一、基本概念

1、基本概念

①MAC：MAC（Media Access Control或者Medium Access Control）地址，意译为媒体访问控制，或称为物理地址、硬件地址，用来定义网络设备的位置。在OSI模型中，第三层网络层负责IP地址 ，第二层数据链路层则负责 MAC地址。因此一个主机会有一个MAC地址，而每个网络地址会有一个专属于它的IP地址。

②IP地址：是指互联网协议地址（Internet Protocol Address，又译为网际协议地址），是IP Address的缩写。IP地址是IP协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。目前还有些ip代理软件，但大部分都收费。

TCP/IP 网络模式
应用层　如HTTP、FTP、DNS
传输层 如TCP、UDP
网络层 如IP、ICMP、IGMP
链锯层 如驱动程序、接口

③端口：可以认为是设备与外界通讯交流的出口。

端口号是用两个字节（16位的二进制数）表示，它的取值范围是0~65 535 ，其中0~1023之间的端口号用于以下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。端口分为物理端口和逻辑端口（软件应用程序的数字标识）。

④TCP (Transmission Control Protocol)和UDP(User Datagram Protocol)协议属于传输层协议。

TCP协议是面向连接的通信协议，提供IP环境下的数据可靠传输，它提供的服务包括数据流传送、可靠性、有效流控、全双工操作和多路复用。通过面向连接、端到端和可靠的数据包发送。通俗说，它是事先为所发送的数据开辟出连接好的通道，然后再进行数据发送；

⑤UDP是无线通信协议，不为IP提供可靠性、流控或差错恢复功能。

2、常见InetAddress类的常用方法示例：

```
public class INetAddress {
        public static void main(String[] args) throws IOException {
                //创建一个表示本地主机的InetAddress对象
        InetAddress localAddress = InetAddress.getLocalHost();
        //获得指定主机的InetAddress对象
        InetAddress remoteAddress = InetAddress.getByName("www.info-soft.cn");
        //得到IP地址的主机名。
        System.out.println("本机的IP地址："+localAddress.getHostName());
        //获得字符串格式的原始IP地址
```

```
        System.out.println("itcast的IP地址："+remoteAddress.getHostAddress());
        //判断指定的时间内地址是否可以到达
        System.out.println("3秒是否可达："+remoteAddress.isReachable(3000));
        System.out.println("itcast的主机名为："+remoteAddress.getHostName());
        }
}
```

3、运行结果：

> 本机的IP地址：DESKTOP-FL1GF59
> itcast的IP地址：47.97.62.97
> 3秒是否可达：true
> itcast的主机名为：www.info-soft.cn

二、UDP通信

1、DatagramPacket

该类类似于集装箱，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制定发送端IP地址和端口号。

DatagramPacket构造方法：

①DatagramPacket（byte[] buf,int length）

用于接收端，创建DatagramPacket对象时，指定了封装数据的字节数组和数据大小。

②DatagramPacket（byte[] buf,int length，InetAddress addr,int port）

用于发送端，创建DatagramPacket对象时，指定了封装数据的字节数组、数据大小、数据包的目标IP地址（addr）以及端口号（port）。

③DatagramPacket（byte[] buf,int offset,int length）

用于接收端，创建DatagramPacket对象时，指定了封装数据的字节数组、数据大小，以及起始位置。offset 参数用于指定接收的数据在放入buf缓冲数组时是从offset处开始的。

④DatagramPacket（byte[] buf,int offset，int length，InetAddress addr,int port）

用于发送端， 创建DatagramPacket对象时，指定了封装数据的字节数组、数据大小、数据包的目标IP地址（addr）以及端口号（port）。offset 参数用于指定发送数据的偏移量为offset，即从offset位置开始发送数据。

