Introduction of the Program

Written by Can Zhu

Program function:

This program is a client/node/server chat system based on UDP, for unstable network communications between clients. And it is a unicasting transmission.

Program process description:

When clients inform server that they are on the line via a stable node, which has constant IP address and port, the server puts their information on a user list. Next, clients could ask server to deliver the online users' information to them. According to the user list, clients can choose one user to exchange messages through the stable node. To simulate the unstable network, two thread mechanisms are separately used to change clients' port number and server's port number.

Definition of protocols:

Related protocols need to be defined for network communication. The protocols used in the whole process could be regarded as Get and Post and their structures are as follows:

Protocol Get – clients or server receive data packets from stable node
Header: get
Sender: SENDER_ID

Protocol Post – clients or server send data packets to stable node
Header: post
Sender: SENDER_ID
Receiver: RECEIVER_ID
Content: messages

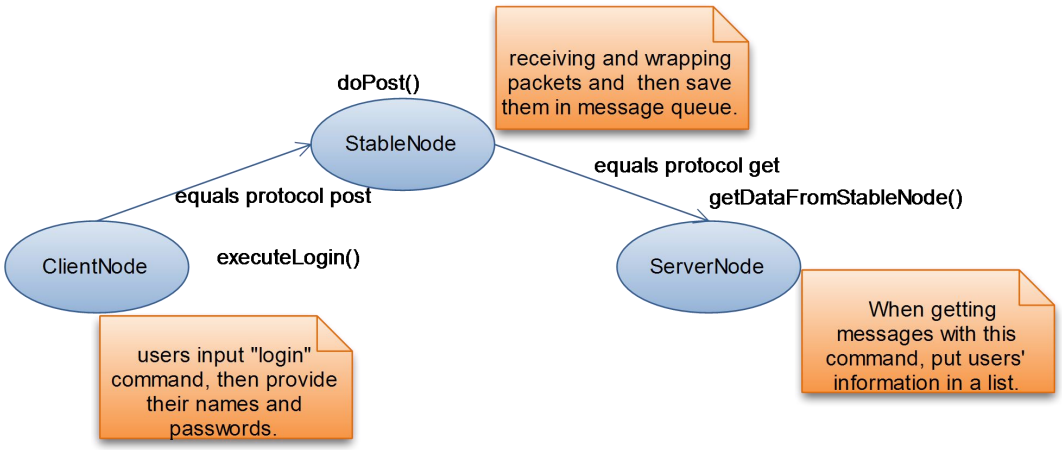
Monitor communication process:

Four cases are included in the communication. They are LOGIN, GET, POST, and USERS. I will further analyze details involved in the four steps.

1. Case: Login

If clients have "login" command, they will be asked to input their personal information (name and password). According to the information, a login protocol can be built via ProtocolBuilder Class and be sent to stable node. After stable node receives packets, it wraps the packets through Message Class. Then the packets are saved in the message queue to wait being sent to server. When server receives related message packets, it finds out these messages' type are login by parsing them.

Finally server will use a list to collect online users' information in these messages.

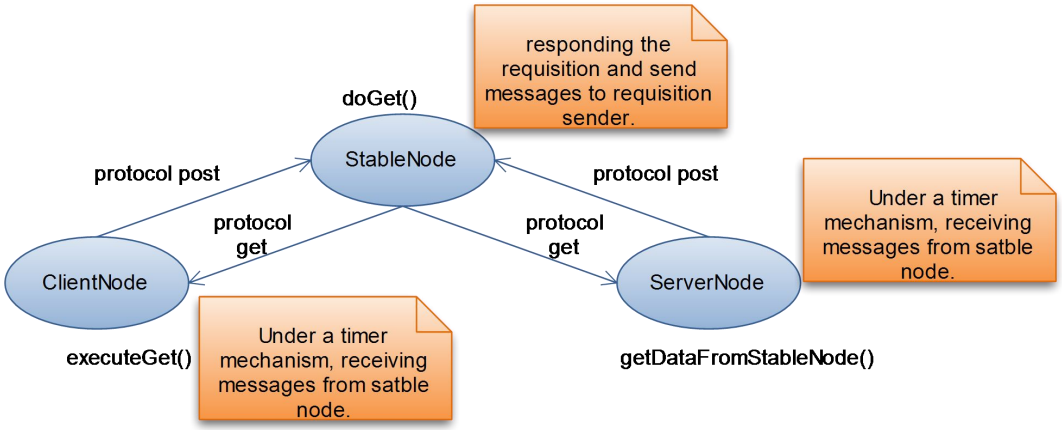



```
public static Message<?> wrap(String data) throws IOException{
    StringReader stringReader = new StringReader(data);
    BufferedReader reader = new BufferedReader(stringReader);
    String protocol = reader.readLine();

    if(protocol.equals(Constant.CMD_LOGIN)){
        String sender = reader.readLine();
        String receiver = reader.readLine();
        String name = reader.readLine();
        String password = reader.readLine();
        User user = new User();
        user.setName(name);
        user.setPassword(password);

        Message<User> msg = new Message<User>();
        msg.setType(MessageType.LOGIN);
        msg.setSender(sender);
        msg.setReceiver(receiver);
        msg.setContent(user);
        msg.setTimestamp(Calendar.getInstance().getTimeInMillis());
        return msg;
```

```
private void doLogin(Message<?> message){
    User newuser = (User)message.getContent();
    for(User user : this.users){
        if(user.getName().equals(newuser.getName())){
            System.out.println("User already loged in.");
            return;
        }
    }
    this.users.add(newuser);
}
```

