netty初探

1.netty是什么，有什么用

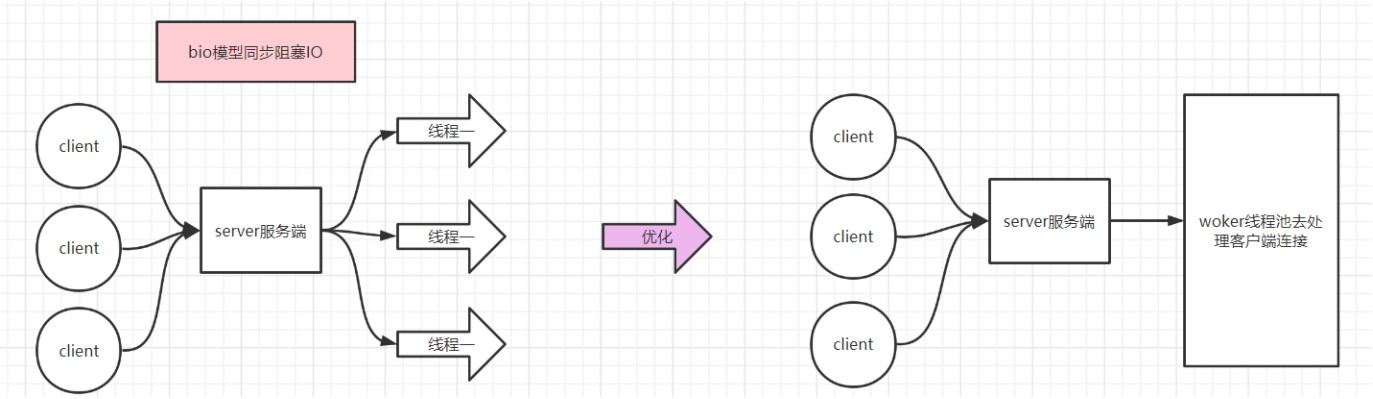
1. 概念：netty是一个**异步的**，**基于事件驱动的**，基于**TCP协议的**通信框架，本质是一个**NIO框架**。
2. 应用背景：我们分布式或者微服务中的大多数中间件都是基于netty通信的，非常火爆。举例：知名的dubboRPC通信框架，redis,nginx，nacos等中间件。

