Messaging with JMS

Paweł Śliwa
Wiktor Sokół
Adam Zych

Jakarta Messaging 3.0

Jakarta Messaging opisuje sposób, w jaki aplikacje Java mogą tworzyć, wysyłać i odbierać wiadomości za pomocą luźno powiązanych, niezawodnych asynchronicznych usług komunikacyjnych.

Ogólna koncepcja przesyłania wiadomości

Przesyłanie komunikatów jest formą luźno sprzężonej komunikacji rozproszonej, przy czym w tym kontekście termin "komunikacja" można rozumieć jako wymianę komunikatów między komponentami oprogramowania. Technologie zorientowane na komunikaty próbują rozluźnić ściśle sprzężoną komunikację (taką jak gniazda sieciowe TCP, CORBA czy RMI) poprzez wprowadzenie komponentu pośredniczącego. Takie podejście pozwala komponentom oprogramowania komunikować się ze sobą w sposób pośredni.

Wskazówka

Do zalet przesyłania komunikatów należy możliwość integracji heterogenicznych platform, zmniejszenie wąskich gardeł systemu, zwiększenie skalowalności i szybsze reagowanie na zmiany

W celu przesyłania wiadomości konieczne jest współdziałanie trzech typów uczestników: nadawców wiadomości, odbiorców wiadomości oraz dostawcy wiadomości, który pośredniczy pomiędzy nadawcami i odbiorcami

JMS Provider - Implementacja interfejsu JMS dla oprogramowania pośredniczącego zorientowanego na komunikaty (MOM). Dostawcy są implementowane jako implementacja Java JMS lub adapter do MOM nie-Java.

JMS Producer - Klient JMS, który tworzy i wysyła komunikaty.

JMS Client - Klient JMS, który odbiera komunikaty.

Wskazówka

Serwer Jakarta EE 8 nie musi zawierać dostawcy komunikatów; w środowisku korporacyjnym często korzysta się z dostawców komunikatów innych firm

Paradygmaty przesyłania wiadomości

➔ Point-To-Point Messaging

W domenie punkt-punkt producenci komunikatów są nazywani nadawcami, a odbiorcy konsumentami. Wymieniają oni komunikaty za pomocą miejsca docelowego zwanego kolejką: nadawcy tworzą komunikaty do kolejki, a odbiorcy konsumują komunikaty z kolejki. Komunikacja punkt-punkt wyróżnia się tym, że komunikat może być konsumowany tylko przez jednego konsumenta..

