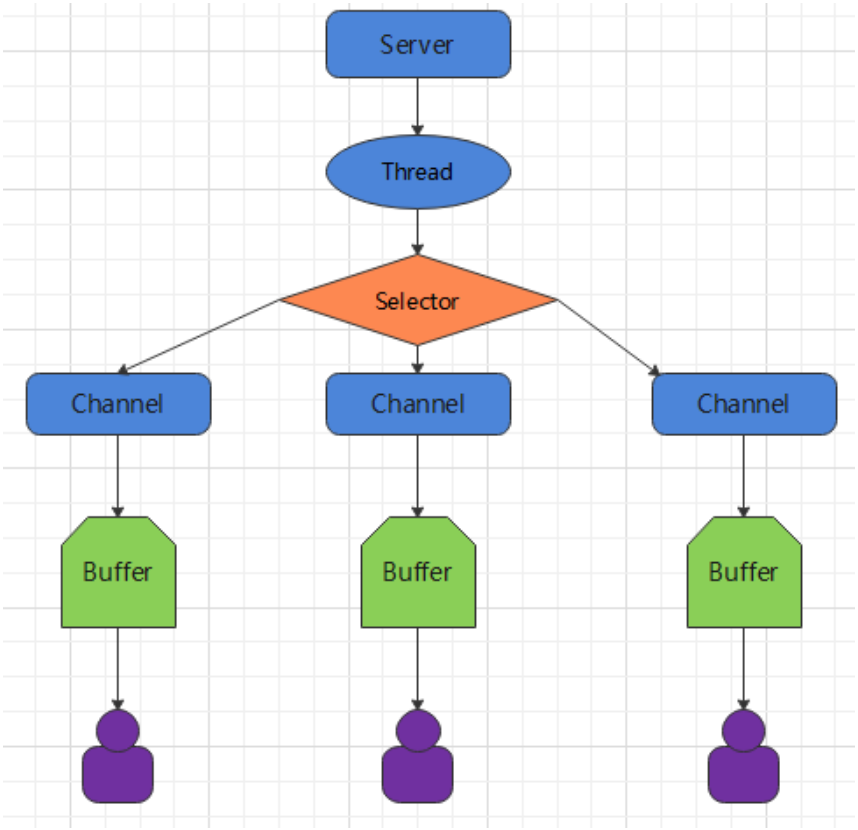
NIO

1. Java NIO 介绍：

Java NIO（New IO）也称为 non-blocking。NIO 是从 Java 1.4 版本开始引入的一个新的 IO API，可以替代标准的 Java IO API。

NIO 与原来的 IO 有同样的作用和目的，但是使用的方式完全不同，NIO 支持面**向缓冲区**的、基于**通道**的 IO 操作。NIO 将以更加高效的方式进行文件的读写操作。NIO 可以理解为非阻塞 IO,传统的 IO 的 read 和 write 只能阻塞执行，线程在读写 IO 期间不能干其他事情，比如调用 socket.read()时，如果服务器一直没有数据传输过来，线程就一直阻塞，而 NIO 中可以配置 socket 为非阻塞模式。NIO 有三大核心部分：**Channel(通道) ， Buffer(缓冲区), Selector(选择器)。**



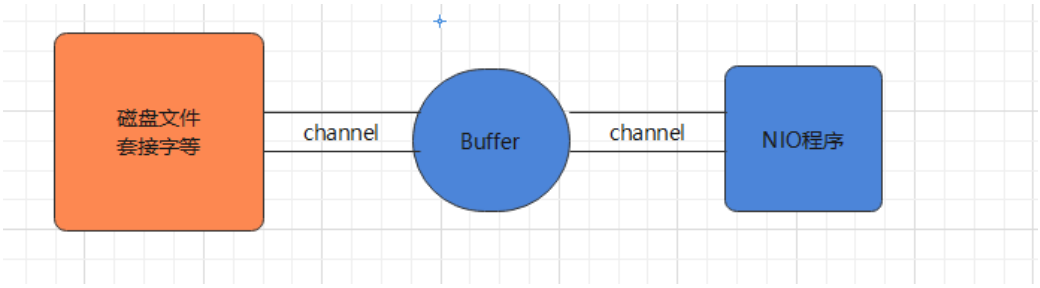
2. BIO 和 NIO 的比较：

- BIO 以流的方式处理数据,而 NIO 以块的方式处理数据,块 I/O 的效率比流 I/O 高很多
- BIO 是阻塞的，NIO 则是非阻塞的
- BIO 基于字节流和字符流进行操作，而 NIO 基于 Channel(通道)和 Buffer(缓冲区)进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。Selector(选择器)用于监听多个通道的事件（比如：连接请求，数据到达等），因此使用单个线程就可以监听多个客户端通道。

