



Práctica 2: Publish/Subscribe con Java Beans(EJB) y java Message Service(JMS)

SISTEMAS DISTRIBUIDOS

Víctor Pérez Piqueras

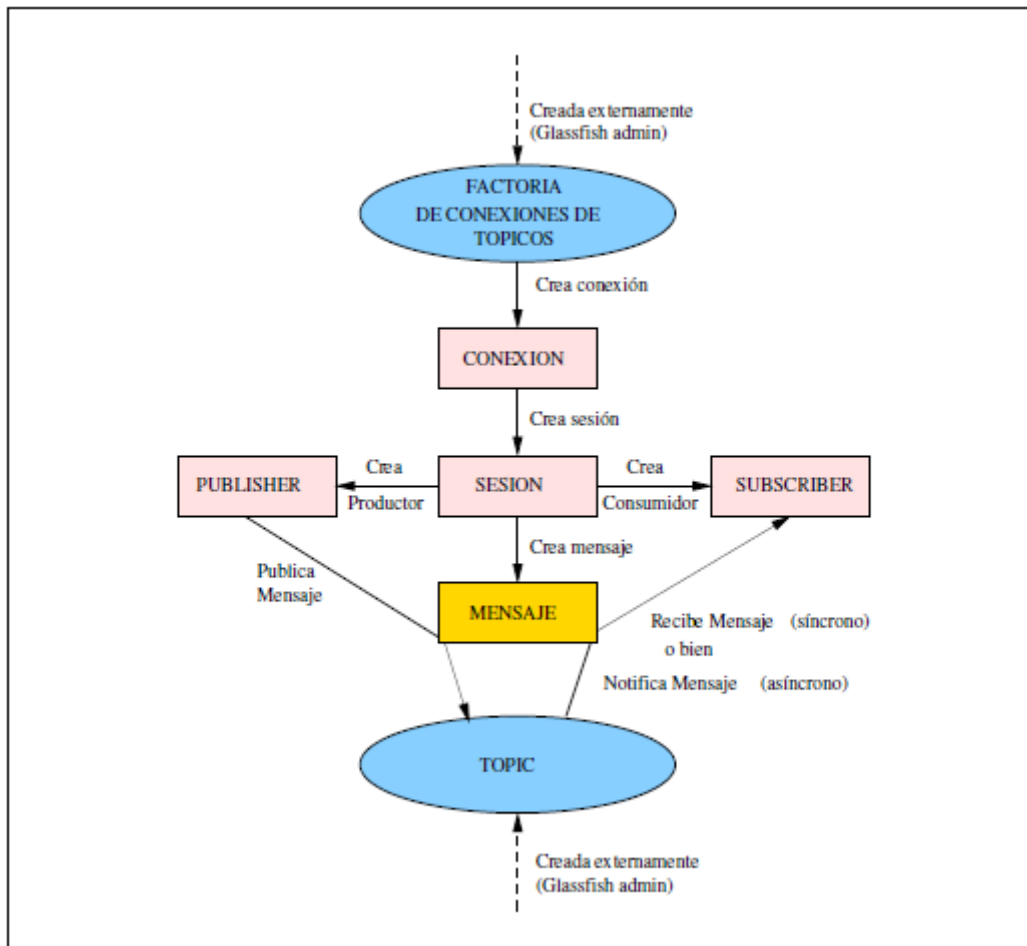
José Ramón Martínez Riveiro

Índice

| | |
|---|----|
| PROPUESTA | 3 |
| IMPLEMENTACIÓN..... | 5 |
| Paso 1: | 5 |
| Paso 2: | 6 |
| ProyectoEJB-ejb: | 6 |
| ProyectoEJB-war: | 9 |
| RESULTADOS | 10 |
| IMPLEMENTACION DEL CLIENTE REMOTO | 11 |
| RESULTADOS DEL CLIENTE REMOTO..... | 12 |

