**2018年10月19日**



[有关BIO与NIO]

# 关于BIO与NIO

## 名称解释

BIO：同步阻塞IO；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端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，当然可以通过线程池机制改善。

NIO：同步非阻塞IO；服务器模式为“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。

NIO是基于事件驱动思想完成的，其主要解决的是BIO的高并发问题。NIO基于Reactor，当socket有流可读或可写入socket时，操作系统会相应的通知引用程序进行处理，应用再将流读取到缓冲区或写入操作系统。 也就是说，这个时候，已经不是一个连接就要对应一个处理线程了，而是有效的请求，对应一个线程，当连接没有数据时，是没有工作线程来处理的

BIO与NIO区别：BIO的时候往往会引入多线程，每个连接一个单独的线程；而NIO则是使用单线程或者只使用少量的多线程，每个连接共用一个线程。

AIO：异步非阻塞IO；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程序启动线程进行处理。

使用场景：

（1）NIO适合处理连接数目特别多，但是连接比较短（轻操作）的场景，Jetty，Mina，ZooKeeper等都是基于java nio实现。

（2）BIO方式适用于连接数目比较小且固定的场景，这种方式对服务器资源要求比较高，并发局限于应用中。

（3）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

在高性能的I/O设计中，有两个比较著名的模式Reactor和Proactor模式，其中Reactor模式用于同步I/O，而Proactor运用于异步I/O操作。

# 从IO到NIO

## 面向流 vs. 面向缓冲

Java IO是面向流的，每次从流（InputStream/OutputStream）中读一个或多个字节，直到读取完所有字节，它们没有被缓存在任何地方。另外，它不能前后移动流中的数据，如需前后移动处理，需要先将其缓存至一个缓冲区。

Java NIO面向缓冲，数据会被读取到一个缓冲区，需要时可以在缓冲区中前后移动处理，这增加了处理过程的灵活性。但与此同时在处理缓冲区前需要检查该缓冲区中是否包含有所需要处理的数据，并需要确保更多数据读入缓冲区时，不会覆盖缓冲区内尚未处理的数据。

## 阻塞 vs. 非阻塞

Java IO的各种流是阻塞的。当某个线程调用read()或write()方法时，该线程被阻塞，直到有数据被读取到或者数据完全写入。阻塞期间该线程无法处理任何其它事情。

Java NIO为非阻塞模式。读写请求并不会阻塞当前线程，在数据可读/写前当前线程可以继续做其它事情，所以一个单独的线程可以管理多个输入和输出通道。

## 选择器（Selector）

Java NIO的选择器允许一个单独的线程同时监视多个通道，可以注册多个通道到同一个选择器上，然后使用一个单独的线程来“选择”已经就绪的通道。这种“选择”机制为一个单独线程管理多个通道提供了可能。

## 零拷贝

Java NIO中提供的FileChannel拥有transferTo和transferFrom两个方法，可直接把FileChannel中的数据拷贝到另外一个Channel，或者直接把另外一个Channel中的数据拷贝到FileChannel。该接口常被用于高效的网络/文件的数据传输和大文件拷贝。在操作系统支持的情况下，通过该方法传输数据并不需要将源数据从内核态拷贝到用户态，再从用户态拷贝到目标通道的内核态，同时也避免了两次用户态和内核态间的上下文切换，也即使用了“零拷贝”，所以其性能一般高于Java IO中提供的方法。