# 一．与IO区别

1. 组件

*\* NIO中  
\* 核心概念：  
\* selector：选择器  
\* channel：通道  
\* buffer：缓冲区  
\* 即面向块儿 （block）或者缓冲区buffer 编程的*

NIO中面向块儿（ block、 buffer ）编程，

*Buffer本身就是一内存，实际上是数组，数据的读写都是通过Buffer来实现的。*

Channel 就像IO中的stream

1数据要从channel 进入到buffer 中，有buffer进入到程序中。也可以将数据写回到buffer中。即通过buffer.flip().

读写的切换就是由flip 切换的。Flip中维护了一些数位

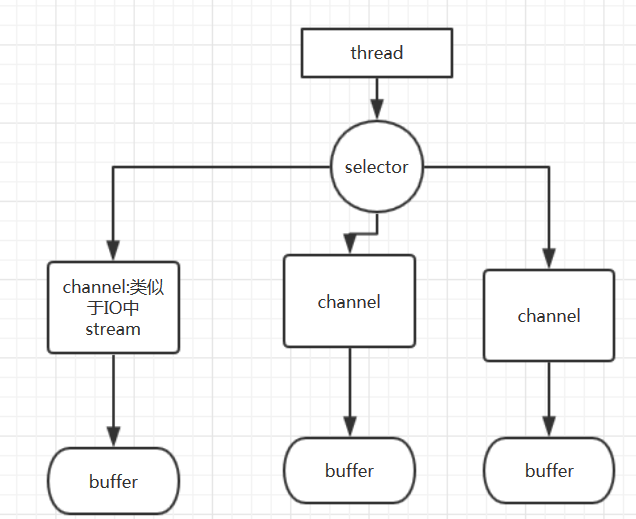
2. 除了数组之外，buffer还提供了对于数据的结构化方式，即读写的位置，

3. java的8中原生类型都有各自对应的buffer类型：但没有BooleanBuffer

IntBuffer，LongBuffer ，ByteBuffer ，CharBuffer

4. channel 是指向其写入或读取数据的对象，类似于IO中的stream 。其所有的数据的读写都要通过buffer操作。

5. channel是双向的，channel 创建后可以进行读或写 。在Linux系统中，底层操作系统的通道就是双向的，



# 二．buffer 属性讲解

1. position ： 读，写元素的下一个位置的索引值

当buffer中读取元素调用flip（）之后，positon重新指向写元素的第一个位置

Positon 永远不会大于limit 的值

2. limit ：

3.capacity ：容量的值大小不会变

# 三．byteBuffer

HeapByteBuffer extend ByteBuffer

# 四．DirectByteBuffer

## 1.

# 五．NIO – selector选择器

1. 传统的网络编程：构造ServerSocket ,将其绑定到某端口上，

ServerSocket会调用accept方法，阻塞等待客户端连接：即服务端的代码就等待阻塞，当有链接时就返回一个Socket对象，标示跟客户端的连接，

**客户端与服务端进行连接的端口就是服务端开启的端口，但两者进行数据传递的端口是由操作系统另外分配的。**

2. 异步处理客户端的请求（ NIO ）

Selector 可以监听客户端的事件event ，注册通道是通过selectionKey

Selector 通过三种selectionKey 维护IO事件的状态，可得到是哪个channel产生的selectionKey

2.1 select 方法，阻塞方法：当其连接的通道有任何变更时，都会通知相应的通道，返回相应的集合

3.

*The first invocation of this method locates the default provider  
\* object as follows*

第一次调用这个方法定位默认程序 。

SelectionKey 在使用完后一定要关闭。

# 六 . 字符集处理

# 七．高性能—Reactor 模式（重点）

1. Reactor：反应器模式（ Scalable IO in java : doug Lea 主导java并发编程的作者 ）

Nio中的EventLoop就是一个reactor 。

* 1. 可伸缩的网络服务
  2. 事件驱动处理

1.3 多线程版本

1.4 netty服务器启动的流程：

Server端：

将注册器注册到Iniitionion Dispatcher中

调用OS层面的selector方法阻塞

1.5 channelPipeline ：管道对象（channel处理器的集合 ，封装了channelhandler集合）：当客户端请求时，会分别请求其中的处理器

2. netty 对reactor的基本实现：

2.1 流程

多个客户端连接 server端的Boss 的selector

Server端还有一个worker group ,里面也有一个selector

2.2

可以禁用堆外内存的申请

PlatformDependent类---

Netty分配堆外内存的类：UnpooledDirectByteBuf

# 八．Proactor 模式