

聊聊Socket、TCP/IP、HTTP、FTP及网络编程

1 这些都是什么

既然是网络传输，涉及几个系统之间的交互，那么首先要考虑的是如何准确的定位到网络上的一台或几台主机，另一个是如何进行可靠高效

的数据传输。这里就要使用到TCP/IP协议。

1.1 TCP/IP协议组

TCP/IP协议（传输控制协议）由网络层的IP协议和传输层的TCP协议组成。

IP层负责网络主机的定位，数据传输的路由，由IP地址可以唯一的确定Internet上的一台主机。

TCP层负责面向应用的可靠的或非可靠的数据传输机制，这是网络编程的主要对象。

TCP/IP是个协议组，可分为三个层次：网络层、传输层和应用层：

网络层：IP协议、ICMP协议、ARP协议、RARP协议和BOOTP协议；

传输层：TCP协议与UDP协议；

应用层：FTP、HTTP、TELNET、SMTP、DNS等协议；



HTTP是应用层协议，其传输都是被包装成TCP协议传输。可以用SOCKET实现HTTP。SOCKET是实现传输层协议的一种编程API，可以是TCP，也可以是UDP。

1.2 TCP

TCP — 传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先双方在之间建立一个TCP连接，之后

才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。理想状态下，TCP连接一

旦建立，在通信双方中的任何一方主动关闭连接前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连

接的请求。

TCP是一种面向连接的保证可靠传输的协议。通过TCP协议传输，得到的是一个顺序的无差错的数据流。发送方和接收方的成对的两个

socket之间必须建立连接，以便在TCP协议的基础上进行通信，当一个socket（通常都是server socket）等待建立连接时，另一个socket

可以要求进行连接，一旦这两个socket连接起来，它们就可以进行双向数据传输，双方都可以进行发送或接收操作。

TCP特点：

1. TCP是面向连接的协议，通过三次握手建立连接，通讯完成时要拆除连接，由于TCP是面向连接协议，所以只能用于点对点的通讯。而且建立连接也需要消耗时间和开销。
2. TCP传输数据无大小限制，进行大数据传输。
3. TCP是一个可靠的协议，它能保证接收方能够完整正确地接收到发送方发送的全部数据。

要了解TCP，一定要知道“三次握手，四次拜拜”所谓的三次握手，就是发送数据前必须建立的连接叫三次握手，握手完了才开始发的，这

也就是面向连接的意思。

第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN (ack=j+1)，同时自己也发送一个SYN包 (syn=k)，即SYN+ACK包，此时服务器进入SYN_RECV状态；

第三次握手：客户端收到服务器的SYN + ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手；

【适用情况】

TCP发送的包有序号，对方收到包后要给一个反馈，如果超过一定时间还没收到反馈就自动执行超时重发，因此TCP最大的优点是可靠。一

般网页（http）、邮件（SMTP）、远程连接(Telnet)、文件(FTP)传送就用TCP

TCP在网络通信上有极强的生命力，例如远程连接（Telnet）和文件传输（FTP）都需要不定长度的数据被可靠地传输。但是可靠的传输是要

付出代价的，对数据内容正确性的检验必然占用计算机的处理时间和网络的带宽，因此TCP传输的效率不如UDP高。

1.3 UDP

UDP — 用户数据报协议，是一个无连接的简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出

去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传

输速度很快。

UDP是一种面向无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的

地，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的。

UDP特点：

1. UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。
2. UDP传输数据时有大小限制，每个被传输的数据报必须限定在64KB之内。
3. UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

