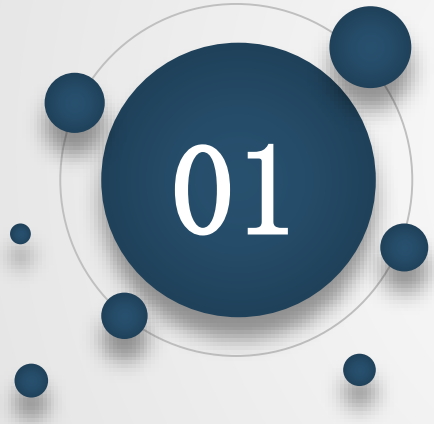

java.net

目录



基本概念

CONCEPT

网络与通信协议

PART ONE

网络

将不同区域的电脑连接到一起， 组成局域网、城域网或广域网。把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件、软件、数据信息等资源。

- 资源共享
- 信息传输与集中处理
- 负载均衡与分布处理

网络

➢ 通讯协议

计算机网络中实现通信必须有一些约定即通信协议，对速率、传输代码、代码结构、传输控制步骤、出错控制等制定标准。

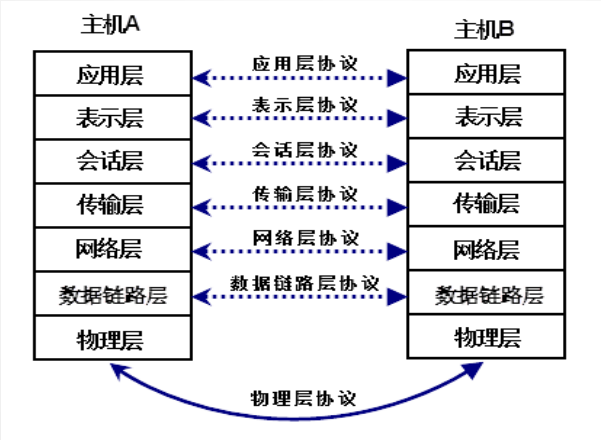
➢ 通信接口

为了使两个结点之间能进行对话，必须在它们之间建立通信工具(即接口)，使彼此之间能进行信息交换。接口包括两部分：

- 硬件装置：实现结点之间的信息传送；
- 软件装置：规定双方进行通信的约定协议

网络分层

由于结点之间联系很复杂，在制定协议时，把复杂成份分解成 一些简单的成份，再将它们复合起来。最常用的复合方式是**层次方式**，即同层间可以通信、上一层可以调用下一层，而与再下一层不发生关系。



OSI参考模式:开放系统互连参考模型（Open System Interconnect）

网络分层

TCP/IP 是一个协议族，也是按照层次划分，共四层：应用层，传输层，互连网络层，网络接口层（物理+数据链路层）。

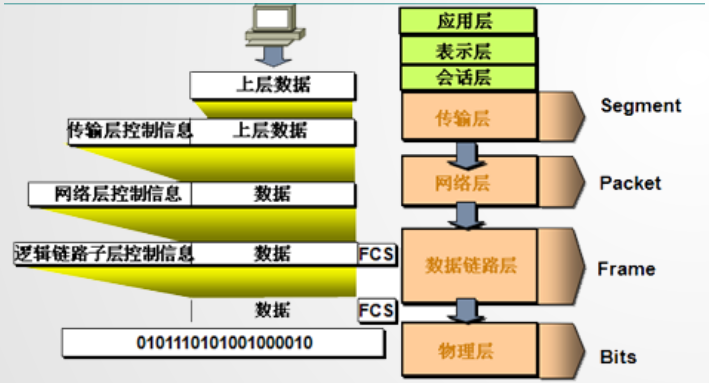
OSI网络通信协议模型，是一个参考模型，而TCP/IP协议是事实上的标准。TCP/IP协议参考了OSI 模型，但是并没有严格按照OSI规定的七层标准去划分，而只划分了四层，这样会更简单点，当划分太多层次时，你很难区分某个协议是属于哪个层次的。

