Servicios de Mensajes en JEE

Sesión1: JMS: Java Message Service (1)

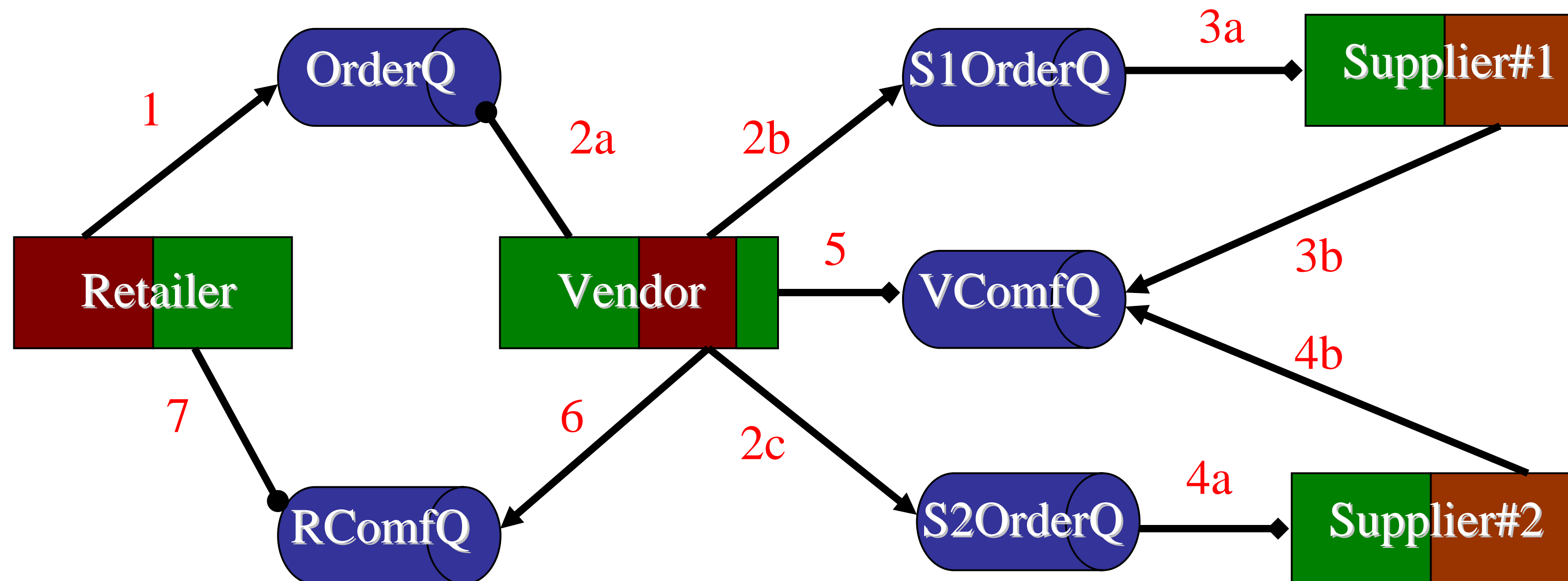
Indice

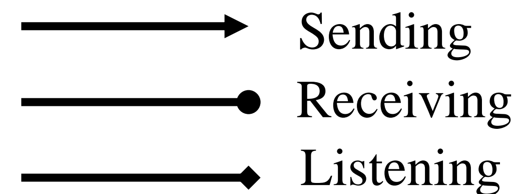
- Motivación: Interacción con JMS
- Arquitectura JMS
- Dominios de mensajes: PTP y Pub/sub
- Creando un cliente PTP Creando un cliente Pub/sub

Indice

- Motivación: Interacción con JMS
- Arquitectura JMS
- Dominios de mensajes: PTP y Pub/sub
- Creando un cliente PTP
- Creando un cliente Pub/sub

Motivación





Indice

- Motivación: Interacción con JMS
- Arquitectura JMS
- Dominios de mensajes: PTP y Pub/sub
- Creando un cliente PTP
- Creando un cliente Pub/sub