PROPUESTA

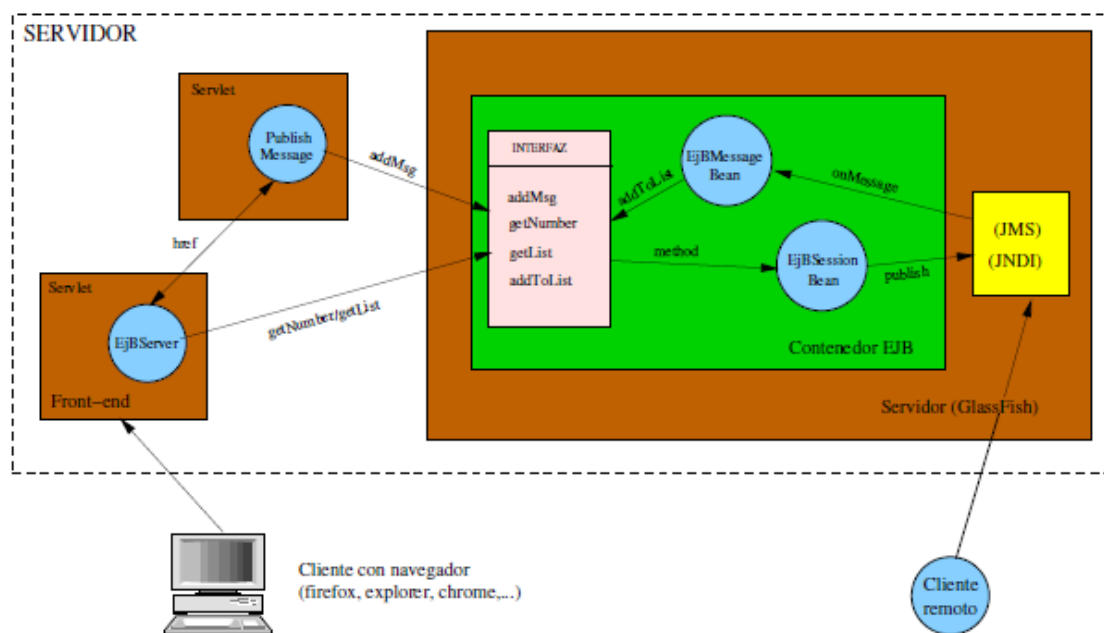
no o varios productores de mensajes (Publishers) publican eventos (mensajes), que los receptores o consumidores (Subscribers) reciben de manera asíncrona, mediante notificaciones. El soporte JMS de este paradigma Publish/Subscribe se basa en los denominados topics, similares a las colas de mensajes, a los que se pueden suscribir los clientes y recibir los mensajes destinados a un tópico en concreto.



En la práctica implementaremos y utilizaremos los siguientes elementos estructurales:

1. Clientes remotos que pueden publicar de forma directa noticias o recibir de manera asíncrona las noticias que hayan sido publicadas.
2. Del lado del servidor tendremos los recursos JMS (factoría de conexiones de tópicos y tópico concreto), un módulo web como front-end, que constará de dos servlets (EjbServer y PublishMessage), un Singleton Session Bean (EjbSessionBean), que implementará la lógica de negocio del servidor (publicar mensajes y gestionar un histórico de los mismos) y un Message-Driven Bean (EjbMessageBean), que permitirá al servidor recibir las notificaciones de los mensajes publicados, tanto de los que se publican usando el módulo web como los que se publiquen directamente de forma remota.

