Project Walkthrough

May 2019

Introduction

This document is aimed to help you understand the purpose of some classes and how they operate and how the data flows between app components. We advice you first to read previous theses to understand more about the app development before moving on to this document, also read more about JMS and Netty for this document to be able to easily understand the flow of data.

1 System Design

In this project there are 4 main apps running, User,Tweet,List and DM app. Every app is responsible for handling a set of requests. Any incoming message will be have an attribute queue to be directed to. For example any incoming message designated to the List app will be directed to the queue "list.INQUEUE" and the response will go to "list.OUTQUEUE".

2 ActiveMQ

2.1 ActiveMQConfig

This class main objective is to create and start a connection to a queue. It takes queue name as an input in its constructor.

2.2 Producer

A producer takes an ActiveMQConfig object as an input to connect to the correct queue. Then when a producer sends a message it is very important to set the JMSCorrelationID before sending it, as correlationID is very crucial for consumers.

2.3 Consumer

A consumer is created usually in this project with an ActiveMQConfig for same reasons as the producer a correlationID. The correlationID is used as a message selector to filter messages back to the correct handler.

3 Netty

Netty is a client-server framework used to deal with network programming. When an HTTP request(message) is received, the a socket is opened and a handler is responsible for it-EduMsgServerHandler. To initialize a handler we use EduMesgServerInitializer Class, this adds some CORS headers and HTTP encoder and decoder to the handler.

3.1 EduMsgServerHandler

The channelRead0() method is executed when a new message has arrived, when a the message is done streaming and all sent to the server, the method writeResponse() is executed.

3.1.1 writeResponse

In here we create a JSON object of the incoming message first. Then we create a NettyNotifier object for the handler to notify it upon the completion of message processing, more on this later. Then the request body and the attribute queue is passed to method sendToActiveMQ(), queue is used to create a producer and body is the actual data send by the producer. The process is halted until a notifier receives a response, and sends it back to the client.

3.1.2 NettyNotifier

In here we create a consumer to the same queue as the producer previously with a message selector the handler correlationID. Then the notifier waits for a message to the queue to receive and send back to the handler. Then it closes all its open connections to save resources.

4 Main Apps

When a message is finally going to its destination queue, few procedures are made before actually manipulating the database. First, to receive a message, every main app have to create a consumer to fetch messages from its designated queue. For example, users app queue will be called "User.INQUEUE". Once the consumer is set, it will keep checking until it receives a message, after receiving a message the method handleMsg() from the main apps superclass is then called.

4.1 handleMsg

This class first checks if the data is already cached or not, if cached is the cache entry valid or not, if valid send it as a response. The superclass needs to know which subclass is it handling right now hence the String subClass attribute. Using the subClass input it knows to checks data in which cache. If cache is invalid or does not exists, then its goes and execute the given method. It checks the given method against a CommandMap-hashtable with all commands class with method names as their keys- and fetch the corresponding Command, if not such command exists an error is thrown, if it does then the command will be executed.

5 Commands

Every command in this project is a subclass from the class Command. Command has some attributes that is used by all its children.

- HashMap map: This what holds all the incoming data from the request.
- Connection dbConn: This is of type SQL.connection, this enables connection to the database.
- CallableStatement proc: To execute any stored procedure we first need a callable statement.
- Statement query: Some stored procedures needs to be called using a query instead of callable statement.
- ResultSet set: This holds any returned resultset from procedures.
- MyObjectMapper mapper: This helps creating json objects from strings.
- The rest are easily deduced.

5.1 Command

When a command finally gets to execute, first thing it connects to the database, then every command calls a certain stored procedure, it sets types of the stored procedure inputs and outputs and set the value of the inputs according to their arrangement in the stored procedure in the database. After that, the callable statement is executed, if there is a resultSet returned then, the set is looped on returning an array with objects depending on the command class it self. After it finished execution, a command then caches its data if needed. If it is a CRUD command then the cache entry is invalidated, if it is a get method the data is cached and set to valid. If any error occurred during any of these operations the appropriate error message is returned.

5.2 CommandsHelp

This class reduces work done by commands. This class is responsible for validating and invalidating cache entry, commands simply passes the cache entry, session ID and method name(and type if needed) and it will do the job. Also, it has all the error messages in the system, based on the queue, method and the throwable error it can identify what kind of error had occurred. Most importantly, the CommandsHelp class is the one responsible for sending back responses to the queue and attach the appropriate headers if an error happens.