Netty 架构解析

2016-7-20

目录

1 Netty介绍

2 Netty架构分析

3 架构质量属性剖析

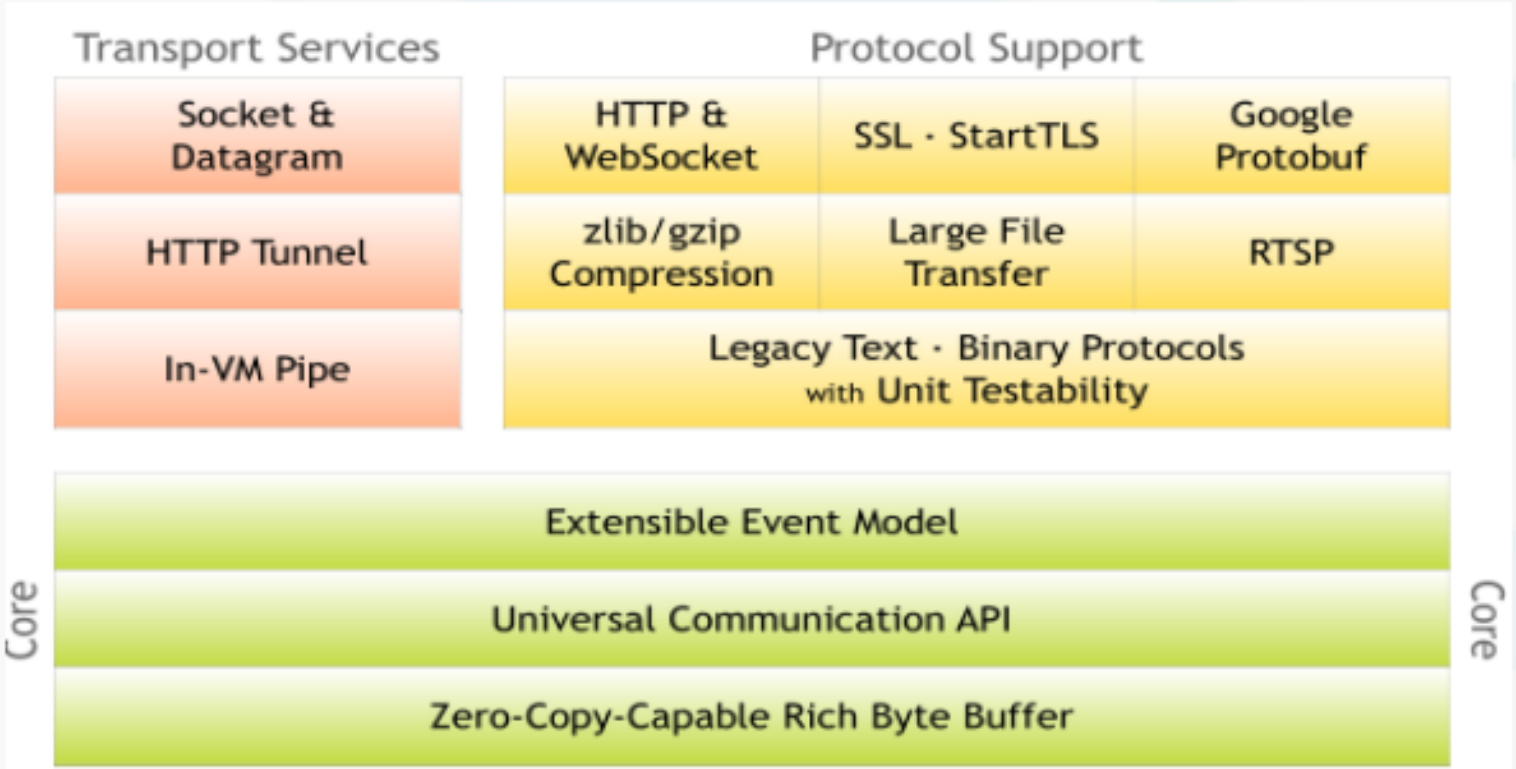
Netty是一个异步、事件驱动的网络应用框架。基于Netty，可以快速的开发和部署高性能、高可用的网络服务端和客户端应用。

为什么选择Netty?
Netty是业界最流行的NIO框架之一，它的健壮性、功能、性能、可定制性和可扩展性在同类框架中都是首屈一指的，它已经得到成百上千的商用项目验证，例如Hadoop的RPC框架avro使用Netty作为底层通信框架；很多其他业界主流的RPC框架，也使用Netty来构建高性能的异步通信能力。