TCP/IP参考模型:传输控制/网际协议 Transfer Controln Protocol/Internet Protocol

数据封装

Data Encapsulation是指将协议数据单元（PDU）封装在一组协议头和协议尾中的过程。在OSI七层参考模型中，每层主要负责与其它机器上的对等层进行通信。该过程是在协议数据单元（PDU）中实现的，其中每层的PDU一般由本层的协议头、协议尾和数据封装构成。

由于用户传输的数据一般都比较大，有的可以达到MB字节，一次性发送出去十分困难，于是就需要把数据分成许多片段，再按照一定的次序发送出去。这个过程就需要对数据进行封装



- 应用层:准备数据
- 传输层:接收应用层数据添加上TCP的控制信息（称为TCP头部），这个数据单元称为段（Segment），加入控制信息的过程称为封装。由此，将段交给网络层。
- 网络层:接收到段，再添加上IP头部，这个数据单元称为包（Packet）。然后，将包交给数据链路层。
- 数据链路层:将包再添加上MAC头部和尾部，这个数据单元称为帧（Frame）。然后，将帧交给物理层。
- 物理层:将接收到的数据转化为比特流，然后在网线中传送。

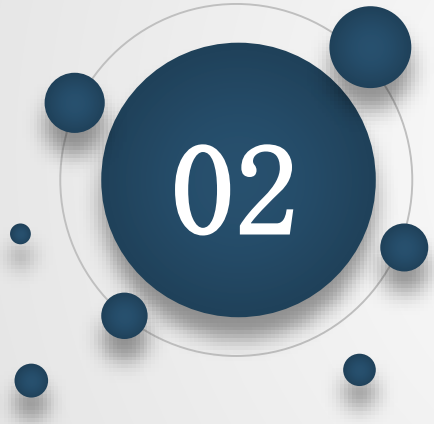
发送方数据处理的方式是从高层到底层，逐层进行数据封装

数据拆封

Data dismantling是指将接收到的数据进行拆包，每一层只把对该层有意义的数据拿走，或者说每一层只能处理发送方同等层的数据，然后把其余的部分传递给上一层，这就是对等层通信的概念。

- 物理层:接收到比特流，经过处理后将数据交给数据链路层。
- 数据链路层:将接收到的数据转化为数据帧，再除去MAC头部和尾部，这个除去控制信息的过程称为解封，然后将包交给网络层。
- 网络层:接收到包，再除去IP头部，然后将段交给传输层。
- 传输层:接收到段，再除去TCP头部，然后将数据交给应用层。
- 应用层:处理数据

接收方数据处理的方式是从底层到高层，逐层进行数据解封装



常用类操作

API

IP、Port、URL

PART TWO

IP地址

用来标识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。 比如互联网的每个服务器都要有自己的IP地址，而每个局域网的计算机要通信也要配置IP地址。路由器是连接两个或多个网络的网络设备。

- IP地址分类
  - IPV4：32位地址，以点分十进制表示，如192.168.0.1
  - IPV6：128位（16个字节）写成8个16位的无符号整数，每个整数用四个十六进制位表示，数之间用冒号（：）分开，如：3ffe:3201:1401:1280:c8ff:fe4d:db39:1984
- 特殊的IP
  - 127.0.0.1 本机地址
  - 192.168.0.0--192.168.255.255私有地址，属于非注册地址，专门为组织机构内部使用。

InetAddress

- 封装计算机的ip地址 ,没有端口

```
//使用getLocalHost方法创建InetAddress对象
InetAddress addr = InetAddress.getLocalHost();
System.out.println(addr.getHostAddress());  //返回：192.168.1.110
System.out.println(addr.getHostName());  //输出计算机名

//根据域名得到InetAddress对象
addr = InetAddress.getByName(“www.163.com”);
System.out.println(addr.getHostAddress());  //返回 163服务器的ip:61.135.253.15
System.out.println(addr.getHostName());  //输出：www.163.com

//根据ip得到InetAddress对象
addr = InetAddress.getByName(“61.135.253.15”);
System.out.println(addr.getHostAddress());  //返回 163服务器的ip:61.135.253.15
System.out.println(addr.getHostName());  //输出ip而不是域名。如果这个IP地 址不存在或DNS服务器不允许进行IP地址和域名的映射，getHostName方法就直接返回这个IP地址。
```

