第三版：Netty 25 道

什么是 Netty

1、 Netty 是一款提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序。

2、 也就是说，Netty 是一个基于 NIO 的客户、服务器端编程框架。使用 Netty 可以确保你快速和简单地开发出一个网络应用，例如实现了某种协议的客户，服务端应用。Netty 相当简化和流线化了网络应用的编程开发过程，例如，TCP 和 UDP 的 socket 服务开发。

Netty 的特点是什么（ 为什么选择 Netty ）

1、 简单易上手：API 使用简单，学习，使用门槛低。

2、 功能强大：预置了多种编解码功能，支持多种主流协议。

3、 灵活度高：可以通过 ChannelHandler 对通信框架进行灵活的扩展。

4、 高性能：通过与其它业界主流的 NIO 框架对比，Netty 的综合性能最优。

5、 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，更多的新功能会被加入。

7、 案例丰富：经历了大规模的商业应用考验，质量已经得到验证。在互联网、大数据、网络游戏、企业应用。

Netty 为什么说使用简单

1、 无需关心 OP_ACCEPT、OP_READ、OP_WRITE 等等 IO 操作，Netty 已经封装，对我们在使用是透明无感的。

2、 使用 boss 和 worker EventLoopGroup ，Netty 直接提供多 Reactor 多线程模型。

3、 在 Netty 中，我们看到有使用一个解码器 FixedLengthFrameDecoder，可以用于处理定长消息的问题，能够解决 TCP 粘包拆包问题，十分方便。如果使用 Java NIO ，需要我们自行实现解码器。

Netty 的使用场景

1、 构建高性能、低时延的各种 Java 中间件，Netty 主要作为基础通信框架提供高性能、低时延的通信服务。例如：RocketMQ ，分布式消息队列。Dubbo ，服务调用框架。Spring WebFlux ，基于响应式的 Web 框架。

2、 公有或者私有协议栈的基础通信框架，例如可以基于 Netty 构建异步、高性能的 WebSocket、Protobuf 等协议的支持。

3、 各领域应用，例如大数据、游戏等，Netty 作为高性能的通信框架用于内部各模块的数据分发、传输和汇总等，实现模块之间高性能通信。

Netty 如何实现高性能

1、 线程模型 ：更加优雅的 Reactor 模式实现、灵活的线程模型、利用 EventLoop 等创新性的机制，可以非常高效地管理成百上千的 Channel 。

2、 内存池设计 ：使用池化的 Direct Buffer 等技术，在提高 IO 性能的同时，减少了对象的创建和销毁。并且，Netty 的内部实现是用一颗二叉查找树，更好的管理内存分配情况。