Arquitectura JMS

- Objetivo e implementación:
  - Comunicación entre componentes software:
    - Independencia de componentes (facilita reemplazamiento).
    - No es necesario que todas estén simultáneamente en ejecución.
    - Permitir comunicación sin que se acuse el recibo.
  - Solución aportada por JMS:
    - Comunicación síncrona y asíncrona, y segura (garantías).
    - Integración en servicios transaccionales.
  - Conceptos básicos de JMS:
    - Arquitectura del API
    - Dominios de mensajes
    - Consumo de mensajes

Arquitectura JMS

- Elementos de la arquitectura JMS:

1. Proveedor JMS:
    - Sistema de mensajes que implementa las interfaces de JMS.
    - Proporciona administración de recursos y control.
    - Incluido en implementaciones de J2EE (p.e. Bea WebLogic)
2. Clientes JMS:
    - Programas o componentes Java que producen y/o consumen mensajes.
3. Mensajes:
    - Objetos que comunican clientes JMS.
    - Estructura: cabecera + propiedades + cuerpo.

© 2006-2007 Depto. Ciencia de la Computación e IA

Arquitectura JMS

- Elementos de la arquitectura JMS:
  4. Objetos administrados:
     - Objetos pre-configurados creados por el administrador.
     - Hay dos tipos: destinos y factorías de conexión.
     - `Destinations:` Objetos usados por los clientes JMS para especificar el destino de los mensajes que producen o el origen de los mensajes que consumen.
     - `Connection factories:` Objetos usados por los clientes JMS para conectar con un proveedor. Encapsulan parámetros definidos por el administrador.
  5. Clientes nativos:
     - Programas que usan un API nativo en lugar de JMS.

Arquitectura JMS

Connection Factory

Creates

Message Producer

Connection

Message Consumer

Sends to

Creates

Creates

Creates

Rec. from

Destination

Session

Destination

Creates

Msg

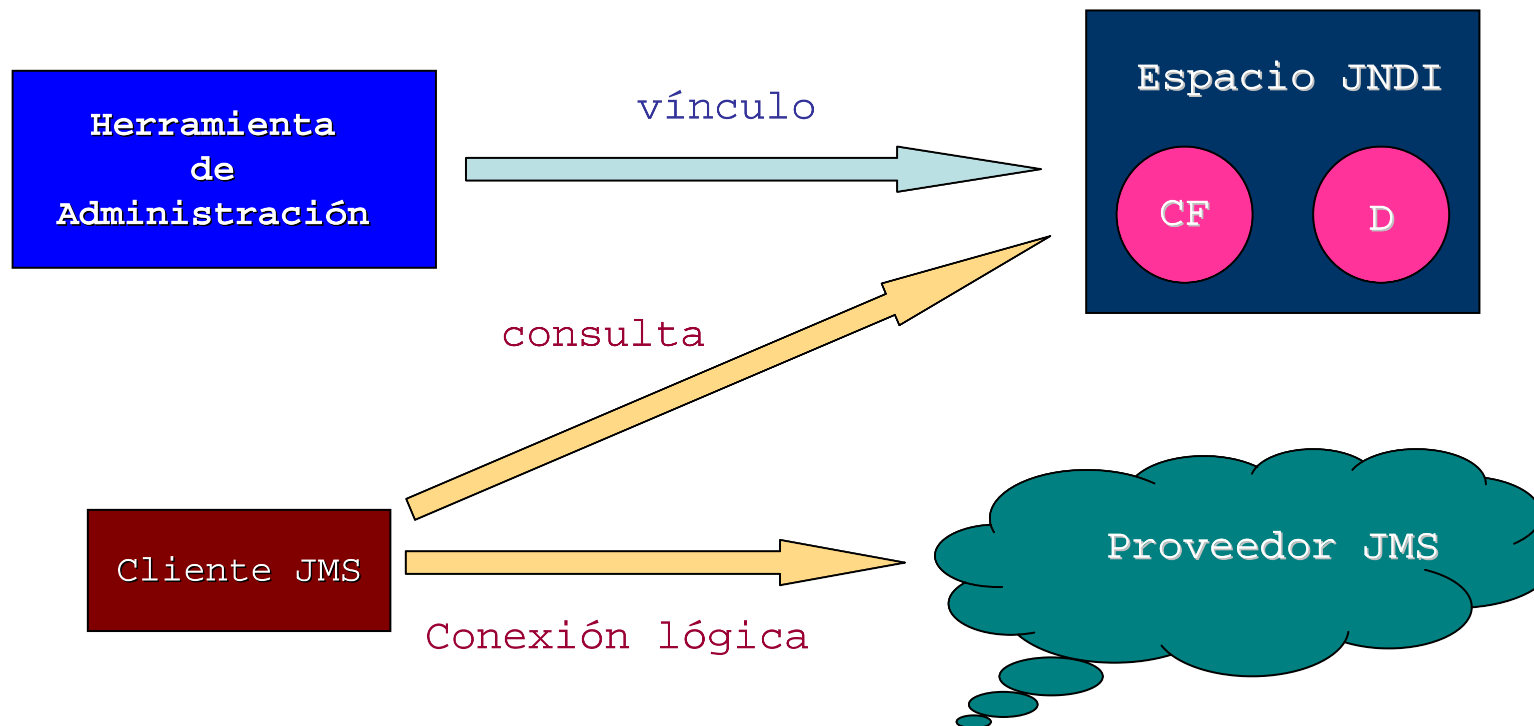
© 2006-2007 Depto. Ciencia de la Computación e IA