端口

IP地址用来标识一台计算机，但是一台计算机上可能提供多种网络应用程序，如何来区分这些不同的程序呢？这就要用到端口。

端口是虚拟的概念，并不是说在主机上真的有若干个端口。通过端口，可以在一个主机上运行多个网络应用程序。 端口的表示是一个16位的二进制整数，2个字节，对应十进制的0-65535。

Oracle、MySQL、Tomcat、QQ、msn、迅雷、360等网络程序都有自己的端口：

- 公认端口 0—1023 比如80端口分配给WWW，21端口分配给FTP
- 注册端口 1024—49151 分配给用户进程或应用程序
- 动态/私有端口 49152--65535

房间号
分机号

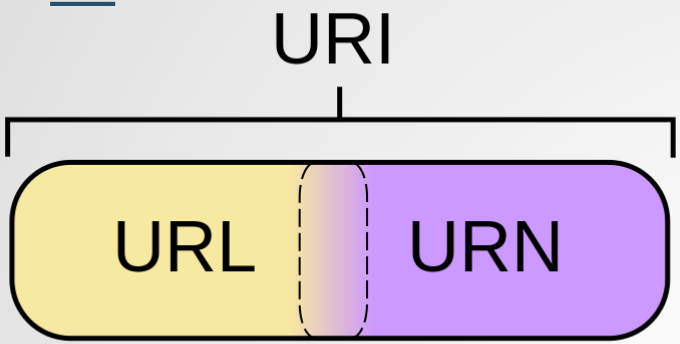
- 查看所有端口:netstat -ano
- 查看指定端口:netstat -aon|findstr "808"
- 查看指定进程:tasklist|findstr "808"
- 查看具体程序:使用任务管理器查看PID

InetSocketAddress

- 包含端口，用于socket通信的

```
//包含端口
InetSocketAddress socketAddress = new
InetSocketAddress("127.0.0.1",8080);
InetSocketAddress socketAddress2 = new
InetSocketAddress(“localhost”,9000);
System.out.println(socketAddress.getHostName());
System.out.println(socketAddress2.getAddress());
```

URL



- URI: Universal Resource Identifier 统一资源标志符，用来标识抽象或物理资源的一个紧凑字符串。
- URL:Universal Resource Locator 统一资源定位符，一种定位资源的主要访问机制的字符串，一个标准的URL必须包括：protocol、host、port、path、parameter、anchor。
- URN:Universal Resource Name 统一资源名称，通过特定命名空间中的唯一名称或ID来标识资源。

在www上，每一信息资源都有统一且唯一的地址，即统一资源定位符Uniform Resource Locator 。
如:http://www.google.com:80/index.html,由4部分组成:

- 协议
- 存放资源的主机域名
- 端口号
- 资源文件名

```
http://www.example.jp/info/contact.html
(1)  (2)              (4)
http://www.example.jp/info/
(1)  (2)              (4)
http://www.example.jp/info/contact.html?arg1=1&arg2=abc
(1)  (2)              (4)                  (5)
http://www.example.jp:8080/info/
(1)  (2)              (3)  (4)
http://www.サンプル.jp/info/
(1)  (2)              (4)
http://192.168.0.1/info/
(1)  (2)          (4)
ftp://www.example.jp/info/
(1)  (2)              (4)
https://www.example.jp/info/
(1)    (2)            (4)
```

