# SocketChannel编程入门

## 1.入门必备

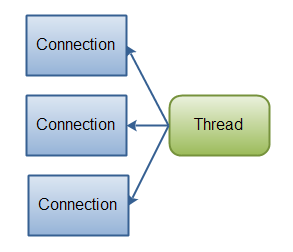
与传统的阻塞的**Socket**和**ServerSocket**相比,在JDK1.4版本时,提供了可以进行非阻塞通信的**SocketChannel**(抽象类）和**ServerSocketChannel**(抽象类)。

我们可以通过SocketChannel#configureBlocking(false)来将客户端通信设置成非阻塞的。非阻塞的实现，改变了传统服务端需要一个线程绑定一个Socket的实现模式。现在，一个线程可以方便的监听和管理多个通道(channel)。

你可以使用一个选择器(selector)来注册多个通道，然后使用一个单独的线程来“选择”通道，这样就使得一个单独的线程很容易来管理多个通道。

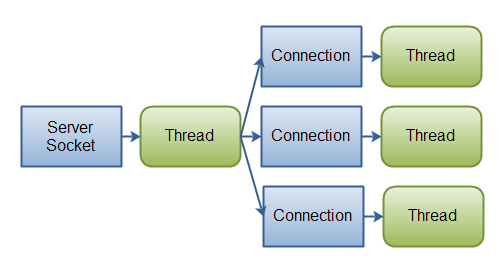
SocketChannel和ServerSocketChannel都是在java.nio包下，NIO可让您只使用一个（或几个）单线程管理多个通道（网络连接或文件），但付出的代价是解析数据可能会比从一个阻塞流中读取数据更复杂。

如果需要管理同时打开的成千上万个连接，这些连接每次只是发送少量的数据，例如聊天服务器，实现NIO的服务器可能是一个优势。同样，如果你需要维持许多打开的连接到其他计算机上，如P2P网络中，使用一个单独的线程来管理你所有出站连接，可能是一个优势。一个线程多个连接的设计方案如下图所示：



**Java NIO: 单线程管理多个连接**

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服务器设计：



**Java IO: 一个典型的IO服务器设计- 一个连接通过一个线程处理.**

## 2.使用细节

我们可以调用SocketChannel#[open](http://www.cjsdn.net/Doc/JDK50/java/nio/channels/SocketChannel.html#open())方法创建套接字通道，这时，新创建的套接字通道已打开，但尚未连接。试图在未连接的通道上调用I/O操作将导致抛出[NotYetConnectedException](http://www.cjsdn.net/Doc/JDK50/java/nio/channels/NotYetConnectedException.html)。可通过调用套接字通道的SocketChannel# [connect](http://www.cjsdn.net/Doc/JDK50/java/nio/channels/SocketChannel.html#connect(java.net.SocketAddress))方法连接该通道。一旦连接后，关闭套接字通道之前它会一直保持已连接状态。可通过调用套接字通道的SocketChannel#[isConnected](http://www.cjsdn.net/Doc/JDK50/java/nio/channels/SocketChannel.html#isConnected())方法来确定套接字通道是否已连接。由于[connect](http://www.cjsdn.net/Doc/JDK50/java/nio/channels/SocketChannel.html#connect(java.net.SocketAddress))方法可能在连接建立之前就返回了，所以我们需要用SocketChannel#finishConnect()来检验连接是否成功建立。