3、 内存零拷贝 ：使用 Direct Buffer ，可以使用 Zero-Copy 机制（避免上下分切换频繁）。

4、 协议支持 ：提供对 Protobuf 等高性能序列化协议支持。

Netty 的高性能体现在哪方面

1、 线程模型 ：采用异步非阻塞的 I/O 类库，基于 Reactor 模式实现，解决了传统同步阻塞 I/O 模式下服务端无法平滑处理客户端线性增长的问题。

2、 堆外内存 ：TCP 接收和发送缓冲区采用直接内存代替堆内存，避免了内存复制，提升了 I/O 读取和写入性能。

3、 内存池设计 ：支持通过内存池的方式循环利用 ByteBuf，避免了频繁创建和销毁 ByteBuf 带来的性能消耗。

4、 参数配置 ：可配置的 I/O 线程数目和 TCP 参数等，为不同用户提供定制化的调优参数，满足不同的性能场景。

5、 队列优化 ：采用环形数组缓冲区，实现无锁化并发编程，代替传统的线程安全容器或锁。

6、 并发能力 ： 合理使用线程安全容器、原子类等，提升系统的并发能力。

7、 降低锁竞争 ： 关键资源的使用采用单线程串行化的方式，避免多线程并发访问带来的锁竞争和额外的 CPU 资源消耗问题。

8、 内存泄露检测 ： 通过引用计数器及时地释放不再被引用的对象，细粒度的内存管理降低了 GC 的频率，减少频繁 GC 带来的时延增大和 CPU 损耗。

Netty 的高可靠体现在哪几方面

1、 链路有效性检测：由于长连接不需要每次发送消息都创建链路，也不需要在消息完成交互时关闭链路，因此相对于短连接性能更高。

2、 内存保护机制，Netty 提供多种机制对内存进行保护。通过对象引用计数器对 ByteBuf 进行细粒度的内存申请和释放，对非法的对象引用进行检测和保护。可设置的内存容量上限，包括 ByteBuf、线程池线程数等，避免异常请求耗光内存。

3、 优雅停机：优雅停机功能指的是当系统推出时，JVM 通过注册的 Shutdown Hook 拦截到退出信号量，然后执行推出操作，释放相关模块的资源占用，将缓冲区的消息处理完成或清空，将待刷新的数据持久化到磁盘和数据库中，等到资源回收和缓冲区消息处理完成之后，再退出。

Netty 的可扩展如何体现

1、 提供大量系统参数 ： 供用户按需设置，增强系统的场景定制性。

2、 提供大量的工厂类 ：通过重载这些工厂类，可以按需创建出用户需要的对象。

3、 基于接口的开发 ：关键的类库都提供了接口或抽象类，便于用户自定义实现。

4、 责任链模式 ：ChannelPipeline 基于责任链模式开发，便于业务逻辑的拦截、定制和扩展。

Netty 的核心组件介绍下

1、 Bootstrap & ServerBootstrap

2、 Channel

3、 ChannelFuture

4、 EventLoop & EventLoopGroup

5、 ChannelHandler

6、 ChannelPipeline

什么是 Reactor 模型

反应器设计模式是一个事件处理模式，用于处理一个或多个输入并发地传递给服务处理程序的服务请求。然后，服务处理程序对传入请求进行多路复用，并将它们同步分派给相关的请求处理程序。

可以参考这篇文章：Reactor 模型

TCP 粘包 / 拆包的产生原因，应该这么解决

1、 TCP 是以流的方式来处理数据，所以会导致粘包 / 拆包。

2、 拆包：一个完整的包可能会被 TCP 拆分成多个包进行发送。

3、 粘包：也可能把小的封装成一个大的数据包发送。

4、 Netty 中提供了多个 Decoder 解析类 用于解决上述问题：

5、 FixedLengthFrameDecoder 、LengthFieldBasedFrameDecoder ，固定长度是消息头指定消息长度的一种形式，进行粘包拆包处理的。

6、 LineBasedFrameDecoder 、DelimiterBasedFrameDecoder ，换行是于指定消息边界方式的一种形式，进行消息粘包拆包处理的。

了解哪几种序列化协议

序列化
反序列化

Netty 怎样实现零拷贝

1、 Netty 的接收和发送 ByteBuffer 采用堆外直接内存 Direct Buffer

2、 使用堆外直接内存进行 Socket 读写，不需要进行字节缓冲区的二次拷贝；使用堆内内存会多了一次内存拷贝，JVM 会将堆内存 Buffer 拷贝一份到直接内存中，然后才写入 Socket 中。Netty 创建的 ByteBuffer 类型，由 ChannelConfig 配置。而 ChannelConfig 配置的 ByteBufAllocator 默认创建 Direct Buffer 类型。

3、 CompositeByteBuf 类，可以将多个 ByteBuf 合并为一个逻辑上的 ByteBuf ，避免了传统通过内存拷贝的方式将几个小 Buffer 合并成一个大的 Buffer 。

4、 通过 FileRegion 包装的 FileChannel 。

5、 通过 wrap 方法，我们可以将 byte[] 数组、ByteBuf、ByteBuffer 等包装成一个 Netty ByteBuf 对象，进而避免了拷贝操作。