通过对Netty的分析，我们将它的优点总结如下:
1、API使用简单，开发门槛低；
2、功能强大，预置了多种编解码功能，支持多种主流协议；
3、定制能力强，可以通过ChannelHandler对通信框架进行灵活地扩展；
4、性能高，通过与其他业界主流的NIO框架对比，Netty的综合性能最优；
5、成熟、稳定，Netty修复了已经发现的所有JDK NIO BUG，业务开发人员不需要再为NIO的BUG而烦恼；
6、社区活跃，版本迭代周期短，发现的BUG可以被及时修复，同时，更多的新功能会加入；
7、经历了大规模的商业应用考验，质量得到验证。在互联网、大数据、网络游戏、企业应用、电信软件等众多行业得到成功商用，证明了它已经完全能够满足不同行业的商业应用了。

Netty介绍

Netty是一个异步、事件驱动的网络应用框架。基于Netty，可以快速的开发和部署高性能、高可用的网络服务端和客户端应用。



优势：

1、Open API高度封装，使用极简

2、异步事件驱动，高性能设计

3、内置丰富的编解码能力

4、更优雅的扩展性设计

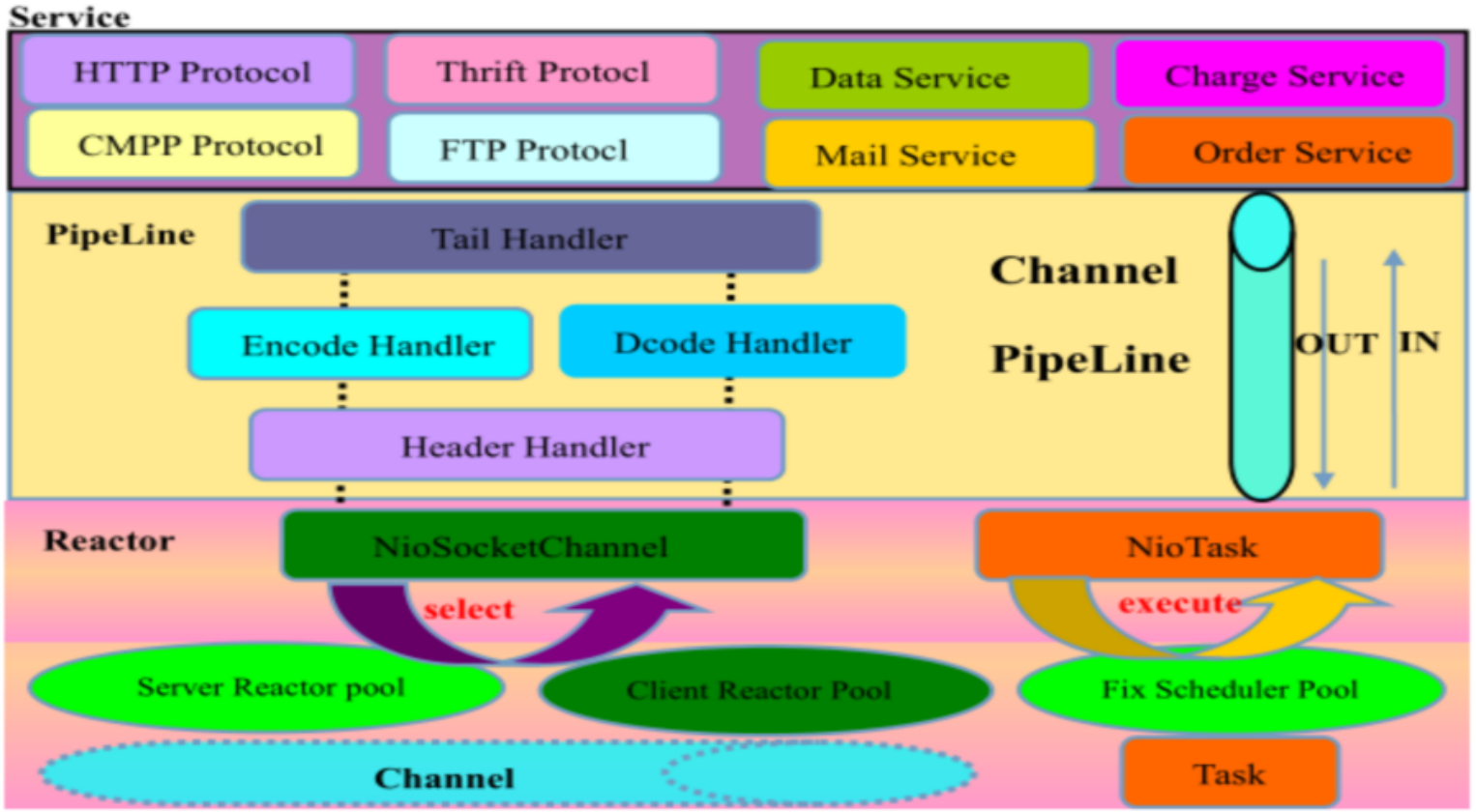
5、完善的可靠性

6、更丰富和详尽的资料

Netty 采用了比较典型的三层网络架构进行设计

第一层：Reactor 通信调度层，它由一系列辅助类完成，包括 Reactor 线程 NioEventLoop 以及其父类、NioSocketChannel/NioServerSocketChannel 以及其父 类、ByteBuffer 以及由其衍生出来的各种 Buffer、Unsafe 以及其衍生出的各种内 部类等。