Java程序设计

2019春季

彭启民

pengqm@aliyun.com

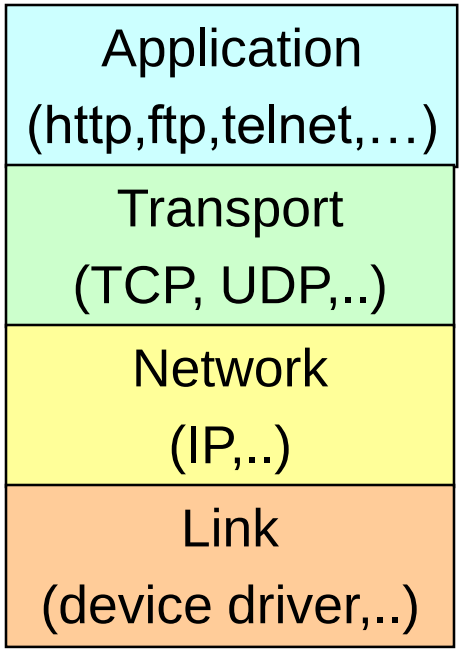
网 络

- OSI七层模型
  - 应用层： 文件传输，电子邮件，文件服务，虚拟终端 TFTP，HTTP，SNMP，FTP，SMTP，DNS，Telnet
  - 表示层： 数据格式化，代码转换，数据加密 没有协议
  - 会话层： 解除或建立与别的接点的联系 没有协议
  - 传输层： 提供端对端的接口 TCP，UDP
  - 网络层： 为数据包选择路由 IP，ICMP，RIP，OSPF，BGP，IGMP
  - 数据链路层： 传输有地址的帧以及错误检测功能 SLIP，CSLIP，PPP，ARP，RARP，MTU
  - 物理层： 以二进制数据形式在物理媒体上传输数据 ISO2110，IEEE802，IEEE802.2
- TCP/IP 五层模型的协议
  - 应用层 ：
  - 传输层：四层交换机、四层路由器
  - 网络层 ：路由器、三层交换机
  - 数据链路层：网桥、以太网交换机（二层交换机）、网卡（一半工作在物理层、一半工作在数据链路层）
  - 物理层 ：中继器、集线器、网线

Networking Basics

- Applications Layer
  - Standard apps
    - HTTP
    - FTP
    - Telnet
  - User apps
- Transport Layer
  - TCP
  - UDP
  - Programming Interface:
    - Sockets
- Network Layer
  - IP
- Link Layer
  - Device drivers

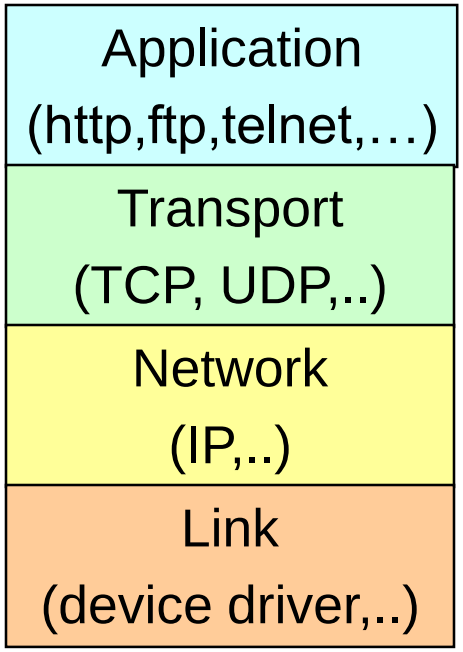
- TCP/IP Stack

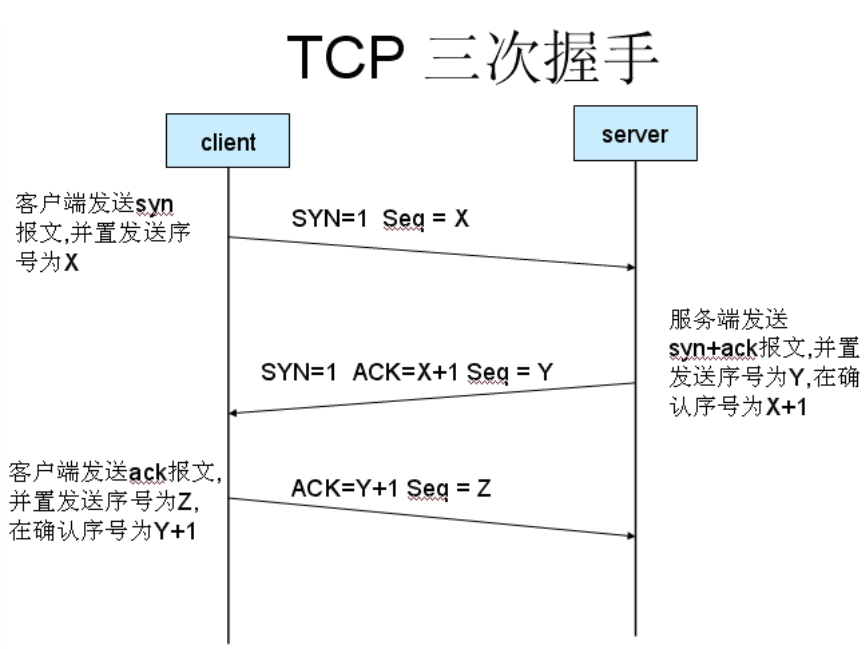


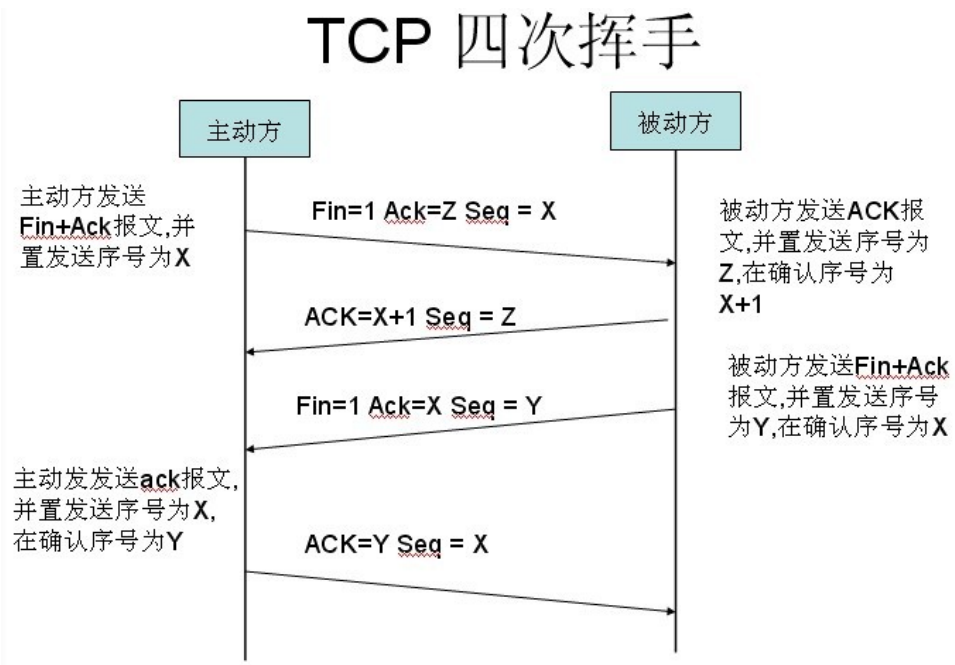
Networking Basics

- TCP (Transport Control Protocol) is a connection-oriented protocol that provides a reliable flow of data between two computers.
- Example applications:
  - HTTP
  - FTP
  - Telnet