DatagramPacket类中的常用方法	
方法声明	功能描述
InetAddress getAddress()	该方法用于返回发送端或者接收端的IP地址，如果发送端的DatagramPacket对象，就返回接收端的IP地址，反之，就返回发送端的IP地址
int getPort()	该方法用于返回发送端或者接收端的端口号，如果发送端的DatagramPacket对象，就返回接收端的端口号，反之，就返回发送端的端口号
byte[]	该方法用于返回将要接收或者将要发送的数据，如果是发送端的DatagramPacket

getData()	对象，就返回将要发送的数据，反之，就返回接收到的数据
int getLength()	该方法用于返回将要接收或者将要发送数据的长度，如果是发送端的DatagramPacket对象，就返回将要发送的数据长度，反之，就返回接收到数据的长度

2、DatagramSocket

DatagramSocket类似与码头，实例对象就可以发送和接收DatagramPacket数据包，在创建发送端和接收端的DatagramSocket对象时，使用的构造方法有所不同。

DatagramSocket构造方法：

①DatagramSocket（）

用于创建发送端的DatagramSocket对象时，在创建对象时，并没有指定端口号，此时，系统会分配一个没有被其他网络程序使用的端口号。

②DatagramSocket（int port）

该方法即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。**

③DatagramSocket（int port,InetAddress addr）

使用该构造方法在DatagramSocket时，不仅指定端口号，还指定了相关的IP地址，这种情况适用于计算上有多块网卡的情况。

DatagramSocket类中的常用方法	
方法声明	功能描述
void receive(DatagramPacket p)	该方法用于将接收到的数据填充到DatagramPacket数据包中，在接收到数据之前会一直处于阻塞状态，只有当接收到数据包时，该方法才会返回。
void send(DatagramPacket p)	该方法用于发送DatagramPacket数据包，发送的数据包中包含将要发送的数据、数据的长度、远程主机的IP地址和端口号
void close()	关闭当前的Socket,通知驱动程序释放为这个Socket保留的资源。

3、UDP 网络程序

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。示例：

```
//接收端程序
public class UDP_Receive {
    public static void main(String[] args) throws IOException {
        //创建一个长度为1024的字节数组，用于接收数据
        byte [] buf = new byte[1024];
        //定义一个DatagramSocket对象，监听的端口为8954
        DatagramSocket ds = new DatagramSocket(8954);
        //定义一个DatagramPacket对象，用于接收数据
        DatagramPacket dp = new DatagramPacket(buf,1024);
        System.out.println("等待接收数据");
        //等待接收数据，如果没有数据则会阻塞
```

```
        ds.receive(dp);
        //调用DatagramPacket的方法获得接收的消息，包括内容、长度、IP地址和端口号
        String str = new String(dp.getData(),0,dp.getLength())
+"from"+dp.getAddress().getHostAddress()+":"+dp.getPort();
        //打印收到的信息
        System.out.println(str);
        //释放资源
        ds.close();
    }
}

//发送端程序
public class UDP_Send {
    public static void main(String[] args) throws Exception {
        //创建一个DatagramSocket对象
        DatagramSocket ds = new DatagramSocket(3000);
        //要发送的数据
        String str = "Hello World!";
        //创建一个要发送的数据包，包括发送数据，数据长度，接收端IP地址以及端口号
        DatagramPacket dp = new DatagramPacket(str.getBytes(),str.length(),
InetAddress.getByName("localhost"),8954);
        //发送数据
        System.out.println("发送消息");
        ds.send(dp);
        //释放资源
        ds.close();
    }
}
```

4、运行结果

> 发送消息
>
> 等待接收数据
> Hello World!from127.0.0.1:3000

解析：发送货物（数据）前，确定到货码头是否能接收。

创建空间（数据容器）接收货物（数据），创建码头【DatagramSocket(8954)】并实时监听发货码头发货通道（端口），创建集装箱并将空间加入用于接收货物，一直等待接收货物，接收码头将货物填充到集装箱中，获取到货物信息（数据等信息）。

发送货物需要建一个码头【DatagramSocket(3000)】，码头可指定发送通道即端口（也可以不指定发送通道），将要发送货物（数据）装进集装箱（DatagramPacket