该层的主要职责就是监听网络的读写和连接操作，负责将网络层的数据 读取到内存缓冲区中，然后触发各种网络事件，例如连接创建、连接激活、读事 件、写事件等等，将这些事件触发到 PipeLine 中，由 PipeLine 充当的职责链来 进行后续的处理。

第二层：职责链 PipeLine，它负责事件在职责链中的有序传播，同时负责动态的 编排职责链，职责链可以选择监听和处理自己关心的事件，它可以拦截处理和向 后/向前传播事件，不同的应用的 Handler 节点的功能也不同，通常情况下，往往会开发编解码 Hanlder 用于消息的编解码，它可以将外部的协议消息转换成内部 的 POJO 对象，这样上层业务侧只需要关心处理业务逻辑即可，不需要感知底层 的协议差异和线程模型差异，实现了架构层面的分层隔离；

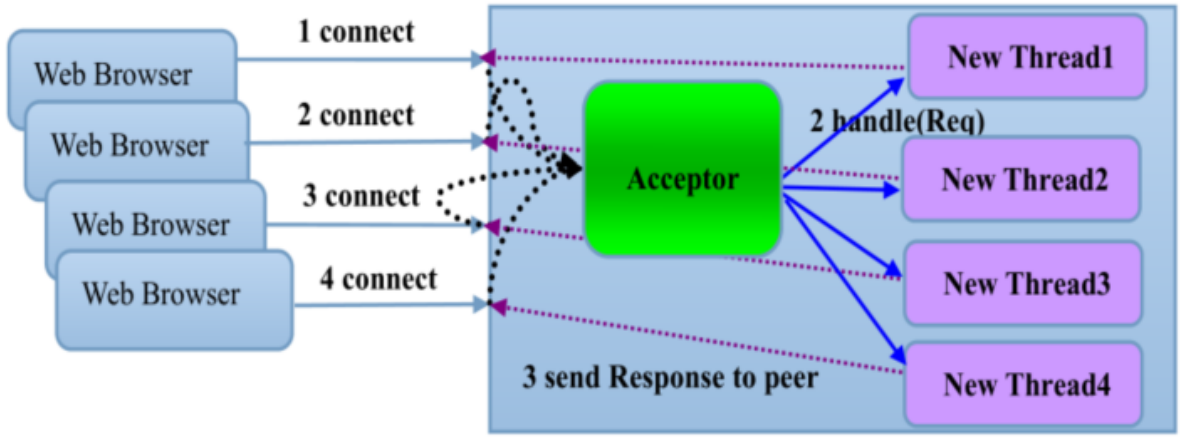


第三层：业务逻辑处理层，可以分为两类：1、纯粹的业务逻辑 处理，例如订单处理2、应用层协议管理，例如HTTP协议、FTP协议等。

Netty的关键架构质量属性

通信性能三要素：
1、I/O模型
2、线程调度模型
3、序列化方式

传统同步阻塞I/O模式：



1、性能问题：一连接一线程模型导致服务端的并发接入数和系统吞吐量受到极大限制；
2、可靠性问题：由于I/O操作采用同步阻塞模式，当网络拥塞或者通信对端处理缓慢会导致I/O线程被挂住，阻塞时间无法预测；
3、可维护性问题：I/O线程数无法有效控制、资源无法有效共享（多线程并发问题），系统可维护性差

