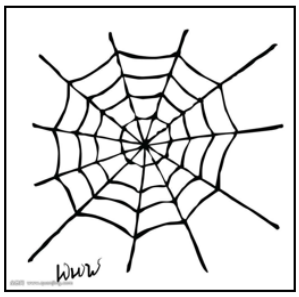
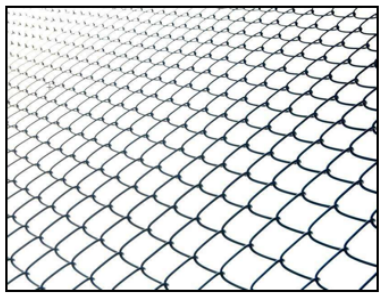
网络编程

一、什么是网络

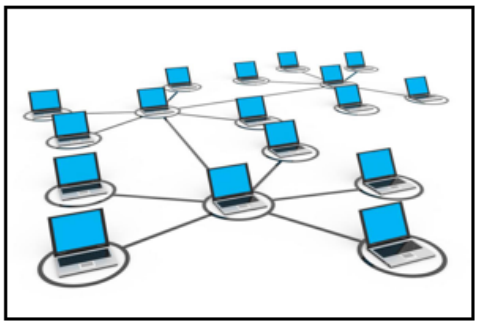
1.1 概念

> 由点和线构成，表示诸多对象间的相互联系。

网络
  

1.2 计算机网络

> 为实现资源共享和信息传递，通过通信线路连接起来的若干主机（Host）。

计算机网络
 

> 常见计算机网络：
>
> - 互联网：（Internet）点与点相连。
> - 万维网：（WWW – World Wide Web）端与端相连。
> - 物联网：（IoT - Internet of things）物与物相连。
> - 网络编程：让计算机与计算机之间建立连接、进行通信。

1.3 网络协议和模型【重点】

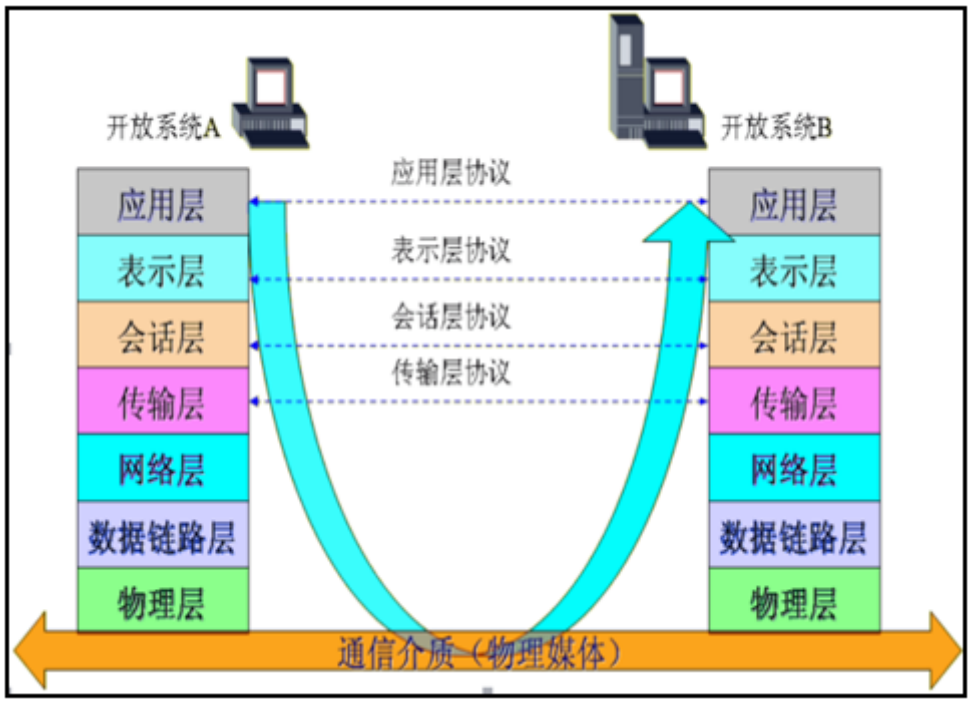
1.3.1 网络协议

- **网络通信协议：**通过计算机网络可以使多台计算机实现连接，位于同一个网络中的计算机在进行连接和通信时需要遵守一定的规则，这就好比在道路中行驶的汽车一定要遵守交通规则一样。在计算机网络中，这些连接和通信的规则被称为网络通信协议，它对数据的传输格式、传输速率、传输步骤等做了统一规定，通信双方必须同时遵守才能完成数据交换。
- **TCP/IP协议：** 传输控制协议/因特网互联协议(Transmission Control Protocol/Internet Protocol)，是Internet最基本、最广泛的协议。它定义了计算机如何连入因特网，以及数据如何在它们之间传输的标准。它的内部包含一系列的用于处理数据通信的协议，并采用了4层的分层模型，每一层都呼叫它的下一层所提供的协议来完成自己的需求。