网络三大基石:html 、http、url

URL

- 统一资源定位符,由4部分组成：协议 、存放资源的主机域名、端口号和资源文件名。

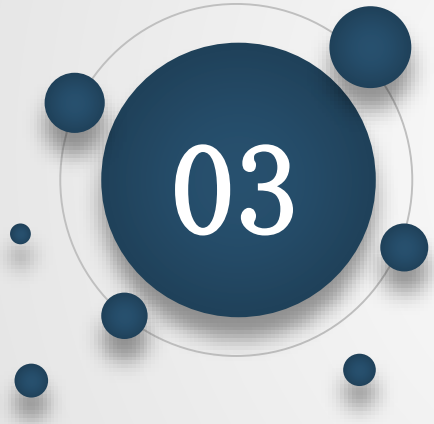
```
URL u = new URL("http://www.baidu.con:80/index.html#aa?cansu=shsxt");
System.out.println("获取与此url关联的协议的默认端口：
"+u.getDefaultPort());
System.out.println(“getFile:”+u.getFile());  //端口号后面的内容
System.out.println("主机名："+u.getHost());  //www.baidu.com
System.out.println(“路径：”+u.getPath());  //端口号后，参数前的内容
System.out.println(“端口：”+u.getPort());  //存在返回80.否则返回-1
System.out.println("协议："+u.getProtocol());
System.out.println("参数部分："+u.getQuery());
System.out.println("锚点："+u.getRef());
URL u = new URL("http://www.abc.com/aa/");
URL u2 = new URL(u,“2.html”);  //相对路径构建url对象
System.out.println(u2.toString());  //http://www.abc.com/aa/2.html
```

URL


```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();;
conn.setRequestMethod("GET");
        conn.setUseCaches(false);
        conn.setReadTimeout(8000);
        conn.setConnectTimeout(8000);
        conn.setInstanceFollowRedirects(false);
        conn.setRequestProperty("User-Agent","Mozilla/5.0 (Windows NT 10.0; WOW64; rv:46.0) Gecko/20100101 Firefox/46.0");
```

API

方法	功能
InetAddress	封装计算机的IP地址和DNS（没有端口信息）。 常用方法：getLocalHost()、getByName()、 getAllByName()、getAddress()、getHostName()
InetSocketAddress	包含IP和端口信息，常用于Socket通信。此类实现 IP 套接字地址（IP 地址 + 端口号），不依赖任何协议。 常用方法：getHostName()、getAddress()
URL	类 URL 代表一个统一资源定位符，它是指向互联网“资源”的指针。资源可以是简单的文件或目录，也可以是对更为复杂的对象的引用，例如对数据库或搜索引擎的查询。 常用方法：getDefaultPort()、getFile()、getHost()、getPath()、getPort()、getProtocol()、getQuery()、getRef()



传输协议

PROTOCOL

UDP、TCP

PART THREE

TCP传输协议

- TCP(transfer control protocol)

一种面向连接（连接导向）的、可靠的、基于字节流的运输层（Transport layer）通信协议 。

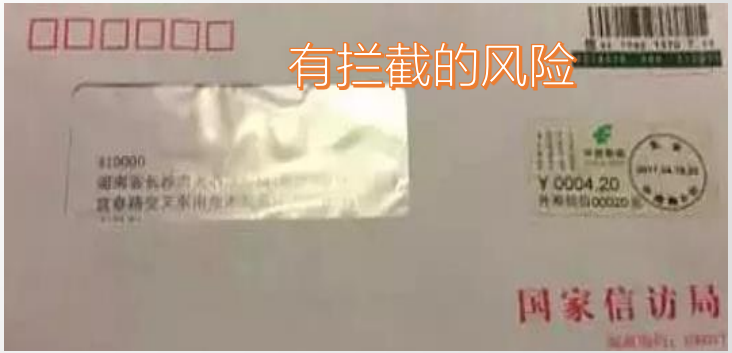
- 特点
  - 面向连接
  - 点到点的通信
  - 高可靠性
  - 占用系统资源多、效率低

UDP传输协议

- UDP(User DatagramProtocol)

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

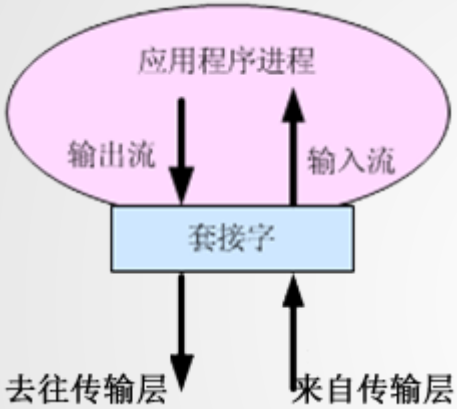
- 特点
  - 非面向连接，传输不可靠，可能丢失
  - 发送不管对方是否准备好，接收方收到也不确认
  - 可以广播发送
  - 非常简单的协议，开销小

