

## Spring JMS con GlassFish

**Autor:** José Díaz

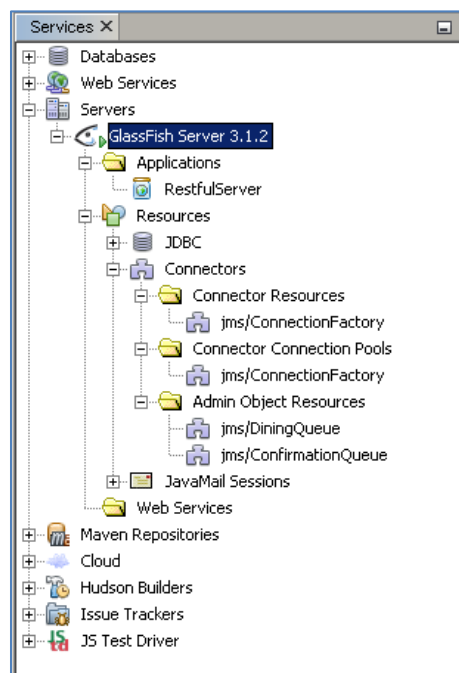
**Código Fuente:** <https://github.com/joedayz/java-samples/tree/master/SpringJMS>

Estimados alumnos, en clase vimos como trabajar con Spring JMS y ActimeMQ embebido y/o ActiveMQ instalado y corriendo en su pc local.

He modificado el ejemplo para que trabaje con GlassFish.

## Configurando La Infraestructura en GlassFish

Vamos a usar el GlassFish que viene con el Netbeans. Hay que asegurarnos en primer lugar que este iniciado.



Luego clic derecho a GlassFish y escoger la opción *View Domain Admin Console*.

Levantara la aplicación web de administración de GlassFish y ahí ud. podrá crear el *connectionFactory* y las colas.

Firefox | Examen Fina y Rec... | JMS con GlassFish ... | The Play Framework... | News and Announc... | Spring Champions a... | Apuntes de Java: ... | JUG Guadalajara: S... | Ofertas y... por...

localhost:4848/common/index.jsf

Página Inicial | Acerca de...

Usuario: admin | Dominio: domain1 | Servidor: localhost

## GlassFish™ Server Open Source Edition

Dominio

- servidor (Servidor de Administrac
- Clusters
- Instancias Independientes
- Nodos
- Aplicaciones
- Módulos de Ciclo de Vida
- Datos de Supervisión
- Recursos
  - JDBC
  - Conectores
  - Configuraciones de Adaptador
  - Recursos de JMS
    - Fábricas de Conexiones
    - Recursos de Destino
      - jms/ConfirmationQueue
      - jms/DiningQueue
    - Sesiones JavaMail
    - JNDI
  - Configuraciones
    - default-config
    - server-config
  - Herramienta de Actualización

### Editar Recurso de Destino JMS

Al editar un recurso de destino JMS (Java Message Service), también se modifica el recurso de objeto de administración asociado.

[Cargar Valores por Defecto](#)

**Nombre JNDI:** jms/ConfirmationQueue

**Nombre de Destino Físico \***   
Nombre de destino en el broker de Message Queue. Si el destino no existe, se creará automáticamente cuando

