网络编程

今日内容介绍

- 网络通信协议
- UDP 通信
- TCP 通信

第1章 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。

网络通信协议有很多种，目前应用最广泛的是 TCP/IP 协议(Transmission Control Protocal/Internet Protoal 传输控制协议/英特网互联协议)，它是一个包括 TCP 协议和 IP 协议，UDP（User Datagram Protocol）协议和其它一些协议的协议组，在学习具体协议之前首先了解一下 TCP/IP 协议组的层次结构。

在进行数据传输时，要求发送的数据与收到的数据完全一样，这时，就需要在原有的数据上添加很多信息，以保证数据在传输过程中数据格式完全一致。TCP/IP 协议的层次结构比较简单，共分为四层，如图所示。

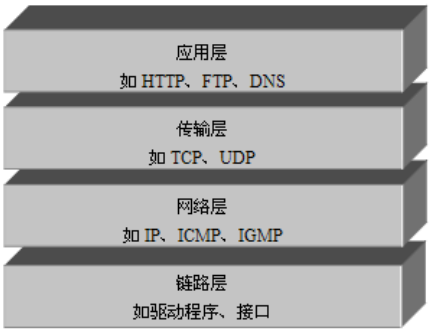


图1-1　TCP/IP 网络模型

上图中，TCP/IP 协议中的四层分别是应用层、传输层、网络层和链路层，每层分别负责不同的通信功能，接下来针对这四层进行详细地讲解。

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。

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。

应用层：主要负责应用程序的协议，例如 HTTP 协议、FTP 协议等。

1.1IP 地址和端口号

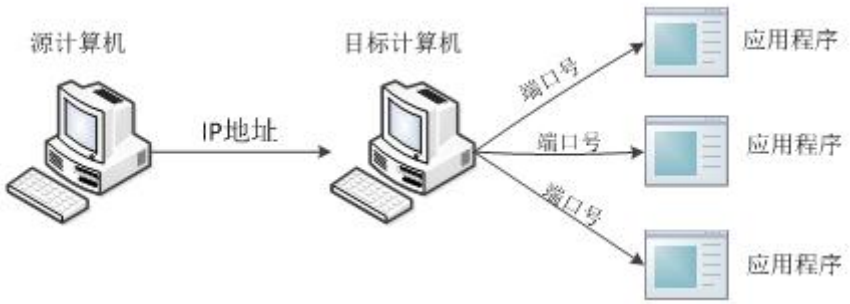
要想使网络中的计算机能够进行通信，必须为每台计算机指定一个标识号，通过这个标识号来指定接受数据的计算机或者发送数据的计算机。

在 TCP/IP 协议中，这个标识号就是 IP 地址，它可以唯一标识一台计算机，目前，IP 地址广泛使用的版本是 IPv4，它是由 4 个字节大小的二进制数来表示，如：00001010000000000000000000000001。由于二进制形式表示的 IP 地址非常不便记忆和处理，因此通常会将 IP 地址写成十进制的形式，每个字节用一个十进制数字(0-255)表示，数字间用符号“.”分开，如 “192.168.1.100”。

随着计算机网络规模的不断扩大，对 IP 地址的需求也越来越多，IPV4 这种用 4 个字节表示的 IP 地址面临枯竭，因此 IPv6 便应运而生了，IPv6 使用 16 个字节表示 IP 地址，它所拥有的地址容量约是 IPv4 的 8×10^{28} 倍，达到 2^{128} 个（算上全零的），这样就解决了网络地址资源数量不够的问题。

通过 IP 地址可以连接到指定计算机，但如果想访问目标计算机中的某个应用程序，还需要指定端口号。在计算机中，不同的应用程序是通过端口号区分的。端口号是用两个字节（16 位的二进制数）表示的，它的取值范围是 0~65535，其中，0~1023 之间的端口号用于一些知名的网络服务和应用，用户的普通应用程序需要使用 1024 以上的端口号，从而避免端口号被另外一个应用或服务所占用。

接下来通过一个图例来描述 IP 地址和端口号的作用，如下图所示。



从上图中可以清楚地看到，位于网络中一台计算机可以通过 IP 地址去访问另一台计算机，并通过端口号访问目标计算机中的某个应用程序。

1.2InetAddress

了解了 IP 地址的作用，我们看学习下 JDK 中提供了一个 InetAdderss 类，该类用于封装一个 IP 地址，并提供了一系列与 IP 地址相关的方法，下表中列出了 InetAddress 类的一些常用方法。

static InetAddress	**getByName**(String host) 在给定主机名的情况下确定主机的 IP 地址。
static InetAddress	**getLocalHost**() 返回本地主机。
String	**getHostName**() 获取此 IP 地址的主机名。
String	**getHostAddress**() 返回 IP 地址字符串（以文本表现形式）。