- TCP/IP Stack





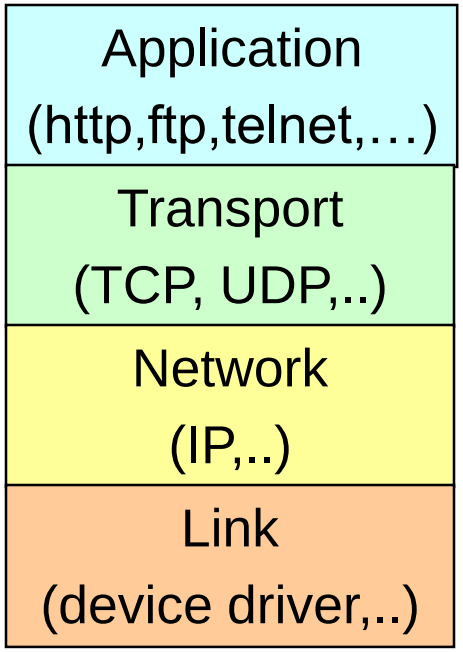


连接时三次握手，连接终止四次挥手。
TCP创建过程和链接折除过程是由TCP/IP协议栈自动创建的，开发者不需要进行控制.

Networking Basics

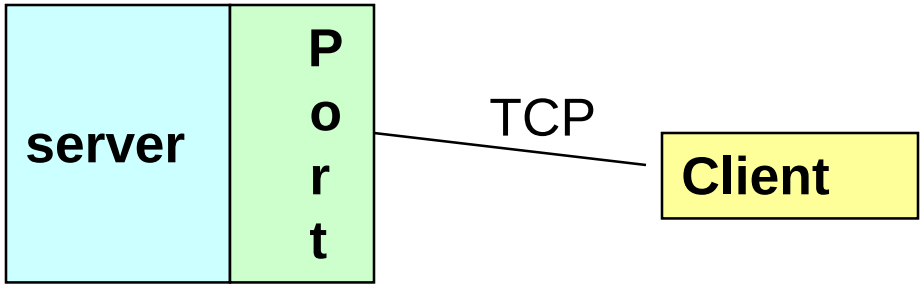
- UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival.
- Example applications:
  - Clock server
  - Ping

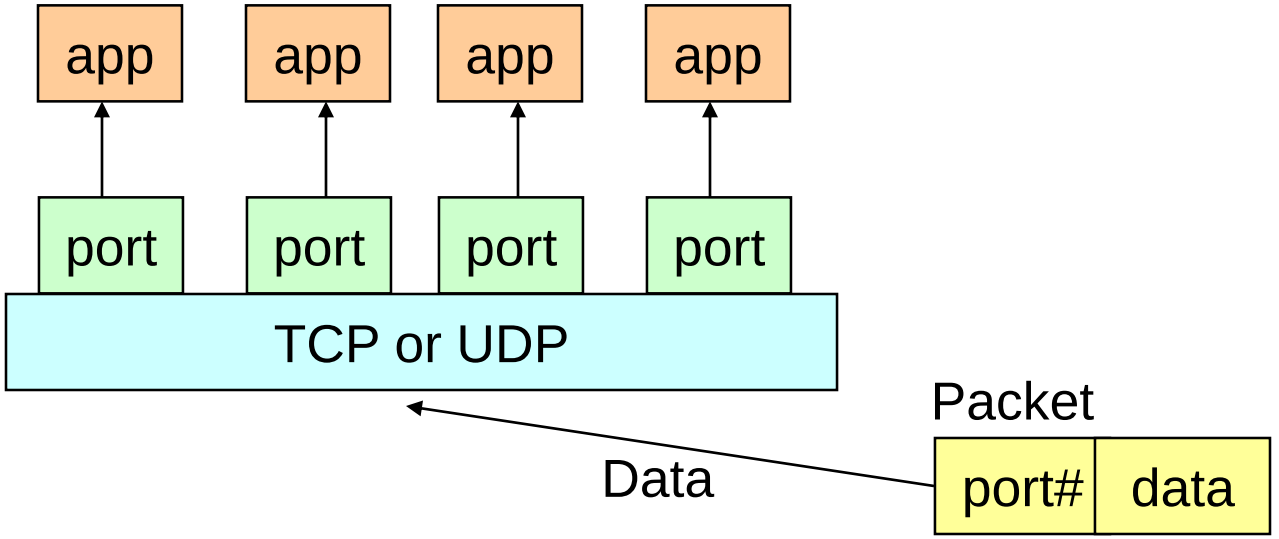
- TCP/IP Stack



Understanding Ports

- The TCP and UDP protocols use *ports* to map incoming data to a particular *process* running on a computer.