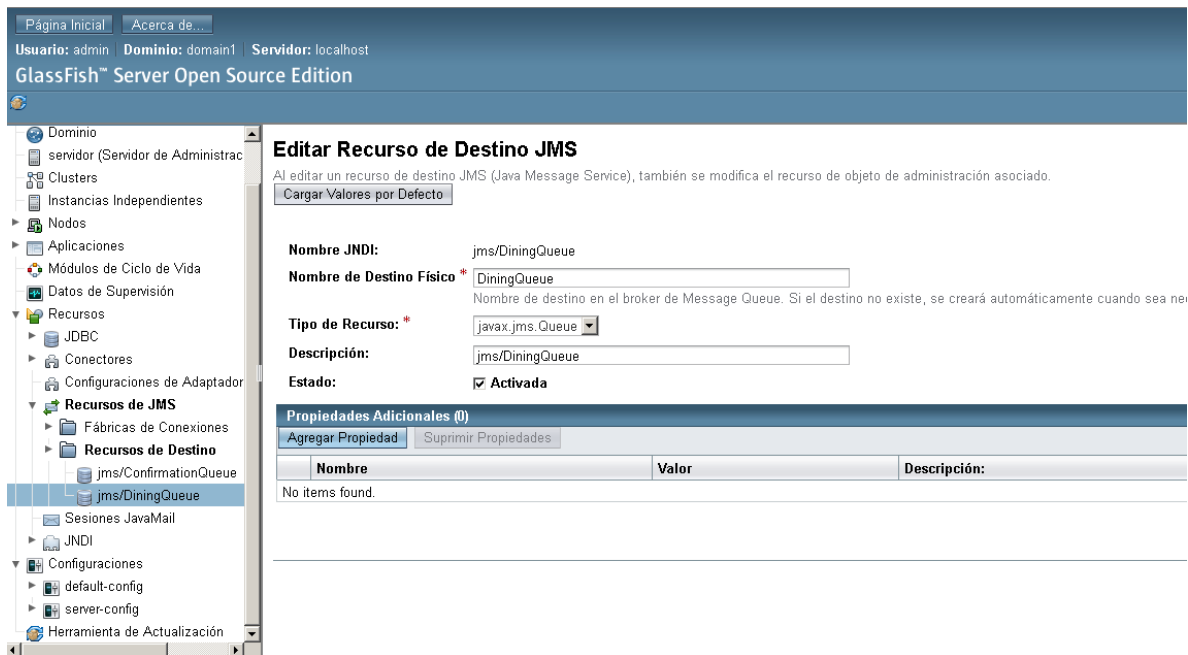
**Tipo de Recurso: \***

**Descripción:**

**Estado:** ☒ **Activada**

#### Propiedades Adicionales (0)

Nombre	Valor	Descripción:
No items found.		



## Configurando La Infraestructura y Spring JMS en nuestro proyecto Web

En este caso si necesitaremos un bean adicional para conectarnos a GlassFish

```
<bean id="jndiTemplate" class="org.springframework.jndi.JndiTemplate">
  <property name="environment">
    <props>
      <prop key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory
    </prop>
      <prop key="java.naming.factory.url.pkgs">com.sun.enterprise.naming</prop>
      <prop key="java.naming.factory.state">com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl
    </prop>
      <prop key="org.omg.CORBA.ORBInitialHost">localhost</prop>
      <prop key="org.omg.CORBA.ORBInitialPort">3700</prop>
    </props>
  </property>
</bean>
```

Definimos a continuación el bean *connectionFactory* haciendo referencia al nombre JNDI (**jms/ConnectionFactory**) que se definió a la hora de crearlo en el Admin Console del GlassFish:

```

<bean id="connectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiTemplate" ref="jndiTemplate" />
    <property name="jndiName" value="jms/ConnectionFactory" />
</bean>

```

De la misma manera para las colas:

```

<bean id="diningQueue" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiTemplate" ref="jndiTemplate" />
    <property name="jndiName" value="jms/DiningQueue" />
</bean>

<bean id="confirmationQueue" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiTemplate" ref="jndiTemplate" />
    <property name="jndiName" value="jms/ConfirmationQueue" />
</bean>

```

El primer listener que leerá de la cola jms/DiningQueue será el siguiente:

```

<jms:listener-container connection-factory="connectionFactory"
    destination-resolver="jmsDestinationResolver">
    <jms:listener ref="rewardNetwork" method="rewardAccountFor"
        destination="jms/DiningQueue" response-destination="jms/ConfirmationQueue" />
</jms:listener-container>