1.3.2 OSI参考模型

> OSI（Open System Interconnect），即开放式系统互联。
>
> - 是ISO组织在1985年研究的网络互联模型。
> - 该体系结构标准定义了网络互联的七层框架（物理层、数据链路层、网络层、传输层、会话层、表示层和应用层）。

OSI参考模型



上图中，OSI参考模型：模型过于理想化，未能在因特网上进行广泛推广。 TCP/IP参考模型(或TCP/IP协议)：事实上的国际标准。

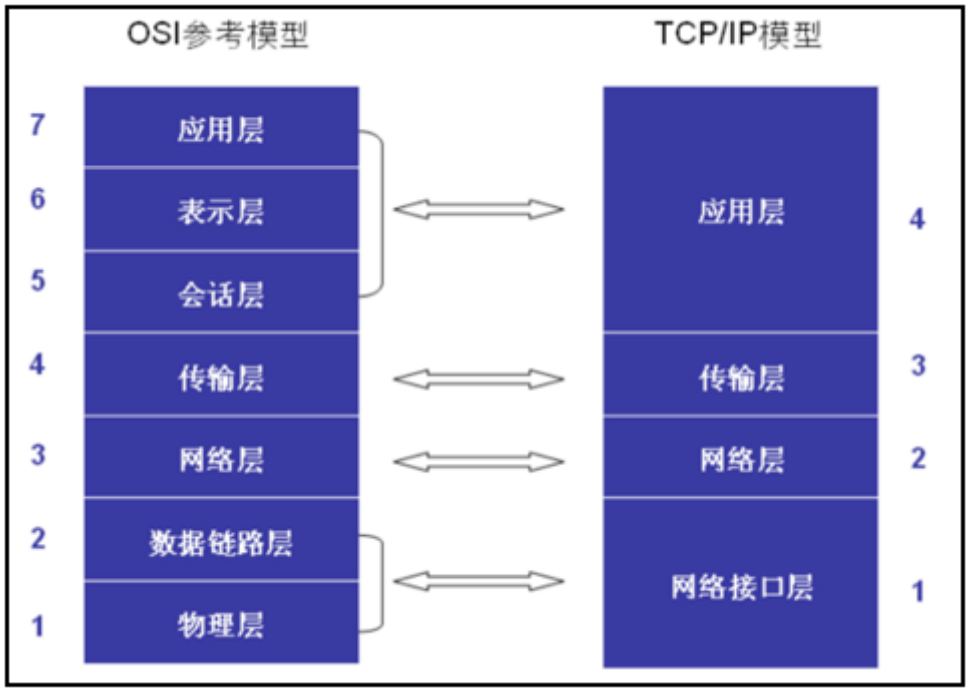
每层功能：

- 第七层：应用层负责文件访问和管理、可靠运输服务、远程操作服务。（HTTP、FTP、SMTP）。
- 第六层：表示层负责定义转换数据格式及加密，允许选择以二进制或ASCII格式传输。
- 第五层：会话层负责使应用建立和维持会话，使通信在失效时继续恢复通信。（断点续传）。
- 第四层：传输层负责是否选择差错恢复协议、数据流重用、错误顺序重排。（TCP、UDP）。
- 第三层：网络层负责定义了能够标识所有网络节点的逻辑地址。（IP地址）。
- 第二层：链路层在物理层上，通过规程或协议（差错控制）来控制传输数据的正确性。（MAC）。
- 第一层：物理层为设备之间的数据通信提供传输信号和物理介质。（双绞线、光导纤维）。

1.3.3 TCP/IP模型

- TCP/IP模型是因特网使用的参考模型，基于TCP/IP的参考模型将协议分成四个层次。
- 该模型中最重要的两个协议是TCP和IP协议。

TCP/IP模型



每层功能：

- 链路层：链路层是用于定义物理传输通道，通常是对某些网络连接设备的驱动协议，例如针对光纤、网线提供的驱动。
- 网络层：网络层是整个TCP/IP协议的核心，它主要用于将传输的数据进行分组，将分组数据发送到目标计算机或者网络。而IP协议是一种非常重要的协议。IP（internet protocal）又称为互联网协议。IP的责任就是把数据从源传送到目的地。它在源地址和目的地址之间传送一种称之为数据包的东西，它还提供对数据大小的重新组装功能，以适应不同网络对包大小的要求。
- 传输层：主要使网络程序进行通信，在进行网络通信时，可以采用TCP协议，也可以采用UDP协议。TCP（Transmission Control Protocol）协议，即传输控制协议，是一种面向连接的、可靠的、基于字节流的传输层通信协议。UDP(User Datagram Protocol，用户数据报协议)：是一个无连接的传输层协议、提供面向事务的简单不可靠的信息传送服务。
- 应用层：主要负责应用程序的协议，例如HTTP协议、FTP协议等。