Arquitectura JMS

- Interacción:
  - Administración - JNDI:
    - Las herramientas de administración permiten vincular destinos y factorías de conexión a través de un espacio de nombres JNDI.
  - Clientes JMS – JNDI – Proveedor JMS:
    - Entonces, un cliente JMS puede consultar los objetos administrados en dicho espacio y establecer conexiones lógicas con ellos a través del proveedor JMS.
    - Estas conexiones le permiten enviar o recibir mensajes de acuerdo con el modelo definido por JMS.

Arquitectura JMS

- Interacción: (esquema)



Indice

- Motivación: Interacción con JMS
- Arquitectura JMS
- Dominios de mensajes: PTP y Pub/sub
- Creando un cliente PTP
- Creando un cliente Pub/sub

© 2006-2007 Depto. Ciencia de la Computación e IA

Dominios de mensajes: PTP y Pub/sub

- JMS = comunicación sinc{asín}crona + segura:
  - PTP (punto a punto):
    - Un cliente produce y envía mensajes a la cola de un cliente receptor que los va leyendo (consume) secuencialmente.
    - Mecanismo 1-a-1: un mensaje solamente tiene un consumidor que puede recoger el mensaje cuando quiera (indicándolo).
  - Pub/sub (Publicar/subscribir):
    - Una aplicación cliente publica sus mensajes en tópicos o apartados a los que otros clientes, interesados en dichos tópicos, se han subscrito.
    - Estas conexiones le permiten enviar o recibir mensajes de acuerdo con el modelo definido por JMS.

- PTP y Pub/sub: (esquemas)

Cliente JMS

Envía

Cola

Consume

Cliente JMS

Acusa recibo

Subscribe

Cliente#2

Recoge

Cliente#1

Publica

Tópico

Subscribe

Cliente#3

Recoge

Indice

- Motivación: Interacción con JMS
- Arquitectura JMS
- Dominios de mensajes: PTP y Pub/sub
- Creando un cliente PTP
- Creando un cliente Pub/sub

© 2006-2007 Depto. Ciencia de la Computación e IA

Creando un cliente PTP

- Ejemplo: `QueueSend.java`
  1. Establecer un contexto JNDI
  2. Configuración de la comunicación:
     1. Localizar una `ConnectionFactory`
     2. Crear una `Connection`
     3. Crear una `Session`
     4. Localizar/crear un `Destination`
     5. Crear `Producers{Consumers}`
     6. Crear el objeto `Message`
     7. Iniciar la conexión
  3. Enviar mensajes
  4. Cerrar la comunicación

