使用NIO实现客户端与服务端通信：

服务端：

package edu.xhu.nio;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.ClosedChannelException;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.nio.channels.ServerSocketChannel;

import java.nio.channels.SocketChannel;

import java.util.Iterator;

public class NioServer {

public static void main(String[] args) throws IOException {

// nio

System.out.println("服务器端已经启动....");

// 1.创建通道

ServerSocketChannel sChannel = ServerSocketChannel.open();

// 2.切换读取模式

sChannel.configureBlocking(false);

// 3.绑定连接

sChannel.bind(new InetSocketAddress(8080));

// 4.获取选择器

Selector selector = Selector.open();

// 5.将通道注册到选择器 "并且指定监听接受事件"

sChannel.register(selector, SelectionKey.OP\_ACCEPT);

// 6. 轮训式 获取选择 "已经准备就绪"的事件

while (selector.select() > 0) {

// 7.获取当前选择器所有注册的"选择键(已经就绪的监听事件)"

Iterator<SelectionKey> it = selector.selectedKeys().iterator();

while (it.hasNext()) {

// 8.获取准备就绪的事件

SelectionKey sk = it.next();

// 9.判断具体是什么事件准备就绪

if (sk.isAcceptable()) {

// 10.若"接受就绪",获取客户端连接

SocketChannel socketChannel = sChannel.accept();

// 11.设置阻塞模式

socketChannel.configureBlocking(false);

// 12.将该通道注册到服务器上

socketChannel.register(selector, SelectionKey.OP\_READ);

} else if (sk.isReadable()) {

// 13.获取当前选择器"就绪" 状态的通道

SocketChannel socketChannel = (SocketChannel) sk.channel();

// 14.读取数据

ByteBuffer buf = ByteBuffer.allocate(1024);

int len = 0;

while ((len = socketChannel.read(buf)) > 0) {

buf.flip();

System.out.println(new String(buf.array(), 0, len));

buf.clear();

}

}

it.remove();

}

}

}

}

客户端：

package edu.xhu.nio;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.SocketChannel;

import java.util.Date;

import java.util.Scanner;

public class NioClient {

public static void main(String[] args) throws IOException {

// nio 异步非阻塞

System.out.println("客户端已经启动....");

// 1.创建通道

SocketChannel sChannel = SocketChannel.open(new InetSocketAddress("127.0.0.1", 8080));

// 2.切换异步非阻塞

sChannel.configureBlocking(false);

// 3.指定缓冲区大小

ByteBuffer byteBuffer = ByteBuffer.allocate(1024);

Scanner scanner = new Scanner(System.in);

while (scanner.hasNext()) {

String str = scanner.next();

byteBuffer.put((new Date().toString() + "\n" + str).getBytes());

// 4.切换读取模式

byteBuffer.flip();

sChannel.write(byteBuffer);

byteBuffer.clear();

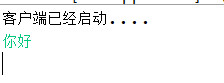
}

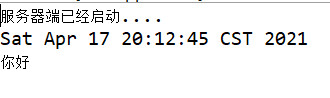
sChannel.close();

}

}

运行截图：





总结：

NIO：NewIＯ、　NON　BLOCKing　IO非阻塞IO。这样就不用持续等待客户端连接，让程序占着ｃｐｕ的资源，服务端的套接字通道注册到轮训器中，采用轮训器监听的方式，监听每个客户端套接字通道上的要办的事件，当这件事准备好后反应给轮训器，它再将该客户连接进来，反应给服务端能进行执行，这种轮训的方法就能让服务端使用一个线程来为监听多个客户请求，从而达到提高并发量的目的。NIO的新概念中采用了通道和缓冲区概念，与传统的IO的流概念不在相同。