上图中，列举了 InetAddress 的四个常用方法。其中，前两个方法用于获得该类的实例对象，第

一个方法用于获得表示指定主机的 InetAddress 对象，第二个方法用于获得表示本地的 InetAddress 对象。通过 InetAddress 对象便可获取指定主机名，IP 地址等，接下来通过一个案例来演示 InetAddress 的常用方法，如下所示。

```
public class Example01 {
    public static void main(String[] args) throws Exception {
        InetAddress local = InetAddress.getLocalHost();
        InetAddress remote = InetAddress.getByName("www.easthome.cn");
        System.out.println("本机的 IP 地址：" + local.getHostAddress());
        System.out.println("easthome 的 IP 地址：" + remote.getHostAddress());
        System.out.println("easthome 的主机名为：" + remote.getHostName());
    }
}
```

第2章 UDP 与 TCP 协议

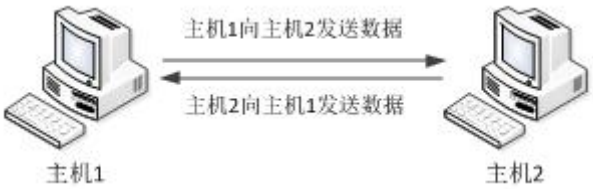
在介绍 TCP/IP 结构时，提到传输层的两个重要的高级协议，分别是 UDP 和 TCP，其中 UDP 是 User Datagram Protocol 的简称，称为用户数据报协议，TCP 是 Transmission Control Protocol 的简称，称为传输控制协议。

2.1 UDP 协议

UDP 是无连接通信协议，即在数据传输时，数据的发送端和接收端不建立逻辑连接。简单来说，当一台计算机向另外一台计算机发送数据时，发送端不会确认接收端是否存在，就会发出数据，同样接收端在收到数据时，也不会向发送端反馈是否收到数据。

由于使用 UDP 协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输例如视频会议都使用 UDP 协议，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

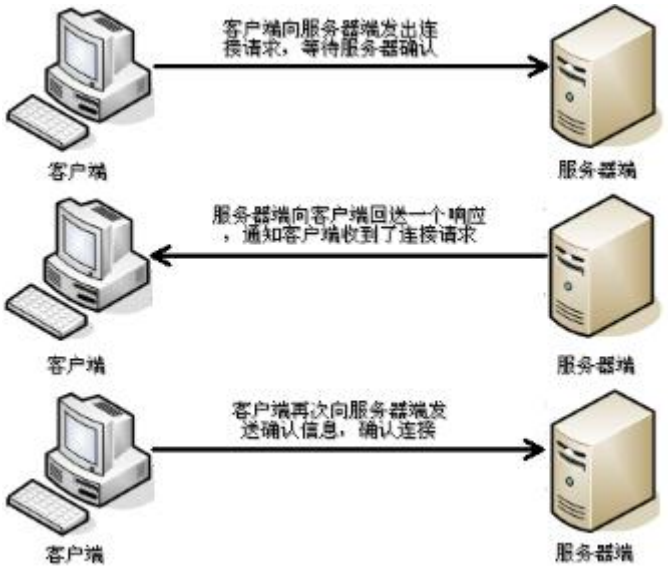
但是在使用 UDP 协议传送数据时，由于 UDP 的面向无连接性，不能保证数据的完整性，因此在传输重要数据时不建议使用 UDP 协议。UDP 的交换过程如下图所示。



2.2 TCP 协议

TCP 协议是面向连接的通信协议，即在传输数据前先在发送端和接收端建立逻辑连接，然后再传输数据，它提供了两台计算机之间可靠无差错的数据传输。在 TCP 连接中必须要明确客户端与服务器端，由客户端向服务端发出连接请求，每次连接的创建都需要经过“三次握手”。第一次握手，客户端向服务器端发出连接请求，等待服务器确认，第二次握手，服务器端向客户端回送一个响应，

通知客户端收到了连接请求，第三次握手，客户端再次向服务器端发送确认信息，确认连接。整个交互过程如下图所示。



由于 TCP 协议的面向连接特性，它可以保证传输数据的安全性，所以是一个被广泛采用的协议，例如在下载文件时，如果数据接收不完整，将会导致文件数据丢失而不能被打开，因此，下载文件时必须采用 TCP 协议。

