**中软国际考题-笔试**

1. 单选题

1、在服务器www.chinaosfti.com上提供了基于TCP的时间服务应用，该应用使用port为6666。创建连接到此服务器的语句是?（A）

Int a=5,b=6,x=6;

If(++a==b) x=++a\*b;

A. Socket s = new Socket("www.chinaosfti.com", 6666);

B. Socket s = new Socket("www.chinaosfti.com:6666");

C. Socket s =Socket.accept("www.chinaosfti.com", 6666);

D. Socket s =Socket.accept("www.chinaosfti.com:6666");

2、TCP连接的建立需要经过几次握手（C）

A.1 B.2 C.3 D.4

3、以下关于socket说法正确的是（B）

A.socket连接是一种短连接

B.socket可以创建upd连接，也可以创建tcp连接

C.socket连接是一种单向连接

D.socket只能使用一种网络协议进行传输

4、网络七层由下往上分别为。（A）

A.物理层、数据链路层、网络层、传输层、会话层、表示层、应用层

B.物理层、应用层、数据链路层、会话层、网络层、传输层、表示层、

C.物理层、应用层、数据链路层、网络层、传输层、会话层、表示层

D.物理层、数据链路层、传输层、会话层、表示层、网络层、应用层

5、**以下协议属于TCP/IP协议栈且其中位于传输层的协议是（  D  ）。**

A. FTP B. HTTP C. SMTP D. UDP

6、下列关于TCP的描述中，不正确的是那个选项（B）

1. TCP是Transmission Control Protocol（传输控制协议）的英文缩写。
2. 无论有无连接该协议皆可适用。
3. TCP仅适用于对可靠性要求高的应用环境中。

TCP使用字节流模式发送数据。

7、UDP无法避免UDP Flood的攻击（B）

A.TGP B. UDP Flood C.CSDN D.SOV

8、TCP包头的最小长度，为20字节 （A）

A.20 B. 16 C.4 D.32

9、网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个。（D）

A.TGP B.BUG C.UPD D. Socket

10、下面关于网络通信正确的描述是（CD）（多选）

a) TCP/IP协议是一种不可靠的网络通信协议。

b) UDP协议是一种可靠的网络通信协议。

c) TCP/IP协议是一种可靠的网络通信协议。

d) UDP协议是一种不可靠的网络通信协议。

11、当使用客户端套接字Socket创建对象时，需要指定（A）

A．服务器主机名称和端口

B、服务器端口和文件

C、服务器名称和文件

D．服务器地址和文件

12.ServerSocket的监听方法accept()的返回值类型是（C）

A. void B.Object C.Socket D.DatagramSocket

13. ( C )类的对象中包含有Internet地址

A．Applet B．Datagramsocket C．InetAddress D．AppletContext

14.当找不到客户的服务器地址时会抛出(B )。

A．IOException B．UnknownHostException C．SocketException

D．MalformedURLExceptin

15.关于 Socket 通信编程，以下描述错误的是：（D）

A.服务器端通过new ServerSocket()创建TCP连接对象

B.服务器端通过TCP连接对象调用accept()方法创建通信的Socket对象

C.客户端通过new Socket()方法创建通信的Socket对象

D.客户端通过new ServerSocket()创建TCP连接对象

16.正常建立一条TCP连接需要（）个步骤，正常关闭一个TCP连接需要（）个步骤

A.3,3

B.3,4

C.4,4

D.4,3

17.关于UDP的说法正确的是(B)

A.UDP的包大小没有限制

B.UDP不会进行错误重传

C.UDP跟TCP一样提供可靠的数据报协议

D.UDP有简单的流控制

1. 多选题
2. Java UDP编程主要用到的两个类是？（B、D）
3. UDPSocket B. DatagramSocket C. UDPPacket
4. DatagramPacket E. 以上都不是
5. 下面哪写协议是基于TCP的？（A、B、C）

A.http B. telnet C. ftp D.dhcp E.bootp

3、下面哪些协议是属于传输层？（C、F）

A IP协议 B. ICMP协议 C. TCP协议

D FTP协议; E. HTTP协议 F. UDP协议

4、建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket ，另一个运行于服务器端，称为ServerSocket 。套接字之间的连接过程分为三个步骤（A、C、D）

