# 数据流

## IO模型

BIO(Blocking IO):同步并阻塞IO，一个连接一个线程

1：每个请求都需要独立的线程完成数据 Read，业务处理，数据 Write 的完整操作问题。

2：当并发数较大时，需要创建大量线程来处理连接，系统资源占用较大。

3：连接建立后，如果当前线程暂时没有数据可读，则线程就阻塞在 Read 操作上，造成线程资源浪费。

NIO(Non Blocking IO):同步非阻塞IO，一个请求一个线程

1：请求会注册到多路复用器Selector上，多路复用器轮询到连接有IO请求时才启动一个线程处理。

2：Netty封装了JDK的NIO，

AIO(Asynchronous IO):异步非阻塞IO，一个有效请求一个线程

1：AIO使用回调的方式处理IO请求，在socket上注册一个回调函数，然后提交请求后直接返回。

# Netty组件Channel、EventLoop、Thread、EventLoopGroup

## Channel

为Netty 网络操作抽象类，可以看作是传入或传出数据的载体

## Future、ChannelFuture

在 Netty 中所有的 IO 操作都是异步的，不能立刻得知消息是否被正确处理。但是可以过一会等它执行完成或者直接注册一个监听，具体的实现就是通过 Future 和 ChannelFutures，他们可以注册一个监听，当操作执行成功或失败时监听会自动触发注册的监听事件。

## EventLoop

## Bootstrap、ServerBootstrap

Bootstrap 意思是引导，一个 Netty 应用通常由一个 Bootstrap 开始，主要作用是配置整个 Netty 程序，串联各个组件，Netty 中 Bootstrap 类是客户端程序的启动引导类，ServerBootstrap 是服务端启动引导类。

## ChannelHandler

充当了所有处理入站和出站数据的应用程序逻辑的容器。ChannelHandler 主要用来处理各种事件，这里的事件很广泛，比如可以是连接、数据接收、异常、数据转换等。

ChannelInboundHandler 用于接收、处理入站数据和事件，而 ChannelOutboundHandler 则相反。

事件可以被分发给ChannelHandler类中某个用户实现的方法。那么，如果 ChannelHandler 处理完成后不直接返回给客户端，而是传递给下一个ChannelHandler 继续处理呢？那么就要说到 ChannelPipeline ！

## ChannelPipeline

# netty细节点

## AttributeKey

AttributeMap这是是绑定在Channel或者ChannelHandlerContext上的一个附件，每一个ChannelHandler都能获取到

底层组成变量：private static final ConstantPool<AttributeKey<Object>> pool

ConstantPool就是维护ConcurrentHashMap constants。

重要方法：

valueOf(String name)：pool.valueOf(name):getOrCreate(name)

constants有则返回，否则新建再返回constants.putIfAbsent的值，值空则返回新建的，说明已经被新建加入了。否则返回已存在的值。谁先抢到返回谁。

## Promise接口

public interface Promise<V> extends Future<V>

相比future只能读取，promise可以进行write操作。

## Selector

使用NIO的时候会创建好大厅Selector。这是一个单线程不停的循环读取客户端发送过来的请求。然后将客户端的请求保存到SelectionKey里面。

# Netty

## 简介

Java NIO编程复杂度太高，而且容易出错，于是netty对NIO进行了改造和封装。形成了一个比较完整的网络框架，可以通过他实现rpc，http服务。

# Reactor

# 常见参数

## TCP\_NODELAY

关系nagle算法，即减少数据发送量，来增进TCP/IP的性能。使用此配置即禁止使用nagle算法，允许小包的发送，对于延时敏感性和数据传输量较小的应用必开。

## SO\_KEEPALIVE

## SO\_REUSEADDR