3. NIO三大核心部分

a. 缓冲区（Buffer）

缓冲区本质上是一块可以写入数据，然后可以从中读取数据的内存。这块内存被包装成NIO Buffer对象，并提供了一组方法，用来方便的访问该块内存。相比较直接对数组的操作，Buffer API更加容易操作和管理。所有缓冲区都是 Buffer 抽象类的子类.。Java NIO 中的 Buffer 主要用于与 NIO 通道进行 交互，数据是从通道读入缓冲区，从缓冲区写入通道中的。

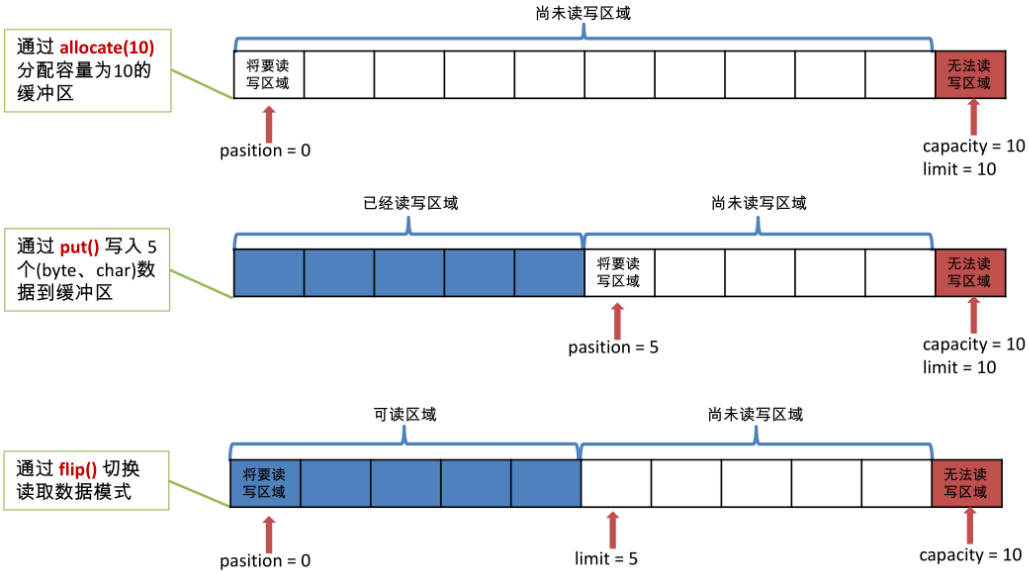


Buffer 就像一个数组，可以保存多个相同类型的数据。根 据数据类型不同 ， 有以下 Buffer 常用子类：

- ByteBuffer
- CharBuffer
- ShortBuffer
- IntBuffer
- LongBuffer
- FloatBuffer
- DoubleBuffer

Buffer 中的重要概念：

- **容量 (capacity)**：作为一个内存块，Buffer具有一定的固定大小，也称为"容量"，缓冲区容量不能为负，并且创建后不能更改。
- **限制 (limit)**：表示缓冲区中可以操作数据的大小（limit 后数据不能进行读写）。缓冲区的限制不能为负，并且不能大于其容量。 **写入模式，限制等于buffer的容量。读取模式下，limit等于写入的数据量。**
- **位置 (position)**：下一个要读取或写入的数据的索引。缓冲区的位置不能为负，并且不能大于其限制
- **标记 (mark)与重置 (reset)**：标记是一个索引，通过 Buffer 中的 mark() 方法 指定 Buffer 中一个特定的 position，之后可以通过调用 reset() 方法恢复到这 个 position.
- **标记、位置、限制、容量遵守以下不变式： 0 <= mark <= position <= limit <= capacity**



