# Netty是什么？为什么要用Netty？

## 为什么要用Netty

1、虽然JAVA NIO框架提供了 多路复用IO的支持，但是并没有提供上层“信息格式”的良好封装。例如前两者并没有提供针对 Protocol Buffer、JSON这些信息格式的封装，但是Netty框架提供了这些数据格式封装（基于责任链模式的编码和解码功能）；

2、直接使用NIO需要需要额外的技能，例如Java多线程，网络编程；

3、要编写一个可靠的、易维护的、高性能的NIO服务器应用。除了框架本身要兼容实现各类操作系统的实现外。更重要的是它应该还要处理很多上层特有服务，例如：客户端的权限、还有上面提到的信息格式封装、简单的数据读取，断连重连，半包读写，心跳等等，这些Netty框架都提供了响应的支持。

4、JAVA NIO框架存在一个poll/epoll bug：Selector doesn’t block on Selector.select(timeout)，不能block意味着CPU的使用率会变成100%（这是底层JNI的问题，上层要处理这个异常实际上也好办）。当然这个bug只有在Linux内核上才能重现。

这个问题在JDK 1.7版本中还没有被完全解决，但是Netty已经将这个bug进行了处理。

这个Bug与操作系统机制有关系的，JDK虽然仅仅是一个兼容各个操作系统平台的软件，但在JDK5和JDK6最初的版本中（严格意义上来将，JDK部分版本都是），这个问题并没有解决，而将这个帽子抛给了操作系统方，这也就是这个bug最终一直到2013年才最终修复的原因(JDK7和JDK8之间)。

## 为什么不用Netty5

1. netty5 中使用了 ForkJoinPool，增加了代码的复杂度，但是对性能的改善却不明显

2. 多个分支的代码同步工作量很大

3. 作者觉得当下还不到发布一个新版本的时候

4. 在发布版本之前，还有更多问题需要调查一下，比如是否应该废弃 exceptionCaught， 是否暴露EventExecutorChooser等等。

## 为什么Netty使用NIO而不是AIO？

Netty不看重Windows上的使用，在Linux系统上，AIO的底层实现仍使用EPOLL，没有很好实现AIO，因此在性能上没有明显的优势，而且被JDK封装了一层不容易深度优化。

AIO还有个缺点是接收数据需要预先分配缓存, 而不是NIO那种需要接收时才需要分配缓存, 所以对连接数量非常大但流量小的情况, 内存浪费很多。

据说Linux上AIO不够成熟，处理回调结果速度跟不上处理需求，有点像外卖员太少，顾客太多，供不应求，造成处理速度有瓶颈。

作者原话：

Not faster than NIO (epoll) on unix systems (which is true)

There is no daragram suppport

Unnecessary threading model (too much abstraction without usage)

# 第一个Netty程序

## Channel

## 回调和Future

## 事件和ChannelHandler

# Netty组件再了解

## Channel、EventLoop 和ChannelFuture

#### Channel 接口

#### EventLoop 接口

#### ChannelFuture 接口

## ChannelHandler 和ChannelPipeline

#### ChannelHandler 接口

#### ChannelPipeline 接口

## 选择合适的内置通信传输模式

## 引导

## ByteBuf

#### 分配

#### 随机访问索引/顺序访问索引/读写操作

# 解决粘包/半包问题

## 回顾我们的的Hello,Netty

# 什么是TCP粘包半包？

## TCP粘包/半包发生的原因

## 解决粘包半包问题

# 编解码器框架

## 什么是编解码器

## 解码器



#### 将字节解码为消息

#### 将一种消息类型解码为另一种

#### TooLongFrameException

## 编码器

#### 将消息编码为字节

#### 将消息编码为消息

## 编解码器类

#### CombinedChannelDuplexHandler 类

## Netty内置的编解码器和ChannelHandler

#### 通过SSL/TLS 保护Netty 应用程序

#### HTTP 解码器、编码器和编解码器

#### WebSocket

#### 空闲的连接和超时

#### 写大型数据

# 序列化问题

## Java序列化的缺点

## 序列化 – 内置和第三方的MessagePack实战

# 如何进行单元测试

## 测试入站消息

## 测试出站消息

## 测试异常处理