© 2006-2007 Depto. Ciencia de la Computación e IA

Creando un cliente PTP

- Ejemplo: `QueueSend.java`
  1. Establecer contexto JNDI:
     - Una ConnectionFactory es un objeto administrado por WebLogic cuya implementación proporciona una por defecto.
     - Para poder localizar una factoría se precisa establecer el contexto JNDI:

```
private static InitialContext getInitialContext(String url)
        throws NamingException
  {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, JNDI_FACTORY);
    env.put(Context.PROVIDER_URL, url);
    return new InitialContext(env);
  }
```

Creando un cliente PTP

- Ejemplo: `QueueSend.java`

2. Configuración: Paso-1:Localizar `ConnectionFactory`
   - Encapsula información sobre la configuración de la conexión.
   - Para localizarla llamaremos a: `Context.lookup()`
   - Este método nos devolvera, en este caso un objeto `QueueConnectionFactory`

```
QueueConnectionFactory qconFactory =
(QueueConnectionFactory) ctx.lookup(JMS_FACTORY);
```

2. Configuración: Paso-2:Crear `Connection`
   - Canal abierto de comunicación entre cliente – sistema de mensajes. Se usa para crear sesiones. Gestiona trasiego.

```
QueueConnection qcon = qconFactory.createQueueConnection();
```

Creando un cliente PTP

- Ejemplo: `QueueSend.java`
  2. Configuración: Paso-3:Crear `Session`
     - Define el orden en que los mensajes se producen y consumen.
     - Puede crear múltiples productores y consumidores.
     - Tipos: transaccionales y no-transaccionales.
     - Transaccionales: Protocolo Commit/Rollback (ver módulo de Servicios Transaccionales)y no es necesario realizar acuses.
     - No-transaccionales: El tipo Session.AUTO_ACKNOWLEDGE indica que la sesión acusa recibo una vez el receptor ha recogido el mensaje. `QueueConnectionFactor`

```
QueueSession qsession = qcon.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
```

Creando un cliente PTP

- Ejemplo: `QueueSend.java`

2. Configuración: Paso-4:Localizar/crear `Destination`

   - Los objetos Queue son sub-clase de Destination.
   - Son objetos administrados por WebLogic.
   - Se obtienen a partir del contexto JNDI y del “destination name” o nombre de la cola en la configuración de WebLogic:

```
QUEUE="weblogic.examples.jms.exampleQueue";
```

```
Queue queue = (Queue) ctx.lookup(queueName);
```

   - O alternativamente:

```
Queue queue = (Queue) qsession.createQueue(queueName);
```

Creando un cliente PTP

- Ejemplo: `QueueSend.java`
  2. Configuración: Paso-5:Crear `Producers/Comsumers`
     - Instancias de sub-clases de MessageProducers{Consumers}.
     - Hay que pasarles el destino (p.e. la cola) al método createSender() o createReceiver() según queramos enviar o recibir mensajes.

```
QueueSender qsender = qsession.createSender(queue);
```

  2. Configuración: Paso-6:Crear `Message`
     - También se crean a partir de objetos de sesión

```
TextMessage msg = qsession.createTextMessage();
```

Creando un cliente PTP

- Ejemplo: `QueueSend.java`

2. Configuración: Paso-6:Crear `Message`

  - Estructura: cabecera + propiedades + cuerpo.
  - Cabecera: algunos para productores, otros para consumidores…
    - `JMSDeliveryMode: PERSISTEN,NOT_PERSISTENT` para forzar o no la persistencia de los mensajes.
    - `JMSDestination` indica el destino del mensaje
  - Propiedades: son pares estándar (atributo, valor)
  - Cuerpo: es el contenido propiamente dicho que puede ser: `BytesMessage, ObjectMessage, MapMessage, XMLMessage, TextMessage`