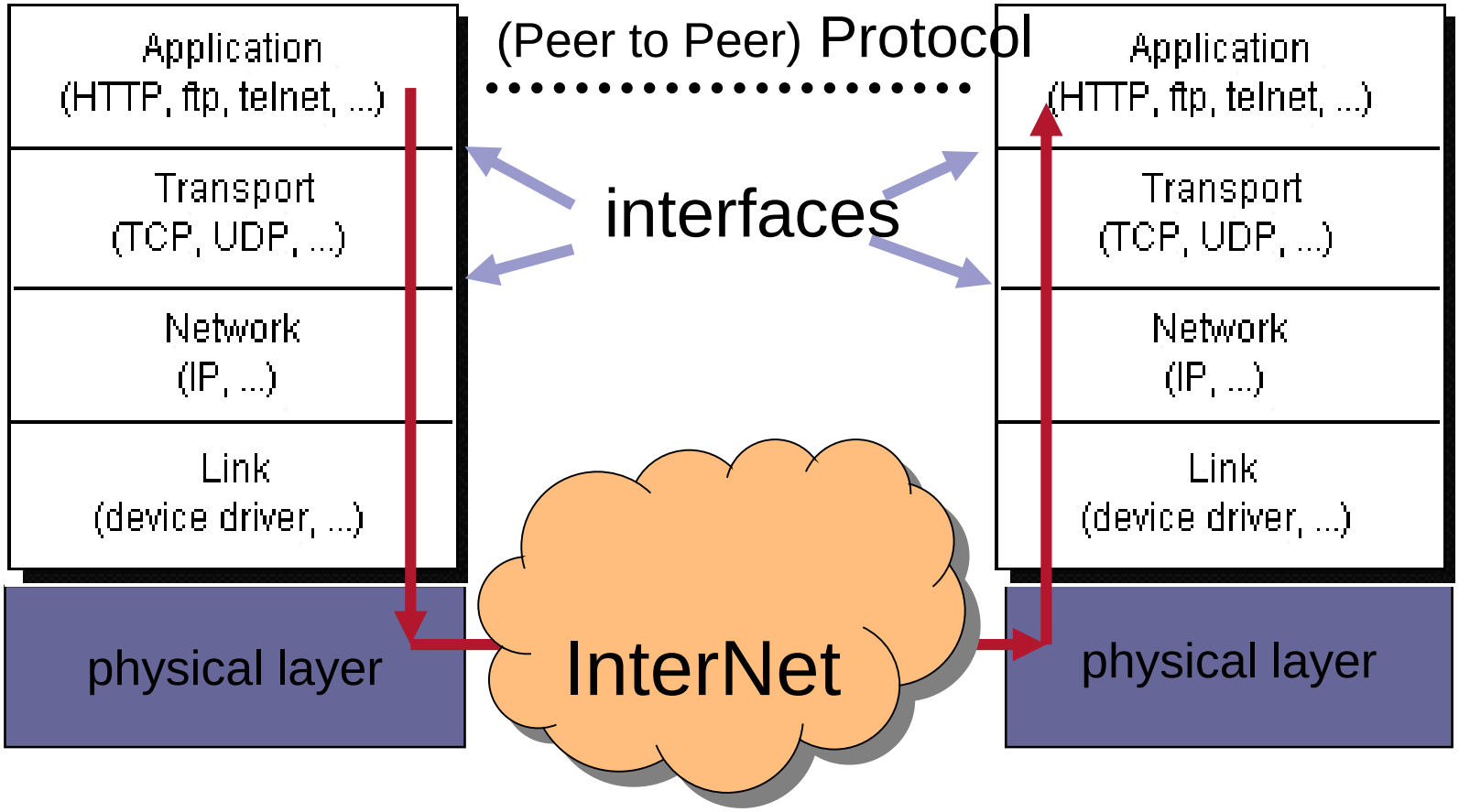


Understanding Ports

- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services:
  - ftp 21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - login 513/tcp
- User level process/services generally use port number value >= 1024

Networking Basics

- TCP/IP Network Protocol



- Java将底层的网络通信细节予以屏蔽，使得使用的编程模型是一个文件模型，可以象操作流一样来操作网络数据传输。
  - 由于在网络连接中，通常都需要一个服务器同时为多个客户端服务，因此Java的多线程机制也大派用场。

- Java网络类都在 java.net 类库中
  - 流套接字使用TCP数据传输
  - 数据报套接字使用UDP
  - Applet技术为对Web程序设计的初步支持
- 高层网络编程
  - RMI, CORBA, EJB, JDBC

- 轻量级的 HTTP 服务器(JavaSE 6)
- 针对WWW浏览、Email等，Java提供了相应的扩展组件，如对于Email应用，Java提供了JavaMail API，使用时只需要调用其提供的方法就可以完成如发送邮件的操作：

```
Transport.send(message);
```

- 网址 (InetAddress类)
  - InetAddress没有构造方法,创建InetAddress类不用构造函数(不用new),必须使用代理方法
    - getHostName( ) 获取InetAddress对象的主机名或域名
    - getHostAddress( ) 获取InetAddress对象的IP地址
    - getLocalHost( )获取本机的IP地址

- getLocalHost方法
  - 返回代表本地主机的InetAddress对象.

public static InetAddress getLocalHost() throws UnknownHostException

  - 例: InetAddress adress= InetAddress.getLocalHost();
- getByName方法
  - 返回代表主机名的InetAddress对象.

public static synchronized InetAddress getByName(String host) throws UnknownHostException

  - 例: InetAddress adress= InetAddress.getByName(“www.ucas.ac.cn”);

- getAllByName方法
  - 返回代表主机名的一组InetAddress对象.

public static synchronized InetAddress[] getAllByName(String host)
throws UnknownHostException

  - 例: InetAddress adress[]= InetAddress.getAllByName(“www.ucas.ac.cn”);
- getByAddress方法
  - 返回代表主机名的InetAddress对象.

static InetAddress getByAddress(byte[] addr)

static InetAddress getByAddress(String host, byte[] addr)

- **String getHostName()**
  - 获取InetAddress对象的主机名或域名
- **byte[] getAdress()**
  - 返回IP地址
- **String toString()**
  - 返回主机名和IP地址字符串
- **Boolean equal(InetAddress other)**
  - 与其它地址比较

- 例:获取本机的IP地址

```
import java.net.*;
public class getLocalHostTest
{
  public static void main(String args[])
   {
        InetAddress myIP=null;
        try {
            myIP=InetAddress.getLocalHost();
        }catch(UnknownHostException e){}
       System.out.println(myIP);
   }
}
```

根据域名自动到DNS上查找IP地址(与DNS服务器的连接减至一行)

```
import java.net.*;
public class getIP
{
  public static void main(String args[])
  {
     InetAddress iscas=null;
     try{
        iscas=InetAddress.getByName(“www.ucas.ac.cn”);
     }catch(UnknownHostException e) {}
    System.out.println(iscas);
  }
}
```

URI-Uniform Resource Identifier

- URI类
  - 统一资源标识符，用于标识/解析一个web资源
  - URL-Uniform Resource Locator
    - 统一资源定位符。能够精确的定位数据的URI,用于通信,即用于访问Internet网上资源
  - URN-Uniform Resource Name
    - 统一资源名称。除了URL的URI
  - 绝对URI：以scheme（后面跟冒号）开头
  - 相对URI：不以scheme开始

URI类

- 格式1：不透明的

  [scheme:]scheme-specific-part[#fragment]

  news:comp.lang.java、mailto:test@noname.com

- 格式2：分层的

  [scheme:][//authority][path][?query][#fragment]

  ftp://guest@noname.com:1234/public/notes?text=shakespeare#hamlet

- 常用方法：
  - •getScheme()获得scheme;
  - •getSchemeSpecificPart()
  - •getPath()
  - •getAuthority()

- URL类
  - 在Internet上的所有网络资源都是用URL(Uniform Resource Locator)来表示的，在Applet的网络编程中也是相对重要的
  - 在因特网上使用通用资源定位符URL（Uniform Resource Locator）来查找资源。
  - URL包含了用于查找某个资源的信息，如一张图片、一个文件等。

构造方法

- **URL(String, String, int, String)**
  - 第一个String类型的参数是协议的类型，可以是http，ftp，file等
  - 第二个String类型参数是主机名
  - int类型参数是指定端口号
  - 最后一个参数是给出文件名或路径名
- **URL(String, String, String)**
  - 参数含义与上相同，使用缺省端口号
- **URL(URL, String)**
  - 使用给出的URL和相对路径，String类型参数是相对路径
- **URL(String)**
  - 使用URL字符串构造一个URL类

- **String getProtocol()** 获取协议
- **String getHost()** 获取主机名
- **int getPort()** 获取端口号
- **String getFile()** 获取文件名
- **String getRef()** 获取参考点
- **String toString()**转换为字符串
- **boolean sameFile(URL other)** 比较两个URL
- **DataInputStream openStream()** 用于读取URL的数据

```
01.//URL&URI
02.
03.import java.net.*;
04.import java.util.*;
05.
06.public class URLTest1 {
07.
08.    public static void main(String[] args) throws Exception {
09.        URL url = new URL("http://www.ucas.ac.cn");
10.        Scanner in = new Scanner(url.openStream());
11.        while (in.hasNextLine()) {
12.            String str = in.nextLine();
13.            System.out.println(str);
14.        }
15.
```

```
15.
16.         URI uri = new URI("http://www.ucas.edu.cn/site/79");
17.         System.out.println(uri.getScheme());
18.         System.out.println(uri.getSchemeSpecificPart());
19.         System.out.println(uri.getAuthority());
20.         System.out.println(uri.getUserInfo());
21.         System.out.println(uri.getHost());
22.         System.out.println(uri.getPort());
23.         System.out.println(uri.getPath());
24.         System.out.println(uri.getQuery());
25.         System.out.println(uri.getFragment());
26.
27.         String str = "/site/195";
28.         URI combined = uri.resolve(str);// 根据uri的路径把str变成绝对地址
29.         System.out.println(combined.getScheme()
30.                 + combined.getSchemeSpecificPart());
31.
32.         URI relative = uri.relativize(new URI(str));
33.         System.out.println(relative.getSchemeSpecificPart());
34.
35.     }
36.
37.}
```