A. 服务器监听 B.回复数据 C. 连接确认 D. 客户端请求

5、以下说法中关于UDP协议的说法正确的是（ AD ）。

A.发送不管对方是否准备好，接收方收到也不确认

B.面向连接

C.占用系统资源多、效率低

D.非常简单的协议，可以广播发送

6、下面关于UDP的描述，哪些选项是正确的？（A、B）

1. UDP是面向无连接的协议。
2. UDP的传输是不可靠的，不保证数据正确。
3. UDP使用字节流模式发送数据。

D．UDP适用于对可靠性要求高的应用环境中。

7、 下面哪些是常用socket函数？（A、C、D）

A.bind

B.Font

C.connect

D.accept

E.float

8、以下协议都属于TCP/IP协议栈，其中位于传输层的协议是（ AD ）。（多选）

A.TCP B.HTTP C.SMTP D.UDP

9、以下关于TCP和UDP的描述正确的有（B D）

A.TCP和UDP提供面向连接的传输，通信前要先建立连接（三次握手机制）

B.TCP提供可靠的传输，UDP提供不可靠的传输

C.TCP是面向数据报的传输，UDP是面向字节流的传输

D.TCP提供拥塞控制和流量控制机制；UDP不提供拥塞控制和流量控制机制

1. 填空题（每空1分，共10分）
2. Java中常用的面向连接的传输层协议是\_\_\_\_\_\_\_\_\_\_，面向非连接的协议为\_\_\_\_\_\_\_\_\_\_。(TCP、UDP)
3. OSI七层结构，由低到高分别是物理层、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、传输层、\_\_\_\_\_\_\_\_\_\_、表示层、应用层。(数据链路层，网络层，会话层）
4. UDP（用户数据报协议）是OSI参考模型中\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_协议，提供面向事务的简单不可靠信息传输服务,它的三大特点分别是\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_,\_\_\_\_\_\_\_。(无连接传输层协议,无连接的,使用尽量大努力交付,面向报文)
5. \_\_ IP地址\_\_\_\_用来标志网络中的一个通信实体的地址。通信实体可以是计算机，路由器等。
6. 网络中的每一个计算机都有一个IP地址，IP地址由四个数字表示，包括\_\_\_\_\_\_和\_\_\_\_\_\_两部分。(网络号、主机号)
7. TCP协议是以\_\_\_\_\_\_\_\_\_\_的方式发送数据, UDP协议以\_\_\_\_\_\_\_\_\_\_作为数据传输的载体。（字节流、数据报）
8. Java UDP Socket编程主要用到的两个类是\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_\_ 组成。（DatagramSocket，DatagramPacket）
9. \_\_\_ IP地址\_\_\_\_用来标志网络中的一个通信实体的地址。通信实体可以是计算机，路由器等。
10. 统一资源定位符URL是指向互联网“资源”的指针,由4部分组成：协议、存放资源的主机域名、\_\_端口\_\_\_和资源路径和文件名。
11. URL  是统一资源定位器的简称，它表示Internet上某一资源的地址。在Socket编程中，IP地址用来标志一台计算机，但是一台计算机上可能提供多种应用程序，使用   端口    来区分这些应用程序。
12. 在Java Socket网络编程中，开发基于TCP协议的服务器端程序使用的套接字是     ServerSocket  。
13. 在Java Socket网络编程中，开发基于UDP协议的程序使用的套接字是   DatagramSocket  。
14. tcp连接建立的时候(3 )次握手
15. 判断题
16. UDP是无连接的,在发送数据之前不需要事先建立连接。（ T ）
17. ServerSocket是TCP连接的服务器端套接字对象。（ T ）
18. TCP/IP通讯是当前最为重要的通讯方式，TCP连接是面向连接的、可靠的传输方式。（ T ）
19. TCP传输通信前要经过4次握手机制。 （ F ）
20. UDP提供拥塞控制和流量控制机制。 （ F ）
21. Socket本身也是一种协议。 （ F ）
22. TCP/IP协议族中的Socket类型为流套接字（streamsocket）和数据报套接字(datagramsocket) （ T ）
23. TELNET、FTP、HTTP都是建立在TCP之上的协议 （ T ）
24. TCP的优点是稳定，可靠。 （ T ）
25. 网络访问层的功能是IP地址的映射,以及将IP封装成帧.基于不同硬件类型的网络接口,网络访问层定义了和物理介质的连接. 。（ F ）
26. UDP信息包的标题很短，只有8个字节，相对于TCP的20个字节信息包的额外开销很小（ T ）
27. HTTP协议（HyperText Transfer Protocol，超文本传输协议）是用于从WWW服务器传输超文本到本地浏览器的传输协议。（ T ）
28. TCP协议（Transmission Control Protocol，传输控制协议）是面向连接的协议，在正式收发数据前必须和对方建立可靠的连接，所以速度会慢。（ T ）
29. 基于TCP协议的通讯，客户端使用Socket对象、服务器端使用ServerSocket对象来获取输入流和输出流，两者之间使用IO流对象读写数据进行通讯。（ F ）
30. ServerSocket的accept方法可以同时为多个客户端提供服务。（ F ）
31. 基于TCP协议的网络通信使用java.net包中的Socket和ServerSocket进行支持；Socket被称为“套接字”，用来描述IP地址和端口。（ T ）
32. TCP/IP是个协议组，可分为三个层次：网络层、传输层和应用层（ T ）
33. 接Socket是传输层供给应用层的编程接口，是应用层与传输层之间的桥梁 。(  T  )
34. TCP/IP传输控制协议是Internet的主要协议，定义了计算机和外设进行通信的规则。TCP/IP网络参考模型包括七个层次：应用层、会话层、表示层、传输层、网络层、链路层和物理层。（  F  ）
35. TCP协议一种面向连接的、可靠的、基于字节流的通信协议 。HTTP、FTP、TELNET、SMTP 都是基于TCP协议的应用层协议。（  T  ）
36. UDP协议是一种面向无连接的、可靠的、基于字节流的传输层通信协议，该协议占用系统资源多、效率较低。（  F  ）
37. Java的大部分网络类都包含在Java.applet包中。（F）
38. UDP是面向连接的协议。（F）
39. 进程利用socket建立与其他进程的连接。（T）
40. 服务器在一个端口等待客户的连接。（T）

