МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №3
«Межпроцессорное взаимодействие с помощью RabbitMQ»
по дисциплине «Сети ЭВМ и телекоммуникации компьютерных сетей»

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель _____ Мулюха В. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

1 Постановка задачи

Используя docker создать контейнеры, необходимые для реализации следующего функционала с использованием RabbitMQ, а также показать, как именно осуществляется передача в этих условиях:

Вариант 10. Организовать отправку сообщений от двух отправителей одному получателю, получатель читает сообщения значительно реже их отправки и происходит переполнение очереди. Задавать раличные стратегии переполнения очереди, показать в чем разница.

2 Ход работы

2.1 Код программы получателя

Класс ‘Recv‘ из пакета ‘dev.tishenko.consumer‘ реализует потребителя сообщений из очереди RabbitMQ с именем ‘hello‘. Он подключается к брокеру ‘rabbitmq‘, создаёт очередь с ограниченной длиной (‘x-max-length‘) и стратегией переполнения (‘x-overflow‘), параметры которых настраиваются через переменные окружения. Консьюмер асинхронно принимает сообщения и после каждой обработки (с задержкой, заданной через ‘CONSUMER_DELAY_MS‘) подтверждает получение сообщения. Благодаря настройке ‘basicQos(1)‘ сообщения обрабатываются по одному, что обеспечивает контроль нагрузки и предотвращает переполнение очереди необработанными сообщениями.

```
1  package dev.tishenko.consumer;
2
3  import com.rabbitmq.client.Channel;
4  import com.rabbitmq.client.Connection;
5  import com.rabbitmq.client.ConnectionFactory;
6  import com.rabbitmq.client.DeliverCallback;
7
8  import java.util.HashMap;
9  import java.util.Map;
10
11 public class Recv {
12     private final static String QUEUE_NAME = "hello";
13
14     public static void main(String[] argv) throws Exception {
15         String delayEnv = System.getenv("CONSUMER_DELAY_MS");
16         int delay = delayEnv != null ? Integer.parseInt(delayEnv) : 5000;
17
18         String maxLengthEnv = System.getenv("QUEUE_MAX_LENGTH");
19         int maxLength = maxLengthEnv != null ? Integer.parseInt(maxLengthEnv) : 5;
20
21         String overflowStrategy = System.getenv("QUEUE_OVERFLOW");
22         if (overflowStrategy == null) {
23             overflowStrategy = "drop-head"; // или "reject-publish"
24         }
25
26         ConnectionFactory factory = new ConnectionFactory();
27         factory.setHost("rabbitmq");
28
29         Connection connection = factory.newConnection();
30         Channel channel = connection.createChannel();
31
32         Map<String, Object> args = new HashMap<>();
33         args.put("x-max-length", maxLength);
34         args.put("x-overflow", overflowStrategy);
35
36         channel.queueDeclare(QUEUE_NAME, false, false, false, args);
37         channel.basicQos(1);
38
39         System.out.println(" [*] Waiting for messages...");
40
```

```
41          DeliverCallback deliverCallback = (consumerTag, delivery) -> {
42              String message = new String(delivery.getBody(), "UTF-8");
43              System.out.println(" [x] Received '" + message + "'");
44              try {
45                  Thread.sleep(delay);
46              } catch (InterruptedException e) {
47                  e.printStackTrace();
48              }
49              channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
50          };
51
52          channel.basicConsume(QUEUE_NAME, false, deliverCallback, consumerTag -> {
53          });
54      }
55  }
```

