Wprowadzenie do systemów rozproszonych

Sprawozdanie z ćwiczenia 4a

Podstawy JMS

Mateusz Jabłoński

1. Wstęp

Celem ćwiczenia jest implementacja poniższych metod klasy JmsSender:

- sendOrderToQueue,
- sendTextToQueue,
- sendMapToTopic,

a także implementacja następujących metod z klasy JmsQueueReceiver:

- JmsQueueReceiver,
- registerCallback,
- shutdown.

W wyniku prawidłowej implementacji wszystkie testy jednostkowe powinny zakończyć się powodzeniem.

Klasa JmsSender zawiera metody służące do wysyłania komunikatów do brokera JMS, natomiast klasa JmsQueueReceiver odpowiada za konsumpcję komunikatów odebranych od brokera JMS.

2. Rozwiązanie

- Pojęcia:
  - **JMS (Java Messaging Service)** – standardowy zestaw interfejsów i modeli asynchronicznego przesyłania komunikatów w języku programowania Java.
  - **kolejka (Queue)** - Point-to-Point za pośrednictwem kolejki komunikatów, nadawca wysyła do określonej nazwanej kolejki (destynacji), odbiorca - odbiera komunikat z tej kolejki:
    - każdy komunikat może być odebrany tylko przez jednego odbiorcę,
    - odbiorca może odebrać wiadomość niezależnie od tego czy nadawca działa czy też juz zakończył działanie,
    - zasadą jest, że odbiorca potwierdza odebranie wiadomości, co zapewnia, że nie będzie mu ona przysłana po raz kolejny.
  - **temat (Topic)** - zasada publikacji i subskrypcji:
    - każda wiadomość może mieć wielu odbiorców (subskrybentów tematu),
    - w danym temacie może publikować wielu nadawców,
    - odbiorca może odbierać tylko te wiadomości z danego tematu, które zostały opublikowane po zapisaniu się przez niego do subskrypcji.

- Modyfikacje kodu
  - sendOrderToQueue

```
46  public void sendOrderToQueue(final int orderId, final String product, final BigDecimal price) {
47    Order order = new Order(orderId, product, price);
48
49    try {
50      ActiveMQConnectionFactory connectionFactory =
51          new ActiveMQConnectionFactory("vm://localhost:62616");
52      connection = connectionFactory.createConnection();
53      connection.start();
54      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
55      Destination destination = session.createQueue(queueName);
56      MessageProducer producer = session.createProducer(destination);
57      producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
58      ObjectMessage objMessage = session.createObjectMessage(order);
59      objMessage.setJMSType(Order.class.getSimpleName());
60      objMessage.setStringProperty("WDSR-System", "OrderProcessor");
61      objMessage.setObject(order);
62      producer.send(objMessage);
63
64    } catch (JMSException e) {
65      e.printStackTrace();
66    } finally {
67      if (connection != null) {
68        try {
69          connection.close();
70        } catch (JMSException e) {
71          e.printStackTrace();
72        }
73      }
74    }
75  }
```

W celu wysłanie komunikatu na kolejkę, w pierwszej kolejności należy utworzyć obiekt typu ActiveMQConnectionFactory (linie 50-51), podając jako argument adres URL brokera. Następnie za pomocą tego obiektu tworzymy nowe połączenie oraz uruchamiamy je. W dalszej kolejności, tworzymy obiekt sesji oraz ustawiamy sposób zatwierdzania transakcji na automatyczny. Kiedy mamy już utworzoną sesje możemy przystąpić kolejno do tworzenia obiektu Destination, któremu przypisujemy nowa utworzoną kolejkę, na którą będą wysyłane komunikaty oraz obiektu MessaageProducer. Instancja obiektu komunikatu deklarowana jest w linii 58. Jako że jest to komunikat typu ObjectMessage, jako argument podajemy obiekt klasy. W tym przypadku jest to obiekt typu Order.

- sendTextToQueue

```
82  public void sendTextToQueue(String text) {
83
84    try {
85      ActiveMQConnectionFactory connectionFactory =
86          new ActiveMQConnectionFactory("vm://localhost:62616");
87      connection = connectionFactory.createConnection();
88      connection.start();
89      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
90      Destination destination = session.createQueue(queueName);
91      MessageProducer producer = session.createProducer(destination);
92      producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
93      Message message = session.createTextMessage(text);
94      producer.send(message);
95
96    } catch (JMSException e) {
97      e.printStackTrace();
98    } finally {
99      if (connection != null) {
100       try {
101         connection.close();
102       } catch (JMSException e) {
103         e.printStackTrace();
104       }
105     }
106   }
107 }
```

