Processamento Paralelo Middleware Orientado a Mensagens

Prof. João Paulo A. Almeida
(jpalmeida@inf.ufes.br)

Middleware Orientado a Mensagens

- Java oferece um serviço de nomes que implementa o "publish-subscriber" design pattern com esse fim → Java Message Service (JMS)
- Várias plataformas de middleware oferecem funcionalidades similares
  - CORBA Event Service, CORBA Notification Service
  - Oracle Advanced Queueing
  - TIBCO Enterprise Message Service
  - Apache ActiveMQ
    - Apart from Java, ActiveMQ can be also used from .NET, C/C++ or Delphi or from scripting languages like Perl, Python, PHP and Ruby via various "Cross Language Clients"
  - JBOSS Messaging

Java Messaging Service (JMS)

- JMS dá suporte a dois modos de interação por mensagem:
- Publisher/subscriber (através de "Topics")
  - m-to-n messaging
- Message queue ("através de "Queues")
  - Point-to-point (PTP) messaging
- O conceito geral de "destination" é usado

JMS

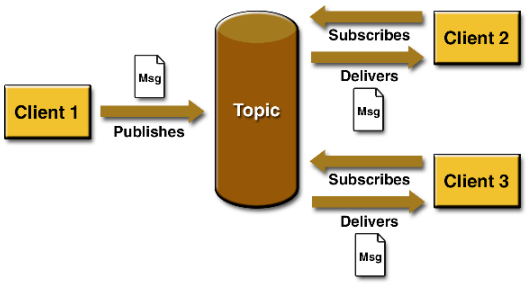
- API para permitir acesso uniforme a várias implementações de middleware orientado a mensagem
  - Apache ActiveMQ
  - Apache Qpid, using AMQP[5]
  - Oracle Weblogic (part of the Fusion Middleware suite) and Oracle AQ from Oracle
  - EMS from TIBCO
  - FFMQ, GNU LGPL licensed
  - JBoss Messaging and HornetQ from JBoss
  - JORAM, from the OW2 Consortium
  - Open Message Queue, from Oracle
  - OpenJMS, from The OpenJMS Group
  - Solace JMS from Solace Systems
  - RabbitMQ by Rabbit Technologies Ltd., acquired by SpringSource
  - SAP Process Integration ESB
  - SonicMQ from Progress Software
  - SwiftMQ
  - Tervela
  - Ultra Messaging from 29 West (acquired by Informatica)
  - webMethods from Software AG
  - WebSphere Application Server from IBM, which provides an inbuilt default messaging provider known as the Service Integration Bus (SIBus), or which can connect to WebSphere MQ as a JMS provider[6]
  - WebSphere MQ (formerly MQSeries) from IBM
  - Fonte: http://en.wikipedia.org/wiki/Java_Message_Service

Motivação dada no tutorial de JMS

http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/overview.html

- An enterprise application provider is likely to choose a messaging API over a tightly coupled API, such as Remote Procedure Call (RPC), under the following circumstances:
  - The provider wants the components not to depend on information about other components' interfaces, so that components can be easily replaced.
  - The provider wants the application to run whether or not all components are up and running simultaneously.
  - The application business model allows a component to send information to another and to continue to operate without receiving an immediate response.

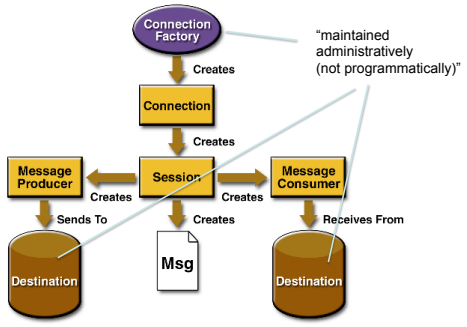
JMS



JMS: Suporte a pull e push para consumidor

- **Pull bloqueante**
  - A subscriber or a receiver explicitly fetches the message from the destination by calling the **receive** method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit. [JMS chama de **synchronous**]
- **Push**
  - A client can register a *message listener* with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's **onMessage** method, which acts on the contents of the message. [JMS chama de **asynchronous**]

Modelo de programação JMS



Destination é um nome geral para Topic / Queue

Criando Destinations (Topics)

- A *destination* is the object a client uses to specify the target of messages it produces and the source of messages it consumes:

Versão antiga J2EE:
j2eeadmin -addJmsDestination <*topic_name*> topic

Versão nova J2EE:
asadmin add-resources glassfish-resources.xml
Glassfish Admin Console:
http://localhost:4848/

Glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>

  <admin-object-resource enabled="true" jndi-name="jms/MyTopic" object-type="user" res-adapter="jmsra" res-type="javax.jms.Topic">
    <description/>
    <property name="Name" value="PhysicalTopic"/>
  </admin-object-resource>

</resources>
```

Criando uma ConnectionFactory

- With the J2EE SDK, for example, you can use the default connection factory objects, named TopicConnectionFactory, to create connections.

Publisher

```
jndiContext = new InitialContext();
topicConnectionFactory = (TopicConnectionFactory)
      jndiContext.lookup("TopicConnectionFactory");
topic = (Topic) jndiContext.lookup("MyTopic");
topicConnection =
      topicConnectionFactory.createTopicConnection();
topicSession = topicConnection.createTopicSession(
      false, Session.AUTO_ACKNOWLEDGE);
topicPublisher = topicSession.createPublisher(topic);
message = topicSession.createTextMessage();
for (int i = 0; i < NUM_MSGS; i++) {
      message.setText("This is message " + (i + 1));
      System.out.println(
            "Publishing message: " + message.getText());
      topicPublisher.publish(message);
} topicConnection.close();
```

Subscriber (push)

```
jndiContext = new InitialContext();
topicConnectionFactory = (TopicConnectionFactory)
      jndiContext.lookup("TopicConnectionFactory");
topic = (Topic) jndiContext.lookup("MyTopic");
topicConnection =
      topicConnectionFactory.createTopicConnection();
topicSession = topicConnection.createTopicSession(
      false, Session.AUTO_ACKNOWLEDGE);
topicSubscriber = topicSession.createSubscriber(topic);
topicListener = new TextListener();
topicSubscriber.setMessageListener(topicListener);
topicConnection.start();
```

As mensagens enviadas a partir deste momento serão recebidas por este Subscriber

Message Listener

```
import javax.jms.*;

public class TextListener implements MessageListener
{
      public void onMessage(Message message) {
         TextMessage msg = null;
         if (message instanceof TextMessage) {
                msg = (TextMessage) message;
                System.out.println(
                    "Reading message: "+msg.getText());
         }
      }
}
```

Tipos de mensagens

Tipo	Conteúdo
TextMessage	A java.lang.String object (for example, the contents of an Extensible Markup Language file).
MapMessage	A set of name/value pairs, with names as String objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
BytesMessage	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
StreamMessage	A stream of primitive values in the Java programming language, filled and read sequentially.
ObjectMessage	A Serializable object in the Java programming language.
Message	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

Cabeçalhos de mensagens

- Além de ter um "corpo" as mensagens podem ter cabeçalhos com atributos e valores (propriedades)
- Property values can be *boolean*, *byte*, *short*, *int*, *long*, *float*, *double*, and *String*.
- Na classe javax.jms.Message:
  - public void **setBooleanProperty**(String name, boolean value) throws JMSException
  - public void **setByteProperty**(String name, byte value) throws JMSException
  - …

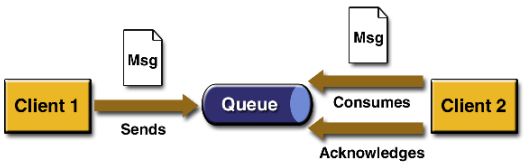
Filtro de mensagens

- JMS permite filtrar mensagens com base em cabeçalhos (através de um Message Selector)
- O consumidor vai receber apenas as mensagens que satisfazem uma expressão booleana, considerando os valores dos cabeçalhos
- A expressão é descrita em um subconjunto da sintaxe de expressões condicionais de SQL92
- "JMSType = 'car' AND color = 'blue' AND weight > 2500"
- "Country IN (' UK', 'US', 'France')"

Filtro de mensagens

- É definido na criação do Subscriber
- Interface java.jmx.TopicSession
- public TopicSubscriber **createSubscriber**(Topic topic, java.lang.String messageSelector, boolean noLocal)

JMS Point-to-Point Messaging



Point-to-Point Messaging

- While any number of producers can send messages to the queue, each message is guaranteed to be delivered, and consumed by one consumer.
- Queues retain all messages sent to them until the messages are consumed or until the messages expire. If no consumers are registered to consume the messages, the queue holds them until a consumer registers to consume them.

Criando Destinations (Queues)

- A *destination* is the object a client uses to specify the target of messages it produces and the source of messages it consumes:

j2eeadmin -addJmsDestination <*queue_name*> queue