课程设计报告二：聊天室系统

1. **课程设计要求与目的**

目的：编写一个小型Java聊天室系统，掌握Java网络通信、多线程、IO文件操作等高级应用编程技能。

要求：以课本第15章 Java网络通信例15.3、15.4的源代码为基础，编写一个小型Java聊天室系统。

完成如下功能：

1、多客户端模式下，实现客户与客户的单独通信，要求信息通过服务器中转

2、端到端的通信，实现并行通信模式（不再是你说一句，我说一句，一端的信息发送不受另一端的影响）

3、实现端到端的文件传输

4、添加图形界面（选做）

**二、多客户端模式下，实现客户与客户的单独通信，通过服务器中转**

1系统设计

1）设计思想

A、客户端与服务器端都建立相应的Socket对象，客户端的Socket是通过显示的调用new完成创建，并指明服务器端的IP地址和端口号，因为客户端与服务端都是在本机完成的，所以，客户端中Socket指明的目的地址的IP和端口号为“127.0.0.1”4700。服务器端的Socket是通过ServerSocket的Accept方法产生的。

B、因为要实现客户端1与客户端2之间的通信，且要通过服务器作为中转，所以在服务器端建立监听线程（与书上的例15.4区别），当服务器端在端口监听到有客户端与之链接时，执行该线程响应客户的请求。

C、因为客户1与服务器建立连接后，会返回一个与客户1对应的Socket，客户2与服务器建立连接后，会返回一个与客户2对应的Socket，而要实现客户1与客户2通信，即要在服务器端保存与客户1对应的Socket，和与客户2对应的Socket，保存的实现通过Socket数组来完成。

D、先启动服务器端的程序，再启动两次客户端的程序，两个客户端启动后，便于服务端建立的链接。首先选择一个客户端“发送消息”，另一个客户端“接收消息”（无论哪个客户端“发送消息”还是“接收消息”都可以，但必须要进行选择），在发送客户端输入发送给另一个客户端的消息，这时接收客户端就会收到相应消息，接收端再进行发送。循环往复实现了两个客户端的通信（这种通信的形式是“你说一句，我说一句”）。

2）整体设计（类与类的关系）

A、客户端：

主类为client，包含内部类GetThread，与SendThread。GetThread为获取消息线程类，SendThread为发送消息线程类

B、服务端：

主类为server，主类即为线程实现类

3）局部设计（单个类的设计）

A、客户端：

(1).在主类的main线程中，实现与服务端的连接，有标准输入对象构造BufferedReader对象，由Socket的到输入流与输出流。进行选择，该客户端是“发送消息”还是“接收消息”，如果选择“发送消息”，则创建SendThread线程对象，并启动。如果选择“接收消息”，则创建GetThread线程对象，并启动。

**public** **class** client {

**static** Socket *socket*;

**static** BufferedReader *sin*;

**static** PrintWriter *os*;

**static** BufferedReader *is*;

**public** **static** **void** main(String args[]) {

**…………….}**

**static** **class** GetThread **implements** Runnable{

…………………}

**static** **class** SendThread **implements** Runnable{

……………………}

}

(2).内部类SendThread，实现了接口Runnable，通过键盘输入发送的消息，如果发送的消息不是“bye”则通过输出流发给服务端，线程阻塞在继续由键盘读入

**static** **class** SendThread **implements** Runnable{

**public** **void** run(){

System.*out*.println("请说话：");

send();

}

**public** **void** send() {

**…………..}**

**}**

(3).内部类GetThread，实现了接口Runable，通过输入流读取流中的内容，如果读取的内容不是”bye”,则打印显示，线程阻塞在继续从输入流中读取内容。

**static** **class** GetThread **implements** Runnable{

**private** String str;

**public** **void** run(){

recMsg();

}

**public** **void** recMsg() {

**…………………………}**

**}**

B、服务端：

有静态成员Socket数组，保存与各客户端建立连接的Socket对象，在主线程main 中，实现端口的监听，监听到客户端后，启动服务线程。该类还含有Socket对象，标识执行当前线程的socket[i]。

**public** **class** Server **extends** Thread{

**static** **int** *clientnum*=0; //静态成员变量，记录当前客户的个数

**static** **int** *i*=0;

**static** Socket *sockets*[]=**new** Socket[1000];

**public** **static** **void** main(String args[]) **throws** IOException {

**…………………………..}**

Socket socket=**null**; //保存与本线程相关的Socket对象

**public** Server(Socket socket){

**this**.socket=socket; //初始化socket变量

}

**public** **void** run() { //线程主体

……………………………….}

}

2.系统实现

（1）在客户端与服务端建立连接后，首先进行选择，该客户端是进行“发送”还是“接收”消息

String s;