URL

- 获取网络资源:

  URL url = new URL("http://www.ucas.edu.cn/site/79");

  Scanner in = new Scanner(url.openStream());

- 获取消息头信息

  •URLConnection connection = url.openConnection();

  •connection.getHeaderFields()返回一个Map<String,List<String>>

  •connection.getContentLength();

  •connection.getContentType();

  •connection.setDoOutput(true)获得输出流

  •connection.getOutputStream();

  •connection.getInputStream();

```
01.//URL&URLConnection
02.
03.import java.net.*;
04.import sun.misc.*;
05.import java.util.*;
06.import java.io.*;
07.
08.public class URLConnectionTest {
09.
10.    public static void main(String[] args) throws Exception {
11.        String urlName = "http://www.ucas.ac.cn";
12.        URL url = new URL(urlName);
13.        URLConnection connection = url.openConnection();
14.        Map<String, List<String>> map = connection.getHeaderFields();
15.        for (Map.Entry<String, List<String>> entry : map.entrySet()) {
16.            String key = entry.getKey();
17.            List<String> value = entry.getValue();
18.            System.out.println(key + ":" + value);
19.        }
20.
21.    }
22.
23.}
```

- URLConnection类
  - 用来获取远程对象的属性
  - **创建URLConnection对象**
    - 利用URL类的**openConnection()**方法
    - 用URLConnection的构造方法
      - 还必须使用URLConnection类中的connect方法建立连接。
  - **getDate()　getContentType()**
  - **getExpiration()　getLastmodified()**
  - **getContentLength()　getInputStream()**
  - **getOuputStream()**

```
import java.net.*;
import java.util.*;
import java.io.*;
class urldemo{
public static void main(String args[]) throws Exception {
    int c;
    URL url=new
URL(“http://cre.ucas.ac.cn/images/articles/2019/201904/203916_685715.jpg");
    URLConnection urlcon=url.openConnection();
    System.out.println("content-type:"+urlcon.getContentType());
    int len=urlcon.getContentLength();
    System.out.println("context-length:"+len);
```

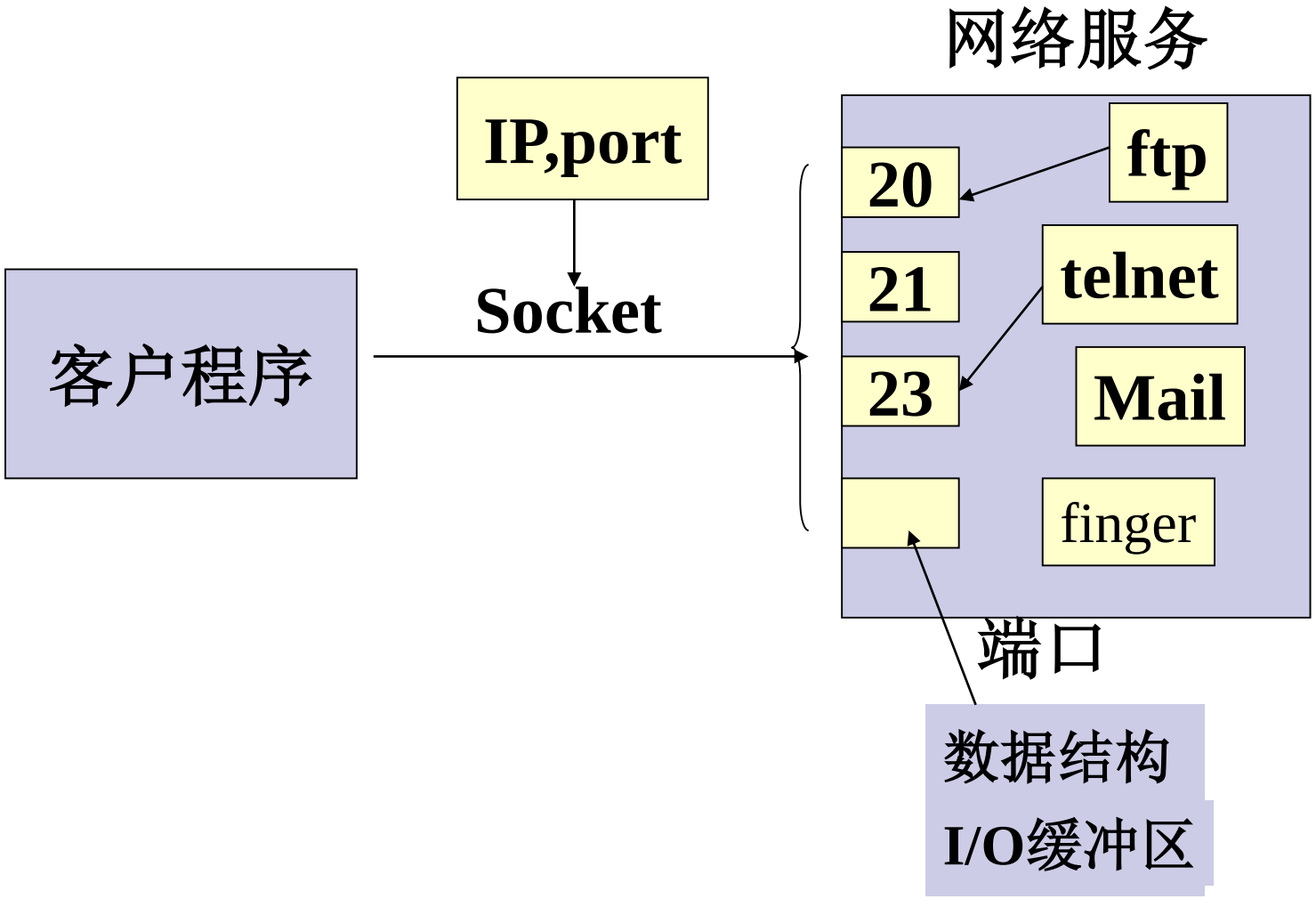
```
        System.out.println("======content------");
        InputStream input=urlcon.getInputStream();
            int i=600;
            while(((c=input.read())!=-1) && (--i>0)){
                System.out.print((char)c);
            }
            input.close();
            try {c=System.in.read();} catch (IOException e ){}
        }
}
```

- 通过URL类的openStream()方法，只能从网络资源中读取数据。
- 如果想同时对数据进行读写操作，则必须利用URLConnection类建立一个URL连接。
- 当与一个URL对象建立连接时，首先要通过URL对象的openConnection()方法生成URLConnection对象。
- URLConnection对象表示Java程序和URL对象在网络上的通信连接。如果连接过程失败，将会产生IOException。

