START

1. NIO和IO的对比



**面向流和面向缓冲**

 Java IO面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。

Java NIO的缓冲导向方法略有不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。但是，还需要检查是否该缓冲区中包含所有您需要处理的数据。而且，需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数据。

**阻塞与非阻塞IO**

当一个线程调用read() 或 write()时，该线程被阻塞，直到有一些数据被读取，或数据完全写入。该线程在此期间不能再干任何事情了。

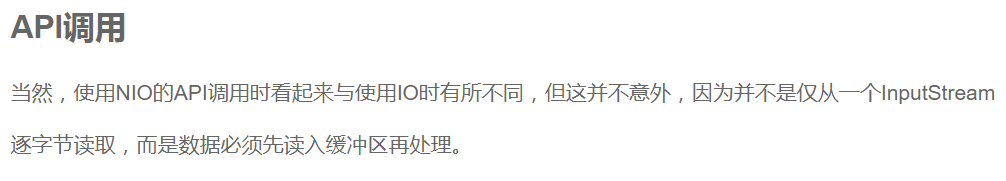
非阻塞：使一个线程从某通道发送请求读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用时，就什么都不会获取。而不是保持线程阻塞，所以直至数据变的可以读取之前，该线程可以继续做其他的事情。

线程通常将非阻塞IO的空闲时间用于在其它通道上执行IO操作，所以一个单独的线程现在可以管理多个输入和输出通道（channel）。也就是我们接下来要讲的selector

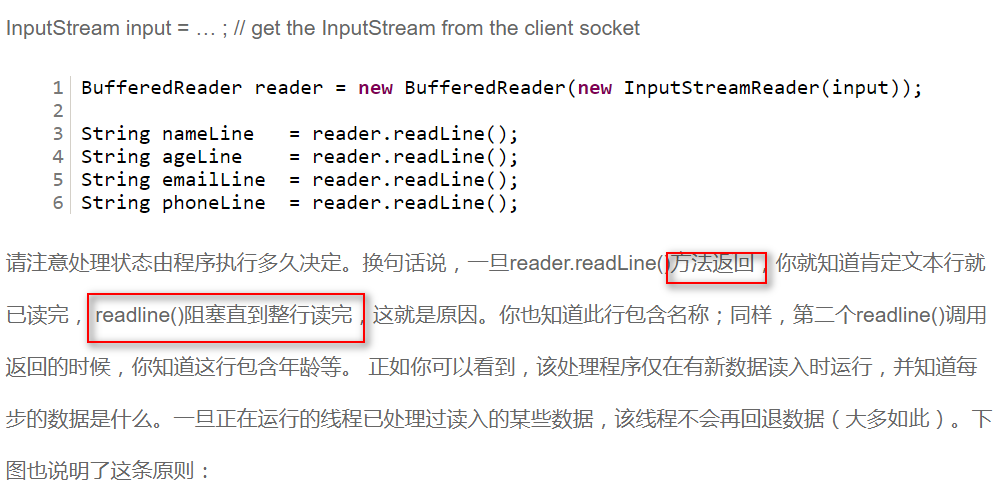
**SELECTOR**

Java NIO的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。

1. 程序设计实现
2. 对NIO或IO类的API调用。
3. 数据处理。
4. 用来处理数据的线程数。

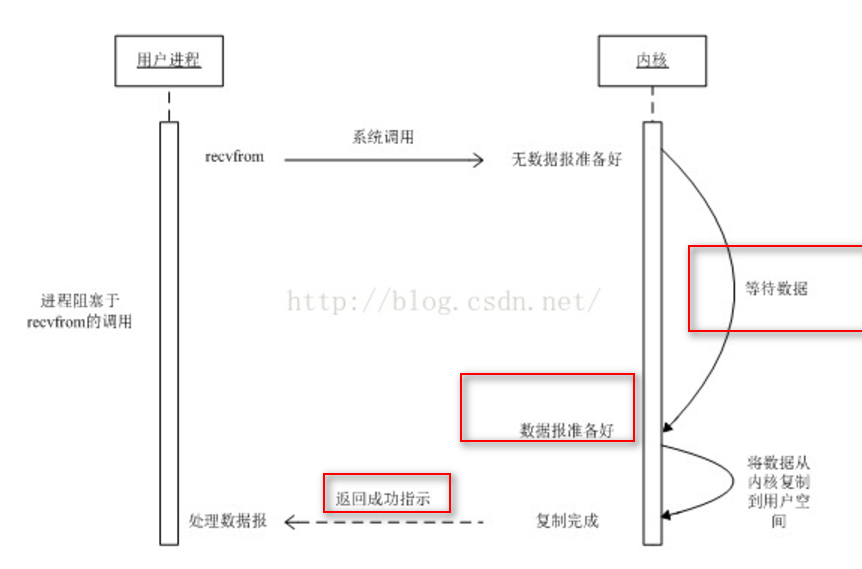


IO对于流的处理

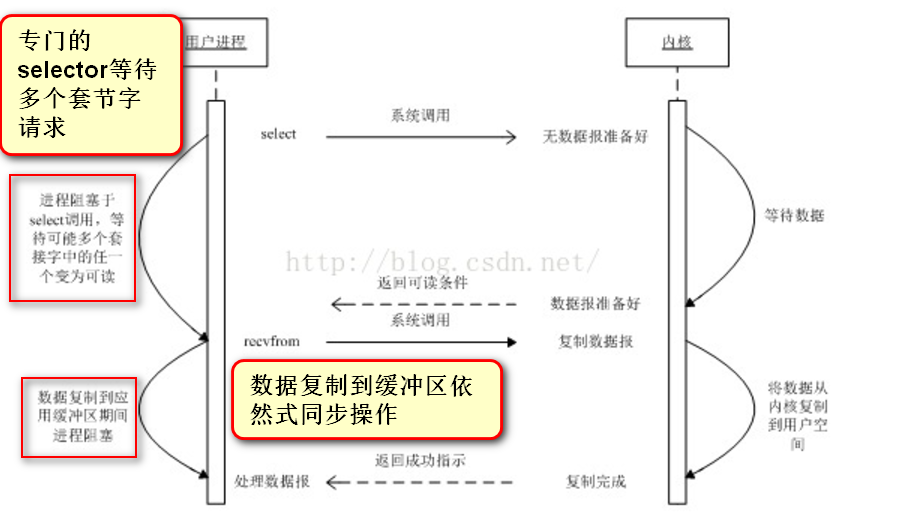




如果你有少量的连接使用非常高的带宽，一次发送大量的数据，也许典型的IO服务器实现可能非常契合。下图说明了一个典型的IO服务器模型



**Java NIO（同步非阻塞式模型）**：



这里Select监听的socket都是Non-blocking的，从代码中可以看出使用Select返回后，仍然需要轮训再检测每个socket的状态（读、写），select () 接口本身需要消耗大量时间去轮询各个句柄。，这样的轮训检测在大量连接下也是效率不高的。

**异步IO模型**

异步I/O（asynchronous I/O）由POSIX规范定义。一般地说，这些函数的工作机制是：告知内核启动某个操作，并让内核在整个操作（包括将数据从内核复制到我们自己的缓冲区）完成后通知我们。

信号驱动式I/O是由内核通知我们何时可以启动一个I/O操作，而异步I/O模型是由内核通知我们I/O操作何时完成。

