# 基本概念

1、Netty 是一个 基于 NIO 的 client-server(客户端服务器)框架，使用它可以快速简单地开发网络应用程序。（高性能的网络编程框架）

2、它极大地简化并优化了 TCP 和 UDP 套接服务器等网络编程,并且性能以及安全性等很多方面甚至都要更好。

3、支持多种协议，如 FTP，SMTP，HTTP 以及各种二进制和基于文本的传统协议。

用官方的总结就是: Netty 成功地找到了一种在不妥协可维护性和性能的情况下实现易于开发，性能，稳定性和灵活性的方法。除了上面介绍的之外，很多开源项目比如我们常用的 Dubbo、RocketMQ、Elasticsearch、gRPC等都用到了 Netty。

# 为什么使用Netty

1、统一的 API，支持多种传输类型，阻塞和非阻塞的。

2、简单而强大的线程模型。

3、自带编解码器解决 TCP 粘包/拆包问题。

4、自带各种协议栈。

5、真正的无连接数据包套接字支持。

6、比直接使用 Java 核心 API 有更高的吞吐量、更低的延迟、更低的资源消

耗和更少的内存复制。

7、安全性不错，有完整的 SSL/TLS 以及 StartTLS 支持。

8、社区活跃。

9、成熟稳定，经历了大型项目的使用和考验，而且很多开源项目都使用到了Netty， 比如我们经常接触的 Dubbo、 RocketMQ 等等。

# 支持的协议

1、**TCP协议**：Netty最常用的功能是通过TCP协议进行网络通信。它提供了高性能、可靠的TCP连接管理和数据传输的实现。

2、**HTTP协议**：Netty提供了HTTP编解码器以支持HTTP和HTTPS协议。它可以用于构建Web服务器或客户端，并提供了对HTTP请求和响应的完整支持。

3、**WebSockets协议**：Netty提供了对WebSockets协议的全面支持，使得构建实时应用程序变得更加容易。WebSockets是一种在单个TCP连接上实现双向通信的协议。

4、**UDP协议**：Netty支持基于UDP协议的数据包传输。它提供了UDP编解码器和多播支持，可以用于构建高性能的实时应用程序或游戏服务器。

5、**WebSocket over HTTP/2协议**：Netty支持将WebSocket与HTTP/2协议结合使用，从而提供更高效的双向通信。这种组合可以提供更好的性能和协议多路复用。

6、**SMTP/IMAP/POP3协议**：Netty提供了一组用于构建邮件客户端和服务器的协议实现。它支持SMTP（Simple Mail Transfer Protocol）、IMAP（Internet Message Access Protocol）和POP3（Post Office Protocol 3）等协议。

7、**DNS协议**：Netty提供了对DNS（Domain Name System）协议的支持，可以用于构建高性能的DNS客户端和服务器。

8、MQTT协议：Netty通过提供相关的MQTT编解码器和处理器，使得构建MQTT客户端和服务器变得更加简单和高效。

# 应用场景

## 1、作为 RPC 框架的网络通信工具

我们在分布式系统中，不同服务节点之间经常需要相互调用，这个时候就需要 RPC 框架了。不同服务节点之间的通信是如何做的呢? 可以使用 Netty 来做。比如我调用另外一个节点的方法的话，至少要让对方知道我调用的是哪个类中的哪个方法以及相关参数吧。

## 2、实现一个自己的 HTTP 服务器

通过 Ntty 我们可以自己实现一个简单的 HTTP 服务器，这个大家应该不陌生。说到 HTTP 服务器的话，作为 Java 后端开发，我们一般使用 Tomcat 比较多。一个最基本的 HTTP 服务器要处理常见的 HTTP Method 的请求，比如 POST 请求、GET 请求等等。

## 3、实现一个即时通讯系统

使用 Netty 我们可以实现一个可以聊天类似微信的即时通讯系统，这方面的开源项目还蛮多的，可以自行去 Github 找一找。

## 4、实现消息推送系统

市面上有很多消息推送系统都是基于 Netty 来做的。

# IO模型

# Reactor 模型

## 基本概念

Reactor 模型是一种用于构建高性能、可扩展的并发系统的设计模式。它将事件驱动和非阻塞 I/O 结合在一起，以实现高效的并发处理。

## 核心思想

将事件处理过程分为两个阶段：事件分发和事件处理。

1、事件分发（Dispatch）：在 Reactor 模型中，有一个专门负责监听事件的组件，称为事件分发器（Dispatcher）。它会不断地监听各种事件，包括外部输入、计时器触发等。当有事件发生时，事件分发器会将事件分发给对应的事件处理器（Handler）进行处理。