2. Configuración: Paso-7:Iniciar la conexión

```
qcon.start();
```

Creando un cliente PTP

- Ejemplo: `QueueSend.java`
  3. Enviar mensajes:
     - Ver método `readAndSend()` del ejemplo, el cual lee de teclado para construir un mensaje de texto.
     - Luego llama al método `send()` que acaba invocando al método `send()` del `QueueSender`

```
public void send(String message) throws JMSException {
    msg.setText(message);
    qsender.send(msg);
}
```

- En `QueueSender.send()` podemos indicar cosas como la persistencia, la prioridad (0-9) y el tiempo de vida (ms)

```
qsender.send(msg, DeliveryMode.PERSISTENT, 4, 3600000);
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Creando un cliente PTP

- Ejemplo: `QueueSend.java`

  4. Cerrar la comunicación:

     - Una vez que decidimos dejar de enviar mensajes, el cliente llama a su método close() que a su vez llama a los métodos:
       - `QueueSender.close()` que cierra el productor
       - `QueueSession.close()` que cierra la sesión
       - `QueueConnection.close()` que cierra la conexión

```
public void close()throws JMSException {
    qsender.close();
    qsession.close();
    qcon.close();
}
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Creando un cliente PTP

- Ejemplo: `QueueReceive.java`
  1. Establecer un contexto JNDI (igual que antes)
  2. Configuración de la comunicación: según sea asíncrona o síncrona.
  3. Recepción de mensajes (asíncrona):
     1. Implementar la interfaz javax.jms.MessageListener
     2. Declara como “listener”
     3. Implementar (opcional) un “listener de excepciones”
  3. Recepción de mensajes (síncrona):
     - No usar “listeners” y llamar al método receive() de la clase QueueReceiver
  4. Cerrar la comunicación

© 2006-2007 Depto. Ciencia de la Computación e IA

Creando un cliente PTP

- Ejemplo: `QueueReceive.java`

2. Configuración:

```
public void init(Context ctx, String queueName)
        throws NamingException, JMSException
  {
    qconFactory = (QueueConnectionFactory)
ctx.lookup(JMS_FACTORY);
    qcon = qconFactory.createQueueConnection();
    qsession = qcon.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
    queue = (Queue) ctx.lookup(queueName);
    qreceiver = qsession.createReceiver(queue);
    qreceiver.setMessageListener(this); // o bien: receive()
    qcon.start();
  }
```

Creando un cliente PTP

- Ejemplo: `QueueReceive.java`
  3. Recepción asíncrona: `onMessage()`
     - Manejador de eventos de mensaje.
     - Este método se activará cuando la cola reciba un mensaje
     - Puesto que el cliente debe implementar la interfaz `javax.jms.MessageListener`.
     - Es imprescindible que el consumidor ya se haya registrado como "`MessageListener`" y de hecho ha llamado al método `setMessageListener()` en `init()`
     - Opcionalmente se puede implementar un "listener de excepciones de la sesión

Creando un cliente PTP

```
public void onMessage(Message msg)
  {
    try {
      String msgText;
      if (msg instanceof TextMessage) {
        msgText = ((TextMessage)msg).getText();
      } else {
        msgText = msg.toString();
      }
      System.out.println("Message Received: "+ msgText );

      if (msgText.equalsIgnoreCase("quit")) {
        synchronized(this) {
          quit = true;
          this.notifyAll(); // Notify main thread to quit
        }
      }
    } catch (JMSException jmse) {
      jmse.printStackTrace();
    }
  }
```

Creando un cliente PTP

- Ejemplo: `QueueReceive.java`

3. Recepción síncrona:
    - Una llamada al método `QueueReceiver.receive()` bloquea indefinidamente al cliente hasta que alguien produce un mensaje para dicha cola o se cierra la aplicación.
    - Si queremos esperar solo un tiempo determinado hay que pasarle un `long` que indique el “timeout”.
    - Si no queremos esperar nada entonces llamaremos al método `receiveNoWait()` sin argumentos que consume un mensaje, si hay alguno disponible, o bien devuelve null en caso contrario.