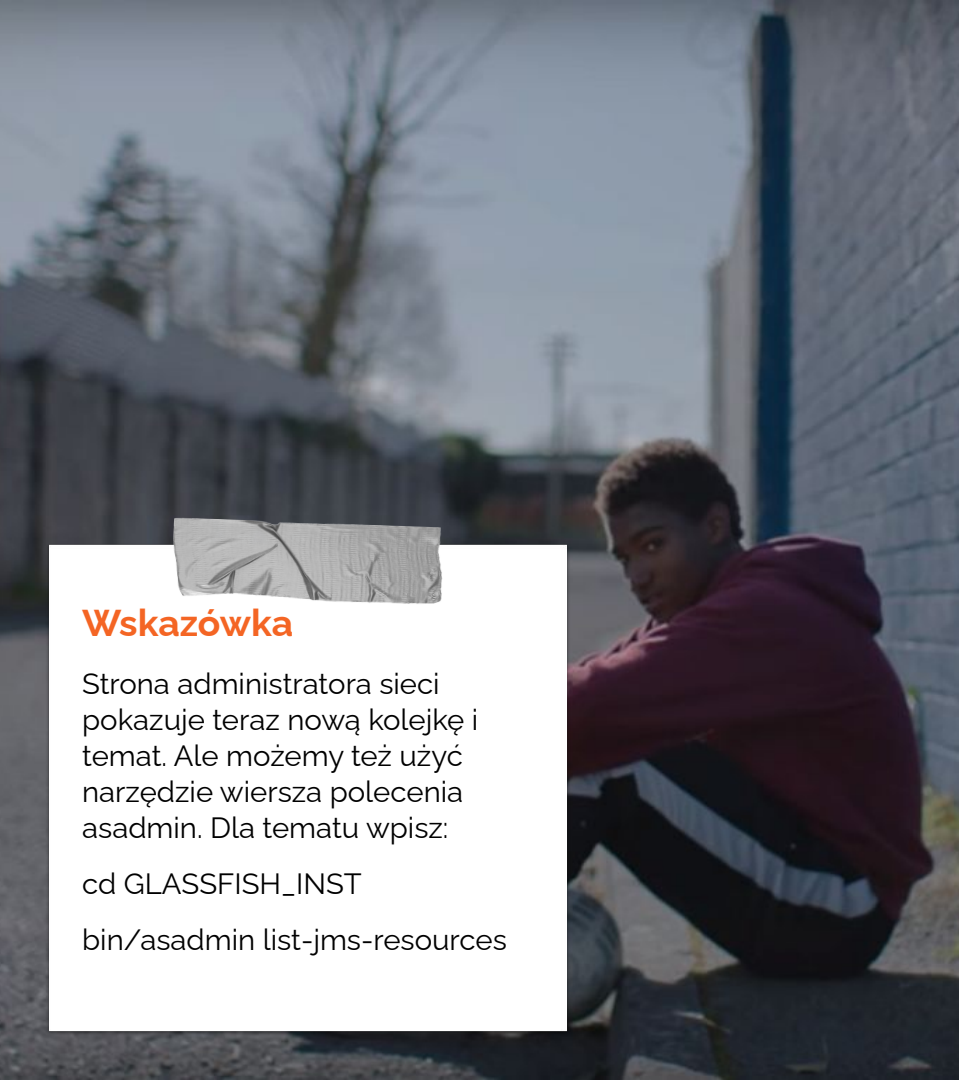
➔ Publish/Subscribe Messaging

Model komunikatów JMS publish/subscribe (Pub-Sub) jest modelem typu jeden do wielu. Wydawca wysyła wiadomość do tematu, a wszyscy aktywni subskrybenci tematu otrzymują tę wiadomość. Subskrybenci, którzy nie słuchają aktywnie tematu, nie otrzymają opublikowanego komunikatu.

Konfigurowanie dostawcy komunikatów

Aby opracować przykład kolejki JMS, należy zainstalować dowolny serwer aplikacji. Tutaj używamy serwera glassfish3, na którym tworzymy dwa JNDI.

Najpierw tworzymy fabrykę połączeń, a następnie tworzymy zasób docelowy Po utworzeniu JNDI tworzymy aplikację serwera i odbiornika. Serwer i odbiornik należy uruchomić w innej konsoli.



Wskazówka

Strona administratora sieci pokazuje teraz nową kolejkę i temat. Ale możemy też użyć narzędzie wiersza polecenia asadmin. Dla tematu wpisz:

cd GLASSFISH_INST

bin/asadmin list-jms-resources

Tworzenie kolejek i tematów

Aby utworzyć kolejkę lub temat w ramach dostawcy JMS serwera Glassfish, przejdź do Resources ➤ JMS Resources ➤ Destination Resources i kliknij przycisk “New...”.

Przykłady utworzenia kolejki i tematu:

```
JNDI Name:                 jms/TestQueue
Physical Destination Name: TestQueue
Resource Type:             javax.jms.Queue
Description:               Test Queue
Status:                    [x]

JNDI Name:                 jms/TestTopic
Physical Destination Name: TestTopic
Resource Type:             javax.jms.Topic
Description:               Test Topic
Status:                    [x]
```