第3章 UDP 通信

3.1 DatagramPacket

前面介绍了 UDP 是一种面向无连接的协议，因此，在通信时发送端和接收端不用建立连接。UDP 通信的过程就像是货运公司在两个码头间发送货物一样。在码头发送和接收货物时都需要使用集装箱来装载货物，UDP 通信也是一样，发送和接收的数据也需要使用“集装箱”进行打包，为此 JDK 中提供了一个 DatagramPacket 类，该类的实例对象就相当于一个集装箱，用于封装 UDP 通信中发送或者接收的数据。

想要创建一个 DatagramPacket 对象，首先需要了解一下它的构造方法。在创建发送端和接收端的 DatagramPacket 对象时，使用的构造方法有所不同，接收端的构造方法只需要接收一个字节数组来存放接收到的数据，而发送端的构造方法不但要接收存放了发送数据的字节数组，还需要指定发送端 IP 地址和端口号。

接下来根据 API 文档的内容，对 DatagramPacket 的构造方法进行逐一详细地讲解。

```
DatagramPacket(byte[] buf, int length)
        构造 DatagramPacket，用来接收长度为 length 的数据包。
```

使用该构造方法在创建 DatagramPacket 对象时，指定了封装数据的字节数组和数据的大小，没有指定 IP 地址和端口号。很明显，这样的对象只能用于接收端，不能用于发送端。因为发送端一定要明确指出数据的目的地(ip 地址和端口号)，而接收端不需要明确知道数据的来源，只需要接收到数据即可。

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。	

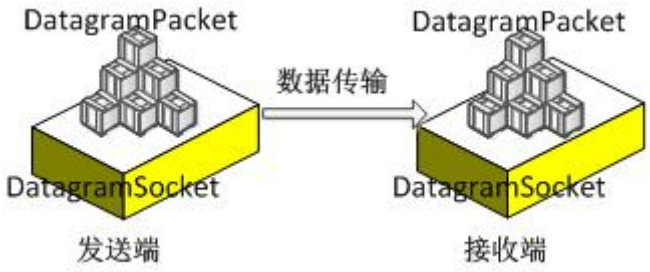
使用该构造方法在创建 DatagramPacket 对象时，不仅指定了封装数据的字节数组和数据的大小，还指定了数据包的目标 IP 地址（addr）和端口号（port）。该对象通常用于发送端，因为在发送数据时必须指定接收端的 IP 地址和端口号，就好像发送货物的集装箱上面必须标明接收人的地址一样。

上面我们讲解了 DatagramPacket 的构造方法，接下来对 DatagramPacket 类中的常用方法进行详细地讲解，如下表所示。

InetAddress	getAddress() 返回某台机器的 IP 地址，此数据报将要发往该机器或者是从该机器接收到的。
int	getPort() 返回某台远程主机的端口号，此数据报将要发往该主机或者是从该主机接收到的。
byte[]	getData() 返回数据缓冲区。
int	getLength() 返回将要发送或接收到的数据的长度。

3.2 DatagramSocket

DatagramPacket 数据包的作用就如同是“集装箱”，可以将发送端或者接收端的数据封装起来。然而运输货物只有“集装箱”是不够的，还需要有码头。在程序中需要实现通信只有 DatagramPacket 数据包也同样不行，为此 JDK 中提供的一个 DatagramSocket 类。DatagramSocket 类的作用就类似于码头，使用这个类的实例对象就可以发送和接收 DatagramPacket 数据包，发送数据的过程如下图所示。



在创建发送端和接收端的 DatagramSocket 对象时，使用的构造方法也有所不同，下面对 DatagramSocket 类中常用的构造方法进行讲解。

	DatagramSocket() 构造数据报套接字并将其绑定到本地主机上任何可用的端口。

该构造方法用于创建发送端的 DatagramSocket 对象，在创建 DatagramSocket 对象时，并没有指定端口号，此时，系统会分配一个没有被其它网络程序所使用的端口号。

	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。

该构造方法既可用于创建接收端的 DatagramSocket 对象，又可以创建发送端的 DatagramSocket 对象，在创建接收端的 DatagramSocket 对象时，必须要指定一个端口号，这样就可以监听指定的端口。