【适用情况】

UDP是面向消息的协议，通信时不需要建立连接，数据的传输自然是不可靠的，UDP一般用于多点通信和实时的数据业务，比如语音广播、

视频、QQ、TFTP(简单文件传送)、SNMP（简单网络管理协议）、RTP（实时传送协议）RIP（路由信息协议，如报告股票市场，航空信

息）、DNS(域名解释)。注重速度流畅。

UDP操作简单，而且仅需要较少的监护，因此通常用于局域网高可靠性的分散系统中client/server应用程序。例如视频会议系统，并不要求

音频视频数据绝对的正确，只要保证连贯性就可以了，这种情况下显然使用UDP会更合理一些。

1.4 Socket

Socket通常也称作“套接字”，用于描述IP地址和端口，是一个通信链的句柄。网络上的两个程序通过一个双向的通讯连接实现数据的交

换，这个双向链路的一端称为一个Socket，一个Socket由一个IP地址和一个端口号唯一确定。应用程序通常通过“套接字”向网络发出请求

或者应答网络请求。Socket是TCP/IP协议的一个十分流行的编程界面，但是，Socket所支持的协议种类也不光TCP/IP一种，因此两者之间

是没有必然联系的。在Java环境下，Socket编程主要是指基于TCP/IP协议的网络编程。

Socket通讯过程：服务端监听某个端口是否有连接请求，客户端向服务端发送连接请求，服务端收到连接请求向客户端发出接收消息，这样

一个连接就建立起来了。客户端和服务端都可以相互发送消息与对方进行通讯。

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的

TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。

由于通常情况下Socket连接就是TCP连接，因此Socket连接一旦建立，通信双方即可开始相互发送数据内容，直到双方连接断开。但在实际

网络应用中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于

非活跃状态的连接而导致 Socket 连接断连，因此需要通过轮询告诉网络，该连接处于活跃状态。

1. **套接字 (socket) 概念：**套接字 (socket) 是通信的基石，是支持TCP/IP协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。应用层通过传输层进行数据通信时，TCP会遇到同时为多个应用程序进程提供并发服务的问题。多个TCP连接或多个应用程序进程可能需要通过同一个TCP协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与TCP / IP协议交互提供了套接字(Socket)接口。应用层可以和传输层通过Socket接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。
2. **建立socket连接：**建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket，另一个运行于服务器端，称为ServerSocket。套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。

服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求；

客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

连接确认：当服务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，双方就正式建立连接。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

3. **SOCKET连接与TCP连接**创建Socket连接时，可以指定使用的传输层协议，Socket可以支持不同的传输层协议（TCP或UDP），当使用TCP协议进行连接时，该Socket连接就是一个TCP连接。

【适用情况】

很多情况下，需要服务器端主动向客户端推送数据，保持客户端与服务器数据的实时与同步。此时若双方建立的是Socket连接，服务器就可

以直接将数据传送给客户端；

1.5 HTTP

HTTP协议是建立在TCP协议之上的一种应用，HTTP连接使用的是“请求—响应”的方式，不仅在请求时需要先建立TCP连接，而且需要客

户端向服务器发出请求后，请求中包含请求方法、URI、协议版本以及相关的MIME样式的消息，服务器端才能回复数据，包含消息的协议版

本、一个成功和失败码以及相关的MIME式样的消息。在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

由于HTTP在每次请求结束后都会主动释放连接，因此HTTP连接是一种“短连接”，要保持客户端程序的在线状态，需要不断地向服务器发

起连接请求。通常的做法是即时不需要获得任何数据，客户端也保持每隔一段固定的时间向服务器发送一次“保持连接”的请求，服务器在

收到该请求后对客户端进行回复，表明知道客户端“在线”。若服务器长时间无法收到客户端的请求，则认为客户端“下线”，若客户端长

时间无法收到服务器的回复，则认为网络已经断开。

HTTP/1.0为每一次HTTP的请求/响应建立一条新的TCP链接，因此一个包含HTML内容和图片的页面将需要建立多次的短期的TCP链接。—

次TCP链接的建立将需要3次握手。

另外，为了获得适当的传输速度，则需要TCP花费额外的回路链接时间（RTT）。每一次链接的建立需要这种经常性的开销，而其并不带有

实际有用的数据，只是保证链接的可靠性，因此HTTP/1.1提出了可持续链接的实现方法。HTTP/1.1将只建立一次TCP的链接而重复地使用

它传输一系列的请求/响应消息，因此减少了链接建立的次数和经常性的链接开销。

结论：HTTP是应用层协议，其传输都是被包装成TCP协议传输。可以用SOCKET实现HTTP。SOCKET是实现传输层协议的一种编程API，可以是TCP，也可以是UDP。

【适用情况】

若双方建立的是HTTP连接，则服务器需要等到客户端发送一次请求后才能将数据传回给客户端，因此，客户端定时向服务器端发送连接请

求，不仅可以保持在线，同时也是在“询问”服务器是否有新的数据，如果有就将数据传给客户端。

1.6 FTP

文件传输协议（File Transfer Protocol, FTP）是TCP/IP网络上两台计算机传送文件的协议，FTP是在TCP/IP网络和INTERNET上最早使用的

