004.使用BIO实现网络通信

前面的课中，其实已经粗略地实现了一个基于BIO的一个网络通信实例，
但是，我们这章中的目标是实现一个基于命令行的能群聊的聊天室案例，
针对这个目标，前面的例子也就只能是一个简单的客户端与服务器端的通信例子，
很多细节的东西，需要我们进一步的完善！

进一步思考，我们要做一个聊天室，既要实现在公共频道说的话，所有的客户端都看得见，知道是哪个说的，
也要实现私聊的功能。应该怎么样进一步的完善，前面的简单例子！？

首先：暂时把服务器端发送给客户端的死信息去掉，需要在服务器端使用List来保存所有链接进来的客户端的Socket，

要用多线程来处理每个链接上来的客户端。服务器端要把收到的所有客户端发送来的数据“广播”给其他的客户端。

客户端也要升级，发送给服务器的信息，不能是死的，也要用多线程，主线程负责读取用户在键盘上输入的信息，

子线程负责读取和显示服务器端发送过来的数据。

再次：实现私聊功能，也就是说一个客户端可以将信息发送给另一个指定的客户端。

实际上，所有客户端只与服务器端链接，客户端直接并没有互联，

也就是说，当一个客户端发送信息后，

服务器端必须要判断信息到底是向所有用户发送，还是向指定用户发送。

这些功能就要求，每个客户端必有有自己的特征属性！

这里更具体的细节：

1、客户端发送来的信息必须要有标记，让服务器能够判断信息是公聊信息，还是私聊信息，是哪个用户发的。

2、如果是私聊信息，客户端信息的目标客户端是哪个，服务器必须知道，才能转发到指定的客户端。

解决办法：在客户端发送的信息上动手脚，在信息前加前后缀，这些特殊前后缀字符就是协议字符。

这些协议字符，有的标记公聊还是私聊，有的标记客户端身份.....