五、简答题

1、简述tcp和udp的区别？

答案：

TCP：是面向连接的流传输控制协议，具有高可靠性，确保传输数据的正确性，有验证重发机制，因此不会出现丢失或乱序。

UDP：是无连接的数据报服务，不对数据报进行检查与修改，无须等待对方的应答，会出现分组丢失、重复、乱序，但具有较好的实时性，UDP段结构比TCP的段结构简单，因此网络开销也小。

1. 简述如何创建一个TCP客户程序的流程？

答案：

1 建立socket

2 获得I/O流

3 对I/O流进行读写操作

4 关闭I/O流

5 关闭socket

3、简要描述TCP连接建立过程中的握手流程？（5分）

答案：（1）第一步：源主机A的TCP向主机B发出连接请求报文段，其首部中的SYN(同步)标志位应置为1，表示想与目标主机B进行通信，并发送一个同步序列号X(例：SEQ=100)进行同步，表明在后面传送数据时的第一个数据字节的序号是X＋1（即101）。SYN同步报文会指明客户端使用的端口以及TCP连接的初始序号。

（2）第二步：目标主机B的TCP收到连接请求报文段后，如同意，则发回确认。在确认报中应将ACK位和SYN位置1，表示客户端的请求被接受。确认号应为X＋1(图中为101)，同时也为自己选择一个序号Y。

（3）第三步：源主机A的TCP收到目标主机B的确认后要向目标主机B给出确认，其ACK置1，确认号为Y＋1，而自己的序号为X＋1。TCP的标准规定，SYN置1的报文段要消耗掉一个序号。

　　运行客户进程的源主机A的TCP通知上层应用进程，连接已经建立。当源主机A向目标主机B发送第一个数据报文段时，其序号仍为X＋1，因为前一个确认报文段并不消耗序号。

　　当运行服务进程的目标主机B的TCP收到源主机A的确认后，也通知其上层应用进程，连接已经建立。至此建立了一个全双工的连接。

4、请详述tcp断开连接的具体过程（5分）

答案：

第一步：源主机A的应用进程先向其TCP发出连接释放请求，并且不再发送数据。TCP通知对方要释放从A到B这个方向的连接，将发往主机B的TCP报文段首部的终止比特FIN置1，其序号X等于前面已传送过的数据的最后一个字节的序号加1。