协议之一，它属于网络协议组的应用层。FTP客户机可以给服务器发出命令来下载文件，上载文件，创建或改变服务器上的目录。

2 N层交换技术

2.1 二层交换

交换原理：根据第二层数据链路层的MAC地址来实现端到端的数据交换；

工作流程：

- (1) 交换机某端口收到数据包，读取源MAC地址，得到源MAC地址机器所连端口；
- (2) 读取目的MAC地址，在地址表中查找对应端口；
- (3) 如果地址表中有目的MAC地址对应端口，直接复制数据至此端口；
- (4) 如果地址表中没有目的MAC地址对应端口，广播所有端口，当目的机器回应时，更新地址表，下次就不需要广播了；

不断的循环上述过程，全网的MAC地址信息都可以学习到，二层交换机就这样学习和维护它的地址表。第二层交换机根据MAC选择端口转发数据，算法又很简单，其方便采用廉价芯片实现，且速度快。

2.2 三层交换

交换原理：根据第三层网络层的IP地址来完成端到端的数据交换；

场景：A(ip1) => 三层交换机 => B(ip2)

工作流程：

- (1) A发数据给B，根据B的ip地址+子网掩码，A能够判断出B和自己是否在同一个网段；
- (2) B如果和A在同一个网段内，但A不知道B的MAC地址，A会发送一个ARP请求，以获取B的MAC地址，并根据MAC通过二层交换机将数据发送给B；
- (3) B如果和A不在同一个网段内，且不知道B的MAC地址，A会将数据包发送给网关（A的本地一定有网关的MAC地址）。网关收到数据包后，将源MAC地址会修改为网关自己的MAC地址，目的IP对应的MAC地址为目的MAC地址，以完成数据交换。

看似第三层交换机是第二层交换机+路由功能的组合，实际并非这样：数据通过第三层转发设备后，会记录IP与MAC的映射关系，下次需要

转发时，不会再经过第三层设备。

2.3 四层交换

二层和三层交换设备都是基于端到端的交换，这种基于IP和MAC地址的交换技术，有着很高效传输率，但是缺乏根据目的主机应用需求动态

交换数据的功能。

四层设备不但能够完成端到端的交换，还能够根据目的主机的应用特点，分配或限制其流量；

四层设备基于传输层数据包交换，是一类建立在TCP/IP应用层至上，实现用户应用需求的设备；它实现一类应用层的访问控制与质量保证服

务，与其说它是硬件设备，不如说它是软件网络管理系统。

四层交换核心技术

1. 包过滤利用四层信息定义过滤规则，能够控制指定端口的TCP/UDP通信，它可以在高速芯片中实现，极大提高包过滤速率；
2. 包优先级三层以下设备只有MAC，PORT，IP等信息，因为缺乏四层信息，无法确认TCP/IP等四层优先级信息；四层设备允许基于目的地址/端口（即应用服务）的组合来区分优先级。
3. 负载均衡将附加有负载均衡服务的IP地址，通过不同的物理服务做成一个集群，提供相同的服务，并将其定义为一个单独的虚拟服务器；这个虚拟服务器是一个有独立IP的逻辑服务器，用户数据流只需要流向虚拟服务器IP，而不与物理服务器进行通信；
只有通过交换机执行网络地址转换（NAT）后，才能得到真实访问；

虚拟服务器组里转换通信流量实现均衡，其中具体关系到OSPF、RIP、VRRP等协议；

4. 主机备用连接同（3）所含技术类似，可以实现主备同IP自动切换；

2.4 七层交换

交换原理：比四层更进一步，可以根据应用层的数据报文来完成更多的复杂交换功能（例如根据http报文路由）。由于七层交换还没有具体的标准，文章也不多展开啦。

3 JDK Socket

在java.net包下有两个类：Socket和ServerSocket。ServerSocket用于服务器端，Socket是建立网络连接时使用的。在连接成功时，应用