几种I/O模型的功能和特性对比：

	同步阻塞I/O（BIO）	伪异步I/O	非阻塞I/O（NIO）	异步I/O（AIO）
客户端个数：I/O线程	1：1	M：N（其中M可以大于N）	M：1（1个I/O线程处理多个客户端连接）	M：0（不需要启动额外的I/O线程，被动回调）
I/O类型（阻塞）	阻塞I/O	阻塞I/O	非阻塞I/O	非阻塞I/O
I/O类型（同步）	同步I/O	同步I/O	同步I/O（I/O多路复用）	异步I/O
API使用难度	简单	简单	非常复杂	复杂
调试难度	简单	简单	复杂	复杂
可靠性	非常差	差	高	高
吞吐量	低	中	高	高

Netty的I/O模型基于非阻塞I/O实现，
底层依赖的是JDK NIO框架的Selector：Selector提供选择已经就绪的任务的能力。简单来讲，Selector会不断地轮询注册在其上的Channel，如果某个Channel上面有新的TCP连接接入、读和写事件，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。
一个多路复用器Selector可以同时轮询多个Channel，由于JDK1.5_update10版本（+）使用了epoll()代替传统的select实现，所以它并没有最大连接句柄1024/2048的限制。这也就意味着只需要一个线程负责Selector的轮询，就可以接入成千上万的客户端，这确实是个非常巨大的技术进步。

使用非阻塞I/O模型之后，Netty解决了传统同步阻塞I/O带来的性能、吞吐量和可靠性问题

线程调度模型 对性能的影响

Netty线程模型设计理念：

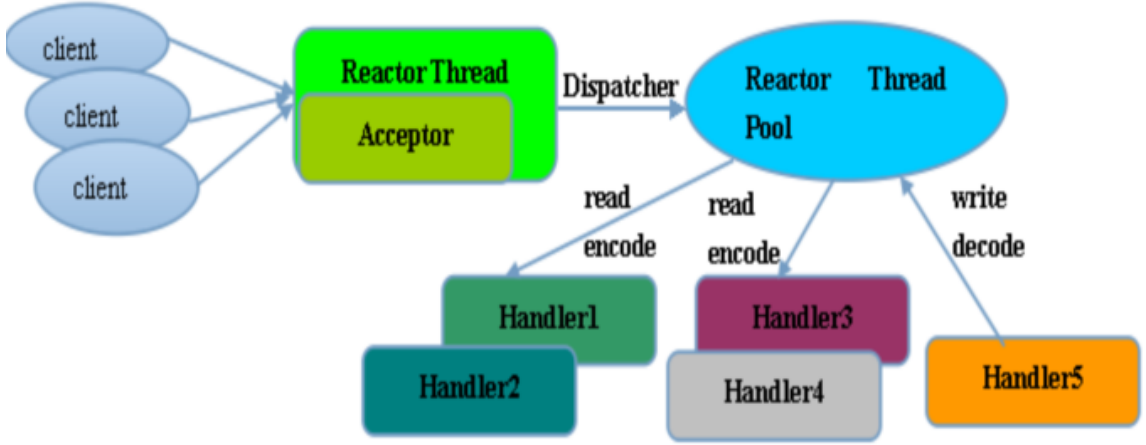
1、Reactor线程模型

2、I/O线程分组，负载均衡

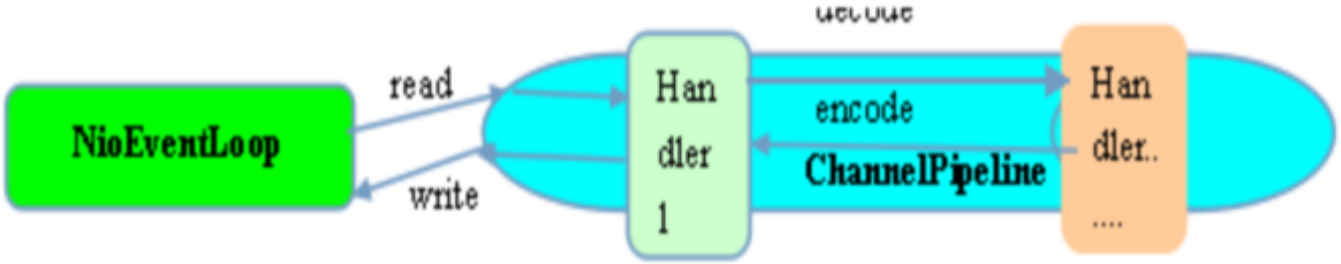
3、无锁化串行设计

4、I/O线程和业务线程分离，各司其职，互不干扰

Reactor线程模型图



无锁化串行设计 模型图：



事实上，Netty的线程模型并非固定不变，通过在启动辅助类中创建不同的EventLoopGroup实例并通过适当的参数配置，就可以支持上述三种Reactor线程模型。

