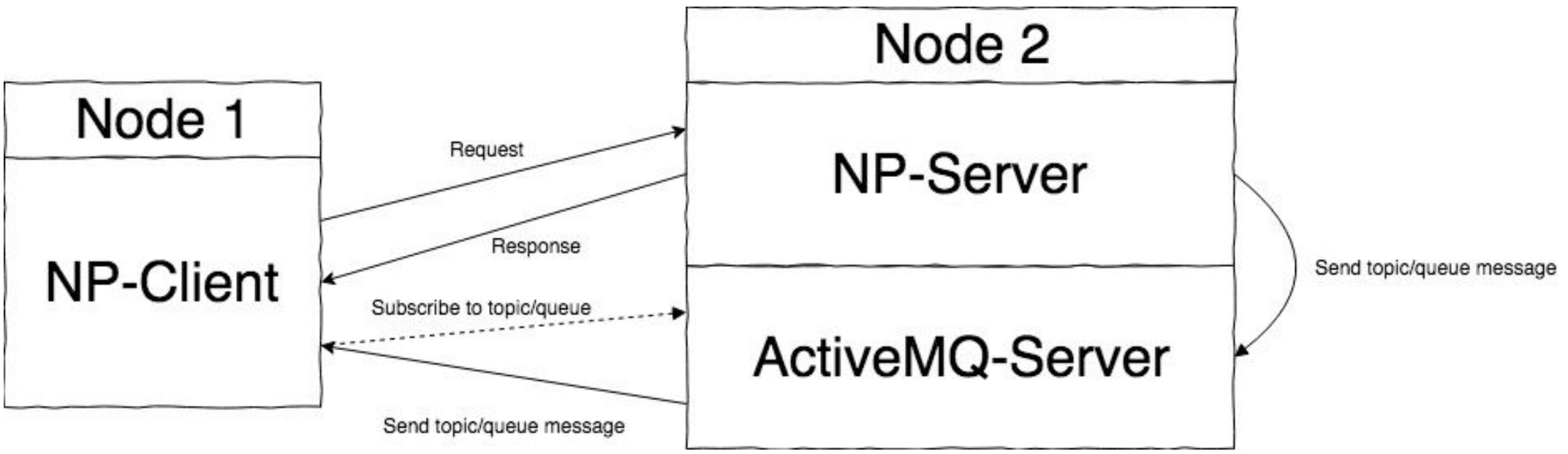
Introduction of Network Programming 2018 Autumn
Homework 4 - Message Broker

Description

In this assignment, you are going to extend your program in homework 2 and 3 in order to implement a Messenger-like function via message broker.

Example Architecture



NP-Client and NP-Server serve as project 2 & 3. Additionally, there are 6 commands added this time.

This is just an example architecture for your reference. If you have a better design, then you don't have to follow this architecture.

Requirement

The server accept the following requests. For client side, like project 2 & 3, client program will help user to store their access token. Therefore, users only need to type their username.

Request Format	Description	Request Status	Client Output (usernames or groupnames are examples, please output the correct names)
login <*id*> <*password*> done	When user login, your client program should subscribe to all the related channels, such as user's personal channel and the groups' channels that user has joined. When request fail, your program should have the same output message as project 2 & 3.	Success	Success!
send <token> <friend> <message> done	Send message to your friend via message broker. ● <message> allow spaces The request will success only when passing the following rules: (A has the highest priority, E has the lowest)	Success	Success!
		Fail-A	Not login yet

	A. You cannot send message with an invalid token B. Correct request format C. You cannot send to an user who doesn't exist. D. You can only send to your friend. E. You can only send to online user, which means that the user already login, but not logout yet.	Fail-B	Usage: send <user> <friend> <message>
		Fail-C	No such user exist
		Fail-D	userA is not your friend
		Fail-E	userA is not online
create-group <token> <group> done	Create a group named <group> ● <group> is unique ● <group> not allowed spaces The request will success only when passing the following rules: (A has the highest priority, C has the lowest) A. You cannot send message with an invalid token B. Correct request format C. The group name must be unique.	Success	Success!
		Fail-A	Not login yet
		Fail-B	Usage: create-group <user> <group>
		Fail-C	EXAMPLE_GROUP already exist
list-group <token> done	List all the groups The request will success only when passing the following rules: (A has the highest priority, B has the lowest) A. You cannot send message with an invalid token B. Correct request format	Success	group_hello group_world
		Success (No groups created)	No groups
		Fail-A	Not login yet
		Fail-B	Usage: list-group <*user*>
list-joined <token> done	List the groups you already added The request will success only when passing the following rules: (A has the highest priority, B has the lowest) A. You cannot send message with an invalid token	Success	group_hello group_world
		Success (No groups created)	No groups
		Fail-A	Not login yet
		Fail-B	Usage: list-joined <*user*>