（1）单线程模型：在简单的 Reactor 模型中，事件分发器和事件处理器共用一个线程。该线程负责监听事件，并根据事件类型将事件分发给相应的事件处理器。事件处理器执行相应的业务逻辑，处理事件所需的计算和 I/O 操作。

（2）多线程模型：在多线程的 Reactor 模型中，事件分发器和事件处理器分别运行在不同的线程中。事件分发器仍然负责监听事件，但它不再直接处理事件，而是将事件派发给线程池中的某个线程进行处理。这样可以利用多个线程同时处理不同的事件，提高系统的并发能力。

2、事件处理（Handler）：事件处理器负责具体的事件处理逻辑。它会根据事件类型执行相应的操作，可以是进行计算、读写数据、发起网络请求等。事件处理器通常采用非阻塞的方式进行操作，以充分利用系统资源并提高并发性能。

## 总结

Reactor 模型是一种用于构建高性能、可扩展的并发系统的设计模式。它通过将事件处理过程分为事件分发和事件处理两个阶段，并采用非阻塞 I/O 和事件驱动的方式，提供了一种高效、可扩展的并发处理方案。在实际应用中，很多网络编程框架如 Netty 就使用了 Reactor 模型来实现高性能的并发通信。

# 核心组件

## Channel（通道）

通道，Netty网络操作抽象类，包括基本的 I/O 操作，如 bind、connect、read、write 等，Netty 的 Channel 接口所提供的 API，大大地降低了直接使用 Socket 类的复杂性。

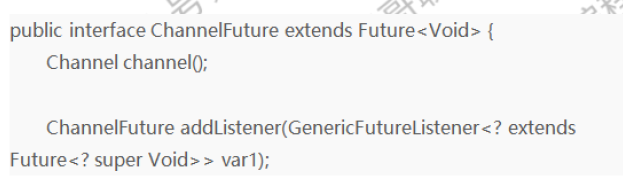
## EventLoop（事件循环）

EventLoop 定义了 Netty 的核心抽象，用于处理连接的生命周期中所发生的事件。说白了EventLoop 的主要作用实际就是负责监听网络事件并调用事件处理器进行相关 I/O操作的处理。

Channel为 Netty 网络操作(读写等操作)的抽象类，EventLoop 负责处理注册到其上的 Channel 处理I/O 操作，两者配合参与 I/O 操作。

## ChannelFuture

Netty 是异步非阻塞的，所有的 I/O操作都为异步的。因此我们不能立刻得到操作是否执行成功，但是你可以通过 ChannelFuture接口的 addListener() 方法注册一个 ChannelFutureListener，当操作执行成功或者失败时监听就会自动触发返回结果，并且你还可以通过 ChannelFuture 的 channel() 方法获取关联的 Channel。



## ChannelHandler

**ChannelHandler 是消息的具体处理器。他负责处理各种任务，这个任务非常广泛，可以是读写事件、连接、解码编码、数据转换、业务逻辑等等，处理完毕之后将数据继续转发到ChannelPipeline中的下一个ChannelHandler。**

通过定制ChannelHandler可对Netty进行扩展。ChannelHandler接口本身并没有提供很多方法，因为这个接口有许多的方法需要实现，为了方便使用，可以继承它的子类：

1、ChannelInboundHandler用于处理入站I/O事件

2、

# TCP粘包拆包

https://www.cnblogs.com/AIPAOJIAO/p/10631551.html

# 接口

## Channel

### 基本概念

在Netty中，Channel接口起着核心作用，用于表示一个开放的连接，如一个网络套接字。Channel接口定义了与底层I/O操作进行交互的方法，并提供了许多功能来管理通信、处理数据以及处理事件。