La figura siguiente ilustra la arquitectura de esta aplicación:



Los elementos principales son:

- **EjbSessionBean:** es un Singleton Session Bean, del cual sólo se instancia una copia, que es compartida por todos los clientes. Es un tipo de bean que mantiene su estado entre invocaciones, aunque no mantendrá su estado en caso de caída o reinicio del servidor. Nos aprovechamos de este hecho para implementar el bean, de modo que mantendrá una lista de los mensajes publicados y un contador de los mismos. El histórico de los mensajes se guardará en un fichero de texto `mensajes.txt`, que deberá ser creado en caso de no existir (primera ejecución). De esta forma, al inicializar el bean, este obtendrá el histórico de mensajes desde el fichero, lo cual nos permite mantener el servicio incluso si el servidor es reiniciado.
- **EjbMessageBean:** es un Message-Driven Bean, que permite al servidor recibir todos los mensajes publicados (directamente por clientes remotos o vía **EjbSessionBean**), y darlos de alta en el histórico.
- **EjbServer:** es el front-end de la aplicación, permite a los clientes conectarse con un navegador, les muestra los mensajes actuales y el número de ellos, y pide si desean publicar un nuevo mensaje. **PublishMessage** es un servlet que se ejecuta para publicar un mensaje. Se limita a pedir el texto del mensaje e invocar el método **addMsg** del contenedor EJB para publicarlo. Pueden publicarse varios mensajes, o volver al servlet **EJBServer**.

IMPLEMENTACIÓN

Paso 1:

Creamos con GlassFish la factoría de conexiones de tópicos y el tópico utilizando estos comandos:

```
cd c:\Program Files\GlassFish-4.1.1\bin
asadmin
create-jms-resource --restype javax.jms.TopicConnectionFactory \
  --description "FactoriaConexiones" jms/FactoriaConexiones
create-jms-resource --restype javax.jms.Topic --property Name="Noticias" jms/Noticias
```

La factoría de tópicos (jms/FactoriaConexiones) será de tipo javax.jms.TopicConnectionFactory

El topic (jms/Noticias) será de tipo javax.jms.Topic

Así se ven la factoría y el tópico creados en Admin-Console de glassfish:

JMS Connection Factories

Java Message Service (JMS) connection factories are objects that allow an application to create other JMS objects programmatically. Click New to create a new connection factory. Click the name of a connection factory to modify its properties.

| Connection Factories (2) | | | | |
|--|-------------------------------|---------------------------------------|-------------------------------------|----------------------------------|
| <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="button" value="New"/> <input type="button" value="Delete"/> <input type="button" value="Enable"/> <input type="button" value="Disable"/> | | | | |
| Select | JNDI Name | Logical JNDI Name | Enabled | Resource Type |
| <input type="checkbox"/> | jms/_defaultConnectionFactory | java:comp/DefaultJMSConnectionFactory | <input checked="" type="checkbox"/> | javax.jms.ConnectionFactory |
| <input type="checkbox"/> | jms/FactoriaConexiones | | <input checked="" type="checkbox"/> | javax.jms.TopicConnectionFactory |

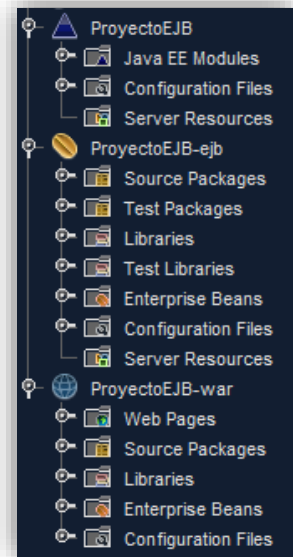
JMS Destination Resources

JMS destinations serve as the repositories for messages. Click New to create a new destination resource. Click the name of a destination resource to modify its properties.

| Destination Resources (1) | | | |
|--|--------------|-------------------------------------|-----------------|
| <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="button" value="New"/> <input type="button" value="Delete"/> <input type="button" value="Enable"/> <input type="button" value="Disable"/> | | | |
| Select | JNDI Name | Enabled | Resource Type |
| <input type="checkbox"/> | jms/Noticias | <input checked="" type="checkbox"/> | javax.jms.Topic |

Paso 2:

Crear un nuevo proyecto ProyectoEJB, de tipo Java EE, Enterprise Application. Dejar marcadas las opciones de crear EJBModule y Web Application Module, que se corresponden con el contenedor EJB, el cual intercepta las invocaciones a los EJB que encierra, y el servidor web, respectivamente.