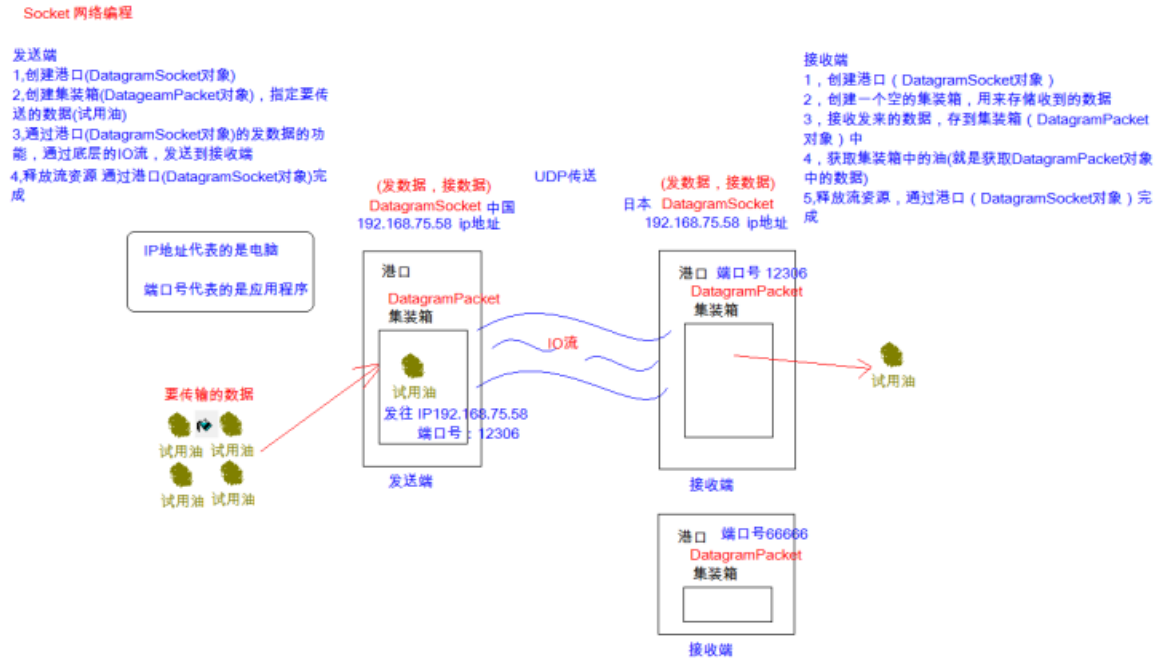
上面我们讲解了 DatagramSocket 的构造方法，接下来对 DatagramSocket 类中的常用方法进行详细地讲解。

void	receive(DatagramPacket p) 从此套接字接收数据报包。
void	send(DatagramPacket p) 从此套接字发送数据报包。

3.3UDP 网络程序

讲解了 DatagramPacket 和 DatagramSocket 的作用，接下来通过一个案例来学习一下它们在程序中的具体用法。

下图为 UDP 发送端与接收端交互图解



要实现 UDP 通信需要创建一个发送端程序和一个接收端程序，很明显，在通信时只有接收端程序先运行，才能避免因发送端发送的数据无法接收，而造成数据丢失。因此，首先需要来完成接收端程序的编写。

- UDP 完成数据的发送

```
/*
 * 发送端
 * 1,创建 DatagramSocket 对象
 * 2，创建 DatagramPacket 对象，并封装数据
 * 3，发送数据
 * 4，释放流资源
 */
public class UDPSend {
```

```
        public static void main(String[] args) throws IOException {
            //1,创建 DatagramSocket 对象
            DatagramSocket sendSocket = new DatagramSocket();
            //2，创建 DatagramPacket 对象，并封装数据
            //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。
            byte[] buffer = "hello,UDP".getBytes();
            DatagramPacket dp = new DatagramPacket(buffer, buffer.length,
InetAddress.getByName("192.168.75.58"), 12306);
            //3，发送数据
            //public void send(DatagramPacket p) 从此套接字发送数据报包
            sendSocket.send(dp);
            //4，释放流资源
            sendSocket.close();
        }
    }
```

- UDP 完成数据的接收

```
    /*
     * UDP 接收端
     *
     * 1,创建 DatagramSocket 对象
     * 2,创建 DatagramPacket 对象
     * 3,接收数据存储到 DatagramPacket 对象中
     * 4,获取 DatagramPacket 对象的内容
     * 5,释放流资源
     */
    public class UDPReceive {
        public static void main(String[] args) throws IOException {
            //1,创建 DatagramSocket 对象,并指定端口号
            DatagramSocket receiveSocket = new DatagramSocket(12306);
            //2,创建 DatagramPacket 对象，创建一个空的仓库
            byte[] buffer = new byte[1024];
            DatagramPacket dp = new DatagramPacket(buffer, 1024);
            //3,接收数据存储到 DatagramPacket 对象中
            receiveSocket.receive(dp);
            //4,获取 DatagramPacket 对象的内容
            //谁发来的数据  getAddress()
            InetAddress ipAddress = dp.getAddress();
            String ip = ipAddress.getHostAddress();//获取到了 IP 地址
            //发来了什么数据  getData()
            byte[] data = dp.getData();
            //发来了多少数据 getLenth()
```

```
            int length = dp.getLength();
            //显示收到的数据
            String dataStr = new String(data,0,length);
            System.out.println("IP 地址: "+ip+ "数据是"+ dataStr);
            //5,释放流资源
            receiveSocket.close();
        }
    }
```