### 方法

#### EventLoop eventLoop();

返回分配给该Channel的EventLoop。EventLoop负责处理Channel的I/O事件，包括读写和连接操作。

#### isActive()

检查Channel是否处于活动状态，即该Channel是否已经打开并且成功连接。

#### SocketAddress localAddress();

返回Channel绑定的本地地址。如果Channel未绑定到任何本地地址，则返回null。

#### SocketAddress remoteAddress();

返回与Channel关联的远程地址。如果Channel未与远程地址关联，则返回null。

#### write()

将指定的消息写入Channel。这个方法是异步的，它会将消息放入发送缓冲区，具体的发送时间取决于底层的I/O事件和网络状况。

#### flush()

将发送缓冲区的消息立即发送出去。该方法会阻塞直到所有消息都被发送完毕。

#### ChannelFuture writeAndFlush(Object msg);

用于将数据写入通道并立即刷新到远程对等方。

用法：

1、获取 Channel 对象：首先我们需要获取到要操作的 Channel 对象。



2、创建消息数据：创建要发送的消息数据，可以是字节数组、字符串或其他可序列化的对象。



3、调用 writeAndFlush 方法：使用 Channel 对象调用 writeAndFlush 方法，将消息写入通道并立即刷新到远程对等方。



4、监听写操作完成：可以通过添加 ChannelFutureListener 来监听写操作的完成情况。



#### ChannelPipeline pipeline();

返回与Channel相关联的ChannelPipeline。ChannelPipeline负责管理Channel中的所有处理器（Handler）并协调消息的处理流程。

#### ChannelConfig config();

返回Channel配置对象，用于设置和获取Channel的各种配置参数。

#### close()

关闭Channel。调用该方法将会触发Channel的关闭事件，并释放相关的资源。

#### boolean isWritable();

检查发送缓冲区是否有足够的空间可供写入。如果发送缓冲区已满，则返回false

#### attr()

返回一个用于绑定和获取Channel属性的AttributeMap。通过AttributeMap，可以为Channel绑定一些自定义的属性。

#### alloc()

用于获取一个新的 ByteBufAllocator 实例。ByteBufAllocator 是用于创建 ByteBuf 实例的工厂。

### 作用

#### 1、提供异步I/O操作

Channel提供了异步的读写操作，可以通过write()方法将数据写入到Channel的发送缓冲区，然后通过flush()方法将缓冲区中的数据发送到目标。这种异步方式可以提高系统的性能和吞吐量。

#### 2、事件驱动机制

Channel基于事件驱动的模型，使用EventLoop处理各种I/O事件，如读取数据、写入数据、连接建立和关闭等。通过注册感兴趣的事件，Channel可以接收并处理这些事件。

#### 3、支持多种传输协议

Channel可以与多种传输协议进行集成，如TCP、UDP、Unix Domain Socket等。这样可以在不同的网络场景下进行灵活的选择和配置。

#### 4、提供高级的特性和功能

Channel提供了一系列高级特性和功能，如流控制、超时管理、自定义编码解码器、拆包粘包处理等。这些功能可以帮助开发人员更方便地构建复杂的网络应用程序。

#### 5、ChannelPipeline的管理

每个Channel都有一个关联的ChannelPipeline，ChannelPipeline中包含了一系列的ChannelHandler。通过ChannelPipeline，可以按顺序添加、移除和替换不同的处理器，从而构建处理消息的流水线。

#### 6、属性的绑定和获取

Channel提供了一个AttributeMap，用于绑定和获取自定义属性。可以将一些自定义的数据与Channel关联起来，并在需要时进行访问。

### 总结

总而言之，Channel接口提供了网络通信的基础能力，是Netty中完成高性能、可扩展网络应用程序开发的基石。它通过异步、事件驱动的方式处理各种I/O操作，提供了丰富的特性和功能，同时还支持多种传输协议的集成。开发人员可以利用Channel接口构建高性能、灵活的网络应用程序，满足各种复杂的业务需求。

## ChannelPipeline

## ChannelId

### 基本概念

ChannelId 接口是用来唯一标识一个 Channel 的。每个 Channel 都有一个对应的 ChannelId 对象，通过它可以快速地查找和访问 Channel。