二、常见协议和概念

2.1 TCP【重点】

TCP协议：Transmission Control Protocol 传输控制协议：

- 是一种面向连接的、可靠的、基于字节流的传输层通信协议。数据大小无限制。建立连接的过程需要三次握手，断开连接的过程需要四次挥手。

2.2 UDP

UDP协议：User Datagram Protocol 用户数据报协议：

- 是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，数据被限制在64kb以内，超出这个范围就不能发送了。

2.3 IP

> IP协议：Internet Protocol Address 互联网协议地址/网际协议地址：
>
> - 分配给互联网设备的数字标签（唯一标识）。

> IP地址版本：
>
> - IPV4：4字节32位整数，并分成4段8位的二进制数，每8位之间用圆点隔开，每8位整数可以转换为一个0~255的十进制整数。
>   格式：D.D.D.D 例如：255.255.255.255
> - IPV6：16字节128位整数，并分成8段十六进制数，每16位之间用圆点隔开，每16位整数可以转换为一个0~65535的十进制数。
>   格式：X.X.X.X.X.X.X.X 例如：FFFF.FFFF.FFFF.FFFF.FFFF.FFFF.FFFF.FFFF

> IP地址分类：
>
> - A类：政府机构，1.0.0.1 ~ 126.255.255.254
> - B类：中型企业，128.0.0.1 ~ 191.255.255.254
> - C类：个人用户，192.0.0.1 ~ 223.255.255.254
> - D类：用于组播，224.0.0.1 ~ 239.255.255.254
> - E类：用于实验，240.0.0.1 ~ 255.255.255.254
> - 回环地址：127.0.0.1，指本机，一般用于测试使用。

> 查看IP命令：ipconfig
> 测试IP命令：ping D.D.D.D

2.4 端口号

> 端口号：在通信实体上进行网络通讯的程序的唯一标识。
> 端口分类：
>
> - 公认端口：0~1023
> - 注册端口：1024~49151
> - 动态或私有端口：49152~65535
>
> 常用端口：
>
> - MySql：3306
> - Oracle：1521
> - Tomcat：8080
> - SMTP：25
> - Web服务器：80
> - FTP服务器：21

三、InetAddress类

> 概念：表示互联网协议（IP）地址对象，封装了与该IP地址相关的所有信息，并提供获取信息的常用方法。

> InetAddress类主要表示IP地址，两个子类：Inet4Address、Inet6Address。

> Internet上的主机有两种方式表示地址：
>
> - 域名(hostName)：www.atguigu.com
> - IP 地址(hostAddress)：202.108.35.210
>
> lInetAddress 类没有提供公共的构造器，而是提供 了 如下几个 静态方法来获取InetAddress 实例
>
> - public static InetAddress getLocalHost()
> - public static InetAddress getByName(String host)
> - public static InetAddress getByAddress(byte[] addr)
>
> InetAddress 提供了如下几个常用的方法
>
> - public String getHostAddress()： 返回 IP 地址字符串（以文本表现形式）。
> - public String getHostName()： 获取此 IP 地址的主机名

方法名	描述
public static InetAddress getLocalHost()	获得本地主机地址对象
public static InetAddress getByName(String host)	根据主机名称获得地址对象
public static InetAddress[] getAllByName(String host)	获得所有相关地址对象
public String getHostAddress()	获取IP地址字符串
public String getHostName()	获得IP地址主机名

> 案例演示：

```
import java.net.InetAddress;
import java.net.UnknownHostException;

import org.junit.Test;

public class TestInetAddress {
    @Test
    public void test01() throws UnknownHostException{
        InetAddress localHost = InetAddress.getLocalHost();
        System.out.println(localHost);
    }

    @Test
    public void test02()throws UnknownHostException{
        InetAddress atguigu = InetAddress.getByName("www.atguigu.com"/IP地址);
        System.out.println(atguigu);
    }

}
```