2.2 Код программы отправителя

Класс ‘Send‘ представляет собой простого отправителя сообщений для RabbitMQ. Он подключается к брокеру (по адресу контейнера ‘rabbitmq‘), использует очередь с именем ‘hello‘ (предполагая, что она уже существует), и в бесконечном цикле отправляет в неё сообщения с уникальным номером, указывая имя продюсера, полученное из переменной окружения ‘PRODUCER_NAME‘. Задержка между отправками настраивается через переменную ‘PRODUCER_DELAY_MS‘. После каждой отправки ожидается подтверждение от брокера, иначе выводится предупреждение.

```
1   package dev.tishenko.producer;
2
3   import com.rabbitmq.client.ConnectionFactory;
4   import com.rabbitmq.client.Connection;
5   import com.rabbitmq.client.Channel;
6
7   public class Send {
8       private static final String QUEUE_NAME = "hello";
9
10      public static void main(String[] args) throws Exception {
11          String name = System.getenv("PRODUCER_NAME");
12          name = name != null ? name : "";
13
14          String delayEnv = System.getenv("PRODUCER_DELAY_MS");
15          int delay = delayEnv != null ? Integer.parseInt(delayEnv) : 1000;
16
17          ConnectionFactory factory = new ConnectionFactory();
18          factory.setHost("rabbitmq");
19
20          try (Connection connection = factory.newConnection();
21                  Channel channel = connection.createChannel()) {
22              channel.queueDeclarePassive(QUEUE_NAME);
23              channel.confirmSelect();
24
25              int count = 0;
26              while (true) {
27                  String message = "Message from " + name + " #" + count++;
28                  channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
```

```
29                 if (!channel.waitForConfirms()) {
30                     System.out.println("␣[!]␣Message␣was␣NOT␣confirmed!");
31                 } else {
32                     System.out.println("␣[x]␣Sent␣'" + message + "'");
33                 }
34                 Thread.sleep(delay);
35             }
36         }
37     }
38 }
```

2.3 Dockerfile для отправителя и получателя

Для программы отправителя и получателя сообщений были созданы два идентичных Dockerfile. Программы написаны на Java и собирались с помощью Gradle, поэтому используется базовый образ `gradle:8.13-jdk21`.

```
1 FROM gradle:8.13-jdk21
2
3 WORKDIR /app
4
5 COPY . .
6
7 RUN gradle build --no-daemon
8
9 CMD ["gradle", "run", "--no-daemon"]
```

2.4 Настройка Docker Compose

Файл `docker-compose.yml` нужен, чтобы управлять сразу несколькими контейнерами. Файл, представленный ниже, состоит из следующих секций:

- **rabbitmq** – контейнер с RabbitMQ. Используется образ `rabbitmq:4.0-management`, который включает веб-интерфейс для мониторинга. Порты 5672 и 15672 проброшены наружу для AMQP-протокола и веб-интерфейса соответственно.
- **producer1** – первый продюсер, отправляющий сообщения в RabbitMQ. Сборка производится из локальной директории `./producer`. В переменных окружения задаётся имя продюсера (`PRODUCER_NAME`) и задержка между отправкой сообщений в миллисекундах (`PRODUCER_DELAY_MS`).
- **producer2** – второй продюсер, аналогичный первому.
- **consumer** – получатель сообщений из очереди RabbitMQ. Сборка происходит из директории `./consumer`. Консьюмер обрабатывает сообщения с заданной задержкой (`CONSUMER_DELAY_MS`). Дополнительно задаются параметры очереди: максимальная длина (`QUEUE_MAX_LENGTH`) и поведение при переполнении (`QUEUE_OVERFLOW`).

- **networks** – определение виртуальной сети `rabbitnet`, через которую все сервисы обмениваются данными. Используется стандартный сетевой драйвер `bridge`.

```
services:
rabbitmq:
    image: rabbitmq:4.0-management
    hostname: rabbitmq
    container_name: rabbitmq
    ports:
    - "5672:5672"
    - "15672:15672"
    networks:
    - rabbitnet

producer1:
    container_name: producer1
    build:
    context: ./producer
    depends_on:
    - rabbitmq
    - consumer
    environment:
    - PRODUCER_NAME=first
    - PRODUCER_DELAY_MS=1000
    networks:
    - rabbitnet

producer2:
    container_name: producer2
    build:
    context: ./producer
    depends_on:
    - rabbitmq
    - consumer
    environment:
    - PRODUCER_NAME=second
    - PRODUCER_DELAY_MS=1000
    networks:
    - rabbitnet

consumer:
    container_name: consumer
    build:
    context: ./consumer
    depends_on:
    - rabbitmq
    environment:
    - CONSUMER_DELAY_MS=1000
    - QUEUE_MAX_LENGTH=10
    - QUEUE_OVERFLOW=reject-publish # drop-head, reject-publish
    networks:
    - rabbitnet

networks:
rabbitnet:
```