原生的 NIO 存在 Epoll Bug 有什么 BUG、Netty 是怎么解决的

Java NIO Epoll 会导致 Selector 空轮询，最终导致 CPU 100% 。Netty 对 Selector 的 select 操作周期进行统计，每完成一次空的 select 操作进行一次计数，若在某个周期内连续发生 N 次空轮询，则判断触发了 Epoll 死循环 Bug 。

Netty 空闲检测

IdleStateHandler ，用于检测连接的读写是否处于空闲状态。如果是，则会触发 IdleStateEvent 。

Netty 如何实现重连

1、 客户端，通过 IdleStateHandler 实现定时检测是否空闲

2、 如果空闲，则向服务端发起心跳*

3、 如果多次心跳失败，则关闭和服务端的连接，然后重新发起连接

4、 服务端，通过 IdleStateHandler 实现定时检测客户端是否空闲

5、 如果检测到空闲，则关闭客户端

6、 如果接收到客户端的心跳请求，要反馈一个心跳响应给客户端。

Netty 自己实现的 ByteBuf 有什么优点

1、 它可以被用户自定义的缓冲区类型扩展

2、 通过内置的符合缓冲区类型实现了透明的零拷贝

3、 读和写使用了不同的索引

4、 支持方法的链式调用

5、 支持池化

Netty 为什么要实现内存管理

频繁分配、释放 buffer 时减少了 GC 压力。
在初始化新 buffer 时减少内存带宽消耗(初始化时不可避免的要给 buffer 数组赋初始值)。
及时的释放 direct buffer 。

BIO 是什么?

1、 BIO ，全称 Block-IO ，是一种阻塞 + 同步的通信模式。一种比较传统的通信方式，模式简单，使用方便。但并发处理能力低，通信耗时，依赖网速。

2、 原理：服务器通过一个 Acceptor 线程，负责监听客户端请求和为每个客户端创建一个新的线程进行链路处理。典型的一请求一应答模式。若客户端数量增多，频繁地创建和销毁线程会给服务器打开很大的压力。后改良为用线程池的方式代替新增线程，被称为伪异步 IO 。

NIO 是什么?

1、 NIO ，全称 New IO ，也叫 Non-Block IO ，是一种非阻塞 + 同步的通信模式。Java NIO(New IO 或者 Non Blocking IO ），从 Java 1.4 版本开始引入的非阻塞 IO ，用于替换标准(有些文章也称为传统，或者 Blocking IO 。下文统称为 BIO) Java IO API 的 IO API 。

2、 NIO 相对于 BIO 来说一大进步。客户端和服务器之间通过 Channel 通信。NIO 可以在 Channel 进行读写操作。这些 Channel 都会被注册在 Selector 多路复用器上。Selector 通过一个线程不停的轮询这些 Channel 。找出已经准备就绪的 Channel 执行 IO 操作。

AIO 是什么?

AIO ，全称 Asynchronous IO ，也叫 NIO2 ，是一种非阻塞 + 异步的通信模式。在 NIO 的基础上，引入了新的异步通道的概念，并提供了异步文件通道和异步套接字通道的实现。AIO 并没有采用 NIO 的多路复用器，而是使用异步通

道的概念。其 read，write 方法的返回类型，都是 Future 对象。而 Future 模型是异步的，其核心思想是：去主函数等待时间。

BIO、NIO 有什么区别?

线程模型不同

1、 BIO：一个连接一个线程，客户端有连接请求时服务器端就需要启动一个线程进行处理。所以，线程开销大。可改良为用线程池的方式代替新创建线程，被称为伪异步 IO 。

2、 NIO：一个请求一个线程，但客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有新的 I/O 请求时，才启动一个线程进行处理。可改良为一个线程处理多个请求，基于 多 Reactor 模型。

3、 BIO 是面向流(Stream)的，而 NIO 是面向缓冲区(Buffer)的。

4、 BIO 的各种操作是阻塞的，而 NIO 的各种操作是非阻塞的。

5、 BIO 的 Socket 是单向的，而 NIO 的 Channel 是双向的。