Message-Oriented Middleware with JMS

Synchronous vs Asynchronous

The Middleware technologies considered so far used synchronous function- or method-calls

- Client initiates a call
- Call is processed on server; client is waiting
- When the server returns a result the execution of the client is continued

Problems

- client is blocked while the server is busy
- If a server fails, all clients that have to interact with the server can not continue with their work

Message-Oriented Middleware (MOM)

Based on asynchronous message exchange

- sending of messages is non-blocking

Users of Message Queuing communicate via inserting and removing messages in queues

- Systems do not communicate directly but only with the queuing system

Client and Server can be active at different periods of time

Java Message Service (JMS)

JMS standardizes an API for Java applications to access a Queuing-System

Queuing-Systems that implement JMS are offered from different vendors and as open source software

The standard distinguishes two roles

- JMS Provider: The Queuing-System, that is implementing the JMS API
- JMS Client: The Java Application, that is sending and receiving messages

Administered Objects

Connection Factory

- Contains all information needed by a JMS Client to establish connections to a JMS Provider
- Provides methods for the initialization of a connection

Destination

- Represents a concrete message storage facility of a JMS Providers
- Is used by a JMS Client for inserting and reading of messages

Both kinds of objects are made available by the JMS Provider through a naming service

- Naming service is usually JNDI (Java Naming and Directory Interface)

Communication-Model Point-to-Point

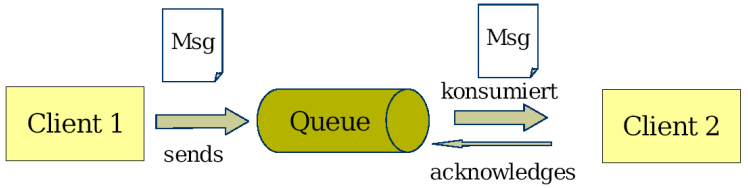
Destination = Queue

Sender (producer) creates message and sends it to a Queue

Each message has only one receiver (consumer)

Receiver acknowledges the receipt of message

Queue stores message persistent until it is read by receiver



Communication-Model Publish/Subscribe

Destination = Topic

Sender is publishing messages for a particular Topic

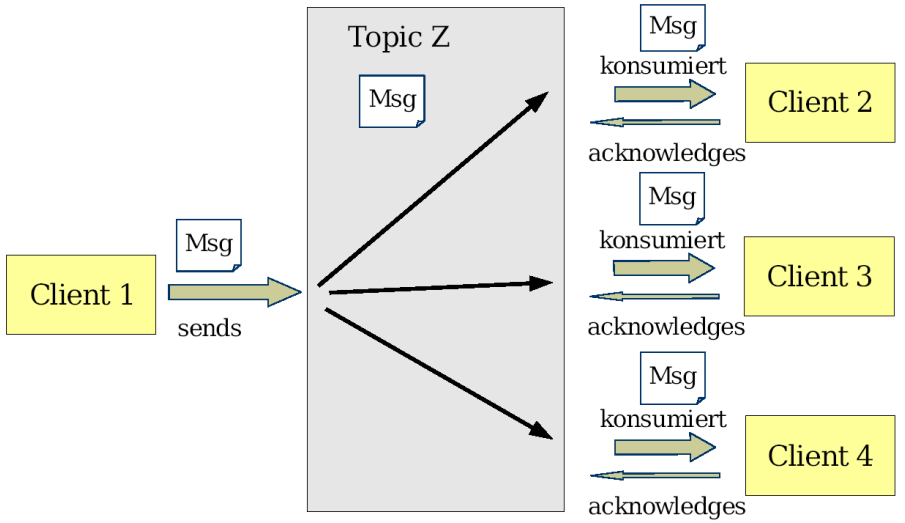
Multiple receiver can subscribe messages