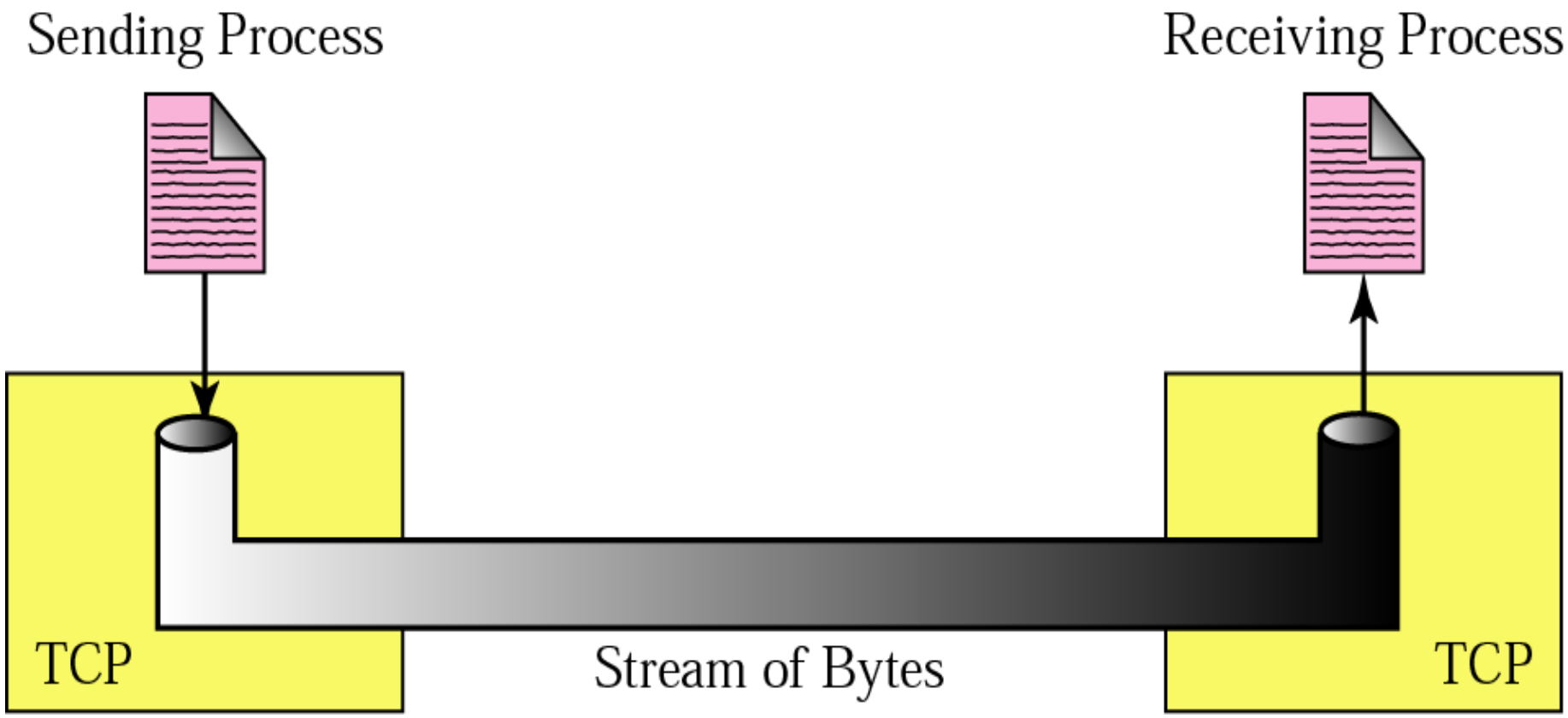
```
import java.net.*;
import sun.misc.*;
import java.util.*;
import java.io.*;
public class URLConnectionTest{
        public static void main(String[] args){
            try{
                URL ucas = new URL("http://www.ucas.edu.cn/");
                URLConnection conn = ucas.openConnection();
                //创建数据输入流
                DataInputStream dis = new DataInputStream(ucas.getInputStream);
                String inputLine;
                while((inputLine = dis.readLine())!=null){
                    System.out.print(inputLine);
                }
                dis.close();
            }catch(MalformedURLException me){
                System.out.print("MalformedURLException:"+me);
            }catch(IOException ioe){
                System.out.print("IOException:"+ioe);
            }
        }
}
```

- URLEncoder编码
- URLDecoder解码

- Socket类
  - 针对client/server(客户端/服务器)模式的应用以及实现某些特殊的协议的应用，它的通讯过程是基于TCP/IP协议中传输层接口socket实现的。


网络服务
IP,port
Socket
客户程序
20
21
23
ftp
telnet
Mail
finger
端口
数据结构
I/O缓冲区

进程与进程之间建立起连接



- Socket因为是基于传输层，所以它是比较原始的通讯协议机制。
- 通过Socket的数据表现形式为字节流信息，因此通讯双方要想完成某项具体的应用则必须按双方约定的方式进行数据的格式化和解释
  - 使用Socket编程比较麻烦，但是具有更强的灵活性和更广泛的使用领域。

- 传输层数据报文结构：TCP

TCP报文格式

16位源端口号								16位目的端口号
32位序号								
32位确认序号								
4位首部长度	保留（6位）	URG	ACK	PSH	RST	SYN	FIN	1位窗口大小
16位校验和								16位紧急指针
选项								
数据								

- Socket类
  - 用在客户端，用户通过构造一个Socket类来建立与服务器的连接。
    - 优点：所有数据都能准确、有序地送到接收方
    - 缺点：速度较慢。

- **Socket类**
  - 构造方法
    - Socket(String, int)
      - 构造一个连接指定主机、指定端口的流Socket。
    - Socket(InetAddress, int)
      - 构造一个连接指定Internet地址、指定端口的流Socket
  - **InetAddress getInetAddress()　　int getPort()**
  - **int getLocalPort()**
  - **getInputStream()　　getOuputStream()**
  - **void close()**

- ServerSocket类
  - **ServerSocket**类用在服务器端，接收用户端传送的数据
  - 构造方法
    - **ServerSocket(int)**
      - 在指定端口上构造一个ServerSocket类。
    - **ServerSocket(int, int)**
      - 在指定端口上构造一个ServerSocket类，并进入监听状态
      - 第二个int类型的参数是监听时间长度。
  - **accept()**方法
    - 等待客户初始化,返回一个Socket,用于与客户进行通讯.

- **建立连接**
  - 首先，在服务端构造一个ServerSocket类，在指定端口上进行监听，这将使线程处于等待状态。
  - 然后在客户端构造Socket类，与服务器上的指定端口进行连接。
  - 服务器监听到连接请求后，就可在两者之间建立连接。
    - 建立连接的具体细节对用户是透明的，用户只需要构造这两个类就可以了。
  - 在建立连接之后，就可以建立相应的输入、输出流，两端通过流来进行通讯。