第4章 TCP 通信

TCP 通信同 UDP 通信一样，都能实现两台计算机之间的通信，通信的两端都需要创建 socket 对象。

区别在于，UDP 中只有发送端和接收端，不区分客户端与服务器端，计算机之间可以任意地发送数据。

而 TCP 通信是严格区分客户端与服务器端的，在通信时，必须先由客户端去连接服务器端才能实现通信，服务器端不可以主动连接客户端，并且服务器端程序需要事先启动，等待客户端的连接。

在 JDK 中提供了两个类用于实现 TCP 程序，一个是 ServerSocket 类，用于表示服务器端，一个是 Socket 类，用于表示客户端。

通信时，首先创建代表服务器端的 ServerSocket 对象，该对象相当于开启一个服务，并等待客户端的连接，然后创建代表客户端的 Socket 对象向服务器端发出连接请求，服务器端响应请求，两者建立连接开始通信。

4.1 ServerSocket

通过前面的学习知道，在开发 TCP 程序时，首先需要创建服务器端程序。JDK 的 java.net 包中提供了一个 ServerSocket 类，该类的实例对象可以实现一个服务器段的程序。通过查阅 API 文档可知，ServerSocket 类提供了多种构造方法，接下来就对 ServerSocket 的构造方法进行逐一地讲解。

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

使用该构造方法在创建 ServerSocket 对象时，就可以将其绑定到一个指定的端口号上（参数 port 就是端口号）。

接下来学习一下 ServerSocket 的常用方法，如表所示。

方法摘要	
Socket	**accept**() 侦听并接受到此套接字的连接。
InetAddress	**getInetAddress**() 返回此服务器套接字的本地地址。

ServerSocket 对象负责监听某台计算机的某个端口号，在创建 ServerSocket 对象后，需要继续调

用该对象的 accept()方法，接收来自客户端的请求。当执行了 accept()方法之后，服务器端程序会发生阻塞，直到客户端发出连接请求，accept()方法才会返回一个 Scoket 对象用于和客户端实现通信，程序才能继续向下执行。

4.2 Socket

讲解了 ServerSocket 对象可以实现服务端程序，但只实现服务器端程序还不能完成通信，此时还需要一个客户端程序与之交互，为此 JDK 提供了一个 Socket 类，用于实现 TCP 客户端程序。

通过查阅 API 文档可知 Socket 类同样提供了多种构造方法，接下来就对 Socket 的常用构造方法进行详细讲解。

```
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。
```

使用该构造方法在创建 Socket 对象时，会根据参数去连接在指定地址和端口上运行的服务器程序，其中参数 host 接收的是一个字符串类型的 IP 地址。

```
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。
```

