几个基本概念：

1.同步：

用户进程触发IO操作并等待或者轮询的去查看IO操作是否就绪，例如自己亲自去银行取钱。

2.异步

用户触发IO操作后，可以干别的事情，IO操作完成以后再通知当前线程。例如让小弟去银行帮你取钱，你可以干别的事情。

3.阻塞

当试图读写文件的时候，发现不可读取或者没东西可读，则进入等待状态直到可读，ATM排队取钱。

4.非阻塞

用户进程访问数据时，会马上返回一个状态值（可读不可读），使用非阻塞IO时，如果不能读写，java调用会马上返回，当IO事件分发器会通知可读写时再继续进行读写，不断循环直到读写完成。

（如在银行柜台上办理业务，先取个号，然后坐在椅子上做其他事情，直到广播通知你去办理）

同步和异步强调提交任务后是否可以做其他事，阻塞非阻塞是当有IO操作时是否可以干其他事

BIO：同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销（一直建立），当然可以通过线程池机制改善。BIO方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中。（当然可以加线程池实现伪异步IO）

NIO：同步非阻塞，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有I/O请求时才启动一个线程进行处理（需要轮询），NIO方式适用于连接数目多且连接比较短的架构，如聊天服务器。Nio 2.0是异步非阻塞，也叫AIO

AIO：异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理（有请求通知服务器去处理）。AIO方式适用于连接数目多且连接较长的架构，如相册服务器。

BIO往往会引入多线程，每个连接一个单独的线程，而NIO则是使用单线程或者只是少量的多线程，每个连接公用一个线程。

NIO最重要的地方是一个连接创建后，不需要对应一个线程，这个连接会被注册到多路复用器上面，所以所有的连接只需要一个线程就可以搞定，当这个线程中的多路复用器进行轮询时候，发现连接上有请求的话，才开启一个线程进行处理，也就是一个请求一个线程模式。在NIO的处理方式中，当一个请求来的话，开启线程进行处理，可能会等待后端应用的资源，其实这个线程就被阻塞了，当并发上来时候，还是会有BIO一样的问题。

AIO是一个有效请求一个线程，对于读操作时，当有流可读时，操作系统会将可读的流传入read方法的缓冲区，然后通知应用程序。对于写操作，当操作系统将write方法传递的流写入完毕时，操作系统主要通知应用程序。

Netty（缺点是会出现粘包、半包等问题）

创建一个通道，利用多路复用器不断进行轮询

缓冲区：在nio库中，所有数据都是用缓冲区处理的。在读取数据时，它是直接读到缓冲区中的；在写入数据时，写入数据时，写入到缓冲区中。任何时候访问nio的数据，都是通过缓冲区进行操作。

通道（channel）：channel是一个通道，可以通过它读取和写入数据，通道与流的不同之处在于通道是双向的，流只是一个方向上移动。

Selector（多路复用器）：selector会不断地轮询注册在其上的channel，如果某个channel上面有新的tcp连接接入、读和写事件，这个channel就处于就绪状态，会被selector轮询出来，然后通过selectionKey可以获取就绪channel的集合，进行后续的I/O操作。一个多路复用器selector可以同时轮询多个channel，这也就意味着只需要一个线程负责selector的轮询，就可以接入成千上万

