//使用套接字进行TCP传输验证服务端（监听多个客户）和客户端通信

源码：

客户端：

package xhu.edu.socket;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.net.Socket;

import java.net.UnknownHostException;

/\*

\* 问题：为什么在网络通讯过程中使用缓冲流需要进行flush(),如果不flush(),另一端套接字就没有的数据

\* 而以前在控制台继续宁输入并且进行写入文件时并不需要进行flush(),

\* 并且为什么使用底层的基本流进行读写就没有这种状况？

\* 答：查询参考手册知道：flush()刷新此缓冲的输出流。 这会强制将任何缓冲的输出字节写出到底层输出流。那么可以推测：

\* 为了保证效率，缓冲字符流会默认将缓冲区读满才会将缓冲区中的数据通过底层基本流写出。而输入流才是要么缓冲区满了，就结束，

\* 要么就是通过整行读取，读一行写一行。 底层基本流不涉及缓冲区所以不会出现这个问题。

\*/

public class Client {

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

Socket socket = new Socket("localhost", 1000);

BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream(), "gbk"));

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream(), "gbk"));

BufferedReader brC = new BufferedReader(new InputStreamReader(System.in, "gbk"));

String string;

while (true) {

string = brC.readLine();

if (string != null) {

bw.write(string);

bw.newLine();

bw.flush();

System.out.println(br.readLine());

}

if (("--bye--").equals(string)) {

break;

}

}

br.close();

bw.close();

brC.close();

socket.close();

}

}

服务端：

package xhu.edu.socket;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.io.UnsupportedEncodingException;

import java.net.ServerSocket;

import java.net.Socket;

//这里我们需要验证一下第二次请求后第一个进程的套接字的连接是否会改变？

public class Server {

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

// 监听1000端口号

ServerSocket serverSocket = new ServerSocket(1000);

while (true) {

// 等待客户连接，该方法为阻塞方法

Socket scSocket = serverSocket.accept();

Thread.sleep(5000);

new Thread(new Runnable() {

// 获取套接字的输入输出流

BufferedReader br = new BufferedReader(new InputStreamReader(scSocket.getInputStream(), "gbk"));

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(scSocket.getOutputStream(), "gbk"));

String string;

@Override

public void run() {

// 循环读写

while (true) {

try {

string = br.readLine();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

try {

// if (string != null) {

System.out.println(scSocket.getInetAddress() + "说：" + string);

bw.write("服务器回复:" + string);

bw.newLine();

bw.flush();

//}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

if (("--bye--").equals(string)) {

break;

}

}

// 关闭资源

try {

br.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

try {

bw.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

try {

scSocket.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}).start();

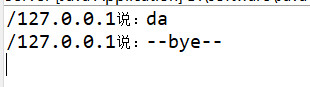
}

}

}

运行截图：





总结：

基于TCP协议的套接字传输：

服务端：

1.使用ServerSocket进行指定端口监听

2.使用ServerSocket的accpet()进行等待连接并获得一个Socket，此时双方就建立了一个连接，注意该方法是阻塞方法，（没有客户连接时阻塞。），

3.通过获取到的socket获取输入输出流（获取到的流是基流，可以进行包装）。

4.通过获取到的流进行读取客户端信息和向客户端写信息。

5.释放连接。（注意Socket不一定会被释放）

客户端：

1.新建获得一个Socket，指定IP和端口号。

2.通过获取到的socket获取输入输出流.

3.通过获取到的流进行读取服务端信息和向服务端写信息。

4.释放连接。