# 基础增强合集 3

## 目录



# 1、BIO

见代码

# 2、NIO

## 2.1、简介

NIO 是 New IO 的简称，在 jdk1.4 里提供的新 API

Sun 官方标榜的特性如下：
为所有的原始类型提供(Buffer)缓存支持
字符集编码解码解决方案

Channel ：一个新的原始 I/O 抽象
支持锁和内存映射文件的文件访问接口
提供多路(non-bloking)非阻塞式的高伸缩性网络 I/O

# 2.2、两对核心概念

**1、阻塞和非阻塞**

阻塞和非阻塞是进程在**访问数据**的时候，数据是否准备就绪的一种处理方式。
当数据没有准备好的时候，
阻塞：往往需要等待缓冲区中的数据准备好之后才处理，否则一直等待。
非阻塞：当我们的进程访问我们的数据缓冲区的时候，数据没有准备好的时候，直接返回，不需要等待。有数据的时候，也直接返回

**2、同步和异步**

同步和异步都是基于应用程序和操作系统**处理 IO 事件**所采用的方式:
同步：应用程序要直接参与 IO 事件的操作；
异步：所有的 IO 读写事件交给操作系统去处理；

同步的方式在处理 IO 事件的时候，必须阻塞在某个方法上面等待我们的 IO 事件完成（阻塞在 IO 事件或者通过轮询 IO 事件的方式）；
对于异步来说，所有的 IO 读写都交给了操作系统，这个时候，我们可以去做其他的事情，并不需要去完成真正的 IO 操作。当操作系统完成 IO 之后，给我们的应用程序一个通知就可以了。

同步有两种实现模式：
1、阻塞到 IO 事件
阻塞到 read 或者 write 方法上，这个时候我们就完全不能做自己的事情。（在这种情况下，我们只能把读写方法放置到线程中，然后阻塞线程的方式来实现并发服务，对线程的性能开销比较大）

2、IO 事件的轮询
在 linux c 语言编程中叫做多路复用技术（select 模式）
读写事件交给一个专门的线程来处理，这个线程完成 IO 事件的注册功能，还有就是不断地去轮询我们的读写缓冲区（操作系统），看是否有数据准备好，然后通知我们的相应的业务处理线程。这样的话，我们的业务处理线程就可以做其他的事情。在这种模式下，阻塞的不是所有的 IO 线程，而是阻塞的只是 select 线程

**同步和异步总结：**
同步就是：如果有多个任务或者事件要发生，这些任务或者事件必须逐个地进行，一个事件或者任务的执行会导致整个流程的暂时等待，这些事件没有办法并发地执行；
异步就是：如果有多个任务或者事件发生，这些事件可以并发地执行，一个事件或者任务的执行不会导致整个流程的暂时等待。

**同步和异步的区别：在于多个任务和事件发生时，一个事件的发生或执行是否会导致整个流程的暂时等待**

**阻塞和非阻塞总结：**

阻塞就是：当某个事件或者任务在执行过程中，它发出一个请求操作，但是由于该请求操作需要的条件不满足，那么就会一直在那等待，直至条件满足；

非阻塞就是：当某个事件或者任务在执行过程中，它发出一个请求操作，如果该请求操作需要的条件不满足，会立即返回一个标志信息告知条件不满足，不会一直在那等待。

**阻塞和非阻塞的区别：关键在于当发出请求一个操作时，如果条件不满足，是会一直等待还是返回一个标志信息。**

参考资料：http://www.cnblogs.com/dolphin0520/p/3916526.html

## 2.3、JAVA IO 模型

基于以上 4 中 IO 模型，JAVA 对应的实现有：

**BIO--同步阻塞：**
JDK1.4 以前我们使用的都是 BIO
阻塞到我们的读写方法，阻塞到线程来提高并发性能，但是效果不是很好

**NIO--同步非阻塞：**JDK1.4 Linux 多路复用技术（select 模式）实现 IO 事件的轮询方式：同步非阻塞的模式，这种方式目前是主流的网络通信模式
Mina 和 Netty　-- 网络通信框架，比自己写 NIO 要容易些，并且代码可读性更好