）中，并指定发送到的码头名字（IP地址或主机名）及接收通道（端口），通过码头把集装箱发出去[send()]，腾出空间（close）。

三、TCP通信

1、ServerSocket

在开发TCP程序时，首先需要创建服务器端程序，其构造方法如下：

①ServerSocket（）

使用该构造方法在创建ServerSocket对象时并没有绑定端口号，不能直接使用，还需要继续调用bind(SocketAddress endpoint)方法将其绑定到指定的端口上，才能正常使用。

②ServerSocket（int port）【最常用】

使用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。

③ServerSocket（int port，int backlog）

backlog 参数用于指定在服务器忙时，可以与之保持连接请求的等待客户数量，如果没有指定这个参数默认为50 。

④ServerSocket（int port，int backlog，InetAddress bindAddr）

指定了相关的IP地址，适用于计算机上有多块网卡和多个IP的情况。

ServerSocket类中的常用方法	
方法声明	功能描述
Socket accept()	该方法用于等待客户端的连接，在客户端连接之前一直处于阻塞状态，如果有客户端连接就会返回一个与之对应的Socket对象
InetAddress getInetAddress()	该方法用于返回一个InetAddress对象，该对象封装了ServerSocket绑定的IP地址
boolean isClosed()	该方法用于判断ServerSocket对象是否为关闭状态，如果是关闭状态则返回true，反之则返回false
void bind (SocketAddress endpoint)	该方法用于判断ServerSocket对象绑定到指定的IP地址和端口号，其中参数endpoint封装了IP地址和端口号。

2、Socket

Socket类常用构造方法：

①Socket（）

使用该构造方法在创建Socket对象时，并没指定IP地址和端口号，创建对象后还需调用connect（SocketAddress endpoint）方法，才能完成与指定服务器的连接，参数endpoint封装了IP地址和端口号。

②Socket（String host,int port）

使用该构造方法在创建Socket对象时，根据参数去连接在指定IP地址和端口上运行的服务器程序，其中参数host接收的一个字符类型的IP地址。

③Socket（InetAddress addres,int port）

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。

Socket类中的常用方法	
方法声明	功能描述
int getPort()	该方法返回一个int类型对象，该对象时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，应将于Socket相关的所有的输入与输出流全部关闭，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
IputStream 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

3、简单的TCP网络程序

```
/**
 * TCP服务器端程序
 */
public class TCP_Server {
    public static void main(String[] args) throws Exception {
        //创建TCPServer对象，并调用listen（）方法
        new TCPServer().listen();
    }
}

//TCP服务器端
class TCPServer{
    //定义一个端口号
    private static final int PORT= 7788;
    //定义一个listen（）方法，抛出一个异常
    public void listen() throws Exception{
        //创建ServerSocket对象
        ServerSocket serverSocket = new ServerSocket(PORT);
        //调用ServerSocket的accept（）方法接收数据
        Socket client=serverSocket.accept();
        //获取客户端的输出流
        OutputStream os = client.getOutputStream();
        System.out.println("开始与客户端交换数据");
```

```
            os.write(("Java欢迎你！").getBytes());
            Thread.sleep(5000);
            //模拟执行其他功能占用的时间
            System.out.println("结束与客户端交互数据");
            os.close();
            client.close();
        }
    }

    /**
     *  TCP客户端程序
     */
    public class TCP_Client {
        public static void main(String[] args) throws Exception{
            //创建TCPClient对象，并调用connect（）方法
            new TCPClient().connect();
        }
    }

    //TCP客户端
    class TCPClient {
        //服务端的端口号
        private static final int PORT = 7788;

        public void connect() throws Exception {
            //创建一个Socket并连接到给出地址和端口号的计算机
            Socket client = new Socket(InetAddress.getLocalHost(), PORT);
            //得到接收数据的流
            InputStream is = client.getInputStream();
            //定义1024个字节数组的缓冲区
            byte[] buf = new byte[1024];
            //将数据读取到缓冲区中
            int len = is.read(buf);
            //将缓冲区中的数据输出
            System.out.println(new String(buf, 0, len));
            //关闭Socket对象，释放资源
            client.close();
        }
    }
```