2.网络IO模型

- 分为三类 bio,nio,aio

2.1 Bio模型: 同步阻塞IO

1. 图解原理

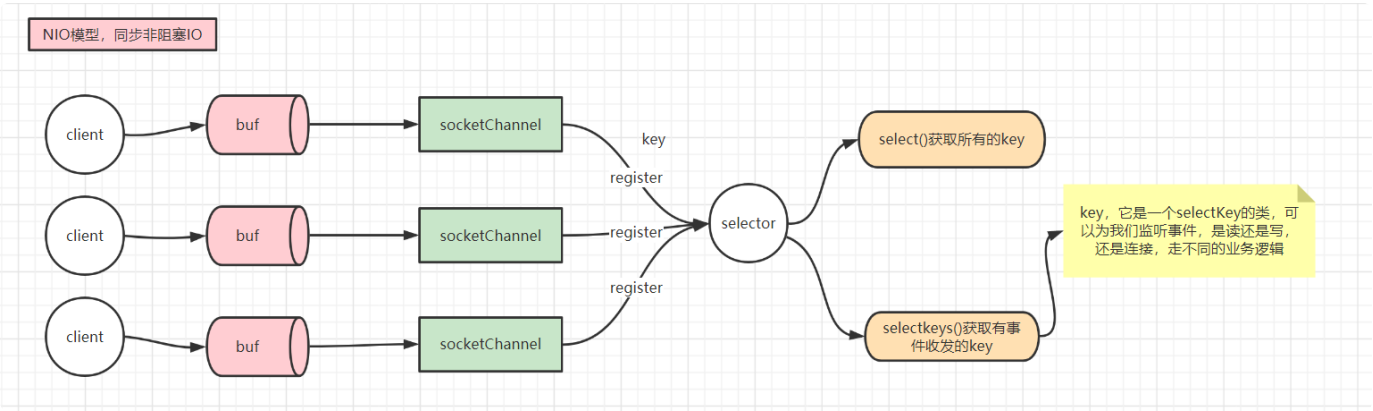


2. 理论分析

- 特点：客户端来一个线程，与服务端建立一个连接，生成socket,服务端创建一个线程去处理，从而支持很多客户端连接。缺点：一个客户端一个线程，可能由于线程过多导致我们cpu100%问题，也有可能oom，最大的缺点是阻塞io模型，客户端如果占用线程，一直不发送数据，形成资源浪费。
- 特点：对传统的BIO线程进行了优化，解决了频繁创建线程的问题。缺点：资源浪费并没有的得到解决，而且并发量也受到了限制。

2.2 NiO模型：同步非阻塞IO

1. 图解原理



2. 理论分析

- buf:是一个可读可写的缓冲区，底层是一个数组，有四个属性，pos，limit cap,标识，如果从读缓冲区换成写，改标识即可,客户端的读写事件都放在buf中，也可以是多级buf
- socketChannel:是由于客户端与服务端serversocketChannel建立连接所生成的套接字，可读可写，用于将buf缓冲区的读写事件注册到多路复用器中
- 多路复用器：selector 可以处理有事件收发的key,通过epoll事件轮询机制，实现的，注意poll,epoll,selectd的区别
- Nio特点：一个客户端 对应独立的buf,一个客户端对应一个socketChannel,一个selector对应多个客户端多个socketChannel。

2.3 Aio模型： 异步非阻塞IO

略：技术不是很成熟，使用范围不广，不做深究。

2.4 NIO中selector底层实现

回答这个问题：需要从三方面解释：内核空间，用户空间；fd文件描述符；select,epoll,poll的区别

1. 内核空间，用户空间：操作系统的空间一般分为内核空间和用户空间两部分，内核空间上运行这操作系统，用户空间运行的是用户程序进程和应用。用户空间不能直接操作硬件，需要借助内核空间操作硬件。
2. fd文件描述符：内核对磁盘文件的每一个读写操作都会生成一个文件描述符，用户空间操作硬件时可以将fd从内核态copy到用户态。
3. 区别：

- select：将内核中的文件描述符，以链表的存储方式从内核态copy到用户空间；
- poll: 将内核中的文件描述符，以红黑树的存储方式从内核态copy到用户空间；
- epoll: 通过事件轮询机制，监听有事件收发的文件描述符，是通过操作系统的内核函数实现的epoll_ctl(内核到用户的copy),epoll_create(监听有事件收发的fd)...