3 Результат работы

Ниже представлены логи для 10 секунд работы контейнеров, в режиме `QUEUE_OVERFLOW=drop-head`. Начиная с 25 строки видно, что получатель начинает получать сообщения только от первого отправителя, потому что сообщения второго просто удаляются.

```
 1 producer1 | [x] Sent 'Message␣from␣first␣#0'
 2 producer1 | [x] Sent 'Message␣from␣first␣#1'
 3 producer2 | [x] Sent 'Message␣from␣second␣#2'
 4 consumer  | [x] Received 'Message␣from␣second␣#1'
 5 producer1 | [x] Sent 'Message␣from␣first␣#2'
 6 producer2 | [x] Sent 'Message␣from␣second␣#3'
 7 consumer  | [x] Received 'Message␣from␣first␣#1'
 8 producer1 | [x] Sent 'Message␣from␣first␣#3'
 9 producer2 | [x] Sent 'Message␣from␣second␣#4'
10 consumer  | [x] Received 'Message␣from␣second␣#2'
11 producer1 | [x] Sent 'Message␣from␣first␣#4'
12 producer2 | [x] Sent 'Message␣from␣second␣#5'
13 consumer  | [x] Received 'Message␣from␣first␣#2'
14 producer1 | [x] Sent 'Message␣from␣first␣#5'
15 producer2 | [x] Sent 'Message␣from␣second␣#6'
16 consumer  | [x] Received 'Message␣from␣second␣#3'
17 producer1 | [x] Sent 'Message␣from␣first␣#6'
18 producer2 | [x] Sent 'Message␣from␣second␣#7'
19 consumer  | [x] Received 'Message␣from␣first␣#3'
20 producer1 | [x] Sent 'Message␣from␣first␣#7'
21 producer2 | [x] Sent 'Message␣from␣second␣#8'
22 consumer  | [x] Received 'Message␣from␣second␣#4'
23 producer1 | [x] Sent 'Message␣from␣first␣#8'
24 producer2 | [x] Sent 'Message␣from␣second␣#9'
25 consumer  | [x] Received 'Message␣from␣first␣#4'
26 producer1 | [x] Sent 'Message␣from␣first␣#9'
27 producer2 | [x] Sent 'Message␣from␣second␣#10'
28 consumer  | [x] Received 'Message␣from␣first␣#5'
29 producer1 | [x] Sent 'Message␣from␣first␣#10'
30 producer2 | [x] Sent 'Message␣from␣second␣#11'
31 consumer  | [x] Received 'Message␣from␣first␣#6'
32 producer1 | [x] Sent 'Message␣from␣first␣#11'
33 producer2 | [x] Sent 'Message␣from␣second␣#12'
34 consumer  | [x] Received 'Message␣from␣first␣#7'
35 producer1 | [x] Sent 'Message␣from␣first␣#12'
36 producer2 | [x] Sent 'Message␣from␣second␣#13'
37 consumer  | [x] Received 'Message␣from␣first␣#8'
38 producer1 | [x] Sent 'Message␣from␣first␣#13'
39 producer2 | [x] Sent 'Message␣from␣second␣#14'
40 consumer  | [x] Received 'Message␣from␣first␣#9'
41 producer1 | [x] Sent 'Message␣from␣first␣#14'
42 producer2 | [x] Sent 'Message␣from␣second␣#15'
43 consumer  | [x] Received 'Message␣from␣first␣#10'
```