## EventLoop

## EventLoopGroup

## ChannelFuture

### 基本概念

## ChannelHandler

### 基本概念

## ChannelInboundHandler

### 基本概念

ChannelInboundHandler 是 Netty 中用于处理入站事件的接口之一。它负责处理从远程对等方接收到的数据，并在通道的生命周期内触发相关事件，例如新连接、读取数据、连接关闭等。它用于处理从远程对等方接收到的数据，并在通道的生命周期内触发事件。

这个接口继承自ChannelHandler接口。

### 方法

#### void channelRegistered(ChannelHandlerContext ctx)

#### void channelActive(ChannelHandlerContext ctx)

当一个新的活动连接建立时被调用，通常在此方法中进行连接初始化操作。

#### void channelInactive(ChannelHandlerContext ctx)

当与远程对等方的连接断开时被调用。

#### void channelRead(ChannelHandlerContext ctx, Object msg)

当从远程对等方接收到新的数据时被调用。通常在此方法中对接收到的消息进行处理。

#### void channelReadComplete(ChannelHandlerContext ctx)

当通道上的数据读取完成时被调用，通常在此方法中进行数据处理的收尾工作。

#### void exceptionCaught(ChannelHandlerContext ctx, Throwable cause)

当发生异常时被调用。通常在此方法中处理异常情况。

### 总结

ChannelInboundHandler 接口是 Netty 中用于处理入站事件的重要组件。通过实现该接口并重写相关方法，你可以自定义处理新连接、数据读取和异常情况等事件。这使得你可以根据需要编写逻辑来处理不同的操作。

## ChannelOutboundHandler

## ChannelHandlerContext

## ChannelFutureListener

## SocketChannel

### 基本概念

SocketChannel 是 Netty 框架中用于网络通信的一个重要接口。它是基于 NIO 技术实现的，提供了与远程主机进行连接和数据传输的能力。

### 作用

#### 1、网络通信

#### 2、实现非阻塞IO

#### 3、支持TCP协议

#### 4、客户端和服务器端通信

#### 5、支持异步IO

#### 6、连接管理和状态控制

### 方法

#### ByteBufAllocator alloc()

## ChannelOutboundInvoker

## FileRegion

## ChannelPipeline

## ByteBufAllocator

# 类

## NioEventLoopGroup

### 基本概念

在 Netty 中，NioEventLoopGroup 是一个非常重要的类，它是用于处理网络事件的事件循环组。NioEventLoopGroup 的作用是管理一组 NioEventLoop，每个 NioEventLoop 对应一个线程，负责处理网络事件和 I/O 操作。它充当了整个网络通信过程中的调度器和执行者的角色。

### 作用

#### 1、****封装了底层的 I/O 线程模型****

Netty 提供了多种不同的 I/O 线程模型，例如基于 NIO 的线程模型（NioEventLoopGroup）、基于 Epoll 的线程模型（EpollEventLoopGroup）等。NioEventLoopGroup 封装了 NIO 线程模型，提供了方便的接口和方法，简化了底层 I/O 处理的复杂性。

#### **2、**管理 I/O 线程池****

NioEventLoopGroup 内部维护了一个线程池，用于执行各种 I/O 相关的任务。它能够根据实际需求动态地调整线程数目，确保高效地处理并发请求。通过使用线程池，可以避免为每个连接创建一个线程的开销，提高资源利用率。

#### **3、**接收和处理网络事件****

NioEventLoopGroup 中的 EventLoop 负责接收和处理各种网络事件，例如新的连接、数据到达、数据发送完成等。它使用高效的事件驱动机制，能够快速响应和处理大量的并发事件。

#### **4、**调度任务的执行****

除了处理 I/O 事件外，NioEventLoopGroup 还可以调度延迟任务和定时任务的执行。通过使用 schedule() 方法，可以在指定的时间点或者延迟一定时间后执行任务，方便地处理一些定时操作。

### 总结

NioEventLoopGroup 提供了一个高性能的线程池，并负责管理和调度底层的 I/O 操作。它简化了网络编程中的复杂性，使开发人员能够更专注于业务逻辑的实现，同时提供了高效、可靠的网络通信能力。