程序两端都会产生一个Socket实例，操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别，不因为在服

务器端或在客户端而产生不同级别。不管是Socket还是ServerSocket它们的工作都是通过SocketImpl类及其子类完成的。

列出几个常用的构造方法：

```
1 Socket(InetAddress address,intport);  
  //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  
2  
3 Socket(String host,intport);  
  //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  
4  
5 Socket(InetAddress address,intport, InetAddress localAddr,intlocalPort);  
  //创建一个套接字并将其连接到指定远程地址上的指定远程端口  
6  
7 Socket(String host,intport, InetAddress localAddr,intlocalPort);  
  //创建一个套接字并将其连接到指定远程主机上的指定远程端口  
8  
9 Socket(SocketImpl impl);  
  //使用用户指定的 SocketImpl 创建一个未连接 Socket  
10  
11 ServerSocket(intport);  
  //创建绑定到特定端口的服务器套接字
```

```
ServerSocket(int port, int backlog);  
//利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号  
  
ServerSocket(int port, int backlog, InetAddress bindAddr);  
//使用指定的端口、侦听 backlog 和要绑定到的本地 IP地址创建服务器
```

构造方法的参数中，address、host和port分别是双向连接中另一方的IP地址、主机名和端口号，stream指明socket是流socket还是数据

报socket，localPort表示本地主机的端口号，localAddr和bindAddr是本地机器的地址（ServerSocket的主机地址），impl是socket的

父类，既可以用来创建serverSocket又可以用来创建Socket。count则表示服务端所能支持的最大连接数。

注意：必须小心选择端口号。每一个端口提供一种特定的服务，只有给出正确的端口，才能获得相应的服务。0~1023的端口号为系统所保

留，例如http服务的端口号为80,telnet服务的端口号为21,ftp服务的端口号为23,所以我们在选择端口号时，最好选择一个大于1023的数以

防止发生冲突。

几个重要的Socket方法：

```
1 public InputStream getInputStream();  
  //方法获得网络连接输入，同时返回一个InputStream对象实例  
2  
3 public OutputStream getOutputStream();  
  //方法连接的另一端将得到输入，同时返回一个OutputStream对象实例  
  
public Socket accept();  
  //用于产生"阻塞"，直到接受到一个连接，并且返回一个客户端的Socket对象实例。
```

“阻塞”是一个术语，它使程序运行暂时“停留”在这个地方，直到一个会话产生，然后程序继续；通常“阻塞”是由循环产生的。

注意：其中getInputStream和getOutputStream方法均会产生一个IOException，它必须被捕获，因为它们返回的流对象，通常都会被另一个流对象使用。

4 基本的Client/Server程序

以下是一个基本的客户端/服务器端程序代码。主要实现了服务器端一直监听某个端口，等待客户端连接请求。客户端根据IP地址和端口号连接服务器端，从键盘上输入一行信息，发送到服务器端，然后接收服务器端返回的信息，最后结束会话。这个程序一次只能接受一个客户连接。

客户端程序：

```
package sock;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.Socket;
```



```

    public class SocketClient {
    public static void main(String[] args) {
        try {
            /** 创建Socket*/

// 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(本处是本机)

            Socket socket =new Socket("127.0.0.1",2013);

// 60s超时

            socket.setSoTimeout(60000);

            /** 发送客户端准备传输的信息 */

// 由Socket对象得到输出流，并构造PrintWriter对象

            PrintWriter printWriter =new PrintWriter(socket.getOutputStream(),true);
            // 将输入读入的字符串输出到Server

            BufferedReader sysBuff =new BufferedReader(new InputStreamReader(System.in
                printWriter.println(sysBuff.readLine()));
// 刷新输出流，使Server马上收到该字符串

            printWriter.flush();

            /** 用于获取服务端传输来的信息 */

// 由Socket对象得到输入流，并构造相应的BufferedReader对象

            BufferedReader bufferedReader =new BufferedReader(new InputStreamReader(so
                // 输入读入一字符串

            String result = bufferedReader.readLine();
            System.out.println("Server say : " + result);

            /** 关闭Socket*/
            printWriter.close();
            bufferedReader.close();
            socket.close();
        }catch (Exception e) {
            System.out.println("Exception:" + e);
        }
    }
}

