NIO是什么，为什么要引入？

Java1.4引入

NIO的优点

**NIO与标准IO比较**

1. 标准IO API基于字节流和字符流；NIO基于channels（通道）和缓冲区（buffers）。数据总是从通道读入缓冲区，或从缓冲区写入通道。
2. 标准IO是阻塞IO；NIO是非阻塞IO，例如，线程通过channel把数据读入buffer时，线程可以执行其他操作，等数据读入buffer后，线程可以继续处理。写数据也一样。
3. NIO包含selectors（选择器）的概念。选择器是可以监视多个通道的事件的对象（如：打开连接，数据到达等等）。因此，单线程可以监视多个通道的数据。

**Channels**

1. Data从Channel读入到Buffer。

2.主要的Channel实现类：覆盖了UDP/TCP网络IO和文件IO

FileChannel：reads data from and to files.

DatagramChannel：read and write data over the network via UDP.

SocketChannel：read and write data over the network via TCP.

ServerSocketChannel：监听过来的TCP连接，类似Web Server。每过来一个TCP连接创建一个SocketChannel。

3.通道与流的区别：

能够同时对channel进行读和写，只能对流进行读或写。

能够对channel进行异步读和写。

channel总是把数据读入buffer或者从buffer写入数据。

**Buffers**

Data从Buffer写入到Channel。

主要的Buffer实现类：覆盖了可以通过IO发送的基本数据类型

ByteBuffer

MappedByteBuffer

CharBuffer

ShortBuffer

IntBuffer

LongBuffer

FloatBuffer

DoubleBuffer

Buffer本质上是一个可以写入数据的内存块，然后可以再次读取。该内存块被包装在一个NIO Buffer对象中，提供了一组方法更容易操作内存块。

使用Buffer读和写数据的基本步骤：

1.Write data into the Buffer

2.Call buffer.flip()，写完数据后，把buffer的写模式翻转为读模式

3.Read data out of the Buffer

4.Call buffer.clear() or buffer.compact()，clear()清空整个buffer，compact()只清除已读的数据，未读数据会移动到buffer开始处，后来的数据被写入未读数据后面。

Buffer Capacity, Position and Limit：



**Capacity**：作为一个内存块，一个buffer具有固定大小，称为容量。写入的数据量不能操过容量。

**Position**：当向buffer写入数据时，是写入position指向的单元。position的初始值为0，当向buffer中写入一个单位数据后，position自动指向下一个单元，position的最大值为capacity-1。

当从buffer读取数据时，也是读取position指向的单元。当通过flip()方法让buffer从写模式切换到读模式，position被重置为0，读一个单位数据，position自动指向下一个单元。

**Limit：**写模式下的limit是能够向buffer中写入的数据的最大数量，其值与capacity相等。

当通过flip()方法让buffer从写模式切换到读模式，limit的值为能够读取的数据的最大数量，因此，其值被设为写模式下的position的值。

**写数据到buffer**：

方式1.channel->buffer

方式2.buffer.put(x)及其系列重载方法

**flip()**：将buffer从写模式切换为读模式，limit设为position，position设为0。

**从buffer读取数据：**

方式1.buffer->channel

方式2.buffer.get()及其系列重载方法

**rewind()：**position设为0，limit的值不变，能够重新读一次数据。

**clear()：**使buffer可写入，position设为0，limit设为capacity。Buffer里的数据并没有被清空，只是标识位mark被清除，即使buffer中有未读数据，也法无找到。

**compact()：**使buffer可写入，如果buffer中有未读数据，会把这些数据复制到buffer的开始处，position指向最后一个未读数据的后面，limit设为capacity。因此后续的写入不会覆盖上次的未读数据。

**mark() and reset()：**

mark()标记一个给定的position，reset()将当前position重置回标记的position。

**equals():**

元素类型相同（byte,char...）并且remaining的数量相等，并且remaining序列中对应元素全部相等才返回true。

**compareTo():**

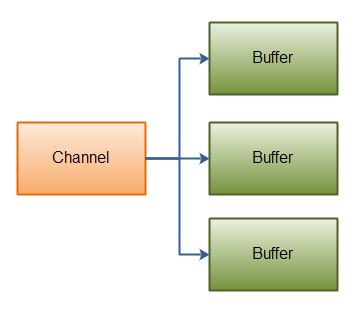
元素类型相同才能比较。从remaining序列中第一个元素开始按字典顺序比较，前者大返回正数，前者小返回负数，一样大继续比较下一个元素，都一样比较remaining序列的长度，长度大的返回正数，长度小的返回负数，长度相等返回0。

**Selectors**

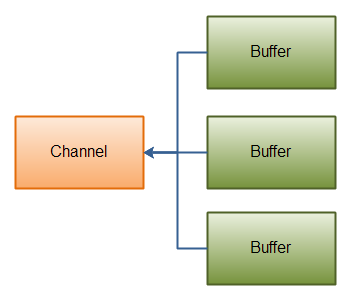
允许单个线程监视多个channel。

**Scatter（分散）和Gather（聚集）**

Scattering Reads:从一个channel读取数据到多个buffer。



Gathering Writes:从多个buffer写入数据到单个channel。



**Channel与Channel之间传递数据：**