常用的Reactor线程模型有三种，分别如下：

1、Reactor单线程模型：Reactor单线程模型，指的是所有的I/O操作都在同一个NIO线程上面完成。对于一些小容量应用场景，可以使用单线程模型。

2、Reactor多线程模型：Rector多线程模型与单线程模型最大的区别就是有一组NIO线程处理I/O操作。主要用于高并发、大业务量场景。

3、主从Reactor多线程模型：主从Reactor线程模型的特点是服务端用于接收客户端连接的不再是个1个单独的NIO线程，而是一个独立的NIO线程池。利用主从NIO线程模型，可以解决1个服务端监听线程无法有效处理所有客户端连接的性能不足问题。

在大多数场景下，并行多线程处理可以提升系统的并发性能。但是，如果对于共享资源的并发访问处理不当，会带来严重的锁竞争，这最终会导致性能的下降。为了尽可能的避免锁竞争带来的性能损耗，可以通过串行化设计，即消息的处理尽可能在同一个线程内完成，期间不进行线程切换，这样就避免了多线程竞争和同步锁。

为了尽可能提升性能，Netty采用了串行无锁化设计，在I/O线程内部进行串行操作，避免多线程竞争导致的性能下降。表面上看，串行化设计似乎CPU利用率不高，并发程度不够。但是，通过调整NIO线程池的线程参数，可以同时启动多个串行化的线程并行运行，这种局部无锁化的串行线程设计相比一个队列-多个工作线程模型性能更优。

序列化方式对性能的影响

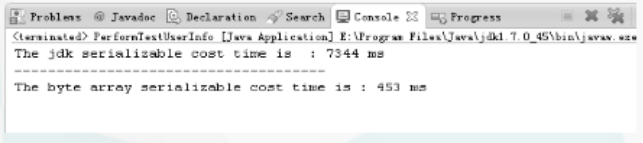
Netty默认提供了对Google Protobuf的支持，通过扩展Netty的编解码接口，用户可以实现其它的高性能序列化框架，例如Thrift的压缩二进制编解码框架。

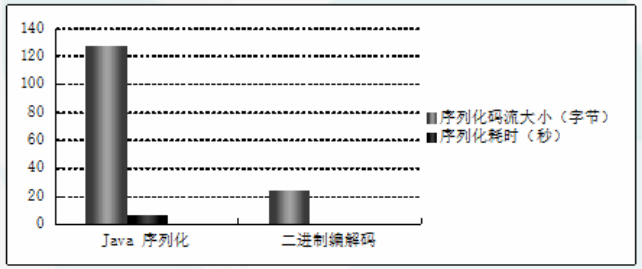
不同的应用场景对序列化框架的需求也不同，对于高性能应用场景Netty默认提供了Google的Protobuf二进制序列化框架，如果用户对其它二进制序列化框架有需求，也可以基于Netty提供的编解码框架扩展实现。