ProyectoEJB-ejb:

EjbSessionBean:

- Creamos un SessionBean llamada *EjbSessionBean*. Esta clase accederá a las factorías de conexiones de tópicos y al tópico creado mediante una inyección de código, con anotaciones Java a nivel de clase:

```
@Resource(mappedName="jms/FactoriaConexiones")
private TopicConnectionFactory tcf;
@Resource(mappedName="jms/Noticias")
private Topic t;
```

Esta clase tendrá **2 atributos**: un contador de mensajes y una lista con los mensajes, y **4 métodos implementados** de la interfaz *EjbSessionLocal* los cuales son:

- **Método addMsg:** Es el método usado por EJBSessionBean para publicar mensajes. Creará la conexión mediante el uso de los métodos createTopicConnection, createTopicSession y createPublisher, así como la sesión, y un Publisher. Ya con todo ello creado, iniciaremos la conexión, crearemos el mensaje a enviar y lo asignaremos al mensaje creado(msg.setText(m)), siendo m el parámetro de tipo string pasado al método addMsg. A continuación publicaremos el mensaje, una vez realizada dicha publicación ya podemos cerrar el Publisher.

```
@Override
public void addMsg(String m) {
    try
    {
        conexion = tcf.createTopicConnection();
        sesion = conexion.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        publisher = sesion.createPublisher(t);
        conexion.start();
        msg = sesion.createTextMessage();
        msg.setText(m);
        publisher.publish(msg);
        publisher.close();
    } catch (JMSException ex)
    {
        System.out.println("Se produjo un error");
    }
}
```

- **Método getNumber:** Devuelve un contador con los mensajes existentes.

```
@Override
public int getNumber() {
    return count_msg;
}
```

- **Método getList:** devuelve la lista de mensajes existentes actualmente.

```
@Override
public LinkedList<String> getList() {
    return list_msg;
}
```

- **Método addToList:** añade un mensaje pasado por parámetro a la lista de mensajes, e incrementa el contador de mensajes.

```

@Override
public void addToList(String m) {
    list_msg.add(m);
    count_msg++;
}

```

Esta clase además de lo anteriormente mencionado poseerá un **código que inicialización** cargará el histórico de mensajes desde un *fichero.txt*, ejecutándose este método al iniciarse el bean:

```

@PostConstruct void inicializacion() {

    count_msg=0;
    list_msg = new LinkedList<String>();
    FileReader fr = null;
    BufferedReader br=null;

    try {
        File archivo = new File("C:\\\\fichero.txt");
        fr = new FileReader(archivo);
        br = new BufferedReader(fr);
        String linea=null;
        try {
            linea = br.readLine();
            while(linea != null){
                addToList(linea);
                linea=br.readLine();
            }
            fr.close();
        } catch (IOException ex) {
            Logger.getLogger(EjbSessionBean.class.getName()).log(Level.SEVERE, null, ex);
        }
    } catch (FileNotFoundException ex) {
        Logger.getLogger(EjbSessionBean.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

EjbMessageBean:

Este Bean se comunicará con el SessionBean creado anteriormente, por tanto, debemos inyectar código EJB para poder acceder al bean:

```

@EJB
private EjbSessionBeanLocal ejbSessionBean;

```

- El método *onMessage* nos permitirá abrir el fichero mensajes.txt en modo “append”, escribir el mensaje recibido en este e invocar al método *addToList* del SessionBean para darlo de alta:


```

@Override
public void onMessage(Message message) {

    FileWriter fw = null;
    try{
        fw = new FileWriter("C:\\\\fichero.txt", true); //modo append
        TextMessage textMessage = (TextMessage) message; //transformar a string
        String me = textMessage.getText();

        PrintWriter msg = new PrintWriter(fw);
        msg.print(me);
        msg.println();
        ejbSessionBean.addToList(me); //añadirlo a la lista
        fw.close();
    } catch (IOException ex)
    {
        System.out.println("Se produjo un error");
    } catch (JMSException ex) {
        System.out.println("Se produjo un error JMSException");
    }
}

```

ProyectoEJB-war:

El módulo web tendrá dos servlets:

EJBServer:

Injectamos código, como en el paso anterior, para poder invocar al SessionBean. Tras esto modificamos el código HTML que generará el servlet, para incluir un enlace al servlet.