第二步：目标主机B的TCP收到释放连接通知后即发出确认，其序号为Y，确认号为X＋1，同时通知高层应用进程，这样，从A到B的连接就释放了，连接处于半关闭状态，相当于主机A向主机B说：“我已经没有数据要发送了。但如果还发送数据，我仍接收。”此后，主机B不再接收主机A发来的数据。但若主机B还有一些数据要发送主机A，则可以继续发送。主机A只要正确收到数据，仍应向主机B发送确认。

第三步：若主机B不再向主机A发送数据，其应用进程就通知TCP释放连接。主机B发出的连接释放报文段必须将终止比特FIN和确认比特ACK置1，并使其序号仍为Y，但还必须重复上次已发送过的ACK＝X＋1。

 第四步：主机A必须对此发出确认，将ACK置1，ACK＝Y＋1，而自己的序号是X＋1。这样才把从B到A的反方向的连接释放掉。主机A的TCP再向其应用进程报告，整个连接已经全部释放。

六、程序实现题

1、网络编程：在MyServerSocket类的main方法中启动服务区器连接，并等待客户端发送消息，接收消息后与客户端进行文字的交互，最后关闭连接。在MyClientSocket类的main方法中启动客户端连接访问服务器连接，在得到服务器返回内容后关闭连接

（10分）

（1）MyServerSocket.java

|  |
| --- |
| import java.io.InputStream;  import java.net.ServerSocket;  import java.net.Socket;  public class MyServerSocket {  public static void main(String[] args) throws Exception {  // 监听指定的端口  int port = 55533;  ServerSocket server = new ServerSocket(port);    // server将一直等待连接的到来  System.out.println("server将一直等待连接的到来");  Socket socket = server.accept();  // 建立好连接后，从socket中获取输入流，并建立缓冲区进行读取  InputStream inputStream = socket.getInputStream();  byte[] bytes = new byte[1024];  int len;  StringBuilder sb = new StringBuilder();  while ((len = inputStream.read(bytes)) != -1) {  //注意指定编码格式，发送方和接收方一定要统一，建议使用UTF-8  sb.append(new String(bytes, 0, len,"UTF-8"));  }  System.out.println("get message from client: " + sb);  inputStream.close();  socket.close();  server.close();  }  } |

（2）MyClientSocket.java

|  |
| --- |
| import java.net.Socket;  import java.io.OutputStream;  public class MyClientSocket {  public static void main(String args[]) throws Exception {  // 要连接的服务端IP地址和端口  String host = "127.0.0.1";  int port = 55533;  // 与服务端建立连接  Socket socket = new Socket(host, port);  // 建立连接后获得输出流  OutputStream outputStream = socket.getOutputStream();  String message="你好 yiwangzhibujian";  socket.getOutputStream().write(message.getBytes("UTF-8"));  outputStream.close();  socket.close();  }  } |

2、使用Socket简单实现客服端和服务器端的数据交互。（10分）

|  |
| --- |
| **服务器端：**  **public** **class** TCPConnection {  **public** **static** **void** main(String[] args) {  **int** port = 8801;  **try** {  ServerSocket server = **new** ServerSocket(port);    Socket socket = server.accept();    InputStream inputStream = socket.getInputStream();    **byte**[] bytes = **new** **byte**[1024];    **int** len;    StringBuffer sb = **new** StringBuffer();    **while**((len = inputStream.read(bytes)) != -1) {  sb.append(**new** String(bytes, 0, len, "utf-8"));  }    System.***out***.println(sb.toString());    inputStream.close();  socket.close();  server.close();      } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  **客户端：**  **public** **class** TCPClientConnection {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  String host = "127.0.0.1";  **int** port = 8801;    **try** {  Socket socket = **new** Socket(host, port);    OutputStream outputStream = socket.getOutputStream();  String message="你好 yiwangzhibujian";  socket.getOutputStream().write(message.getBytes("UTF-8"));  outputStream.close();  socket.close();  } **catch** (UnknownHostException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

3、编写一个网络应用程序，有客户端与服务器端，客户端向服务器发送一个字符串，服务器收到该字符串后将其打印到命令行上，然后向客户端返回该字符串的长度，最后，客户端输出服务器端返回的该字符串的长度（采用TCP）