System.*out*.println("接收1，发送2");

s=*sin*.readLine();

**if**(s.equals("1")){

GetThread getthread=**new** GetThread();

**new** Thread(getthread).start();

}

**else** {

SendThread sendthread=**new** SendThread();

**new** Thread(sendthread).start();

}

（2）发送消息线程的核心代码为：

**try** {

String readline;

readline=*sin*.readLine();

**while**(!readline.equals("bye")){

*os*.println(readline);

*os*.flush(); readline=*is*.readLine();

System.*out*.println(Thread.*currentThread*().getName()+":"+readline);

readline=*sin*.readLine();

}

*os*.close(); //关闭Socket输出流

*is*.close(); //关闭Socket输入流

*socket*.close(); //关闭Socket

} **catch** (IOException e) {

System.*out*.println("错");

e.printStackTrace();

}

（3）接收消息线程的核心代码：

**try** {

str = *is*.readLine();

**while**(!str.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+str);

str = *sin*.readLine();

*os*.println(str);

*os*.flush();

str=*is*.readLine();

}

*os*.close(); //关闭Socket输出流

*is*.close(); //关闭Socket输入流

*socket*.close(); //关闭Socket

} **catch** (IOException e) {

e.printStackTrace();

}

（4）服务端监听客户的连接

**while**(listening){ //循环监听

*sockets*[*i*]=serverSocket.accept();

**new** Server(*sockets*[*i*]).start();

*i*++;

*clientnum*++; //增加客户计数\*

}

服务端实现两个客户通信的中转线程：

**while**(*sockets*[0]==socket){

BufferedReader is1=**new** BufferedReader(**new**

InputStreamReader(*sockets*[0].getInputStream()));

line1=is1.readLine();

PrintStream os1=**new** PrintStream(*sockets*[1].getOutputStream());

PrintWriter os=**new** PrintWriter(os1);

**/\*os.write(line1);-----------------------当有这条语句时，回复的内容会重复两遍，即这条输出留上的内容在os.println(line1)时会输出两遍\*/**

**if**(!line1.equals("bye")){

/\*os.println(line1);**----------------------------不能写成os.println(is1.readLine());\*/**

os.flush();

