# IO 与NIO

NIO是在jdk1.4才引入的，

1. NIO主要用到的是块，所以NIO的效率比IO高很多。
2. Java API提供了两套NIO，一套针对标准输入输出流，一套就是网络编程NIO。
3. IO是面向流的，NIO是面向缓冲区的。
4. Java的IO是阻塞的，当一个线程调用read()或者write()方法时，该线程被阻塞。
5. NIO是非阻塞的，他仅能得到目前可用的数据，如果目前没有可用的数据时，就什么都不会获取，直到数据变的可以读取之前，该线程可以继续做其他的事情。一个线程请求写入一些数据到某通道，但不需要等待他完全写入，这个线程同时可以去做别的事情。所以一个单独的线程可以管理多个输入输出通道。

# NIO

非阻塞

## Channel---通道

和Stream是一个等级的，但是Stream是单向的，Channel是双向的。

FileChannel

DatagramChannel

SocketChannel

ServerSocketChannel

## Buffer---缓冲区

NIO中的关键Buffer实现有：ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer，分别对应基本数据类型: byte, char, double, float, int, long, short。当然NIO中还有MappedByteBuffer, HeapByteBuffer, DirectByteBuffer等这里先不进行陈述。

## Selector---选择器

Selector运行单线程处理多个Channel，如果你的应用打开了多个通道，但每个连接的流量都很低，使用Selector就会很方便。例如在一个聊天服务器中。要使用Selector, 得向Selector注册Channel，然后调用它的select()方法。这个方法会一直阻塞到某个注册的通道有事件就绪。一旦这个方法返回，线程就可以处理这些事件，事件的例子有如新的连接进来、数据接收等。

NIO的核心类库多路复用器Selector就是基于epoll的多路复用技术实现的。

当同时处理多个请求时，可以用多线程或者I/O的多路复用技术进行实现。

## 数据读取

向Buffer中写数据：

从Channel写到Buffer (fileChannel.read(buf))

通过Buffer的put()方法 （buf.put(…)）

从Buffer中读取数据：

从Buffer读取到Channel (channel.write(buf))

使用get()方法从Buffer中读取数据 （buf.get()）

# BIO

同步阻塞式IO，BIO。

# AIO

基于NIO2.0的异步非阻塞。

# Socket与ServerSocket

Socket：使一个应用从网络读取和写入数据，不同计算机上的两个应用可以通过连接发送和接受字节流。是一个进行通信的类。客户端用Socket进行创建。

ServerSocket：是一个服务端，等待连接。一旦连接成功，就会调用accept方法创建一个Socket与客户端进行通信。如果多个客户端同时请求，如果一个一个顺序响应肯定会有不好的体验。所以使用多线程来为多个客户端提供服务。

# Netty

Netty是一个NIO框架，就像Spring是java的一个框架