```

Como se aprecia, hay un listener que hace referencia a un bean denominado **rewardNetwork** y que al leer de la cola dado por el atributo **destination="jms/DiningQueue"** envía el objeto leído al método **rewardAccountFor** que le debe si o si pertenecer a **rewardNetwork**. Finalmente la respuesta o retorno de dicho método se colocará en **response-destination="jms/ConfirmationQueue"**.

```

    */
    @Service("rewardNetwork")
    public class RewardNetworkImpl implements RewardNetwork {

        private AccountRepository accountRepository;

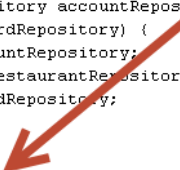
        private RestaurantRepository restaurantRepository;

        private RewardRepository rewardRepository;

        /**
         * Creates a new reward network.
         * @param accountRepository the repository for loading accounts to reward
         * @param restaurantRepository the repository for loading restaurants that determine how much to reward
         * @param rewardRepository the repository for recording a record of successful reward transactions
         */
        @Autowired
        public RewardNetworkImpl(AccountRepository accountRepository, RestaurantRepository restaurantRepository,
                                RewardRepository rewardRepository) {
            this.accountRepository = accountRepository;
            this.restaurantRepository = restaurantRepository;
            this.rewardRepository = rewardRepository;
        }

        @Transactional
        public RewardConfirmation rewardAccountFor(Dining dining) {
            Account account = accountRepository.findByCreditCard(dining.getCreditCardNumber());
            Restaurant restaurant = restaurantRepository.findByMerchantNumber(dining.getMerchantNumber());
            MonetaryAmount amount = restaurant.calculateBenefitFor(account, dining);
            AccountContribution contribution = account.makeContribution(amount);
        }
    }

```



Cómo se observa en la figura el objeto `RewardConfirmation` que se retorne será puesto en la cola que está asociado al nombre JNDI `jms/ConfirmationQueue`.

Finalmente el último listener que lee de esta última cola es **confirmationLogger**.

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jms="http://www.springframework.org/schema/jms"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
           http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-3.0.xsd">

    <bean id="diningBatchProcessor" class="rewards.jms.client.JmsDiningBatchProcessor">
        <property name="jmsTemplate" ref="jmsTemplate"/>
    </bean>

    <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="connectionFactory"/>
        <property name="defaultDestination" ref="diningQueue"/>
    </bean>

    <bean id="confirmationLogger" class="rewards.jms.client.RewardConfirmationLogger"/>

    <jms:listener-container connection-factory="connectionFactory"
                          destination-resolver="jmsDestinationResolver">
        <jms:listener ref="confirmationLogger" method="log" destination="jms/ConfirmationQueue"/>
    </jms:listener-container>

</beans>

```

De la misma manera lee de `jms/ConfirmationQueue` y lo que saque de ahí lo manda al método `log` de `confirmationLogger`.

## Test para validar que está funcionando

Enviamos 5 cenas y esperamos 5 confirmaciones.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package rewards.jms.client;

import java.util.ArrayList;
import java.util.List;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.AbstractTransactionalJUnit4SpringContextTests;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import static org.junit.Assert.*;

import rewards.Dining;

/**
 * Tests the Dining batch processor
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath:rewards/system-test-config.xml",
                                   "classpath:rewards/jms/client/client-config.xml",
                                   "classpath:rewards/jms/jms-rewards-config.xml",
                                   "classpath:rewards/jms/jms-infrastructure-config.xml"})
public class DiningBatchProcessorTests extends AbstractTransactionalJUnit4SpringContextTests {

    @Autowired
    private DiningBatchProcessor diningBatchProcessor;

    @Autowired
    private RewardConfirmationLogger confirmationLogger;

    @Test
    public void testBatch() throws Exception {
        Dining dining1 = Dining.createDining("80.93", "1234123412341234", "1234567890");
        Dining dining2 = Dining.createDining("56.12", "1234123412341234", "1234567890");
    }
}
```

```
Dining dining3 = Dining.createDining("32.64", "1234123412341234", "1234567890");
Dining dining4 = Dining.createDining("77.05", "1234123412341234", "1234567890");
Dining dining5 = Dining.createDining("94.50", "1234123412341234", "1234567890");
```