}

**else**{

os.close(); //关闭Socket输出流

is1.close(); //关闭Socket输入流

socket.close(); //关闭Socket

}

}

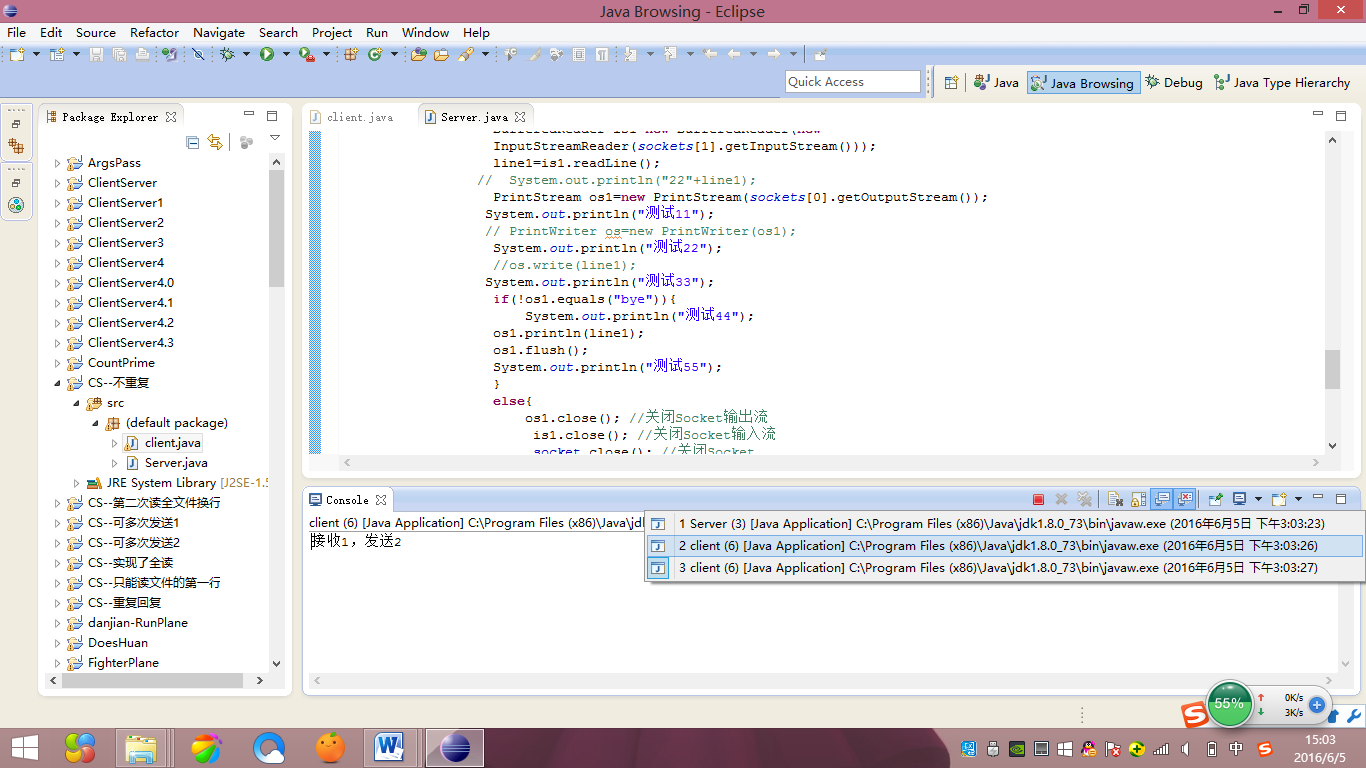
**while**(*sockets*[1]==socket){

同理

}

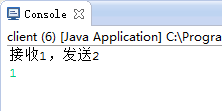
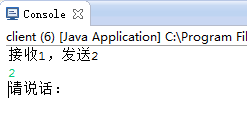
3、系统测试

测试一：

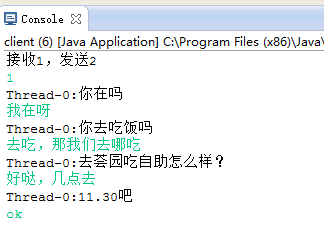
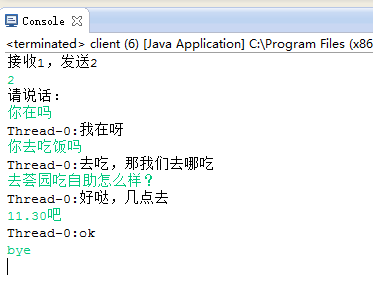
1）先启动服务端，再启动两个客户端

2） 一方客户端选择“发送”，一方选择“接收”

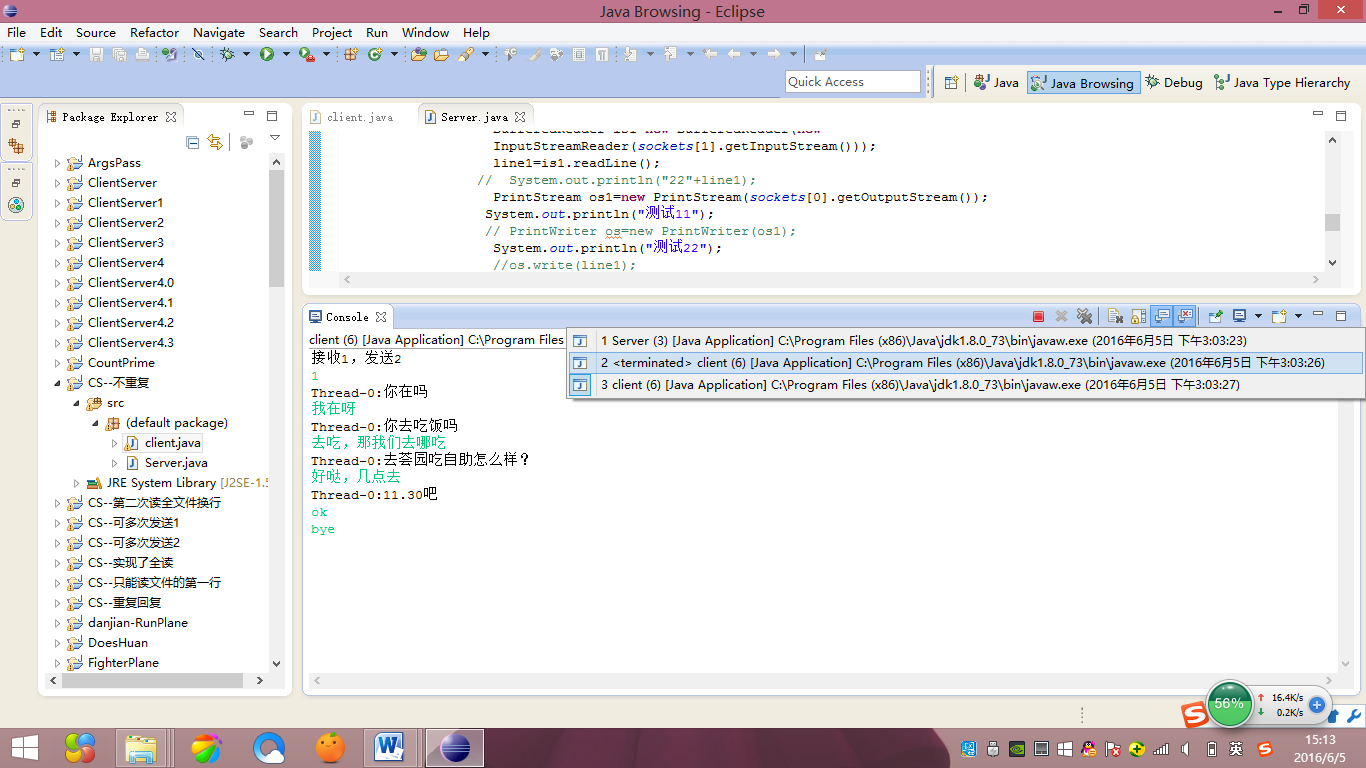
这里：先开启的客户端为“发送”，后开启的客户端为“接收”