<table>
<tr><td></td><td>B. Correct request format</td><td></td><td></td></tr>
<tr><td rowspan="5">join-group <token> <group>

done</td><td rowspan="5">Become the member of the group

The request will success only when passing the following rules: (A has the highest priority, D has the lowest)
A. You cannot send message with an invalid token
B. Correct request format
C. You cannot join a non-exist group
D. You cannot join the same group multiple times</td><td>Success</td><td>Success!</td></tr>
<tr><td>Fail-A</td><td>Not login yet</td></tr>
<tr><td>Fail-B</td><td>Usage: join-group <user> <group></td></tr>
<tr><td>Fail-C</td><td>EXAMPLE_GROUP does not exist</td></tr>
<tr><td>Fail-D</td><td>Already a member of EXAMPLE_GROUP</td></tr>
<tr><td rowspan="5">send-group <token> <group> <message></td><td rowspan="5">Send message to group members, including yourself.
● <message> allow spaces

The request will success only when passing the following rules: (A has the highest priority, D has the lowest)
A. You cannot send message with an invalid token
B. Correct request format
C. You cannot send message to a non-exist group
D. You cannot send message to a group that you didn’t join</td><td>Success</td><td>{
“status”: 0,
“message”: “Success!”
}</td></tr>
<tr><td>Fail-A</td><td>Not login yet</td></tr>
<tr><td>Fail-B</td><td>Usage: send-group <user> <group> <message></td></tr>
<tr><td>Fail-C</td><td>No such group exist</td></tr>
<tr><td>Fail-D</td><td>You are not the member of EXAMPLE_GROUP</td></tr>
</table>

General

- All the commands in project 2 & 3 must work as before.
- You have to setup an ActiveMQ server to forward the messages.
- You may have to modify the **login** feature to integrate with the message broker. For the failing cases, output the same message as project 2 & 3.
- You need to implement the rest of the features, to handle all the successful request and failing request.
- Your client program must output the correct message according to different commands and different situations.
- You server should reply the correct message according to the **priorities**.
- Output format of receiving message from other users
  - When USER_A send message “HELLO WORLD” to USER_B, your client program should show the message with following format to USER_B.
    - <<<USER_A->USER_B: HELLO WORLD>>>
  - When USER_A send message “HELLO WORLD” to GROUP_A, your client program should show the message with following format with all the group members, including USER_A.
    - <<<USER_A->GROUP<GROUP_A>: HELLO WORLD>>>

Do the UNSUBSCRIBE for logout and delete_user!!!

Grade (100%)

- Receiving send message from other users - (20%)
- Receiving send-group message from groups - (20%)
- Every commands above except **login** 10% - (60%)

Demo

It’s your responsibility to prepare the demo environment. TA will not help you prepare the environment, including the ActiveMQ. You can either bring your laptop, desktop, or using remote control to run the server.

When it’s your turn to demo, we will ask you to download the version you uploaded to new-e3.

1. Launch server (ip and port may change during demo): ./server 0.0.0.0 8888
2. Run with testcase: ./client TARGET_IP TARGET_PORT < testcase > output

Please learn how to run the above commands in your shell, no matter on Windows, Mac, or Linux.

Submit

Please upload a zip file called “hw4_{$student_id}.zip” that includes your source code. Submission that don’t follow the rule will get 20% punishment on the grade.

Demo time will be announced before the deadline. You are not allowed to modify your code after demo. Please submit your code on time.

If you have any questions, please ask your questions on course forum(https://e3new.nctu.edu.tw/)

Reference

1. **JSON format (https://zh.wikipedia.org/wiki/JSON)**
2. **C socket (http://man7.org/linux/man-pages/man2/socket.2.html)**
3. **Python socket (https://docs.python.org/3/library/socket.html)**
4. **C JSON (https://github.com/json-c/json-c)**
5. **Python JSON (https://docs.python.org/3/library/json.html)**
6. **C++ ODB (https://www.codesynthesis.com/products/odb/)**
7. **Python peewee (http://docs.peewee-orm.com/en/latest/)**
8. **Activemq (http://activemq.apache.org/)**