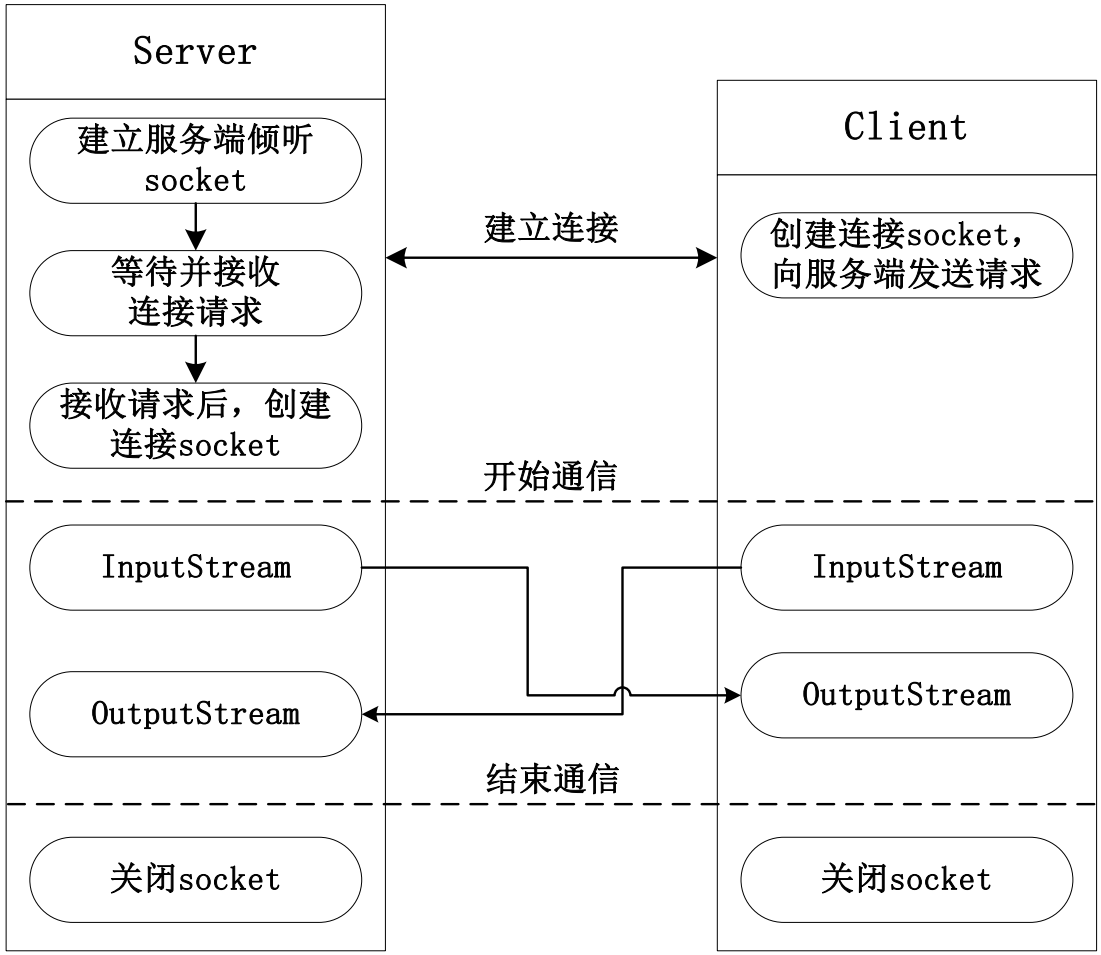
Socket主要方法：

在客户端：建立socket连接后,还应该建立输入输出数据流。

getInputStream( ) 获得输入流
getOutputStream( ) 获得输出流

在服务器端：

ServerSocket类的accept( )方法使服务器处于阻塞状态,等待用户请求。


Server
Client
建立服务端倾听socket
等待并接收连接请求
接收请求后，创建连接socket
建立连接
创建连接socket，向服务端发送请求
开始通信
InputStream
OutputStream
InputStream
OutputStream
结束通信
关闭socket
关闭socket

Java网络模型图

- 服务端Socket程序实现流程
  - 首先调用ServerSocket类以某个端口号为参数，创建一个ServerSocket对象，即是服务器端的服务程序在该指定端口监听的Socket。
  - 使用ServerSocket对象的accept()方法，接收来自客户机程序的连接请求，此时服务器端将一直保持停滞状态，直到收到客户端发来的连接请求，此时该方法将返回一个新建的Socket类的实例，代表和客户机建立的通讯链路在服务程序内的通讯端点。
    - 如果采用Java的多线程编程方法，可以实现并发服务器，继续监听来自其他客户的连接请求。
  - 使用新建的Socket对象创建输入、输出流对象。
  - 使用流对象的方法完成和客户端的数据传输，按约定协议识别并处理来自客户端的请求数据，并把处理的结果返回给客户端。
  - 客户端工作完毕后，则服务器端程序关闭和客户端通讯的流和通讯的Socket。
  - 在服务器程序运行结束前关闭用来监听的Socket.

- 客户端Socket程序流程
  - 首先调用Socket类的构造函数，以服务器的指定的IP地址或指定的主机名和指定的端口号为参数，创建一个Socket流，在创建Socket流的过程中包含了向服务器请求建立通讯连接的过程实现。
  - 使用Socket的方法getInputStream()和getOutputStream()来创建输入/输出流。这样，使用Socket类后，网络输入输出也转化为使用流对象的过程。
  - 使用输入输出流对象的相应方法读写字节流数据，因为流连接着通讯所用的Socket，Socket又是和服务器端建立连接的一个端点，因此数据将通过连接从服务器得到或发向服务器。这时我们就可以对字节流数据按客户端和服务器之间的协议进行处理，完成双方的通讯任务。
  - 待通讯任务完毕后，用流对象的close()方法来关闭用于网络通讯的输入输出流,在用Socket对象的close()方法来关闭Socket。

服务器上的程序：

```
ServerSocket server;
Socket ssocket;
try {
   server = new ServerSocket(80);
     ssocket = server.accept();
     InputStream in = ssocket.getInputStream();
   OutputStream out = ssocket.getOutputStream();
   out.write(.....);
}
catch (IOException e) {       }
```

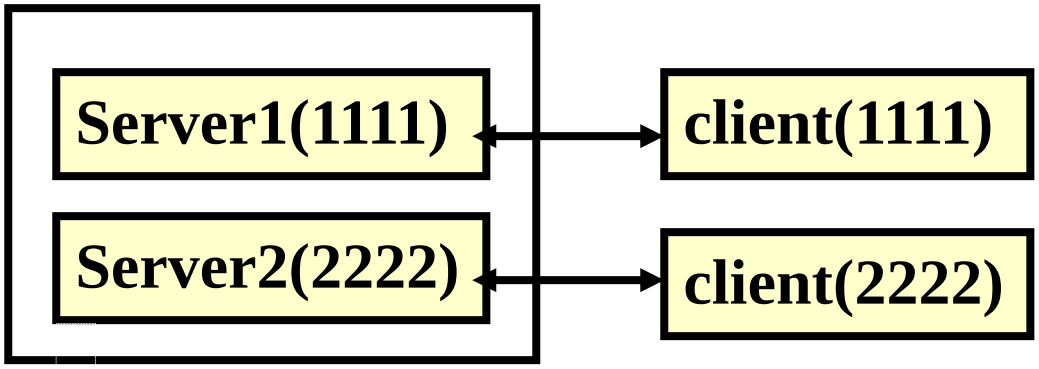
客户端的程序：

```
Socket csocket;
try {
   cSocket = new Socket (“127.0.0.1",80);
   InputStream in = cSocket.getInputStream();
   OnputStream out = cSocket.getOnputStream();
   System.out.println("client reads: " + in.read());
}
catch (UnknownHostException e) {       }
catch (IOException e) {
}
```