3）在发送端输入发送的消息，接收端会收到相应消息，“你一句，我一句”进行两个客户端通信：



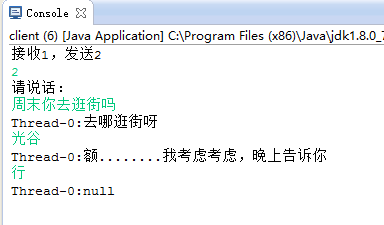
4）当任意客户端发送消息“bye”后，断开连接。

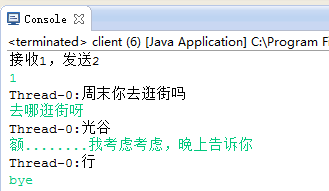


这里是发送客户端先输入的bye，则断开连接

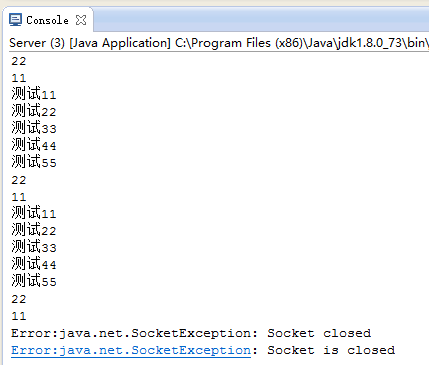
测试二：

1. 后开启的客户端为“发送”，先开启的为“接收”，实现通信





1. 接收端先发送“bye”，则断开连接



说明：

尽管客户端有发送消息与接收消息的线程，但在任何一个线程中都实现了接收与发送消息两种功能，对任一个客户端而言，初始时选择“发送”还是“接收 ”决定了它启动的是哪个线程，在这个线程中除第一次（发/接，只能是其一外）以后即可以发送消息，又可以接收。所以在输出显示时，每个客户端窗口，需要显示对方的线程名时，相对自己而言（主线程）都是“Thread-0”

**三、端到端的通信，实现并行通信模式（不再是你说一句，我说一句，一端的信息发送不受另一端的影响）**

**（一）不论发送还是接收线程实际仍然实现的既是发送又是接收**

1、系统设计

1）服务端的程序不变，客户端的主类不变，对发送消息线程内部类与接收消息线程内部类进行修改。

2）在发送线程中，除第一次为发送消息外，接下来需要选择是“是否接收 Yes”还是“继续发送 NO”

当选择“继续发送 NO”时，则由键盘读入；

当选择“是否接收 Yes”时，则从输入流中读取内容输出显示

3）在接收线程中，除第次为接收消息外，修改同发送线程

2、系统实现

1)在发送线程中修改如下;

**public** **void** send() {

**try** {

String readline;

readline=sin.readLine();

**while**(!readline.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+"我说-"+readline);

os.println(readline);

os.flush();

**System.*out*.println("是否接收： Yes,继续发送： No");**

**String ans;**

**ans=sin.readLine();**

if(ans.equals("Yes")){

readline=*is*.readLine(); System.*out*.println(Thread.*currentThread*().getNa me()+":"+"他说-"+readline);