3. NIO中的零拷贝

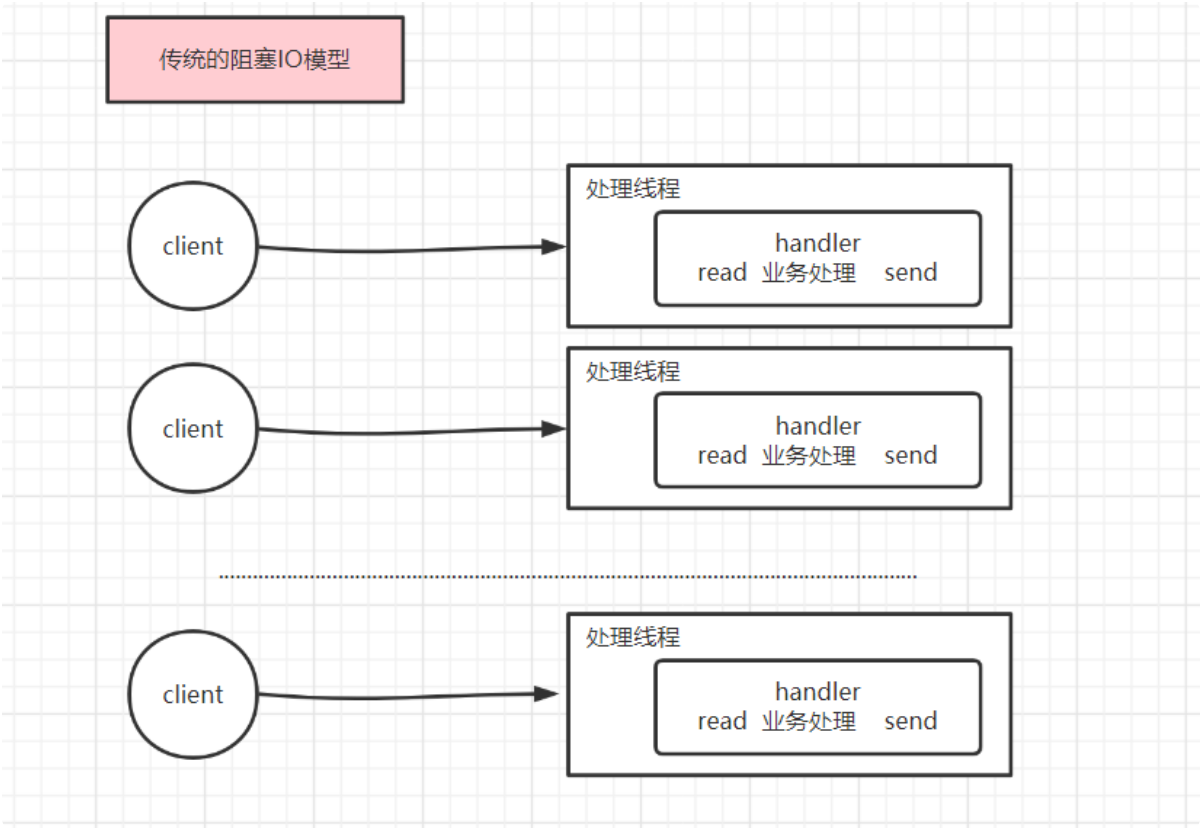
1. 零拷贝的概念：所谓的零拷贝并不是真正的无copy，而是没有cpu拷贝，DMAcopy是磁盘到内存的copy是不可能不读取磁盘数据的。
2. 传统的IO：**磁盘-------DMA---->内核-----cpu----->用户-------cpu------>socketbuffer-----DMA--------->协议栈**
3. mmap内存映射技术：将**内核的文件映射到了用户空间**，这段空间被称之为用户和内核的**共享空间，**是减少了一次cpu拷贝,还不是真正的零拷贝；磁盘-------DMA---->内核---->用户--------cpu---------------------->socketbuffer-----DMA--------->协议栈
4. sendFile优化：可以将数据直接copy到协议栈,**不在经过用户空间和socketBuffer,**磁盘-------DMA---->内核------DMA--------->协议栈
5. NIO中的transfer就可以将文件直接copy到通道实现了零拷贝。