考虑多用户

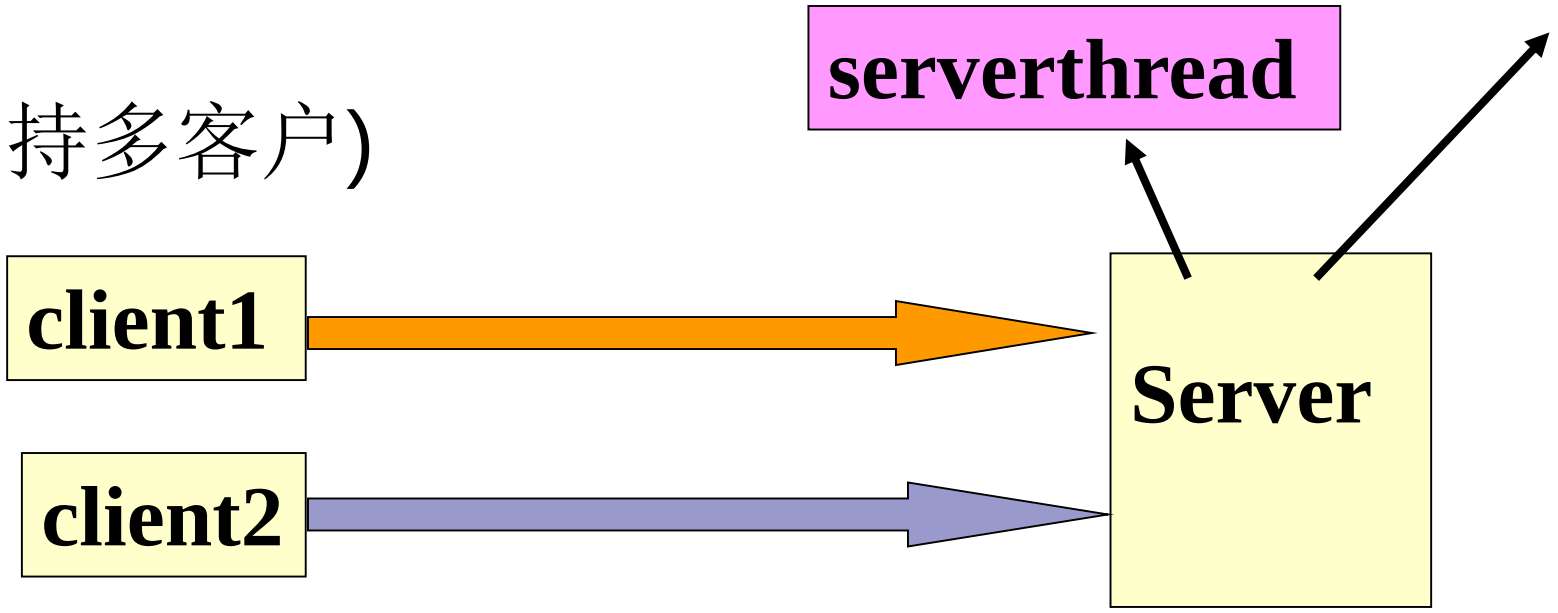
第一种解决方案:

- 一台计算机上一次启动多个服务器程序,只是端口号不同。



多用户的第二种方案

(支持多客户)



将服务器写成多线程的,不同的线程为不同的客户服务.

main()只负责循环等待

线程负责网络连接,接收客户输入的信息

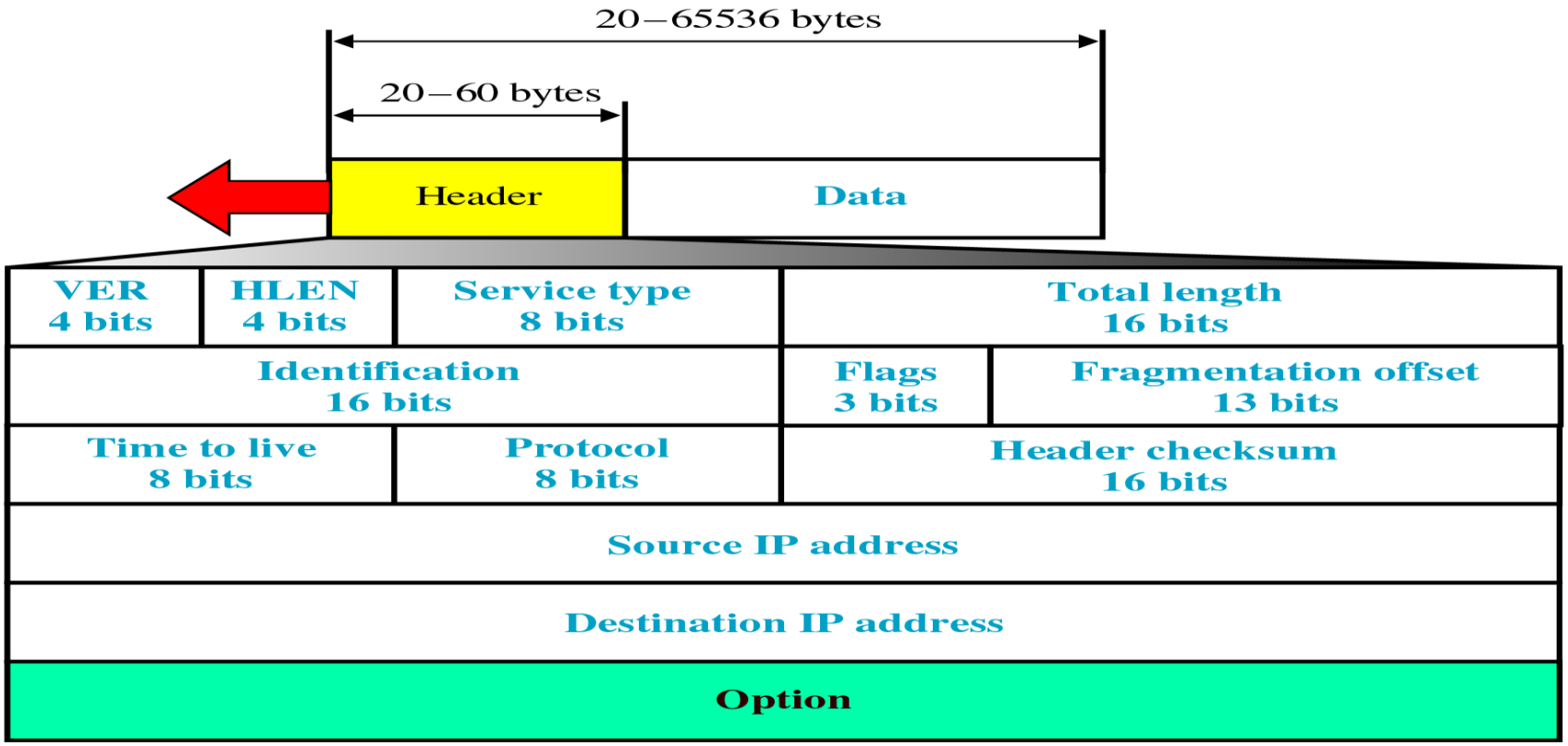
```
import java.net.*;
import java.io.*;
public class MultiUser extends Thread {
    private Socket client;
    public MultiUser(Socket c) {
      this.client = c;
    }
    public void run() {
      try {
         BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
         PrintWriter out = new PrintWriter(client.getOutputStream());
         while (true) {
            String str = in.readLine();
            System.out.println(str);
            out.println("receive...");
            out.flush();
            if (str.equals("end"))
               break;
          }
         client.close();
      } catch (IOException ex) {
      } finally {
      }
    }
   public static void main(String[] args) throws IOException {
      ServerSocket server = new ServerSocket(6678);
      while (true) {
        MultiUser mu = new MultiUser(server.accept());
        mu.start();
      }
   }
}
```