}

**readline=sin.readLine();**

}

os.close(); //关闭Socket输出流

*is*.close(); //关闭Socket输入流

*socket*.close(); //关闭Socket

} **catch** (IOException e) {

System.*out*.println("错");

e.printStackTrace();

}

}

2)接收线程（同理）修改如下：

str = *is*.readLine();

**while**(!str.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+"他说-"+str);

System.*out*.println("是否回答： Yes,继续接收： No");

String ans;

ans=*sin*.readLine();

**if**(ans.equals("Yes")){

str = *sin*.readLine();//----------你好一

System.*out*.println(Thread.*currentThread*().getName()+":"+"我说-"+str);

*os*.println(str);

*os*.flush();

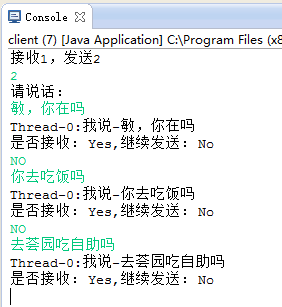
}

str=*is*.readLine()；

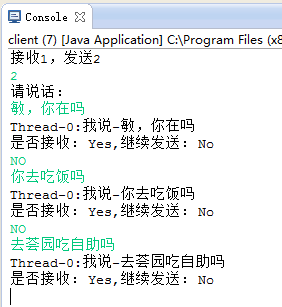
}

3.系统测试

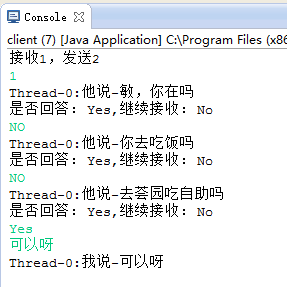
1）发送客户端可连续发送：



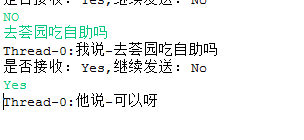
2） 接收客户端可连续接收：



1. 接收客户端选择恢复



1. 另一方选择接收



**（二）发送与接收线程实现各自独立的功能**

1、系统设计

1）服务端保持不变

2）客户端主类循环实现该客户端是接收信息，还是发送信息，进行选择后，必须执行完该线程，才能进入下一次的循环

3）客户端的发送线程进行修改，只能实现通过键盘输入发送的消息，通过输出流发送给中转站服务端

4）客户端的而接受线程进行修改，只能实现从输入流读取信息输出显示

2、系统实现

1）客户端循环选择：发送，接收

**while**(**true**){

System.*out*.println("接收1，发送2");

s=*sin*.readLine();

**if**(s.equals("1")){

GetThread getthread=**new** GetThread();

Thread get=**new** Thread(getthread);

get.start();

**get.join();**

}

**if**(s.equals("2")) {

SendThread sendthread=**new** SendThread();

Thread send=**new** Thread(sendthread);

send.start();

**send.join();**

}

}

2）接收线程修改

**public** **void** recMsg() {

**try** {

str = *is*.readLine();

**if**(!str.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+"他说-"+str);

}

**else**{

*os*.close(); //关闭Socket输出流

*is*.close(); //关闭Socket输入流

*socket*.close(); //关闭Socket\*/

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

3）发送线程修改

**public** **void** send() {

**try** {

String readlin;

readline=sin.readLine();

**if**(!readline.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+"我说-"+readline);

os.println(readline);

os.flush();

}

**else**{

os.close(); //关闭Socket输出流

*is*.close(); //关闭Socket输入流

*socket*.close(); //关闭Socket\*/

}

} **catch** (IOException e) {

System.*out*.println("错");

e.printStackTrace();

}

}

3、系统测试

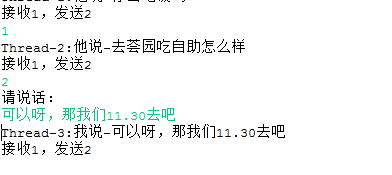
1）发送客户端连续发送消息，每个发送的消息为不同线程，可以通过输出的线程名可知



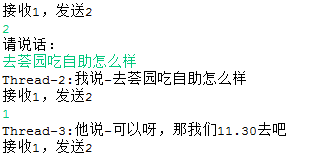
2）接收端可连续接收，每接收一次，启动一个线程



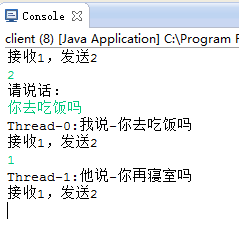
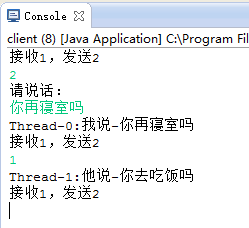
3）接收端可以选择回复



