## 基于java的网络协议消息的构建和解析

姓名：李晓鸿

学号：20131204028

年级：2013

专业：网络工程

**NIo 简介**

nio 是 java 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 。

本文将围绕这几个特性进行学习和介绍。

**1 . Buffer&Chanel**

Channel 和 buffer 是 NIO 是两个最基本的数据类型抽象。

Buffer:

–        是一块连续的内存块。

–        是 NIO 数据读或写的中转地。

Channel:

–        。数据的源头或者数据的目的地

–        用于向 buffer 提供数据或者读取 buffer 数据 ,buffer 对象的唯一接口。

2.     异步 I/O 支持

Buffer 常见方法：

flip(): 写模式转换成读模式

rewind() ：将 position 重置为 0 ，一般用于重复读。

clear() ：清空 buffer ，准备再次被写入 (position 变成 0 ， limit 变成 capacity) 。

compact(): 将未读取的数据拷贝到 buffer 的头部位。

mark() 、 reset():mark 可以标记一个位置， reset 可以重置到该位置。

Buffer 常见类型： ByteBuffer 、 MappedByteBuffer 、 CharBuffer 、 DoubleBuffer 、 FloatBuffer 、 IntBuffer 、 LongBuffer 、 ShortBuffer。

channel 常见类型 :FileChannel 、 DatagramChannel(UDP) 、 SocketChannel(TCP) 、 ServerSocketChannel(TCP)

主要参数：

position

当前写入的单位数据数量。

当前读取的单位数据位置。

limit

代表最多能写多少单位数据和容量是一样的。

代表最多能读多少单位数据，和之前写入的单位数据量一致。

capacity

buffer 容量

buffer 容量

## 基于java的网络协议消息的构建和解析

客户端：

首先用Socket创建一个连接，监听服务器端的端口和地址，

创建非阻塞式SocketChannel

连接服务器，创建读写缓冲区，反复循环知道发送和接受完所有字节

打印和接受的数据，关闭通道 。

服务器端：

构建函数，从键中获取信道，并接受来连接。设置为非阻塞式

为信道注册选择器

获取关联的信道

获取键关联的缓冲区

从信道中读数据

检查数据流和结束并关闭信道

如果接受完数据，将其标记为可写

获取包含数据的缓冲区

准备缓冲区的写操作

获取信道

向信道写数据

压缩缓冲区

Java客户端代码：

package com.lxh.nio.message;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.InetSocketAddress;

import java.net.SocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.nio.channels.SocketChannel;

import java.nio.charset.Charset;

public class NioClient {

/\*\*

\* @author lxh version 1.0

\* @param args

\* @throws IOException

\*/

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

int port=8080;

SocketChannel socketchannel=SocketChannel.open();

socketchannel.configureBlocking(false);

SocketAddress socketaddress=new InetSocketAddress("localhost", port);

socketchannel.connect(socketaddress);

Selector selector=Selector.open();

socketchannel.register(selector, SelectionKey.OP\_CONNECT);

BufferedReader systemIn=new BufferedReader(new InputStreamReader(System.in));

while(true){

if(socketchannel.isConnected()){

String command=systemIn.readLine();

socketchannel.write(Charset.forName("UTF-8").encode(command));

if(command==null||"quit".equalsIgnoreCase(command.trim())){

systemIn.close();

socketchannel.close();

selector.close();

System.out.println("Client quit !");

System.exit(0);

}

}

int nKeys=selector.select(1000);

if(nKeys>0){

for(SelectionKey key:selector.selectedKeys()){

if(key.isConnectable()){

SocketChannel sc=(SocketChannel) key.channel();

sc.configureBlocking(false);

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

sc.finishConnect();

}else if(key.isReadable()){

ByteBuffer buffer=ByteBuffer.allocate(1024);

SocketChannel sc=(SocketChannel) key.channel();

int readBytes=0;

try{

int req=0;

try{

while((req=sc.read(buffer))>0){

readBytes+=req;

}

}finally{

buffer.flip();

}

if (readBytes > 0) {

System.out.println(Charset.forName("UTF-8")

.decode(buffer).toString());

buffer = null;

}

}finally {

if (buffer != null) {

buffer.clear();

}

}

}

}

selector.selectedKeys().clear();