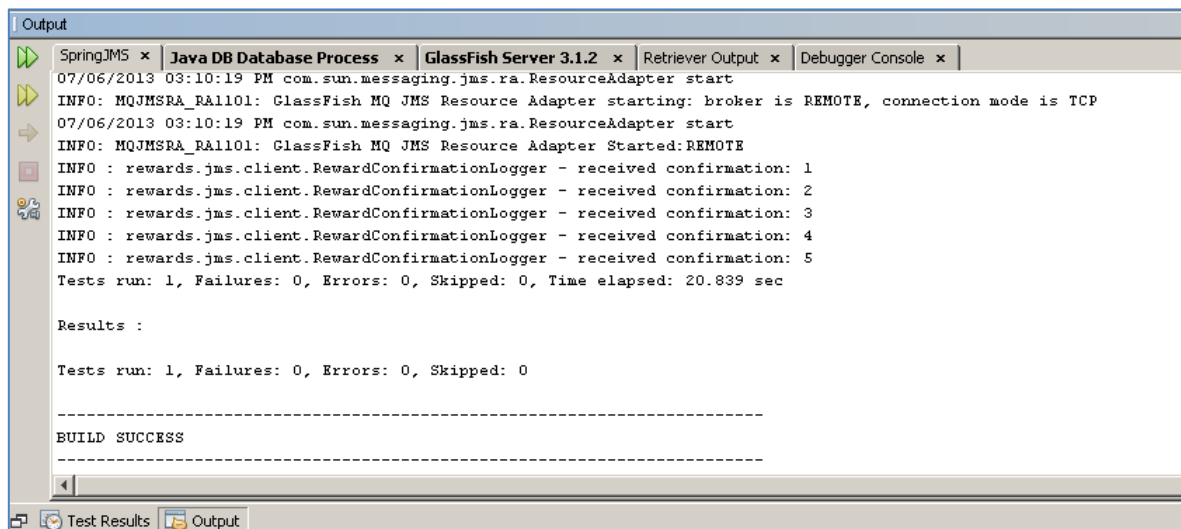
```
List<Dining> batch = new ArrayList<Dining>();
batch.add(dining1);
batch.add(dining2);
batch.add(dining3);
batch.add(dining4);
batch.add(dining5);
```

```
diningBatchProcessor.processBatch(batch); //ENVIAMOS LAS 5 CENAS
waitForBatch(batch.size(), 5000);
```

```
//EL NUMERO DE CENAS DEBE SER IGUAL AL NUMERO DE CONFIRMACIONES.
assertEquals(batch.size(), confirmationLogger.getConfirmations().size()); }
```

```
private void waitForBatch(int batchSize, int timeout) throws InterruptedException {
    int sleepTime = 100;
    while (confirmationLogger.getConfirmations().size() < batchSize && timeout > 0) {
        Thread.sleep(sleepTime);
        timeout -= sleepTime;
    }
}
```

Resultado:



```
Output
SpringJMS x Java DB Database Process x GlassFish Server 3.1.2 x Retriever Output x Debugger Console x
07/06/2013 03:10:19 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RAL101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
07/06/2013 03:10:19 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RAL101: GlassFish MQ JMS Resource Adapter Started:REMOTE
INFO : rewards.jms.client.RewardConfirmationLogger - received confirmation: 1
INFO : rewards.jms.client.RewardConfirmationLogger - received confirmation: 2
INFO : rewards.jms.client.RewardConfirmationLogger - received confirmation: 3
INFO : rewards.jms.client.RewardConfirmationLogger - received confirmation: 4
INFO : rewards.jms.client.RewardConfirmationLogger - received confirmation: 5
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 20.839 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

-----
BUILD SUCCESS
-----
```

Test Results

com.mycompany:SpringJMS:jar:1.0-SNAPSHOT x

▶▶

100,00 %

🔊

📁

✅

⚠️

❌

⬆️

⬇️

The test passed.(20,838 s)

📁

✅ rewards.jms.client.DiningBatchProcessorTests passed

📁

✅ testBatch passed (20,025 s)

📄 Test Results

📄 Output