JMS – Java Messaging Service, using JMS two different java applications can interact with each other

RabbitMQ is a message broker which consists of exchange and queue, so basically whatever the message send by the app1 which is producer will route it to the exchange and queue and sends to the consumers which are app2,3,4.

Exchange – when we want to apply some conditions and based on that condition we want to route the message to particular queue.

There are four types of exchange:-

1.Direct

2.Fan out

3.Headers

4.Topic

Without having the exchange we can have the queue and publish the messages.

To publish the message to the queue, we need to use the connection Factory class and connection in general interfaces inside the RabbitMQ client.

**Publisher:-**

<dependency>

<groupId>com.rabbitmq</groupId>

<artifactId>amqp-client</artifactId>

<version>5.6.0</version>

</dependency>

ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

Using the channel object we can publish the message to the queue.

channel.basicPublish(exchange, routingKey, props, body);

String message = “First message from RabbitMQ”;

channel.basicPublish(“”, “Queue-1”, null, message);

channel.close();

connection.close();

**Consumer:-**

ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

In order to get the message the RabbitMQ client provides one functional interface that is deliver callback and it has only one abstract method that is handle with the two parameters.

First is String and the second is type of delivery.

DeliverCallback deliverCallback = (consumerTag, delivery) -> {

String message = new String(delivery.getBody());

System.out.println(“Message received = “+message);

};

channel.basicConsume(queue, autoAck, deliverCallback, cancelCallback);

channel.basicConsume(“Queue-1”, true, deliverCallback, consumerTag ->{});

At the consumers end we cannot close the channel and connection because at any moment publisher can publish the message to the Queue and consumer should be in the running mode.

We can close the channel and connection in publisher but not in consumer.

In order to handle the multiple consumers queue will follow the Round-Robin fashion

For example if we have two consumers and four messages then first message will go to the first consumer and second will go to the second cosumer and third will go to the again first cosumer and fourth will go the second consumer again.

For creating the consumers Right click on the project -> export ->Runnable JAR

If we want to remove all the ready messages from the queue then we have to use purge in rabbit page or we can perform using cmd by taking the location of RabbitMQ bin and then passing the command of rabbitmqctl.bat purge\_queue Queue-1(Queue name)

Publishing Json message to the queue :-

We have to add the dependency

<dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20180813</version>

</dependency>

ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

JSONObject json = new JSONObject();

Json.put(“from\_date”, “04-05-2023”);

Json.put(“to\_date”,”31-12-2023”);

Json.put(“email”, “xyz@gmail.com”);

Json.put(“query”, “select \* from data”);

Channel.basicPublish(“”, “Queue-1”, null, json.toString().getBytes());

Direct Exchange matches the key that is coming with the message to the key that is provided at the time of the binding.

In direct exchange the routing key plays very very important role.

**DirectPublisher**:-

ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String message = “this is mobile”;

channel.basicPublish(“Direct-Exchange”, “mobile”, null, message.getBytes());

channel.close();

connection.close();

Consumer deals with the queue, he doesn’t deals with the exchange

Fanout Exchange:- In this there will be no key like direct exchange. So whatever message will come to the fanout exchanger will go to every queue.

If we provide null value to the routingkey of Fanout Exchange it will generate the exception, so we have to pass the “”.

Topic Exchange:- include regular expression kind of things in key and exchange values

\*- means exactly one word

#-means one or more than one word

Header Exchange:- In this type Message is having header which is having key value pair. In header exchange we need to provide the header not the routing key.

x-match = any -> any match of the values. Or condition

x-match = all -> must match all of the values. And condition

Publisher

ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

String message = “Message for Mobile and TV”;

Map<String, Object> headerMap = new HashMap<String, Object>();

headerMap.put(“item1”, “mobile”);

headerMap.put(“item2”, “television”);

BasicProperties br = new BasicProperties();

br = br.builder().headers(headersMap).build();

channel.basicPublish(“Headers-Exchange”, “”, br, message.getBytes());

channel.close();

connection.close();

**RabbitMQ with spring boot:-**

For using in boot application we need to add two properties in application.properties

1.spring.rabbitmq.host=localhost

2.spring.rabbitmq.port=5672

Convertandsend uses the simple message converter and simple message converter only supports string byte array and serializable objects.

If we don’t implement the serializable interface and we try to send the object of class person to the queue then it will throw the exception.