Example4 运行结果：
开始与客户端交换数据
结束与客户端交互数据

Example5 运行结果：
Java欢迎你！

4、TCP案例——文件上传

实现图片上传到服务器的功能。

服务端程序：

```
public class TCP_UploadServer {
    public static void main(String[] args) throws Exception{
        //创建ServerSocket对象
        ServerSocket serverSocket = new ServerSocket(10001);
        while (true){
            //调用accept（）方法接收客户端请求，得到Socket对象
            Socket s = serverSocket.accept();
            //每当和客户端建立Socket连接后，单独开启一个线程处理和客户端的交互
            new Thread(new ServerThread(s)).start();
        }
    }
}

class ServerThread implements Runnable{
    //持有一个Socket类型的属性
    private Socket socket ;
    //构造方法中吧Socket对象作为实参传入
    public ServerThread(Socket socket){
        this.socket=socket;
    }

    @Override
    public void run() {
        //获取客户端的IP地址
        String ip = socket.getInetAddress().getHostAddress();
        int count =1;        //上传图片个数
        try{
            InputStream in = socket.getInputStream();
            //创建上传图片目录的File对象
            File parentFile =new File("d:/upload");
            //如果不存在，就创建这个目录
```

```
            if (!parentFile.exists()){
                parentFile.mkdir();
            }
            //把客户端的IP地址作为上传出文件的文件名
            File file = new File(parentFile,ip+"("+count+").jpeg");
            while (file.exists()){
                //如果文件名存在，则把count++
                file=new File(parentFile,ip+"("+(count++)+").jpeg");
            }
            //创建FileOutputStream对象
            FileOutputStream fos = new FileOutputStream(file);
            //定义一个字节数组
            byte[] buf=new byte[1024];
            //定义一个int类型的变量len，初始值为0
            int len=0;
            //循环读取数据
            while ((len=in.read(buf))!=-1){
                fos.write(buf,0,len);
            }
            //获取服务端的输出流
            OutputStream out = socket.getOutputStream();
            //上传成功后向客户端写出"上传成功"
            out.write(("上传成功").getBytes());
            //关闭输出流对象
            fos.close();
            //关闭Socket对象
            socket.close();
        }catch (Exception e){
            throw new RuntimeException(e);
        }
    }
}
```

客户端程序：

```
public class TCP_UploadClient {
    public static void main(String[] args) throws Exception{
        //创建客户端Socket对象，指定IP地址和端口号
        Socket socket= new Socket(InetAddress.getLocalHost(),10001);
        //获取Socket的输出流对象
        OutputStream out= socket.getOutputStream();
        //创建FileInputStream对象
```

```
        FileInputStream fis = new FileInputStream("d:/1111.jpg");
        //定义一个字节数组
        byte[] buf =new byte[1024];
        //定义一个int类型的变量len
        int len;
        //循环读取数据
        while ((len=fis.read(buf))!=-1){
            out.write(buf,0,len);
        }
        //关闭客户端输出流
        socket.shutdownOutput();
        //获取Socket的输入流对象
        InputStream in = socket.getInputStream();
        //定义一个字节数组
        byte[] bufMsg = new byte[1024];
        //接收服务端的信息
        int num =in.read(bufMsg);
        String Msg = new String(bufMsg,0,num);
        System.out.println(Msg);
        //关闭输入流对象
        fis.close();
        //关闭Socket对象
        socket.close();
    }
}
```

需注意：shutdownOutput()方法非常重要，因为服务器端程序在while循环中读取客户端发送的数据，当读取到-1时才会结束循环，如果客户端不调用shutdownOutput（）方法关闭输出流，服务器端就不会读到-1，而会一直执行while循环，同时客户端服务器端的read（byte[]）方法也是一个阻塞方法，这样客户端与服务器端进入一个“死锁”状态。