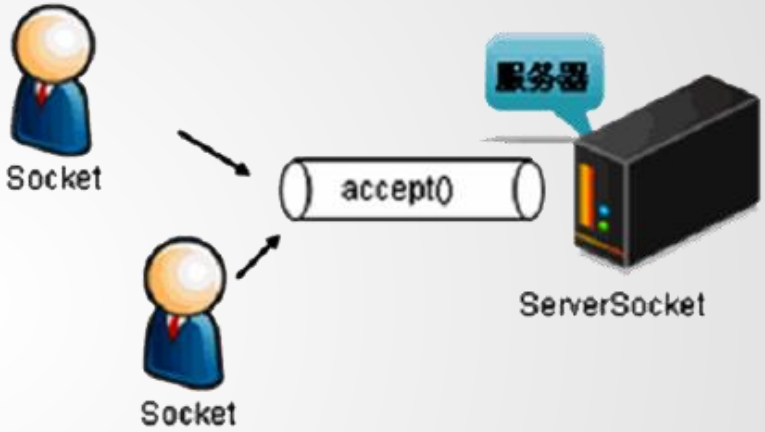


套接字Socket

- 我们开发的网络应用程序位于应用层，TCP和UDP属于传输层协议，在应用层如何使用传输层的服务呢？在应用层和传输层之间，则是使用套接字来进行分离。

套接字Socket





- 套接字就像是传输层为应用层开的一个小口，应用程序通过这个小口向远程发送数据，或者接收远程发来的数据；而这个小口以内，也就是数据进入这个口之后，或者数据从这个口出来之前，是不知道也不需要知道的，也不会关心它如何传输，这属于网络其它层次的工作。

SOCKET编程

基于TCP协议的Socket编程	基于UDP协议的Socket编程
通信双发需要建立连接	通信双发不需要建立连接
连接建立时双方存在主次之分	通信双方完全平等
114查号台	QQ聊天模式

UDP编程

PROGRAMMING

UDP

PART FOUR

UDP编程

需求：完成在线咨询功能：

- 学生和咨询师在线一对一交流

分析

- 使用基于UDP协议的Socket网络编程实现
- 不需要利用IO流实现数据的传输
- 每个数据发送单元被统一封装成数据包的方式，发送方将数据包发送到网络中，数据包在网络中去寻找他的目的地。

UDP基本概念

- DatagramSocket：用于发送或接收数据包的套接字
- DatagramPacket：数据包

UDP编程

```
/**客户端向服务器端发送信息（最基本的操作） */
public class UDPClient {
	public static void main(String[] args) throws Exception {
	byte[] b = "aaaa".getBytes();
//必须告诉数据包要发到哪里去
	DatagramPacket dp = new DatagramPacket(b,b.length,new InetSocketAddress("localhost",8999));
//我本身占用9000端口向外面机器发数据包
		DatagramSocket ds = new DatagramSocket(9000);
		ds.send(dp);
		ds.close();
	}
}
```

```
/**客户端向服务器端发送信息（最基本的操作） */
public class UDPServer {
	public static void main(String[] args) throws Exception {
		DatagramSocket ds = new DatagramSocket(8999);
		byte[] b = new byte[1024];
		DatagramPacket dp = new DatagramPacket(b,b.length);

		ds.receive(dp);  //阻塞式方法
		String string = new String(dp.getData(),0,dp.getLength());
//dp.getLength()返回实际收到的数据的字节数
		System.out.println(string);
		ds.close();
	}
}
```