## NioEventLoop

### 基本概念

NioEventLoop 是 Netty 框架中的一个关键类，它是 NioEventLoopGroup 的成员之一，用于处理具体的 I/O 事件。在 Netty 中，NioEventLoop 实际上是一个事件循环，负责处理特定的任务，例如接受新的连接、读取数据、写入数据等。每个 NioEventLoop 对象都与一个线程相关联，并且在生命周期内不会改变。

### 作用

#### 1、****负责处理 I/O 事件****

NioEventLoop 通过不断地轮询注册在其中的 SelectableChannel，以便处理各种 I/O 事件，包括连接已建立、数据已到达、可以进行数据读取或写入等等。根据不同的事件类型，NioEventLoop 会调用相应的回调方法对事件进行处理，并触发相应的业务逻辑。

#### **2、**管理 Selector 和 SelectionKey****

NioEventLoop 内部维护了一个 Selector 对象，用于监听和接收事件。当有新的连接到达或者数据可读写时，Selector 会将事件通知给关联的 NioEventLoop，然后由该 NioEventLoop 处理事件。此外，NioEventLoop 还负责管理 SelectionKey，用于对感兴趣的事件进行注册和取消注册。

#### **3、**执行任务和定时任务****

除了处理 I/O 事件外，NioEventLoop 还能执行其他类型的任务。它有一个任务队列，用于存放用户提交的任务和定时任务，并按顺序依次执行。通过提交任务或者定时任务，可以在 NioEventLoop 的事件循环中执行一些非 I/O 相关的任务，例如业务逻辑的处理或者定时操作。

#### **4、**提供线程安全的特性****

NioEventLoop 提供了线程安全的特性，保证了在多线程并发环境下对事件和任务的处理是安全的。它使用了一些同步机制，如单线程执行、互斥锁等，以保障内部状态的一致性和正确性。

### 总结

NioEventLoop 是 Netty 框架中用于处理 I/O 事件的核心类。它负责监听、接收和处理网络事件，管理 Selector 和 SelectionKey，并执行用户提交的任务和定时任务。通过高效地利用底层的 NIO 特性和线程模型，NioEventLoop 实现了高性能、高并发的网络通信能力，为 Netty 提供了卓越的性能和可靠性。

## Selector

## SelectionKey

## ChannelOption

### 基本概念

### 常量

#### SO\_BACKLOG

用于设置服务器套接字的等待连接队列的最大长度。

#### SO\_KEEPALIVE

用于设置套接字的 keep-alive 属性。

#### SO\_RCVBUF

表示网络套接字（socket）的接收缓冲区大小，接收缓冲区大小单位是字节（bytes）。

接收缓冲区是用于存储接收到的数据的内存区域。当一个网络套接字在接收数据时，接收缓冲区将存储接收到的数据，然后应用程序可以从缓冲区读取数据进行处理。

## ServerBootstrap

### 基本概念

ServerBootstrap 类是 Netty 中用于创建服务器端应用程序的关键类之一。主要作用是配置和启动 Netty 服务器。它提供了一组方法，用于设置服务器的各种属性、协议、处理器等，并最终创建并启动一个服务器实例。

### 方法

#### group(EventLoopGroup parentGroup, EventLoopGroup childGroup)

指定服务器的父组和子组。父组用于接受传入的连接请求，子组用于处理已经建立的连接。通常情况下，可以使用同一个 NioEventLoopGroup 对象作为父组和子组，以共享线程池资源。

#### channel(Class<? extends ServerChannel> channelClass)

指定服务器使用的通道类型。通道是网络数据的流动通道，不同的通道类型实现了不同的底层协议和处理方式。常见的通道类型有 NioServerSocketChannel、EpollServerSocketChannel 等。

#### option(ChannelOption<T> option, T value)

设置服务器的选项。ChannelOption 表示服务器的一些配置选项，例如 TCP 的连接参数、接收和发送缓冲区大小等。通过这个方法，我们可以设置这些选项的值。

#### childOption(ChannelOption<T> option, T value)

设置服务器子通道的选项。这些选项将应用于已经建立的连接的子通道，例如接收和发送缓冲区大小、心跳检测等。

#### handler(ChannelHandler handler)

#### childHandler(ChannelHandler handler)

#### bind(int inetPort)

#### bind(SocketAddress localAddress)

### 主要作用

#### 1、创建服务器

ServerBootstrap 可以通过调用 bind() 方法将服务器绑定到指定的网络地址（如 IP 地址和端口号）。这样，服务器就可以监听指定地址上的连接请求，并与客户端建立连接。