1. 另一方接收相应消息



1. 当两方首次都为发送消息时，显示如下：

两者之间不受影响

**（三）并行通信不需要进行选择“发送”和“接收”**

1.系统设计

1）服务端修改

判断当前的Socket属于哪一个客户端由while改为if

判断输入流的内容是否为“bye”由if改为while

**if(*sockets*[0]==socket){**

BufferedReader is1=**new** BufferedReader(**new**

InputStreamReader(*sockets*[0].getInputStream()));

line1=is1.readLine();

PrintStream os1=**new** PrintStream(*sockets*[1].getOutputStream());

PrintWriter os=**new** PrintWriter(os1);

**while(!line1.equals("bye")){**

os.println(line1);

os.flush();

line1=is1.readLine();

}

os.close(); //关闭Socket输出流

is1.close(); //关闭Socket输入流

socket.close(); //关闭Socket

}

**当有客户端连接成功时，向相应客户端发送“启动成功”的消息。**

2）客户端

发送消息与接收消息仍然为独立线程，循环阻塞读输入流与发送消息

主函数由以前的循环选择当前操作是“接收”还是“发送”修改为去掉循环，直接顺序调用接收线程与发送线程（接收在发送之前）调用。

2.系统实现

1）服务端接收客户端连接请求后，并成功连接，发送启动成功的消息：

*sockets*[*i*]=serverSocket.accept();

PrintStream os2=**new** PrintStream(*sockets*[*i*].getOutputStream());

os2.println("启动成功");

os2.flush();

**new** Server(*sockets*[*i*]).start();

2）客户端接收与发送线程核心代码

str = *is*.readLine();

**while**(!str.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+str);

str=*is*.readLine();

}

readline=sin.readLine();

**while**(!readline.equals("bye")){

System.*out*.println(Thread.*currentThread*().getName()+":"+"我说-"+readline);

os.println(readline);

os.flush();

readline=sin.readLine();

}

3）客户端主函数，先启动接收线程，再启动发送线程

**try**{

GetThread getthread=**new** GetThread();

Thread get=**new** Thread(getthread);

get.start();

}**catch**(Exception e){ }

SendThread sendthread=**new** SendThread();

Thread send=**new** Thread(sendthread);

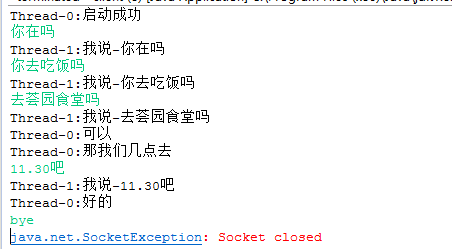
send.start();

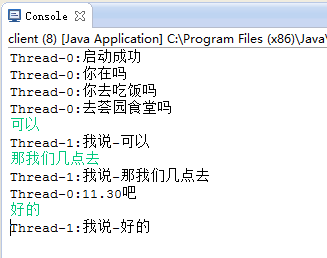
能够先执行接收消息的线程，读取输入流内容，是因为在客户与服务

端连接成功后，服务端发送给客户端“启动成功”的消息

3.系统测试

两客户端的通信





**四、实现端到端的文件传输**

1、系统设计

1）为实现客户端发送端读取src包中的“HelloWorldApplet.java”文件，将其转发到中转站服务端，服务端将文件字节流发送到客户接收端，客户接收端从套接字中读取字节流，写入到本地文件src包中的“Write.txt”中。

2）服务端程序保持不变

3）客户端主类增加静态成员字节数组，用于数据缓冲；增肌文件输入流FileInputStream，文件输出流FileOutputStream；主类的main线程的具体实现同两客户端并行通信一致。

4）客户端的发送线程实现对文件的读取，通过输出流发送到中转站服务端。

5）客户端的接收线程，实现从Socket的输入流中读取信息，并通过文件输出流，写到本地文件

2、系统实现

1）客户端主类增加成员

**static** FileInputStream *rf*;

**static** FileOutputStream *wf*;

**static** **int** *n*=512,*c*=0;

**static** **byte** *buffer*[]=**new** **byte**[*n*];

2）发送线程：

**static** **class** SendThread **implements** Runnable{

**public** **void** run(){

send();

}

**public** **void** send() {

**try** {

File fl=**new** File("src\\HelloWorldApplet.java");