```

```
}

```

服务器端程序:

```
package sock;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServer {
    public static void main(String[] args) {
        try {
            /** 创建ServerSocket*/

            // 创建一个ServerSocket在端口2013监听客户请求

            ServerSocket serverSocket =new ServerSocket(2013);
            while (true) {

                // 侦听并接受到此Socket的连接,请求到来则产生一个Socket对象,并继续执行

                Socket socket = serverSocket.accept();

                /** 获取客户端传来的信息 */

                // 由Socket对象得到输入流,并构造相应的BufferedReader对象

                BufferedReader bufferedReader =new BufferedReader(new InputSteamRea

                // 获取从客户端读入的字符串

                String result = bufferedReader.readLine();
                System.out.println("Client say : " + result);

                /** 发送服务端准备传输的 */

                // 由Socket对象得到输出流,并构造PrintWriter对象

                PrintWriter printWriter =new PrintWriter(socket.getOutputStream());
                printWriter.print("hello Client, I am Server!");
                printWriter.flush();

                /** 关闭Socket*/
            }
        }
    }
}
```

```

        printWriter.close();
        bufferedReader.close();
        socket.close();
    }
} catch (Exception e) {
    System.out.println("Exception:" + e);
}finally{
    //serverSocket.close();
}
}
}

```

5 多客户端连接服务器

上面的服务器端程序一次只能连接一个客户端，这在实际应用中显然是不可能的。通常的网络环境是多个客户端连接到某个主机进行通讯，

所以我们要对上面的程序进行改造。

设计思路：服务器端主程序监听某一个端口，客户端发起连接请求，服务器端主程序接收请求，同时构造一个线程类，用于接管会话。当一

个Socket会话产生后，这个会话就会交给线程进行处理，主程序继续进行监听。

下面的实现程序流程是：客户端和服务器建立连接，客户端发送消息，服务端根据消息进行处理并返回消息，若客户端申请关闭，则服务器

关闭此连接，双方通讯结束。

客户端程序：

```

package sock;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class SocketClient {
    public static void main(String[] args) {
        try {
            Socket socket =new Socket("127.0.0.1",2013);

```

```

        socket.setSoTimeout(60000);

        PrintWriter printWriter =new PrintWriter(socket.getOutputStream(),true);

        BufferedReader bufferedReader =new BufferedReader(new InputStreamReader(so
            String result ="";
        while(result.indexOf("bye") == -1){
            BufferedReader sysBuff =new BufferedReader(new InputStreamReader(Syste
                printWriter.println(sysBuff.readLine());
            printWriter.flush();
            result = bufferedReader.readLine();
            System.out.println("Server say : " + result);
        }

        printWriter.close();
        bufferedReader.close();
        socket.close();
    }catch (Exception e) {
        System.out.println("Exception:" + e);
    }
}
}

```

服务器端程序:

```

package sock;
import java.io.*;
import java.net.*;

public classServerextendsServerSocket {
    private static final int SERVER_PORT =2013;

    public Server()throws IOException {
        super(SERVER_PORT);

        try {
            while (true) {
                Socket socket = accept();
                new CreateServerThread(socket);
//当有请求时，启一个线程处理

            }
        }catch (IOException e) {
        }finally {
            close();
        }
    }
}

```

```
}
```

```
//线程类
```

```
class CreateServerThread extends Thread {
    private Socket client;
    private BufferedReader bufferedReader;
    private PrintWriter printWriter;

    public CreateServerThread(Socket s) throws IOException {
        client = s;

        bufferedReader = new BufferedReader(new InputStreamReader(client.getInputStream()));
        printWriter = new PrintWriter(client.getOutputStream(), true);
        System.out.println("Client(" + getName() + ") come in...");

        start();
    }

    public void run() {
        try {
            String line = bufferedReader.readLine();

            while (!line.equals("bye")) {
                printWriter.println("continue, Client(" + getName() + ")!");
                line = bufferedReader.readLine();
                System.out.println("Client(" + getName() + ") say: " + line);
            }
            printWriter.println("bye, Client(" + getName() + ")!");

            System.out.println("Client(" + getName() + ") exit!");
            printWriter.close();
            bufferedReader.close();
            client.close();
        } catch (IOException e) {
        }
    }
}

public static void main(String[] args) throws IOException {
    new Server();
}
}
```