**AIO--异步非阻塞 IO：**JDK1.7（NIO2）真正的异步非阻塞 IO(基于 linux 的 epoll 模式）AIO 目前使用的还比较少

小结：
1）BIO　阻塞的 IO
2）NIO　select 多路复用+非阻塞　　同步非阻塞
3）AIO　异步非阻塞 IO

## 2.4、NIO 中的核心抽象

### 2.4.1、缓冲区 Buffer

Buffer 是一个对象，它包含一些要写入或者要读出的数据。在 NIO 类库中加入 Buffer 对象，体现了新库与原 I/O 的一个重要区别。在面向流的 I/O 中，可以将数据直接写入或者将数据直接读到 Stream 对象中。

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，写入到缓冲区中。任何时候访问 NIO 中的数据，都是通过缓冲区进行操作。

缓冲区实质上是一个数组。通常它是一个字节数组(ByteBuffer)，也可以使用其他种类的数组。但是缓冲区不仅仅是一个数组，缓冲区提供了对数据的结构化访问以及维护读写位置(limit)等信息。

最常用的缓冲区是 ByteBuffer，一个 ByteBuffer 提供了一组功能用于操作 byte 数组。除了 ByteBuffer，还有其他的一些缓冲区，事实上，每一种 Java 基本类型（除了 Boolean 类型）都对应有一种缓冲区，具体如下：
ByteBuffer：字节缓冲区
CharBuffer：字符缓冲区
ShortBuffer：短整型缓冲区
IntBuffer：整型缓冲区
LongBuffer：长整型缓冲区
FloatBuffer：浮点型缓冲区
DoubleBuffer：双精度浮点型缓冲区

每一个 Buffer 类都是 Buffer 接口的一个子实例。除了 ByteBuffer,每一个 Buffer 类都有完全一样的操作，只是它们所处理的数据类型不一样。因为大多数标准 I/O 操作都是使用 ByteBuffer，所以它除了具有一般缓冲区的操作之外还提供一些特有的操作，方便网络读写。

所有缓冲区都有4个属性：capacity、limit、position、mark，并遵循：capacity>=limit>=position>=mark>=0，下表格是对着4个属性的解释：

| 属性 | 描述 |
| --- | --- |
| Capacity | 容量，即可以容纳的最大数据量；在缓冲区创建时被设定并且不能改变 |
| Limit | 上界，缓冲区中当前数据量 |
| Position | 位置，下一个要被读或写的元素的索引 |
| Mark | 标记，调用mark()来设置mark=position，再调用reset()可以让position恢复到标记的位置即position=mark |

## 2.4.2、通道 Channel

Channel 是一个通道，可以通过它读取和写入数据，它就像自来水管一样，网络数据通过 Channel 读取和写入。通道与流的不同之处在于通道是双向的，流只是在一个方向上移动（一个流必须是 InputStream 或者 OutputStream 的子类），而且通道可以用于读、写或者同时读写。因为 Channel 是全双工的，所以它可以比流更好地映射底层操作系统的 API。

## 2.4.3、多路复用器 Selector

多路复用器 Selector 是 Java NIO 编程的基础，熟练地掌握 Selector 对于掌握 NIO 编程至关重要。多路复用器提供选择已经就绪的任务的能力。简单来讲，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，由于 JDK 使用了 epoll()代替传统的 select 实现，所以它并没有最大连接句柄 1024/2048 的限制。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的客户端，这确实是个非常巨大的进步。

Selector 感兴趣的事件有：
OP_ACCEPT
OP_CONNECT
OP_READ
OP_WRITE

## 2.5、NIO 示例代码

见代码

# 3、Netty

# 4、AIO

# 5、RPC