}

}

}

}

服务器端代码：

package com.lxh.nio.message;

import java.io.IOException;

import java.net.InetSocketAddress;

import java.net.ServerSocket;

import java.nio.ByteBuffer;

import java.nio.channels.SelectionKey;

import java.nio.channels.Selector;

import java.nio.channels.ServerSocketChannel;

import java.nio.channels.SocketChannel;

import java.nio.charset.Charset;

public class NIOServer {

/\*\*

\* @author lxh version 1.0

\* @param args

\* @throws IOException

\*/

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

int port = 8080;

Selector selector = Selector.open();

ServerSocketChannel aa = ServerSocketChannel.open();

ServerSocket serverSoket = aa.socket();

serverSoket.bind(new InetSocketAddress(port));

System.out.println("the server listening on port:" + port);

aa.configureBlocking(false);

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

while (true) {

int nkeys = selector.select(1000);

if (nkeys > 0) {

for (SelectionKey key : selector.selectedKeys()) {

if (key.isAcceptable()) {

ServerSocketChannel server = (ServerSocketChannel) key.channel();

SocketChannel sc = server.accept();

if (sc == null) {

continue;

}

sc.configureBlocking(false);

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

} else if (key.isReadable()) {

ByteBuffer buffer = ByteBuffer.allocate(1024);

SocketChannel sc = (SocketChannel) key.channel();

int readBytes = 0;

String message = null;

try {

int req;

try {

while ((req = sc.read(buffer)) > 0) {

readBytes += req;

}

} catch (Exception e) {

readBytes = 0;

} finally {

buffer.flip();

}

if (readBytes > 0) {

message = Charset.forName("UTF-8").decode(buffer).toString();

buffer = null;

}

} finally {

if (buffer != null)

buffer.clear();

}

if (readBytes > 0) {

System.out.println("message is received from client:" + message);

if ("quit".equalsIgnoreCase(message.trim())) {

sc.close();

selector.close();

System.out.println("Server has been shutdown!");

System.exit(0);

}

String outMessage = "server response:" + message;

sc.write(Charset.forName("UTF-8").encode(outMessage));

}

}

}

selector.selectedKeys().clear();

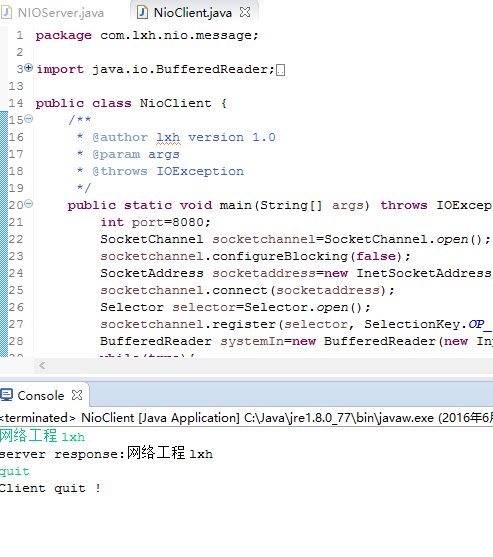
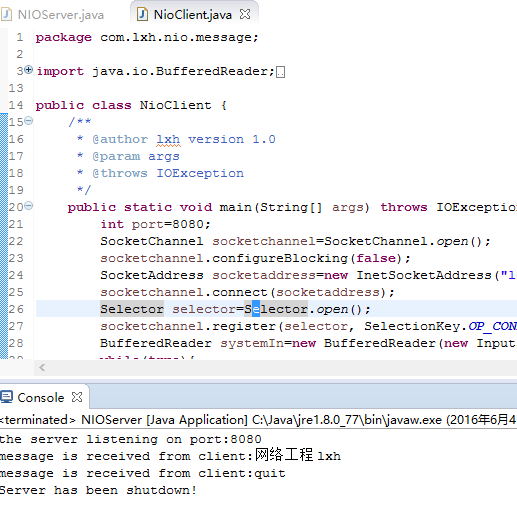
}

}

}

}

结果：



体会：

通过本实验，了解和认识到NIO的基本操作，了解和掌握了I/O 的基本操作，对数据的传输和解析的边界有一定的了解。对Socket的数据传输和操作进一步的了解，下一步进一步学习和拓展netty和miny的一些知识。