Wysyłanie wiadomości

Na następnym slajdzie znajduje się przykład demonstrujący użycie klasycznego interfejsu API. Tworzy on prostą wiadomość tekstową i wysyła ją do kolejki. Ten sam komunikat jest następnie odczytywany przez konsumenta komunikatów z tej samej kolejki.

```java
package lab00.classic.helloworld;

import javax.jms.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class TestHelloWorldQueue {
  public static void main(String[] args) throws NamingException {
    InitialContext initialContext = null;

    try {
      initialContext = new InitialContext();

      //Step-1 Create ConnectionFactory
      ConnectionFactory connectionFactory
          = (ConnectionFactory) initialContext.lookup("jms/__defaultConnectionFactory");

      //Step-2 Create connection
      Connection connection = connectionFactory.createConnection();

      //Step-3 Create Session
      //Any number of sessions can be created from a connection
      Session session = connection.createSession();

      //Step-4 Get the Queue
      Queue queue = (Queue) initialContext.lookup("jms/PTPQueue");

      //Step-5a Create the message Producer
      MessageProducer messageProducer = session.createProducer(queue);
      //Step-6a Create the Text Message
      TextMessage textMessage = session.createTextMessage("Test message - Hello");
      //Step-7a Send the message
      messageProducer.send(textMessage);

      //Step-5b Create the message consumer
      MessageConsumer messageConsumer = session.createConsumer(queue);
      //Step-6b Start the connection
      connection.start(); //to start delivery
      //step-7b Receive the message
      TextMessage message = (TextMessage) messageConsumer.receive();
      System.out.println(message.getText());
    } catch (NamingException | JMSException ex) {
      ex.printStackTrace();
    } finally {
      if (initialContext != null) {
        initialContext.close();
      }
    }
  }
}
```

1. Najpierw tworzymy ConnectionFactory przy użyciu JNDI lookup. Informacje na temat JNDI konfigurujemy w pliku jndi.properties. Pojedyncze ConnectionFactory w zupełności wystarczy dla każdej aplikacji lub usługi.