四、Socket类

> 通信的两端都要有Socket（也可以叫“套接字”），是两台机器间通信的端点。网络通信其实就是Socket间的通信。

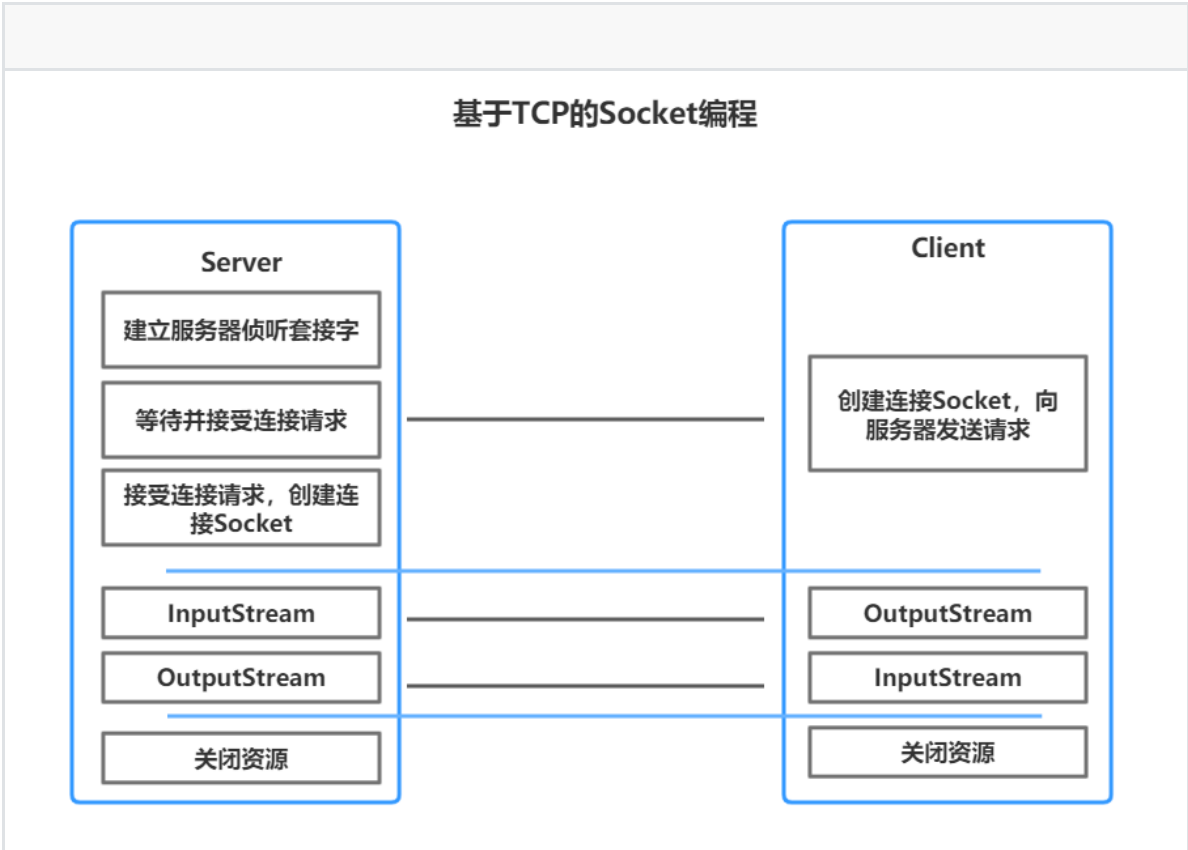
Socket可以分为：

- 流套接字（stream socket）：使用TCP提供可依赖的字节流服务
  - ServerSocket：此类实现TCP服务器套接字。服务器套接字等待请求通过网络传入。
  - Socket：此类实现客户端套接字（也可以就叫“套接字”）。套接字是两台机器间通信的端点。
- 数据报套接字（datagram socket）：使用UDP提供“尽力而为”的数据报服务
  - DatagramSocket：此类表示用来发送和接收UDP数据报包的套接字。

五、基于TCP的网络编程【重点】

> Socket编程：
>
> - Socket（套接字）是网络中的一个通信节点。
> - 分为客户端Socket与服务器ServerSocket。
> - 通信要求：IP地址 + 端口号。

5.1 通信步骤



> 服务器端
>
> 服务器 程序的工作过程包含以下四个基本的 步骤：

- 调用 ServerSocket(int port) ： 创建一个服务器端套接字，并绑定到指定端口上。用于监听客户端的请求。
- 调用 accept()： 监听连接请求，如果客户端请求连接，则接受连接，返回通信套接字对象。
- 调用 该Socket 类对象的 getOutputStream() 和 getInputStream ()： 获取输出流和输入流，开始网络数据的发送和接收。
- 关闭Socket 对象：客户端访问结束，关闭通信套接字。

客户端

客户端Socket 的工作过程包含以下四个基本的步骤：

- 创建 Socket： 根据指定服务端的 IP 地址或端口号构造 Socket 类对象。若服务器端响应，则建立客户端到服务器的通信线路。若连接失败，会出现异常。
- 打开连接到 Socket 的输入/ 出流： 使用 getInputStream()方法获得输入流，使用 getOutputStream()方法获得输出流，进行数据传输
- 按照一定的协议对 Socket 进行读/ 写操作：通过输入流读取服务器放入线路的信息（但不能读取自己放入线路的信息），通过输出流将信息写入线路。
- 关闭 Socket： 断开客户端到服务器的连接，释放线路

