Copyright © Todos los Derechos Reservados - Cibertec Perú SAC



|  |  |  |
| --- | --- | --- |
| **Programa** | **:** | Java 7.0 Architect Application Developer |
| **Curso** | **:** | Implementando una Arquitectura Orientada a Mensajes |
| **Duración** | **:** | 120 minutos |

**Laboratorio Nº C2.2**

# Objetivo

Consumir mensajes de forma asíncrona y el uso de selectores.

# Requisitos

Los siguientes elementos son necesarios para la realización del laboratorio:

1. JDK instalado (versión 1.6+)
2. Eclipse Luna instalado y configurado
3. Maven 3.0 instalado.
4. Glassfish 3.0 instalado

# Ejecución del Laboratorio

## Importar el proyecto

1. Inicie el Eclipse y seleccione el workspace wksm02. Puede crear esta carpeta y luego seleccionarlo en caso no haberlo hecho previamente.
2. Elija la opción **File –> Import…**
3. En la ventana mostrada, seleccione Maven, luego “Existing maven project” y finalmente “Next”.
4. Clic en “Browse” y seleccione $JAAD\03\_Laboratorios\Modulo02\LABM201\_P2PAsync.

## Creación de la factoría de conexiones y las colas

Ejecutaremos los siguientes pasos:

1. Para iniciar el glassfish nos debemos ubicar en la línea de comandos en la ruta:

$JAAD\02\_Programas\glassfish3\bin

1. Ejecutamos:

asadmin start-domain --verbose

1. Ingresamos a la ruta:

http://localhost:4848

1. En el menú lateral seleccionamos:

JMS Resources🡪Connection Factory

En el listado mostrado, elegimos “New”.

1. Completamos los datos siguientes y el resto lo dejamos con los datos por omisión:

Pool Name: jms/QueueCFA

Resource Type: javax.jms.ConnectionFactory

Descripción: Factoría de conexiones JMS

Le damos clic en “OK”

1. En el menú lateral seleccionamos:

JMS Resources🡪Destination Resources

En el listado mostrado, elegimos “New”.

1. Completamos los datos siguientes y el resto lo dejamos con los datos por omisión:

JNDI Name: jms/QueueIN

Physical Name: QueueIN

Resource Type: java.jms.Queue

Description: Cola de Mensajes Entrantes

Le damos clic en “OK”

1. Repetimos el paso 6 y completamos los datos siguientes y el resto lo dejamos con los datos por omisión:

JNDI Name: jms/QueueOUT

Physical Name: QueueOUT

Resource Type: java.jms.Queue

Description: Cola de Mensajes Salientes

Le damos clic en “OK”

## Recepción asíncrona del mensaje

Empezaremos realizando el envío de la forma como ya se venía haciendo:

Creamos la clase JMSSenderReceiver:

|  |
| --- |
| package edu.cibertec.jaad.jms;  public class JMSSenderReceiver {  private static final int WAITING\_MSG = 60;  private static final String JMS\_QUEUE\_OUT = "jms/QueueOUT";  private static final String JMS\_QUEUE\_IN = "jms/QueueIN";  private static final String JMS\_CONNFACT = "jms/QueueCFA";  private static final Logger LOG = Logger.getLogger(JMSSenderReceiver.class);  public static void main(String[] args) {  try {  Context ctx = new InitialContext();  ConnectionFactory factory = (ConnectionFactory)ctx  .lookup(JMS\_CONNFACT);  Connection connection = factory.createConnection();  Session session = connection  .createSession(false, Session.AUTO\_ACKNOWLEDGE);    Destination colaIN = (Destination)ctx.lookup(JMS\_QUEUE\_IN);  connection.start();    //Enviando el mensaje  MessageProducer producer = session.createProducer(colaIN);  MapMessage msgReq = session.createMapMessage();  msgReq.setString("OPERACION", "Recarga");  msgReq.setDouble("MONTO", 35.0);  producer.send(msgReq);    producer.close();  connection.close();  System.exit(0);  } catch (Exception ex) {  LOG.error("Error al enviar/recibir el mensaje", ex);  }  }  } |

