15ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 5-6

Desarrollo de aplicaciones empresariales con Jakarta EE

Uriel Hernández

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

50 AÑOS
Educación Continua
1971 - 2021

Message Driven Bean

Pixup y Notificaciones

- Tras un análisis de la arquitectura del sistema, se identificó que el registro del usuario debería ser independiente del mecanismo de notificación, por lo que la notificación podría ser ejecutada de manera asíncrona.
- La solución a este problema es implementar un patrón llamado Productor/Consumidor para el mecanismo de Notificaciones. El Productor generará un mensaje cuando se realice el registro del usuario. El Consumidor tomará este mensaje y realizará la notificación (envío de email de confirmación).
- Este mecanismo permite desacoplar 2 aplicaciones (Pixup y Notificaciones) ya que ninguna de ellas tiene que conocer de la otra, es decir, el Productor y el Consumidor se encuentran desacoplados.

Pixup y Notificaciones

- Al contar con un mecanismo asíncrono, resiliente y durable como es el uso de colas de mensajes por medio de JMS, nos permite procesar los mensajes cuando la aplicación se encuentre disponible.
- En nuestro caso, si el sistema de Notificaciones está fuera de línea no afecta la Disponibilidad (Availability) del sistema Pixup, ya que Pixup puede seguir registrando usuarios sin depender del sistema de Notificaciones.
- Cuando el sistema de Notificaciones esté de nuevo en línea procesará todos los mensajes en la cola generados por Pixup.

Message Driven Bean (MDB)

- Corresponde al Consumidor en el patrón Productor/Consumidor.
- Se utiliza la anotación **@MessageDriven.**
- Java Message Service (JMS) es el api que soporta mensajería.
- JMS proporciona 2 modelos de mensajes:
  - Peer to peer (Queue)
  - Publish/Subscribe (Topic)
- Para nuestro sistema utilizaremos el modelo de mensajes peer to peer (Queue - colas).
- Un MDB es un componente invocado por el contenedor cada vez que un mensaje llega al sistema de mensajería.

Message Driven Bean – Ciclo de Vida

1. El contenedor crea un message driven bean usando el constructor sin argumentos.
2. Las dependencias del bean son inyectadas.
3. El bean se encuentra creado y listo para recibir mensajes.

Pasos para enviar un mensaje a una cola JMS (Producer)

1. Crear una fábrica de conexiones de colas en el application server.
2. Crear una cola destino (cola donde se enviarán los mensajes) en el application server.
3. Inyectar en el Bean Producer un Resource QueueConnectionFactory indicando el JNDI especificado en el application server.
4. Inyectar en el Bean Producer una Queue indicando el JNDI especificado en el application server.

Pasos para enviar un mensaje a una cola JMS (Producer)

5. Crear un contexto JMS (JMSContext) a través de la fábrica de conexiones (QueueConnectionFactory).
6. Construir el mensaje que se enviará a la cola.
7. Crear un Producer JMS (JMSProducer) por medio del JMSContext.
8. Enviar el mensaje a la cola.
9. Liberar los recursos de conexión.

Tipos de Mensaje JMS

Mensaje	Descripción
TextMessage	El cuerpo del mensaje contiene un objeto String
MapMessage	El cuerpo del mensaje contiene un conjunto de pares clave/valor, siendo la clave un String y el valor un String o tipo primitivo
ObjectMessage	El cuerpo del mensaje contiene un objeto java Serializable
ByteMessage	El cuerpo del mensaje contiene un stream de bytes
StreamMessage	El cuerpo del mensaje contiene un stream de valores primitivos, llenados y leídos en orden secuencial

Queue

```
<jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>
<jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>
<jms-queue name="NotificacionQueue" entries="java:/jms/queue/NotificacionQueue"/>
<connection-factory name="InVmConnectionFactory" entries="java:/ConnectionFactory" connectors="in-vm"/>
<connection-factory name="RemoteConnectionFactory" entries="java:jboss/exported/jms/RemoteConnectionFactory"
<pooled-connection-factory name="activemq-ra"
entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory" connectors="in-vm" transaction="xa"/>
```

Producer

NotificacionProducer

```java
@Log
@Stateless
public class NotificacionProducerImpl implements NotificacionProducer {

    @Resource(name="java:/JmsXA")
    private QueueConnectionFactory connectionFactory;

    @Resource(mappedName="java:/jms/queue/NotificacionQueue")
    private Queue notificacionQueue;
```

NotificacionProducer

```java
@Override
public boolean enviarNotificacionAltaUsuario(Integer idUsuario, String email) {
    try (JMSContext context = connectionFactory.createContext()) {
        MapMessage mapMessage = context.createMapMessage();
        mapMessage.setInt("idUsuario", idUsuario);
        mapMessage.setString("email", email);
        JMSProducer producer = context.createProducer();
        producer.send(notificacionQueue, mapMessage);
        log.info("Mensaje enviado con exito " +
                 "a la cola jms/queue/NotificacionQueue, [idUsuario: " +
                idUsuario + ", email: " + email + "]");
        return true;
    } catch (Exception e) {
        log.log(Level.SEVERE,
                "Error al enviar el mensaje a la cola jms/queue/NotificacionQueue", e);
        return false;
    }
}
```

UsuarioServiceImpl

```java
@Stateless
public class UsuarioServiceImpl implements UsuarioService {

    @Inject
    private UsuarioRepository usuarioRepository;
    @Inject
    private DomicilioRepository domicilioRepository;
    @Inject
    private ColoniaRepository coloniaRepository;
    @Inject
    private TipoDomicilioRepository tipoDomicilioRepository;
    @Inject
    private NotificacionProducer notificacionProducer;
```

