**为什么需要Netty？**

1、精细分工：如HTTP协议不适合用于传输大文件，不适合用于传输Email等，此时我们需要的是**一个专门针对此种应用设计的高度优化的协议**。

2、处理遗留系统中的旧协议。

**Netty的目标和特点**

基于**事件驱动的异步**(asynchronous event-driven)网络应用框架，可维护的高性能、高可伸缩性协议下的服务端、客户端开发工具。

基于NIO模块，从FTP、SMTP、HTTP和大量二进制、文本协议中成功找到一条轻松开发之路，在性能、稳定性、灵活性上不做妥协。

**包、接口和类**

Bootstrap：启动服务端、客户端Channel的帮助类。

ByteBuf(C)：带缓冲的消息载体。ByteBufAllocator(I)分配ByteBuf。

ByteBufInputStream(C)：从ByteBuf以流的方式读取数据。

ByteBufUtil (C)：操作ByteBuf的工具类。

尽量从Allocator去获取ByteBuf，再使用ByteBuf的相关方法获取子类。

Channel(I)：通信通道，拥有众多子类。

**Java NIO**

Buffer：信息载体，其中最重要的是ByteBuffer。

Channel：分为两种，file和socket，具体是FileChannel、SocketChannel、ServerSocketChannel和DatagramChannel。FileChannel只能从RandomAccessFile获取，Socket类型的Channel可以通过工厂方法获取。只有Sockets和Pipes可以使用非阻塞模式。

ServerSocketChannel没有实现读写接口，因为它只负责监听传入的连接和创建新的SocketChannel，本身并不传输数据。

SocketChannel、ServerSocketChannel和DatagramChannel都会创建对等的Socket。

SelectableChannel为Socket通道提供了可设置为非阻塞模式的选项。

SocketChannel基于TCP/IP，面向流；DatagramChannel基于UDP，面向包。

Pipe(管道)：进程内使用，连接一个SourceChannel和一个SinkChannel。主要用于配合Selector使用以及辅助测试(一头输入一头输出)。

Channels是工具类，提供Channel到InputStream/OutputStream、Reader/Writer之间的转换。其中，转换到Reader/Writer时，因为涉及到字节到字符的转换，需要字符集支持。

**Selector(选择器)**：提供选择执行已经就绪的任务的能力，使得多元I/O成为可能。如果没有选择器，我们检查通道是否就绪的方式比较有限，比如轮询。选择器从操作系统层面来检查I/O请求是否已准备好。

相关的类有这么几个。Selector：管理着一个通道集合的信息和它们的就绪状态，通道需要注册到Selector中。SelectableChannel：只有从该类继承的类才能注册到Selector中，因为它们都从该类继承了支持就绪检查的方法。FileChannel不支持，所有SocketChannel都支持，包括从Pipe获得的Channel。SelectionKey：封装了通道与选择器之间的注册关系。

IP v4地址：127.0.0.1回环，自动配置：169.254，多播地址：第一个数字在224~239之间。

keep-alive：经过一段不活动时间向另一端发送一个探测消息，如果反复测试几次都没有回复则认为连接已经断掉，关闭套接字。该功能默认是关闭的。

ByteBuffer的三个方法：flip，将ByteBuffer设置为准备读的状态，做法是将limit改为当前position的值，将position的值改为0；rewind，将ByteBuffer再次置为读，它的做法是将position置为0，limit不变，mark无效；clear，将position置为0，并将limit置为capacity。