University College Dublin

Course #COMP30220: Distributed Systems

Final Project: Multiple Chatrooms

December 5, 2019

Member	*Student ID*
Joseph Spielman	19201691
Jinyang Li	19201844

Contents

1 Introduction

In this final project, we aim to create a system that will allow clients to access multiple chat rooms which can be also called "channels". Channels will have some kind of authentication service in place to allow "all" or "specific" clients into the channel.
Moreover, global independent services called bots will be allowed to enter any of the channels. They can listen to all the channels and if they find specific keywords, it would respond as intended. For example, a remind bot will listen for the key word like "!Remind" and then send the date message to the specific channel which sent the command.

2 Background

Distributed system is a software system based on network. Because of the characteristics of software, distributed system has a high degree of cohesion and transparency. Therefore, the difference between the network and the distributed system lies more in the high-level software (especially the operating system), rather than the hardware. Cohesion refers to the highly autonomous distributed nodes of each database with local database management system. Transparency means that each database distribution node is transparent to the user's application, and it can't be seen whether it is local or remote. In the distributed database system, users do not feel that the data is distributed, that is, users do not need to know whether the relationship is divided, whether there is a copy, which site the data is stored in, and which site the transaction is executed.
For a distributed chatting room based on network, one classic way to realize it is to use basic web services. User sends HTTP request through client's browser, and then get the page of chatting room to chat online. Web server provides chat service for clients and manages chat rooms. Also, there is another classic technology which is to realize distributed chatting room using Sockets. The connection between the server and the client is established through socket, and then they can communicate. First, ServerSocket will listen for a port on the server side. When it finds that the client has a socket to try to connect to it, it will accept the connection request of the socket and establish a corresponding socket on the server side to communicate with it.
Apart from basic web services and Sockets, nowadays there are also many convenient specific technologies that can realize a distributed chatting room, such as Remote Method Invocation (RMI), RESTful web services, Zookeeper, Netty and so on. While realizing the basic chatting function of the distributed chatting room, these technologies can achieve other functions like granting authority for the clients, automatic replying messages and so on, in which our aims of the project are contained.

3 Analysis and Design

3.1 Analysis of the system

Mainly, the system we suppose to create contains three functions:
(1) Authentication to grant the entry of clients
(2) Fundamental chatting function

(3) Bots that reply automatically when receiving specific commands

So considering the main functions we aim to realize, we suppose to use RESTful Services to generate the distributed clients and servers. , and use NoSQL (Mongodb) to store the settings and messages of channels.

3.2 Design of the structure

Like the analysis above, we can get the overall system structure as following:

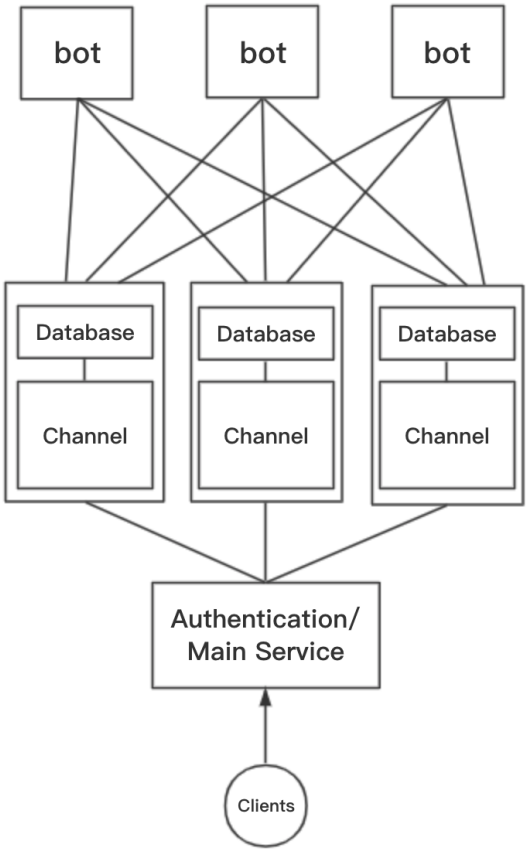

Figure 1: System Structure of Multiple Chatrooms

4 Implementation

In this section, we'll have a specific introduction of the functions and URL address supplied by RESTful Web Service of each part in the distributed system, which contains **Authentication Service, three bots services, Channel and Client**.