4.netty的架构设计

- 从三方面回答：传统的阻塞IO模型，reactor模型，netty架构模型。

4.1 传统的阻塞IO模型

1. 图解原理：



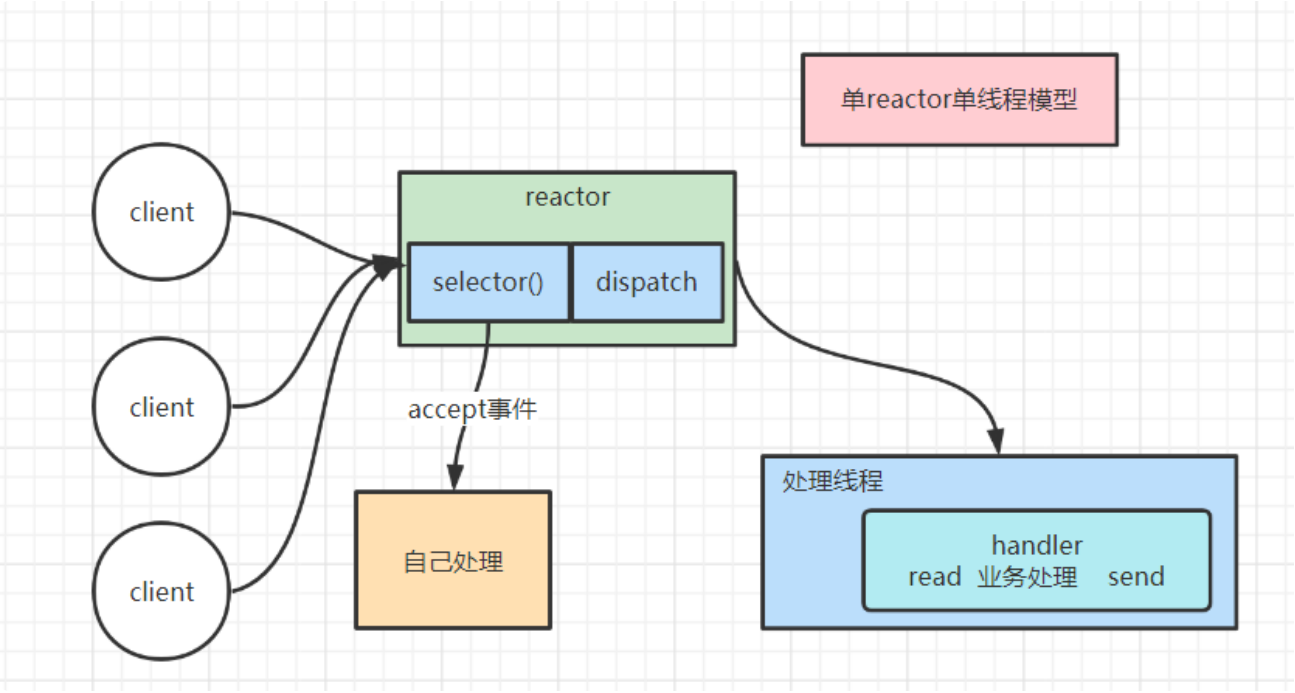
2. 理论分析

缺点：1.并发很大，cpu100%　2.线程阻塞，每个线程一直阻塞，资源浪费

4.2 reactor模型

1. 单reactor单线程模型

- 图解原理

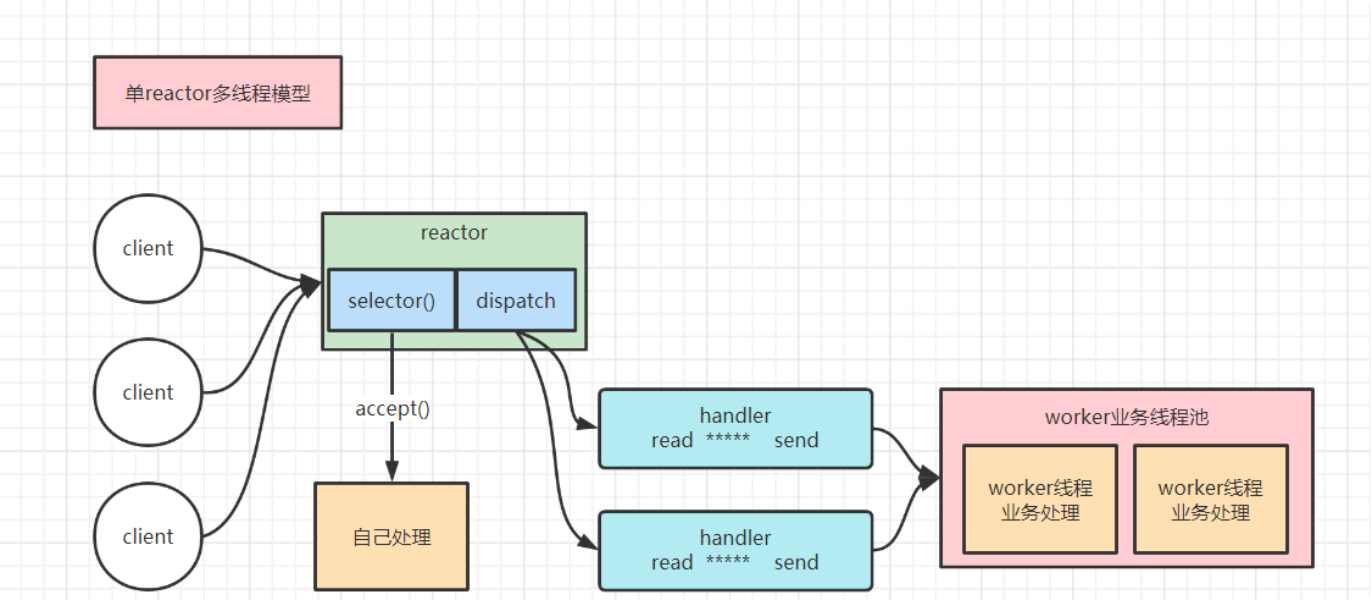