Indice

- Motivación: Interacción con JMS
- Arquitectura JMS
- Dominios de mensajes: PTP y Pub/sub
- Creando un cliente PTP
- Creando un cliente Pub/sub

© 2006-2007 Depto. Ciencia de la Computación e IA

Creando un cliente Pub/sub

- Ejemplo: `TopicSend.java`
  1. Establecer contexto JNDI.
  2. Configurar comunicación:

```
public void init(Context ctx, String topicName)
        throws NamingException, JMSException
  {
    tconFactory = (TopicConnectionFactory)
ctx.lookup(JMS_FACTORY);
    tcon = tconFactory.createTopicConnection();
    tsession = tcon.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
    topic = (Topic) ctx.lookup(topicName);
    tpublisher = tsession.createPublisher(topic);
    msg = tsession.createTextMessage();
    tcon.start();
  }
```

Creando un cliente Pub/sub

- Ejemplo: `TopicSend.java`
  3. Enviar mensajes:

```
public void send(String message)
        throws JMSException
  {
    msg.setText(message);
    tpublisher.publish(msg);
  }
```

```
tpublisher.publish(msg, DeliveryMode.PERSISTENT, 4, 3600000);
```

Creando un cliente Pub/sub

- Ejemplo: `TopicReceive.java`
  1. Establecer contexto JNDI.
  2. Configurar comunicación:

```
public void init(Context ctx, String topicName)
        throws NamingException, JMSException
  {
    tconFactory = (TopicConnectionFactory)
ctx.lookup(JMS_FACTORY);
    tcon = tconFactory.createTopicConnection();
    tsession = tcon.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
    topic = (Topic) ctx.lookup(topicName);
    tsubscriber = tsession.createSubscriber(topic);
    tsubscriber.setMessageListener(this);
    tcon.start();
  }
```

Ejercicios

- Lanzar ejemplos en WebLogic:
  - Directorio de ejemplos:
    - $SAMPLES_HOME=$HOME/bea/weblogic92/samples
    - $SAMPLES_HOME/server/examples/src desde el que se pueden ver los javadoc de todos los ejemplos de WebLogic. Del subdirectorio examples cuelga un subdirectorio por tema: jms.

1. "Setup" de variables de entorno: (abrir terminal)
    - > cd $SAMPLES_HOME/domains/examples
    - > setExamplesEnv.cmd
    - Comprobar que el $CLASSPATH contiene el acceso a los *.jar de weblogic. Estas variables de entorno se necesitan para compilar y ejecutar correctamente los clientes JMS en esta misma terminal.

Ejercicios

- Lanzar ejemplos en WebLogic:

2. Compilar con “ant”:
   - > cd $SAMPLES_HOME/server/examples/src/examples/jms/queue
   - > Invocar “ant build” que a su vez procesa el fichero build.xml del directorio actual. El “deployment” de las clases suele consistir en ubicar los *.class en el subdirectorio correspondiente:$SAMPLES_HOME/server/examples/build/clientclasses/examples
3. Lanzar el servidor de ejemplos: (otro terminal)
   - > cd $SAMPLES_HOME/domains/wl_server
   - > setExamplesEnv.cmd
   - > startWebLogic.cmd

Ejercicios

- Lanzar ejemplos en WebLogic:
  - Cliente#1 (productor): (un terminal)

    ant run.send

  - Cliente#2 (consumidor): (desde otro terminal)

    ant run.receive

© 2006-2007 Depto. Ciencia de la Computación e IA

Ejercicios

- Comunicación síncrona
  - Modificar `TopicReceive.java` para que trabaje de forma síncrona.
  - Incorporar al package de WebLogic.
- Filtro de mensajes
  - Modificar `QueueSend.java` para que envíe mensajes con una cierta prioridad.
  - Modificar `Receive.java` para que solo consuma mensajes con una cierta prioridad mínima.
  - Incorporar al package de WebLogic.

© 2006-2007 Depto. Ciencia de la Computación e IA