方式，这些线程被称为 IO 线程。由于 NIO 下的 Selector 可以同时处理连接、读取和写入，因此 NIO 实际上可以使用固定数量的线程来同时处理超过线程数量的连接，这也是 NIO 优于传统阻塞式 IO 的地方。

处理模型的变化，特别是从单线程改成多线程，会对处理逻辑有很大的影响，而且 JDK 的 NIO 相对低层，如果想要编写相对安全、完整、健壮的 NIO 网络程序，建议还是使用封装好的针对 NIO 的网络库，比如 Netty。

简单来说，Netty 是一个在 JDK 所提供的 NIO 的 API 上隐藏了复杂性，特别是 NIO 的编程复杂性的网络框架。上述单线程模型与多线程模型通过简单的配置就可以快速切换，不会对处理框架有太大影响。

Netty 同样有一个 Channel 模型，而且可以直接调用 Channel 实例的 write 方法写入，不用担心线程安全问题。

Netty 特有的一个设计是 Pipeline，如图 6-6 所示。如果有 Web 开发经验，那么可以将其理解为 Servlet 中的 Filter。Pipeline 主要分为 Inbound Processor 和 Outbound Processor，请求通过入口处理器（Inbound Processor）的处理链，然后再通过出口处理器（Outbound Processor）的处理链。这样的好处是可以把前置、后置、协议之类的处理分离，简化为主要处理逻辑。

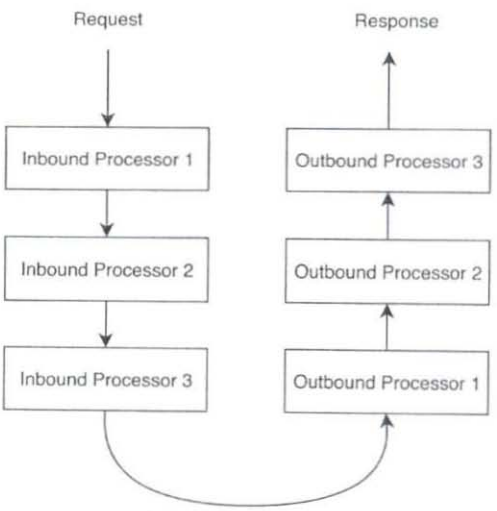

图 6-6　Pipieline 设计

例如，网络程序中的 SSL 加密、模型序列化和反序列化、安全校验等处理，通过把常见前置和后置逻辑从网络程序中抽出来并模块化，主体业务逻辑就会变得更加简单和清晰。

在 Pipeline 中，处理链在每次创建新的 Channel 时会实例化一个新的处理链（准确来说是调用 ChannelInitializer 的 initChannel 方法），间接为每个 Channel 创建了单独的入口处理器和出口处理器。这样即使针对每个 Channel 在入口处理器或出口处理器中保存了可变的数据，在 Netty 的处理链设计中也不会有线程安全问题。