2. Case: Get

When stable node starts, the clients and server automatically retrieve messages from it through a Timer mechanism. And the clients will be demanded to provide individual ID numbers, which be regarded as sender. But for server, the ID is constant. After that stable node compares sender with its messages' receiver, if they are matched, messages will be sent back.

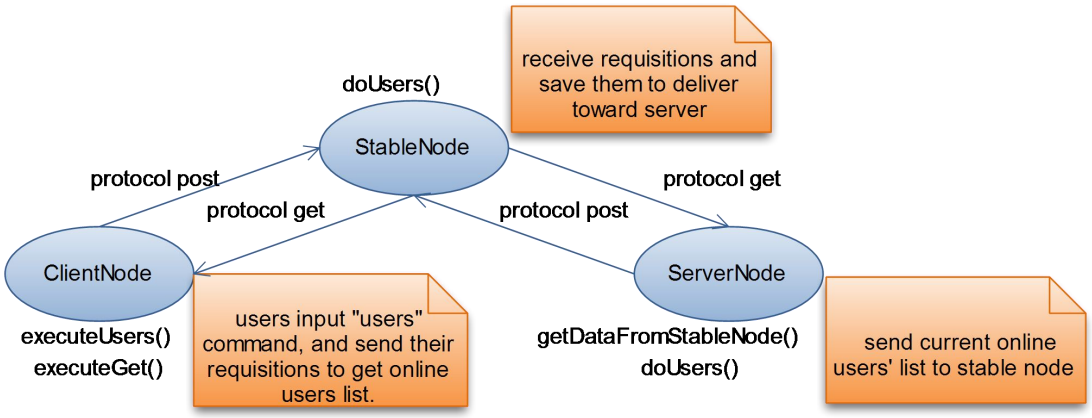



```
Timer t = new Timer();
t.scheduleAtFixedRate(
    new TimerTask(){
        @Override public void run() {
            try {
                excuteGet();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    },
    1000, 3000
);
```

```
private void doGet(Message<?> getCmd, DatagramSocket socket, InetAddress ip,
        int port) throws IOException {
    List<Message<?>> result = new ArrayList<Message<?>>();
    synchronized(this){
        List<Message<?>> remain = new ArrayList<Message<?>>();
        for(Message<?> msg : this.messages){
            System.out.println(msg.getType() + "\t" + msg.getReceiver() +
                    "\t" + getCmd.getSender());
            if(msg.getReceiver().equals(getCmd.getSender())){
                result.add(msg);
            }else{
                remain.add(msg);
            }
            this.messages = remain;
        }
    }
    this.sendBackMessages(result, socket, ip, port);
}
```

3. Case: users

In this case, clients send “users” to stable node as a command. By using ProtocolBuilder Class, a USER_LIST protocol is built and sent to stable node. Now it’s time to stable node for wrapping and save messages. The next time server gets these messages with type “USERS”, it will deliver the online users’ list to clients via the stable node.