4.1 Authentication Service

API Function Name	Address	Method
channelList	http://localhost:8080/channel	PUT
getChannelInformation	http://localhost:8080/channel/{channelName}	GET
channelList	http://localhost:8080/channels	GET
botList	http://localhost:8080/bot	PUT
getBotInformation	http://localhost:8080/bot/{botName}	GET
botList	http://localhost:8080/botslist	GET
botsInformation	http://localhost:8080/botsInformation	GET

Table 1: RESTful APIs of Authentication Service

The authentication service is set to verify the status that the channels or bots services are working normally, and acquiring information about some specific channel or bot. In this part we created the RESTful APIs above, which have functions as following:

(1)channelList (for putting) : Allow **channel** to transmit its own information to this API.
(2)getChannelInformation : If it receive a message containing an available channel's name, it would return the information of that channel.
(3)channelList (for getting) : Return all names of available channels.
(4)botList (for putting) : Allow **bots** to transmit their own information to this API.
(5)getBotInformation : If it receive a message containing an available bot's name, it would return the information of that bot.
(6)botList (for getting) : Return all names of available bots as a string.
(7)botsInformation : Return the lists of available bots as an array.

Apart from them, the authentication service implements another thread to act as **HeartBeatChecker** when running, which would verify the availability of all channels and bots every 10 seconds. Channels and bots that are not valid would be removed by the authentication service.

4.2 Bots Services

Three bots services (Alphabetizer, AlphaOmegaQuoter and Timer) have been set in the project. And via this structure[1], new bots realizing more functions could be added in the future.
All the bots have the same communicating mechanism with channel, which is also indirectly communicating with client. In this process, one "BotCommand" message would be sent to a API provided by channel service, and then channel service would acquire bot's name in that command message, then sending the "BotCommand" message to the specific bot service.[2] Moreover, there exists an "AuthServerChecker" function in each bot service. It is to verify that the bot service is valid and still exists in the list of bots of authentication service. This part is realized as a way of heartbeat checking by a new thread. [3]

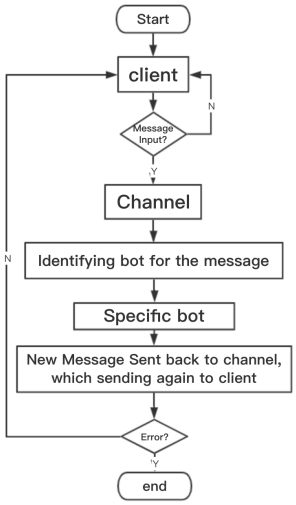

Figure 2: Communicating mechanism between bots and client

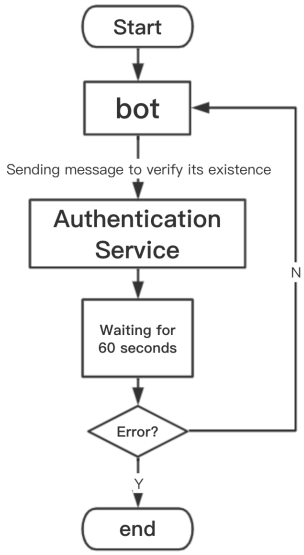

Figure 3: Heartbeat checking between bots and authentication service

4.2.1 Alphabetizer

API Function Name	Address	Method
orderWords	http://localhost:8090/run	PUT

Table 2: RESTful API of Alphabetizer Bot Service

Alphabetizer Bot's job is to sort the message generated by client in ascending order, and the function "orderWords" is the core part to realize it. For example, when it get a message as "1 3 2 4", it would sort the message and return a new one as "1 2 3 4". If if get a message "a c d b" then a new message as "a b c d" would be returned.

4.2.2 AlphaOmegaQuoter

API Function Name	Address	Method
getQuote	http://localhost:8089/run	PUT

Table 3: RESTful API of AlphaOmegaQuoter Bot Service

The bot AlphaOmegaQuoter does not need a text message but just the message calling it. When this bot is called, it would randomly select a quote from a preset list of quotes and return that quote. For example, it would randomly send back a quote like "Someone come help me find my contact lens? And maybe restart my heart...?" or "Do they make breath mints for dogs? They should..." when the bot get the same message "!AlphaOmegaQuoter" from client.

4.2.3 Timer

API Function Name	Address	Method
startTimer	http://localhost:8088/run	PUT

Table 4: RESTful API of Timer Bot Service

Obviously, Timer bot works to timing for client. When the bot get a message commanding it to timing, it would start a new thread and start timing, then sending the notification back to client after timing is done. For example, when a client sends message "!Timer 60 Second" or "!Timer 60 Seconds", the timer would start timing.