```

out.println("<h1>Servlet EJBServer at " + request.getContextPath() + "</h1>");
out.println("<a href='PublishMessage'> Publicar noticia</a>");

```

Y otras modificaciones para poder mostrar los mensajes publicados y su número total:

```

out.println("<h1>List: "+ejbSessionBean.getList()+"</h1>"); //imprimir lista y contador
out.println("<h1>Number: "+ejbSessionBean.getNumber()+"</h1>");

```

PublishMessage:

Injectamos código como en el paso anterior e incluimos el siguiente código para recoger el valor tecleado como mensaje:

```

String m=request.getParameter("MESSAGE");
if (m != null){
    ejbSessionBean.addMsg(m);
}

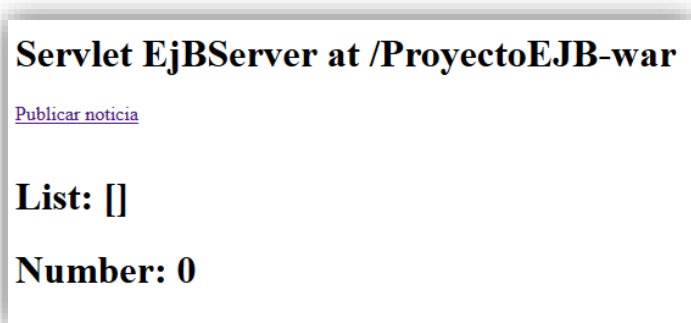
```

y procedemos a modificar el código HTML:

```
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet PublishMessage</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Servlet PublishMessage at " + request.getContextPath() + "</h1>");
out.println("</body>");
out.println("</html>");
out.println("<form>");
out.println("NOTICIA: <input type='text' name='MESSAGE'><br/>");
out.println("<input type='submit' value='Submit'><br/>");
out.println("<br><br><br>");
out.println("<a href='EJBServer'> Volver a pagina anterior</a>");
out.println("</form>");
```

RESULTADOS

Con todo esto configurado, hacemos *deploy* del servidor y lo ejecutamos. Con lo que, desde un navegador podemos ver esta interfaz:

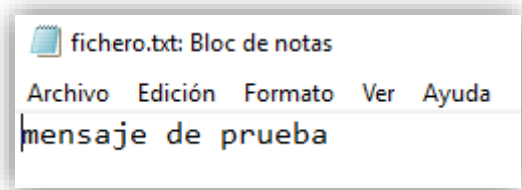


The screenshot shows a web browser displaying the page titled "Servlet EjBServer at /ProyectoEJB-war". Below the title is a link "Publicar noticia". Underneath, there is a section labeled "List: []" and "Number: 0".

Si queremos publicar un mensaje solo basta con darle a publicar noticia y escribirlo, y aparecerá en el fichero.txt:



The screenshot shows a web browser displaying the page titled "Servlet PublishMessage at /ProyectoEJB-war". It features a form with a label "NOTICIA:" followed by a text input field containing "mensaje de prueba". Below the input field is a "Submit" button. At the bottom, there is a link "Volver a pagina anterior".



The screenshot shows a text editor window titled "fichero.txt: Bloc de notas". The menu bar includes "Archivo", "Edición", "Formato", "Ver", and "Ayuda". The main text area contains the phrase "mensaje de prueba".

Servlet EJBServer at /ProyectoEJB-war

[Publicar noticia](#)

List: [mensaje de prueba]

Number: 1

IMPLEMENTACION DEL CLIENTE REMOTO

Crearemos un cliente remoto que se conectará al *topic* “jms/Noticias” de forma remota, y le permitirá dar de alta noticias nuevas, a la vez que se suscribirá para poder recibir noticias publicadas, que se mostrarán por pantalla.

Utilizamos las clases *TopicConnectionFactory*, *TopicConnection*, *TopicSession*, *TopicPublisher* y *TopicSubscriber*.

Desde el objeto *TopicSubscriber* ejecutaremos el método *setMessageListener* que creará un objeto de la clase *MsgListener* que nos permite estar a la escucha de los mensajes entrantes.