（10分）

|  |
| --- |
| //服务器端代码public class Server {  public static void main(String[] args) throws Exception {  ServerSocket ss = new ServerSocket(65000);  while (true) {  Socket socket = ss.accept();  new MyThread(socket).start();  }  }  }  class MyThread extends Thread {  private Socket socket;  public MyThread(Socket socket) {  this.socket = socket;  }  @Override  public void run() {  try {  OutputStream os = socket.getOutputStream();  InputStream is = socket.getInputStream();  int ch = 0;  byte[] buff = new byte[1024];  ch = is.read(buff);  String content = new String(buff, 0, ch);  System.out.println(content);  os.write(String.valueOf(content.length()).getBytes());  is.close();  os.close();  socket.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //客户端代码public class Client {  public static void main(String[] args) throws Exception {  Socket socket = new Socket("127.0.0.1", 65000);  OutputStream os = socket.getOutputStream();  InputStream is = socket.getInputStream();  os.write(new String("hello world").getBytes());  int ch = 0;  byte[] buff = new byte[1024];  ch = is.read(buff);  String content = new String(buff, 0, ch);  System.out.println(content);  is.close();  os.close();  socket.close();  }  } |

4 尝试请用UDP通信方式完成上述问题

|  |
| --- |
| //服务端代码  public class UDPServer {  public static void main(String[] args) throws Exception {  // 服务端接受客户端发送的数据报  DatagramSocket socket = new DatagramSocket(65001); //监听的端口号  byte[] buff = new byte[100]; //存储从客户端接受到的内容  DatagramPacket pocket = new DatagramPacket(buff, buff.length);  //接受客户端发送过来的内容，并将内容封装进DatagramPacket对象中  socket.receive(pocket);  byte[] data = pocket.getData(); //从DatagramPacket对象中获取到真正存储的数据  //将数据从二进制转换成字符串形式  String content = new String(data, 0, pocket.getLength());  System.out.println(content);  //将要发送给客户端的数据转换成二进制  byte[] sendedContent = String.valueOf(content.length()).getBytes();  // 服务端给客户端发送数据报  //从DatagramPacket对象中获取到数据的来源地址与端口号  DatagramPacket packetToClient = new DatagramPacket(sendedContent,  sendedContent.length, pocket.getAddress(), pocket.getPort());  socket.send(packetToClient); //发送数据给客户端  }  }  //客户端的代码  public class UDPClient {  public static void main(String[] args) throws Exception {  // 客户端发数据报给服务端  DatagramSocket socket = new DatagramSocket();  // 要发送给服务端的数据  byte[] buf = "Hello World".getBytes();  // 将IP地址封装成InetAddress对象  InetAddress address = InetAddress.getByName("127.0.0.1");  // 将要发送给服务端的数据封装成DatagramPacket对象 需要填写上ip地址与端口号  DatagramPacket packet = new DatagramPacket(buf, buf.length, address,  65001);  // 发送数据给服务端  socket.send(packet);  // 客户端接受服务端发送过来的数据报  byte[] data = new byte[100];  // 创建DtagramPacket对象用来存储服务端发送过来的数据  DatagramPacket receivedPacket = new DatagramPacket(data, data.length);  // 将接受到的数据存储到DatagramPacket对象中  socket.receive(receivedPacket);  // 将服务器端发送过来的数据取出来并打印到控制台  String content = new String(receivedPacket.getData(), 0,  receivedPacket.getLength());  System.out.println(content);  }  } |

5 用socket编写一个服务端与客户端的聊天程序

|  |
| --- |
| //服务器端代码//监听端口，每成功连接一个客户机就开启一个线程进行读写处理public class Server {  public static void main(String[] args) throws Exception {  ServerSocket ss = new ServerSocket(65000);  while (true) {  Socket socket = ss.accept();  new MyThread(socket).start(); //开启一个线程用于处理已连接上的socket  }  }  }  //开启两个线程分别用来处理socket中的输入输出流class MyThread extends Thread {  private Socket socket;  public MyThread(Socket socket) {  this.socket = socket;  }  @Override  public void run() {  try {  //开启两个线程分别来处理服务端的读写  new OutputThread(socket.getOutputStream()).start();  new InputThread(socket.getInputStream()).start();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //处理socket的输出流class OutputThread extends Thread {  private OutputStream os;  public OutputThread(OutputStream os) {  this.os = os;  }  @Override  public void run() {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  String line = null;  try {  while ((line = br.readLine()) != null) {  os.write((line).getBytes());  os.flush();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }  //处理socket的输入流class InputThread extends Thread {  private InputStream is;  public InputThread(InputStream is) {  this.is = is;  }  @Override  public void run() {  int ch = 0;  byte[] buff = new byte[1024];  try {  while ((ch = is.read(buff)) != -1) {  String content = new String(buff, 0, ch);  System.out.println(content);  }  } catch (IOException e) {  e.printStackTrace();  }  }  }  //客户端代码  public class Client {  public static void main(String[] args) throws Exception {  Socket socket = new Socket("127.0.0.1", 65000);  new MyThread(socket).start(); //开启一个线程用于处理已连接上的socket  }  } |

