户很难具备独立构建异步协议栈的能力。

5、异步NIO编程门槛比较高，开发和维护一款基于NIO的协议栈对很多中小型公司来说像是一场噩梦

- 。
- 6、业界NIO框架不成熟，很难商用。
- 7、国内研发界对NIO的陌生和认识不足，没有充分重视。

如果说个人能够改变自己命运的话，对于程序员来说，唯有通过不断地学习和实践，努力提升自己的技能，才有可能找到更好的机会，充分发挥和体现自己的价值。

12、《Netty权威指南（第2版）》的笔记-第2页

根据UNIX网络编程对I/O模型的分类，UNIX提供了5种I/O模型，分别如下。

- 1、阻塞I/O模型：最常用的I/O模型就是阻塞I/O模型，缺省情况下，所有文件操作都是阻塞的。我们以套接字接口为例来讲解此模型：在进程空间中调用recvfrom，其系统调用直到数据包到达且被复制到应用进程的缓冲区或者发生错误时才返回，在此期间一直会等待，进程在从调用recvfrom开始到它返回的整段时间内都是被阻塞的，因此被称为阻塞I/O模型。
- 2、非阻塞I/O模型：recvfrom从应用层到内核的时候，如果该缓冲区没有数据的话，就直接返回一个EWOULDBLOCK错误，一般都对非阻塞I/O模型进行轮询检查这个状态，看内核是不是有数据到来。
- 3、I/O复用模型：Linux提供select/poll，进程通过将一个或多个fd传递给select或poll系统调用，阻塞在select操作上，这样select/poll可以帮我们侦测多个fd是否处于就绪状态。select/poll是顺序扫描fd是否就绪，而且支持的fd数量有限，因此它的使用受到了一些制约。Linux还提供了一个epoll系统调用，epoll使用基于事件驱动方式代替顺序扫描，因此性能更高。当有fd就绪时，立即回调函数rollback。
- 4、信号驱动I/O模型：首先开启套接接口信号驱动I/O功能，并通过系统调用sigaction执行一个信号处理函数（此系统调用立即返回，进程继续工作，它是非阻塞的）。当数据准备就绪时，就为该进程生成一个SIGIO信号，通过信号回调通知应用程序调用recvfrom来读取数据，并通知主循环函数处理数据。
- 5、异步I/O：告知内核启动某个操作，并让内核在整个操作完成后（包括将数据从内核复制到用户自己的缓冲区）通知我们。这种模型与信号驱动模型的主要区别是：信号驱动I/O由内核通知我们何时可以开始一个I/O操作；异步I/O模型由内核通知我们I/O操作何时已经完成。

13、《Netty权威指南（第2版）》的笔记-第1页

异步高性能内部协议栈、异步HTTP、异步SOAP、异步SMPP，所有的协议栈都是异步非阻塞。基于Reactor模型统一调度的长连接和短连接协议栈，无论是性能、可靠性还是可维护性，都可以秒杀传统基于BIO开发的应用服务器和各种协议栈，这种差异本质上是一种代差。

14、《Netty权威指南（第2版）》的笔记-第6页

1. 支持一个进程打开的socket描述符（FD）不受限制（仅限制于操作系统的最大文件句柄数）
1G内存的机器大约支持10万个句柄。

\$ cat /proc/sys/fs/file-max 可查系统支持我句柄数

2. IO效率不会随着FD数目的增加而线性下降。
3. 使用mmap加速内核与用户空间的消息传递。
epoll通过内核和用户空间mmap共享同一块内存来实现。
4. epoll的API更简单。

15、《Netty权威指南（第2版）》的笔记-第116页

JBoss Marshalling

优点如下：

- 1、可拔插的类解析器，提供更加便捷的类加载定制策略，通过一个接口即可实现定制。
- 2、可拔插的对象替换技术，不需要通过继承的方式。
- 3、可拔插的预定义类缓存表，可以减少序列化的字节数组长度，提升常用类型的对象序列化性能。
- 4、无须实现java.io.Serializable接口，即可实现Java序列化。
- 5、通过缓存技术提升对象的序列化性能。

《Netty权威指南（第2版）》

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:www.tushu000.com