InputStreamReader isr = **new** InputStreamReader(**new** FileInputStream(fl),"gbk");

BufferedReader br=**new** BufferedReader(isr);

String s=br.readLine();

**while**(s!=**null**){

System.*out*.println(s);

*os*.write(s);

/\*os.println();------如果写在这，发送过去的就是第一行的内容

os.flush();\*/

s=br.readLine();//----------连续读文件的所有内容

}

*os*.println();//----------读完文件中的所有内容后一起发送；

*os*.flush();

} **catch** (IOException e) {

e.printStackTrace();

}

}}

说明：

通过如上留的装配，才能实现对文件的多行读取以及空格和换行也能读取

3）接收线程：

**static** **class** GetThread **implements** Runnable{

**public** **void** run(){

rec();

}

**public** **void** rec() {

**try** {

String readline;

readline=*is*.readLine();

readline+="\r\n";//----------在第二次写入整个文件时换行了

*buffer*=readline.getBytes();

*wf*=**new** FileOutputStream("src\\write.txt",**true**);

*wf*.write(*buffer*);

} **catch** (IOException e) {

System.*out*.println("错");

e.printStackTrace();

}

}

}

说明：

为实现发送客户端把多个文件的内容发送客户接收端，且都写在同一个本地文件中，在每次写入时都是接在上一次写入之后，文件输出流表示如下：

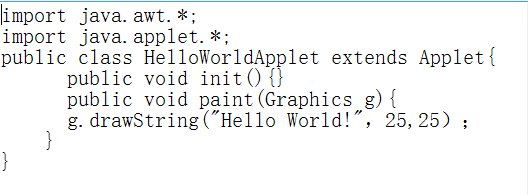
FileOutputStream("src\\write.txt",**true**);

为实现第二次的写入，为换行显示，则增加语句：

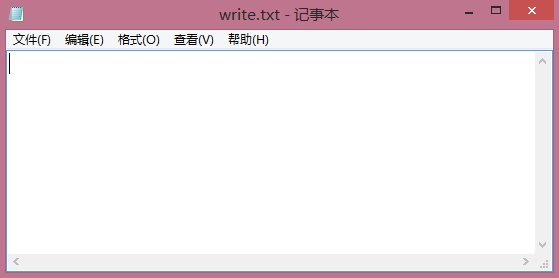
readline+="\r\n";

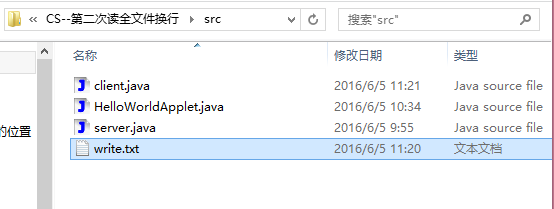
3、系统测试

1） 首先在当前工程的src包中，增加“HelloWorldApplet.java”文件，该文件的内容如下



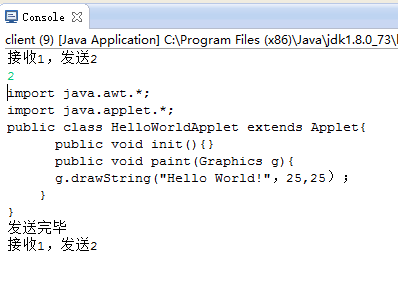
在src包中创建一个空的“write.txt”文件

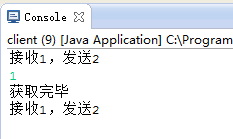




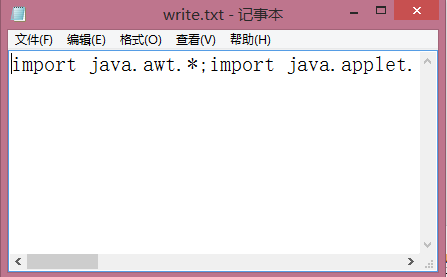
2）先启动服务端，再启动两次客户端

在发送客户端，选择发送，接收客户端选择接收。这时文件的一次传输完成





3）这时“write.txt”中有如下内容：

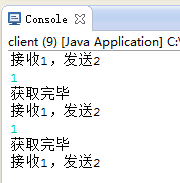


“HelloWorldApplet.java”中的内容全部写到“.txt”文件中的，但是在一行

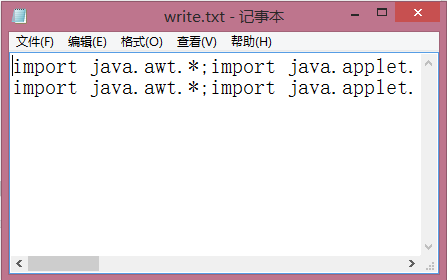
显示的。

4）继续执行，把“HelloWorldApplet.java”中的内容再写进“.txt”一遍





“write.txt”内容显示如下：



“HelloWorldApplet.java”中的内容仍全部写入，接在上一次的后面换行显示。

**五、课程设计总结**

1、实现客户端到客户端的双向通信，基于Socket套接字，服务端作为中转站。该过程是面向连接的，所以首先要实现客户端与服务器端建立连接，且在服务器端保存与每个客户端通信的Socket。