#### 2、配置服务器

ServerBootstrap 提供了一系列方法来设置服务器的各种属性和行为。通过这些方法，可以轻松配置服务器的选项、通道类型、处理器等。例如，可以设置 TCP 参数、接收和发送缓冲区大小、通道类型为 NIO 或者 Epoll、添加处理器来处理连接请求和业务逻辑等。

#### 3、管理连接

ServerBootstrap 通过内部的 Channel 和 EventLoopGroup 管理服务器的连接和事件处理。Channel 表示服务器的通道，它负责接收和发送数据。EventLoopGroup 则负责处理连接请求、读写数据等事件。通过配置合适的 Channel 类型和 EventLoopGroup，可以实现高效的事件处理和并发性能。

#### 4、处理请求

使用 ServerBootstrap，可以为服务器配置处理器（ChannelHandler），用于对连接请求和数据进行处理。可以根据不同的事件类型，例如接收请求、读取数据、写入数据等，在处理器中编写相应的业务逻辑。通过添加合适的处理器，可以实现自定义的协议解析、数据转换、安全认证等功能。

### 总结

## Bootstrap

### 基本概念

Bootstrap 是 Netty 框架中用于创建和配置客户端的类。它提供了一种方便而灵活的方式来构建高性能、可伸缩的网络客户端。

### 作用

#### 1、创建客户端

#### 2、配置客户端

#### 3、管理连接

#### 4、处理请求

### 方法

#### group(EventLoopGroup group)

#### channel(Class<? extends C> channelClass)

#### handler(ChannelHandler handler)

#### option(ChannelOption<T> option, T value)

#### connect(SocketAddress remoteAddress)

#### connect(SocketAddress remoteAddress, SocketAddress localAddress)

#### remoteAddress(SocketAddress remoteAddress)

#### localAddress(SocketAddress localAddress)

## SocketAddress

### 基本概念

SocketAddress 类是 Java 中用于表示网络套接字地址的抽象类。它可以用来表示不同类型的网络地址，如 IP 地址和端口号。SocketAddress 类的作用是将网络套接字的地址信息封装在一个对象中，以便在网络通信中使用。

### 作用

## InetSocketAddress

### 基本概念

### 作用

## NioServerSocketChannel

### 基本概念

NioServerSocketChannel 是 Netty 框架中用于创建服务器端的通道类。它是基于 NIO 技术实现的，通过底层的 Java NIO 类库提供了高性能的非阻塞 I/O 网络通信能力。

### 作用

1、创建服务器端通道

2、基于NIO实现

3、非阻塞IO模型

4、配置通道选项

5、事件处理链

6、接受连接

### 总结

## ChannelInitializer

### 基本概念

ChannelInitializer 是 Netty 框架中的一个特殊的 ChannelHandler，用于初始化通道的处理链（ChannelPipeline）。它在通道被创建时自动被调用，用于添加一系列的处理器和配置到通道的处理链中。

### 作用

#### 1、****初始化通道处理链****

ChannelInitializer 主要的作用是初始化通道的处理链。当通道被创建时，ChannelInitializer 的 initChannel() 方法会被自动调用，开发者可以在该方法中添加不同类型的处理器到通道的处理链中。

#### ****2、添加处理器****

通过 ChannelInitializer 的 initChannel() 方法，可以添加各种类型的处理器（ChannelHandler）到通道的处理链中。这些处理器可以包括编码器、解码器、业务处理器等，用于对数据进行读写、编解码、业务逻辑处理等操作。

#### ****3、配置通道选项****

除了添加处理器之外，ChannelInitializer 也可以用于配置通道的选项。通过重写 initChannel() 方法，可以设置如 TCP 参数、接收和发送缓冲区大小、心跳机制等通道选项，以满足具体的业务需求。

#### ****4、支持复杂的管道配置****

#### ****5、适用于服务器和客户端****

### 总结

## MessageToByteEncoder

### 基本概念

## ByteToMessageDecoder

### 基本概念

ByteToMessageDecoder 是 Netty 提供的一个核心类，用于将字节数据解码为消息对象。ByteToMessageDecoder 的作用是在接收到字节数据后，将字节数据解码成应用程序可以理解的消息对象。它是 Netty 中解码器的基类，在实际使用中，通常需要继承 ByteToMessageDecoder 类，并根据实际需求重写其中的方法。