```
public class Main {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) throws JMSException, NamingException, InterruptedException {  
  
        InitialContext iniCtx = new InitialContext();  
        Object tmp = iniCtx.lookup("jms/FactoriaConexiones");  
        TopicConnectionFactory tcf=(TopicConnectionFactory)tmp;  
        Topic t=(Topic)iniCtx.lookup("jms/Noticias");  
  
        TopicConnection connection = tcf.createTopicConnection();  
        TopicSession session = connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);  
        TopicPublisher publisher = session.createPublisher(t);  
        TopicSubscriber subscriber = session.createSubscriber(t);  
  
        subscriber.setMessageListener(new MsgListener());  
        connection.start();  
  
        TextMessage mensaje=session.createTextMessage();  
        String noticia=null;
```

También implementamos el método *onMessage*, que mostrará los mensajes que van siendo publicados. Siendo estas recepciones asíncronas. Mientras, pedirá mensajes por teclado indefinidamente:

```
private static class MsgListener implements MessageListener {

    public MsgListener() {
    }

    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage)message;

        try {
            System.out.println("Recibo: " + textMessage.getText());
        }
        catch (JMSException ex) {
            System.err.println("Se produjo un error");
        }
    }
}
```

Para probar su funcionamiento ejecutamos el *main* del código del cliente, y escribimos un mensaje. Este bucle del *main* muestra el proceso de lectura y escritura por pantalla para el cliente:

```
while(noticia!="") {
    System.out.print("Escribir noticia: ");
    Scanner scan=new Scanner(System.in);
    noticia=scan.nextLine();
    mensaje.setText("Cliente: "+noticia);
    publisher.publish(mensaje);
    Thread.sleep(100);
    publisher.close();
}
```

RESULTADOS DEL CLIENTE REMOTO

El resultado por consola al introducir un mensaje es este:

```
Output
Java DB Database Process x GlassFish Server 4.1.1 x Lab20istClient (run-single) x
OCT 11, 2018 4:03:23 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
OCT 11, 2018 4:03:25 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_R1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
OCT 11, 2018 4:03:25 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_R1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
OCT 11, 2018 4:03:25 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_R1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Escribir noticia:
hola
Recibo: Noticia: hola
Escribir noticia:
```

También podemos leer el mensaje desde el navegador:

Servlet EjBServer at /ProyectoEJB-war

[Publicar noticia](#)

List: [mensaje de prueba, Noticia: hola]

Number: 2

Del otro lado, si escribimos un mensaje en el navegador, podremos verlo por consola en el cliente:

Servlet PublishMessage at /ProyectoEJB-war

NOTICIA:

[Volver a pagina anterior](#)

```
Output
Java DB Database Process x GlassFish Server 4.1.1 x Lab2DistClient (run-single) x
INFO: HV000001: Hibernate validator 5.1.2.Final
oct 14, 2018 2:29:44 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_Ra1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build
oct 14, 2018 2:29:44 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_Ra1101: GlassFish MQ JMS Resource Adapter starting: broker is REM
oct 14, 2018 2:29:44 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_Ra1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Escribir noticia: hola server

Recibo: Cliente: hola server
Escribir noticia:
Recibo: hola cliente
```