UDP编程

```
/**客户端向服务器端发送信息（传递对象） */
public class Client {
public static void main(String[] args) throws
Exception {
        Person person = new Person(20,"aa");
        ByteArrayOutputStream bos = new
ByteArrayOutputStream();
        ObjectOutputStream oos = new
ObjectOutputStream(bos);
        oos.writeObject(person);
        byte[] b = bos.toByteArray();
        //必须告诉数据包要发到哪里去
        DatagramPacket dp = new
DatagramPacket(b,b.length,new
InetSocketAddress("localhc
        //我本身占用9000端
        DatagramSocket ds
        ds.send(dp);
        ds.close();
        }
}
```

```
/**客户端向服务器端发送信息（传递对象） */
public class Server {
        public static void main(String[] args)
throws Exception {
        DatagramSocket ds = new
DatagramSocket(8999);
        byte[] b = new byte[1024];
        DatagramPacket dp = new
DatagramPacket(b,b.length);
        ds.receive(dp);  //阻塞式方法
        ByteArrayInputStream bis = new
ByteArrayInputStream(dp.getData());
        ObjectInputStream ois = new
m(bis);
rson = (Person) ois.readObject();
t.println(person.name);
s.close();
```

```
/**传递对象 */
class Person implements Serializable{
        int age;
        String name;
}
```

UDP编程

TCP编程

PROGRAMMING

TCP

PART FIVE

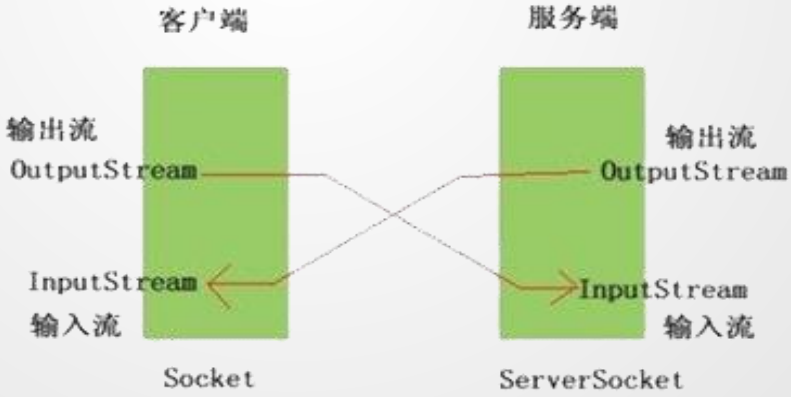
TCP编程

需求：完成网络登录功能：

- 用户输入用户名密码，服务器给出登录成功或失败的提示

分析

- 使用基于TCP协议的Socket网络编程实现
- TCP协议基于请求-响应模式
- 在网络通讯中，第一次主动发起通讯的程序被称作客户端(Client)程序
- 第一次通讯中等待连接的程序被称作服务器端(Server)程序
- 利用IO流实现数据的传输



TCP编程

详细步骤(通信原理)

- 服务器创建ServerSocket，在指定端口监听并并处理请求
- 客户端创建Socket，向服务器发送请求

TCP编程

网络登录功能分解

- 单向：客户端向服务器端发送字符串，服务器获取字符串并输出
- 双向：服务器端给出客户端反馈，客户端得到反馈并输出
- 文件：客户端向服务器端上传文件，服务器端获取文件并反馈结果
- 多线程：服务器接收多个客户端的请求，并给出反馈　每个客户请求开启一个线程

TCP编程

```
/** 最简单的服务器端代码*/
public class BasicSocketServer {
    public static void main(String[] args) {
        try {   //建立服务器端套接字
                ServerSocket serverSocket = new ServerSocket(8888);
        //tcp端口一共多少个??
                //监听，等待客户端请求，并愿意接收连接。
                System.out.println(“服务端建立监听”);
                Socket socket = serverSocket.accept();
                //通过流向客户端发送数据
                //ObjectOutputStream oos = new
        ObjectOutputStream(socket.getOutputStream());
                //oos.writeObject(“aaaaa”);
                //oos.close();
                BufferedWriter bw = new BufferedWriter(new
        OutputStreamWriter(socket.getOutputStream()));
                        bw.write("hhhh");
                        bw.close();
                        socket.close();
                } catch (IOException e) {e.printStackTrace();}
        }
    }
```