Netty架构剖析之性能篇-序列化方式

影响序列化性能的关键因素总结如下：

- ✓ 序列化后的码流大小（网络带宽占用）
- ✓ 序列化&反序列化的性能（CPU资源占用）
- ✓ 并发调用的性能表现：稳定性、线性增长、偶现的时延毛刺等





对Java序列化和二进制编码分别进行性能测试，编码100万次，测试结果表明：Java序列化的性能只有二进制编码的6.17%左右。

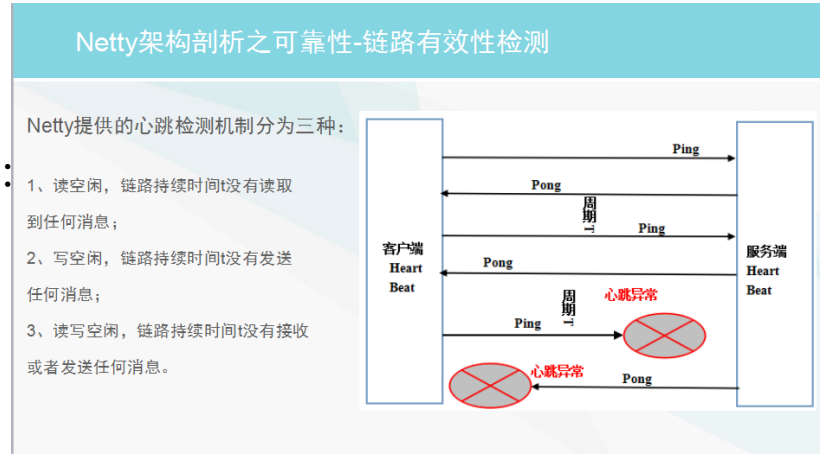
Netty面临的可靠性挑战

1、作为RPC框架的基础网络通信框架，一旦故障将导致无法进行远程服务（接口）调用
2、作为应用层协议的基础通信框架，一旦故障将导致应用协议栈无法正常工作
3、网络环境复杂（例如手游或者推送服务的GSM/3G/WIFI网络），故障不可避免，业务却不能中断

当网络发生单通、连接被防火墙Hang住、长时间GC或者通信线程发生非预期异常时，会导致链路不可用且不易被及时发现。特别是异常发生在凌晨业务低谷期间，当早晨业务高峰期到来时，由于链路不可用会导致瞬间的大批量业务失败或者超时，这将对系统的可靠性产生重大的威胁。从技术层面看，要解决链路的可靠性问题，必须周期性的对链路进行有效性检测。目前最流行和通用的做法就是心跳检测。

心跳检测机制分为三个层面：
1、TCP层面的心跳检测，即TCP的Keep-Alive机制，它的作用域是整个TCP协议栈；
2、协议层的心跳检测，主要存在于长连接协议中。例如SMPP协议；
3、应用层的心跳检测，它主要由各业务产品通过约定方式定时给对方发送心跳消息实现。

心跳检测的目的就是确认当前链路可用，对方活着并且能够正常接收和发送消息。做为高可靠的NIO框架，Netty也提供了基于链路空闲的心跳检测机制：
1、读空闲，链路持续时间t没有读取到任何消息；
2、写空闲，链路持续时间t没有发送任何消息；
3、读写空闲，链路持续时间t没有接收或者发送任何消息。



流量整形功能

流量整形（Traffic Shaping）是一种主动调整流量输出速率的措施。一个典型应用是基于下游网络结点的TP指标来控制本地流量的输出。流量整形与流量监管的主要区别在于，流量整形对流量监管中需要丢弃的报文进行缓存——通常是将它们放入缓冲区或队列内，也称流量整形（Traffic Shaping，简称TS）。当令牌桶有足够的令牌时，再均匀的向外发送这些被缓存的报文。流量整形与流量监管的另一区别是，整形可能会增加延迟，而监管几乎不引入额外的延迟。

Netty支持两种流量整形模式：
1、全局流量整形：全局流量整形的作用范围是进程级的，无论你创建了多少个Channel，它的作用域针对所有的Channel。用户可以通过参数设置：报文的接收速率、报文的发送速率、整形周期。
2、链路级流量整形：单链路流量整形与全局流量整形的最大区别就是它以单个链路为作用域，可以对不同的链路设置不同的整形策略。

可靠性之优雅停机

Java的优雅停机通常通过注册JDK的ShutdownHook来实现，当系统接收到退出指令后，首先标记系统处于退出状态，不再接收新的消息，然后将积压的消息处理完，最后调用资源回收接口将资源销毁，最后各线程退出执行。

通常优雅退出需要有超时控制机制，例如30S，如果到达超时时间仍然没有完成退出前的资源回收等操作，则由停机脚本直接调用kill -9 pid，强制退出。

在实际项目中，Netty作为高性能的异步NIO通信框架，往往用作基础通信框架负责各种协议的接入、解析和调度等，例如在RPC和分布式服务框架中，往往会使用Netty作为内部私有协议的基础通信框架。