5.2 常用API

ServerSocket类的构造方法：

- ServerSocket(int port)： 创建绑定到特定端口的服务器套接字。

ServerSocket类的常用方法：

- Socket accept()：侦听并接受到此套接字的连接。

Socket类的常用构造方法：

- public Socket(InetAddress address,int port)：创建一个流套接字并将其连接到指定 IP 地址的指定端口号。
- public Socket(String host,int port)：创建一个流套接字并将其连接到指定主机上的指定端口号。

Socket类的常用方法：

- public InputStream getInputStream()：返回此套接字的输入流，可以用于接收消息
- public OutputStream getOutputStream()：返回此套接字的输出流，可以用于发送消息
- public void close()：关闭此套接字。套接字被关闭后，便不可在以后的网络连接中使用（即无法重新连接或重新绑定）。需要创建新的套接字对象。 关闭此套接字也将会关闭该套接字的 InputStream 和 OutputStream。
- public void shutdownInput()：如果在套接字上调用 shutdownInput() 后从套接字输入流读取内容，则流将返回 EOF（文件结束符）。 即不能在从此套接字的输入流中接收任何数据。
- public void shutdownOutput()：禁用此套接字的输出流。对于 TCP 套接字，任何以前写入的数据都将被发送，并且后跟 TCP 的正常连接终止序列。 如果在套接字上调用 shutdownOutput() 后写入套接字输出流，则该流将抛出 IOException。 即不能通过此套接字的输出流发送任何数据。

注意：先后调用Socket的shutdownInput()和shutdownOutput()方法，仅仅关闭了输入流和输出流，并不等于调用Socket的close()方法。在通信结束后，仍然要调用Scoket的close()方法，因为只有该方法才会释放Socket占用的资源，比如占用的本地端口号等。

5.3 演示单连接TCP

> 案例演示1：TCP编程实现客户端发送数据给服务器端。
>
> 需求：客户端连接服务器，连接成功后给服务发送“lalala”，服务器收到消息后，给客户端返回“欢迎登录”，客户端接收消息后，断开连接

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    public static void main(String[] args)throws Exception {
        //1、准备一个ServerSocket对象，并绑定8888端口
        ServerSocket server =  new ServerSocket(8888);
        System.out.println("等待连接....");

    //2、在8888端口监听客户端的连接，该方法是个阻塞的方法，如果没有客户端连接，将一直等待
        Socket socket = server.accept();
        System.out.println("一个客户端连接成功！！");

        //3、获取输入流，用来接收该客户端发送给服务器的数据
        InputStream input = socket.getInputStream();
        //接收数据
        byte[] data = new byte[1024];
        StringBuilder s = new StringBuilder();
        int len;
        while ((len = input.read(data)) != -1) {
            s.append(new String(data, 0, len));
        }
        System.out.println("客户端发送的消息是：" + s);

        //4、获取输出流，用来发送数据给该客户端
        OutputStream out = socket.getOutputStream();
        //发送数据
        out.write("欢迎登录".getBytes());
        out.flush();

        //5、关闭socket，不再与该客户端通信
        //socket关闭，意味着InputStream和OutputStream也关闭了
        socket.close();

        //6、如果不再接收任何客户端通信，可以关闭ServerSocket
        server.close();
    }
}
```

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Client {

    public static void main(String[] args) throws Exception {
        // 1、准备Socket，连接服务器，需要指定服务器的IP地址和端口号
```

```
        Socket socket = new Socket("127.0.0.1", 8888);

        // 2、获取输出流，用来发送数据给服务器
        OutputStream out = socket.getOutputStream();
        // 发送数据
        out.write("lalala".getBytes());
        //会在流末尾写入一个“流的末尾”标记，对方才能读到-1，否则对方的读取方法会一致阻塞
        socket.shutdownOutput();

        //3、获取输入流，用来接收服务器发送给该客户端的数据
        InputStream input = socket.getInputStream();
        // 接收数据
        byte[] data = new byte[1024];
        StringBuilder s = new StringBuilder();
        int len;
        while ((len = input.read(data)) != -1) {
            s.append(new String(data, 0, len));
        }
        System.out.println("服务器返回的消息是：" + s);

        //4、关闭socket，不再与服务器通信，即断开与服务器的连接
        //socket关闭，意味着InputStream和OutputStream也关闭了
        socket.close();
    }
}
```