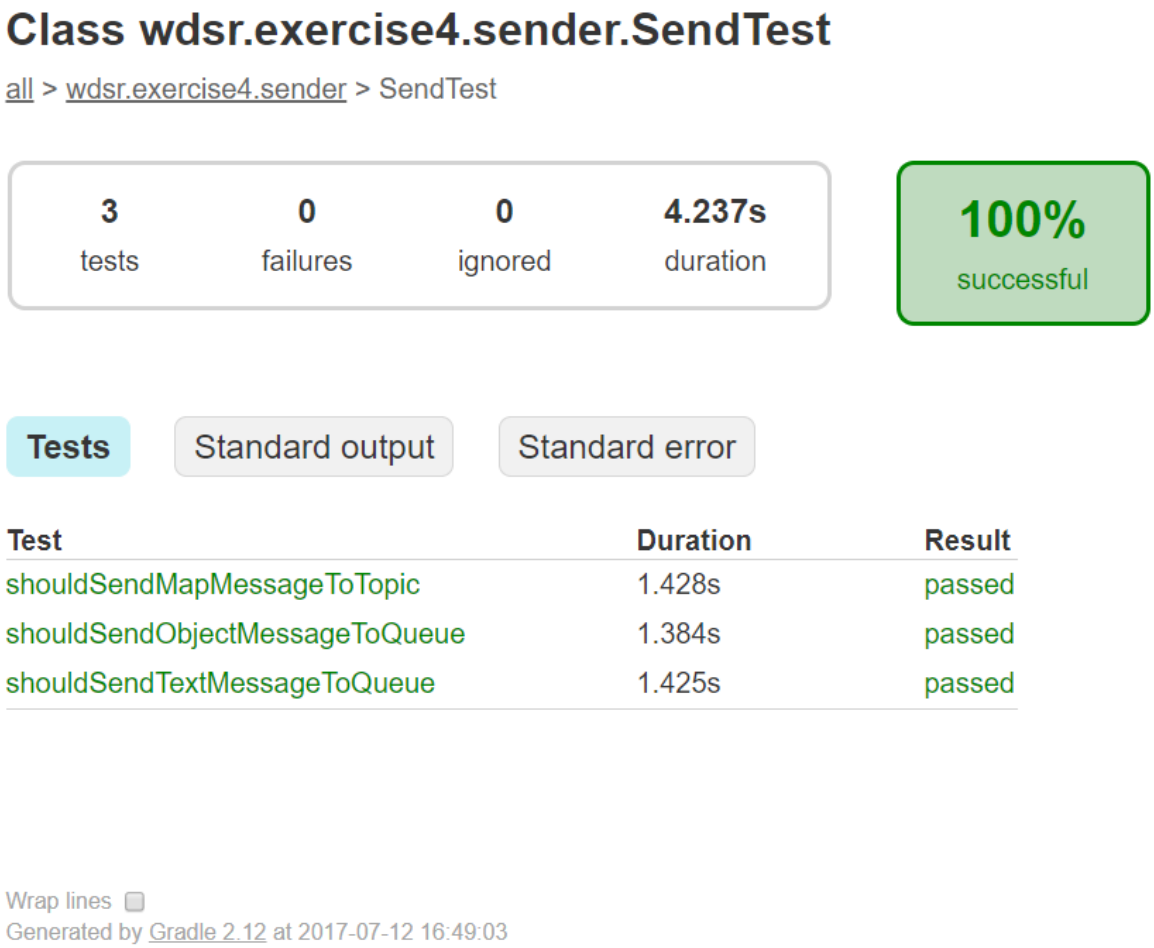
W przypadku wysyłania komunikatu typu TextMessage na kolejkę, należ w linii 93 zmienić typ tworzonego obiektu z ObjectMessage na TextMessage.

- sendMapToTopic

```
114 public void sendMapToTopic(Map<String, String> map) {
115   try {
116     ActiveMQConnectionFactory connectionFactory =
117         new ActiveMQConnectionFactory("vm://localhost:62616");
118     connection = connectionFactory.createConnection();
119     connection.start();
120     Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
121     Destination destination = session.createTopic(topicName);
122     MessageProducer producer = session.createProducer(destination);
123     producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
124     MapMessage message = session.createMapMessage();
125     Set<String> keys = map.keySet();
126
127     for (String key : keys) {
128       String value = map.get(key);
129       message.setObject(key, value);
130     }
131
132     producer.send(message);
133
134   } catch (JMSException e) {
135     e.printStackTrace();
136   } finally {
137     if (connection != null) {
138       try {
139         connection.close();
140       } catch (JMSException e) {
141         e.printStackTrace();
142       }
143     }
```

W celu wysłania mapy do topicu została utworzona kolekcja kluczy odebranej mapy (linia 125), a następnie dla każdego z kluczy pobierana jest odpowiadająca mu wartość w mapie (linie 127-128). Wykorzystując odpowiadające sobie pary klucz-wartość tworzony jest nowy element, który jest następnie dodawany do przygotowywanego do wysłania komunikatu (linia 129).

- Testy



Class wdsr.exercise4.sender.SendTest

all > wdsr.exercise4.sender > SendTest

3	0	0	4.237s
tests	failures	ignored	duration

100% successful

Tests | Standard output | Standard error

Test	Duration	Result
shouldSendMapMessageToTopic	1.428s	passed
shouldSendObjectMessageToQueue	1.384s	passed
shouldSendTextMessageToQueue	1.425s	passed

Wrap lines

Generated by Gradle 2.12 at 2017-07-12 16:49:03

3. Wnioski

JMS umożliwia przesyłanie komunikatów między aplikacjami z wykorzystaniem kolejki lub kanału. Rozwiązanie te, różnią się przede wszystkim tym, że w kolejce komunikat może zostać odebrany tylko przez jednego odbiorcę, natomiast w przypadku kanału przez żadnego, jednego lub wielu, w zależności od liczby subskrybentów danego kanału. Komunikaty mogą być odbierane synchronicznie, poprzez wywołanie metody receive lub asynchronicznie przy użyciu obiektu klasy implementującej interfejs MessageListener.

4. Literatura

1. Java EE 6 Zaawansowany przewodnik, praca zbiorowa:
   - Zagadnienia technologii JMS: s. 337-370.
2. https://docs.oracle.com/javaee/7/tutorial/jms-concepts003.htm
3. http://activemq.apache.org/hello-world.html
4. http://activemq.apache.org/uri-protocols.html