Ниже представлены логи для 10 секунд работы контейнеров, в режиме `QUEUE_OVERFLOW=reject-publish`. Начиная с 27 строки, второй отправитель выдаёт сообщение `Message was NOT confirmed!`, это означает, что он получает отказ от RabbitMQ и его сообщение не добавляется в очередь.

```
 1  producer1  | [x] Sent 'Message_from_first_#0'
 2  producer1  | [x] Sent 'Message_from_first_#1'
 3  consumer   | [x] Received 'Message_from_second_#1'
 4  producer2  | [x] Sent 'Message_from_second_#2'
 5  producer1  | [x] Sent 'Message_from_first_#2'
 6  consumer   | [x] Received 'Message_from_first_#1'
 7  producer2  | [x] Sent 'Message_from_second_#3'
 8  producer1  | [x] Sent 'Message_from_first_#3'
 9  consumer   | [x] Received 'Message_from_second_#2'
10  producer2  | [x] Sent 'Message_from_second_#4'
11  producer1  | [x] Sent 'Message_from_first_#4'
12  consumer   | [x] Received 'Message_from_first_#2'
13  producer2  | [x] Sent 'Message_from_second_#5'
14  producer1  | [x] Sent 'Message_from_first_#5'
15  consumer   | [x] Received 'Message_from_second_#3'
16  producer2  | [x] Sent 'Message_from_second_#6'
17  producer1  | [x] Sent 'Message_from_first_#6'
18  consumer   | [x] Received 'Message_from_first_#3'
19  producer2  | [x] Sent 'Message_from_second_#7'
20  producer1  | [x] Sent 'Message_from_first_#7'
21  producer2  | [x] Sent 'Message_from_second_#8'
22  producer1  | [x] Sent 'Message_from_first_#8'
23  consumer   | [x] Received 'Message_from_second_#4'
24  producer2  | [x] Sent 'Message_from_second_#9'
25  producer1  | [x] Sent 'Message_from_first_#9'
26  consumer   | [x] Received 'Message_from_first_#4'
27  producer2  | [!] Message was NOT confirmed!
28  producer1  | [x] Sent 'Message_from_first_#10'
29  consumer   | [x] Received 'Message_from_second_#5'
30  producer2  | [!] Message was NOT confirmed!
31  producer1  | [x] Sent 'Message_from_first_#11'
32  consumer   | [x] Received 'Message_from_first_#5'
33  producer2  | [!] Message was NOT confirmed!
34  producer1  | [x] Sent 'Message_from_first_#12'
35  consumer   | [x] Received 'Message_from_second_#6'
36  producer2  | [!] Message was NOT confirmed!
37  producer1  | [x] Sent 'Message_from_first_#13'
38  consumer   | [x] Received 'Message_from_first_#6'
39  producer1  | [x] Sent 'Message_from_first_#14'
40  consumer   | [x] Received 'Message_from_second_#7'
41  producer2  | [!] Message was NOT confirmed!
42  producer1  | [x] Sent 'Message_from_first_#15'
43  consumer   | [x] Received 'Message_from_first_#7'
44  producer2  | [!] Message was NOT confirmed!
45  producer1  | [x] Sent 'Message_from_first_#16'
46  consumer   | [x] Received 'Message_from_second_#8'
47  producer2  | [!] Message was NOT confirmed!
48  producer1  | [x] Sent 'Message_from_first_#17'
49  consumer   | [x] Received 'Message_from_first_#8'
50  producer2  | [!] Message was NOT confirmed!
51  consumer   | [x] Received 'Message_from_second_#9'
```

```
52  producer2  |  [!] Message was NOT confirmed!
53  producer1  |  [x] Sent 'Message_from_first_#18'
54  consumer   |  [x] Received 'Message_from_first_#9'
55  producer2  |  [!] Message was NOT confirmed!
56  producer1  |  [x] Sent 'Message_from_first_#19'
57  consumer   |  [x] Received 'Message_from_first_#10'
```