- 理论分析

优点：解决了阻塞和cpu100%,传统io上的问题，**缺点**：只有一个线程在处理，不能充分利用多核cpu，效率还有问题

2. 单reactor多线程模型

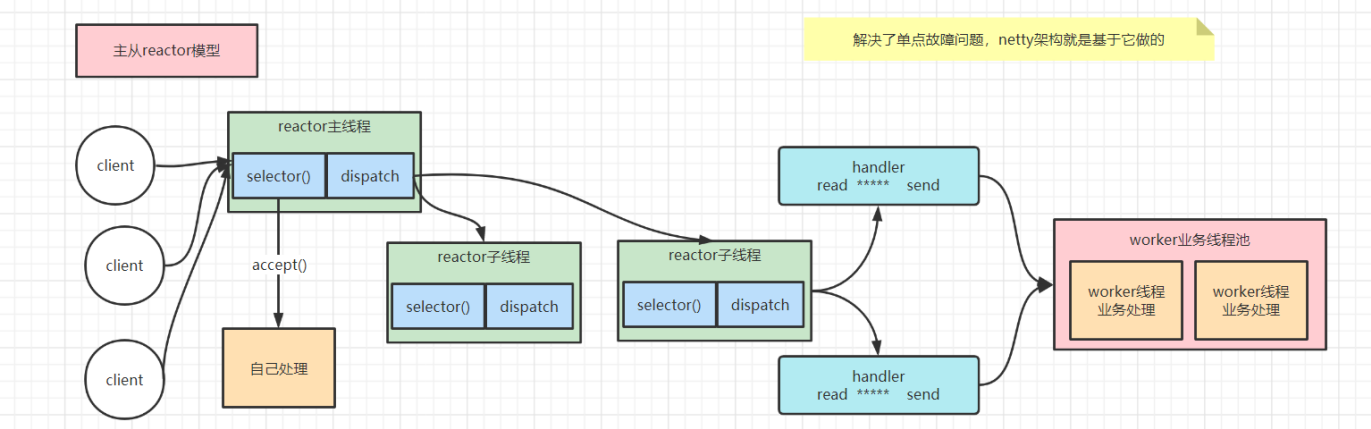
- 图解原理



- 理论分析：**优点**：解决了单reactor单线程的大量cpu空闲问题，**缺点**：多线程数据共享和访问比较复杂，reactor要处理所有的请求可能会出现单点故障的问题，redis是基于这种架构搭建的。