当应用进程优雅退出时，作为通信框架的Netty也需要优雅退出，主要原因如下：

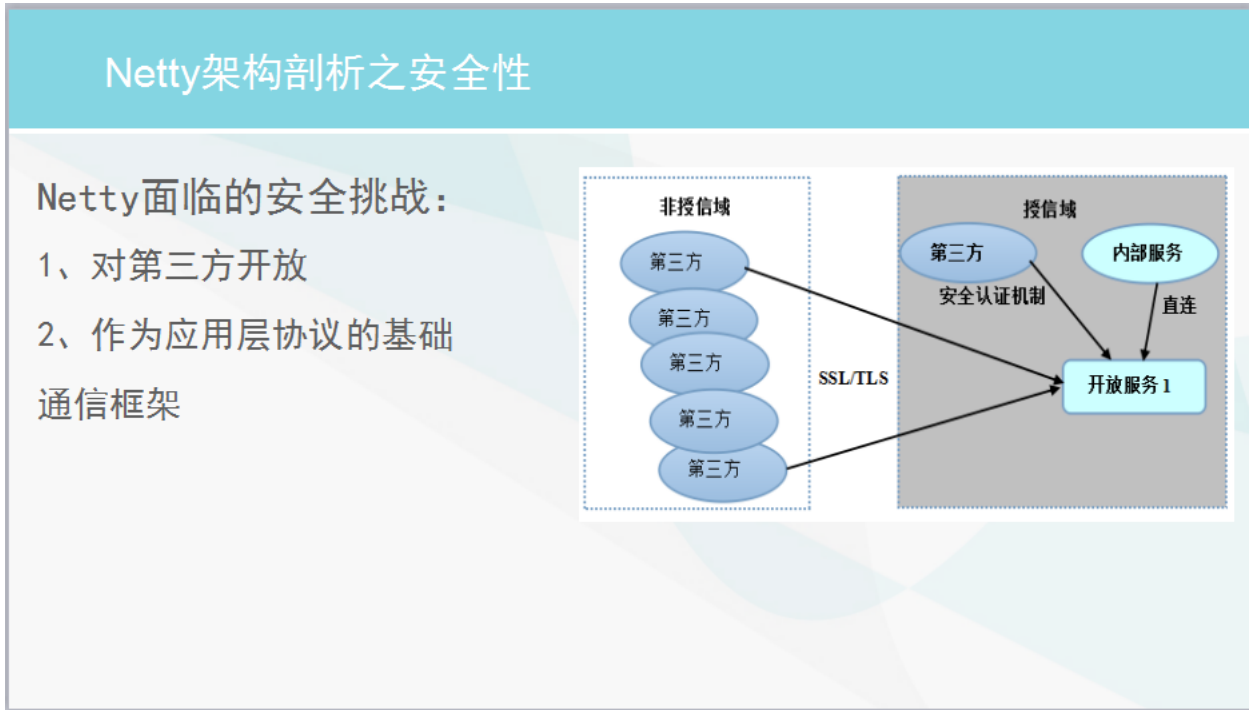
1、尽快的释放NIO线程、句柄等资源；
2、如果使用flush做批量消息发送，需要将积攒在发送队列中的待发送消息发送完成；
3、正在write或者read的消息，需要继续处理；
4、设置在NioEventLoop线程调度器中的定时任务，需要执行或者清理。

安全性相关设计

Netty面临的安全挑战：
1、对第三方开放
2、作为应用层协议的基础通信框架

1、对第三方开放的通信框架：如果使用Netty做RPC框架或者私有协议栈，RPC框架面向非授信的第三方开放，例如将内部的一些能力通过服务对外开放出去，此时就需要进行安全认证，如果开放的是公网IP，对于安全性要求非常高的一些服务，例如在线支付、订购等，需要通过SSL/TLS进行通信