5.4 演示多客户端通信

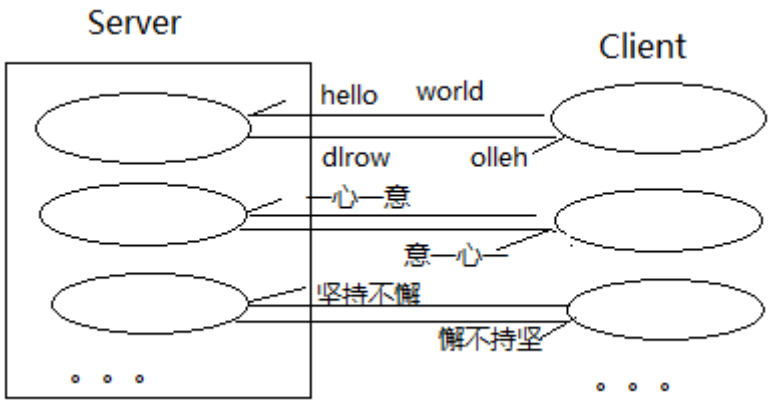
通常情况下，服务器不应该只接受一个客户端请求，而应该不断地接受来自客户端的所有请求，所以Java程序通常会通过循环，不断地调用ServerSocket的accept()方法。

如果服务器端要“同时”处理多个客户端的请求，因此服务器端需要为**每一个客户端单独分配一个线程**来处理，否则无法实现“同时”。

咱们之前学习IO流的时候，提到过装饰者设计模式，该设计使得不管底层IO流是怎样的节点流：文件流也好，网络Socket产生的流也好，程序都可以将其包装成处理流，甚至可以多层包装，从而提供更多方便的处理。

案例需求：多个客户端连接服务器，并进行多次通信

- 每一个客户端连接成功后，从键盘输入英文单词或中国成语，并发送给服务器
- 服务器收到客户端的消息后，把词语“反转”后返回给客户端
- 客户端接收服务器返回的“词语”，打印显示
- 当客户端输入“stop”时断开与服务器的连接
- 多个客户端可以同时给服务器发送“词语”，服务器可以“同时”处理多个客户端的请求



客户端

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;

public class Client {
    public static void main(String[] args) throws Exception {
        // 1、准备Socket，连接服务器，需要指定服务器的IP地址和端口号
        Socket socket = new Socket("127.0.0.1", 8888);

        // 2、获取输出流，用来发送数据给服务器
        OutputStream out = socket.getOutputStream();
        PrintStream ps = new PrintStream(out);

        // 3、获取输入流，用来接收服务器发送给该客户端的数据
        InputStream input = socket.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(input));

        Scanner scanner = new Scanner(System.in);
        while(true){
            System.out.println("输入发送给服务器的单词或成语：");
            String message = scanner.nextLine();
            if(message.equals("stop")){
                socket.shutdownOutput();
                break;
            }

            // 4、 发送数据
            ps.println(message);
            // 接收数据
            String feedback  = br.readLine();
            System.out.println("从服务器收到的反馈是：" + feedback);
        }