6 信息群发共享

以上虽然实现了多个客户端和服务端连接，但是仍然是消息在一个客户端和服务端之间相互传播。现在我们要实现信息共享，即服务器可以

向多个客户端发送广播消息，客户端也可以向其他客户端发送消息。类似于聊天室的那种功能，实现信息能在多个客户端之间共享。

设计思路：客户端循环可以不停输入向服务器发送消息，并且启一个线程，专门用来监听服务器端发来的消息并打印输出。服务器端启动

时，启动一个监听何时需要向客户端发送消息的线程。每次接受客户端连接请求，都启一个线程进行处理，并且将客户端信息存放到公共集

合中。当客户端发送消息时，服务器端将消息顺序存入队列中，当需要输出时，从队列中取出广播到各客户端处。客户端输入showuser命令

可以查看在线用户列表，输入bye向服务器端申请退出连接。

客户端代码：

```
package sock;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class SocketClient extends Socket{

    private static final String SERVER_IP ="127.0.0.1";
    private static final int SERVER_PORT =2013;

    private Socket client;
    private PrintWriter out;
    private BufferedReader in;

    /**
     * 与服务器连接，并输入发送消息
     */
    public SocketClient()throws Exception{
        super(SERVER_IP, SERVER_PORT);
        client =this;
        out =new PrintWriter(this.getOutputStream(),true);
        in =new BufferedReader(new InputStreamReader(this.getInputStream()));
        new readLineThread();

        while(true){
```

```

        in =new BufferedReader(new InputStreamReader(System.in));

        String input = in.readLine();          out.println(input);
    }
}

/**
 * 用于监听服务器端向客户端发送消息线程类
 */
classreadLineThreadextendsThread{

    private BufferedReader buff;
    public readLineThread(){
        try {

            buff =new BufferedReader(new InputStreamReader(client.getInputStream())
                start();          }catch (Exception e) {
        }
    }

    @Override
    public void run() {
        try {
            while(true){
                String result = buff.readLine();
                if("byeClient".equals(result)){
//客户端申请退出，服务端返回确认退出

                    break;
                }else{
//输出服务端发送消息

                    System.out.println(result);
                }
            }
            in.close();
            out.close();
            client.close();
        }catch (Exception e) {
        }
    }
}

    public static void main(String[] args) {
        try {
            new SocketClient();
//启动客户端

```

```

        }catch (Exception e) {

        }

    }

}

```

服务器端代码：

```

package sock;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class Server extends ServerSocket{

    private static final int SERVER_PORT =2013;

    private static boolean isPrint =false;
    //是否输出消息标志

    private static List user_list =new ArrayList();
    //登录用户集合

    private static List<ServerThread> thread_list =new ArrayList<ServerThread>();
    //服务器已启用线程集合

    private static LinkedList<String> message_list =new LinkedList<String>();
    //存放消息队列

    /**
     * 创建服务端Socket,创建向客户端发送消息线程,监听客户端请求并处理
     */
    public Server()throws IOException{
        super(SERVER_PORT);
        //创建ServerSocket

        new PrintOutThread();
        //创建向客户端发送消息线程

```



```

        try {
            while(true){
//监听客户端请求，启个线程处理

                Socket socket = accept();
                new ServerThread(socket);
            }
        }catch (Exception e) {
        }finally{
            close();
        }
    }

/**
 * 监听是否有输出消息请求线程类,向客户端发送消息
 */
class PrintOutThread extends Thread{

    public PrintOutThread(){
        start();
    }

    @Override
    public void run() {
        while(true){
            if(isPrint){
//将缓存在队列中的消息按顺序发送到各客户端，并从队列中清除。

                String message = message_list.getFirst();
                for (ServerThread thread : thread_list) {
                    thread.sendMessage(message);
                }
                message_list.removeFirst();
                isPrint = message_list.size() >0 ?true :false;
            }
        }
    }
}

/**
 * 服务器线程类
 */
class ServerThread extends Thread{
    private Socket client;
    private PrintWriter out;
    private BufferedReader in;

```

```

private String name;

public ServerThread(Socket s) throws IOException{
    client = s;
    out =new PrintWriter(client.getOutputStream(),true);

    in =new BufferedReader(new InputStreamReader(client.getInputStream()));
    in.readLine();
    out.println("成功连上聊天室,请输入你的名字: ");
    start();
}

```

```

@Override
public void run() {
    try {
        int flag =0;
        String line = in.readLine();
        while(!"bye".equals(line)){

```

//查看在线用户列表

```

        if ("showuser".equals(line)) {
            out.println(this.listOnlineUsers());
            line = in.readLine();
        }