### 工作原理

1、在 Netty 接收到字节数据时，会调用 ByteToMessageDecoder 的 decode() 方法。

2、decode() 方法将字节数据解码为一个或多个消息对象。

3、解码后的消息对象将被添加到解码器的输出列表中，供后续的处理器进行处理。

### 方法

#### decode()

#### channelRead()

#### decodeLast()

### 使用场景

## StringEncoder

### 基本概念

## StringDecoder

### 基本概念

StringDecoder是Netty中的一个解码器（Decoder），用于将字节流解码为字符串。它属于Netty的编解码器框架，用于简化网络通信中的数据传输和处理。具体来说，StringDecoder的主要作用是将接收到的字节流按照特定的字符集编码方式转换为字符串。在网络通信中，数据往往以字节流的形式进行传输，因此需要对字节流进行解码才能正确地处理数据。StringDecoder便是专门负责将字节流解码为字符串的解码器之一。

### 使用方法

1、创建一个StringDecoder对象。

2、将该解码器添加到Netty的ChannelPipeline中，作为数据处理的一部分。

3、在数据传输过程中，当接收到的数据需要被解码为字符串时，StringDecoder会自动将字节流转换为字符串。

需要注意的是，StringDecoder并非只能用于解码字符串类型的数据。在Netty中，解码器的逻辑是可以自定义的，你可以根据具体需求继承StringDecoder并重写相应的方法来实现自己的解码逻辑。例如，你可以在继承StringDecoder后实现 JSON 数据的解码。这表明StringDecoder不仅仅局限于字符串的解码，更重要的是它提供了一种常用数据解析的示例。

### 方法

#### protected void decode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out)

该方法是ChannelInboundHandlerAdapter类中的一个抽象方法，在StringDecoder中被实现。它负责将字节流解码为字符串，并将解码后的字符串添加到输出列表out中。

需要注意的是，该方法可能会被多次调用，每次调用的输入数据不一定完整，因此解码逻辑需要处理好数据的边界情况。

## LineBasedFrameDecoder

### 基本概念

LineBasedFrameDecoder是Netty中的一个解码器（Decoder），用于将基于换行符的文本数据切分为一行一行的数据帧。它通常与StringDecoder搭配使用，用于处理基于文本的协议。LineBasedFrameDecoder的主要作用是根据换行符（通常为\n或\r\n）将接收到的字节流切分为独立的一行数据。

### 使用方法

1、创建一个LineBasedFrameDecoder对象，指定每行数据的最大长度。

2、将该解码器添加到Netty的ChannelPipeline中，作为数据处理的一部分。

3、在数据传输过程中，LineBasedFrameDecoder会自动将接收到的字节流按照换行符进行切分，并将切分后的数据帧交给后续的处理器进行处理。

需要注意的是，LineBasedFrameDecoder并非只能用于解码换行符分隔的文本数据。实际上，它是一种通用的解码器，可以根据自定义的分隔符将数据切分为不同的数据帧。

### 总结

LineBasedFrameDecoder是Netty中的一个解码器，用于将基于换行符的文本数据切分为独立的一行数据。它能够简化处理基于文本的协议的开发工作，并提高数据传输的效率。使用时，需要指定每行数据的最大长度，并将该解码器添加到ChannelPipeline中。

## DelimiterBasedFrameDecoder

## LengthFieldBasedFrameDecoder

## FixedLengthFrameDecoder

### 基本概念

FixedLengthFrameDecoder是Netty中的一个解码器（Decoder），用于将固定长度的字节数据切分为固定大小的数据帧。它可以帮助我们处理一些基于固定长度的数据传输协议。

## JsonObjectDecoder

## SimpleChannelInboundHandler

### 基本概念

SimpleChannelInboundHandler是Netty中用于处理入站消息的抽象类。它是Netty提供的一种特殊类型的ChannelInboundHandler，用于处理从远程对等方接收到的数据。SimpleChannelInboundHandler的作用是将处理逻辑集中在消息的内容上，而不需要显式释放资源。它通过继承ChannelInboundHandlerAdapter类，并实现泛型接口来定义具体的消息处理逻辑。