3.主从reactor模型

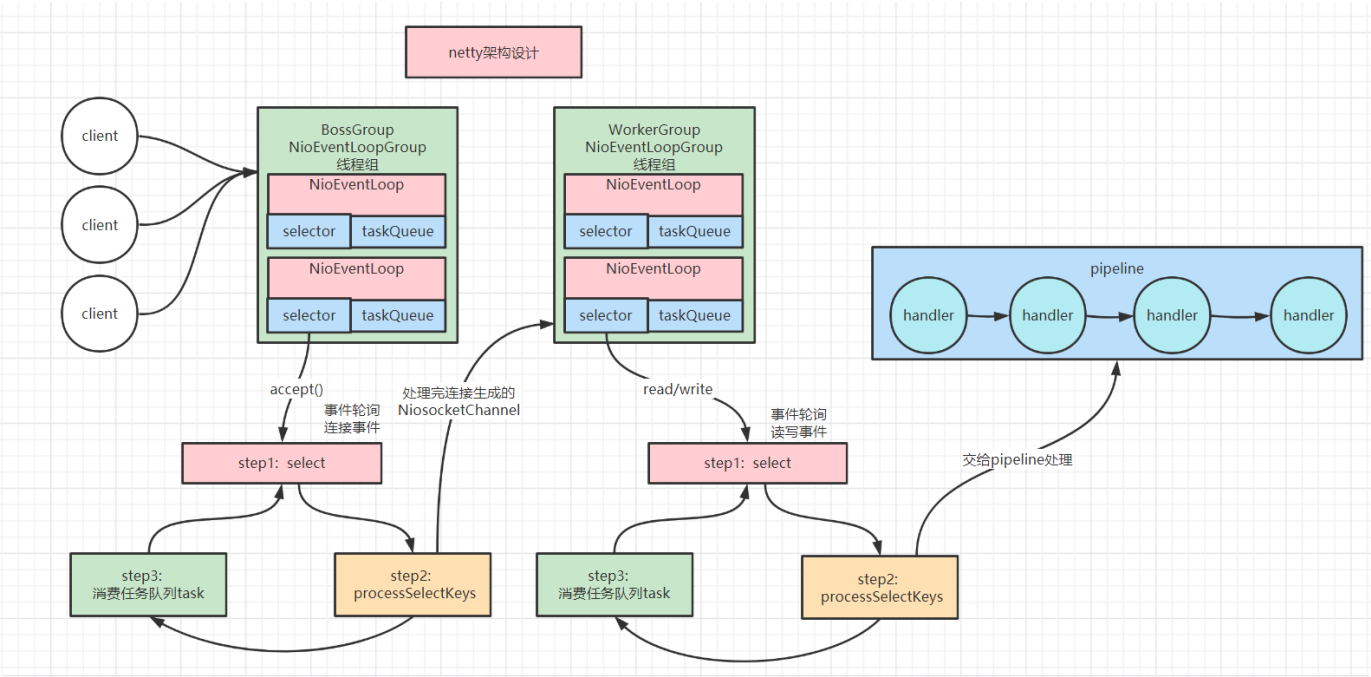
- 图解原理:



- 理论分析：解决了单点故障问题，netty架构就是基于它做的

4.3 netty架构模型

1. 图解原理:



2. 理论分析:

- netty模型是通过两个事件轮询组，分别叫bossGroup和WorkerGroup,实际都是NioEventLoopGroup线程组
- 客户端发送请求到bossGroup,bossGroup建立连接，交给一个NioEventLoop线程去处理，通过监听机制监听是不是连接事件，是，就加到taskQueue任务队列，让这个线程通过事件轮询这个任务队列，生成NioSocketChannel都注册到workerGroup上
- workerGroup同样也通过轮询机制轮询读写事件，将每一个连接事件交给pipeline管道中的handler处理链条去处理，所以tomcat关注servlet,而我们netty关注handler。