2. Następnie z ConnectionFactory pobieramy obiekt Connection

3. Na podstawie połączenia tworzymy sesję.

4. Tworzymy obiekty MessageProducer i MessageConsumer.

5. Uruchamiamy Connection.

Odbieranie wiadomości

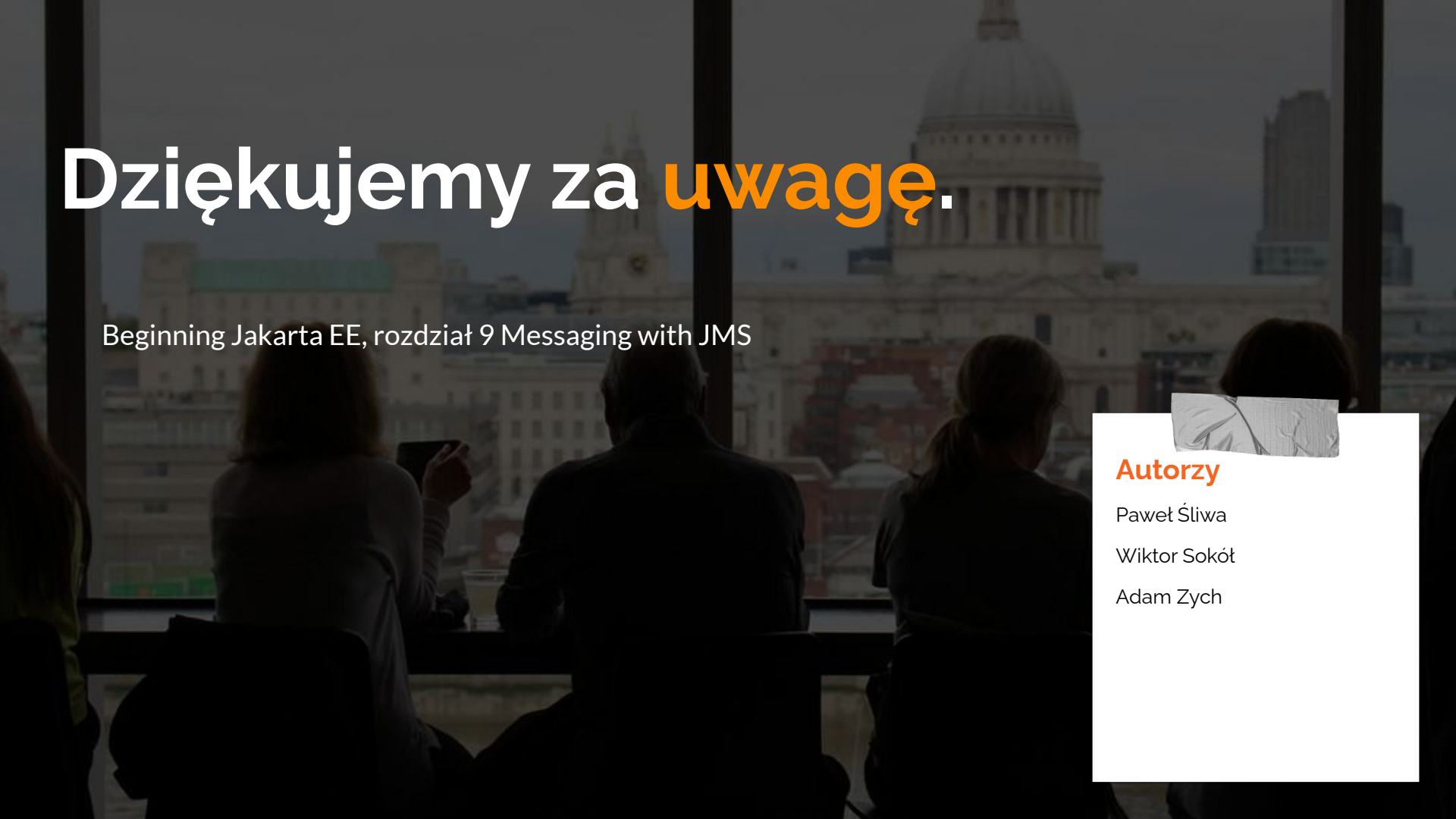
Wykonamy to samo zadanie polegające na wysłaniu wiadomości “hello” do kolejki i odczytaniu jej za pomocą Konsumenta. Przyjrzyjmy kodowi na następnym slajdzie.

```java
import javax.jms.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class TestHelloWorld {
  public static void main(String[] args) {

    ConnectionFactory connectionFactory = null;
    Queue queue = null;
    try {

      InitialContext initialContext = new InitialContext();
      //Step-1 Create ConnectionFactory
      connectionFactory
          = (ConnectionFactory) initialContext.lookup("jms/__defaultConnectionFactory");

      //Step-2
      queue = (Queue) initialContext.lookup("jms/PTPQueue");
    } catch (NamingException e) {
      e.printStackTrace();
    }

    //Step-3
    try (JMSContext jmsContext = connectionFactory.createContext()) {

      //Step-4a
      TextMessage textMessage = jmsContext.createTextMessage("Message using JMS 2.0");
      JMSProducer jmsProducer = jmsContext.createProducer().send(queue, textMessage);

      //Step-4b
      TextMessage message = (TextMessage) jmsContext.createConsumer(queue).receive();
      System.out.println(message.getText());
    } catch (JMSException e) {
      e.printStackTrace();
    }
  }
}
```

1. Stwórz **ConnectionFactory**. Nie ma potrzeby tworzenia Connection i Session w przeciwieństwie do Classic API.
2. Uzyskaj **Destination**, w tym przykładzie jest to Queue z InitialContext.
3. Utwórz **JMSContext** z connectionFactory.
4. Użyj **JMSProducer**, aby wysłać wiadomość do serwera Messaging.
5. Podobnie użyj **JMSConsumer**, aby odebrać wiadomość z JMS.

Dziękujemy za uwagę.

Beginning Jakarta EE, rozdział 9 Messaging with JMS



Autorzy

Paweł Śliwa

Wiktor Sokół

Adam Zych