服务器端第二次升级：定义一个接口规定协议字符串，定义一个数据结构保存聊天室用户和对应Socket关联的输出流直接的映射关系，更新Server类添加异常捕获，更新

ServerThread类添加判断信息类型。

客户端第二次升级：更新Client类添加用户名的输入，也就是登录功能，更新获取键盘信息并发送的功能，要加上协议字符前后缀，更新ClientThread类关闭流。

最后完整的代码：

```
public class Server {
    //服务器监听的端口好
    public static final int SERVER_PORT = 40000;
    //把原理的list换成我们自己定义的新的数据结构，来保存链接进来的所有客户端，前面保存是Socket对象
    //现在我们保存的是代表客户端的用户名称和对应的Socket关联的输出流
    public static ChatRoomMap<String,PrintStream> clients = new ChatRoomMap<>();
    //绑定ip地址和端口的启动服务器的代码，封装起来init方法
    public void init(){
        try{
            //根据流程第一步，创建ServerSocket
            ServerSocket serverSocket = new ServerSocket(); //无参数，表示没有链接Socket
            serverSocket.bind(new InetSocketAddress("127.0.0.1",SERVER_PORT));
            //用一个循环来不断的接收客户端的链接
            while(true){
                //接收客户端的链接请求,获取链接进来的客户端的Socket
                Socket clientSocket = serverSocket.accept(); //此方法会阻塞，他会返回一个与连进来的客户端一对一对应的Socket
                //不能想前面一样，单线程去出来链接进来的客户端，效率太低，我们要用多线程来处理，每来一个客户端，就分配一条线程
                new Thread(new ServerThread(clientSocket)).start();
            }
        }catch (Exception e){
            System.out.println("服务器启动失败，可能是端口号："+SERVER_PORT+"被占用!");
        }
    }
    public static void main(String[] args){
        Server server = new Server();
        server.init();
    }
}


public class ServerThread implements Runnable {
    private Socket socket = null;
    private BufferedReader br = null;
    private PrintStream ps = null;
    public ServerThread(Socket socket) throws IOException {
        this.socket = socket;
    }
    @Override
    public void run() {
        try {
            //获取客户端对应的输入流
            br = new BufferedReader(new InputStreamReader(socket.getInputStream(), "GBK"));
            //在拿到客户端对应的输出流
            ps = new PrintStream(socket.getOutputStream(), true, "GBK");
            //先通过br读数据
            String lines = null;
            //用循环不断地读取客户端发送来的信息
```

```
            while ((lines = br.readLine()) != null) {
                //先读取客户端发送来的用户名
                //协议规定，客户端发送来的用户名信息，必须是USER_ROUND作为信息的前后缀
                if (lines.startsWith(ChatRoomProtocol.USER_ROUND) &&
lines.endsWith(ChatRoomProtocol.USER_ROUND)) {
                    //接受到的是用户名称
                    //拿到真正的用户名称
                    String userName = getRealMsg(lines);
                    //判断用户不能重复
                    if (Server.clients.map.containsKey(userName)) {
                        System.out.println("用户名重复了");
                        ps.println(ChatRoomProtocol.NAME_REP);
                    } else {
                        System.out.println("用户登录成功！");
                        ps.println(ChatRoomProtocol.LOGIN_SUCCESS);
                        Server.clients.put(userName,ps);
                    }
                }else if(lines.startsWith(ChatRoomProtocol.PRIVATEMSG_ROUND) &&
lines.endsWith(ChatRoomProtocol.PRIVATEMSG_ROUND)){
                    //客户端发送来的信息是私聊
                    //拿到真正的信息,信息里包含了目标用户和消息
                    String userAndMsg = getRealMsg(lines);
                    //上面的信息是用ChatRoomProtocol.SPLIT_SIGN来隔开的
                    String targetUser = userAndMsg.split(ChatRoomProtocol.SPLIT_SIGN)[0];
                    String privatemsg = userAndMsg.split(ChatRoomProtocol.SPLIT_SIGN)[1];

                    //服务器就可以转发给指定的用户了三

Server.clients.map.get(targetUser).println(Server.clients.getKeyByValue(ps) + " 私聊地说:" +
privatemsg);
                }else {
                    //最后一种可能就是公聊信息
                    //拿到真正的信息
                    String publicmsg = getRealMsg(lines);
                    //广播
                    for(PrintStream clintsPs : Server.clients.getValueSet()){
                        clintsPs.println(Server.clients.getKeyByValue(ps) + " 说： " +
publicmsg);
                    }
                }
            }
        } catch (Exception e) {
            //在上面的过程中，发生异常，标记服务器和客户端的Socket发送数据交换异常
            //这个客户端可能已经关闭了，这个客户端应该从我的clints集合里删除
            Server.clients.removeByValue(ps);
            System.out.println(Server.clients.map.size());
            //关闭io 网络
            try{
                if(br!=null){
                    br.close();
                }
                if(ps!=null){
                    ps.close();
                }
                if(socket!=null){
                    socket.close();
                }
            }catch (Exception ex){
                ex.printStackTrace();
            }
        }
    }
```

```
    //去除协议字符的方法
    private String getRealMsg(String lines) {
        return lines.substring(ChatRoomProtocol.PROTOCOL_LEN, lines.length() -
ChatRoomProtocol.PROTOCOL_LEN);
    }
}



//下面的字符要不得
public interface ChatRoomProtocol {
    //定义一个协议字符串的长度
    int PROTOCOL_LEN = 2;
    String PUBLICMSG_ROUND = "】⊿"; //公聊消息的前后缀
    String USER_ROUND = "$〇"; //用户名称的前后缀
    String LOGIN_SUCCESS = "1"; //登录成功的前后缀
    String NAME_REP = "-1"; //后面在客户端发送信息前，要求输入用户名，重复了返回这个标记
    String PRIVATEMSG_ROUND = "$&"; //私聊信息的前后缀
    String SPLIT_SIGN = "? ※"; //信息的分割标记
}



public class ChatRoomMap<K,V> {
    //本质上保存数据的是一个特殊的HashMap，我们定义过的，线程安全
    public Map<K,V> map = Collections.synchronizedMap(new HashMap<>());