TCP编程

```
/** 最简单的Socket客户端 */
public class BasicSocketClient {
        public static void main(String[] args) {
        try {
        //指定的是所要连接的服务器的ip和端口。
        //而不是自己机器的端口。发送端口是随机的。
        Socket socket = new Socket(InetAddress.getLocalHost(),8888);
        //ObjectInputStream ois = new
                ObjectInputStream(socket.getInputStream());
        //String string = (String) ois.readObject();
        //System.out.println(string);
        BufferedReader br = new BufferedReader(new
                InputStreamReader(socket.getInputStream()));
        System.out.println(br.readLine());
        br.close();
        socket.close();
        } catch (Exception e) {e.printStackTrace();}
        }
}
```

TCP编程

```
//文件上传服务端
ServerSocket server = new ServerSocket(8888);
Socket socket = server.accept();
InputStream in = socket.getInputStream();
OutputStream out = new
BufferedOutputStream(new
FileOutputStream(new File("src/1.jpg")));
byte[] flush = new byte[1024];
int len = 0;
while (-1 != (len = in.read(flush))) {
            out.write(flush, 0, len);
            out.flush();
}
```

```
//文件上传客户端
Socket client = new Socket("127.0.0.1", 8888);
File file = new  File("D:/12.jpg");
InputStream  in = new BufferedInputStream(new
FileInputStream(file));
OutputStream out = client.getOutputStream();
byte[] flush = new byte[1024];
int len  = 0;
while (-1 != (len = in.read(flush))) {
            out.write(flush,0, len);
}
out.flush();
out.close();
in.close();
```

TCP编程

```
/**客户端和服务器端可以双向交流的Socket程序：*/
public class Server {
public static void main(String[] args) throws Exception {
        ServerSocket server=new ServerSocket(8888);
        Socket socket=server.accept();
        BufferedReader in=new BufferedReader(new
                                InputStreamReader(socket.getInputStream()));
        BufferedWriter out=new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        while(true){
                String str=in.readLine();
                System.out.println("客户端说："+str);
                String str2 = "";
                str2 = br.readLine();   //读到\n为止，因此一定要输入换行符！
                out.write(str2+"\n");
                out.flush();
                if(str.equals("end"))
                        break;
        } in.close();   out.close();socket.close();
}}
```

TCP编程

```
/**客户端和服务器端可以双向交流的Socket程序： */
public class Client {
        static Socket server;
        public static void main(String[] args) throws Exception {
                server=new Socket(InetAddress.getLocalHost(),8888);
                BufferedReader in=new BufferedReader(new
InputStreamReader(server.getInputStream()));
                BufferedWriter out=new BufferedWriter(new
OutputStreamWriter(server.getOutputStream()));
                BufferedReader wt=new BufferedReader(new
InputStreamReader(System.in));
                while(true){
                        String str=wt.readLine();
                         out.write(str+“\r\n” );  //一定要加。不加的话，那边按
照readLine读的话，如果没读到\n就会一直阻塞！
                         out.flush();
                         if(str.equals("end")) break;
                         System.out.println("服务器发说："+in.readLine());
                 }
                 out.close(); in.close(); wt.close(); server.close();
}}
```

在线聊天室

CHAT

综合案例

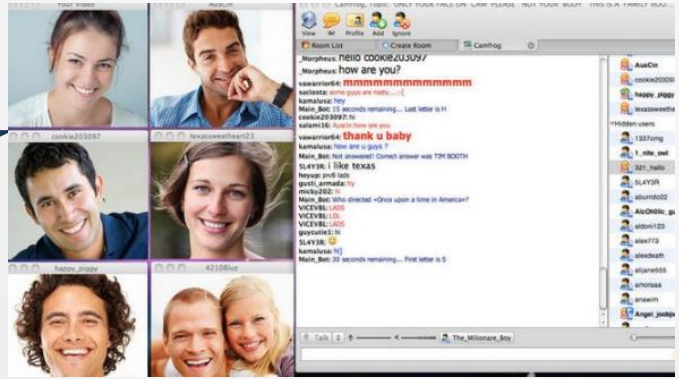
PART SIX

聊天室

使用TCP的Socket实现一个聊天室！

- 服务器端：一个线程专门发送消息，一个线程专门接收消息。
- 客户端：一个线程专门发送消息，一个线程专门接收消息。

群聊



私聊

感谢您的支持与信任

THANK YOU FOR WATCHING