        //5、关闭socket，断开与服务器的连接
        scanner.close();
        socket.close();
    }
}
```

服务端

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server2 {
    public static void main(String[] args) throws IOException {
        // 1、准备一个ServerSocket
        ServerSocket server = new ServerSocket(8888);
        System.out.println("等待连接...");

        int count = 0;
        while(true){
            // 2、监听一个客户端的连接
            Socket socket = server.accept();
            System.out.println("第" + ++count + "个客户
端"+socket.getInetAddress().getHostAddress()+"连接成功！！");

            ClientHandlerThread ct = new ClientHandlerThread(socket);
            ct.start();
        }

        //这里没有关闭server，永远监听
    }
    static class ClientHandlerThread extends Thread{
        private Socket socket;

        public ClientHandlerThread(Socket socket) {
            super();
            this.socket = socket;
        }

        public void run(){

            try{
                //（1）获取输入流，用来接收该客户端发送给服务器的数据
                BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                //（2）获取输出流，用来发送数据给该客户端
                PrintStream ps = new PrintStream(socket.getOutputStream());
                String str;
                //  （3）接收数据
                while ((str = br.readLine()) != null) {
                    //（4）反转
                    StringBuilder word = new StringBuilder(str);
                    word.reverse();

                    //（5）返回给客户端
                    ps.println(word);
                }
                System.out.println(socket.getInetAddress().getHostAddress()+"正常
退出");
            }catch(Exception  e){
```

```
                System.out.println(socket.getInetAddress().getHostAddress()+"意外
退出");
            }finally{
                try {
                    //（6）断开连接
                    socket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

六、基于UDP的网络编程【了解】

UDP(User Datagram Protocol，用户数据报协议)：是一个无连接的传输层协议、提供面向事务的简单不可靠的信息传送服务，类似于短信。

UDP协议是一种**面向非连接**的协议，面向非连接指的是在正式通信前不必与对方先建立连接，不管对方状态就直接发送，至于对方是否可以接收到这些数据内容，UDP协议无法控制，因此说，UDP协议是一种**不可靠的**协议。无连接的好处就是快，省内存空间和流量，因为维护连接需要创建大量的数据结构。UDP会尽最大努力交付数据，但不保证可靠交付，没有TCP的确认机制、重传机制，如果因为网络原因没有传送到对端，UDP也不会给应用层返回错误信息。

UDP协议是面向数据报文的信息传送服务。UDP在发送端没有缓冲区，对于应用层交付下来的报文在添加了首部之后就直接交付于ip层，不会进行合并，也不会进行拆分，而是一次交付一个完整的报文。比如我们要发送100个字节的报文，我们调用一次send()方法就会发送100字节，接收方也需要用receive()方法一次性接收100字节，不能使用循环每次获取10个字节，获取十次这样的做法。

UDP协议没有拥塞控制，所以当网络出现的拥塞不会导致主机发送数据的速率降低。虽然UDP的接收端有缓冲区，但是这个缓冲区只负责接收，并不会保证UDP报文的到达顺序是否和发送的顺序一致。因为网络传输的时候，由于网络拥塞的存在是很大的可能导致先发的报文比后发的报文晚到达。如果此时缓冲区满了，后面到达的报文将直接被丢弃。这个对实时应用来说很重要，比如：视频通话、直播等应用。

因此UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境，数据报大小限制在64K以下。

6.1 相关API

基于UDP协议的网络编程仍然需要在通信实例的两端各建立一个Socket，但这两个Socket之间并没有虚拟链路，这两个Socket只是发送、接收数据报的对象，Java提供了DatagramSocket对象作为基于UDP协议的Socket，使用DatagramPacket代表DatagramSocket发送、接收的数据报。

DatagramSocket 类的常用方法：

- public DatagramSocket(int port)创建数据报套接字并将其绑定到本地主机上的指定端口。套接字将被绑定到通配符地址，IP 地址由内核来选择。
- public DatagramSocket(int port,InetAddress laddr)创建数据报套接字，将其绑定到指定的本地地址。本地端口必须在 0 到 65535 之间（包括两者）。如果 IP 地址为 0.0.0.0，套接字将被绑定到通配符地址，IP 地址由内核选择。
- public void close()关闭此数据报套接字。
- public void send(DatagramPacket p)从此套接字发送数据报包。DatagramPacket 包含的信息指示：将要发送的数据、其长度、远程主机的 IP 地址和远程主机的端口号。

- public void receive(DatagramPacket p)从此套接字接收数据报包。当此方法返回时，DatagramPacket 的缓冲区填充了接收的数据。数据报包也包含发送方的 IP 地址和发送方机器上的端口号。 此方法在接收到数据报前一直阻塞。数据报包对象的 length 字段包含所接收信息的长度。如果信息比包的长度长，该信息将被截短。

DatagramPacket类的常用方法：

- public DatagramPacket(byte[] buf,int length)构造 DatagramPacket，用来接收长度为 length 的数据包。 length 参数必须小于等于 buf.length。
- public DatagramPacket(byte[] buf,int length,InetAddress address,int port)构造数据报包，用来将长度为 length 的包发送到指定主机上的指定端口号。length 参数必须小于等于 buf.length。
- public int getLength()返回将要发送或接收到的数据的长度。

6.2 UDP代码演示

发送端

```
package com.atguigu.udp;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.ArrayList;

public class Send {

    public static void main(String[] args)throws Exception {
//      1、建立发送端的DatagramSocket
        DatagramSocket ds = new DatagramSocket();

        //要发送的数据
        ArrayList<String> all = new ArrayList<String>();
        all.add("尚硅谷让天下没有难学的技术！");
        all.add("学高端前沿的IT技术来尚硅谷！");
        all.add("尚硅谷让你的梦想变得更具体！");
        all.add("尚硅谷让你的努力更有价值！");

        //接收方的IP地址
        InetAddress ip = InetAddress.getByName("127.0.0.1");
        //接收方的监听端口号
        int port = 9999;
         //发送多个数据报
        for (int i = 0; i < all.size(); i++) {
//          2、建立数据包DatagramPacket
            byte[] data = all.get(i).getBytes();
            DatagramPacket dp = new DatagramPacket(data, data.length, ip, port);
//          3、调用Socket的发送方法
            ds.send(dp);
        }

//      4、关闭Socket
        ds.close();
    }
}
```

接收方

```
package com.atguigu.udp;

import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class Receive {

    public static void main(String[] args) throws Exception {
//      1、建立接收端的DatagramSocket，需要指定本端的监听端口号
        DatagramSocket ds = new DatagramSocket(9999);

        //一直监听数据
        while(true){
    //      2、建立数据包DatagramPacket
            byte[] buffer = new byte[1024*64];
            DatagramPacket dp = new DatagramPacket(buffer , buffer.length);

    //      3、调用Socket的接收方法
            ds.receive(dp);

            //4、拆封数据
            String str = new String(buffer,0,dp.getLength());
            System.out.println(str);
        }
    }
}
```

> LoginThread类：