- In this case they will get all messages published for a certain topic

Receiver is getting messages only while its process is active

- Variant: Durable Subscriber for the storage of messages

Publish/Subscribe



Important Classes

Connection: represents the JMS Provider

Session: a sequence of message exchanges with a JMS Provider

MessageProducer: Sender

- Subclasses: QueueSender and TopicPublisher

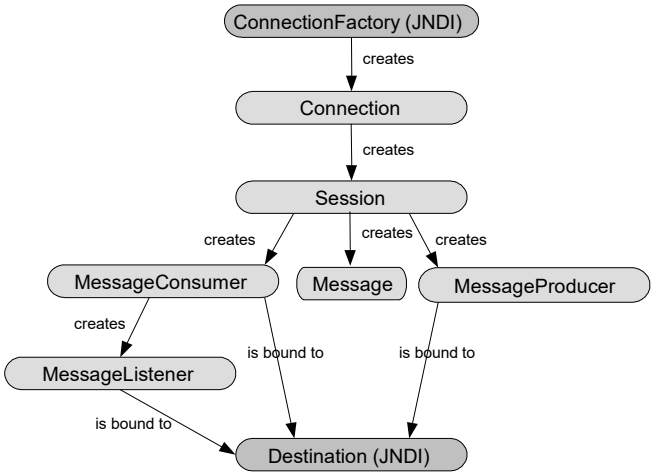
MessageConsumer: Receiver

- Subclasses : QueueReceiver and TopicSubscriber

Destination: the messages storage facility

- Subclasses : Queue und Topic

Relationship of Objects



Structure Sender

1. Create JNDI-Context
2. Find ConnectionFactory
3. Create Connection and Session
4. Start Connection
5. Find Destination
6. Create MessageProducer
7. Create and send message
8. Close Connection and Session

Sender Publish/Subscribe (1)

```
import javax.jms.*;
import javax.naming.*;
import java.util.*;

public class SenderT {
  public static void main(String argv[]) throws Exception {

    Hashtable properties = new Hashtable<String, String>();
    properties.put(Context.INITIAL_CONTEXT_FACTORY,
          "org.apache.activemq.jndi.ActiveMQInitialContextFactory");
    properties.put(Context.PROVIDER_URL,
                        "tcp://localhost:61616");

   Context context = new InitialContext(properties);
```

Sender Publish/Subscribe (2)

```
    TopicConnectionFactory connFactory = (TopicConnectionFactory)
                    context.lookup("ConnectionFactory");
    TopicConnection conn = connFactory.createTopicConnection();
    conn.start();
    TopicSession session =
     conn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
    Topic topic = (Topic) context.lookup("dynamicTopics/topic1");
    TopicPublisher publisher = session.createPublisher(topic);
    TextMessage message = session.createTextMessage("Hello
                                                        World!");
    System.out.println("Sending of: "+ message.getText());
    publisher.publish(message);
    session.close();
    conn.close();
  }
}
```


Structure Receiver

1. Create JNDI-Context
2. Find ConnectionFactory
3. Create Connection and Session
4. Start Connection
5. Find Destination
6. Create MessageConsumer
7. Receive and process message
8. Close Connection and Session

Alternative for non-blocking receive: use and register a MessageListeners

Receiver Publish/Subscribe (Modifications)

```
TopicSubscriber subscriber =
session.createSubscriber(topic);
TextMessage message =
      (TextMessage) subscriber.receive();
System.out.println("received message: " +
       message.getText());
```


Session Parameters (1)

When we create a session, we specify two parameters

- createTopicSession(boolean transacted, int acknowledgeMode) or
- createQueueSession(boolean transacted, int acknowledgeMode)

First parameter: Use a transaction or not?

- when set to true, we use a transaction and the second parameter is ignored
- transaction: all or nothing property
- is relevant for sending as well as receiving
- for a transaction we have to specify the outcome:
  - commit() or
  - rollback()

Session Parameters (2)

Second parameter: When does the consumer send an acknowledgement?

- only after an acknowledgement a persistent message is deleted from the destination
- this approach makes sure that receiving and processing of the message is successful
- Session.AUTO_ACKNOWLEDGE means that an acknowledgement is immediately send by the receiver without explicit actions
- Session.CLIENT_ACKNOWLEDGE means that we have to send an explicit acknowledgement for each received message
  - method acknowledge() invoked on the message

JMS Message Types

TextMessage
- contains String

MapMessage
- contains set of name/value-pairs

ByteMessage
- stream of uninterpreted Bytes

StreamMessage
- stream of primitive data types

ObjectMessage
- serialized Object

Conclusions

Message-Oriented Middleware and JMS aim at a loose coupling of systems

JMS is the only technology we discussed that allows communication between partners not active at the same time

Functionality focused on pure data exchange
- MOM is a good candidate to integrate legacy applications

MOM systems provide powerful mechanisms to realize a transaction-oriented execution of message exchange