2、在中转站主要实现的是从一个客户端的输入流中读取发送过来的内容，并将该内容通过另一个客户端的输出流发送给对方

3、互相通信是通过循环阻碍实现的

4、实现两个客户并行通信，一方发送的消息不受另一方的影响，即要在客户端建立两个实现独立功能的线程，一个只负责“发送消息”的线程，一个只负责“接收消息”线程

5、当只能你说一句我说一句进行通信时，初始时，首先要选择哪一方进行首次发送消息。当两方通信互不受影响时，两者都可以在初始时发送消息。

6、进行文件传输时，要指明文件的路径，如果在当前工程的src包中，需要表示为：File fl=**new** File("src\\HelloWorldApplet.java");

7、对文件的读取要进行流的装配。

File fl=**new** File("src\\HelloWorldApplet.java"); InputStreamReader isr = **new** InputStreamReader(**new** FileInputStream(fl),"gbk");

BufferedReader br=**new** BufferedReader(isr);

如果采用如下形式，只能读取文件中的一行，遇到换行即结束读取。

*rf*=**new** FileInputStream("src\\HelloWorldApplet.java");

**while**((*c*=*rf*.read(*buffer*,0,*n*))!=-1){

String s=**new** String(*buffer*,0,*c*);

os.write(s);

}

8、因为在客户端中包含了发送线程类与接受线程类，两者都为内部类，所以要声明为静态的static。

9、服务端作为中转的两客户间通信，不同于客户与服务端的通信，服务端实现信息通过流的转发时，不能写成os.println(is1.readLine())；否则会阻塞在这里，不能将消息发送给另一方，而应该表示为：

line1=is1.readLine();

os.println(line1);

10、在服务端用数组保存Socket，需要规定数组大小，即客户端的数目是确定值，而现实生活中连接的客户数目不确定，因此服务端用于保存与客户端对应的Socket最好用集合类ArrayList，该类用于存储数量可变的对象集合。

在服务端用集合类ArrayList保存Socket实现如下

**//static List<Socket> sockets=new ArrayList<Socket>();//jdk1.5版本以后才可以实现**

**static** List *sockets*=**new** ArrayList();

**static** Socket *s*;

监听客户端连接

***s*=(Socket)*sockets*.get(*i*);//jdk1.5之前需要强转**

PrintStream os2=**new** PrintStream(*s*.getOutputStream());

os2.println("启动成功");

os2.flush();

**new** Server(*s*).start();

修改后程序仍然运行正确

11、当开启多个客户端时，需要根据数组下标选择谁和谁通信，这样较麻烦。但目前还没想到更好的实现方法。

12、为增加界面，做过如下尝试

用AWT图形用户界面实现：在客户端增加框架Frame，设置标题，大小，位置，长，宽。文本框Textfield，用来输入信息。文本域TextArea用来显示输入与接收的信息。

**private** **static** Frame *f*;

**private** **static** TextField *tfText*;

**private** **static** TextArea *taContent*;

*f*=**new** Frame("聊天");

*tfText* = **new** TextField();

*taContent* = **new** TextArea();

*f*.setSize(300,300);

*f*.setLocation(300,300);

*tfText*.addActionListener(**new** TFListener());

*f*.add(*tfText*,BorderLayout.*SOUTH*);

*f*.add(*taContent*,BorderLayout.*NORTH*);

*f*.addWindowListener(**new** WindowAdapter(){

**public** **void** windowClosing(WindowEvent e) {

System.*exit*(0);

}})

*f*.pack();

*f*.setVisible(**true**)

监听线程的实现

**static** **class** TFListener **implements** ActionListener{

**private** String str;

**public** **void** actionPerformed(ActionEvent e) {

**try**{

str = *tfText*.getText();

*dos*.writeUTF(str);

*taContent*.setText(str);

}**catch**(IOException ee){

}

}

}

运行结果只显示对话框，监听文本框的输入与接收消息的监听没有实现