NIO 服务端通信序列图如图 2-10 所示。

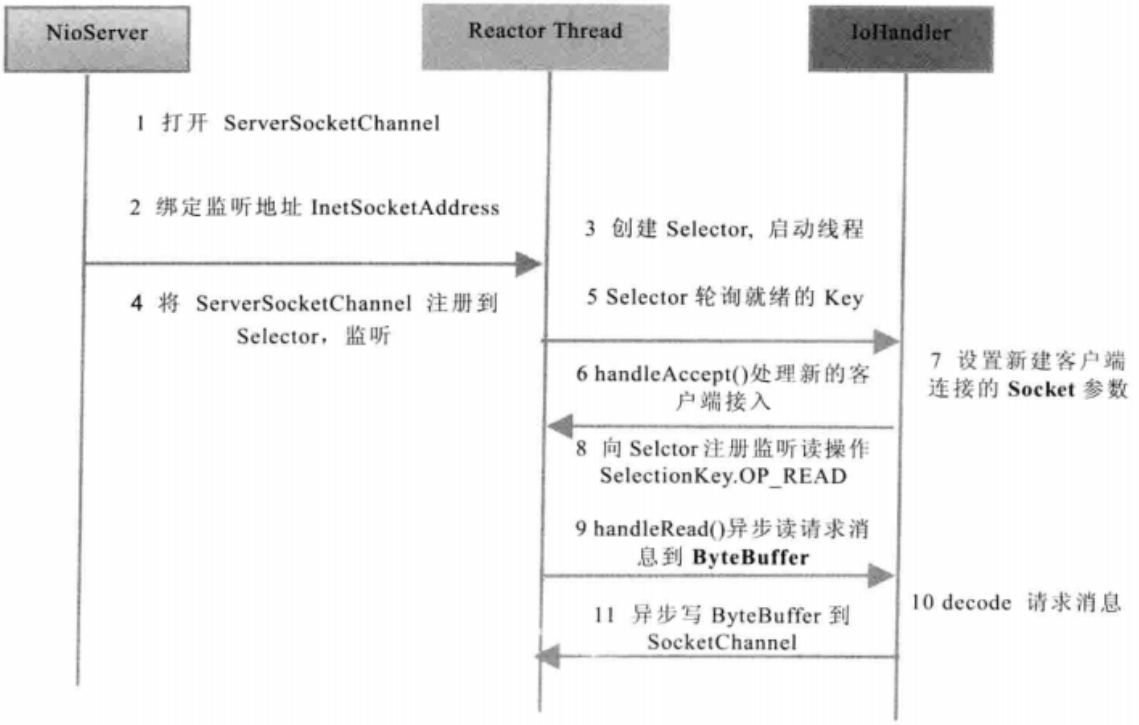


图 2-10　NIO 服务端通信序列图

在开始使用 Netty 开发 TimeServer 之前，先回顾一下使用 NIO 进行服务端开发的步骤。

（1）创建 ServerSocketChannel，配置它为非阻塞模式；

（2）绑定监听，配置 TCP 参数，例如 backlog 大小；

（3）创建一个独立的 I/O 线程，用于轮询多路复用器 Selector；

（4）创建 Selector，将之前创建的 ServerSocketChannel 注册到 Selector 上，监听 SelectionKey.ACCEPT；

（5）启动 I/O 线程，在循环体中执行 Selector.select()方法，轮询就绪的 Channel；

（6）当轮询到了处于就绪状态的 Channel 时，需要对其进行判断，如果是 OP_ACCEPT 状态，说明是新的客户端接入，则调用 ServerSocketChannel.accept()方法接受新的客户端；