4.3 Channel&Database

API Function Name	Address	Method
requiresPin	http://localhost:8084/authenticate/{user}	GET
authenticateUser	http://localhost:8084/authenticate/{user}	PUT
banUser	http://localhost:8084/ban	PUT
unbanUser	http://localhost:8084/ban/{issuer}/{subject}	DELETE
showBans	http://localhost:8084/ban	GET
giveAdmin	http://localhost:8084/admin	PUT
takeAdmin	http://localhost:8084/{issuer}/{subject}	DELETE
showAdmins	http://localhost:8084/admin	GET
addBot	http://localhost:8084/bot	PUT
removeBot	http://localhost:8084/bot/{issuer}/{subject}	DELETE
listBots	http://localhost:8084/bots	GET
sendMessage	http://localhost:8084/message	PUT
loginMessages	http://localhost:8084/message	GET
getMessages	http://localhost:8084/message/currentMessageId	GET
commandBot	http://localhost:8084/commandBot/botName	PUT

Table 5: RESTful APIs of Channel Service

Channel service works as an important broker in the system. Its job contains providing space for multiple clients to chat, storing chatting messages data, transmitting messages between

clients and bots, securing itself using a password(PIN) and managing the users. Most of the functions are presented as the RESTful APIs above, which are stated in detail as following:

(1)requiresPin : It's a boolean function. When the function receives a username, it would determine whether this user is authorized. If this user is banned or it inputting a wrong PIN, the user would not be authorized.
(2)authenticateUser : It works to verify the PIN inputted by user. If the PIN is right, channel service would authorize the user, or it would throw an InvalidPinException.
(3)banUser : Used by admin user. When this function receives a command message to ban a user from admin user, it would add that user to bannedUser list.
(4)unbanUser : It's the inversion of (3), which would delete the selected user from bannedUser list when it receives a command message to unban a user.
(5)showBans : Show all the names of banned users.
(6)giveAdmin : Used by admin user. Admin user could send a command to channel and add another user as an admin.
(7)takeAdmin : Used by admin user, which is the inversion of (6). Admin user could send a command to channel and take out the authorization of other admin users.
(8)showAdmins : Show all the names of admin users.
(9)addBot : If channel does not add all available bots in at initialization stage, admin user could send specific message to this API to add a new bot(only for preset bots) in.
(10)removeBot : It's the inversion of (9) to remove a bot from current channel.
(11)listBots : List all the names of bots which exist in current channel.
(12)sendMessage : It collects the messages received by channel and store them into NOSQL, MongoDB we used.
(13)loginMessages : It provides the list of latest 20 messages.
(14)getMessages : Provide an API for user to keep updating other users' messages.
(15)commandBot : Broker between clients and bots. When it receives a command from some user to call one bot. It would acquire the bot's name from user's message and send the message to that specific bot.

Besides, to start a channel service, we need to input some parameters using inputs from console at the initialization stage, which is realized by "serverConfiguration" function in Channel service. It contains:
① Channel's name
② Channel's address (e.g. localhost:8084)
③ Channel's description
④ Choosing the bots to add in the channel. (It requires that bots services are online)
⑤ Channel's PIN. We can choose not to set the PIN.
⑥ Name of admin user
After this process, a channel can be finished and starting to run.
Besides, a new thread is used to check Authentication sever to see if the channel is on its registry. If the channel is not found, it will add itself back.

4.4 Client

Client basically interacts with channel service, and realize some functions via calling the RESTful API of channel. Apart from chatting with other clients in the same channel, clients could call the bots services and do some manipulations to the chatroom using specific

command texts. The following table lists the functions of clients in detail.

Command	Description	Method in program
N/A	Chatting with other users of same channel.	MessageUpdater
/add_bot [botname]	Add an available new bot to channel, used by admin.	addBot
/remove_bot [botname]	Remove a bot from channel, used by admin.	removeBot
/show_bots	Show all the bots in channel.	showBots
/ban [username]	Ban a user in channel, used by admin.	banUser
/unban [username]	Unban a user in channel, used by admin.	unbanUser
/show_bans	Show the names of banned users.	showBans
/add_admin [username]	Authorize one user with admin power, used by admin.	addAdmin
/remove_admin [username]	Remove the admin power of another admin user, used by admin.	removeAdmin
/show_admins	Show all the admins in channel.	showAdmins
/join	ReSelect a channel and join in.	joinChannel
/quit	Shut down Client.	main
!Alphabetizer [text1] [text2] ...	Call the Alphabetizer bot and sort text messages in ascending manner.	botCommand
!AlphaOmegaQuoter	Call the AlphaOmegaQuoter bot and get a random quote.	botCommand
!Timer [duraction] [Units]	Call the Timer bot and start a timing.	botCommand

Table 6: Special Commands of Client

5 Testing

Test plan – how the program/system was verified. Put the actual test results in the Appendix. This section is useful if your project is more on the software engineering side than research focused.

6 Results

7 Conclusion, Evaluation and Further Work

What have you achieved? Give a critical appraisal (evaluation) of your own work - how could the work be taken further (perhaps by another student next year)?