http://wiki.jikexueyuan.com/project/java-nio-zh/java-nio-vs-io.html

当学习Java的NIO和IO时，有个问题会跳入脑海当中：什么时候该用IO，什么时候用NIO?

下面的章节中笔者会试着分享一些线索，包括两者之间的区别，使用场景以及他们是如何影响代码设计的。

NIO和IO之间的主要差异（Mian Differences Between Java NIO and IO）

下面这个表格概括了NIO和IO的主要差异。我们会针对每个差异进行解释。

IO	**NIO**
Stream oriented	Buffer oriented
Blocking IO	No blocking IO
	Selectors

面向流和面向缓冲区比较(Stream Oriented vs. Buffer Oriented)

第一个重大差异是IO是面向流的，而NIO是面向缓存区的。这句话是什么意思呢?

Java IO面向流意思是我们每次从流当中读取一个或多个字节。怎么处理读取到的字节是我们自己的事情。他们不会再任何地方缓存。再有就是我们不能在流数据中向前后移动。如果需要向前后移动读取位置，那么我们需要首先为它创建一个缓存区。

Java NIO是面向缓冲区的，这有些细微差异。数据是被读取到缓存当中以便后续加工。我们可以在缓存中向向后移动。这个特性给我们处理数据提供了更大的弹性空间。当然我们仍然需要在使用数据前检查缓存中是否包含我们需要的所有数据。另外需要确保在往缓存中写入数据时避免覆盖了已经写入但是还未被处理的数据。

阻塞和非阻塞IO比较（Blocking vs. No-blocking IO）

Java IO的各种流都是阻塞的。这意味着一个线程一旦调用了read(),write()方法，那么该线程就被阻塞住了，知道读取到数据或者数据完整写入了。在此期间线程不能做其他任何事情。

Java NIO的非阻塞模式使得线程可以通过channel来读数据，并且是返回当前已有的数据，或者什么都不返回如果当前没有数据可读的话。这样一来线程不会被阻塞住，它可以继续向下执行。

通常线程在调用非阻塞操作后，会通知处理其他channel上的IO操作。因此一个线程可以管理多个channel的输入输出。

Selectors

Java NIO的selector允许一个单一线程监听多个channel输入。我们可以注册多个channel到selector上，然后然后用一个线程来挑出一个处于可读或者可写状态的channel。selector机制使得单线程管理过个channel变得容易。

NIO和IO是如何影响程序设计的（How NIO and IO Influences Application Design）

开发中选择NIO或者IO会在多方面影响程序设计：

1. 使用NIO、IO的API调用类
2. 数据处理
3. 处理数据需要的线程数

API调用(The API Calls)

显而易见使用NIO的API接口和使用IO时是不同的。不同于直接冲InputStream读取字节，我们的数据需要先写入到buffer中，然后再从buffer中处理它们。

数据处理（The Processing of Data）

数据的处理方式也随着是NIO或IO而异。 在IO设计中，我们从InputStream或者Reader中读取字节。假设我们现在需要处理一个按行排列的文本数据，如下：

```
Name: Anna
Age: 25
Email: anna@mailserver.com
Phone: 1234567890
```

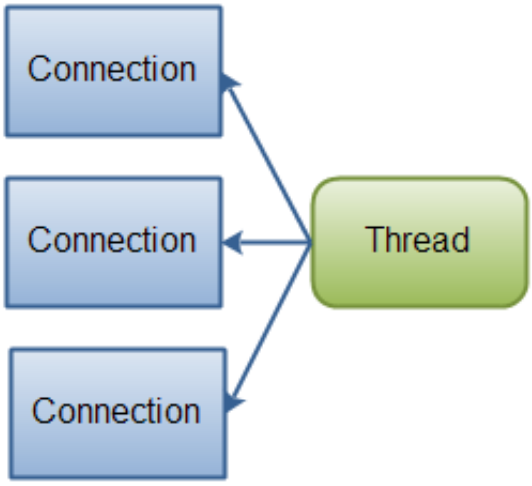
这个处理文本行的过程大概是这样的：

```
InputStream input = ... ; // get the InputStream from the client
socket
BufferedReader reader = new BufferedReader(new
InputStreamReader(input));
String nameLine   = reader.readLine();
String ageLine    = reader.readLine();
String emailLine  = reader.readLine();
String phoneLine  = reader.readLine();
```

小结

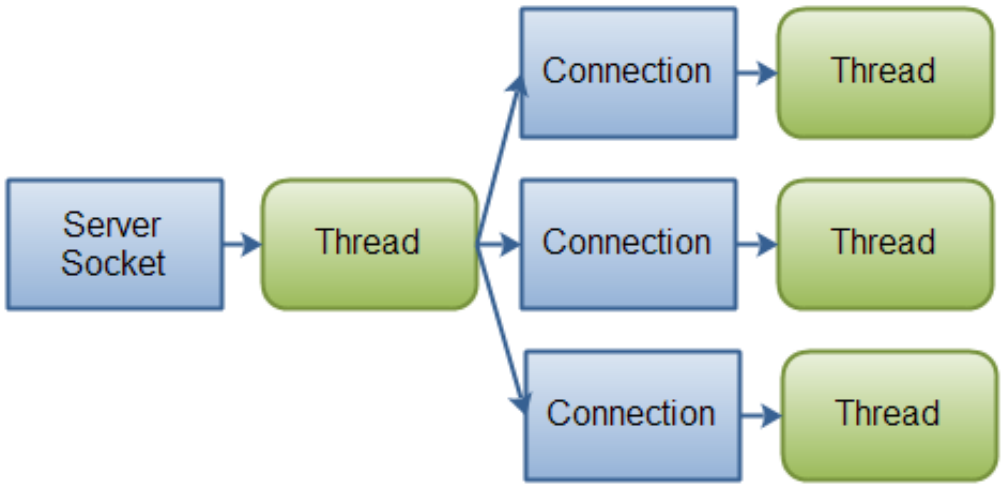
NIO允许我们只用一条线程来管理多个通道（网络连接或文件），随之而来的代价是解析数据相对于阻塞流来说可能会变得更加的复杂。

如果你需要同时管理成千上万的链接，这些链接只发送少量数据，例如聊天服务器，用NIO来实现这个服务器是有优势的。类似的，如果你需要维持大量的链接，例如P2P网络，用单线程来管理这些 链接也是有优势的。这种单线程多连接的设计可以用下图描述：



Java NIO: A single thread managing multiple connections

如果链接数不是很多，但是每个链接的占用较大带宽，每次都要发送大量数据，那么使用传统的IO设计服务器可能是最好的选择。下面是经典IO服务设计图：



Java IO: A classic IO server design - one connection handled by

one thread.