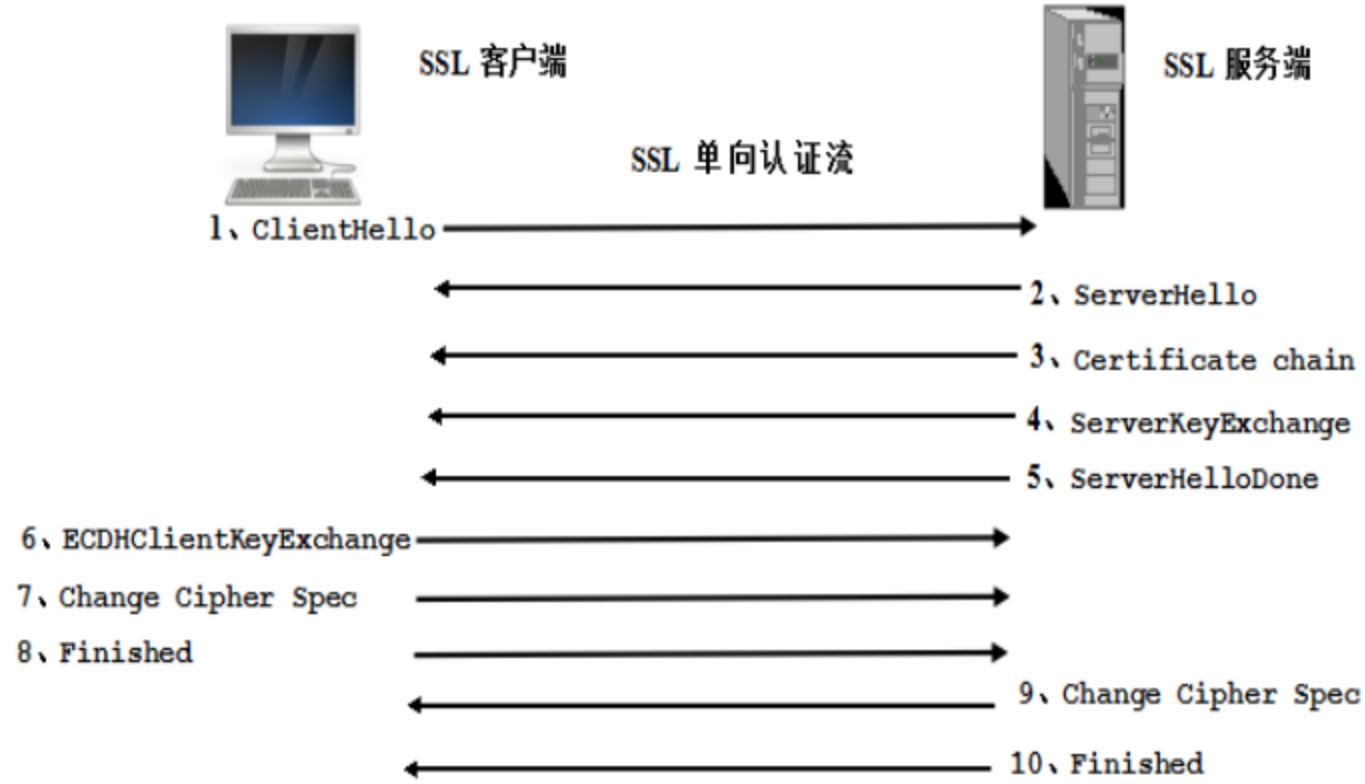
2、应用层协议的安全性。作为高性能、异步事件驱动的NIO框架，Netty非常适合构建上层的应用层协议。由于绝大多数应用层协议都是公有的，这意味着底层的Netty需要向上层提供通信层的安全传输功能。



Netty安全传输特性

Netty安全传输特性：
1、支持SSL V2和V3
2、支持TLS
3、支持SSL单向认证、双向认证和第三方CA认证。

Netty可扩展的安全特性
1、IP地址黑名单机制
2、接入认证
3、敏感信息加密或者过滤机制



Netty通过SslHandler提供了对SSL的支持，它支持的SSL协议类型包括：SSL V2、SSL V3和TLS：
1、单向认证：单向认证，即客户端只验证服务端的合法性，服务端不验证客户端。
2、双向认证：与单向认证不同的是服务端也需要对客户端进行安全认证。这就意味着客户端的自签名证书也需要导入到服务端的数字证书仓库中。
3、CA认证：基于自签名的SSL双向认证，只要客户端或者服务端修改了密钥和证书，就需要重新进行签名和证书交换，这种调试和维护工作量是非常大的。因此，在实际的商用系统中往往会使用第三方CA证书颁发机构进行签名和验证。我们的浏览器就保存了几个常用的CA_ROOT。每次连接到网站时只要这个网站的证书是经过这些CA_ROOT签名过的。就可以通过验证了。

Netty架构的扩展性设计

通过Netty的扩展特性，可以自定义安全策略：
1、线程模型可扩展
2、序列化方式可扩展
3、上层协议栈可扩展
4、提供大量的网络事件切面，方便用户功能扩展

Netty的架构可扩展性设计理念如下：
1、判断扩展点，事先预留相关扩展接口，给用户二次定制和扩展使用；
2、主要功能点都基于接口编程，方便用户定制和扩展。

Netty架构学习材料

InfoQ网站Netty专栏系列文章：
1、Netty系列之Netty高性能之道：http://www.infoq.com/cn/articles/netty-high-performance/
2、Netty系列之Netty可靠性分析：http://www.infoq.com/cn/articles/netty-reliability
3、Netty系列之Netty安全性：http://www.infoq.com/cn/articles/netty-security
4、在infoQ搜索netty关键字，更多文章...

案例集锦系列：
1、《Netty5.0架构剖析和源码解读.pdf》
2、《Netty多线程案例集锦.pdf》

Netty社区：
1、公众号：Netty之家，大量原创的Netty分享资料