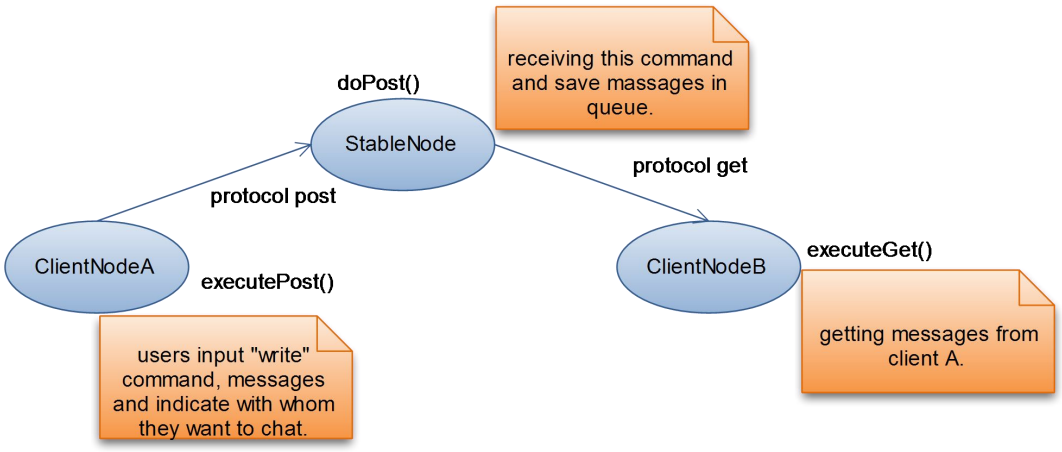



```
private void doUsers(Message<?> message) throws IOException{
    String result = ProtocolBuilder.buildUserList(users, message.getSender());
    byte[] buffer = result.getBytes();
    this.getCurrentSocket().send(new DatagramPacket(
        buffer, buffer.length,
        InetAddress.getByName(this.getStableNodeHost()),
        this.getStableNodePort()
    ));
}
```

```
public static String buildUserList(List<User> users, String receiver){
    StringBuffer sb = new StringBuffer();
    sb.append(Constant.CMD_USERS).append("\r\n");
    sb.append(Constant.SERVER_ID).append("\r\n");
    sb.append(receiver).append("\r\n");
    for(User user : users){
        sb.append(user.getName()).append("\r\n");
    }
    return sb.toString();
}
```

4. Case: post

When one client wants to chat with another, it needs to use command “write” following user’s name to inform stable node. The stable node retrieves messages’ sender and receiver through Message Class and save these massages in a queue. After that, when another client sends its client ID to get messages, stable node will compare it with messages’ receiver. If they are matched, massages can be delivered to this client.




```
private void excutePost() throws IOException {
    if(this.getClientId() == null){
        System.out.println("Please login first.");
        return;
    }

    String receiver = Utils.getLineInput("To Receiver : ");
    String message = Utils.getUserInput();
    message = ProtocolBuilder.buildPost(this.getClientId(), receiver, message);
    this.sendUDP(message);
}
```

```
  public static String getUserInput() throws IOException {
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in)
      String line = null;
      StringBuffer sb = new StringBuffer();
      while (true) {
          line = reader.readLine();
          if (line != null && !line.trim().equals("")) {
              if(line.toUpperCase().equals(Constant.CMD_SEND)){
                  return sb.toString();
              }else{
                  sb.append(line).append("\r\n");
              }
          }else{
              System.out.println("input nothing, try again.");
          }
      }
  }
```

Pictures of running program:

1. Stable node starts:

```
D:\>cd work\Server-2014-03-19\bin

D:\work\Server-2014-03-19\bin>java cn.edu.swu.StableNode
startup .....
```

2. Client node starts:

```
D:\work\Server-2014-03-19\bin>java cn.edu.swu.ClientNode
Please input startup parameters
Stable Node IP :
```

3. Server node starts:

```
D:\>cd work\Server-2014-03-19\bin

D:\work\Server-2014-03-19\bin>java cn.edu.swu.ServerNode
Please input startup parameters
Stable Node IP : 127.0.0.1
Stable Node Port : 1234
Refresh interval : 1000
```

4. Client login:

```
D:\work\Server-2014-03-19\bin>java cn.edu.swu.ClientNode
Please input startup parameters
Stable Node IP : 127.0.0.1
Stable Node Port : 1234
-> login
User name: bob
Password : 222
-> login
bob current is login.
->
```

5. “USERS” is used by client:

```
User name: ellen
Password : 111
-> users
->
----------------------------------------
          Online User List
----------------------------------------
  1. bob
  2. jane
  3. ellen
----------------------------------------
```

6. Sending messages to another client:

7. Response to the messages: