A Design of a Server-Controlled and Multicast-Based Text Chat System

Mohammed Salman

Contents

Introduction

In this application, a Multicast chat application is designed in a form of client-server architecture. The server accepting the request from a specific client and then forward a welcome message to all connected clients informing them a client has just joined the group. The same scenario for the regular messages where the server broadcast the message to that specific group only. The server checks first in which group the client is connected, then broadcast the message to the group. In this project no serialization is used, only sending the messages as bytes. The application is multithreaded. After the server accepting a request from the client run a thread to receive messages from that client where every client has a specific thread. To understand how the application work, the work will be explained in five steps: connecting to the Server, the Server accepting requests, the client sends the messages, forwarding messages to the Client, and receiving the messages at the Client.

Connecting to the Server

As mentioned in the introduction, there is no serialization in this project. The client is sending a specific message format in order for the server to be able to identify it as a request to join a group. This message contains the username of the client and the chatroom name along with the time of the request.

The Server accepting requests

On the server side, the server is running an infinite loop to accept the requests from the clients. When the request is received, the server accept the request and extract the needed information from the message. The server maintains a static variable to identify the number of the objects and using that number as a clientId. The server maintains two hashmap, one hashmap for storing clientId along with all the information of that client and another hashmap is for storing room name with the multicast IP to be used for this room. The server generates the multicast IP using a random function and then storing this IP along with the room name in the hashmap. Generating the multicast IP as a random value is a bad habit and it is only used here for simplicity and time saving.

After extracting the information and storing them, the server runs a thread to receive a message from the client where every client has its thread in the server. Before listening to the new messages from the clients, the server broadcast a special message with a special format to tell all the clients that a user with his name has just joined the group. The server is waiting for two seconds before broadcasting this message in order for the new client run all of its components.

The Client sends the message

The client will request for a chat using the same socket that was created to connect to the server. Before sending the message, the date attached to the message to be displayed later in all of the text area of all the clients.

Forwarding messages to the Client

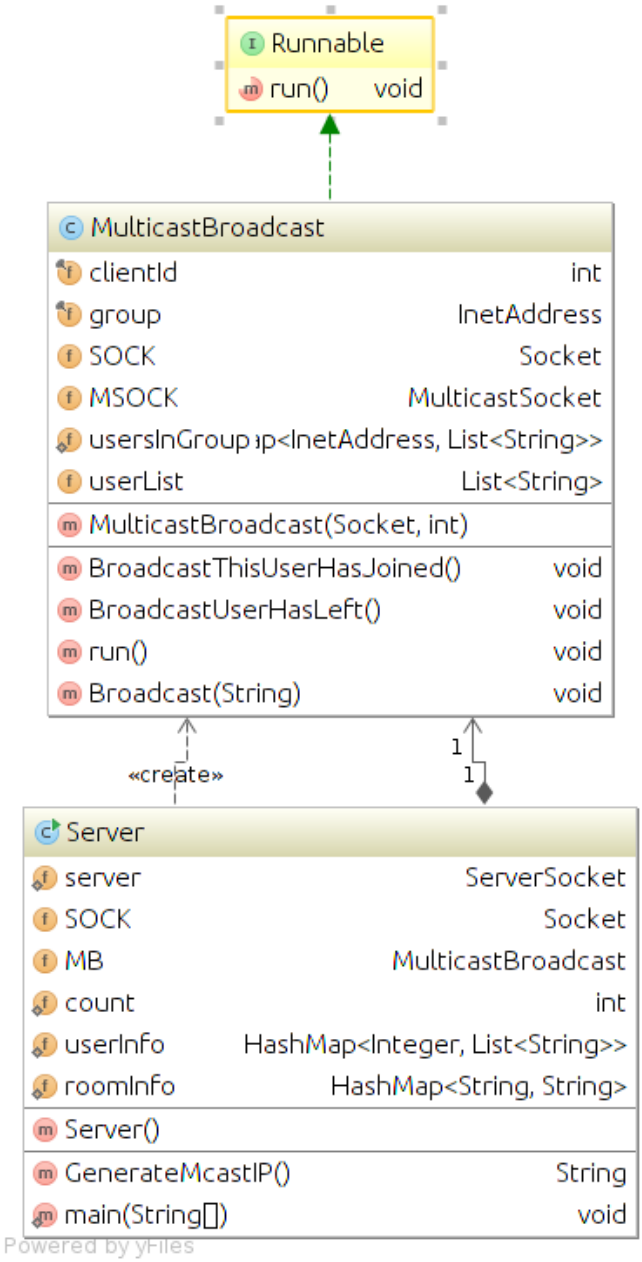
When the server received the message, the message are forwarded to the group. Three types of messages is forwarded, a message to inform the clients about the new user, a message to inform the clients about a specific user has just left the group, or a normal message that has to be received by all clients and displayed by their text areas.

Receiving the messages at the Client

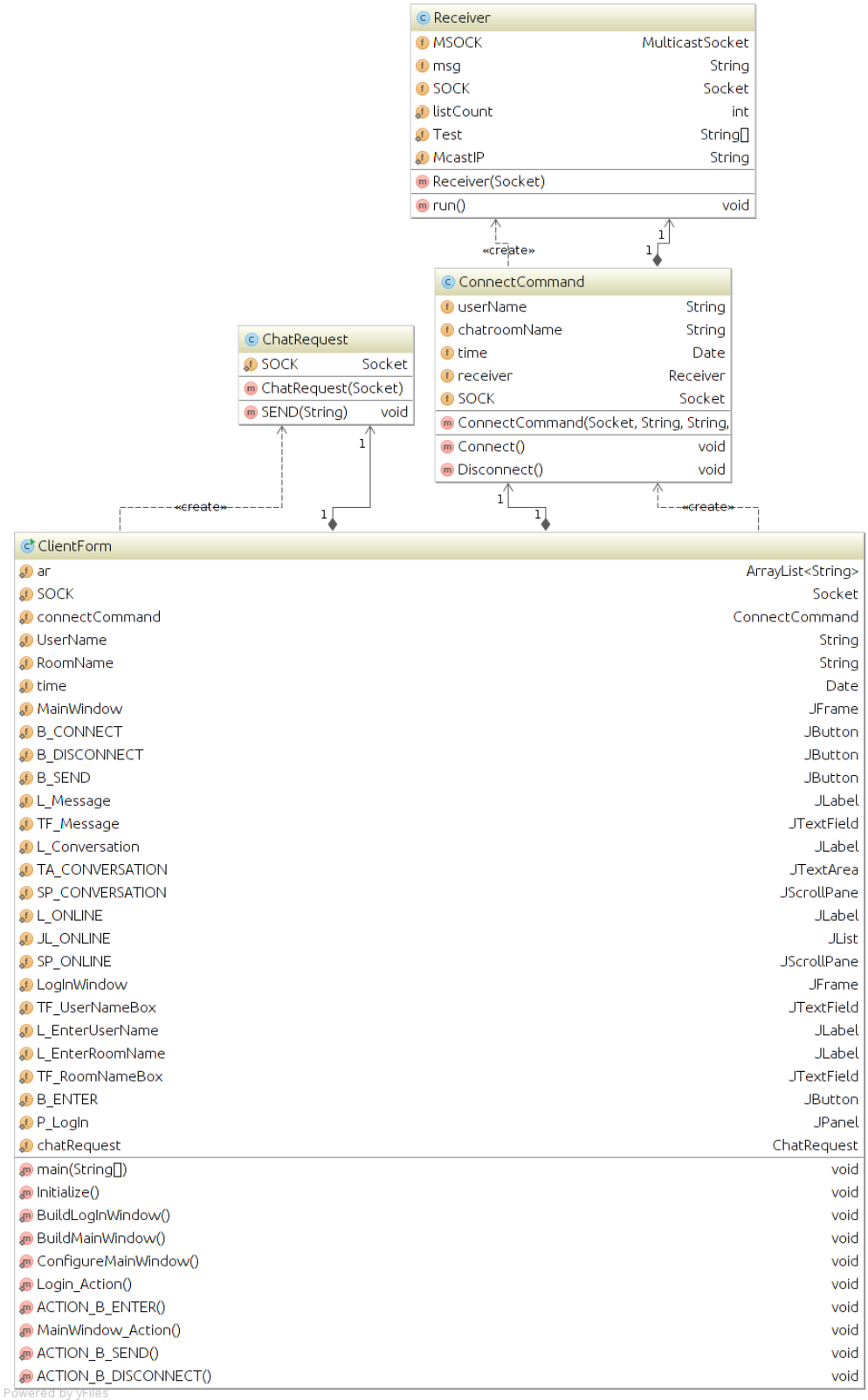
After connecting to the server, the client will start to listen to receive messages from the server through the multicast socket. As mentioned in the last paragraph, the client will be able to differentiate between the type of messages by identifying the message format. A message to inform the client about the new user, a message to inform the client if another client has just left the group, or regular message that need to be appended to the text area.

UML diagrams

Below is the UML diagram for the server side:



And below is the diagram for the client side:



Suggested future work and discussion

Sending messages as bytes between server and clients is a headache. Serialization is the key solution to the problem and there is a need to implement this technique. Another issue is the way the multicast IP's are generated. The multicast IP's are generated randomly and this is wrong in case the program generate the same IP for another room by chance. Although the chance of this is rare, it is safe to generate the IP just by incrementing starting from the IP 244.0.0.1. Finally, the application is tested on local machine and remotely with two different platform Windows and Ubuntu and everything went well.

References

Java - Sockets - Coding a Client-Server CHAT Room. (2012, March 12). Retrieved February 01, 2016, from https://www.youtube.com/watch?v=Uo5DY546rKY