5.netty常识小知识

5.1 ctx ，通道，管道的区别

1. ctx 可以获取 当前通道channel;也可以获取handler处理，和pipeline
2. channel 你中有我，我中有你 pipeline
3. BootStrap和ServerBootStrap: group(); channel(); bind(); connect(); option(); childOption();--->处理handler();
4. selector事件驱动的 pipeline管道 里面是双向链表维护的handler

5.2 netty异步模型

1. netty的任务队列

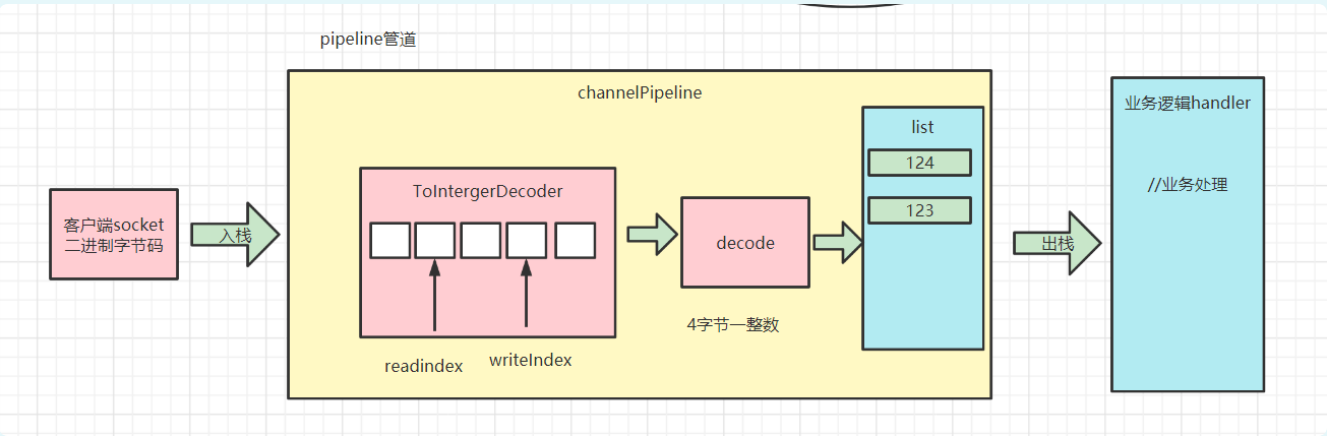
- 服务端有一个非常耗时的业务，---->异步执行-------->提交该channel对应的NioEventLoop的taskQueue中
- 推送业务

2. channelFuture异步I/O监听:

- 表示异步执行的结果，可以通过它提供的方法检测是否执行完成，channelFuture可添加监听器，事件发生，通知监听器。

6. netty的出栈和入栈

- 出栈：从channelPipeline 出来的叫出栈
- 入栈：进入channelPipeline的叫入栈



7.tcp的粘包和拆包问题

1. 问题描述：tcp是一个流协议，没有界限的一串数据，我们在传输的过程中会出现，正常传输，小包合成大包，大包拆成小包等问题，由于我们也不知道这一次请求何时结束，所以数据可能在一块，可能分散开
2. 如何解决：

- 消息长度固定；意思就是客户端和服务端约定好，我的每一个包都是固定多长，当服务端读到该长度时，就取出一个数据包。
- 将回车换行符作为消息结束符，就是说我们tcp发送一次请求后，在这个数据包后面加个回车换行符，如果服务端读到这个符号，认为它是一个数据包。
- 将特殊的分隔符作为消息结束标志，回车换行是特殊的一种。
- 通过在消息头中定义长度字段，来表示数据包的总长度。

3. netty为我们提供了几种编解码器

- lineBasedFramedDeceder：回车换行符那种
- DelimiterBasedFrameDeceder：分隔符做结束标志消息解码
- FixedLengthFrameDeceder：消息头指定固定长度的编解码器