    //可以根据V来删除指定的项目
    public synchronized void removeByValue(Object value){
        for(Object key : map.keySet()){
            if(map.get(key) == value){
                map.remove(key);
                break;
            }
        }
    }
    //获取所有的Value组合成的set集合
    public synchronized Set<V> getValueSet(){
        Set<V> res = new HashSet<>();
        //将map中的value添加到res集合里
        for(Object key : map.keySet()){
            res.add(map.get(key));
        }
        return res;
    }
    //根据value值来找到key
    public synchronized K getKeyByValue(V val){
       for(K key : map.keySet()){
           if(map.get(key) == val || map.get(key).equals(val)){
               return key;
           }
       }
       return null;
    }
    //实现添加数据到ChatRoomMap中,此数据结构规定value不能重复
    public synchronized V put(K key,V value){
        //遍历所有的value值判断有重复的没得
        for(V val : getValueSet()){
            if(val.equals(value) && val.hashCode() == value.hashCode()){
                throw new RuntimeException("Map实例中不允许有重复的value值！");
            }
        }
        return map.put(key,value);
    }
```

```
}


public class Client {
    private static final int SERVER_PORT = 40000;
    private Socket socket = null;
    private PrintStream ps = null;
    private BufferedReader inServer = null;
    private BufferedReader inKey = null;
    /**
     * 客户端链接服务器的功能,并且实现用户的登录
     */
    public void init(){
        try{
            //首先键盘的输入流初始化
            inKey = new BufferedReader(new InputStreamReader(System.in,"GBK"));
            //链接到服务器
            socket = new Socket("127.0.0.1",SERVER_PORT);
            //获取socket对应的输入输出流
            ps = new PrintStream(socket.getOutputStream(),true,"GBK");
            inServer = new BufferedReader(new
InputStreamReader(socket.getInputStream(),"GBK"))
            //用一个循环来进行服务器的登录
            String tip = "";
            while(true){
                //虽然我们还没见过到GUI，这里小小用一个gui里的弹出对话框
               String userName = JOptionPane.showInputDialog(tip + "输入用户名：");
                //就把用户输入的用户名发送给服务器
                 ps.println(ChatRoomProtocol.USER_ROUND + userName +
ChatRoomProtocol.USER_ROUND);
                //发送后，紧接着获取服务器的响应
                String res = inServer.readLine();
                //用户名重复了，返回-1
                if(res.equals(ChatRoomProtocol.NAME_REP)){
                    tip = "用户名重复，请重新";
                    continue;
                }
                if(res.equals(ChatRoomProtocol.LOGIN_SUCCESS)){
                    break;
                }
            }
        }catch (UnknownHostException e1){
            System.out.println("找不到服务器，请确认服务器是后启动！");
            closeRes();
            System.exit(1);
        }catch (IOException e2){
            System.out.println("网络异常，请确实网络是否链接！");
            closeRes();
            System.exit(1);
        }
        //启动线程，获取服务器的响应信息，在控制台显示
        new Thread(new ClientThread(inServer)).start();
    }

    /**
     * 客户端获取键盘上的信息并且发送给服务器的功能
     */
    private void readAndSend(){
        try{
            //通过循环不断地获取键盘上信息，包装发送
            String line = null;
```

```
            while((line=inKey.readLine())!=null){
                //对line的内容进行判断，发送的是私聊信息，还是公聊信息
                //规定：发送的信息如果有冒号，并且是以"//"开头，表示你发送的信息是私聊信息
                if(line.indexOf(":") > 0 && line.startsWith("//")){
                    line = line.substring(2);
                    ps.println(ChatRoomProtocol.PRIVATEMSG_ROUND +
                            line.split(":")[0] + ChatRoomProtocol.SPLIT_SIGN +
                            line.split(":")[1] + ChatRoomProtocol.PRIVATEMSG_ROUND);
                }else{
                    //就是公聊信息
                    ps.println(ChatRoomProtocol.PUBLICMSG_ROUND +
                            line + ChatRoomProtocol.PUBLICMSG_ROUND);
                }
            }
        }catch (IOException e){
            System.out.println("网络通信异常，请检查网络是否通畅！");
            closeRes();
            System.exit(1);
        }
    }

    /**
     * 对资源关闭的功能
     */
    private void closeRes(){
        try{
            if(inKey!=null) inKey.close();
            if(inServer!=null) inServer.close();
            if(ps!=null) ps.close();
            if(socket!=null) socket.close();
        }catch (Exception e){
            e.printStackTrace();
        }

    }
    public static void main(String[] args) throws IOException {
        Client client = new Client();
        client.init();
        client.readAndSend();
    }
}



public class ClientThread implements Runnable {
    private BufferedReader in = null;
    public ClientThread(BufferedReader in) {
        this.in = in;
    }
    @Override
    public void run() {
        try{
            //正式的实现获取和显示从服务器哪里发送来的信息
            String content = null;
            //用循环，不断地获取
            while((content=in.readLine())!=null){
                System.out.println(content);
            }
        }catch (Exception e){
            e.printStackTrace();
        }finally {
            try{
```

```
                if(in!=null) in.close();
            }catch (Exception e){
                e.printStackTrace();
            }
        }

    }
}
```