该方法在使用上与第二个构造方法类似，参数 address 用于接收一个 InetAddress 类型的对象，该对象用于封装一个 IP 地址。

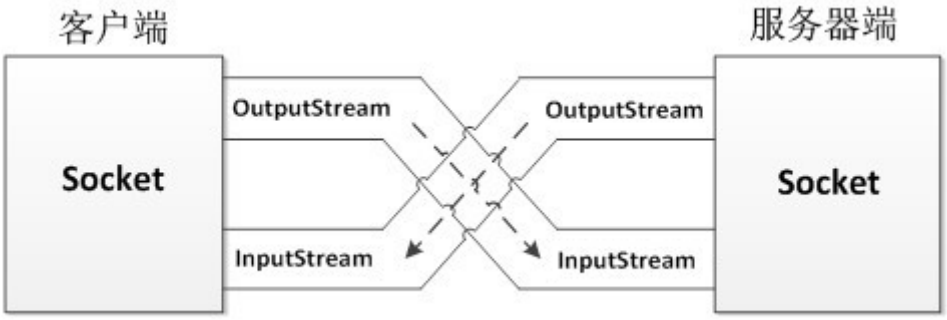
在以上 Socket 的构造方法中，最常用的是第一个构造方法。

接下来学习一下 Socket 的常用方法，如表所示。

方法声明	功能描述
int getPort()	该方法返回一个 int 类型对象，该对象是 Socket 对象与服务器端连接的端口号
InetAddress getLocalAddress()	该方法用于获取 Socket 对象绑定的本地 IP 地址，并将 IP 地址封装成 InetAddress 类型的对象返回
void close()	该方法用于关闭 Socket 连接，结束本次通信。在关闭 socket 之前，应将与 socket 相关的所有的输入/输出流全部关闭，这是因为一个良好的程序应该在执行完毕时释放所有的资源
InputStream getInputStream()	该方法返回一个 InputStream 类型的输入流对象，如果该对象是由服务器端的 Socket 返回，就用于读取客户端发送的数据，反之，用于读取服务器端发送的数据
OutputStream getOutputStream()	该方法返回一个 OutputStream 类型的输出流对象，如果该对象是由服务器端的 Socket 返回，就用于向客户端发送数据，反之，用于向服务器端发送数据

在 Socket 类的常用方法中，getInputStream()和 getOutStream()方法分别用于获取输入流和输出流。当客户端和服务端建立连接后，数据是以 IO 流的形式进行交互的，从而实现通信。

接下来通过一张图来描述服务器端和客户端的数据传输，如下图所示。



4.3 简单的 TCP 网络程序

了解了 ServerSocket、Socket 类的基本用法，为了让大家更好地掌握这两个类的使用，接下来通过一个 TCP 通信的案例来进一步学习.

要实现 TCP 通信需要创建一个服务器端程序和一个客户端程序，为了保证数据传输的安全性，首先需要实现服务器端程序。

```
    /*
     * TCP 服务器端
     *
     * 1,创建服务器 ServerSocket 对象（指定服务器端口号）
     * 2，开启服务器了，等待客户端的连接，当客户端连接后，可以获取到连接服务器的客户端 Socket 对象
     * 3,给客户端反馈信息
     * 4,关闭流资源
     */
    public class TCPServer {
        public static void main(String[] args) throws IOException {
            //1,创建服务器 ServerSocket 对象（指定服务器端口号）
            ServerSocket ss = new ServerSocket(8888);
            //2,开启服务器了,等待客户端的连接,当客户端连接后,可以获取到连接服务器的客户端 Socket
对象
            Socket s = ss.accept();
            //3,给客户端反馈信息
            /*
             * a,获取客户端的输出流
             * b,在服务端端，通过客户端的输出流写数据给客户端
             */
            //a,获取客户端的输出流
            OutputStream out = s.getOutputStream();
            //b,在服务端端，通过客户端的输出流写数据给客户端
            out.write("你已经连接上了服务器".getBytes());
            //4,关闭流资源
            out.close();
            s.close();
```

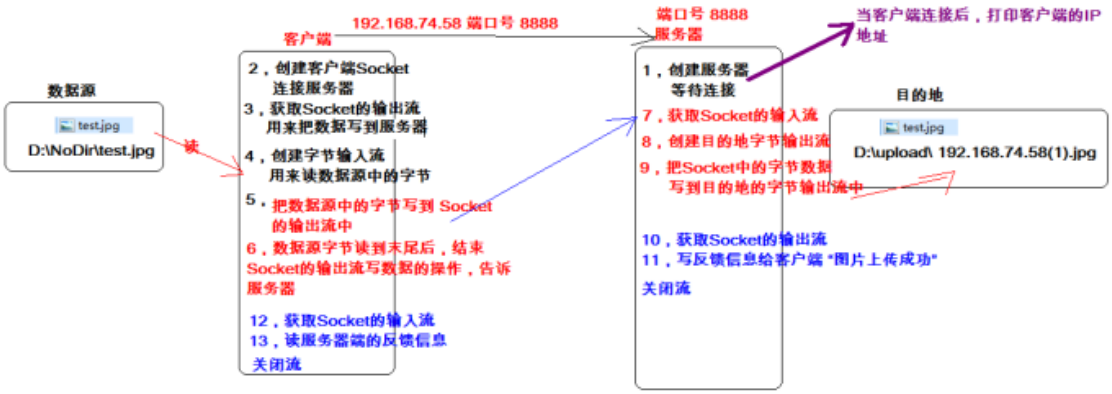
```
            //ss.close();  服务器流 通常都是不关闭的
        }
    }
```

完成了服务器端程序的编写，接下来编写客户端程序。

```
/*
 * TCP 客户端
 *
 * 1，创建客户端 Socket 对象,（指定要连接的服务器地址与端口号）
 * 2,获取服务器端的反馈回来的信息
 * 3,关闭流资源
 */
public class TCPClient {
    public static void main(String[] args) throws IOException {
        //1，创建客户端 Socket 对象,（指定要连接的服务器地址与端口号）
        Socket s = new Socket("192.168.74.58", 8888);
        //2,获取服务器端的反馈回来的信息
        InputStream in = s.getInputStream();
        //获取获取流中的数据
        byte[] buffer = new byte[1024];
        //把流中的数据存储到数组中，并记录读取字节的个数
        int length = in.read(buffer);
        //显示数据
        System.out.println( new String(buffer, 0 , length) );
        //3,关闭流资源
        in.close();
        s.close();
    }
}
```

4.4 文件上传案例

目前大多数服务器都会提供文件上传的功能，由于文件上传需要数据的安全性和完整性，很明显需要使用 TCP 协议来实现。接下来通过一个案例来实现图片上传的功能。如下图所示。原图：文件上传.bmp



- 首先编写服务器端程序，用来接收图片。

```
/*
 * 文件上传  服务器端
 *
 */
public class TCPServer {
    public static void main(String[] args) throws IOException {
        //1,创建服务器，等待客户端连接
        ServerSocket serverSocket = new ServerSocket(8888);
        Socket clientSocket = serverSocket.accept();
        //显示哪个客户端 Socket 连接上了服务器
        InetAddress ipObject = clientSocket.getInetAddress();//得到 IP 地址对象
        String ip = ipObject.getHostAddress(); //得到 IP 地址字符串
        System.out.println("小样，抓到你了，连接我!! " + "IP:" + ip);

        //7,获取 Socket 的输入流
        InputStream in = clientSocket.getInputStream();
        //8,创建目的地的字节输出流   D:\\upload\\192.168.74.58(1).jpg
        BufferedOutputStream fileOut = new BufferedOutputStream(new
FileOutputStream("D:\\upload\\192.168.74.58(1).jpg"));
        //9,把 Socket 输入流中的数据，写入目的地的字节输出流中
        byte[] buffer = new byte[1024];
        int len = -1;
        while((len = in.read(buffer)) != -1){
            //写入目的地的字节输出流中
            fileOut.write(buffer, 0, len);
        }

        //-----------------反馈信息----------------------
        //10,获取 Socket 的输出流，作用：写反馈信息给客户端
        OutputStream out = clientSocket.getOutputStream();
        //11,写反馈信息给客户端
```

```
            out.write("图片上传成功".getBytes());

            out.close();
            fileOut.close();
            in.close();
            clientSocket.close();
            //serverSocket.close();
        }
    }
```

- 编写客户端，完成上传图片

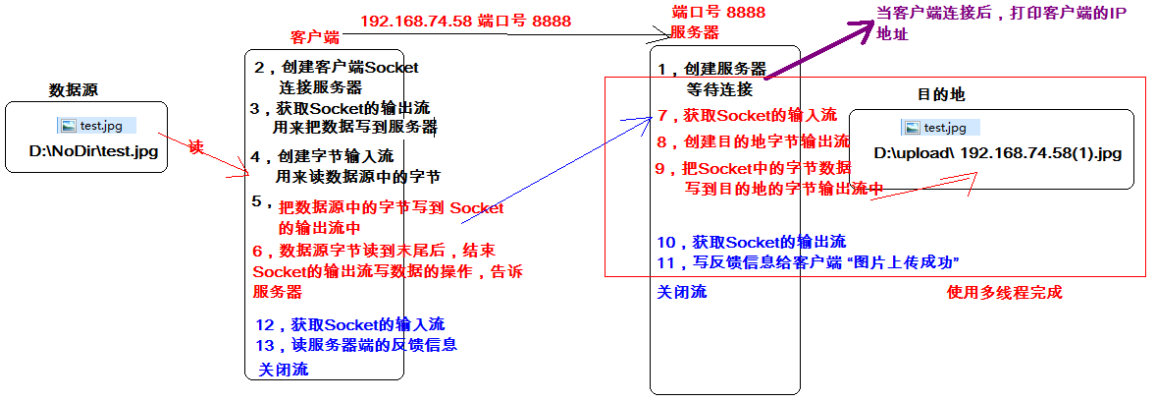
```
    /*
     * 文件上传 客户端
     *
     * public void shutdownOutput()  禁用此 Socket 的输出流,间接的相当于告知了服务器数据写入完毕
     */
    public class TCPClient {
        public static void main(String[] args) throws IOException {
            //2,创建客户端 Socket，连接服务器
            Socket socket = new Socket("192.168.74.58", 8888);
            //3,获取 Socket 流中的输出流，功能：用来把数据写到服务器
            OutputStream out = socket.getOutputStream();
            //4,创建字节输入流，功能：用来读取数据源(图片)的字节
            BufferedInputStream fileIn = new BufferedInputStream(new FileInputStream("D:\\NoDir\\test.jpg"));
            //5,把图片数据写到 Socket 的输出流中(把数据传给服务器)
            byte[] buffer = new byte[1024];
            int len = -1;
            while ((len = fileIn.read(buffer)) != -1){
                //把数据写到 Socket 的输出流中
                out.write(buffer, 0, len);
            }
            //6,客户端发送数据完毕，结束 Socket 输出流的写入操作，告知服务器端
            socket.shutdownOutput();

            //-----------------反馈信息----------------------
            //12,获取 Socket 的输入流  作用： 读反馈信息
            InputStream in = socket.getInputStream();
            //13,读反馈信息
            byte[] info = new byte[1024];
            //把反馈信息存储到 info 数组中，并记录字节个数
            int length = in.read(info);
            //显示反馈结果
```

```
            System.out.println( new String(info, 0, length) );

            //关闭流
            in.close();
            fileIn.close();
            out.close();
            socket.close();
        }
    }
```

4.5 文件上传案例多线程版本



实现服务器端可以同时接收多个客户端上传的文件。

- 我们要修改服务器端代码

```
    /*
     * 文件上传多线程版本，服务器端
     */
    public class TCPServer {
        public static void main(String[] args) throws IOException {
            //1,创建服务器，等待客户端连接
            ServerSocket serverSocket = new ServerSocket(6666);

            //实现多个客户端连接服务器的操作
            while(true){
                final Socket clientSocket = serverSocket.accept();
                //启动线程，完成与当前客户端的数据交互过程
                new Thread(){
                    public void run() {
                        try{
                            //显示哪个客户端 Socket 连接上了服务器
                            InetAddress ipObject = clientSocket.getInetAddress();//得
到 IP 地址对象

                            String ip = ipObject.getHostAddress(); //得到 IP 地址字符串
                            System.out.println("小样，抓到你了，连接我!! " + "IP:" + ip);
```

```
                            //7,获取 Socket 的输入流
                            InputStream in = clientSocket.getInputStream();
                            //8,创建目的地的字节输出流   D:\\upload\\192.168.74.58(1).jpg
                            BufferedOutputStream fileOut = new BufferedOutputStream(new
FileOutputStream("D:\\upload\\"+ip+"("+System.currentTimeMillis()+").jpg"));
                            //9,把 Socket 输入流中的数据，写入目的地的字节输出流中
                            byte[] buffer = new byte[1024];
                            int len = -1;
                            while((len = in.read(buffer)) != -1){
                                //写入目的地的字节输出流中
                                fileOut.write(buffer, 0, len);
                            }

                            //------------------反馈信息----------------------
                            //10,获取 Socket 的输出流，作用：写反馈信息给客户端
                            OutputStream out = clientSocket.getOutputStream();
                            //11,写反馈信息给客户端
                            out.write("图片上传成功".getBytes());

                            out.close();
                            fileOut.close();
                            in.close();
                            clientSocket.close();
                        } catch(IOException e){
                            e.printStackTrace();
                        }
                    };
                }.start();
            }

            //serverSocket.close();
        }
    }
```

第5章 总结

5.1知识点总结

- IP 地址：用来唯一表示我们自己的电脑的，是一个网络标示
- 端口号： 用来区别当前电脑中的应用程序的
- UDP：传送速度快，但是容易丢数据，如视频聊天，语音聊天

- TCP：传送稳定，不会丢失数据，如文件的上传、下载
- UDP 程序交互的流程
  - 发送端

```
1,创建 DatagramSocket 对象
2，创建 DatagramPacket 对象，并封装数据
3，发送数据
4，释放流资源
```

  - 接收端

```
1,创建 DatagramSocket 对象
2,创建 DatagramPacket 对象
3,接收数据存储到 DatagramPacket 对象中
4,获取 DatagramPacket 对象的内容
5,释放流资源
```

- TCP 程序交互的流程
  - 客户端

```
1,创建客户端的 Socket 对象
2,获取 Socket 的输出流对象
3,写数据给服务器
4,获取 Socket 的输入流对象
5，使用输入流，读反馈信息
6,关闭流资源
```

  - 服务器端

```
1，创建服务器端 ServerSocket 对象，指定服务器端端口号
2，开启服务器，等待着客户端 Socket 对象的连接，如有客户端连接，返回客户端的 Socket 对象
3,通过客户端的 Socket 对象，获取客户端的输入流，为了实现获取客户端发来的数据
4,通过客户端的输入流，获取流中的数据
5,通过客户端的 Socket 对象，获取客户端的输出流，为了实现给客户端反馈信息
6,通过客户端的输出流，写数据到流中
7,关闭流资源
```