```

//第一次进入，保存名字

```

        if(flag++ ==0){
            name = line;
            user_list.add(name);
            thread_list.add(this);
            out.println(name +"你好,可以开始聊天了...");
            this.pushMessage("Client<" + name +">进入聊天室...");
        }else{
            this.pushMessage("Client<" + name +"> say : " + line);
        }
        line = in.readLine();
    }
    out.println("byeClient");
} catch (Exception e) {
    e.printStackTrace();
}finally{

```

//用户退出聊天室

```

    try {
        client.close();
    }catch (IOException e) {

```

```

        e.printStackTrace();
    }

    thread_list.remove(this);
    user_list.remove(name);
    pushMessage("Client<" + name + ">退出了聊天室");
}

}

//放入消息队列末尾，准备发送给客户端

    private void pushMessage(String msg){
        message_list.addLast(msg);
        isPrint =true;
    }

//向客户端发送一条消息

    private void sendMessage(String msg){
        out.println(msg);
    }

//统计在线用户列表

    private String listOnlineUsers() {
        String s = "--- 在线用户列表 ---1512";
        for (int i =0; i < user_list.size(); i++) {
            s += "[" + user_list.get(i) + "]"1512";
        }
        s += "-----";
        return s;
    }
}

    public static void main(String[] args)throws IOException {
        new Server();
//启动服务端

    }
}

```

7 文件传输

客户端向服务器端传送文件，服务端可获取文件名用于保存，获取文件大小计算传输进度，比较简单，直接贴代码。

客户端代码：

```
package sock;

import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.net.Socket;

/**
 * 客户端
 */
public class Client extends Socket{

    private static final String SERVER_IP ="127.0.0.1";
    private static final int SERVER_PORT =2013;

    private Socket client;
    private FileInputStream fis;
    private DataOutputStream dos;

    public Client(){
        try {
            try {
                client =new Socket(SERVER_IP, SERVER_PORT);

//向服务端传送文件

                File file =new File("c:/test.doc");
                fis =new FileInputStream(file);
                dos =new DataOutputStream(client.getOutputStream());

//文件名和长度

                dos.writeUTF(file.getName());
                dos.flush();
                dos.writeLong(file.length());
                dos.flush();

//传输文件

                byte[] sendBytes =new byte[1024];
```

```

        int length =0;
        while((length = fis.read(sendBytes,0, sendBytes.length)) >0){
            dos.write(sendBytes,0, length);
            dos.flush();
        }
    }catch (Exception e) {
        e.printStackTrace();
    }finally{
        if(fis !=null)
            fis.close();
        if(dos !=null)
            dos.close();
        client.close();
    }
}catch (Exception e) {
    e.printStackTrace();
}
}

public static void main(String[] args)throws Exception {
    new Client();
}
}

```

服务器端代码:

```

package sock;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * 服务器
 */
public class ServerextendsServerSocket{

    private static final int PORT =2013;

    private ServerSocket server;
    private Socket client;
    private DataInputStream dis;
    private FileOutputStream fos;

```

```

        publicServer()throws Exception{
    try {
        try {
            server =new ServerSocket(PORT);

            while(true){
                client = server.accept();

                dis =new DataInputStream(client.getInputStream());

//文件名和长度

                String fileName = dis.readUTF();
                long fileLength = dis.readLong();
                fos =new FileOutputStream(new File("d:" + fileName));

                byte[] sendBytes =new byte[1024];
                int transLen =0;

                System.out.println("----开始接收文件<" + fileName + ">,文件大小为<" +
                    while(true){
                        read = dis.read(sendBytes);
                        if(read == -1)
                            break;
                        transLen += read;

                        System.out.println("接收文件进度" +100 * transLen/fileLength +
                            fos.write(sendBytes,0, read);
                            fos.flush();
                        }
                System.out.println("----接收文件<" + fileName + ">成功-----");
                client.close();
            }
        }catch (Exception e) {
            e.printStackTrace();
        }finally {
            if(dis !=null)
                dis.close();
            if(fos !=null)
                fos.close();
            server.close();
        }
    }catch (Exception e) {
        e.printStackTrace();
    }
}

publicstaticvoidmain(String[] args)throws Exception {

```

```
|         new Server();  
|         }  
|     }
```