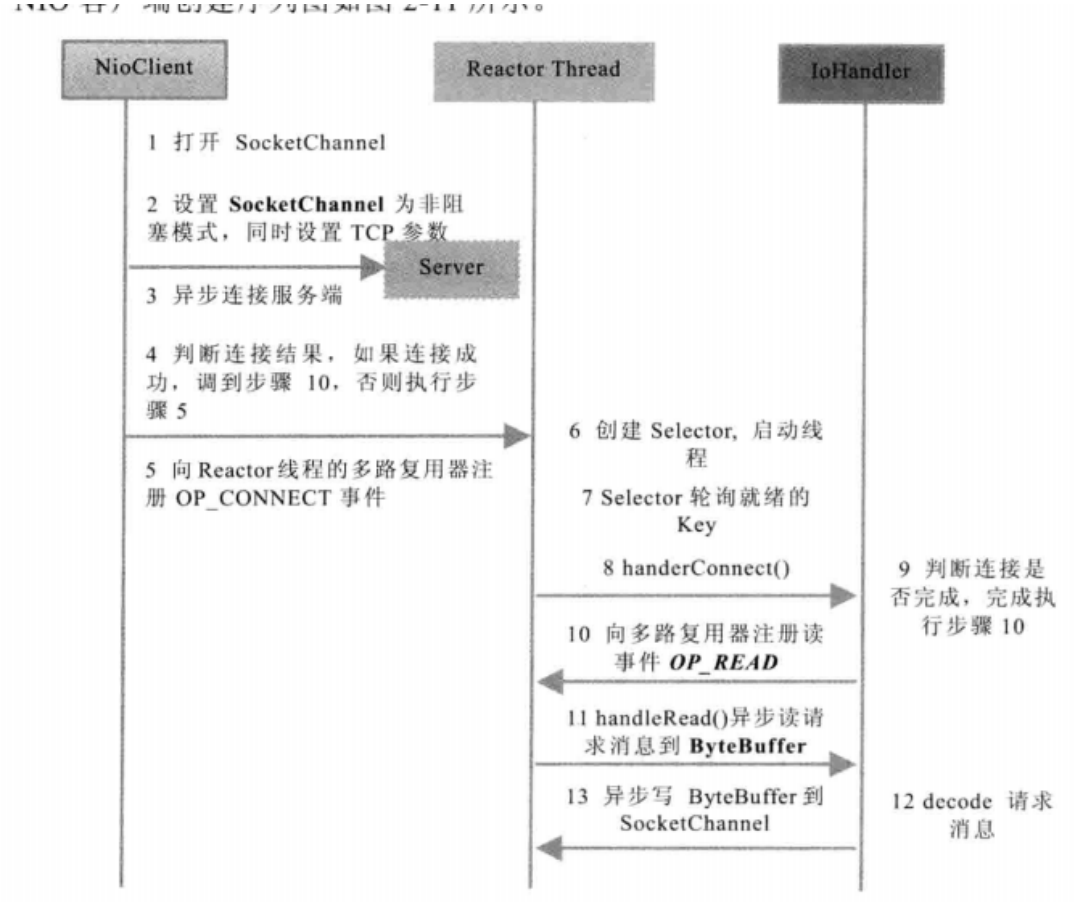
（7）设置新接入的客户端链路 SocketChannel 为非阻塞模式，配置其他的一些 TCP 参数；

（8）将 SocketChannel 注册到 Selector，监听 OP_READ 操作位；

（9）如果轮询的 Channel 为 OP_READ，则说明 SocketChannel 中有新的就绪的数据包需要读取，则构造 ByteBuffer 对象，读取数据包；

（10）如果轮询的 Channel 为 OP_WRITE，说明还有数据没有发送完成，需要继续发送。

客户端：



Netty的NIO实现方式：
服务器端向外暴露bind方法，客户端向外暴露connect方法。
在bind方法中：
生成2个EventLoopGroup对象，其实质是reactor线程组，1组进行连接处理、另1组进行网络读写；

生成ServerBootStrap对象，调用其group方法将2个ELG对象组合，调用channel方法指定Channel对象（服务器端为NioServerSocketChannel），调用option方法进行配置，调用childHandler方法配置消息处理器（childHandler继承channelInitializer<SocketChannel>类，需要自己实现）；
调用boostrack的bind（port）方法（客户端调用connect方法）进行异步连接（方法返回ChannelFuture对象，保存连接信息）。
实现childHandler类，在initChannel中，对SocketChannel的管道加入解码器和事件处理器（调用addLast方法，事件处理器需要自己实现）；
实现事件处理器（ServerHandler），重写其中的部分方法，主要包括channelRead、channelReadComplete、exceptionCaught等（客户端主要重写方法：channelActive、channelRead等）。

JDK原生的NIO方式：
服务器端生成ServerSocketChannel实例，在多路复用器（selector）上注册，监听连接事件（OP_ACCEPT），主线程轮询selector，结果以Set<SelectionKey>的方式保存，遍历轮询结果，当得到了key时，判断该key属于连接请求还是读数据请求；若收到连接请求，获取socket，并在selector上注册，监听读请求事件（OP_READ）;若收到读请求，用ByteBuffer读取并解析数据。
客户端生成SocketChannel，尝试进行连接（connect方法），若连接成功，在selector上注册OP_READ监听，否则注册OP_CONNECT监听（等待服务器返回ack），其余方式和服务器相同，获取Set<SelectionKey>，使用迭代器对set进行遍历，获取到key后进行处理，并从iterator中remove。

有时间看看源码？？？