直接与非直接缓冲区

byte buffer可以是两种类型，一种是基于直接内存（也就是非堆内存）；另一种是非直接内存（也就是堆内存）。对于直接内存来说，JVM将会在IO操作上具有更高的性能，因为它直接作用于本地系统的IO操作。而非直接内存，也就是堆内存中的数据，如果要作IO操作，会先从本进程内存复制到直接内存，再利用本地IO处理。

从数据流的角度，非直接内存是下面这样的作用链：

本地IO-->直接内存-->非直接内存-->直接内存-->本地IO

而直接内存是：

本地IO-->直接内存-->本地IO

很明显，在做IO处理时，比如网络发送大量数据时，直接内存会具有更高的效率。直接内存使用allocateDirect创建，但是它比申请普通的堆内存需要耗费更高的性能。不过，这部分的数据是在JVM之外的，因此它不会占用应用的内存。所以呢，当你有很大的数据要缓存，并且它的生命周期又很长，那么就比较适合使用直接内存。只是一般来说，如果不是能带来很明显的性能提升，还是推荐直接使用堆内存。字节缓冲区是直接缓冲区还是非直接缓冲区可通过调用其isDirect()方法来确定。

使用场景

1 有很大的数据需要存储，它的生命周期又很长

2 适合频繁的IO操作，比如网络并发场景

b. 通道（Channel）

通道（Channel）：由 java.nio.channels 包定义 的。Channel 表示 IO 源与目标打开的连接。 Channel 类似于传统的“流”。只不过 Channel 本身不能直接访问数据，Channel 只能与 Buffer 进行交互。

NIO 的通道类似于流，但有些区别如下：

- 通道是双向的，可以同时进行读写，而流是单向的，只能读或者只能写
- 通道可以实现异步读写数据
- 通道可以从缓冲读数据，也可以写数据到缓冲

常见的Channel实现类：

- FileChannel：用于读取、写入、映射和操作文件的通道。
- DatagramChannel：通过 UDP 读写网络中的数据通道。
- SocketChannel：通过 TCP 读写网络中的数据。

- ServerSocketChannel：可以监听新进来的 TCP 连接，对每一个新进来的连接都会创建一个 SocketChannel。【ServerSocketChanne 类似 ServerSocket , SocketChannel 类似 Socket】

c. 选择器（Selector）

选择器（Selector） 是 SelectableChannle 对象的多路复用器，Selector 可以同时监控多个 SelectableChannel 的 IO 状况，也就是说，利用 Selector可使一个单独的线程管理多个 Channel。**Selector 是非阻塞 IO 的核心。**

选择器的特点:

- Java 的 NIO，用非阻塞的 IO 方式。可以用一个线程，处理多个的客户端连接，就会使用到 Selector(选择器)
- Selector 能够检测多个注册的通道上是否有事件发生(注意:多个 Channel 以事件的方式可以注册到同一个Selector)，如果有事件发生，便获取事件然后针对每个事件进行相应的处理。这样就可以只用一个单线程去管理多个通道，也就是管理多个连接和请求。
- 只有在 连接/通道 真正有读写事件发生时，才会进行读写，就大大地减少了系统开销，并且不必为每个连接都创建一个线程，不用去维护多个线程
- 避免了多线程之间的上下文切换导致的开销

4. AIO编程：

Java AIO(NIO.2) ： 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。

与NIO不同，当进行读写操作时，只须直接调用API的read或write方法即可，这两种方法均为异步的，对于读操作而言，当有流可读取时，操作系统会将可读的流传入read方法的缓冲区,对于写操作而言，当操作系统将write方法传递的流写入完毕时，操作系统主动通知应用程序。即可以理解为，read/write方法都是异步的，完成后会主动调用回调函数。

5. 三种IO的使用场景：

- 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，JDK1.4以前的唯一选择，但程序直观简单易理解。
- NIO方式适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4开始支持。
- AIO方式使用于连接数目多且连接比较长（重操作）的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂，JDK7开始支持。