Lab 3 - Ping Pong

Distributed System

11th of March 2024

—

Submitted by Groupe K

PHAN Manh Tung

ESCAT Gregory

MOSIG M1

INTRODUCTION

The ping pong application is a simple application where two separate processes repeatedly send to each other the message “PING” or “PONG”. This application was made in the context of a lab in the *Introduction to Distributed Systems* course of M1 MoSIG at UGA. The goal was to familiarise ourselves with using RabbitMQ. The project can be found in the following github page: https://github.com/Graigauri/DS_PingPong.

RUN THE PROGRAM

The first step is to execute the RabbitMQ server with the following command line in a terminal:

```
rabbitmq-server
```

Then, you can create a variable to make the process faster:

```
export CP=.:amqp-client-5.16.0.jar:slf4j-api-1.7.36.jar:slf4j-simple-1.7.36.jar
```

To use our chat application, you will need three terminals, two to run the two ping pong processes and one to start the ping pong process between them.

```
java -cp $CP CustomNodeStart 1 2
```

```
java -cp $CP CustomNodeStart 2 1
```

```
java -cp $CP CustomClient 1
```

STRUCTURE

The structure described in the provided code can be characterized as a Distributed Messaging System using a Work Queue Pattern. Let's break down the components of this structure:

Distributed System:
- The system involves multiple nodes (players) that operate independently, communicating with each other through message passing.

Messaging System:
- Nodes exchange messages to coordinate and perform tasks.

Work Queue Pattern:
- The nodes utilize a work queue (implemented with RabbitMQ) to send and receive messages.
- The work queue acts as a communication channel between the nodes.

In this pattern:

- Each node is a participant in the distributed system, acting as both a producer (sending messages) and a consumer (receiving messages).
- Messages are enqueued in the work queue, and nodes consume messages from the queue to perform work based on the content of the messages.
- The "START," "PING," and "PONG" messages represent different tasks or actions within the system.

This messaging pattern is particularly suitable for scenarios where independent nodes need to coordinate their activities without direct synchronous communication, making it a form of asynchronous communication in a distributed environment.

Node Class (CustomNodeStart.java)

Constants:
- START, PING, and PONG constants are defined to represent different message types.

Instance Variables:
- queue_name: Represents the queue name for the current node.
- id: Represents the unique identifier for the node.
- output_to: Represents the output queue to which the node sends messages.
- started: Indicates whether the node has already started the ping-pong process.

start Method:

- Initializes the node with an ID and output queue information.
- Establishes a connection to RabbitMQ and creates channels.
- Declares queues for the node and its output.
- Sets up a DeliverCallback to handle incoming messages.
- Consumes messages from the node's queue (queue_name).
- Processes different message types (START, PING, PONG) and sends corresponding responses.

Client Class (CustomClient.java)

Constants:

- START, PING, and PONG constants are defined for message types.

main Method:

- Takes a command-line argument to determine the target queue (queue_name) for sending the "START" message.
- Establishes a connection to RabbitMQ and creates a channel.
- Declares the target queue (queue_name) for sending the "START" message.
- Publishes a "START" message to the target queue.

Overview of the Execution

- Nodes represent players in a ping-pong game, and each node has a unique identifier (id).
- Nodes communicate via RabbitMQ queues (queue_name) and exchange messages of different types (START, PING, PONG).
- The start method in the Node class is responsible for setting up the node, consuming messages, and responding accordingly.
- The main method in the Client class initiates the ping-pong process by sending a "START" message to a specific node's queue.

Message Flow

A client (Client program) sends a "START" message to a specific node's queue. The corresponding node (Node program) receives the "START" message, initiates the ping-pong process, and responds with "PING" or "PONG" messages based on the game logic. Nodes take turns sending "PING" and "PONG" messages to each other.