- 数据报
  - 数据报是一种无连接的通信方式
  - 速度比较快
  - 但是由于不建立连接，不能保证所有数据都能送到目的地。
    - 一般用于传送非关键性的数据。

- 在UDP中，要使用二个类
  - DatagramSocket 类
    - 发送时，用 send( )方法发送数据；
    - 接收时，用 receive( )方法接收数据。
  - DatagramPacket 类　用于打包或拆包
    - 发送时打包：包由数据、接收地址、端口号组成；
    - 接收时拆包：取出包中的数据、接收地址、端口号。

- UDP通信(无连接的数据报)
  - 对于类似传输速度更重要的应用，使用无连接的数据报协议UDP，即“用户数据报协议”。
    - UDP并不刻意追求数据包会完全发送出去，也不能担保它们抵达的顺序与它们发出时一样，因此，这是一种“不可靠协议”。由于其速度比TCP快得多，所以还是能够在很多应用中使用。
  - Java对数据报的支持与它对TCP套接字的支持大致相同，使用DatagramSocket类来表示无连接的socket，接收和发送数据报。接收和要发送的数据报内容保存在DatagramPacket对象中。
  - UDP也有自己的端口，和TCP端口是相互独立的。也就是说，可以在端口8080同时运行一个TCP和UDP服务程序，两者之间不会产生冲突。

- 数据报结构


20–65536 bytes
20–60 bytes
Header
Data
VER
4 bits
HLEN
4 bits
Service type
8 bits
Total length
16 bits
Identification
16 bits
Flags
3 bits
Fragmentation offset
13 bits
Time to live
8 bits
Protocol
8 bits
Header checksum
16 bits
Source IP address
Destination IP address
Option

- **DatagramPacket类(数据包容器)**
  - 是进行数据报通信的基本单位，它包含了需要传送的数据、数据报的长度、IP地和端口等。
  - 构造方法:
    - **DatagramPacket(byte [], int)**
      - 构造一个用于**接收数据报**的DatagramPacket类
      - byte []类型的参数是接收数据报的缓冲
      - int类型的参数是接收的字节数。
    - **DatagramPacket(byte [], int, InetAddress, int)**
      - 构造一个用于**发送数据**的DatagramPacktet类
      - byte []类型参数是发送数据的缓冲区
      - int类型参数是发送的字节数
      - InetAddress类型参数是接收机器的Internet地址
      - 最后一个参数是接收的端口号。

- 其他方法
  - **InetAddress getInetAddress()**
    - 获取数据报中的**IP**地址。
  - **int getPort()**
    - 获取数据报中的端口号。
  - **Byte[] getData()**
    - 获取存放在数据报中的数据。获取存放在数据报中的数据。
  - **void setData(byte []buf)**
    - 设置数据报中的内容为**buf**所存储的内容。
  - **int getLength()**
    - 获取数据的长度。

- **DatagramSocket类(发送或接收DatagramPacket的机制)**
  - 构造方法
    - **DatagramSocket()**
      - 创建一个套接字，绑定到本地地址和一个随机的端口号
    - **DatagramSocket(int port)**
      - 创建一个套接字，绑定到特定端口号
    - **DatagramSocket(int port, InetAddress iad)**
      - 创建一个套接字，绑定到特定的端口号及指定地址
    - **DatagramSocket(SocketAddress sad)**
      - 创建一个套接字，绑定到特定的套接字地址
  - 构造完DatagramSocket类后，就可以发送和接收数据报。
    - **send()**
    - **receive()**

- **UDP发送和接收过程**
  - 接收端
    - 建立一个接收的**DatagramSocket**，在**指定端口**上监听
    - 构造一个**DatagramPacket**类指定接收的缓冲区（DatagramSocket的监听将阻塞线程
  - 发送端
    - 构造**DatagramPacket**类，指定要发送的数据、数据长度、接收主机地址及端口号
    - 使用**DatagramSocket**类来发送数据报。
      - 接收端接收到后，将数据保存到缓冲区，发送方的主机地址和端口号一并保存。
      - 随后将接收到的数据报返回给发送方，并附上接收缓冲区地址，缓冲长度、发送方地址和端口号等信息，等待新的数据。

接收端的程序：

byte [] inbuffer = new byte[1024]; //接收缓冲

DatagramPacket inpacket = new DatagramPacket(inbuffer, inbuffer.length);

DatagramSocket insocket = new DatagramSocket(80);

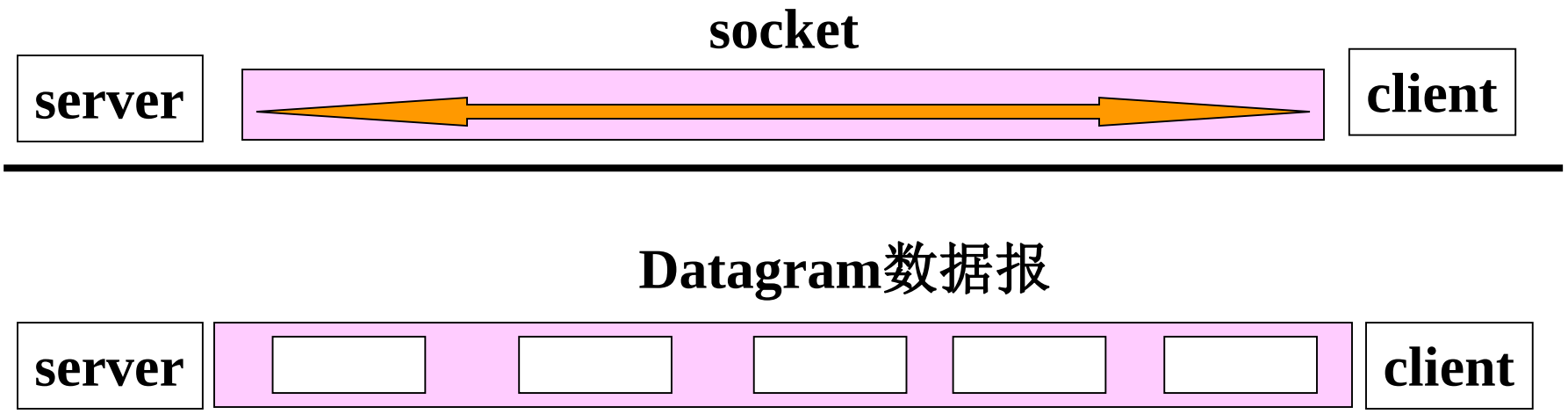
insocket.receive(inpacket); //监听数据

String s = new String(inbuffer, 0, 0, inpacket.getlength); //将接收的数据存入字符串s

- 发送端的程序：

```
DatagramPacket outpacket = new
DatagramPacket(message, 200, “127.0.0.1", 80);
DatagramSocket outsocket = new DatagramSocket(xxx);
outsocket.send(outpacket);
```

- TCP的可靠来源于确认和超时重传.要求先建立连接,传输后关闭,次序不变.
- UDP不需要实现连接,数据报是独立的,没有次序要求.



- httpserver
- httpclient
- Javamail
- …….