Consumer

EmailServiceImpl

```java
@Stateless
public class EmailServiceImpl implements EmailService {

    @Resource(mappedName="java:jboss/mail/Default")
    private Session mailSession;
```

EmailServiceImpl

```java
@Override
public void enviarEmail(
        String email, String titulo, String mensaje) {
    try {
        // Crear email
        MimeMessage message = new MimeMessage(mailSession);
        message.setRecipients(Message.RecipientType.TO,
                InternetAddress.parse(email, false));
        message.setSubject(titulo);
        message.setContent(mensaje, "text/html");
        message.setSentDate(new Date());
        // Enviar email
        Transport.send(message);
    } catch (Exception e) {
        e.printStackTrace();
        throw new ServiceUnavailableException("EmailService");
    }
}
```

NotificacionServiceImpl

```java
@Override
@Transactional(value=Transactional.TxType.REQUIRED)
public Notificacion enviarNotificacionAltaUsuario(Integer idUsuario, String email) {
    String descTipoNotificacion = "ALTA_USUARIO";
    String emailTitulo = "Creación de cuenta " + email + " exitosa";
    String emailMensaje = "<h1>Pixup</h1><h3>" +
            "La cuenta " + email + " fue creada exitosamente</h3>";

    Optional<Usuario> usuario = usuarioRepository.findByEmail(email);
    if (usuario.isEmpty()) {
        throw new UsuarioNotFoundException(email);
    }
    Optional<TipoNotificacion> tipoNotificacion =
            tipoNotificacionRepository.findByDescripcion(descTipoNotificacion);
    if (tipoNotificacion.isEmpty()) {
        throw new TipoNotificacionNotFoundException(descTipoNotificacion);
    }
    Notificacion notificacion = new Notificacion();
    notificacion.setUsuario(usuario.get());
    notificacion.setTipoNotificacion(tipoNotificacion.get());
    notificacionRepository.save(notificacion);
    emailService.enviarEmail(email, emailTitulo, emailMensaje);
    return notificacion;
}
```

NotificacionConsumerImpl

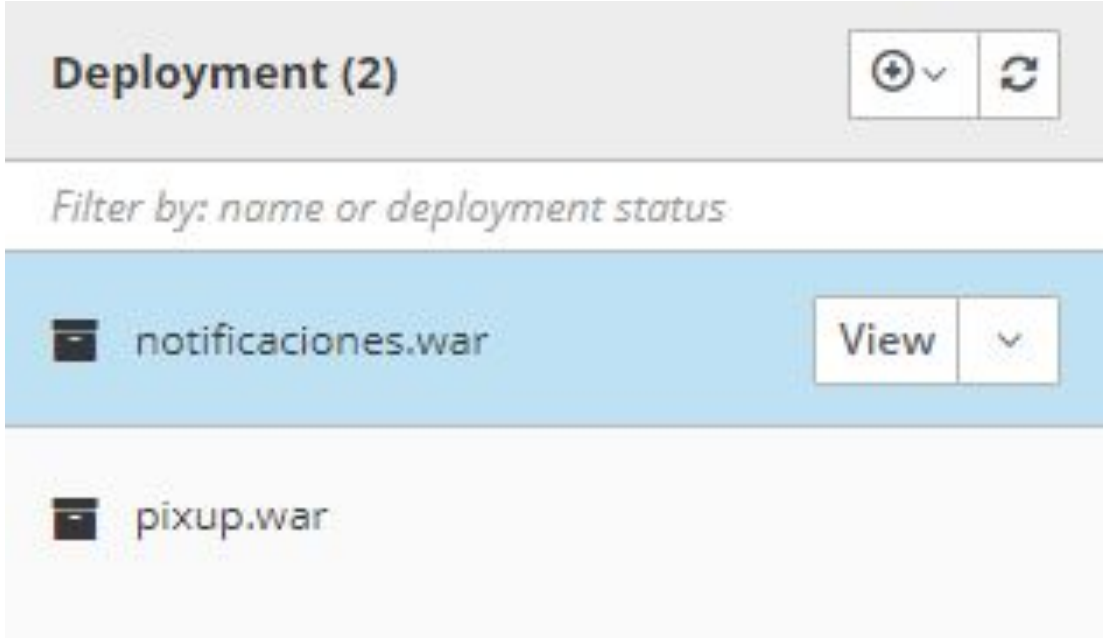
```
@Log
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
          propertyName="destination", propertyValue="java:/jms/queue/NotificacionQueue"),
        @ActivationConfigProperty(
          propertyName="destinationType", propertyValue="jakarta.jms.Queue")
    })
public class NotificacionConsumerImpl implements MessageListener {

    @Inject
    private NotificacionService notificacionService;
```

NotificacionConsumerImpl

```java
@Override
public void onMessage(Message message) {
    try {
        MapMessage mapMessage = (MapMessage)message;
        Integer idUsuario =
                mapMessage.itemExists("idUsuario") ? mapMessage.getInt("idUsuario") : 0;
        String email =
                mapMessage.itemExists("email") ? mapMessage.getString("email") : "";
        log.info("Mensaje recibido: [idUsuario: " + idUsuario + ", email: " + email + "]");
        Notificacion notificacion =
            notificacionService.enviarNotificacionAltaUsuario(idUsuario, email);
        log.info("Se procesó de manera exitosa el mensaje, notificacion generada: " +
                notificacion);
    } catch (Exception e) {
        log.log(Level.SEVERE, "Error al procesar el mensaje en NotificacionConsumerImpl", e);
    }
}
```

Aplicaciones Desacopladas



Contacto

Uriel Hernández

Solution Architect

urielhdezorozco@yahoo.com.mx

Redes sociales:

https://www.linkedin.com/in/juho-mex