6、使用socket实现一个回声服务器，就是服务器会将客户端发送过来的数据传回给客户端

（10分）

|  |
| --- |
| import java.io.\*;  import java.net.\*;  public class EchoServer {  public static void main(String args[]) {  // declaration section:  // declare a server socket and a client socket for the server  // declare an input and an output stream  ServerSocket echoServer = ;  String line;  DataInputStream is;  PrintStream os;  Socket clientSocket = ;  // Try to open a server socket on port 9999  // Note that we can't choose a port less than 1023 if we are not  // privileged users (root)  try {  echoServer = new ServerSocket(9999);  }  catch (IOException e) {  System.out.println(e);  }  // Create a socket object from the ServerSocket to listen and accept  // connections.  // Open input and output streams  try {  clientSocket = echoServer.accept();  is = new DataInputStream(clientSocket.getInputStream());  os = new PrintStream(clientSocket.getOutputStream());  // As long as we receive data, echo that data back to the client.  while (true) {  line = is.readLine();  os.println(line);  }  } catch (IOException e) {  System.out.println(e);  }  }  } |

7、使用基于TCP的Java Socket编程，完成如下功能：

1)   要求从客户端录入几个字符，发送到服务器端。

2)   由服务器端将接收到的字符进行输出。

3)   服务器端向客户端发出“您的信息已收到”作为响应。

4)   客户端接收服务器端的响应信息。

提示：

服务器端:PrintWriter out =new PrintWriter(socket.getOutputStream(),true);

客户端:BufferedReader line=new BufferedReader(new InputStreamReader(System.in));

|  |
| --- |
| 服务器  public class Server  { // 服务器端类  private ServerSocket ss; // 声明ServerSocket对象  private Socket socket; // 声明Socket对象  private BufferedReader in; // 声明输入流对象BufferedReader  private PrintWriter out; // 声明输出流对象  public Server()  {  try  {  // 创建一个ServerSocket，指定等待端口  ss = new ServerSocket(10000);  while (true)  {  // 使用ServerSocket接收用户请求（处于监听状态）  socket = ss.accept(); // 获得客户端的socket对象  // 创建输入流并读取信息  in = new BufferedReader(new InputStreamReader(socket  .getInputStream()));  String line = in.readLine();  System.out.println("客户端说:" + line);  // 创建输出流并输出信息  out = new PrintWriter(socket.getOutputStream(), true);  out.print("您的信息已接收!");  // 关闭流和socket  out.close();  in.close();  socket.close();  }  } catch (IOException e)  {  e.printStackTrace();  out.println("出错了!");  } // 创建ServerSocket对象  }  public static void main(String[] args)  {// 程序主入口  // 创建对象  new Server();  }  } |
| 客户端  public class Client  { // 声明客户端类  Socket socket; // 声明Socket对象  BufferedReader in; // 声明输入流对象  PrintWriter out; // 声明输出流对象  public Client()  { // 构造方法  try  {  // 创建Socket对象，用来发送请求  socket = new Socket(InetAddress.getLocalHost(), 10000);  // 创建字符缓冲区对象  BufferedReader line = new BufferedReader(new InputStreamReader(  System.in));  // 创建输出流对象并输出  out = new PrintWriter(socket.getOutputStream(), true);  out.println(line.readLine());  // 创建输入流对象  in = new BufferedReader(new InputStreamReader(socket  .getInputStream()));  System.out.println(in.readLine());  // 关闭流对象  out.close();  in.close();  }catch(Exception e){  e.printStackTrace();  System.out.println("出错了!");  }  }  public static void main(String[] args)  {  // 创建对象  new Client();  }  } |