```
public class LoginThread extends Thread {
    @Override
```

```
    public void run() {
        try {
            //1创建Serversocket
            ServerSocket listener=new ServerSocket(7777);
            //2调用accept方法
            System.out.println("登录服务器已启动......");
            Socket socket=listener.accept();
            //3获取输入输出流
            BufferedReader br=new BufferedReader(new
InputStreamReader(socket.getInputStream(),"utf-8"));
            BufferedWriter bw=new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream(),"utf-8"));
            //4接收客户端发送的数据{id : 1001, pwd :123}
            String json=br.readLine();
            //id : 1001 pwd :123
            String[] infos=json.substring(1, json.length()-1).split(",");
            String id=infos[0].split(":")[1];
            //5加载属性文件
            Properties properties=Tools.loadProperties();
            //6判断是否存在
            if(properties.containsKey(id)) {
                //判断密码是否正确
                String pwd=infos[1].split(":")[1];
                String value=properties.getProperty(id);
                String[] arr=value.substring(1, value.length()-1).split(",");
                String pwd2=arr[2].split(":")[1];
                if(pwd.equals(pwd2)) {
                    bw.write("登录成功");
                }else {
                    bw.write("密码错误");
                }

            }else {
                //保存属性文件
                bw.write("用户名或密码错误");
            }
            bw.newLine();
            bw.flush();
            bw.close();
            br.close();
            socket.close();
            listener.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

> Tools工具类:

```
public class Tools {
    //1加载属性文件
    public static Properties loadProperties() {
        //1创建属性集合
        Properties properties=new Properties();
        //2判断文件是否存在
```

```
        File file=new File("users.properties");
        if(file.exists()) {
            FileInputStream fis=null;
            try {
                fis = new FileInputStream(file);
                properties.load(fis);
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                if(fis!=null) {
                    try {
                        fis.close();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }

        }

        return properties;
    }

    //2保存属性文件

    public static void saveProperties(String json) {
        String[] infos=json.substring(1, json.length()-1).split(",");
        String id=infos[0].split(":")[1];
        //保存
        FileOutputStream fos=null;
        try {
            fos=new FileOutputStream("users.properties",true);
            Properties properties=new Properties();
            properties.setProperty(id, json);
            properties.store(fos, "");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            if(fos!=null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }

    }
}
```

> UserServer类:

```
public class UserServer {
    public static void main(String[] args) {
        new RegistThread().start();
        new LoginThread().start();
    }
}
```

> UserClient类:

```
public class UserClient {
    public static void main(String[] args) throws Exception {
        System.out.println("---------请选择 1 注册 2 登录-----------");
        Scanner input = new Scanner(System.in);
        int choice = input.nextInt();
        switch (choice) {
        case 1:
            regist();
            break;
        case 2:
            login();
        default:
            break;
        }
    }

    public static void regist() throws Exception {
        // 1创建Socket
        Socket socket = new Socket("192.168.0.103", 6666);
        // 2获取流
        BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream(), "utf-8"));
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream(), "utf-8"));
        // 3获取用户信息
        String json = getRegistInfo();
        // 4发送
        bw.write(json);
        bw.newLine();
        bw.flush();
        // 5接收
        String reply = br.readLine();
        System.out.println("服务器回复:" + reply);
        // 6关闭
        bw.close();
        br.close();
        socket.close();
    }

    public static String getRegistInfo() {
        Scanner input = new Scanner(System.in);
        System.out.println("请输入用户编号");
        int id = input.nextInt();
        System.out.println("请输入姓名");
        String name = input.next();
        System.out.println("请输入密码");
```

```
        String pwd = input.next();
        System.out.println("请输入年龄");
        int age = input.nextInt();
        // {id : 1001, name :tom, pwd :123, age : 20 }
        String json = "{id:" + id + ",name:" + name + ",pwd:" + pwd + ",age:" +
pwd + "}";
        return json;
    }

    public static void login() throws Exception {
        // 1创建Socket
        Socket socket = new Socket("192.168.0.103", 7777);
        // 2获取流
        BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream(), "utf-8"));
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream(), "utf-8"));
        // 3获取用户信息
        String json = getLoginInfo();
        // 4发送
        bw.write(json);
        bw.newLine();
        bw.flush();
        // 5接收
        String reply = br.readLine();
        System.out.println("服务器回复:" + reply);
        // 6关闭
        bw.close();
        br.close();
        socket.close();
    }
    public static String getLoginInfo() {
        Scanner input = new Scanner(System.in);
        System.out.println("请输入用户编号");
        int id = input.nextInt();
        System.out.println("请输入密码");
        String pwd = input.next();
        // {id : 1001, name :tom, pwd :123, age : 20 }
        String json = "{id:" + id+",pwd:"+ pwd+"}";
        return json;
    }
}
```