Código 1 – Clase JMSSenderReceive v.1

Creamos el Consumidor

Junto con el proyecto se acompaña un archivo jndi.properties. Este archivo es el que tiene los datos de la ubicación del servidor:

|  |
| --- |
| public class JMSReceiverSender implements MessageListener{  private static final String JMS\_QUEUE\_IN = "jms/QueueIN";  private static final String JMS\_QUEUE\_CF = "jms/QueueCFA";  private static final Logger LOG = Logger  .getLogger(JMSReceiverSender.class);    private Session session;    public void start() {  try {  Context ctx = new InitialContext();  ConnectionFactory factory = (ConnectionFactory)ctx  .lookup(JMS\_QUEUE\_CF);  Connection connection = factory.createConnection();  session = connection  .createSession(false, Session.AUTO\_ACKNOWLEDGE);  Destination colaIN = (Destination)ctx.lookup(JMS\_QUEUE\_IN);  connection.start();  MessageConsumer consumer = session  .createConsumer(colaIN);  consumer.setMessageListener(this);  LOG.info("Esperando por mensaje...");  } catch (Exception ex) {  LOG.error("Error al iniciar el lector", ex);  }  }    @Override  public void onMessage(Message message) {  try {  MapMessage msg = (MapMessage)message;  LOG.info("Recibido=[" + msg + "]");  LOG.info("Operacion=" + msg.getString("OPERACION"));  LOG.info("Monto=" + msg.getDouble("MONTO"));  } catch (Exception ex) {  LOG.error("Error al recibir el mensaje", ex);  }  }  public static void main(String[] args) {  JMSReceiverSender rs = new JMSReceiverSender();  rs.start();  }  } |

Código 2 – Clase JMSReceiverSender v.1

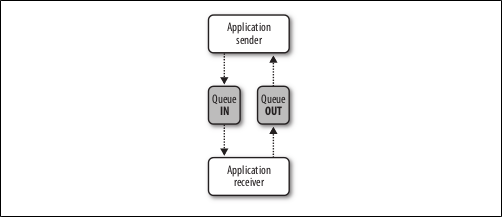
Ejecutamos el archivo de la forma habitual y se deberá mostrar el consumo del mensaje de forma asíncrona.

Discuta en clase el código utilizado.

## Creación del filtro y respuesta por otra Cola

Es usual que en los sistemas se establezca una cola de entrada y otra de salida de forma de mantener en un sentido los envíos. También es usual agregar un filtro de mensajes cuando se espera la respuesta de un mensaje.

Los mensajes fluirían del siguiente modo:

****

Para ello se creó inicialmente dos colas, y seguidamente le agregaremos al código la capacidad de respuesta al mensaje, pero el nombre de la cola será enviado en el mensaje.

Finalmente agregaremos filtro para la atención exclusiva de tipos de mensaje, así como la espera de la respuesta por la cual se consultó.

Modificamos la clase JMSSenderReceiver, deberá quedar del siguiente modo:

|  |
| --- |
| package edu.cibertec.jaad.jms;  public class JMSSenderReceiver {  private static final int WAITING\_MSG = 60;  private static final String JMS\_QUEUE\_OUT = "jms/QueueOUT";  private static final String JMS\_QUEUE\_IN = "jms/QueueIN";  private static final String JMS\_CONNFACT = "jms/QueueCFA";  private static final Logger LOG = Logger.getLogger(JMSSenderReceiver.class);  public static void main(String[] args) {  try {  Context ctx = new InitialContext();  ConnectionFactory factory = (ConnectionFactory)ctx  .lookup(JMS\_CONNFACT);  Connection connection = factory.createConnection();  Session session = connection  .createSession(false, Session.AUTO\_ACKNOWLEDGE);    Destination colaIN = (Destination)ctx.lookup(JMS\_QUEUE\_IN);  Destination colaOUT = (Destination)ctx.lookup(JMS\_QUEUE\_OUT);  connection.start();    //Enviando el mensaje  MessageProducer producer = session.createProducer(colaIN);  MapMessage msgReq = session.createMapMessage();  msgReq.setString("OPERACION", "Recarga");  msgReq.setDouble("MONTO", 35.0);  String id = UUID.randomUUID().toString();  msgReq.setJMSCorrelationID(id);  //Cola destino por la cual se debe responder  msgReq.setJMSReplyTo(colaOUT);  msgReq.setStringProperty("OPERACION", "Recarga");  producer.send(msgReq);    //Recibiendo la respuesta  String selector = "JMSCorrelationID = '" + id + "'";  MessageConsumer consumer = session  .createConsumer(colaOUT, selector);  LOG.info("Esperando " + WAITING\_MSG + "seg.");  TextMessage msgResp = (TextMessage)consumer  .receive(WAITING\_MSG\*1000);  String result = msgResp == null ?  "SIN\_RESPUESTA" : msgResp.getText();  LOG.info("===>" + result);  LOG.info("MSG=" + msgResp);    producer.close();  consumer.close();  connection.close();  System.exit(0);  } catch (Exception ex) {  LOG.error("Error al enviar/recibir el mensaje", ex);  }  }  } |

Código 3 – JMSSenderReceiver v.2

Observe que ahora se han creado dos colas, se agregó un correlationID, y se agregó un selector para que se pueda filtrar los mensajes.

El código del JMSReceiverSender, quedará del siguiente modo:

|  |
| --- |
| package edu.cibertec.jaad.jms;  public class JMSReceiverSender implements MessageListener{  private static final String JMS\_QUEUE\_IN = "jms/QueueIN";  private static final String JMS\_QUEUE\_CF = "jms/QueueCFA";  private static final Logger LOG = Logger  .getLogger(JMSReceiverSender.class);    private Session session;    public void start() {  try {  Context ctx = new InitialContext();  ConnectionFactory factory = (ConnectionFactory)ctx  .lookup(JMS\_QUEUE\_CF);  Connection connection = factory.createConnection();  session = connection  .createSession(false, Session.AUTO\_ACKNOWLEDGE);  Destination colaIN = (Destination)ctx.lookup(JMS\_QUEUE\_IN);  connection.start();  String selector = "OPERACION = 'Recarga'";  MessageConsumer consumer = session  .createConsumer(colaIN, selector);  consumer.setMessageListener(this);  LOG.info("Esperando por mensaje...");  } catch (Exception ex) {  LOG.error("Error al iniciar el lector", ex);  }  }    @Override  public void onMessage(Message message) {  try {  MapMessage msg = (MapMessage)message;  LOG.info("Recibido=[" + msg + "]");  LOG.info("Operacion=" + msg.getString("OPERACION"));  LOG.info("Monto=" + msg.getDouble("MONTO"));  TextMessage msgResp = session.createTextMessage("OK");  msgResp.setJMSCorrelationID(message.getJMSCorrelationID());    MessageProducer producer = session  .createProducer(msg.getJMSReplyTo());  producer.send(msgResp);  producer.close();  LOG.info("Mensaje enviado=[" + msgResp + "]");  } catch (Exception ex) {  LOG.error("Error al recibir el mensaje", ex);  }  }  public static void main(String[] args) {  JMSReceiverSender rs = new JMSReceiverSender();  rs.start();  }  } |

Código 4 – JMSReceiverSender v.2

En este caso le hemos agregado un selector para que filtre sólo OPERACIÓN=’Recarga’, además se responde al mensaje con el mismo correlationID debido a que el que espera la respuesta filtrará los mensajes entrantes por dicho valor. Finalmente se responde el mensaje, obteniendo el nombre de la cola a partir del mensaje recibido: getJMSReplyTo.

Ejecute el código de la forma habitual.

Revise el funcionamiento y discútalo en clase.