### 方法

#### channelRead0(ChannelHandlerContext ctx, I msg)

## ChannelInboundHandlerAdapter

### 基本概念

## ByteBuf

### 基本概念

ByteBuf 是 Netty 框架中用于高效处理字节数据的缓冲区类。它提供了一组强大且灵活的方法，用于读取、写入和操作字节数据。

### 作用

#### 1、字节数据存储

#### 2、读写操作

#### 3、可扩展的缓冲区

#### 4、零拷贝

#### 5、片段和汇总

#### 6、支持多种字节编解码

### 方法

#### 读取方法

##### getByte()

##### public abstract byte readByte()

读取一个字节

##### public abstract ByteBuf readBytes(byte[] dst)

##### public abstract int readableBytes()

返回可读的字节数

#### 写入方法

##### writeByte(byte value)

写入一个字节

#### 标记和重置位置

##### public abstract ByteBuf markReaderIndex()

##### resetReaderIndex()

#### 查找方法

### 总结

ByteBuf 是 Netty 框架中用于高效处理字节数据的缓冲区类。它提供了一组丰富的读写方法，支持自动扩容和缩容，具备零拷贝特性，可以进行片段和汇总操作，同时还支持多种字节编解码。通过使用 ByteBuf，可以方便、高效地处理字节数据，在网络通信和协议解析等场景中发挥重要作用。

## PooledSlicedByteBuf

## Unpooled

### 基本概念

Unpooled 类是 Netty 提供的一个实用工具类，用于创建和管理 ByteBuf 实例。它提供了各种静态方法，用于创建不同类型的 ByteBuf。

### 方法

### 作用和方法

#### 1、创建堆缓冲区

buffer()：创建一个基于堆内存的 ByteBuf 实例。

buffer(int capacity)：创建一个指定容量的基于堆内存的 ByteBuf 实例。

Unpooled 类方法返回的 ByteBuf 实例可以直接使用，无需手动释放内存，因为 Netty 会在适当的时候自动进行内存回收。它提供了一种方便、高效的方式来创建和处理字节缓冲区，特别适用于网络通信和协议解析的场景。需要注意的是，Unpooled 类的方法创建的 ByteBuf 实例默认使用的是大端字节顺序（Big Endian），可以通过调用 order() 方法设置使用小端字节顺序（Little Endian）。

### 总结

Unpooled 类是 Netty 中用于创建和管理 ByteBuf 实例的工具类，简化了字节缓冲区的创建和操作过程，提高了开发效率。

## TextWebSocketFrame

### 基本概念

### 方法

### 作用

# 注解

## @ChannelHandler.Sharable

### 基本概念

@ChannelHandler.Sharable 是 Netty 框架提供的一个注解，用于标记一个 ChannelHandler 是否可共享。Sharable 注解可以被应用在实现了 Netty 的 ChannelHandler 接口的类上。当一个 ChannelHandler 被标记为 @ChannelHandler.Sharable 时，表示该处理器是可共享的，也就是说可以被多个 ChannelPipeline 共用，即可以被多个 Channel 实例共享使用。

### 作用

#### 1、线程安全性

被 @ChannelHandler.Sharable 注解标记的处理器必须保证其线程安全性，因为它可以在多个 Channel 上同时被调用。这意味着在处理器中不应该使用实例变量来存储会被修改的状态，或者必须采取线程安全的方式来处理状态。

#### 2、资源重用

#### 3、共享处理逻辑

### 总结

@ChannelHandler.Sharable 注解用于标记一个 Netty ChannelHandler 是否可共享，通过共享处理器实例可以提升性能和资源利用率，但需要确保处理器的线程安全性。

## @Skip

### 基本概念

在 Netty 中，@Skip 注解用于标记方法或类，表示该方法或类不会被 Netty 的代码生成工具处理。具体来说，@Skip 注解的作用是告诉 Netty 在代码生成过程中跳过对被注解的方法或类的处理。

# 问题

## 1、java.lang.UnsupportedOperationException: unsupported message type: MessageBean (expected: ByteBuf, FileRegion)